TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Lehrstuhl für Bildverarbeitung und Künstliche Intelligenz / IX

# Few-Shot Object Detection in Industrial Applications
Training Accurate Models with Few Annotations

Patrick Moritz Follmann

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

| | | |
|---|---|---|
| Vorsitzender: | | Prof. Gudrun J. Klinker,. Ph.D. |
| Prüfer der Dissertation: | | |
| | 1. | Hon.-Prof. Dr. Carsten Steger |
| | 2. | Prof. Dr. Bernt Schiele |

Die Dissertation wurde am 15.02.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 23.06.2021 angenommen.

# Abstract

For engineers, recent developments in deep learning make it easier to detect or segment instances of objects within an image. In particular, convolutional neural networks can be trained in an end-to-end manner solely based on images where the expected output is labeled as a bounding box or a pixel-precise mask. This saves time spent for manual tuning of parameters, but shifts the work load towards the annotation of thousands of instances within hundreds of images. In this thesis, we focus on few-shot methods that only require few annotated images for training, but are accurate enough to be successfully applied in an industrial application.

To be able to evaluate such approaches, we introduce *D2S*, a novel industrial dataset for detection and instance segmentation. The dataset has high-quality annotations that allow to judge the localization accuracy up to subtle differences and is designed to encourage the development of few-shot methods. Moreover, we propose a change of the predominant evaluation measure for object detection models, the mean average precision, that includes the actual precision of methods and hence, penalizes the high number of false positive predictions that has been mostly neglected so far. We show that with class-specific confidence thresholds, which can be automatically computed with the new measure, $AP^{\star}$, the precision of current models can be significantly improved at almost the same recall.

Moreover, we present methods that reduce the enormous annotation effort that is required for the supervised models that are predominantly used today. On the one hand, we introduce a simple yet efficient pipeline to generate large artificial training datasets with minimal labeling effort. On the other hand, we change the models themselves to be less data-hungry during training, but still predict very accurate results in terms of classification and localization: First, we present a novel rotational convolution and pooling module that extracts rotationally invariant features from an image. Second, we combine a deep-learning-based oriented box detection method with a classic matching approach in a hybrid model. This allows to train the model with a single template image per class, and at the same time to predict results with sub-pixel precision efficiently. Third, we use oriented boxes as basis to predict pixel-precise instance masks. With this novel approach, we can predict more accurate masks and require fewer of the expensive mask annotations during training. Fourth and finally, we present a method that simultaneously predicts the amodal masks and occluded parts of each instance within an image. This paves the way for a more effective grasping of objects with a robot.

With the presented methods in this thesis, the ease of use and accuracy of current object detection and instance segmentation methods is improved, which helps to increase their use in industrial applications.

# Zusammenfassung

Die jüngsten Entwicklungen im Deep Learning erleichtern Ingenieuren die Erkennung oder Segmentierung von Objekten innerhalb eines Bildes. Insbesondere können Convolutional Neural Networks in einem Stück trainiert werden, und zwar ausschließlich auf Basis von Bildern, in denen die erwartete Ausgabe mit umschließenden Rechtecken oder pixelgenauen Regionen eingezeichnet wurde. Dies spart Zeit, die bisher für die manuelle Einstellung der Modellparameter notwendig war, verlagert aber den Arbeitsaufwand auf die Annotation von Tausenden von Instanzen in Hunderten von Bildern. In dieser Arbeit konzentrieren wir uns auf sogenannte "Few-Shot"-Methoden, die nur wenige gelabelte Bilder zum Training benötigen, aber genau genug sind, um erfolgreich in einer industriellen Anwendung eingesetzt zu werden.

Um solche Ansätze evaluieren zu können, stellen wir *D2S* vor, einen neuartigen industriellen Datensatz für Objekterkennung und Instance Segmentation. Der Datensatz verfügt über qualitativ hochwertige Annotationen, die es erlauben, die Lokalisierungsgenauigkeit bis hin zu feinen Unterschieden zu beurteilen. Er ist so konzipiert, dass er die Entwicklung von "Few-Shot"-Methoden fördert. Darüber hinaus schlagen wir eine Änderung des vorherrschenden Evaluationsmaßes für Objekterkennungsmodelle, der Mean Average Precision, vor, die die tatsächliche Präzision der Methoden mit einbezieht und somit eine hohe Anzahl an falsch-positiven Vorhersagen bestraft, die bisher unbeachtet blieb. Wir zeigen, dass mit klassenspezifischen Konfidenz-Schwellenwerten, die mit dem neuen Maß, $AP^{\star}$, automatisch berechnet werden können, die Präzision aktueller Modelle bei nahezu gleichem Recall deutlich verbessert werden kann.

Des Weiteren stellen wir Methoden vor, die den enormen Annotationsaufwand reduzieren, der für die heute meist genutzten überwachten Modelle erforderlich ist. Einerseits präsentieren wir einen einfachen und effizienten Ansatz, um große künstliche Trainingsdatensätze mit minimalem Labelaufwand zu erzeugen. Andererseits verändern wir die Modelle selbst so, dass sie während des Trainings weniger datenhungrig sind, aber dennoch sehr genaue Ergebnisse vorhersagen: Erstens präsentieren wir ein neuartiges Rotational Convolution und Pooling-Modul, das rotationsinvariante Merkmale aus einem Bild extrahiert. Zweitens kombinieren wir eine auf Deep Learning basierende Methode zur Objekterkennung mit orientierten Boxen mit einem klassischen Matching-Ansatz in einem Hybridmodell. Dies ermöglicht es, das Modell mit einem einzigen Beispielbild pro Klasse zu trainieren und gleichzeitig effizient subpixel-genaue Ergebnisse vorherzusagen. Drittens verwenden wir orientierte Boxen als Grundlage für die Instance Segmentation. Mit diesem neuartigen Ansatz können wir genauere Regionen vorhersagen und benötigen weniger der teuren Annotationen während des Trainings. Viertens und letztens stellen wir eine Methode vor, die gleichzeitig die amodalen Masken und verdeckten Teile jeder Instanz innerhalb eines Bildes vorhersagen kann. Dies ebnet den Weg für ein effektiveres Greifen von Objekten mit einem Roboter.

Mit den in dieser Arbeit vorgestellten Methoden werden die Benutzerfreundlichkeit und Genauigkeit aktueller Methoden zur Objekterkennung und Instance Segmentation verbessert, was dazu beiträgt, ihren Einsatz in industriellen Anwendungen zu erhöhen.

# Acknowledgments

People around me know that, throughout my life, I tend to set myself ambitious goals. Clearly, this thesis, that was the focus of my (work-)life during the last five years was one of these goals. At the beginning, of course, I naïvely dreamed of a big breakthrough that will lead to very large gains. But, over the years, I learned that, despite some desperate efforts you might invest, the progress of your work often makes only seemingly tiny steps and not all of them are into the right direction. However, if you keep up your work and motivation these steps evolve and might form something bigger once they are put together. Personally, there were of course some ups and downs during this thesis, but, in the end, I am proud of what we and I achieved. Taken together, I think that we could do our part of the large research progress that was made in the last years, and, more importantly, that we all learned a lot.

The contribution of this work would not have been possible without the support of my team at MVTec Software GmbH: First, I want to thank my academic advisor, Prof. Carsten Steger, for giving me the opportunity to write this thesis while being a full team member of the Research Department, for the fruitful discussions, the feedback and for his openness. I thank Prof. Markus Ulrich for sharing his huge experience in all kinds of computer vision areas and for always being there for me. From my colleagues and friends at MVTec, I want to especially mention Rebecca König, who always made it fun to implement our own networks in HALCON and helped me a lot when we had a hard time to think about angles, rotations, or anchor boxes — thank you so much, Becci! A special thanks also goes to my co-workers and colleagues: Dr. Bertram Drost, Michael Fauser, Philipp Härtinger, Michael Klostermann, Jan-Hendrik Neudeck, Kilian Batzner, Paul Bergmann, Dr. Markus Glitzner, and Dr. David Sattlegger. Moreover, I want to thank my friends from the engineering department: Christina Eisenhofer, Matthias Meßner, Roman Moie, Dr. Horst Osberger, and many more.

The biggest thanks goes to Dr. Tobias Böttger-Brill, who encouraged me to apply for my PhD at MVTec and went through all of this with me together, as a co-worker, adviser, and as a close friend, not only in difficult or fun moments at work, but also in our free time, crossing the alps with our mountain bikes, or as my best man. Thank you, Tobi, for being so awesome!

I want to thank my family. My parents, for giving me all the opportunities that I had throughout my life, for telling me that despite the sadness I might feel in a moment of failure, that I could learn more from it than from the moments of success. You always encouraged me to finish the things that I started and to stand up again when I fell. My sister, Anna, for keeping to tell me that there are far more things in life than work and computer vision.

Finally, and most importantly, I want to thank my wife Karin, who had to share me with my laptop during the last five years. Thank you that you withstood all my ups and downs, thank you for your love and that you kept believing in and smiling at me — I hope that I can give it back to you in the near future!

▲

# Contents

# 1 Introduction

## 1.1 Background and Motivation

Today, machines are all around us and the use of robots[1] is increasing steadily. Industrial robots[2] are already widely used, predominantly in the automotive and electronics industries. In 2018, the worldwide annual installation count reached its maximum with more than 422 000 units [139, p. 542]. However, for a majority of people around the world, robots are still found rather seldom in their daily life. Today, domestic service robots are mostly used for cleaning purposes (vacuum, pool, or window cleaning robots) or outdoors, such as robotic lawn mowers [140, p. 27]. Also here, according to Müller et al. [140, p. 27] *"[t]he total number of service robots for personal and domestic use increased by 34% to more than 23.2 million units sold in 2019."* Also for professional service robots, which are mainly used in logistics applications, but are also increasingly present in medical or agriculture applications, the market is growing with equally high rates [140, p. 21ff]. A key factor for the increased use of robots to solve difficult tasks is that they are equipped with more and more intelligence. For example, this allows them to navigate in their environment and take decisions autonomously.

Bringing intelligence to machines is a dream that goes back at least to the 1950s, when the term Artificial Intelligence (AI) was born at the *Dartmouth Summer Research Project on Artificial Intelligence* [133, 181]. According to the first report of the *'One Hundred Year Study on Artificial Intelligence'* [181], *"AI is a science and a set of computational technologies that are inspired by [...] the ways people use their nervous systems and bodies to sense, learn, reason, and take action."* This thesis is focused on computer vision, which today involves the first three categories *sense, learn, and reason*. Further, Stone et al. [181] claim that computer vision *"is currently the most prominent form of machine perception. It has been the sub-area of AI most transformed by the rise of deep learning. For the first time, computers are able to perform some vision tasks better than people."*

Certainly, recent developments in machine learning and in particular convolutional

[1] Siciliano and Khatib [173, p. 1] introduce robots as *"machines that are skilled and intelligent."*

[2] According to [139, p. 23], an industrial robot is defined by ISO 8373:2012 as *"an automatically controlled, reprogrammable multipurpose manipulator programmable in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications."*

neural networks (CNNs) [107] have led to breakthroughs in different fields of computer vision. Among them are image classification [68, 95, 177], object detection [58, 120, 159], and semantic segmentation [126, 208], which are all used as key components of more complex machine vision applications such as robot grasping, defect detection, or automatic checkout systems in grocery stores. In this regard, deep neural networks or deep learning (DL) plays an important role because it is presently the predominant approach to tackle these challenges.

The evolution towards DL during the last decade was enabled mainly by the availability of two essential ingredients. One is the improved compute power of modern graphics processing units (GPUs) that enables to apply and train large CNNs with millions of parameters in a manageable time. The second, and for this thesis more interesting component, is data. Millions of images have not only been collected, but also have been annotated, to form datasets that have driven the progress in computer vision during the last years. Among them are *ImageNet* [28, 168] for classification of 1000 different object categories, *Pascal VOC* [41] and *COCO* [118] for object detection, and *ADE20k* [211] and *cityscapes* [22] for semantic segmentation. With these annotated images, CNNs can learn to transform the input images into the desired output, i.e., a class label, bounding boxes around the objects, or pixel-precise segmentations. Once the annotated data is given, the challenges that are posed by the above mentioned datasets are restricted to a domain where the machine can utilize its advantages, such as a large memory and a fast processing time of samples during the learning process. This means that GPUs and annotated data are like the motors and fuel to train networks for different tasks efficiently.

One of the application fields where people expect an increased use of AI-based systems within the next years is medical diagnosis and, in particular, cancer screening. Let us consider the example of skin cancer: the expectations for the use of AI-based systems are high because, on the one hand, there is a huge number of cancer cases (e.g., 5.4 million new skin cancer cases per year in the US) leading to a high number of deaths (10 000 annually in the US), and, on the other hand, the detection of cancer at an early stage leads to a dramatically higher chance to survive [37, p. 115]. According to Esteva et al. [37, p. 115], *"the estimated 5-year survival rate for melanoma drops from over 99% if detected in its earliest stages to about 14% if detected in its latest stages."* Hence, non-invasive techniques for the skin cancer screening are important. However, screening techniques like dermatoscopy require the expert knowledge of a trained dermatologist. Back in 2017, Esteva et al. [37] were the first to reach a similar performance as 21 dermatologists on the task to classify keratinocyte carcinomas vs. benign seborrheic keratoses, and malignant melanomas versus benign nevi using Google's Inception v3 CNN architecture [184] (cf. Section 2.6), which was finetuned on 129 450 dermatologist-labeled images of skin lesions and pretrained on almost 1.3 million annotated images of *ImageNet* [168]. In subsequent years, this result has been extended, e.g., by Haenssle et al. [65] comparing a more recent Inception v4 CNN architecture [185] to 58 dermatologists from 17 countries. While the CNN outperformed the performance of a clear majority of the dermatologists in dichotomous classification (i.e., differentiating between melanoma and benign nevi), the authors only suggest that *"a CNN algorithm may be a suitable tool to aid physicians in melanoma detection"* and, further, they raise the concern that *"operating physicians may not follow the recommendations of a CNN they not fully trust, which may diminish the reported diagnostic performance"* [65, p. 1841]. In a recent study, Brinker et al. [13] use an *ImageNet* pretrained ResNet 50 [68] CNN and reduce the training set for finetuning to 12 378 open-source images of melanoma and atypical nevi. On a test set of 100 images (containing 20 melanoma images), the CNN outperforms 136 of the 157 dermatologists from all levels of the clinical hierarchy (from junior to chief physicians) of twelve university hospitals in Germany. Despite the super-human performance, also here, the authors conclude that *"aritificial intelligence algorithms may successfully assist dermatologists with melanoma detection in clinical practice which needs to be carefully evaluated in prospective trials"* [13, p. 53].

This example shows that still a remarkable effort is necessary to outperform humans: To obtain the ground truth labels for the training and evaluation images in [13, p. 48] the diagnosis was either done by *"histopathological examination of biopsies, by expert consensus, or by another diagnosis method, such as a series of images that showed no temporal changes."* Hence, a tedious annotation process was necessary to be able to train the CNN. Moreover, the model was trained for 13 epochs (i.e., iterations over the whole training set), which means that the model saw each of the 12 378 images approximately thirteen times during the training. A human would never have the endurance and memory to perform this kind of training in reasonable time. Thus, the central question that we raise in this thesis is: Could the algorithm also outperform the human if it had fewer training samples to learn from?

Although the medical example above is not taken from a typical industrial application, it vividly shows that despite the success of recent DL-based methods on the above mentioned computer vision research datasets, in our experience, they are not yet ready to be used in every industrial application. This is because the following requirements for a method that should be used successfully within the industry are either greater or somewhat complementary to the challenges that are posed by those datasets. Or, the other way around, industrial datasets should be designed such that they can be used to evaluate the requirements of industrial methods:

1. **Localization Accuracy:** In a production process, very small details or defects can decide whether a part is okay or not okay. Therefore, a method running in an industrial application must localize objects very accurately. Hence, an industrial dataset should have high-quality annotations, such that the accuracy of models can be compared up to subtle differences.

2. **Fine-Grained Classification:** In industrial classification problems, methods must be capable to distinguish different categories based on small details. For example, there might be two classes of screws that have the same color, the same length, and the same head, but just differ marginally with respect to the widths of their threads. However, due to a fixed acquisition setup within the production process, the intra-class variations are often small in industrial applications, e.g., the length of a particular screw type only differs due to changes in perspective. In contrast, in

everyday photography as collected in *COCO* or *VOC*, a major challenge is given by the large intra-class variations. For example, a *person* is articulated, can wear different clothes, and can have different gender, age, or skin-color. The inter-class variation is often large, i.e., a *person* can be clearly distinguished from a *car*.

3. **Robustness:** The model should identify objects correctly independent of nearby clutter, a changed background, or other changes within the environment. For example, if the lighting changes due to weather influences or because the light source had to be replaced for some reason, the model should still predict correct results. To evaluate the robustness of methods, the validation and test set of a dataset should contain objects that are sampled at the boundary or outside of the distribution of the training samples. In other words, there should be objects of the known classes that have a slightly different appearance than those of the training set and there should be clutter objects that do not belong to any of the classes within the training set.

4. **Reliability and Uncertainty Estimation:** The user should be able to rely on the model's predictions. Regarding the example above, for physicians false positive predictions could lead to unnecessary surgeries and hence to unnecessary pain for patients and other side-effects, such as risk of infection and increased costs. Hence, the number of false predictions with a wrong class or with an inaccurate localization should be as small as possible. Moreover, the model should assign a high confidence to a prediction only if this result is correct. Vice-versa, for objects that are unfamiliar to the model, since they do not belong to the known categories of the training set, no prediction should be made or the predicted confidence should be low. This criterion is only indirectly related to the dataset requirements. It generally must be measured using an appropriate evaluation protocol that penalizes the number of false predictions. Moreover, false predictions with high confidences should be penalized more than false predictions where the method is obviously unsure — indicated by a low confidence. However, we can still infer a requirement for the dataset from this method criterion: The dataset must be difficult enough such that these hard cases and an appropriate number of false predictions occur. If there is only a very small number of such cases, e.g., below twenty, we cannot clearly decide whether an advantage of a method with respect to uncertainty estimation is significant.

5. **Ease of Use:** The implementation effort to apply the method in a new application should be low. If the product or application is changed, it should not be necessary to finetune a lot of parameters manually. Moreover, engineers that adapt the method to the new application or environment should get first and intermediate results quickly, such that they can adapt the method and improve the results in a reasonable time. For example, if a tedious annotation of many images needs to be done, just to find out after the first training that the method does not meet the accuracy requirements, this is annoying and expensive. Therefore, the model should only need a few examples of each object to be trained, which we refer to as

few-shot learning. To encourage the design of models that fulfill this criterion, the challenge posed by the dataset should be set accordingly. That means, the training set should be small and relatively easy to annotate such that a first training can be done quickly. Few-shot methods should be favored. Moreover, the validation set should be moderately sized such that evaluations are fast, but are still expressive enough to answer whether the above criteria are met.

6. **Scalability:** In many industrial settings, there is not enough space to fit a large GPU into the housing of the machine, a suitable power supply might not be given, and the lost heat could be a problem. Therefore, the algorithm is often computed on an embedded device with limited memory and computing power. At the same time, the runtime needs to be low such that results can be computed in real-time, e.g., such that it fits to the speed of a conveyor belt on which objects must be detected. The method's runtime and memory requirements should scale well with the number of categories, the size of the input images, and the number of objects within an image. Hence, with respect to the number of classes and the number of objects within an image, an industrial dataset should be large-scale enough to get an impression about the scalability. If the final application is already known, it is enough if the dataset meets the requirements of this application. In most cases, it is advisable to use several different datasets to measure the scalability.

Of course, for all of the above criteria, there is almost unlimited room for improvement and most algorithms do not satisfy all of them. In particular, one can apply the method to increasingly difficult images or one can try to locate objects that are very difficult to separate from the background. Moreover, there will always be methods that perform better than others in certain cases, but worse than others in different cases.

## 1.2 Objectives

In research, the machine learning recipe has been successfully applied to many different problems for which annotated data is available. Of course, DL-based methods already have found their way into industry and other commercial applications. For companies, it is especially appealing that in comparison to classic, rule-based algorithms less expert knowledge and engineering time is necessary to solve a problem. However, with deep learning this engineering time is moved from manually deciding about rules and their parameters towards data collection and annotation.

Because it is undesirable to spend weeks for data annotation before the first results can be computed, the central research question of this thesis is: **Is it possible to build methods that learn from few annotated samples, but localize and classify objects accurately?**

This means that the focus of this work is on a combination of the criteria ease of use, localization accuracy, and fine-grained classification. With this main objective in mind, we also look at combinations of the remaining requirements within the following chapters.

## 1.3 Outline and Contributions

Chapter 2: Convolutional Neural Networks and Deep Learning

This chapter gives an overview of the used notation, introduces some of the basic mathematical concepts, and provides an introduction to machine learning and in particular to convolutional neural networks. Hence, this chapter explains the general structure of the models and examines the process of how they are trained and evaluated.

Chapter 3: Rotational Invariance for CNNs

An approach to reduce the number of training samples is to build models that are invariant to certain transformations of their inputs. This chapter shows how rotationally invariant features can be learned by a CNN. This allows to correctly classify arbitrarily rotated inputs independent of the rotation of the training images.

Chapter 4: An Introduction to Object Detection

From this chapter onwards, we focus on object detection via bounding boxes or pixel-precise masks. Therefore, this chapter gives a detailed introduction to modern DL network architectures for bounding box object detection and instance segmentation. It thoroughly describes their shared concepts and presents an overview of recent works on datasets and methods.

Chapter 5: D2S: Densely Segmented Supermarket Dataset

This chapter presents a novel industrial dataset for detection and instance segmentation with high-quality annotations that meets the above mentioned criteria for industrial datasets. It encourages to build methods that generalize well with a limited amount of training data and allows to measure differences between very accurate methods.
Moreover, a detailed analysis of baseline models on this dataset is given and a number of structural failure cases of current detection methods is identified. These are the motivation for further work carried out in the following chapters.

Chapter 6: $AP^\star$ — Fixing the Recall Bias of Average Precision

One conclusion of Chapter 5 is that seemingly successful models make a very high number of wrong detections. This chapter reviews the computation of the most frequently used detection evaluation measure $AP$ and identifies several of its shortcomings. Subsequently, certain changes that lead to an improved $AP^\star$ measure that directly takes the model precision into account are proposed. Hence, the chapter addresses the need for reliable models with a good uncertainty estimation.

Chapter 7: Data Generation for Few-Shot Detection

This chapter presents the first step towards few-shot object detection models. It presents simple, yet efficient methods for data generation to reduce the annotation effort in DL-based object detection. The methods use only a handful of annotations per category and generate large training sets. With artificial training data for the networks, the robustness of models, e.g., to reflections or with respect to close-packed objects, is improved while at the same time ease of use requirement is met.

Chapter 8: Oriented Box Detection

This chapter explains how detection models can be extended from axis-aligned to oriented boxes. Generally, oriented bounding boxes yield a better approximation of the underlying objects than axis-aligned bounding boxes and thus, localization accuracy of the methods is improved. The novel methods are evaluated on different datasets and it is shown how they can be compared to the axis-aligned box detection baselines.

Chapter 9: A Comparison to Shape-Based Matching

In most cases, classic rule-based algorithms need significantly fewer training examples than DL-based methods. Shape-Based Matching is an example that can setup the model on a single image and yields very accurate results. The method has been applied successfully in industrial applications for the last 20 years. This chapter compares the accuracy and performance of DL-based object detection with Shape-Based Matching. Moreover, Shape-Based Matching is combined with DL-based object detection in a hybrid approach that predicts accurate results from few training samples and has a good scalability with respect to the number of classes.

Chapter 10: Oriented Boxes for Few-Shot Instance Segmentation

In this chapter, a novel method for instance segmentation based on oriented boxes that increases the accuracy of mask predictions is presented. In comparison to their axis-aligned counterparts, oriented bounding boxes are not only visually more pleasant, but also lead to a higher mask to background ratio and to more consistent mask targets. Moreover, the proposed model obtains the same quality of results, while the number of annotated masks can be significantly reduced.

Chapter 11: Amodal Instance Segmentation

Occluded objects pose a major challenge to current instance segmentation methods. This chapter presents a novel network architecture for amodal instance segmentation that directly predicts the visible and occluded parts of an object. Moreover, the *D2S* dataset of Chapter 5 is extended with annotations for amodal detection and novel metrics to measure the performance of amodal instance segmentation methods are introduced.

Chapter 12: Conclusion

We conclude the work by discussing open problems and possible extensions of the presented methods.

## 1.4 Publications

Parts of this thesis contain material previously published in the following publications. In case of Ulrich et al. 'A Comparison of Shape-Based Matching with Deep-Learning-Based Object Detection' [194], the first and the second author share the main contributions.

- P. Follmann and T. Böttger. A Rotationally-Invariant Convolution Module by Feature Map Back-Rotation. In *2018 IEEE Winter Conference on Applications of Computer Vision, WACV*, pages 784–792, 2018. doi:10.1109/WACV.2018.00091
- P. Follmann, T. Böttger, P. Härtinger, R. König, and M. Ulrich. MVTec D2S: Densely Segmented Supermarket Dataset. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Proceedings, Part X*, volume 11214 of *Lecture Notes in Computer Science*, pages 581–597. Springer, 2018. doi:10.1007/978-3-030-01249-6_35
- P. Follmann, B. Drost, and T. Böttger. Acquire, Augment, Segment and Enjoy: Weakly Supervised Instance Segmentation of Supermarket Products. In T. Brox, A. Bruhn, and M. Fritz, editors, *Pattern Recognition - 40th German Conference, GCPR, Proceedings*, volume 11269 of *Lecture Notes in Computer Science*, pages 363–376. Springer, 2018. doi:10.1007/978-3-030-12939-2_25
- P. Follmann, R. König, P. Härtinger, M. Klostermann, and T. Böttger. Learning to See the Invisible: End-to-End Trainable Amodal Instance Segmentation. In *IEEE Winter Conference on Applications of Computer Vision, WACV*, pages 1328–1336, 2019. doi:10.1109/WACV.2019.00146
- M. Ulrich, P. Follmann, and J.-H. Neudeck. A comparison of shape-based matching with deep-learning-based object detection. *tm-Technisches Messen*, 86(11):685–698, 2019. doi:10.1515/teme-2019-0076

The following publications of the author are not part of this thesis:

- T. Böttger, P. Follmann, and M. Fauser. Measuring the Accuracy of Object Detectors and Trackers. In V. Roth and T. Vetter, editors, *Pattern Recognition - 39th German Conference, GCPR 2017, Proceedings*, volume 10496 of *Lecture Notes in Computer Science*, pages 415–426. Springer, 2017. doi:10.1007/978-3-319-66709-6_33
- T. Böttger and P. Follmann. The Benefits of Evaluating Tracker Performance Using Pixel-Wise Segmentations. In *2017 IEEE International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1983–1991, 2017. doi:10.1109/ICCVW.2017.232

- P. Follmann and B. Radig. Detecting Animals in Infrared Images from Camera-Traps. *Pattern Recognit. Image Anal.*, 28(4):605–611, 2018. doi:10.1134/S1054661818040107
- B. Radig and P. Follmann. Training a Classifier for Automatic Flash Detection in Million Images from Camera–Traps. In *International Conference on Pattern Recognition and Artificial Intelligence, (ICPRAI)*, volume 12, pages 589–591, 2018

# 2 Convolutional Neural Networks and Deep Learning

This chapter introduces the basic concepts, nomenclature, and notations used in this thesis. Since very detailed explanations are beyond the scope of this thesis, we concentrate on some specific topics that reappear in later chapters. A more detailed introduction can be found in the standard works of Bishop [7], Goodfellow et al. [59], or Steger et al. [180]. Our contribution of this chapter is to briefly introduce the most important aspects that are necessary to understand the ideas presented within the remainder of the work.

## 2.1 Notation

A summary of the general notation used within this thesis is displayed in Table 2.1. Most of the concepts should be familiar, others are explained in this chapter.

| | Sets and Spaces |
|---|---|
| $\mathbb{R}$ | Set of real numbers |
| $\mathbb{Z}$ | Set of integers |
| $\mathbb{N}_{(0)}$ | Set of natural numbers (including zero) |
| $\mathbb{R}^+_{(0)}$ | Set of positive real numbers (including zero) |
| $\{0, 1\}$ | Set containing 0 and 1. |
| $\{0, \dots, n\}$ | Set of all integers between 0 and $n$. |
| $(a, b]$ | The real interval excluding $a$ and including $b$. |

| | Numbers and Arrays |
|---|---|
| $x$ | A scalar |
| $\mathbf{x}$ | A vector (in column form) |
| $W$ | A matrix |
| $\mathbf{x}^T, W^T$ | The transpose of $\mathbf{x}$ or $W$ |
| $\mathbf{x}_i$ | Element $i$ of $\mathbf{x}$ |
| $W_{i,j}$ | Element in row $i$ and column $j$ of $W$ |

| | |
|---|---|
| $\boldsymbol{\theta}_k$ | The $k$-th update of $\boldsymbol{\theta}$ |
| | **Datasets, Distributions and Probability Theory** |
| $\boldsymbol{x}$ | a vector-valued random variable |
| $\boldsymbol{x} \sim p$ | $\boldsymbol{x}$ has distribution $p$ |
| $p(\mathbf{y}\|\mathbf{x})$ | The probability of (output) $\mathbf{y}$ given (input) $\mathbf{x}$ |
| $p_{\text{data}}$ | The data generating distribution |
| $\hat{p}_{\mathcal{D}}$ | The empirical distribution of set $\mathcal{D}$ |
| $\mathbf{x}^{(l)}$ | The $l$-th example (input) of a dataset |
| $\mathbf{y}^{(l)}$ | The label or target associated with $\mathbf{x}^{(l)}$ |
| | **Functions** |
| $f : X \to Y$ | The function $f$ with domain $X$ and range $Y$ |
| $f \circ g$ | Composition of functions $f$ and $g$ |
| $f(\mathbf{x}, \boldsymbol{\theta})$ | function depending on $\mathbf{x}$ and parameters $\boldsymbol{\theta}$. Sometimes $\boldsymbol{\theta}$ is omitted |
| $\frac{dy}{dx}$ | Derivative of $y$ with respect to $x$ |
| $\frac{\partial y}{\partial x}$ | Partial derivative of $y$ with respect to $x$ |
| $\nabla_{\mathbf{x}} y$ | Gradient of $y$ with respect to $\mathbf{x}$ |
| $\frac{\partial f}{\partial \mathbf{x}}$ | Jacobian matrix $J \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$ |
| | **Evaluation** |
| pp | percentage points |
| TP | true positive |
| FP | false positive |
| FN | false negative |

**Table 2.1: General notation used throughout the thesis.**

## 2.2 Machine Learning Overview

As humans, we have a very strong visual system. Without explicitly thinking about it, when we scan our surroundings, we efficiently fulfill tasks, such as localizing and simultaneously classifying objects. This is necessary to achieve higher-order goals, such as drinking a cup of coffee. We need to find the cup and classify the content as coffee. Of course, we not only use our visual system to do this, but it is a very important part of the workflow. When we want to teach a service robot to bring us a fresh cup of coffee, we need to transfer the visual part of the pipeline to an algorithm, such that the robot can use his visual input, acquired by cameras, and make something useful out of it. To make the problem more specific, let us assume that the cup has already been found, and the robot can hold it in front of the camera such that the liquid within the cup is visible in the images it acquires. Hence, we denote image input as $\mathbf{x}$, and the task of the robot is to decide if there is coffee within the cup, given $\mathbf{x}$. To continue with the mathematical modeling, we denote the true answer - *yes* if coffee is present, *no*, else - by $\mathbf{y}$. Furthermore,

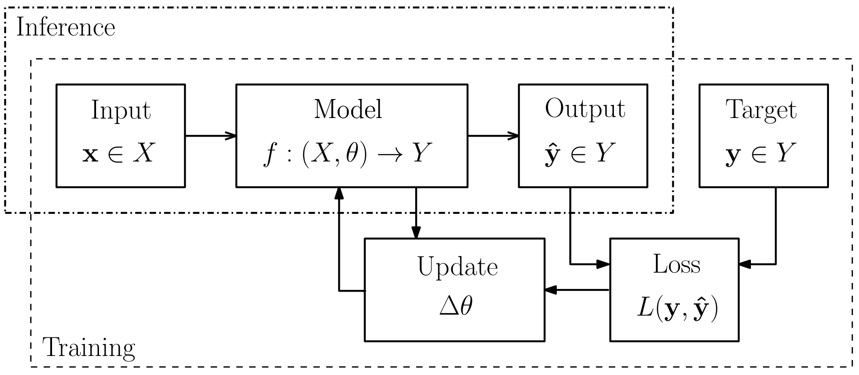


**Figure 2.1: Machine learning pipeline.** For inference, the model $f$ takes input $\mathbf{x}$ and predicts $\hat{\mathbf{y}}$. During training (cf. Section 2.3), a loss $L$ is calculated based on the difference of output $\hat{\mathbf{y}}$ and target $\mathbf{y}$, and using the current model state an update $\Delta\boldsymbol{\theta}$ is computed for the model weights $\boldsymbol{\theta}$.

we assume that there is a true solution to our problem. That means, we assume there is a mapping $f^*$ from inputs $\mathbf{x} \in \{\text{image of a cup}\}$ to $\mathbf{y} \in \{\textit{yes, no}\}$. Unfortunately, $f^*$ is unknown and therefore, we try to find an algorithm to approximate it.

For example, we could use a rule-based approach as follows: We assume the content of the cup is always centered and extract the RGB color values within a circle around the center of the image. If they are close enough to *coffee brown*, i.e., the absolute distance of the mean RGB values from $(39, 19, 0)$ is not larger than $(10, 10, 10)$, the answer should be *yes*, else *no*. This is a simple and fast algorithm, however, it is not very good: the cup needs to be centered and there should be no reflections, *crema*, or other things on the coffee to disturb the real coffee color. Also if these things can be assured, the algorithm is not robust at all, as RGB is not a good color space to measure distances and if the lighting or exposure time is only slightly changed the whole system collapses. Another problem is that we have chosen the parameters of the *coffee brown* color $(39, 10, 0)$ as well as the channel-wise maximum difference $(10, 10, 10)$ using a mixture of intuition and experience. Even if we could come up with more sophisticated color spaces and distances, this major drawback of a rule-based algorithm remains: it is really difficult to tune parameters by hand such that they fit not only for a single input, but such that the rules generalize well for all new, previously unseen inputs. In this case, another class of algorithms to approximate $f^*$ is very practical: machine learning.

In simple words *"a machine learning algorithm is an algorithm that is able to learn from data"* [59]. In comparison to many rule-based algorithms, the benefit for the user is that most of the model parameters do not have to be manually selected. Before we will see many of the advantages and possible disadvantages of machine learning, in the following we will introduce the general concept in a more formal way. Fig. 2.1 gives an overview of the general machine learning pipeline.

Consider an input, e.g., an image or a feature vector $\mathbf{x} \in \mathbb{R}^n$, where $n$ is the input dimension. For example, an image of size $960 \times 720$ in RGB color space has $n = 960 \times 720 \times 3 = 2\,073\,600$ dimensions. We assume that whenever we have discrete input values, e.g., the intensity values of byte images, the input is converted, such that $x_i \in \mathbb{R}, i = 1, \ldots, n$.

The algorithm that should approximate $f^*$ is modeled as a mapping $f : X \subseteq \mathbb{R}^n \to Y, f(\mathbf{x}) = \hat{\mathbf{y}}$, where for the target space $Y$ typically it holds: $Y \subseteq \mathbb{R}^m$ or $Y \subseteq \mathbb{N}_0$. In other words, during *inference*, i.e., when we apply the *model*, $f$ takes $\mathbf{x}$ as input and predicts $\hat{\mathbf{y}}$. For example, if we want to predict the number of bicycles within an image, we have $\hat{\mathbf{y}} \in Y = \mathbb{N}_0$, but if we want to predict the speed of the (single) cyclist within an image, we have $\hat{\mathbf{y}} \in Y = \mathbb{R}_0^+, m = 1$ (assuming the cyclist can only stand still or go forward).

The model $f$ has parameters $\boldsymbol{\theta} \in \mathbb{R}^N$, and, in the settings within this thesis, $N$ is usually in the order of millions. Consider the simple model

$$\hat{\mathbf{y}} = f(\mathbf{x}, \boldsymbol{\theta} = \{W, \mathbf{b}\}) = W\mathbf{x} + \mathbf{b},$$
$$\hat{y}_j = \sum_{i=1}^{n} w_{ji} x_i + b_j, \quad j = 1, \ldots, m.$$

Here, the parameters $\boldsymbol{\theta} = \{W, \mathbf{b}\}$ are the entries within the bias $\mathbf{b} \in \mathbb{R}^m$ and the matrix $W \in \mathbb{R}^{m \times n}$. Because the inputs $x_i$ are weighted by $W$, model parameters are also denoted as model *weights*.

To learn the parameters $\boldsymbol{\theta}$ and measure the quality of our model, we use a dataset $\mathcal{D} = \left\{\{\mathbf{x}^{(l)}, \mathbf{y}^{(l)}\}, l = 1, \ldots, n_{\mathcal{D}}\right\}$, where each input $\mathbf{x}_l$ is annotated or *labeled* in the sense that we know the *ground truth* output or *target* $\mathbf{y}^{(l)}$ that we want to obtain when we apply $f$ on $\mathbf{x}^{(l)}$. $\mathbf{y}^{(l)}$ can be interpreted as measurements of $f^*(\mathbf{x}^{(l)})$ that contain some measurement noise.

Further, suppose that all images $\mathbf{x}$, on which we want to apply our model, are drawn from a true but unknown distribution $p_{\text{data}}$. Hence, we have a random variable $\boldsymbol{x} \sim p_{\text{data}}$. However, as soon as we use a dataset $\mathcal{D}$ with a finite number of $n_{\mathcal{D}}$ samples, these samples have an empirical distribution $\hat{p}_{\mathcal{D}} \neq p_{\text{data}}$. Especially for large models, where the number of parameters $N$ is high, we can often easily achieve that $f(\mathbf{x}^{(l)}, \boldsymbol{\theta}) = \mathbf{y}^{(l)}, l = 1, \ldots, n_{\mathcal{D}}$ holds. However, if we draw a new sample $\mathbf{x} \notin \mathcal{D}$, we get $f(\mathbf{x}, \boldsymbol{\theta}) \neq f^*(\mathbf{x}, \boldsymbol{\theta})$. In other words, our model has learned the dataset $\mathcal{D}$ by heart, but does not generalize well to unseen new data. This problem is referred to as *overfitting*. One way to avoid overfitting is to enlarge the dataset, such that the distribution of the dataset is a good approximation of the real data distribution $\hat{p}_{\mathcal{D}} \approx p_{\text{data}}$. Apart from that, strategies exist to *regularize* the model such that it is less likely to overfit.

Given the current model with parameters $\boldsymbol{\theta}$, we can measure the difference of the prediction $\hat{\mathbf{y}}$ and labels $\mathbf{y}$ with a *loss* or objective function $L : Y \times Y \to \mathbb{R}$. $L$ should be small or zero if the model is good and become large if predictions and targets do not coincide. Examples for $L$ are the $L_p$ losses (usually $p \in \{1, 2\}$)

$$L_p(\hat{\mathbf{y}}, \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|_p = \left(\sum_{j=1}^{m} |\hat{y}_j - y_j|^p\right)^{\frac{1}{p}}$$

or

$$L_\infty(\hat{\mathbf{y}}, \mathbf{y}) = \max_{j=1}^{m} |\hat{y}_j - y_j|.$$

More examples for loss functions are given later in the context of specific applications.

Our goal is to find $\boldsymbol{\theta}^*$ such that we have

$$f(\mathbf{x}^{(l)}, \boldsymbol{\theta}^*) = \hat{\mathbf{y}}^{(l)} \overset{!}{=} \mathbf{y}^{(l)} \tag{2.1}$$

for all $l = 1, \ldots, n_{\mathcal{D}}$. Generally, $\boldsymbol{\theta}^*$ is unknown. Using a loss, (2.1) can be reformulated such that $\boldsymbol{\theta}^*$ is the solution to the following optimization problem:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{\{\mathbf{x},\mathbf{y}\}\in\mathcal{D}} L\left(f(\mathbf{x}, \boldsymbol{\theta}), \mathbf{y}\right) := \arg\min_{\boldsymbol{\theta}\in\Theta} L(f(\boldsymbol{\theta}), \mathcal{D}), \tag{2.2}$$

where $\Theta$ is the parameter space. In (2.2), we assume that $\boldsymbol{\theta}^*$ exists. We will see later that for the types of models used within this thesis, this assumption is valid and we will see that usually more than one solution exists for which (2.2) holds.

## 2.3 Training

As already mentioned, in machine learning, the number of model parameters $N$ is large and we cannot set the model weights by hand. Instead, they are *learned* from data $\mathcal{D}$. If no analytical solution for (2.2) exists, we use an iterative *training* scheme: At the beginning, weights are initialized constant or randomly, e.g., by drawing from a Gaussian distribution, to obtain $\boldsymbol{\theta}_0$. For $k \in \mathbb{N}$, we update the weights by

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \lambda_k \Delta\boldsymbol{\theta}_k, \tag{2.3}$$

where $\lambda_k$ is the *learning rate* or step size of the optimization method. Note that in practice we limit $k < K$ or stop when $\Delta\boldsymbol{\theta}_k$ is close to zero.

With $L$ being our objective function, for a good update of the model weights $\lambda_k \Delta\boldsymbol{\theta}_k$, it holds that

$$L(f(\boldsymbol{\theta}_k), \mathcal{D}) < L(f(\boldsymbol{\theta}_{k-1}), \mathcal{D}), \tag{2.4}$$

i.e., we iteratively decrease the loss. Note that in most cases the objective function is highly non-convex. Moreover, because the number of parameters $N$ is large, it is not feasible to use second order methods for the optimization. This would require $\mathcal{O}(N^2)$ operations and the respective storage to compute and store the Hessian matrix. Hence, to solve (2.1), usually first order nonlinear optimization methods are used.

A well-known and simple optimization method to solve (2.2) is *gradient descent*, first proposed by Cauchy in 1847 [16]: the negative gradient is used for the update direction.

$$(\theta_i)_k = (\theta_i)_{k-1} - \lambda_k \frac{\partial}{\partial\theta_i} L(f(\boldsymbol{\theta}_{k-1}), \mathcal{D}), \quad i = 1, \ldots, N \tag{2.5}$$

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \lambda_k \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{\theta}_{k-1}), \mathcal{D}) \tag{2.6}$$

If in (2.5) $\lambda_k$ is chosen small enough, and $\|\lambda_k\| > 0$, (2.4) holds, as long as $\nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{\theta}_{k-1}), \mathcal{D})$

$\neq \mathbf{0}$, i.e., we are not directly at a local minimum or saddle point. However, exactly meeting a local extremal point is very unlikely for the high dimension of $\Theta$.

We will see later that models used within this thesis are a composition of functions

$$f = f^{(d)} \circ f^{(d-1)} \circ \dots f^{(2)} \circ f^{(1)}, \tag{2.7}$$

where we denote $f^{(l)}$ a layer. An application of $f$ is termed *forward* pass, since the input is passed through the model step by step. Let us denote the parameters of $f^{(l)}$ by $\boldsymbol{\theta}^{(l)}$. We thus get

$$f(\mathbf{x}, \boldsymbol{\theta}) = f^{(d)}(\mathbf{x}, \boldsymbol{\theta}) := f^{(d)}\left(f^{(d-1)}\left(\dots f^{(2)}\left(f^{(1)}(\mathbf{x}, \boldsymbol{\theta}^{(1)}\right), \boldsymbol{\theta}^{(2)}\right) \dots, \boldsymbol{\theta}^{(d-1)}\right), \boldsymbol{\theta}^{(d)}\right). \tag{2.8}$$

To calculate the gradient of the loss with respect to the model weights in (2.5), for neural networks *backpropagation* [166] is the method of choice. The idea is to first perform a forward pass and store the intermediate results $F^{(l)} := f^{(l)}(\mathbf{x}, \boldsymbol{\theta}^{(l)})$, for $l = 1, \dots, d$.

If we define $\boldsymbol{\theta} := (\boldsymbol{\theta}^{(1)^T}, \dots, \boldsymbol{\theta}^{(d)^T})^T$, to update our model we need to compute the partial derivatives of $f$ with respect to all model weights. Using the chain rule, we get

$$\frac{\partial f}{\partial \theta_k^{(l)}} = \frac{\partial f}{\partial F^{(l)}} \frac{\partial F^{(l)}}{\partial \theta_k^{(l)}} = \frac{\partial f}{\partial F^{(l+1)}} \frac{\partial F^{(l+1)}}{\partial F^{(l)}} \frac{\partial F^{(l)}}{\partial \theta_k^{(l)}}, \tag{2.9}$$

for all $j = 1, \dots, d$ and $k$. Hence, when we calculate the gradient, we apply the chain rule and go from back to front, i.e., perform a *backward pass*. We start with the loss and compute the gradient to the last layer's activations. From there, for each layer, we compute the gradient with respect to the weights and also *propagate the error* further by calculating the gradient with respect to the layer's inputs. This means that for each layer we need to calculate $\partial F^{(l)} / \partial F^{(l-1)}$ and $\partial F^{(l)} / \partial \boldsymbol{\theta}^{(l)}$.

### 2.3.1 Stochastic Gradient Descent

In practice, the gradient computation is done for inputs $\mathbf{x}$ and targets $\mathbf{y}$ of the dataset $\mathcal{D}$. Taking only a single pair $\{\mathbf{x}, \mathbf{y}\} \in \mathcal{D}$, our model quickly overfits to this particular example. On the other hand, averaging the gradient over the whole dataset $\mathcal{D}$, is computationally expensive. Therefore, *stochastic gradient descent* (SGD) is the method of choice, where a small *batch* $\mathcal{B} \subset \mathcal{D}$ of randomly chosen examples is used to approximate the model's gradient.

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \lambda_k \frac{1}{n_{\mathcal{B}}} \sum_{\{\mathbf{x}, \mathbf{y}\} \in \mathcal{B}} \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}, \boldsymbol{\theta}_{k-1}), \mathbf{y}) \tag{2.10}$$

The number of examples $n_{\mathcal{B}} = |\mathcal{B}|$ is termed *batch size*, and during learning $n_{\mathcal{D}} / n_{\mathcal{B}}$ iterations are referred to as one *epoch*. Hence, after one epoch the model has seen each example in $\mathcal{D}$ once.

The preferred batch size depends on the variations of images and objects within the dataset: For small variations, typically a small batch size is sufficient, whereas for larger variations a larger batch size might improve the result. Moreover, because for a larger batch size the approximation of the gradient is better, usually also a larger learning rate can be chosen.

Adaptive strategies for choosing the learning rate $\lambda_k$, such as ADAM [92], adadelta [205], adagrad [33] or RMSprop exist (see e.g., [165] for an overview). However, in many cases the learning rate is initialized with a constant and only multiplied by a factor $\gamma < 1$ at specific, manually chosen iterations. The *steps* of the learning rate are done when the loss stagnates. However, to find these points of time requires to train the model multiple times or a software interface that allows to adapt the learning rate online during training.

For networks with weights that are initialized randomly, the training can be very unstable in the beginning, especially if the batch size is very small. Therefore, it is common practice to start the training with a warmup learning rate scheme: first, the initial learning rate is reduced by a warmup factor and then, it is iteratively increased over the first $n_W$ warmup iterations [60].

### 2.3.2 Momentum

It is well known that gradient descent can be very slow if the problem is poorly conditioned. In this case, the direction of the gradient is heavily changing from iteration to iteration, especially if the learning rate is not set adaptively. This effect can be intensified if the randomly chosen batches in SGD lead to gradients with opposite directions.

A popular method to reduce those effects is to average the latest gradients in SGD with *momentum* [153, 182], where exponential smoothing is applied to the gradient:

$$\mathbf{v}_0 = \mathbf{0}, \tag{2.11}$$

$$\mathbf{g}_k = \frac{1}{n_\mathcal{B}} \sum_{\{\mathbf{x},\mathbf{y}\}\in\mathcal{B}} \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}, \boldsymbol{\theta}_{k-1}), \mathbf{y}), \tag{2.12}$$

$$\mathbf{v}_k = \eta \mathbf{v}_{k-1} - \mathbf{g_k}, \tag{2.13}$$

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \lambda_k \mathbf{v}_k, \tag{2.14}$$

where $\eta \in [0, 1)$ is the momentum hyperparameter. Typically, the momentum is set relatively high to 0.9 or even 0.95.

### 2.3.3 Weight Decay

Since large deep learning models are very prone to overfitting, different *regularization* techniques exist. These should help the model to generalize better to unseen data. The idea of *weight decay* [71, 96, 138] is to add an additional $L^2$ norm penalty to the objective function

$$L(\boldsymbol{\theta}, \mathcal{B}) = \frac{1}{n_\mathcal{B}} \sum_{\{\mathbf{x},\mathbf{y}\}\in\mathcal{B}} L(f(\mathbf{x}, \boldsymbol{\theta}) + \frac{\alpha}{2} \|\boldsymbol{\theta}\|_2, \tag{2.15}$$

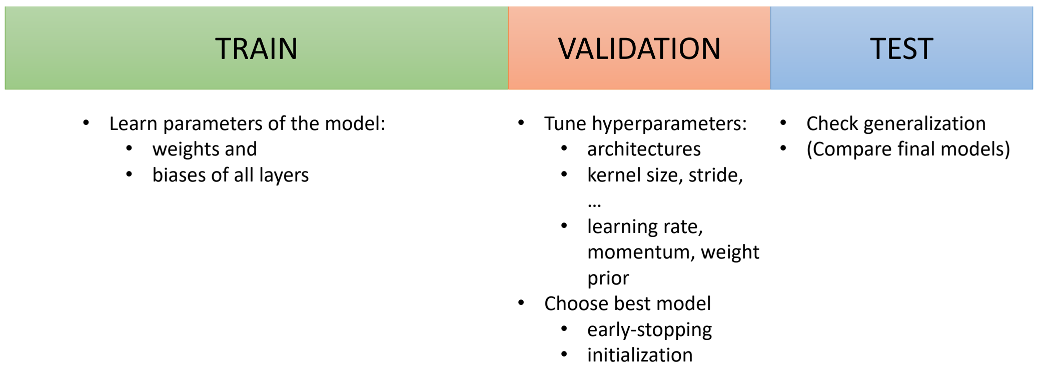


**Figure 2.2: Dataset splits.** We learn the parameters on the training set, measure the quality of the model and choose the best hyperparameters on the validation set, and finally measure how well the model generalizes on the test set.

where $0 < \alpha \ll 1$ is referred to as *weight prior*. The norm penalty enforces a decay of the parameter norm such that parameters are only increased if this helps to reduce the error of the training examples. The gradient computation of the loss in (2.12) then changes to

$$\mathbf{g}_k = \frac{1}{n_{\mathcal{B}}} \sum_{\{\mathbf{x},\mathbf{y}\}\in\mathcal{B}} \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}, \boldsymbol{\theta}_{k-1}), \mathbf{y}) + \alpha\boldsymbol{\theta}. \tag{2.16}$$

### 2.3.4 Dataset Splits and Early Stopping

Another simple and natural way to regularize the model is the use of *early stopping*. Until now, we have always referred to a single dataset $\mathcal{D}$, but in practice it is important to *split* this dataset into at least two disjoint sets: one to use for training the model and update the parameters, and one to measure the current quality of the model and especially its ability to generalize well to new and unseen data. As shown in Fig. 2.2, in machine learning, we usually split the dataset into three disjoint sets: the training set $\mathcal{D}_{\text{train}}$, the validation set $\mathcal{D}_{\text{val}}$, and the test set $\mathcal{D}_{\text{test}}$.

We use $\mathcal{D}_{\text{train}}$ for training the model, i.e., computing the loss and the gradient with respect to the parameters on batches $\mathcal{B} \in \mathcal{D}_{\text{train}}$ to apply SGD. During training, we can evaluate the current model state on the validation set $\mathcal{D}_{\text{val}}$ in intervals of $K$ iterations. For evaluation, we can either compute the loss or any other quality measure $Q$. It is important to note that the parameters of the model are not updated during these intermediate evaluations. Whenever the loss is lower or the quality measure $Q$ is higher than during the previous evaluation, we store the current model as the best model and write it to disk. At some point during the training process, we will eventually start to overfit to $\mathcal{D}_{\text{train}}$, i.e., the loss on $\mathcal{D}_{\text{train}}$ continues to decrease, but the loss on $\mathcal{D}_{\text{val}}$ starts to increase again. At the end of the training, we still know at which evaluation point we achieved the best result and can use this best model as the final result.

The validation set is not only helpful for early stopping. The models also have hyperparameters, such as the learning rate strategy, momentum, or weight prior. Moreover, we will see in the following sections, that different architectural choices can be made by choosing other hyperparameters. In many cases, the model's performance is also influenced by the random initialization of the models parameters. Also here, the validation set can be used to choose the best model among differently initialized versions. Since we are indirectly using the examples of the validation set to choose the best model or tune the hyperparameters, this can be seen as another optimization.

Finally, if we are confident that for this type of model we have chosen a good set of hyperparameters and selected a good initialization and the best iteration to stop the training, we can evaluate how well this final model generalizes to the still unseen images of the test split $\mathcal{D}_{\text{test}}$. If the test split is a good representation of the real data distribution, this directly relates to the expected error rate of our model when it is used within a machine. In benchmarks or challenges, the test set is also used to compare different models against each other. However, one needs to be careful not to use the test set to tune the hyperparameters of a model as this would just make it another kind of validation set.

To get a better idea of these so far quite abstract introduced concepts, we will now explain different types of models in the next sections.

## 2.4 Classifiers

Because different techniques to classify a feature vector $\mathbf{x}$ into one of $n_c$ different categories of interest will often reappear during the thesis, in this section we want to give a brief overview of different classifiers.

### 2.4.1 Support Vector Machines

Support vector machines (SVM) [23] have been very popular before the massive use of deep-learning-based end-to-end trainable neural networks. SVMs belong to the class of maximum margin classifiers. In their initial form, they are built for a two-class problem $\mathbf{y} \in \{-1, 1\}$. For data points that are linearly separable, finding the hyperplane that separates the points with maximum margin can be formulated as a quadratic programming problem. It has been shown that instead of all data points, only a sparse subset of *support vectors* has to be considered for the solution and, therefore, the large problem can be split into some small subproblems which makes the method efficient. If the data is not linearly separable, *kernel* functions are used to transform the features into a higher-dimensional space such that linear separability is given. However, using the *kernel trick*, this does not mean that computations have to be carried out in this higher-dimensional space, but can be done in the input space. Because even with the use of suitable kernels not all datasets are linearly separable, Cortes and Vapnik [23] have added slack-variables to the problem formulation such that a solution can be found while outliers are omitted.

Today, SVMs are often replaced by MLPs or CNN classifiers (see below). The reason is that for multi-class classifications — which are the topic of this thesis — the classification has to be split into $n$ one vs. rest SVM classifiers or into $n(n-1)/2$ one vs. one SVMs. This is more complicated to evaluate than a simple MLP. For an in depth explanation of SVMs, the interested reader is referred to Bishop [7] or Schölkopf and Smola [169].

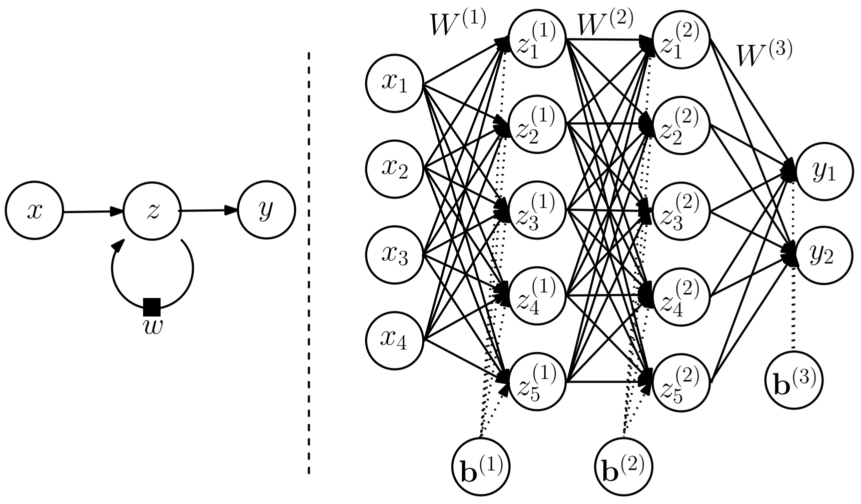


**Figure 2.3: Graph of a neural network.** (*left*) Example for a recurrent neural network with a cycle. (*right*) Example for a feed-forward neural network or MLP with two hidden layers.

### 2.4.2 Neural Networks and MLPs

Inspired by biological systems and in particular the human brain, *neural networks* have been proposed as models that consist of a graph of interconnected *neurons* [134, 162, 198]. Each neuron or *unit* is a simple parameterized processing function that applies an *activation function* (see below) to a weighted sum of the transformed inputs

$$z_j = h(\sum_{i=1}^{n} w_{ji}\phi(x_i) + b_j), \tag{2.17}$$

where $\mathbf{b} = \{b_j\}$ is the *bias* and $W = \{w_{ji}\}$ is the weight matrix. In its most frequently used and simplest form, $\phi$ is the identity function, which leads to a linear combination of the inputs to which the activation function $h$ is applied:

$$z_j = h(\sum_{i=1}^{n} w_{ji}x_i + b_j). \tag{2.18}$$

For general neural networks (NN), there are no further restrictions for the connections of neurons within a graph. For example, in a *recurrent* neural network that is used for processing of a sequence of input data, it is common that cycles occur and in particular a neuron can be connected to itself.

However, in this thesis we are focused on directed acyclic graphs or *feed-forward neural networks* that have one or several inputs $\mathbf{x}$ that are processed by the network step by step going through multiple consecutive *layers* to produce outputs $\mathbf{y}$. In the example in the right of Fig. 2.3 we have that

$$\begin{aligned} y_j &= \sum_{k=1}^{5} w_{jk}^{(3)} z_k^{(2)} + b_j^{(3)} \\ &= \sum_{k=1}^{5} w_{jk}^{(3)} \left( \sum_{l=1}^{5} w_{kl}^{(2)} z_l^{(1)} + b_k^{(2)} \right) + b_j^{(3)} \\ &= \sum_{k=1}^{5} w_{jk}^{(3)} \left( \sum_{l=1}^{5} w_{kl}^{(2)} \left( \sum_{m=1}^{4} w_{lm}^{(1)} x_m + b_l^{(1)} \right) + b_k^{(2)} \right) + b_j^{(3)}. \end{aligned} \tag{2.19}$$

Using $F^{(0)} := \mathbf{x}$, $F^{(3)} := \mathbf{y}$, and defining

$$f^{(l)} : \mathbb{R}^{n^{(l-1)}} \to \mathbb{R}^{n^{(l)}}, f^{(l)}(F^{(l-1)}) := W^{(l)} F^{(l-1)} + \mathbf{b}^{(l)}, \tag{2.20}$$

where $n^{(l-1)}$ is the number of input and $n^{(l)}$ the number of output *units* for each layer, (2.19) can be reformulated as

$$\mathbf{y} = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}))) := f(\mathbf{x}). \tag{2.21}$$

From (2.21), the meaning of layers becomes clear and we can also see the feed-forward structure. Because each neuron of the input is connected to each neuron of the current layer, these layers are also named *fully connected* (fc) or *dense* layers. This particular type of NNs is termed *multi layer perceptron* (MLP) [166, 197], in our example case with two *hidden* layers in the middle. Those intermediate units are hidden, because their *"actual or desired states are not specified by the task"* [166, p. 533], which is in contrast to an elementary perceptron without hidden layers and a single output that can only take values $y \in \{-1, 1\}$ due to a thresholding activation function [162]. The total number of layers is called the *depth* of a NN and as in recent years the depth of common NNs was dramatically increased, the term *deep learning* arose.

But interestingly, Hornik et al. [75] have shown that already MLPs with a single hidden layer are a class of universal approximators in the sense that they can approximate any Borel measurable function from a finite dimensional space to another up to any desired degree of accuracy if enough hidden units are present. However, if the input dimension is large and the first hidden layer has a large number of neurons, the weight matrix $W$ gets very large in turn. Therefore, instead of increasing the dimension of a single hidden layer, it is more common to find a good trade-off between layer dimension and depth of the NN.

To compute the derivatives of a fc layer with respect to its inputs we denote $\boldsymbol{\partial}^{(l)} = \partial L / \partial F^{(l)}$ and obtain

$$\begin{aligned} &\partial_j^{(l)} = \frac{\partial L}{\partial F_j^{(l)}} = \sum_k \frac{\partial L}{\partial F_k^{(l+1)}} \frac{\partial F_k^{(l+1)}}{\partial F_j^{(l)}} = \sum_k \partial_k^{(l+1)} \frac{\partial \sum_{j'} w_{kj'}^{(l+1)} F_{j'}^{(l)} + b_k^{(l+1)}}{\partial F_j^{(l)}} = \sum_k w_{kj}^{(l+1)} \partial_k^{(l+1)}, \\ &\boldsymbol{\partial}^{(l)} = W^{(l+1)^T} \boldsymbol{\partial}^{(l+1)}, \end{aligned} \tag{2.22}$$

and similarly for the derivatives with respect to the weights and biases

$$\frac{\partial L}{\partial w_{ji}^{(l)}} = \sum_{j'} \frac{\partial L}{\partial F_{j'}^{(l)}} \frac{\partial F_{j'}^{(l)}}{\partial w_{ji}^{(l)}} = \sum_{j'} \partial_{j'}^{(l)} \frac{\partial \sum_{i'} w_{j'i'}^{(l)} F_{i'}^{(l)} + b_{j'}^{(l)}}{\partial w_{ji}^{(l)}} = \partial_j^{(l)} F_i^{(l)}, \tag{2.23}$$

$$\frac{\partial L}{\partial W^{(l)}} = \boldsymbol{\partial}^{(l)} {F^{(l)}}^T, \quad \text{and} \tag{2.24}$$

$$\frac{\partial L}{\partial \mathbf{b}^{(l)}} = \boldsymbol{\partial}^{(l)}. \tag{2.25}$$

## Activation Functions

While in the original perceptron the activation function was a simple threshold

$$h(z) = \begin{cases} +1, & \text{if } z \geq 0 \\ -1, & \text{otherwise,} \end{cases} \tag{2.26}$$

today there are many choices for activation functions. In a NN with multiple layers that should approximate a nonlinear function, it is important that the activation function is nonlinear. Otherwise, the network has only linear components and since the compostion of linear functions is again linear, the MLP would be linear as a whole. Hence, it would make no sense to use more than one layer. Moreover, we will see in Section 2.3 that for the training of NNs the gradient needs to be computed. Therefore, activation functions should be (at least partially) continuously differentiable. Moreover, to be useful within the learning process and avoid vanishing gradients the gradient of the activation function should be at least partially non-zero, which does e.g., not hold for the threshold in (2.26).

In earlier works [105, 106], mostly saturating functions, such as the hyperbolic tangent

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.27}$$

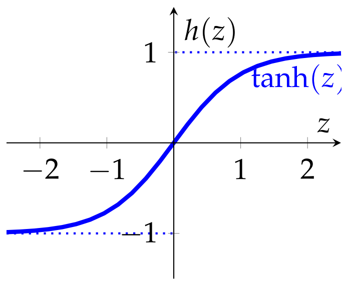
or the sigmoid or logistic function, which is a scaled and shifted version of tanh

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z} = \frac{1}{2}\left(1 + \tanh\frac{z}{2}\right), \tag{2.28}$$

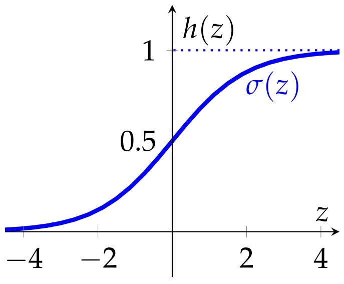
were used. Note that the derivatives are convenient from a computational perspective when the result of $\tanh(z)$ or $\sigma(z)$ has been stored:

$$\frac{d}{dz}\sigma(z) = \frac{e^z}{(1+e^z)^2} = \sigma(z)(1-\sigma(z)), \tag{2.29}$$

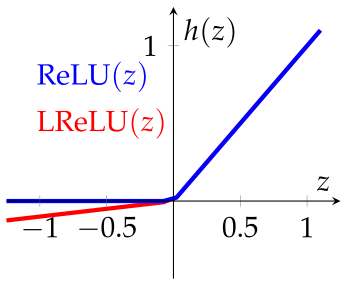
$$\frac{d}{dz}\tanh(z) = 1 - \tanh^2(z). \tag{2.30}$$



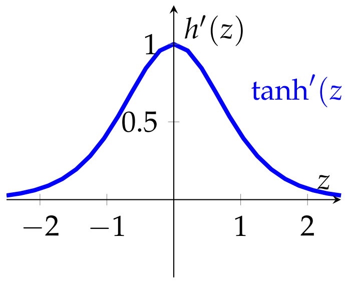
**(a)** $\tanh(z)$



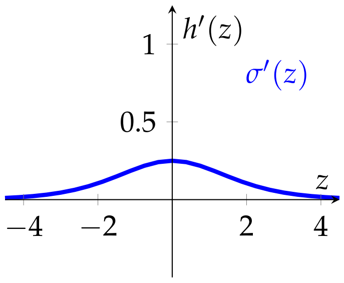
**(b)** $\sigma(z)$



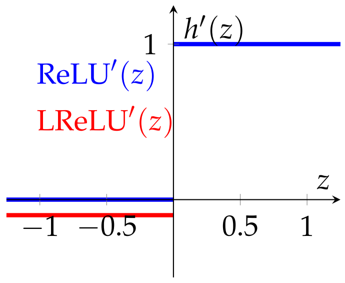
**(c)** ReLU$(z)$ and LReLU$(z)$



**(d)** $\frac{d}{dz}\tanh(z)$



**(e)** $\frac{d}{dz}\sigma(z)$



**(f)** $\frac{d}{dz}$ ReLU and $\frac{d}{dz}$ LReLU

**Figure 2.4: Activation functions and derivatives.** *((a)–(c))* The most popular activation functions are the bounded and continuously differentiable hyperbolic tangent and sigmoid functions or the widely used unbounded ReLU or LReLU functions. *((e)–(f))* shows their derivatives. The rapidly decaying derivative of tanh and $\sigma$ for large absolute values of $z$ can lead to vanishing gradients.

A more modern activation function is the *rectified linear unit* (ReLU) [66, 84, 144]

$$\operatorname{ReLU}(z) = \begin{cases} z, & \text{if } z \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.31}$$

Interestingly, ReLU is not continuously differentiable at $z = 0$

$$\frac{d}{dz}\operatorname{ReLU}(z) = \begin{cases} 1, & \text{if } z \geq 0, \\ 0, & \text{otherwise,} \end{cases} \tag{2.32}$$

but this does not harm its popularity due to its computational efficiency.

A variant of the ReLU with non-zero gradient for negative inputs is the leaky ReLU (LReLU) [130] activation

$$\operatorname{LReLU}(z) = \begin{cases} z, & \text{if } z \geq 0, \\ \alpha z, & \text{otherwise,} \end{cases} \tag{2.33}$$

where usually $0 < \alpha \ll 1$.

The introduced activation functions are shown in Fig. 2.4. Recently, several new activation functions, such as SiLU [35], Swish [155], or Mish [137] have shown to be outperforming ReLU in some cases, but not without some computational overhead.

**Softmax**

In the context of MLP and classification we want to predict a probability for each of the $n_c$ different categories in our data. Therefore, the MLP has $n_c$ output neurons $z_1, \dots, z_{n_c}$ that can take any value $z_i \in \mathbb{R}$. One possibility to achieve that the outputs can be interpreted as probabilities $y_i \in [0, 1]$ is to apply a sigmoid $y_i = \sigma(z_i)$. However, in most cases, we want that the probabilities for all different categories sum up to one. This is where the softmax function can be used

$$S_j := \mathrm{softmax}(\mathbf{z})_j = \frac{\exp(z_j)}{\sum_{k=1}^{n_c} \exp(z_k)}, \quad j = 1, \dots, n_c, \tag{2.34}$$

$$\frac{\partial}{\partial_j} S_j = S_j(1 - S_j), \quad \frac{\partial}{\partial_i} S_j = -S_i S_j. \tag{2.35}$$

The use of the softmax is very popular and it generally works very well, as it outputs high probabilities if the prediction is correct. However, if the model is unsure, i.e., two different *logits* $z_i$ and $z_j$ are on the same level, the exponential function inside the softmax usually leads to a clear winner with a very high confidence. For example, for $\mathbf{z} = (-5, -3, -2, 10, 12)^T$, we get $\mathrm{softmax}(\mathbf{z}) \approx (0, 0, 0, 0.12, 0.88)^T$. As stated in [59], the softmax is rather a continuously differentiable approximation of the arg max function than the max function. Thus, the interpretation of the softmax as the model's confidence is very difficult.

## 2.5 CNNs for Feature Extraction

Consider an image of a pill that you want to classify into one of the three defect categories *contamination*, *crack*, or *good*. A popular strategy to accomplish this task is visualized in Fig. 2.5: A rule-based approach is used to extract features $\tilde{x} \in \mathbb{R}^M$ from the input image $\mathbf{x} \in \mathbb{R}^n$. In a second, independent step, these features are fed into a classifier that outputs for each class a probability how likely it is to be present, given the input $\hat{\mathbf{y}} = p(\mathbf{y}|\tilde{x}) = p(\mathbf{y}|x)$. In our setting with a labeled dataset $\mathcal{D}$, for the second part we have targets $\mathbf{y}$, such that a machine learning approach can be used, while we do not know what the intermediate outputs $\tilde{\mathbf{x}}$ should look like. Hence, the first part is in many cases hand-crafted. That means, the engineer designs the feature extractor based on his expert knowledge and intuition.

The goal of the feature extraction is to compress the information ($M \ll n$) and obtain a representation of the image that is invariant to certain transformations, such as translations, rotations, or deformations. Moreover, algorithms [6, 128, 164] have been found that make $\tilde{\mathbf{x}}$ invariant to changes in illumination and scale or robust to noise. To obtain the dimension reduction, feature descriptors are computed only at *keypoints* that have to be found in turn. Today, there are numerous ways to identify keypoints in an image [49, 67, 128, 111]. An example is shown in Fig. 2.6.

Both the descriptors and the keypoint detectors are often based on *filtered* versions

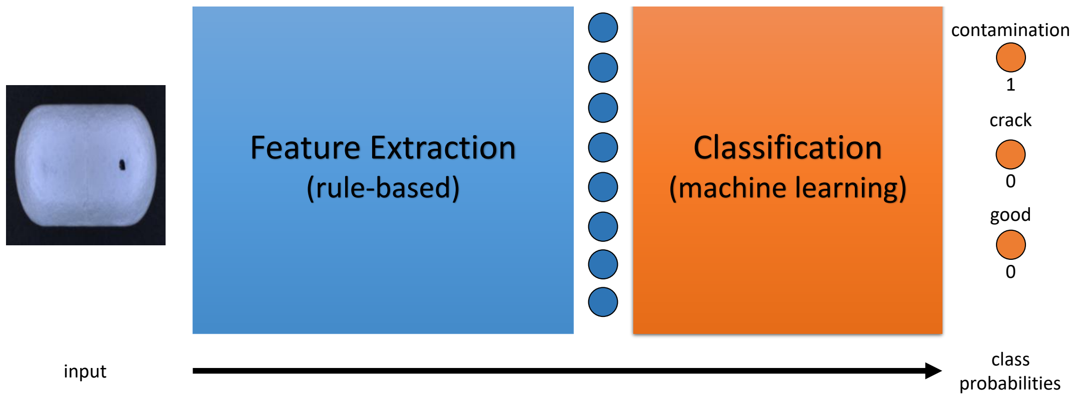


**Figure 2.5: Rule-based workflow.** From an input image $\mathbf{x}$ as shown on the left, first, features $\tilde{\mathbf{x}}$ are extracted by a rule-based algorithm. Second, these features are fed into a trainable classifier that outputs class probabilities $p(\mathbf{y}|\tilde{\mathbf{x}}) = p(\mathbf{y}|\mathbf{x})$. Typically, the first part is hand-crafted, while the second part is a machine learning based classifier.

of the image, e.g., to approximate the image gradient or to smooth the image. For multi-channel images $I_{d,r,c}$, a filtering operation is nothing but a *convolution* of the image, which can be written in its discrete form as

$$F_{r,c} = (K * I)_{r,c} = \sum_k \sum_m \sum_n I_{k,r-m,c-n} K_{k,m,n} \quad , \tag{2.36}$$

where $K$ is a *kernel* and the summation for $m$ and $n$ over spatial dimensions is only done over the domain where the kernel has non-zero values, in practice. Each kernel produces a single-channel output. Note that a convolution is the same as a *cross-correlation* with a flipped kernel. In machine-learning libraries the convolution layer is often implemented as a cross-correlation because it is easier to read and for learned kernels it makes no difference if the kernel is flipped or not. A comprehensive introduction to filtering is given in [180, Section 3.2.3].

In Fig. 2.6, we show examples of images that are transformed applying convolutions. For example, a discretized version of a two-dimensional symmetric Gaussian distribution is often used to smooth an image, i.e., to remove sensor noise. For example, a discrete filter-kernel of size $5 \times 5$ that approximates a Gaussian with $\sigma = 1$ is given by:[1]

$$K = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} . \tag{2.37}$$

Many low-level computer vision tasks are based on edges. Therefore, a single-channel image is interpreted as a function that outputs gray values for each pixel-position $I(i, j)$. An edge can then be defined as connected positions within the image where the gradient

[1]R. Fisher, S. Perkins, A. Walker, and E. Wolfart, Hypermedia Image Processing Reference, 2003, https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm, accessed 2021-02-01.

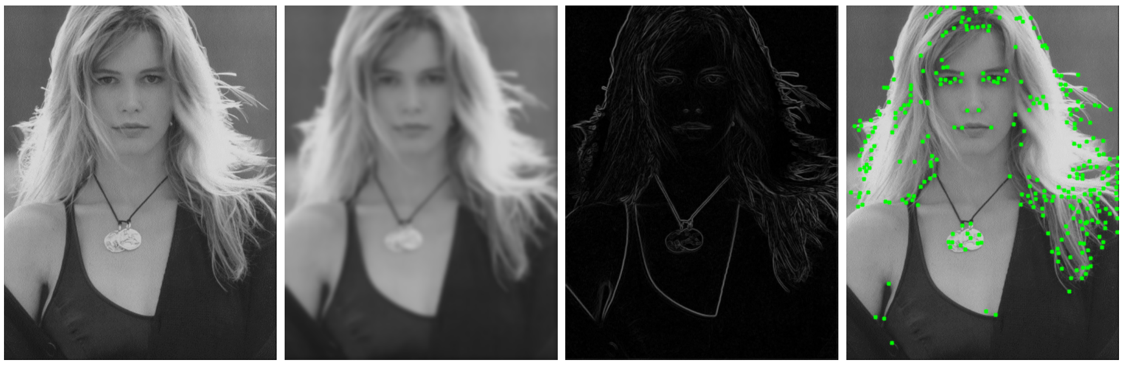


**Figure 2.6: Filters and keypoints.** (*From left to right*) The input image, and after a convolution with a Gaussian kernel of size $11 \times 11$. Edge amplitudes extracted with the Sobel operator. Keypoints obtained with the Harris corner detector.

of this function reaches a local maximum. The simplest kernels that can be used to approximate the one-dimensional directional derivatives into column and row directions are $K = [1, -1]$ and $K = [1, -1]^T$, respectively. Depending on the origin of the filter, their application is the same as applying the forward or backward finite differences algorithm to approximate the directional derivative. Accordingly, $K = [1, 0, -1]$ or $K = [1, 0, -1]^T$ we obtain the central difference as numerical approximation to the directional derivatives.

The idea of the Sobel operator is to combine the central difference filters with a perpendicular smoothing:

$$G_c = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad G_r = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{2.38}$$

To infer the edge location efficiently, typically the pixel-wise edge magnitude $\|G\|_2 = \sqrt{(G_r * I)^2 + (G_c * I)^2}$ or $\|G\|_1 = |(G_r * I)| + |(G_c * I)|$ is thresholded. The edge direction can be obtained by $\phi = -\text{atan2}((G_r * I), (G_c * I))$.

In the past, engineers came up with brilliant ideas to construct deterministic algorithms that extract information from images such that they could solve their specific tasks. Convolutions have often played an important role as only a small number of parameters, $HW$ for a single-channel filter of size $H \times W$, have to be set. Moreover, especially for separable or recursive filters, the convolution operation can be implemented efficiently. Another important benefit of the convolution operation is its translational equivariance, i.e., no matter where an object is located within the image, the same local answer to a filter can be expected and thus the filter responses are only translated versions of each other. More formally, if the input is translated by function $\pi$ before the filter $F$ is applied, there exists a function $\tilde{\pi}$ such that

$$F(\pi(\mathbf{x})) = \tilde{\pi}(F(\mathbf{x})). \tag{2.39}$$

Note that invariance is a special case of equivariance, where $\tilde{\pi}$ is the identity mapping.

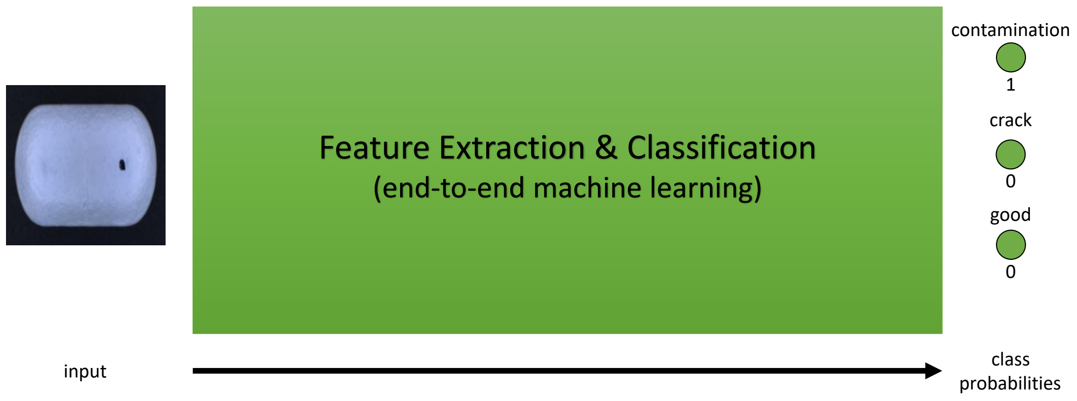

Feature Extraction & Classification
(end-to-end machine learning)
contamination
1
crack
0
good
0
input
class probabilities

**Figure 2.7: End-to-end workflow.** From an input image $\mathbf{x}$ as shown on the left, the model simultaneously extracts features and uses them inside a classifier to output class probabilities $p(\mathbf{y}|\mathbf{x})$ in an end-to-end manner.

If an object can only be identified by more complex features, such as certain textures or shapes, it can only be detected with large filter masks and robustly setting the parameters of those is infeasible. *Convolutional neural networks* (CNN) [51, 103, 104] are a parameter efficient way to address this issue by incorporating the feature extraction into the machine learning approach. Instead of first extracting features and then applying a classifier, a single model is used that takes an input image $\mathbf{x}$ and directly outputs class probabilities $p(\mathbf{y}|\mathbf{x})$. Since the user has no direct insight into the intermediate features, these kinds of models are referred to as *end-to-end* or *blackbox* models (cf. Fig. 2.7).

CNNs have the same layer-wise structure as MLPs and also here, no feedback loops occur. The feature extraction part of a CNN usually consists of several consecutive blocks that have the following repetitive structure: a convolution layer is applied, the output is normalized by an affine transformation (e.g., batch normalization) and fed into an activation function, before the next convolution, normalization and activation block starts. After multiple such blocks, the output is downsampled by a max-pooling layer and the process is continued, before finally an MLP-like classifier is attached. We will now explain each of the new steps in more detail and explain certain design principles.

### Convolution Layer

For image processing, convolutional layers are very effective as they can be efficiently applied to inputs with a high spatial dimension. Moreover, as already mentioned, by construction they bring a translational equivariance with them.

The $l$-th 2D convolution layer has $d^{(l)}$ different 3D filters $K^{(l,j)}, j = 1, \dots d^{(l)}$ with the same dimensions $(D^{(l)}, H^{(l)}, W^{(i)})$, where $D^{(l)}$ is the filter kernel depth that can be directly inferred from the input dimensions $(d^{(l-1)} \overset{!}{=} D^{(l)}, h^{(l-1)}, w^{(l-1)})$, and $H^{(l)}, W^{(l)}$ are its row and column spatial dimensions. Each filter produces a single-channel response, referred to as a channel in the output *feature map*. Hence, each value in the output feature map $F^{(l)}$ is one response of a filter kernel to the input feature map $F^{(l-1)}$. Accordingly, the depth of the output of a convolution layer is equal to the number of filters $d^{(l)}$. For convenience, we will omit the superscript $(l)$ if it is clear that we mean the $l$-th layer. For the output feature map, similar to (2.36), we have

$$F^{(l)}_{d,r,c} = (K^{(l,d)} * F^{(l-1)})_{r,c} = \sum_{k=0}^{D-1} \sum_{m=-\lfloor H/2 \rfloor}^{\lfloor H/2 \rfloor} \sum_{n=-\lfloor W/2 \rfloor}^{\lfloor W/2 \rfloor} F^{(l-1)}_{k,r-m,c-n} K^{(l,d)}_{k,m,n} \ . \tag{2.40}$$

In the same way as for a dense layer, also here a bias term can be added. However, modern networks are usually using batch normalization (see below), which makes the bias within the convolution layer obsolete. Therefore, we neglect it here for convenience.

To preserve the spatial dimension of the input, the input feature map is *padded* with half the kernel size $P_r = \lfloor H/2 \rfloor, P_c = \lfloor W/2 \rfloor$, such that the kernel just fits into the feature map when the convolution is applied to its border pixels. In most implementations, a simple zero-padding is used, although the introduced edges can lead to artifacts that can even be carried forward through multiple consecutive layers. Other methods, such as constant padding, or mirroring add computational overhead and are used rarely.

The learnable parameters $\boldsymbol{\theta}$ of a convolution layer are the kernel weights. They are usually randomly initialized before the training starts. Moreover, the layer has certain *hyperparameters* that affect the architecture of the network and that the user has to choose either manually or by the use of an expensive meta-learning algorithm.

Larger spatial kernel dimensions $H, W$ have access to a wider local context around the center pixel. Especially when multiple convolution layers are connected, the area within the input image that influences the value within the current feature map is systematically enlarged. This area is also referred to as the *receptive field*. However, on the one hand, it is not always preferable to have a large receptive field as the result might be influenced by prominent objects in the neighborhood. On the other hand, the storage for the $DHW$ parameters is larger and the computational complexity $\mathcal{O}(h^{(l-1)}w^{(l-1)}d^{(l-1)}H^{(l)}W^{(l)})$ is increased. Therefore, small kernel sizes of $3 \times 3$ up to $7 \times 7$ are the common choice.

An alternative to increase the receptive field without increasing the storage and computation cost is to use a *dilated* convolution, which means that the kernel is spread by $L_r, L_c$ (omitting the exact summation bounds)

$$F^{(l)}_{d,r,c} = \sum_k \sum_m \sum_n F^{(l-1)}_{k,r-L_r \cdot m, c-L_c \cdot n} K^{(l,d)}_{k,m,n}. \tag{2.41}$$

In most cases, a symmetric dilation is used such that we can simply write $L = L_r = L_c$.

The processing of an image can be accelerated if the feature map dimensions are successively reduced. Another reason for the size reduction is that the algorithm is forced to compress the essential information and only keep what is really necessary. One option to obtain a spatial dimension reduction are *strided* convolutions, namely

$$F^{(l)}_{d,r,c} = \sum_k \sum_m \sum_n F^{(l-1)}_{k,r \cdot S_r - m, c \cdot S_c - n} K^{(l,d)}_{k,m,n} \ , \tag{2.42}$$

where $S_r, S_c$ are the stride in row and column dimension and the dilation was omitted for readability. Using a stride means that the filter is only applied at every $S_r$-th and $S_c$-th pixel location in row and column direction, respectively. For example, using a stride $S = S_r = S_c = 2$ halves the feature map width and height and leads to a total dimension reduction by a factor of four.

The derivatives of the convolution layer with respect to its inputs can be calculated as follows:

$$\begin{aligned}
\partial^{(l)}_{d,r,c} := \frac{\partial L}{\partial F^{(l)}_{d,r,c}} &= \sum_{d'} \sum_{r'} \sum_{c'} \frac{\partial L}{\partial F^{(l+1)}_{d',r',c'}} \frac{\partial F^{(l+1)}_{d',r',c'}}{\partial F^{(l)}_{d,r,c}} \\
&= \sum_{d'} \sum_{r'} \sum_{c'} \partial^{(l+1)}_{d',r',c'} \frac{\partial \sum_k \sum_m \sum_n F^{(l)}_{k,r'-m,c'-n} K^{(l,d')}_{k,m,n}}{\partial F^{(l)}_{d,r,c}} \\
&= \sum_{d'} \sum_{r'} \sum_{c'} \partial^{(l+1)}_{d',r',c'} K^{(l,d')}_{d,r'-r,c'-c} = \sum_{d'} \sum_{\tilde{m}} \sum_{\tilde{n}} \partial^{(l+1)}_{d',r+\tilde{m},c+\tilde{m}} K^{(l,d')}_{d,\tilde{m},\tilde{n}} \\
&= \sum_{d'} (\mathrm{rot180}(K^{(l,d')}_{d}) * \partial^{(l+1)}_{d'})_{r,c} \quad ,
\end{aligned} \tag{2.43}$$

where rot180 is rotating the kernel by 180 degrees (which is the same as flipping in row and column direction). Similarly, the derivatives with respect to the weights are

$$\begin{aligned}
\frac{\partial L}{\partial K^{(l,d)}_{k,m,n}} &= \sum_r \sum_c \partial^{(l)}_{d,r,c} \frac{\partial F^{(l)}_{d,r,c}}{\partial K^{(l,d)}_{k,m,n}} = \sum_r \sum_c \partial^{(l)}_{d,r,c} \frac{\partial \sum_{k'} \sum_{m'} \sum_{n'} F^{(l-1)}_{k',r-m',c-n'} K^{(l,d)}_{k',m',n'}}{\partial K^{(l,d)}_{k,m,n}} \\
&= \sum_r \sum_c \partial^{(l)}_{d,r,c} F^{(l-1)}_{k,r-m,c-n} = \sum_{r'} \sum_{c'} \partial^{(l)}_{d,m+r',n+c'} F^{(l-1)}_{k,r',c'} \\
&= (\mathrm{rot180}(F^{(l-1)}_{k}) * \partial^{(l)}_{d})_{m,n} \quad .
\end{aligned} \tag{2.44}$$

This shows that during the backward pass a number of single-channel convolutions with flipped kernels have to be performed.

### Transposed Convolutions

In some cases, such as networks for semantic segmentation or autoencoders, an encoder-decoder architecture is used. That is, the spatial dimension of feature maps successively decreases down to a so-called *bottleneck* of small dimension, before it is increased again step by step. One option for upscaling a feature map is to use a bilinear zooming before the output is further processed. Another option is to use a *transposed convolution*, where each pixel of the input is multiplied by a kernel and the results are summed up like in the backward pass of the normal convolution layer (2.43):

$$F^{(l)}_{d,r,c} = \sum_{d'} (\mathrm{rot180} K^{(l,d')}_{d} * F^{(l-1)}_{d'})_{r,c} = \sum_{d'} \sum_{\{r',i:\ S_r r'+i=r\}} \sum_{\{c',j:\ S_c c'+j=c\}} F^{(l-1)}_{d',r',c'} K^{(l,d')}_{d,i,j} \quad . \tag{2.45}$$

Also see [34] for visualizations and examples of different variations of transposed convolutions (only for the case $d^{(l)} = D^{(l)} = 1$). Note that in comparison to the normal

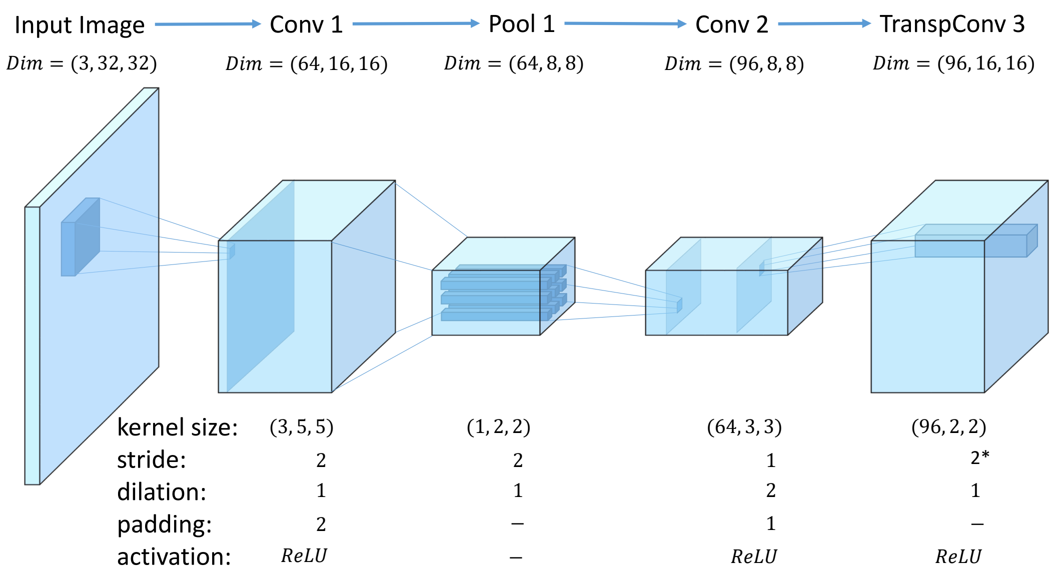


**Figure 2.8: Convolution types.** (*From left to right*) In the first layer, a convolution is applied to the input image. Each filter application results in a single pixel of the output feature map. A pooling operation is used to reduce the feature dimension and introduce robustness to small distortions. A dilated convolution leads to an increased receptive field. The feature map spatial resolution is increased by a transposed convolution. *Note that for transposed convolutions it is more intuitive to apply stride and padding to the output shape instead of the input.

convolution layer, the output depth is given by the kernel depth $D^{(l)}$ and the number of kernels $d^{(l)}$ is equal to the input depth $d^{(l-1)}$.

### Grouped Convolutions

In modern architectures, the convolutions are frequently separated into groups. That means, that the input depth dimensions $d^{(l-1)}$ are split into $g$ distinct groups and the filter kernels are split into kernels of depth $d^{(l-1)}/g$. The first $d^{(l)}/g$ filters only act on the first group of $d^{(l-1)}/g$ input-channels, the second group of filters act only on the second group of input channels, and so on.

In an extreme case, the number of groups can be set to the number of input channels. This is done in *depth-wise separable* convolutions, where instead of performing a convolution with kernel-size $k$, a grouped convolution with kernel size $k$ and $g = d^{(l-1)}$ is followed by a convolution with kernel size $1 \times 1$. The benefit of depth-wise separable convolutions is that they are highly efficient. For example, for a convolution with stride one, we have $whd^{(l-1)}(HW + d^{(l)})$ multiply-add operations instead of $whd^{(l-1)}HWd^{(l)}$. Since $d^{(l)}$ is often in the range of 128 to 2048, depth-wise separable convolutions can lead to remarkable speedups together with a reduction of storage costs of factor $d^{(l)}$.

Some examples and visualizations of most of the mentioned types of convolutions are shown in Fig. 2.8.

**Pooling Layer**

To handle large input dimensions, a common strategy is to distill the typically sparse information within the input image by a suitable dimensionality reduction. A method that has shown to work well is the introduction of *pooling* layers into the network. Pooling layers act on patches of the input channels. As for convolution layers, the spatial patch dimensions $H \times W$, the stride $S$ with which the pooling is applied, and the padding $P$ of the input can be chosen and have an influence on the output dimensions. Pooling layers usually are applied channel-wise, and therefore, the depth of the patches equals one. This leads to an output feature map depth that is equal to the input feature map depth $d^{(l)} = d^{(l-1)}$.

A max-pooling layer computes the output $F^{(l)}$ as

$$F^{(l)}_{d,r,c} = \max_{m=-\lfloor H/2 \rfloor}^{\lfloor H/2 \rfloor} \max_{n=-\lfloor W/2 \rfloor}^{\lfloor W/2 \rfloor} F^{(l-1)}_{d,r\cdot S+m,c\cdot S+n} \ , \tag{2.46}$$

and for an average-pooling layer we have

$$F^{(l)}_{d,r,c} = \frac{1}{HW} \sum_{m=-\lfloor H/2 \rfloor}^{\lfloor H/2 \rfloor} \sum_{n=-\lfloor W/2 \rfloor}^{\lfloor W/2 \rfloor} F^{(l-1)}_{d,r\cdot S+m,c\cdot S+n} \ . \tag{2.47}$$

According to (2.46) and (2.47), a pooling layer has no learnable weights. A pooling layer introduces local invariance because it is not important at which position inside the patch we pool from a specific input value occurs. Moreover, for a max-pooling another nonlinearity in addition to the activation is introduced to the network. Additionally, one can argue that max-pooling layers make the representation of the input more robust to noise since small activations are filtered out. However, one should keep in mind that this argument is based on the hypothesis that the important information is contained in large activations and that only those should be forwarded through the network. The same hypothesis is underlying the use of ReLU activation functions, which set all negative inputs to zero.

The derivatives with respect to the inputs are for non-overlapping max-pooling with $S = H = W$:

$$\partial^{(l)}_{d,r,c} = \begin{cases} \partial^{(l+1)}_{d,\lfloor r/H \rfloor,\lfloor c/W \rfloor} & \text{if } (r,c) = \arg\max_{m=-\lfloor H/2 \rfloor}^{\lfloor H/2 \rfloor} \arg\max_{n=-\lfloor W/2 \rfloor}^{\lfloor W/2 \rfloor} F^{(l)}_{d,r\cdot S+m,c\cdot S+n} \ , \\ 0, & \text{otherwise.} \end{cases}$$

Accordingly, if the pooling patches overlap, the gradient is propagated to the argmax position and the sum over all overlapping patches is taken in the backward pass.

For average pooling, the gradient of the loss with respect to the output, $\boldsymbol{\partial}^{(l+1)}$ is distributed with factor $1/HW$ to each pixel within the input gradient. Also here, if patches overlap, the gradient of each patch has to be summed up.

If $H$ and $W$ are set (dynamically) to the input dimensions $h^{(l-1)}$ and $w^{(l-1)}$, respectively, we refer to the layer as *global* pooling layer. Hence, the output size of a global pooling is always $(d^{(l-1)}, 1, 1)$, independent of the input spatial dimension.

Typically, within the first stages of a network $2 \times 2$ pooling layers with stride $S = 2$ are used and before the first fully connected layer of a classifier, a global pooling might be used for the following reason: If a network only contains a combination of convolution, activation, batch norm, or pooling layers up to this point, we are able to apply it on arbitrary input image sizes. The intermediate feature map sizes are changing, but the weight-shapes remain constant. However, for a fc-layer we need a constant input dimension because the dimension of the weight matrix depends on it. The use of a global pooling layer compresses the input feature map to a constant output size such that it fits to the following fc-layer. Note that networks that only consist of convolution, pooling, activation, or pooling layers, such that they can digest variable input sizes, are often referred to as *fully convolutional* neural networks or FCNs [126], for short.

### Dropout

*Dropout* [73] is an effective method to regularize large networks. By setting a hidden neuron's output to zero with a given dropout probability $\alpha$ during training, the model is forced to not only rely on the activation of this particular neuron. This can also be seen as a model averaging over all possible networks that have only $1 - \alpha$ times the number of neurons in the layer where dropout is used. Dropout is most often used in large fully connected layers at the end of CNNs, but it can also be applied to convolution layers [52].

In its initial form, dropout was only used during training and was switched off during inference. However, in some cases dropout is also used during inference to improve the model's output confidence: Instead of relying on the softmax output of a single forward pass, the model is evaluated several times with dropout and the softmax outputs are averaged [52, 90].

### Batch Normalization

In each iteration of the training, the input changes according to the images within the current batch. This leads to largely varying activations within the feature maps of all layers from iteration to iteration. With backpropagation, we calculate the gradients of the loss with respect to the weights of each layer. This gradient tells us the direction in which we should adapt the weights of each layer to reduce the loss for the current batch of images. However, thereby we assume that all other layers remain constant. In practice, this is not the case because we update all layers simultaneously. This leads to a changing distribution of the layer's input, denoted as *internal covariate shift* [82]. In theory, higher order optimization methods are a way to address this problem, but in practice they are not feasible due to the large number of parameters that lead to exploding computational and storage costs.

A popular method to reduce the internal covariate shift is *batch normalization* [82], where the inputs to each convolution layer are normalized over each mini-batch $\mathcal{B} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$:

$$\begin{aligned}
\boldsymbol{\mu}_{\mathcal{B}} &= \frac{1}{mhw}\sum_{i=1}^{m}\sum_{j=1}^{hw}\mathbf{x}_{ij}, \\
\boldsymbol{\sigma}_{\mathcal{B}}^2 &= \frac{1}{mhw}\sum_{i=1}^{m}\sum_{j=1}^{hw}(\mathbf{x}_{ij}-\boldsymbol{\mu}_{\mathcal{B}})^2, \\
\hat{\mathbf{x}}_{ij} &= \frac{\mathbf{x}_{ij}-\boldsymbol{\mu}_{\mathcal{B}}}{\sqrt{\boldsymbol{\sigma}_{\mathcal{B}}^2+\epsilon}}, \\
\mathbf{y}_i &= \gamma\hat{\mathbf{x}}_i+\beta \quad ,
\end{aligned} \tag{2.48}$$

where $\mathbf{x}_{ij}$ is a vectorized version along the spatial dimensions $h, w$ of $\mathbf{x}_i$, $\epsilon = 10^{-8}$ is a small constant for numerical stability and $\gamma$ and $\beta$ can be learned. Note that $\mathbf{x}_{ij}, \gamma, \beta, \boldsymbol{\mu}_{\mathcal{B}}$, and $\boldsymbol{\sigma}_{\mathcal{B}}$ are of dimension $d$, i.e., each depth-channel is handled separately. In practice, the mean $\boldsymbol{\mu}_{\mathcal{B}}$ and standard deviation $\boldsymbol{\sigma}_{\mathcal{B}}$ are updated as running averages during training and fixed during inference. Thus, during inference they can be combined with $\gamma$ and $\beta$, leading to a single linear transformation.

#### Example Activations

Fig. 2.9 gives an impression how the activations inside a typical CNN for classification of pill images look like. As has been empirically shown by Bau et al. [5], the first layers respond to low-level features such as edges or colors, while the later feature maps show higher-level features such as the pill's outline, or a defect. In this particular example, low-quality activation maps of the classes of interest are shown even though the model was not trained for this task.

## 2.6 Modern CNN Architectures

Today, CNNs play an important role whenever strong features are necessary to solve almost any computer vision task. Therefore, the underlying CNN-architecture to extract those features is often called *backbone* of the model. Since the backbone is usually only one module of many that together form the model, it is exchangeable based on the users preferences. Usually, there exists a runtime–accuracy trade off depending on the number of parameters and floating point operations (FLOPS) required to perform for one forward pass. Research on CNN architectures is still evolving quickly and every year new concepts are introduced where it is sometimes unclear if the proposed benefits are statistically significant. To give an overview, in this section we briefly describe some modern CNN architectures and design principles that were developed during the last decade without claim for completeness.

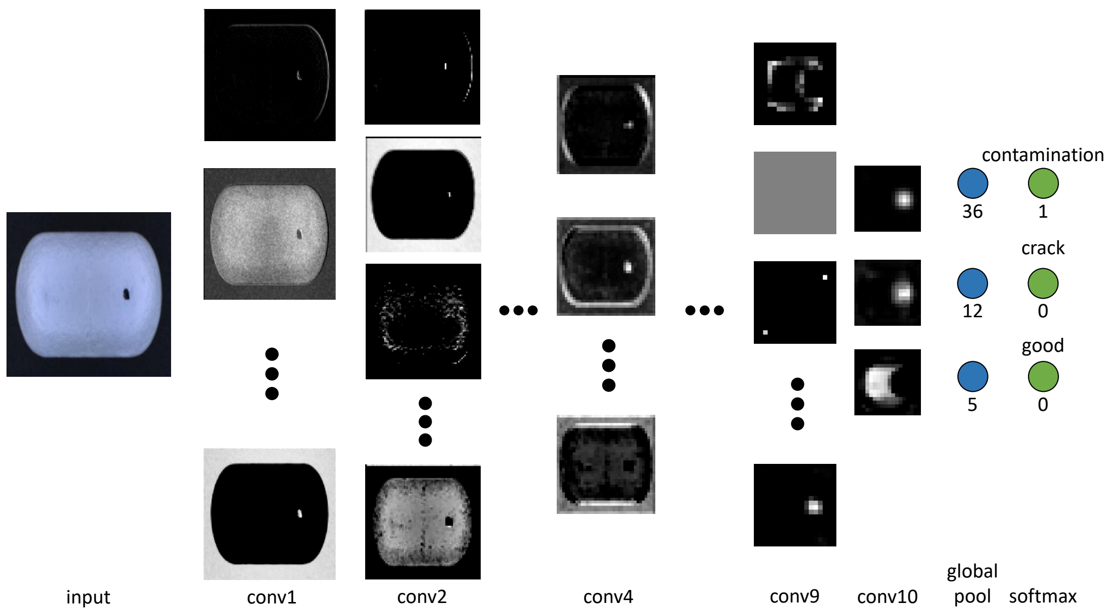


**Figure 2.9: CNN activations.** Different feature map channels of a SqueezeNet [81] applied to an image of a pill containing a contamination defect. The first layers respond to low-level features, such as color or edges. The later layers show more complex structures or even semantic concepts, such as the pill's outline or the contamination. Looking at layer *conv10*, the model seems not to be absolutely sure if the defect is a crack or a contamination. However, the exponential within the softmax clearly predicts class *contamination*.

### 2.6.1 AlexNet

In 2012, Krizhevsky et al. [95] won the ImageNet large-scale visual recognition challenge (ILSVRC) with a novel CNN architecture, commonly referred to with respect to the first author's name as *AlexNet*. Their method reduced the top-5 error for this classification problem with 1000 different categories from 26.2% to 15.3%. Although, some research on CNNs was going on, this breakthrough led to a dramatic increase of the popularity of CNNs in the computer vision field. Interestingly, this architecture with its seven layers is often related to the raise of deep learning's popularity. With hindsight, it is maybe not due to the depth of the CNN, but because it was the first architecture that was designed for the use with multiple GPUs, and additionally, because it was trained on a large scale dataset like ImageNet.

*CaffeNet*, a slight variation of the architecture as implemented in the popular *Caffe* framework [85], is depicted in Fig. 2.10. We abbreviate a kernel size of $m \times m$ with k$m$, and correspondingly a stride by s$m$. #$m$ denotes the number of filters in convolution layers and output neurons in fc layers, respectively. d$\alpha$ is the dropout probability in the fc layers if dropout is present.

In comparison to more recent architectures, the first layers have remarkably large kernel sizes, which are computationally expensive. Also, the three adjacent fc-layers with their high number of neurons make the model very large. The weight matrix of the dense *fc8*-layer alone has $4096 \times 4096 \approx 16\,000\,000$ parameters.

The model also introduced a *local response normalization* (LRN) layer that normal-

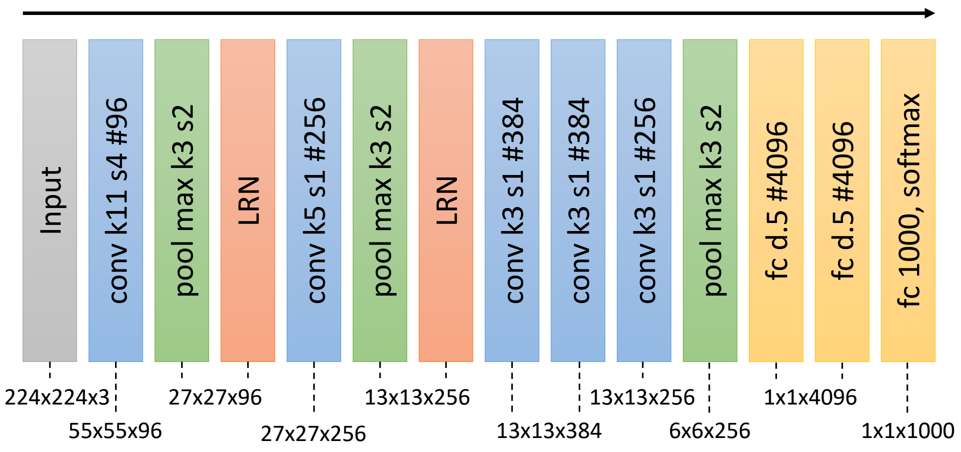


**Figure 2.10: CaffeNet architecture.** (*Top*) Sequence of layers from input to output with most important hyperparameters. All convolution and fc layers are followed by a ReLU activation. (*Bottom*) The corresponding output feature map dimensions of the respective layers.

ized the activation at position $(r,c)$ within feature map $d$, $a(d,r,c)$, with respect to the activations of neighboring feature maps:

$$\mathrm{LRN}(d,r,c) = a(d,r,c) / \left( k + \alpha \sum_{j=\max(0,d-n/2)}^{\min(d^{(i-1)}-1,d+n/2)} a(j,r,c)^2 \right)^{\beta}, \tag{2.49}$$

with $k=2, n=5, \alpha=10^{-4}$, and $\beta=0.75$ chosen manually. Krizhevsky et al. empirically found that LRN let to a significant reduction of the error. However, it has not been used in many models since then.

### 2.6.2 Inception — GoogLeNet

Szegedy et al. [183] proposed the *GoogLeNet* or *Inception* architecture, which was the winner of the ILSVRC 2014. The idea behind their model is to build a deeper network while conserving the computational cost. Moreover, with the introduced Inception modules (cf. Fig. 2.11), features responding to different receptive fields should be created. Therefore, activations of convolutions with different kernel sizes are concatenated along the depth axis and form the input of the following inception module or another layer that is attached.

Throughout the network, heavy usage of $1 \times 1$ convolutions is made, with the main goal to reduce the input depth, and therefore the kernel depth, of the following convolution. This idea was first proposed in the network in network (NIN) architecture of Lin et al. [117]. Another idea copied from the NIN model is to use a global pooling before the final fc layer, again to reduce the feature dimension, but also to make the model's input image size variable.

When GoogLeNet was proposed, people were often struggling to train deep networks

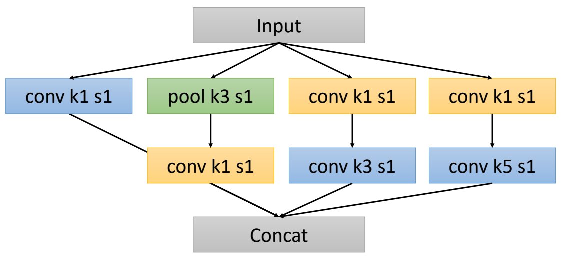


**Figure 2.11: Inception module.** The output of an Inception module consists of concatenated activations of convolution and pooling layers with different kernel sizes. Intermediate $1 \times 1$ convolutions are used to reduce the computational cost.

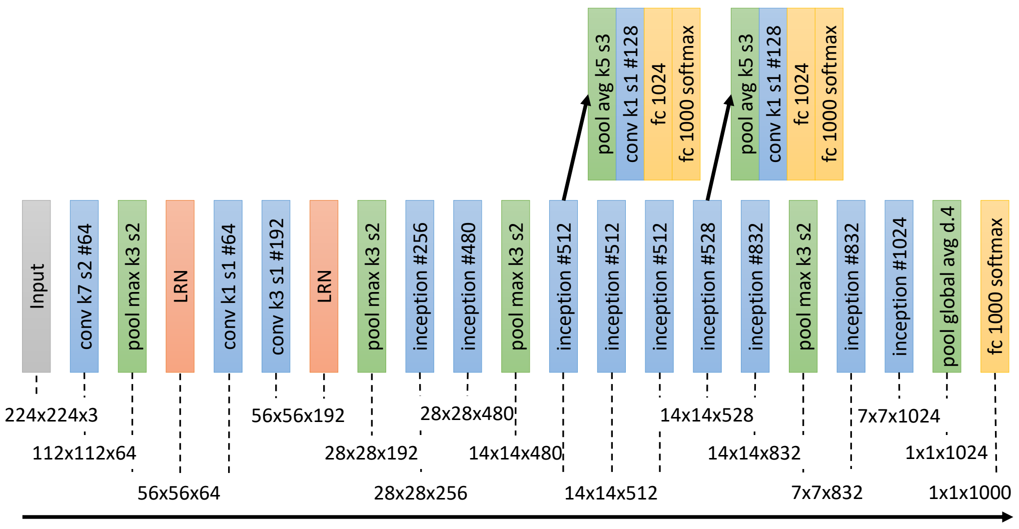


**Figure 2.12: GoogLeNet architecture.** (*Top*) Auxiliary loss branches are used during training to address the vanishing gradients problem. (*Middle*) Sequence of layers from input to output with most important hyperparameters. All convolution and fc layers are followed by a ReLU activation. (*Bottom*) The corresponding output feature map dimensions of the respective layers.

due to the *vanishing gradients* problem: when the gradient is back-propagated through the network, due to small (conv layers), negative (ReLU), or non-maximal (max-pooling) intermediate activations, the gradient becomes smaller and smaller until it eventually vanishes. To address this problem, two auxiliary intermediate loss branches are attached in the middle of the network. These loss branches are only used during training and their respective gradients are summed to the gradient that comes from the final loss layer. The *Inception v1* architecture is visualized in Fig. 2.12.

The Inception modules have been further refined in [184], where convolution layers with large kernel sizes, e.g., $5 \times 5$, have been replaced by two successive $3 \times 3$ convolution layers. Moreover, three different kinds of Inception modules have been introduced, where sometimes a $n \times n$ convolution has been replaced by an $n \times 1$ followed by a $1 \times n$ convolution to further improve the efficiency.

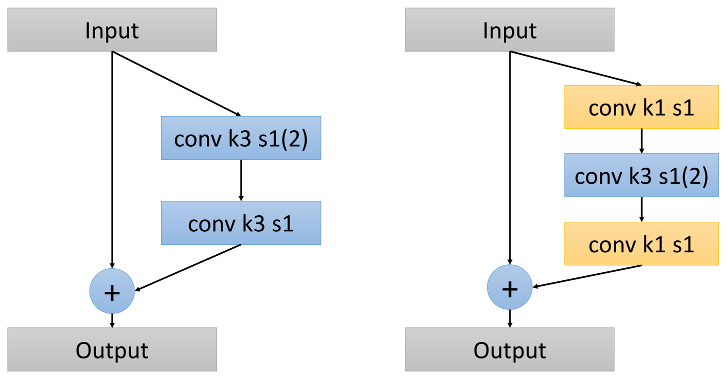


**Figure 2.13: ResNet building blocks.** (*Left*) A *basic* residual block as used in ResNets with a comparably low number of layers such as ResNet18/34. (*Right*) A *bottleneck* block that is used in very deep networks such as ResNet50/101/152.

Another important change that was incorporated from version two onward is the use of intermediate batch normalization layers before every ReLU activation throughout the whole network. This technique has proven to stabilize the training of deep networks and is used in all recent architectures. The use of batch normalization also allowed to train the version two network with only a single auxiliary loss.

### 2.6.3 ResNets

He et al. [68] noticed that deeper networks, seen as universal function approximators, should always be able to approximate the a function at least as well as a shallower counterpart. In the simplest case, the additional layers should be able to approximate the identity function. Therefore, in theory a deeper network should converge to a lower training set loss than a corresponding shallow network. However, naively constructed very deep networks converged to states with worse performance on the training and validation sets than their shallower counterparts. The reason could not only be the vanishing gradient problem, since this was already addressed and fixed by normalized inputs and batch normalization layers, as well as appropriate initialization of weights.

Led by these observations, the authors propose *deep residual networks (ResNets)*. The main idea of ResNet architectures is to introduce *skip-connections* throughout the network, that simplify the gradient flow. Instead of learning functions $\mathbf{y} = f(\mathbf{x})$, a ResNet learns functions $\mathbf{y} = f(\mathbf{x}) + \mathbf{x}$. In practice, this is modeled by the building blocks shown in Fig. 2.13. In very deep ResNets, the *bottleneck* block is used to avoid computationally expensive operations. For both block types, a batch normalization and ReLU layer is used before each convolution layer, i.e., normalization is done before the activation (*pre-activation* [69]). To reduce the spatial dimension of feature maps, convolutions with stride 2 are used instead of max-pooling layers. In this case, to avoid a representational bottleneck, the number of filters is doubled. To avoid neglecting 75% of input activations, in bottleneck building blocks the stride is applied to the $3 \times 3$ convolution. In the case of strided convolutions or a change in the number of filter kernels, the depth and spatial dimensions of input and output feature maps do not coincide. Therefore, a $1 \times 1$

convolution with stride 2 is used within the skip-connection path.

ResNets are still very popular today, especially as efficient feature extractors in object detection or semantic segmentation applications. To increase their classification accuracy, they have been further extended to ResNext [201] by the use of grouped convolutions. Moreover, squeeze-excitation (SE) networks [76] use rescaling branches to further improve the performance, but at the cost of additional computations.

# 3 Rotational Invariance for CNNs

Looking at the ImageNet [28, 168] pre-trained filters of AlexNet [95] (cf. Section 2.6.1) in Fig. 3.1, many filters seem to be redundant because they are similar to rotated versions of others. This is the brief motivation for the work that was done in this chapter, where we look at classification CNNs with rotated filters. In particular, we investigate the rotational invariance of CNNs. Moreover, we develop a model that is trained only on upright images, but can be successfully applied to rotated versions of the input. This leads to models that can be trained with fewer annotated training images. Hence, it is a step that lowers the initial hurdle of labeling many training images for a user that wants to apply a deep-learning-based classifier. The content of this chapter has been published in [44].

Despite breakthroughs in image classification due to the evolution of deep learning and, in particular, convolutional neural networks (CNNs), state-of-the-art models only possess a very limited amount of rotational invariance. Known workarounds include artificial rotations of the training data or ensemble approaches, where several models are evaluated. These approaches either increase the workload of the training or inflate the number of parameters. Further approaches add rotational invariance by globally pooling over rotationally equivariant features.

Instead, we propose to incorporate rotational invariance into the feature extraction part of the CNN directly. This allows to train on unrotated data and perform well on a rotated test set. We use rotational convolutions and introduce a rotational pooling layer that performs a pooling over the back-rotated output feature maps. We show that when training on the original, unrotated MNIST training dataset, but evaluating on rotations of the MNIST test dataset, the error rate can be reduced substantially from 58.20% to 12.20%. Similar results are shown for the CIFAR-10 and CIFAR-100 datasets.

## 3.1 Introduction

Deep learning and CNNs have lead to breakthroughs in various tasks such as image classification [68, 95, 183] and object detection [58, 158, 159]. Due to the nature of convolution- and max-pooling-layers, already the first CNNs built for handwritten digit recognition [104] incorporated invariance to translations and to minor distortions.

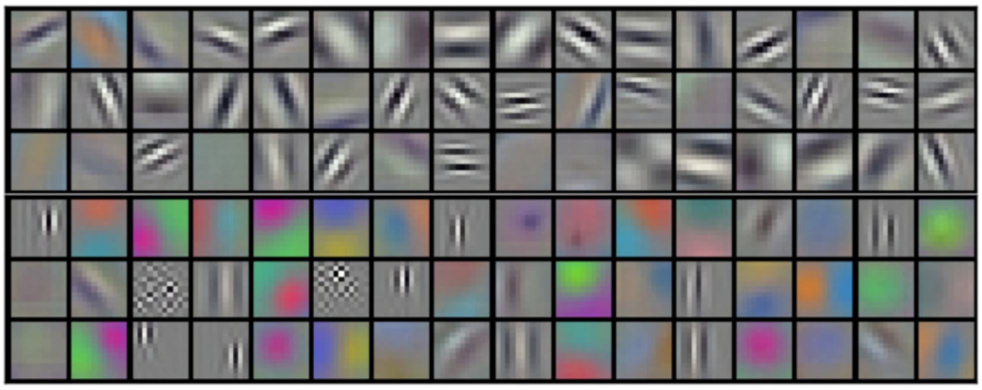
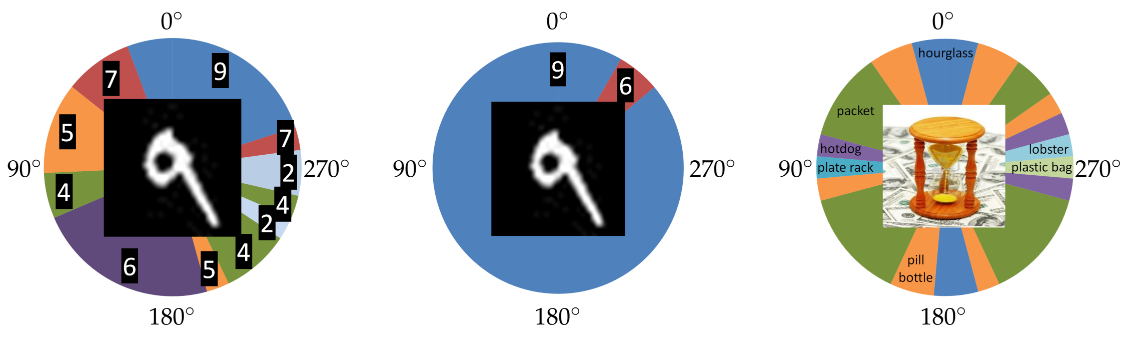


**Figure 3.1: Conv1-layer weights of AlexNet.** Weights of the first convolution in the AlexNet architecture [95] pre-trained on ImageNet [28]. Many filters look similar to rotated versions of others. Figure taken from [95] with the friendly permission of Krizhevsky et al.



**(a)** `reference` **(b)** `RP_RF_1` **(c)** hourglass from ImageNet

**Figure 3.2: Rotational invariance in image classification.** (a) predicted output class of a reference CNN-model (with an architecture similar to LeNet-5 [104]) depending on the rotation-angle of the input image. Only for 6 of the 36 rotated images (in steps of 10°) the predicted class is correct. (b) result when we replace the first convolution by a rotational convolution and pooling module with 12 filter rotations. Here, in 34 of 36 rotations the model predicts the ground truth class 9. For the remaining two rotations the most similar class 6 is predicted. (c) AlexNet [95] trained on ImageNet can predict the correct class *hourglass* only for an upright or nearly 180° rotated image.

However, even state-of-the-art networks are not invariant to rotations of the input. Since most datasets used in the computer vision community, such as MNIST[1], CIFAR-10[2], ImageNet [28], or Places [209], mainly include objects in an upright pose, this deficit is not emphasized by common benchmarks such as the ILSVRC [168]. Nevertheless, in recent years, CNN-based image classification and object detection algorithms have started to be used in biomedical and industrial environments, where objects can occur in an arbitrary orientation, e.g., in microscopic images or objects on a conveyor belt.

In typical CNNs, such as LeNet [104], AlexNet [95], GoogleNet [183], or ResNet [68] (cf. Section 2.6), several modules of convolution and max-pooling layers are stacked on

[1] Y. LeCun, C. Cortes, and C. J. C. Burges. *The MNIST database of handwritten digits*, 1998. `http://yann.lecun.com/exdb/mnist/`, accessed: 2017-05-19.

[2] A. Krizhevsky, V. Nair, and G. Hinton, *The CIFAR-10 dataset*, 2014. `https://www.cs.toronto.edu/~kriz/cifar.html`, accessed: 2017-05-19.

top of each other. They first extract low-, then mid-, and finally high-level features of the image before these are fed into a classifier that usually consists of one or several fully connected layers followed by a softmax. Convolutions with a small filter size and stride (compared to the input feature maps) followed by max-pooling subsequently reduce the feature dimension. At the same time, these feature extraction modules introduce invariance to translations of the objects inside the input image and invariance to smaller distortions and scale changes. Nevertheless, if the object is rotated by medium or large angles, the activations of the filters change in most cases. Therefore, the classifier is not able to predict the correct object category, as is visualized in Fig. 3.2c. For example, as depicted in Fig. 3.2a, a model that is trained on unrotated MNIST data achieves an error rate of 0.87% on unrotated test data. However, when each sample of the test set is rotated 36 times in steps of 10°, the same model only predicts the correct class 59.25% of the time. Hence, the question arises, is it possible to train a CNN that can classify rotated images, although it has only seen unrotated training examples?

In this chapter, we introduce a new module using rotational convolution and pooling layers in order to generate features that are rotationally invariant. In the forward pass, the $n_f$ filter masks of one layer are rotated $n_r$ times and produce in total $n_r \cdot n_f$ feature maps. In a second step, we back-rotate the feature maps by the negative angle by which their generating filter was rotated with previously. We follow this layer by a rotational pooling layer that performs pixel-wise max-pooling over the $n_r$ feature maps corresponding to the different rotations of a single filter. This leads to a module that is invariant to rotations of the input.

The main contributions of this chapter are:

1. With rotational convolution and pooling over the back-rotated output feature maps, we are the first to introduce a generalized convolution and pooling module for CNNs that is approximately rotationally invariant.

2. Therefore, in comparison to equivariant models [20, 110], it is not necessary to provide rotated training data to our model, even without the use of a global pooling layer.

3. In our experiments, we show that we can outperform the current state of the art, while at the same time being more parameter-efficient. In particular, the proposed new module is evaluated on unrotated and rotated versions of the MNIST, CIFAR-10, and CIFAR-100 datasets. When training on unrotated data and evaluating on rotated data, the error can be reduced from 58.20% to 12.20% for MNIST, from 67.55% to 55.88% for CIFAR-10, and from 83.65% to 77.06% for CIFAR-100.

## 3.2 Related Work

Two notions often used in the literature are those of *equivariant* and *invariant* models. For a rotation of the input, an equivariant model is expected to produce the same rotation of the output. In contrast, an invariant model is expected to produce the *exact* *same output* for a rotation of the input. For example, for robust image classification in industrial applications, a certain degree of invariance to deformations, translations scale changes and rotations is essential. To tackle demanding tasks such as texture classification or defect detection, early schemes were built on features that incorporated certain invariances, such as SIFT [127], local binary patterns [147] or methods that are based on bag-of-words, e.g., [146]. On the other hand, even though CNN-based methods offer a great performance boost for general image classification tasks, they lack an inherent invariance to rotations.

**Ensemble approaches and data augmentation.** Several approaches that introduce rotational equivariance are based on feeding the network with transformed data $T(x)$ for various, sometimes randomly selected, transformations $T$. Cireşan et al. [19] introduce multi-column deep neural networks, where a separate model is trained for each transformation of the input and the outputs of these models are averaged. Dieleman et al. [30] transform the images of galaxies to generate artificial viewpoints and concatenate the resulting feature maps as input to the classifier. The method of Laptev et al. [99] is quite similar, as the input to the model is transformed and fed through siamese networks that share feature weights. Instead of a concatenation, a max-pooling is performed over the feature maps and the result is fed into the classifier. All these methods have in common that the input to the CNN is transformed, and the gains in equivariance or invariance of those nets are essentially data-driven.

**Learning transformations.** One of the first approaches to learn transformations was the transforming autoencoder by Hinton et al. [72]. The building block of the transforming autoencoder is a *capsule*. Each capsule learns a single simple transformation that relates the input vector and a target output vector. For example, Hinton et al. [72] propose to learn the translation of an input image. Similarly, Jaderberg et al. [83] propose *spatial transformer* modules that learn transformations of the input data in a weakly supervised manner. The differentiable module can be plugged into any part of a CNN. It applies a transformation to the input image or feature map that is then passed through to the later layers. Dai et al. [26] go a step further and allow arbitrary deformations of every element of a filter in their *deformable convolution* and *deformable RoI pooling* modules. They present impressive results for image segmentation. In contrast to our work, all of the above mentioned modules need to see the possible transformations in the training stage. Hence, it is not possible to train them on unrotated data and expect good results on a rotated test set.

**Steerable approaches.** As opposed to methods that transform the input, the following approaches transform the filters of a network. Those approaches aim to produce *steerable* representations of the input that *transform in a predictable linear manner under transformations of the input* [21]. Recently, a number of closely related works have proposed convolution modules for rotational invariance with only subtle differences [20, 21, 199, 132, 213].

In [21], Cohen and Welling provide a theoretical framework for steerable CNNs and equivariant filter banks. They show that conventional CNNs are equivariant to translations but not to rotations and reflections. Furthermore, in [20], they propose group equivariant convolutional neural networks (GCNNs) as a special case of steerable CNNs. GCNNs consist of group convolutions containing rotations of filters and pooling operations over the rotations. Compared to our approach, the underlying transformations are restricted to be elements of a certain symmetry group. In particular, the proposed group convolutions are limited to $90°$ rotations and flips.

*Harmonic Networks* (H-Nets) [199] also use rotations of filters and extend GCNNs as they are not limited to $90°$ degree rotations. Although they achieve a very compact representation, they restrict the filters to be from the family of circular harmonics. Another more recent type of equivariant models are *Rotation Equivariant Vector Field Networks* (RotEqNet) by Marcos et al. [132]. In their orientation pooling, additionally to the maximal activation, also the corresponding orientation is propagated through the network. In their experiments on MNIST-rot they are able to outperform H-Nets as well as *Oriented Response Networks* (ORN) [213]. Nevertheless, the authors rely on test time data rotations to further improve their results. In contrast to our work, all of the above networks consist of equivariant instead of invariant convolution modules and require a global pooling in order to achieve invariance.

In comparison to the methods mentioned above, our goal is to train with non-transformed inputs $x$ exclusively, but evaluate the model with transformed data $T(x)$. Hence, we are indeed striving for a rotationally invariant module instead of an equivariant one. From the above mentioned approaches, a similar experiment was conducted only by Zhou et al. [213], where the authors train on MNIST and evaluate on MNIST-rot. In ORNs the feature maps and filters are extended by the number of base-filter rotations. This leads to heavy models both in terms of memory and computation and restricts the feasible size of filters to $3 \times 3$.

In our work we use filter rotations and pooling over orientations similar to GCNNs, but we do not restrict the transformations to be from a symmetry group acting on $\mathbb{Z}^2$. Furthermore, our module is rotationally invariant. The key ingredient is to back-rotate the feature maps that originated from a rotation of the filter before applying an orientation pooling. This is in contrast to the ORAlign operation of ORNs, where all feature maps corresponding to one base-filter are back-rotated by the orientation of the feature map with maximal activation. In their experiments Zhou et al. [213] apply the ORAlign or ORPooling operations only to the last feature-map before the classifier. Due to its spatial size of $1 \times 1$, the *back-rotation* in their ORAlign layer degenerates to a permutation of feature maps.

## 3.3 Rotational Convolution and Pooling

We propose to generalize the conventional convolution layer of CNNs in order to make them less sensitive to rotations of the input. The presented module consists of an extended convolution layer, named rotational convolution, that is followed by a rotational pooling

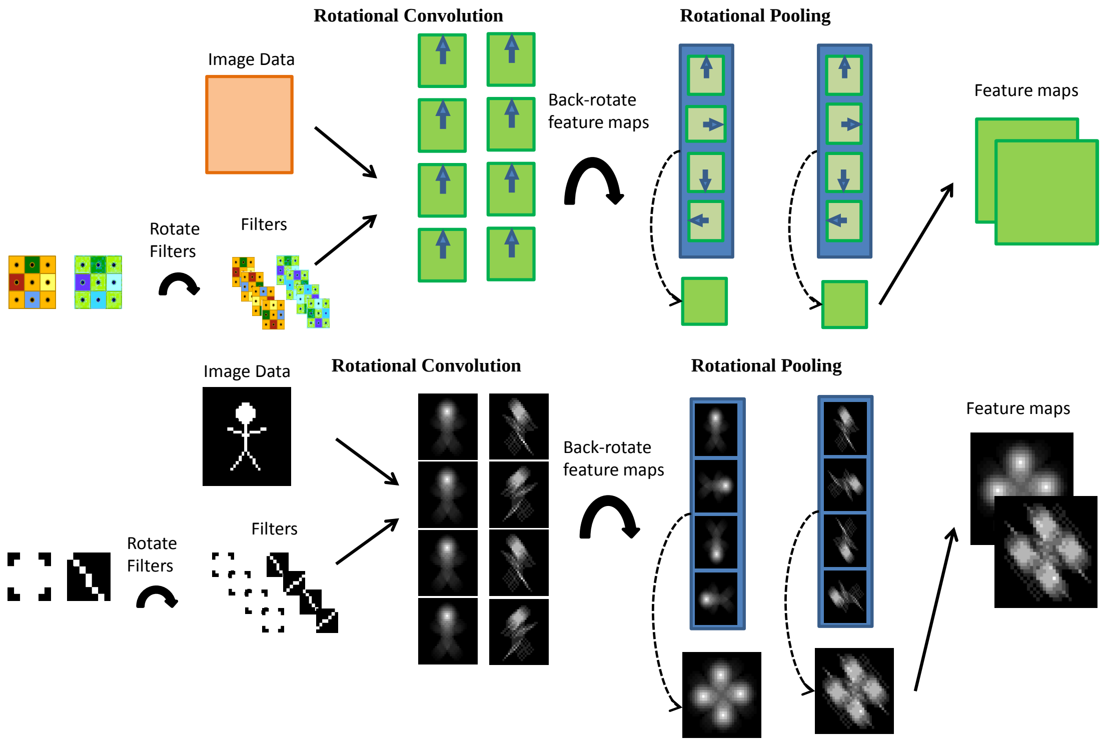


**Figure 3.3: Rotational convolution and pooling with feature map back-rotation.** (*Top*) Steps of rotational convolution and pooling module: First, the filters are rotated by $n_r$ different angles and applied as in a conventional convolution layer. To gain rotational invariance, each feature map is back-rotated with the negative angle the generating filter was rotated with previously. Thereafter, a feature map pooling over the rotations is performed. The pooling over the back-rotated feature maps is essential to gain rotational invariance. (*Bottom*) Toy example with two different filters acting on a single-channel image. Note that for 90°, 180°, and 270° rotated copies of the input, although the input to the rotational pooling are ordered differently, the final output feature maps are identical.

layer. The latter pools over the back-rotated feature maps of the preceding rotational convolution layer. The whole module, together with a toy example, is visualized in Fig. 3.3. In the following, we will think of images, feature maps, or filters as matrices or vectors on a discrete and finite pixel grid in $\mathbb{Z}^2$. For rotations, we will always assume a 2D rotation around the center of the filter, image, or feature map. Rotating a multi-channel instance means rotating each of the channels individually.

### 3.3.1 Rotational Convolution

Let us consider the typical scenario of a convolution with input feature maps $F \in \mathbb{R}^{d\times h\times w}$, where $h \cdot w$ is the number of pixels and $d$ is the number of channels. Furthermore, we denote the convolution kernels by $K_j \in \mathbb{R}^{D\times H\times W}, j = 1, \ldots, n_f$, where $H$ and $W$ are the spatial dimensions of the kernel and $D = d$. For convenience, we look at the case of square filter kernels where $H = W$.

Note that we slightly change the notation here in comparison to Chapter 2 to avoid misleading superscripts and indices: Suppose, we are talking about the $l$-th layer. We write $F$ for $F^{(l-1)}$ and we use $Z$ to refer to the layer's output features $F^{(l)}$. Moreover, we omit the spatial indices $d, r, c$ of kernels $K$ and we write $K_j$ for $K^{(l,j)}$. Instead, in this chapter, the superscript $K^{(i)}$ refers to the $i$-th rotation of a kernel. $Z_j^{(i)}$ is the feature map that is obtained by applying the $j$-th kernel with rotation $i$ to $F$, as we will introduce shortly.

Similar to the group convolutions of Cohen and Welling [20], we propose to use $n_r$ rotations of the filters to obtain $n_f \cdot n_r$ rotated filters

$$K_j^{(i)} = R_i(K_j), \quad i = 1, \ldots, n_r, \quad j = 1, \ldots, n_f, \tag{3.1}$$

with $R_i$ representing a rotation by the angle $(i-1) \cdot 2\pi / n_r$. One may also think of the rotation of each filter channel as a coordinate transformation that can be represented by a matrix multiplication $R^K \bar{K}$, where $\bar{K}$ is a vectorized version of a single channel of $K$, and $R^K$ has dimension $H^2 \times H^2$. Note that for angles that are not multiples of 90°, rotated coordinates do not lie directly on the pixel grid. Therefore, we use a bilinear interpolation of the four neighboring pixel values. For convenience, we incorporate this interpolation into $R_i$ when necessary, although $R_i$ is then no longer part of a symmetry group which holds for 90° rotations. In particular, $R_i$ is no longer invertible.

We use correlations in the forward, and convolutions in the backward pass. Hence, for each filter and rotation we can compute the feature map with a conventional planar correlation as

$$\hat{Z}_j^{(i)} = F \star K_j^{(i)}, \quad i = 1, \ldots, n_r, \quad j = 1, \ldots, n_f. \tag{3.2}$$

It is shown in [20] that for the symmetry group of 90° rotations, the stack of feature maps with respect to the $n_r$ rotations is group equivariant, but single feature maps are not. Consider the toy example in Fig. 3.3. Every filter is rotated in 4 different angles, 0°, 90°, 180°, and 270°. As a result, for each filter, we obtain 4 feature maps.

### Feature Map Back-Rotation

The next step for approximate rotation invariance is to back-rotate the *stack* of feature maps generated by the rotations of each filter. Each feature map is individually back-rotated by the negative angle its generating filter was rotated with in the previous rotational convolution step. This is essential to ensure that the subsequent max-pooling leads to rotational invariance. For example, if the input image is rotated by 90°, the resulting feature maps $\hat{Z}_j^{(i)}$ are not only rotated by 90°, but are also permuted. A max-pooling over the stack of feature maps in the following step results in the same output for both the rotated and the unrotated input image (see Fig. 3.3).

Hence, to obtain back-rotated feature maps, we rotate the feature maps $\hat{Z}_j^{(i)}$ backwards by

$$Z_j^{(i)} = R_i^{-}(\hat{Z}_j^{(i)}), \quad i = 1, \ldots, n_r, \quad j = 1, \ldots, n_f. \tag{3.3}$$

Note that when you think of $R_i^-$ as a matrix multiplication as explained above, the dimension of the matrix is different than that of $R_i$, because $R_i^-$ is applied to a feature map, while $R_i$ is applied to a filter. For convenience, in the following, we omit the filter index $j$ and look at the case $n_r = 4$ and hence rotations $R$ by 90° that form a symmetry group. In particular, we have $R^4 := R \circ R \circ R \circ R$ is the identity. We use the result of Equation 11 in Chapter 5 of [20]:

$$R(F) \star K = R(F \star R^-(K)), \tag{3.4}$$

which means that the correlation of a rotated image $R(F)$ with a filter $K$ is the same as the rotation by $R$ of the image $F$ correlated with the inverse-rotated filter $R^-(K)$. To show that the feature map back-rotation of (3.3) indeed leads to rotational invariance, we define the mapping of a rotational convolution that takes a feature map $F$ as input and outputs $n_r$ back-rotated feature maps as $C(F) = (Z^{(1)}, \dots, Z^{(n_r)})$. We have that

$$C(F) = \begin{cases} F \star K & = & F \star K \\ R^-(F \star R(K)) & = & R^3(F \star R(K)) \\ (R^2)^-(F \star R^2(K)) & = & R^2(F \star R^2(K)) \\ (R^3)^-(F \star R^3(K)) & = & R(F \star R^3(K)), \end{cases} \tag{3.5}$$

and using (3.4), we obtain

$$C(RF) = \begin{cases} R(F) \star K & = & R(F \star R^3(K)) \\ R^3(R(F) \star R(K)) & = & F \star K \\ R^2(R(F) \star R^2(K)) & = & R^3(F \star R(K)) \\ R(R(F) \star R^3(K)) & = & R^2(F \star R^2(K)). \end{cases} \tag{3.6}$$

This means that despite the fact that single output feature maps are not rotationally invariant, the stack of output feature maps is.

As mentioned above, for rotations with angles different from multiples of 90°, we approximate the inverse rotation by interpolation: $R_i^-$ in (3.3) is the rotation with the negative angle $-(i-1) \cdot 2\pi / n_r$ followed by the bilinear interpolation.

### Backpropagation for Rotational Convolution

Now that the forward pass of a rotational convolution has been explained, we still have to derive the gradient of the layer with respect to the shared weights and data. Following the conventional backpropagation algorithm, we are able to calculate the gradient of the loss function $L$ with respect to the rotational convolution output feature maps $\partial L / \partial Z_j^{(i)}$ and are interested in $\partial L / \partial F$ and $\partial L / \partial K_j$.

Since both the derivative as well as the rotation with $R_i$ are linear operators, from (3.3) we see that we can approximate

$$\frac{\partial L}{\partial \hat{Z}_j^{(i)}} \approx R_i \frac{\partial L}{\partial Z_j^{(i)}}, \tag{3.7}$$

where equality holds for rotations by multiples of 90°. This means that we just have to rotate the incoming gradients $\partial L / \partial Z_j^{(i)}$ forward. From there on, we can use the backward pass of a conventional convolution to obtain the data and weight gradients with respect to the rotated filters before we sum over the rotations.

### 3.3.2 Rotational Pooling

The last step of our module is rotational pooling. As the stack of back-rotated feature maps $(Z_j^{(1)}, \dots, Z_j^{(n_r)})$ is approximately rotation-invariant, we perform a pixel-wise max-pooling over the rotations to obtain the final output of our module:

$$Z_j(r, c) = \max_{i=1}^{n_r} Z_j^{(i)}(r, c). \tag{3.8}$$

Here, $(r, c)$ denotes the coordinates within the output feature maps. By the derivation above, it follows that the rotational convolution and pooling module is a rotationally invariant map for rotations with angles that are multiples of 90° and approximately rotationally invariant for the general case $n_r \neq 4$.

In the backward pass, the gradient computation is the same as for a conventional spatial max-pooling within a single feature map. The gradient is fully propagated to the pixels that contained the maximum values. For all other pixels the gradient is set to zero.

## 3.4 Experiments

We evaluate the proposed module on rotated versions of the MNIST, CIFAR-10, and CIFAR-100 datasets. Our main objective is to see whether we can train a model only on unrotated data and still obtain reasonable results when evaluating on rotated versions of the previously unseen images of the test set.

In general, rotational invariant features are less descriptive than rotational variant ones, since they ignore the rotational information. This holds for all learned features and handcrafted features simultaneously. Hence, in applications where the rotation of the objects is a key discriminative feature, the use of rotational invariant features is unfavorable. For example, usually it is reasonable to assume a car is standing on its 4 wheels. Nevertheless, in applications where the orientation of the object should have no influence to the classification result, rotational invariant features are a key asset. Typical examples include objects on a conveyor belt, and pick-and-place or overhead scenarios, such as augmented or virtual reality applications in computer aided surgery.

The architectures used in our experiments are shown in Table 3.1. We compare our model to a reference model without rotational convolution and poolings, to an extension of a GCNN [20] with $n_r = 8$ filter rotations, and to the ORN model of Zhou et al. [213] using ORPooling. Additional results for other variants of GCNNs, e.g., without rotational pooling, and our proposed models with feature map back-rotation can be found in Appendix A. We denote models with rotational pooling by `RP` and models with additional feature map back-rotation by `RP_RF`. Without the back-rotation of the feature

CNN architectures

| | | conv1 | | | conv2 | | | conv3 | | | conv4 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | RF | $n_f$ | $n_r$ | RP | $n_f$ | $n_r$ | RP | $n_f$ | $n_r$ | RP | $n_f$ | $n_r$ | RP | $n_p$ |
| `reference` | | 10 | 1 | | 20 | 1 | | 40 | 1 | | 80 | 1 | | 130 050 |
| `reference_large` | | 80 | 1 | | 160 | 1 | | 320 | 1 | | 80 | 1 | | 382 320 |
| `RP_1234` [20] | | 10 | 8 | ✓ | 20 | 8 | ✓ | 40 | 8 | ✓ | 80 | 8 | ✓ | 130 050 |
| `ORN` [213] | | 10 | 8 | | 20 | 8 | | 40 | 8 | | 80 | 8 | ✓ | 382 320 |
| `RP_RF_1` (ours) | ✓ | 10 | 8 | ✓ | 20 | 1 | ✓ | 40 | 1 | ✓ | 80 | 1 | ✓ | 130 050 |
| `RP_RF_1 MNIST` (ours) | ✓ | 40 | 8 | ✓ | 120 | 1 | | | - | | | - | | 1 810 000 |

**Table 3.1: CNN architectures.** Except for `RP_RF_1 MNIST/CIFAR`, all convolution kernels are of size $3 \times 3$ and followed by a max-pooling layer with stride 2. For each model, the convolution layers are followed by a fc-layer with 1024 units and Dropout with ratio 0.5. As classifier, another fc-layer with $n_c$ units and a softmax is used. A ReLU nonlinearity is applied to the activations of the convolution- and first fc-layers. The columns $n_f$ and $n_r$ show the number of learnable filters and the number of rotations, respectively. The checkmark in column RF indicates that feature maps are back-rotated, RP denotes rotational pooling, and $n_p$ the total number of learnable parameters when applied to MNIST. Please see Appendix A for the detailed description of RP_RF_1 MNIST/CIFAR and the computation of $n_p$.

maps and with $n_r = 4$ rotations, our module is similar to a group convolution. Hence, the model `RP_1234` is built of modules similar to those of model P4CNNRotationPooling in [20, Table 1], but uses eight instead of four rotations of the filters.

In order to compare our models with ORNs [213], we used a similar architecture with 4 convolution layers and filter sizes of $3 \times 3$. We exchanged one or several of the convolution modules by rotational convolution and pooling modules with feature map back-rotation (`RP_RF`). For example, in model `RP_RF_1`, the first convolution of the `reference` model is replaced by an `RP_RF` module.

Since ORNs use filters that act on orientation-specific feature maps and only use a rotational pooling after the last convolution layer, their models have approximately three times as many learnable parameters as our models.

### 3.4.1 MNIST

**Setup.** To evaluate with respect to a large set of rotations, we expand the original MNIST dataset by rotating each image from the training and test sets of MNIST 36 times in steps of $10°$ and denote the resulting dataset by MNIST-rot36. Please note that it is not the same dataset as *mnist-rot* proposed by [100], where each image is only rotated once with a random rotation. Hence, the set-sizes of *mnist-rot* and MNIST-rot36 training and test datasets are different.

**Rotational invariance.** The experiment of most interest is to train on the training set of MNIST while evaluating on the test set of MNIST-rot36. For each architecture, we trained with 5 different random initializations and training was performed for 30 epochs using stochastic gradient descent with a base learning rate of 0.01, batch size of 256,

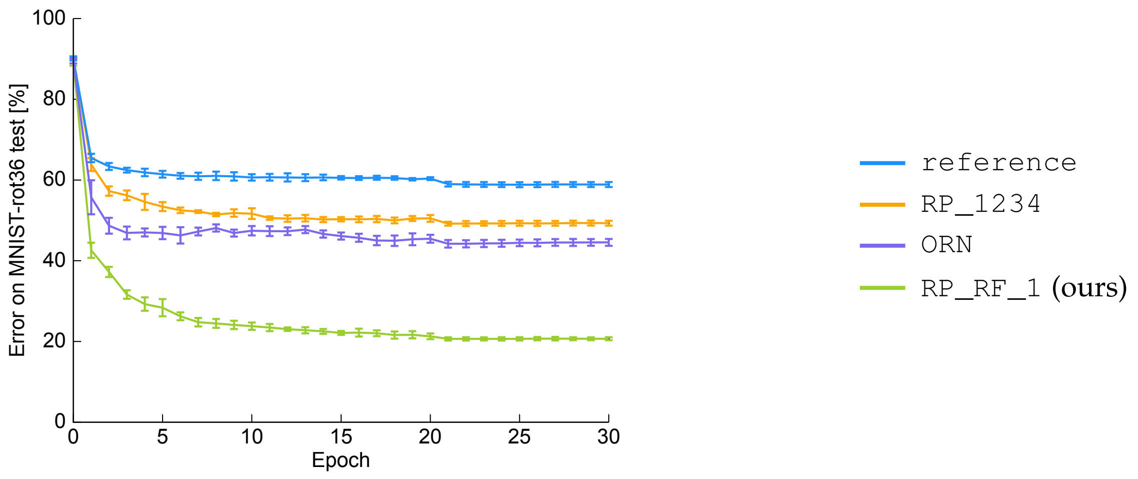


**Figure 3.4: MNIST training.** Evolution of the mean test error on MNIST-rot36 on MNIST for 5 randomly initialized runs. Only the model including feature map back-rotation and rotational pooling in the first convolution layer (`RP_RF_1`) is able to learn a rotationally invariant representation.

momentum of 0.9 and weight decay of 0.0005. The learning rate was multiplied by 0.1 after 20 epochs.

In Fig. 3.4, we visualize the evolution of the error on the MNIST-rot36 test dataset when training on the unrotated MNIST training dataset. `RP_1234`, `ORN`, and `RP_RF_1` models are compared to the `reference` model, as those are the ones with the smallest error. All models come close to their lowest error after just a few epochs. While for `reference`, `ORN`, and `RC_1234` the error on the rotated dataset is almost stagnating after around 10 epochs, the error for `RP_RF_1` is slightly decreasing until the end of the training. In general, `RP_RF_1` clearly outperforms all other models already after the second epoch.

The results for MNIST are summarized in Table 3.2. Our models, including feature map back-rotation and rotational pooling in the first rotational convolution (`RP_RF_1*`), are the only ones that are able to learn rotationally invariant representations in the sense that they can predict the class of a rotated image correctly, although only trained on upright images. By using the rotational convolution and pooling module with $n_r = 8$ rotations as the first feature extraction layer, we can reduce the error of the `reference` model from 58.20% to 19.85%. In comparison, as shown in the previous section, models without feature map back-rotation are only learning rotationally equivariant representations. Therefore, without training on rotated data, the models cannot learn the equivariance relation properly and show high error rates of 48.24% and 42.59% for the GCNN variant `RP_1234` and the `ORN`, respectively.

In order to see how far we can get, we also used another model `RP_RF_1 MNIST` that does not contain a global pooling, nor a down-sized last feature map of size $1 \times 1$ that is fed into the classifier. The model uses significantly more parameters, especially in the first fc-layer. A detailed description of the model can be found in Appendix A. In combination with using $n_r = 32$ number of rotations in the rotational convolution module, the error can be reduced to 12.20%.

| Training on MNIST, evaluating on MNIST-rot36 | | |
|---|---|---|
| model | min error test [%] | min loss training |
| `reference` | 58.20 | 0.019 |
| `RP_1234` [20] | 48.24 | 0.021 |
| `ORN (ORPooling)` [213] | 42.59 | 0.002 |
| `RP_RF_1` | **19.85** | 0.099 |
| `RP_RF_2` | 43.46 | 0.020 |
| `RP_RF_3` | 59.94 | 0.017 |
| `RP_RF_4` | 58.61 | 0.016 |
| `RP_RF_12` | 24.00 | 0.125 |
| `RP_RF_123` | 22.44 | 0.119 |
| `RP_RF_1234` | 23.71 | 0.124 |
| `RP_RF_1 MNIST` $n_r = 32$ | **12.20** | 0.026 |

| `RP_RF_1 MNIST` | |
|---|---|
| $n_r$ | min error [%] test [%] |
| 1 | 59.65 |
| 2 | 46.24 |
| 4 | 16.59 |
| 8 | 14.11 |
| 16 | 12.37 |
| 32 | **12.20** |

**Table 3.2: Results for MNIST.** (*Left*) When training on unrotated, but evaluating on rotated data, our models `RP_RF_1*` clearly outperform all other models that do not use feature map back-rotation or rotational pooling. The `RP_RF`-module is necessary in the first convolution. (*Right*) A higher number of rotations within the rotational convolution and pooling module lowers the minimal test error.

**Influence of $n_r$.** We further evaluate the influence of the parameter for the number of rotations $n_r$ for the best model `RP_RF_1 MNIST`. In order to reduce the training time, we only train for 20 epochs in these experiments, and drop the learning rate already after 17 epochs. As the standard deviation of the error with respect to different initializations is very low, we only use one repetition in this case.

The result is depicted in the right part of Table 3.2. Although, with an increasing number of rotations $n_r$, the error can be monotonically reduced, the number of internal calculations and the runtime is linearly increasing with $n_r$.

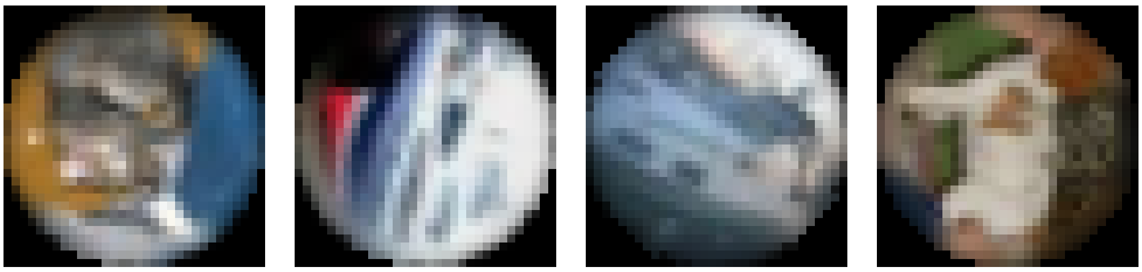
**Data augmentation.** For reference, we also show results for models trained on MNIST-rot36. As mentioned above, adding rotational invariance leads to a minor decrease in accuracy. The results are displayed in Table 3.3. `ORN` outperforms `RP_RF_1, reference` and `RP_1234` but also has a lot higher number of learnable parameters. `ORN` also outperforms `reference_large`, a usual CNN with the same number of parameters. All models trained on the augmented data outperform the models only trained on upright MNIST. This means that for cases where we can simulate the data transformations that might occur in the test set, this possibility should be exploited.

### 3.4.2 CIFAR-10 and CIFAR-100

**Setup.** In the same manner as done for MNIST, we rotate CIFAR-10 and CIFAR-100 to obtain the rotated versions CIFAR-10-rot36 and CIFAR-100-rot36, respectively. Several post-processing steps are done to remove rotation artifacts as visualized and explained in Fig. 3.5. The CNN architectures are similar to the architectures used on MNIST. On

| Train and evaluate on MNIST-rot36 | |
|---|---|
| model | min error test [%] |
| `reference` | 1.41 |
| `reference_large` | 1.34 |
| `RP_1234` [20] | 1.44 |
| `ORN` [213] | **0.76** |
| `RP_RF_1` (ours) | 3.51 |
| `RP_RF_1 MNIST` (ours) | 2.04 |

**Table 3.3: Results for MNIST-rot36.** When training on rotated and evaluating on rotated data, the `ORN` model outperforms the other models. In general, adding rotational invariance slightly decreases the accuracy when training on rotated data.



**(a)** `cat` rotated **70°** **(b)** `ship`, rotated **250°** **(c)** `ship`, rotated **320°** **(d)** `dog`, rotated **270°**

**Figure 3.5: CIFAR-10-rot36 example images.** The object of the corresponding class is always centered inside the image. In order to avoid boundary-artifacts when rotating, the images are cropped by a circle with radius of half the image size. Note that we apply the same preprocessing steps also to the unrotated training images. Additionally, at the boundary to the black background, the cropped images are smoothed by a Gaussian with a kernel size of five pixels.

CIFAR, we always use $n_r = 8$ rotations for the rotational convolution modules, because the trade-off between accuracy and additional computational effort was best for this case on MNIST. Since the standard deviation of the minimal error for different initializations was very low on MNIST, we only did one experiment per model on CIFAR.

**CIFAR-10.** As is shown in Table 3.4, using rotational convolutions and poolings clearly improves the results compared to the reference model. The error on the CIFAR-10-rot36 test set can be reduced from 67.55% to 55.88% with our `RP_RF_1` model using $n_r = 8$ rotations. Analogously to MNIST, on CIFAR-10, the GCNN variant `RP_1234` and the `ORN` model are showing higher error-rates of 62.55% and 59.31% respectively.

**CIFAR-100.** Also for CIFAR-100-rot36, we significantly improve the accuracy on the test set of the `reference` model from 16.35% to 22.94% by exchanging the first convolution module with the proposed RP_RF module. But, in comparison to MNIST and CIFAR-10, the relative improvement of `RP_RF_1` to `ORN` and `RP_1234` is only marginal (6% and 9%, respectively).

| Train on CIFAR, evaluate on CIFAR-rot36 | | |
|---|---|---|
| model | CIFAR-10 min error [%] | CIFAR-100 min error [%] |
| reference | 67.55 | 83.65 |
| RP_1234 [20] | 62.55 | 79.03 |
| ORN [213] | 59.31 | 78.36 |
| RP_RF_1 (ours) | **55.88** | **77.06** |

**Table 3.4: Results for CIFAR-10 and CIFAR-100.** All models are trained on unrotated, but evaluated on rotated data. For CIFAR-10, the model RP_RF_1 reduces the error rate by 11.7, 6.7 and 3.4 percentage points compared to the reference, RP_1234 and ORN model, respectively. For CIFAR-100, a similar behavior is observable, but the difference is less pronounced.

In general for CIFAR-10 and CIFAR-100, in line with MNIST, we obtain a substantial improvement when feature maps are back-rotated. Nevertheless, the benefit of using the rotational convolution and pooling module is less significant than for MNIST.

## 3.5 Conclusion

In this chapter, we presented the rotational convolution and pooling module, a generalization of the conventional convolution layer. By rotations of the convolution filters and back-rotation of the corresponding feature maps, in combination with a pooling over the different rotations, the module achieves approximate invariance to rotations of the input. For the symmetry group of $90^\circ$ rotations, exact invariance was derived.

Furthermore, we have shown that when we exchange the first, or the first few, convolution layers in a CNN by rotational convolution and pooling modules with feature back-rotation, it suffices to train the model on unrotated data to achieve good results on the rotated test set. The resulting models are parameter and memory efficient. Nevertheless, the error on the MNIST-rot36 test set could be reduced to 12.20%.

For the CIFAR-10-rot36 and CIFAR-100-rot36 datasets, the improvement was less pronounced. This leaves room for further improvement on this rather difficult task.

# 4 An Introduction to Object Detection

This part of the thesis is dedicated to a very helpful concept that is a fundamental part of many computer vision algorithms: object detection. It can be defined as the simultaneous localization and classification of objects within an image. To date, algorithms are restricted to search for the presence of $n_c$ specific predefined categories of interest. In many applications, $n_c = 1$, i.e., only a single object is of interest. However, in comparison to classification, due to the localization it is possible to distinguish every instance of a category within the image — which makes it possible to count them.

As shown in Fig. 4.1, there are various ways to address the problem of detecting objects. Many of the recent algorithms localize the objects by drawing an axis-aligned bounding box around them. As these box detection methods are often the basis for more advanced algorithms, we give an extensive introduction to them in Section 4.3. However, there are also plenty of other ways to formalize the localization, such as using keypoints or oriented bounding boxes (cf. Chapter 8). Therefore, it is often unclear where exactly the border of object detection to other computer vision tasks is. In this thesis, we look at methods that take 2D RGB or grayscale images as input. In comparison to matching algorithms, e.g., shape-based matching [178, 179], the presented detection algorithms are not restricted to rigid objects that are transformed by rigid transformations. Objects can be deformable and the intra-class variations can be high.

Instance-aware semantic segmentation methods, where a pixel-precise mask is predicted for each object, are introduced in Section 4.4. The major difference of instance segmentation to semantic segmentation is that a region is predicted for each instance of the object. This allows to distinguish touching or overlapping objects and makes it possible to count them like in the box detection approach. However, the pixel-precise mask is often a much richer and more precise description of the objects location. Semantic segmentation methods predict a category for each pixel within an image individually. The context and the shape of objects is only learned indirectly by sharing features, which have a large receptive field, in turn.

Keypoint detection, and in particular face detection or human pose detection, are beyond the scope of this thesis. Although, the underlying algorithms are to some extent similar.

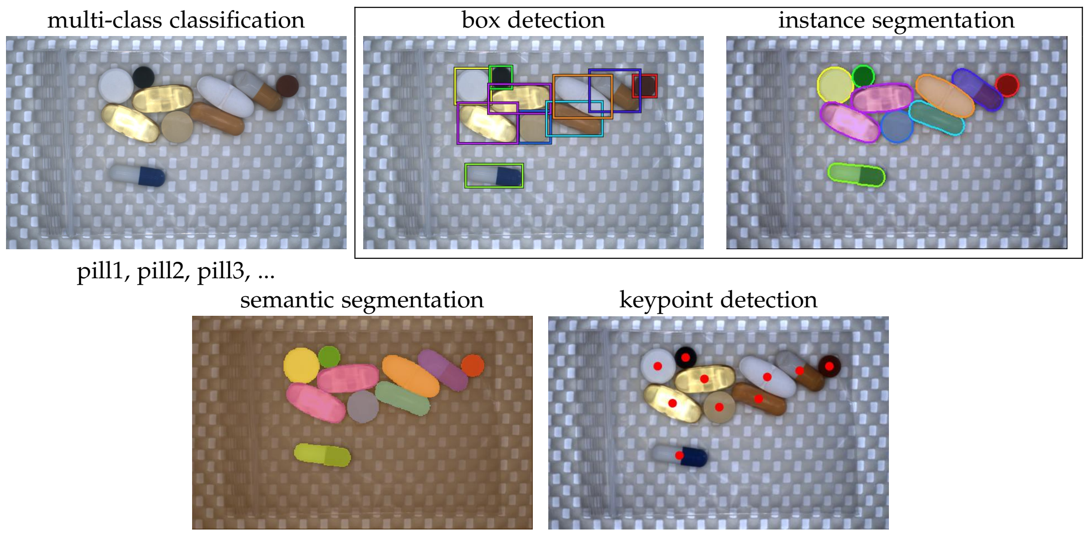


**Figure 4.1: Different types of detection.** For multi-class classification, each image is tagged with the categories that are contained in the image. For box detection, a bounding box is drawn around each object and the boxes are classified into one of $n_c$ categories. In instance segmentation, the bounding box is replaced by a pixel-precise mask for each object. In comparison, semantic segmentation classifies each pixel into one of $n_c + 1$ classes (class 0 is background). This allows to generate one region for each class, but touching or overlapping objects cannot be distinguished (e.g., the pink region contains two pills). Keypoint detection predicts one or several keypoints for each object (here, the center-point of each pill is visualized). In this work our focus is on box detection and instance segmentation methods.

## 4.1 Detection Datasets and Challenges

To give the reader a sense what the object detection challenge typically looks like, we start this part of the thesis with a short history of detection datasets and challenges in the past 20 years. As we just want to give an impression how the field has changed during the past years, we do not strive for completeness. Of course, with continuously improving algorithms, the datasets and corresponding challenges that drive the progress in object detection become more and more difficult. In Section 4.1.2, we introduce two industrial datasets that are used for experiments within this thesis.

### 4.1.1 Benchmark Datasets

In earlier works, $n_c$ is small and also the number of objects $n_o$ that is present within an image is small or only one. For example, in the *Caltech*-datasets [43, 196], images only contain one or a few ($n_o \ll 10$) instances of the object categories *leave, cars (rear), cars (side), human face, motorcycles*, or *airplanes*, but no mixtures of categories are present. To test the robustness of algorithms another dataset of *background* images was used. The TUD cow dataset [109, 131] or the UIUC car dataset [2] are in similar style, but are more difficult as the cows are articulated and the cars are appearing in different scales and with occlusions. The methods that tried to solve these datasets were usually detectors for a single category of interest. For example, a car detector was applied to the *car* dataset and to the *background* dataset, but not to the *motorcycles* dataset, where a motorcycle detector

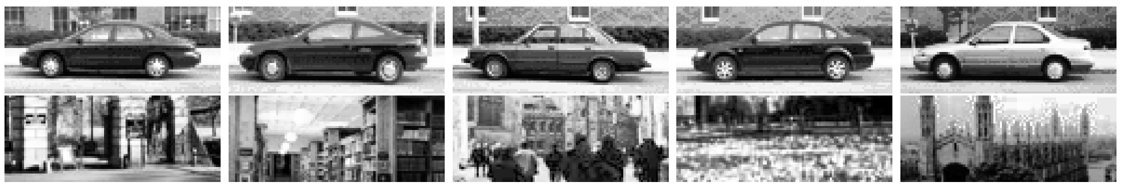


**Figure 4.2: Example images from *UIUC cars*.** (*Top*) *Car* images, with the object centered and without scale changes. (*Bottom*) *Background* images where no car is present to test the robustness of the method. Images are in grayscale and have a low resolution of $100 \times 40px$.

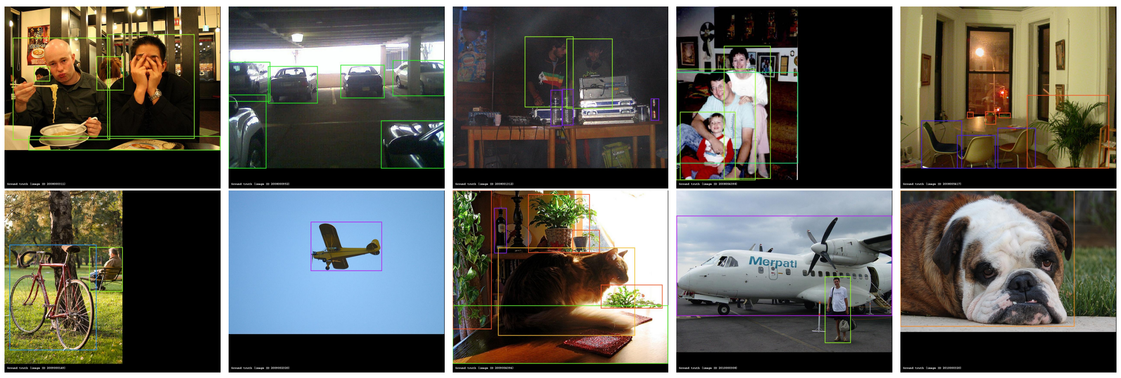


**Figure 4.3: Example images from *VOC*.** In comparison to previous detection datasets, objects appear with severe occlusions, in different scales and high intra-class variations. The number of objects per image is varying a lot from single objects covering almost the whole image to multiple tiny objects covering only a small portion of the image. Images also have very different aspect ratios and resolutions. Ground truth annotations are indicated by colored boxes.

was applied instead. Therefore, the field was also termed object class recognition at that time [43]. Some example images of UIUC cars are shown in Fig. 4.2.

Prominent datasets that paved the way for the evaluation of multi-class object detectors are the different versions of the *PASCAL Visual Object Classes Challenge (VOC)*, where the more typical use case of $1 < n_c < 100$ is adressed. In 2005 *VOC* [39] started with a collection of existing datasets for four categories: *motorbike*, *bicycle*, *people*, and *cars*. Eight different tasks were formulated, four for classification, i.e., "does this image contain an object of type $x$", and the other four for object detection by an axis-aligned bounding box. The four tasks differed in the training data, and the test set. The participants could decide if they only use the provided training and validation data or if they use additional training data from other sources. Moreover, one test set was more challenging with objects appearing in different scales and with more objects per image, on average. To lower the hurdle for contestants to participate, they were allowed to enter the challenge only for a subset of the categories.

The *VOC* challenges[1] were refined over the following years by adding more categories and collecting images with more variations from flickr[2] [40, 41]. From 2007, the dataset contained objects of 20 categories and 4 supercategories: *person* (*person*), *animal* (*bird, cat, cow, dog, horse, sheep*), *vehicle* (*aeroplane, bicycle, boat, bus, car, motorbike, train*), and *indoor*

[1]The PASCAL Visual Object Classes Homepage. `http://host.robots.ox.ac.uk/pascal/VOC/`, accessed 2021-02-07

[2]https://www.flickr.com/

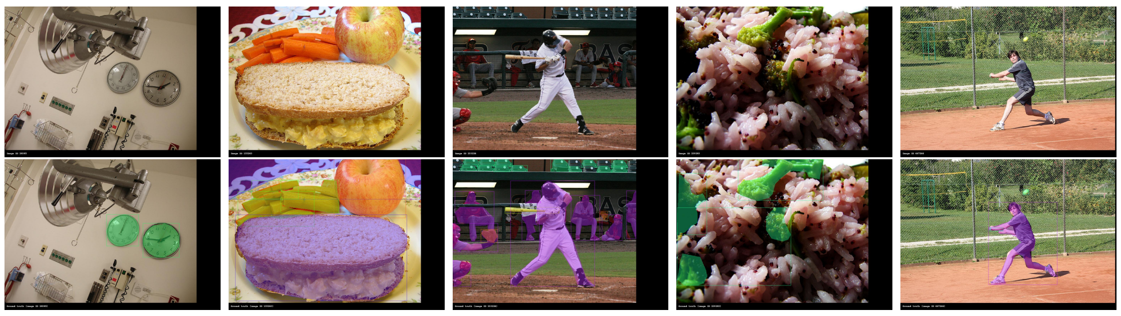


**Figure 4.4: Example images from *COCO*.** Images (*top*) and corresponding ground truth instance mask and box annotations indicated by colored overlays (*bottom*). In comparison to *VOC*, *COCO* has more images, more categories and more objects per image. The size of objects is varying a lot and in many cases, objects can just be determined correctly given the context (e.g., the tennis racket in the hand of the tennis-player in the right image). Unfortunately the dataset is not labeled consistently and masks are not always very accurate.

(*bottle, chair, dining table, potted plant, sofa, tv/monitor*). From 2007 to 2012, the *train* and *val* splits have been increased from 4340 images with 10 363 box annotations to 11 530 images with 27 450 box annotations and 6929 pixel-precise mask annotations. Moreover, additional annotations, such as a truncation or occlusion flag, or action classification annotations have been added from year to year. Over time, segmentation and action classification have been introduced as new tasks.

*VOC* also contributed by establishing the Average Precision (*AP*) evaluation measure and the definition of true positives considering the intersection over union (IoU) (cf. Section 4.2). Because the definition of the ground truth box was considered to be "somewhat subjective, [...] for example for a highly non-convex object, e.g., a person with arms and legs spread" [40, p. 314], the IoU threshold was set relatively low to 0.5.

In 2009, Deng et al. [28] published the large scale *ImageNet* dataset and the *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* [168] was held in conjunction with the *VOC* challenges. The focus of *ILSVRC* was on image classification into one of 1000 different classes. With 1.2 million training images, 50K validation images and 100K test images, a new order of magnitude in dataset size was reached. Moreover, in 2011, a subset was labeled with bounding box annotations for over 350 000 instances and a corresponding detection challenge was held as a taster competition [167]. Although in most of the *ImageNet* images the object of interest is centered, Russakovsky et al. [167] argue that a subset of the images have the same characteristics as the *VOC* images and that a valuable contribution is given by the high number of classes to distinguish. In particular, the dataset is the first to combine object detection with fine-grained recognition in this scale. However, the *ImageNet* dataset is primarily known as a benchmark dataset for classification and has not gained much popularity in the context of object detection.

In 2014, the next milestone in the development of object detection datasets was reached with the introduction of the *Microsoft Common Objects in Context* (COCO) dataset [118]. As the name already indicates, the hypothesis of the authors is that some — especially small — objects like smartphones, can only be detected correctly if they are seen in the context of other objects, e.g., persons holding them in their hands or close to their ears. Moreover, objects should occur in non-iconic views, i.e., not centered in the image, but "in the background, partially occluded, or amid clutter - reflecting the composition of actual everyday scenes"[118, p. 741]. Therefore, this challenging dataset of natural images was collected with 2.5 million instances in 328 000 images from 91 categories.[3] In addition to its large scale, another contribution that makes *COCO* the preferred benchmark dataset to date, in particular for instance segmentation (c.f. Section 4.4), is that every object is labeled with a pixel-precise segmentation mask. To obtain this very large amount of annotated data, the authors propose a novel annotation pipeline with heavy usage of *Amazon Mechanical Turk*[4] that allows to spread the labeling effort to the crowd. Some example images are shown in Fig. 4.4.

Recently, Gupta et al. [62] relabeled the images of *COCO* with a large-scale vocabulary of classes to obtain the *LVIS* dataset. In order to handle the more than 1000 categories and allow a more precise and comprehensive labeling of the instance masks, the authors propose a *federated* design of the dataset. This means that for each category $c$, there exist two subsets of the whole dataset $\mathcal{D}$, $P_c \subset \mathcal{D}$ where instances of $c$ are exhaustively annotated, and $N_c \subset \mathcal{D}$, where it is clear that $N_c$ is not present. During evaluation, each category $c$ is only evaluated on $P_c \cup N_c$ and $|P_c \cup N_c| \ll |\mathcal{D}|$. Since *LVIS* reuses the images of *COCO*, some categories occur much more often than others. In particular, for a large fraction of *rare* categories the number of instances within the training set is below 50. Therefore, *LVIS* proposes a new challenge of large-scale few-shot detection.

*ADE20K* [211] is another natural images dataset with fewer images (25K), but much more categories (2693) than *COCO*. It is worth mentioning that the whole dataset has been labeled by a single annotator. In comparison to *COCO*, in *ADE20K* also *stuff* classes like *sky* or *wall*, where instances cannot be distinguished, are annotated. Moreover, many objects are labeled on a part basis. Because all annotations are given by a segmentation mask and the class label, *ADE20K* is mainly used for instance segmentation and scene understanding, but of course object detection can also be trained and evaluated on it.

But not all modern and challenging detection datasets have a large number of classes. In many applications, such as counting of penguins [4], one is only interested in the localization of one particular class $n_c = 1$. Here, the difficulty is to handle the high intra-class variations, to deal with severe occlusions and touching objects, and to be robust to clutter in the background.

### 4.1.2 Industrial Detection Datasets

Today, industrial 2D object detection plays only a minor role in research. Industrial images often give the impression that the task is rather easy as variations between the images are often small. The objects, often parts of a product within the production process, are lying on a conveyor belt or another background that can typically be kept constant between different samples. Usually, the scenes are captured indoors and thus the lighting is controlled. But the most important difference to everyday photography

[3]For the official *COCO* detection challenge, the number of categories had been reduced to 80.
[4]Amazon Mechanical Turk, Inc., `https://www.mturk.com/`, accessed 2020-04-27.

might be the following: the intra-class variations are negligible and mostly occur due to reflections, perspective changes or deformations.

However, due to the high requirements for product quality, a method that *"solves"* a research dataset such as *VOC* and has a reasonable runtime only starts to get interesting for industrial purposes. In industry, constraints on the used methods are often measured in parts per million (ppm), which means that a failure can only occur in a few of a million applications of the algorithm.

Challenges of industrial detection datasets are often complementary to those of *VOC*, *COCO*, or *ADE20K*. Methods need to deal with occluding and touching objects, and very small inter-class variations, such as two types of screws that can only be distinguished by their slightly different color. Moreover, the requirements for the classification and localization accuracy are typically very high.

Another very important aspect is that methods should have a user centered design (UCD), which involves having no or only a few parameters that need to be intuitive to tune, and a low effort to apply the method on a new dataset. In particular, users want to avoid the tedious and expensive task of labeling a lot of data, but prefer algorithms that can be trained with only a few instances per class, termed *few-shot learning*.

In the following, we present two novel industrial datasets that show some, but not all of the above mentioned characteristics. In particular, we will see that the datasets are relatively small and do not have large variations between different images. Moreover, the classes are balanced in the sense that the frequency of their occurrence is uniformly distributed.

The datasets have been collected at MVTec as example datasets for the deep-learning-based object detection feature released with HALCON 18.11 [141] (*Pill Bags*), and for the deep-learning-based oriented object detection feature released with HALCON 19.05 [142] (*Screws*). The development of these features and the design of the datasets was part of the work that went into writing this thesis.

**Pill Bags.** The *Pill Bags* dataset contains 398 images of a textured background on which a plastic bag with up to 10 different classes of pills are placed. All images are 8-bit three-channel RGB color images. Examples are shown in Fig. 4.5. The categories are either round or of approximately elliptic shape. They mainly differ in their texture, color, or shape. Objects are frequently touching each other, but only rarely overlap. Annotations for *Pill Bags* exist as pixel-precise masks for each pill, from which we can infer the axis-aligned bounding boxes (AABBs) or oriented bounding boxes (OBBs) automatically.

Per-class examples and detailed statistics about the number of objects within the *Pill Bags* dataset are given in Table 4.1. Each class is present in at least 74% of the images and the relative number of instances for each category ranges from 8% to 13% in each split, respectively. This also reflects the application that is addressed by the dataset: An inspection system that controls whether each pill is present exactly once in each plastic bag.

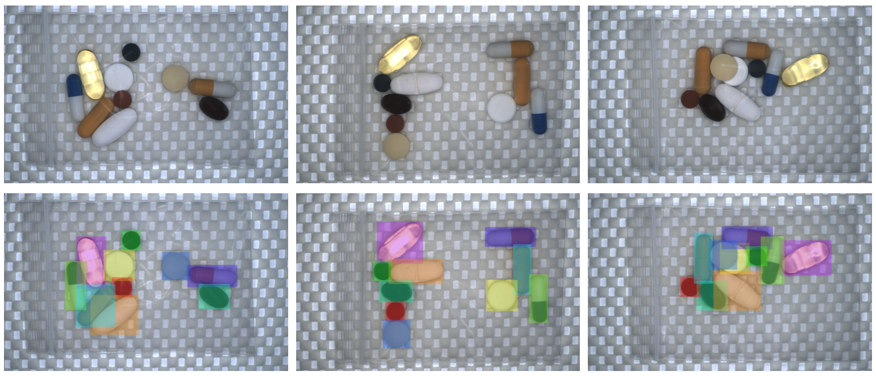


**Figure 4.5: *Pill Bags* example images**. (*Top*) Objects are lying on a homogeneous textured background and are inside a plastic bag. (*Bottom*) Ground truth annotations in AABB format. Objects are frequently touching each other and sometimes overlap.

| class | *Magentabletten* | *Omega-3* | *KMW* | *Ginseng* | *Ginko* | all |
|---|---|---|---|---|---|---|
| | | | | | | |
| Number of objects per class and split (abs/rel[%]) | | | | | | |
| *train* | 278 / 10 | 280 / 10 | 295 / 10 | 303 / 11 | 274 / 10 | 2826 |
| *val* | 65 / 10 | 62 / 10 | 52 / 8 | 67 / 11 | 61 / 10 | 622 |
| *test* | 66 / 11 | 60 / 10 | 62 / 10 | 65 / 11 | 54 / 9 | 600 |
| *total* | 409 / 10 | 402 / 10 | 409 / 10 | 435 / 11 | 389 / 10 | 4048 |
| Number of images, in which each class is contained, per split (abs/rel[%]) | | | | | | |
| *train* | 238 / 86 | 229 / 82 | 235 / 85 | 251 / 90 | 234 / 84 | 278 |
| *val* | 56 / 92 | 50 / 82 | 48 / 79 | 56 / 92 | 52 / 85 | 61 |
| *test* | 50 / 85 | 49 / 83 | 53 / 90 | 53 / 90 | 48 / 81 | 59 |
| *total* | 344 / 86 | 328 / 82 | 336 / 84 | 360 / 90 | 334 / 84 | 398 |

| class | *Cognivia* | *Glukosamin* | *Eisentabletten* | *Vitamin-B* | *Capsularum I* | all |
|---|---|---|---|---|---|---|
| | | | | | | |
| Number of objects per class and split (abs/rel[%]) | | | | | | |
| *train* | 312 / 11 | 276 / 10 | 275 / 10 | 230 / 8 | 303 / 11 | 2826 |
| *val* | 60 / 10 | 62 / 10 | 79 / 13 | 49 / 8 | 65 / 10 | 622 |
| *test* | 68 / 11 | 67 / 11 | 60 / 10 | 48 / 8 | 50 / 8 | 600 |
| *total* | 440 / 11 | 405 / 10 | 414 / 10 | 327 / 8 | 418 / 10 | 4048 |
| Number of images, in which each class is contained, per split (abs/rel[%]) | | | | | | |
| *train* | 246 / 88 | 226 / 81 | 225 / 81 | 207 / 74 | 239 / 86 | 278 |
| *val* | 51 / 84 | 50 / 82 | 58 / 95 | 44 / 72 | 51 / 84 | 61 |
| *test* | 52 / 88 | 54 / 92 | 46 / 78 | 44 / 75 | 47 / 80 | 59 |
| *total* | 349 / 88 | 330 / 83 | 329 / 83 | 295 / 74 | 337 / 85 | 398 |

**Table 4.1: *Pill Bags* statistics.** For each class, an example object and the absolute and relative number of instances is shown, in total and per split, respectively. Moreover, the number of images in which at least one object of the class is shown, also for each class and split.

| class | *type_01* | *type_02* | *type_03* | *type_04* | *type_05* | *type_06* | *type_07* | all |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| Number of objects per class and split (abs/rel[%]) | | | | | | | | |
| *train* | 219 / 7 | 219 / 7 | 221 / 7 | 219 / 7 | 270 / 9 | 270 / 9 | 252 / 8 | 3119 |
| *val* | 47 / 7 | 47 / 7 | 47 / 7 | 46 / 7 | 60 / 9 | 59 / 9 | 58 / 9 | 647 |
| *test* | 47 / 7 | 48 / 7 | 49 / 7 | 48 / 7 | 61 / 9 | 60 / 9 | 55 / 8 | 660 |
| *total* | 313 / 7 | 314 / | 317 / 7 | 313 / 7 | 391 / 9 | 389 / 9 | 365 / 8 | 4426 |
| Number of images, in which each class is contained, per split (abs/rel[%]) | | | | | | | | |
| *train* | 158 / 59 | 158 / 59 | 158 / 59 | 158 / 59 | 183 / 68 | 183 / 68 | 174 / 65 | 269 |
| *val* | 34 / 62 | 34 / 62 | 34 / 62 | 34 / 62 | 38 / 69 | 38 / 69 | 37 / 67 | 55 |
| *test* | 33 / 55 | 34 / 57 | 34 / 57 | 34 / 57 | 40 / 67 | 40 / 67 | 37 / 62 | 60 |
| *total* | 225 / 59 | 226 / 59 | 226 / 59 | 226 / 59 | 261 / 68 | 261 / 68 | 248 / 65 | 384 |

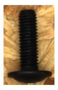
| class | *type_08* | *type_09* | *type_10* | *type_11* | *type_12* | *type_13* | | all |
|---|---|---|---|---|---|---|---|---|
| |  | | | | | | | |
| Number of objects per class and split (abs/rel[%]) | | | | | | | | |
| *train* | 254 / 8 | 232 / 7 | 249 / 8 | 249 / 8 | 232 / 7 | 233 / 7 | | 3119 |
| *val* | 59 / 9 | 44 / 7 | 46 / 7 | 46 / 7 | 44 / 7 | 44 / 7 | | 647 |
| *test* | 55 / 8 | 45 / 7 | 51 / 8 | 51 / 8 | 45 / 7 | 45 / 7 | | 660 |
| *total* | 368 / 8 | 321 / 7 | 346 / 8 | 346 / 8 | 321 / 7 | 322 / 7 | | 4426 |
| Number of images, in which each class is contained, per split (abs/rel[%]) | | | | | | | | |
| *train* | 175 / 65 | 159 / 59 | 168 / 62 | 168 / 62 | 160 / 59 | 160 / 59 | | 269 |
| *val* | 37 / 67 | 34 / 62 | 35 / 64 | 35 / 64 | 34 / 62 | 34 / 62 | | 55 |
| *test* | 37 / 62 | 33 / 55 | 36 / 60 | 36 / 60 | 33 / 55 | 33 / 55 | | 60 |
| *total* | 249 / 65 | 226 / 59 | 239 / 62 | 239 / 62 | 227 / 59 | 227 / 59 | | 384 |

**Table 4.2: *Screws* statistics.** For each class, an example object and the absolute and relative number of instances is shown, in total and per split, respectively. Moreover, the number of images in which at least one object of the class is shown, also for each class and split.

**Screws.** The screws dataset contains 384 images of a varying wooden background on which 13 different types of screws and nuts are placed. All images are 8-bit three-channel RGB color images. Examples are shown in Fig. 4.6. The categories differ in length and width of the screw or the diameter of the nut, in the color of the metal, and in the shape of the screw's head, tip, or thread. Objects might be touching or overlapping each other and the number of objects within an image varies.

*Screws* was specifically acquired to evaluate oriented object detection methods (cf. Chapter 8). Therefore, the annotations are given as OBBs. For elongated screw classes,

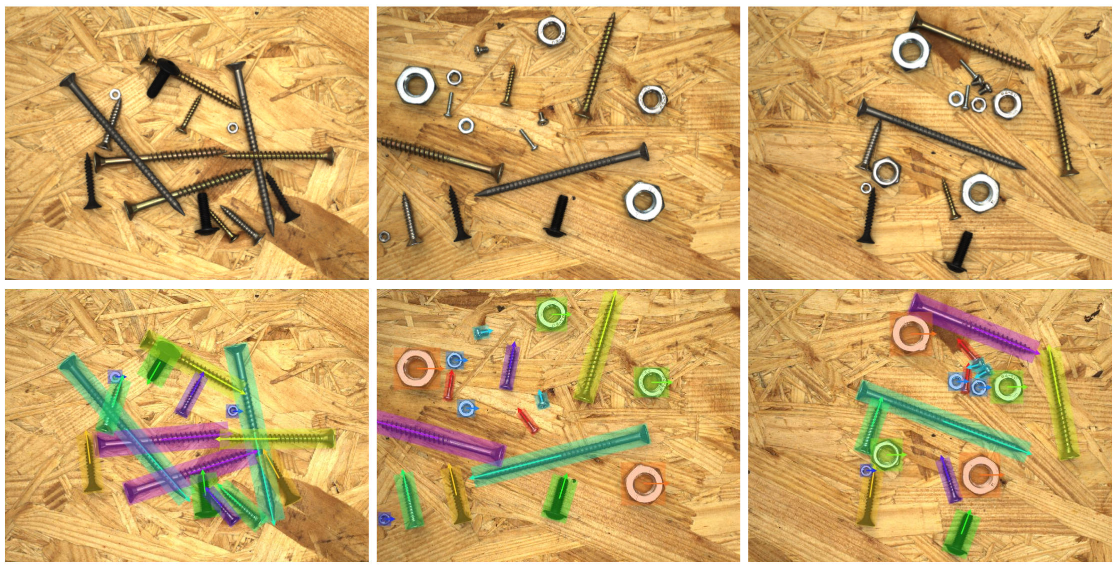


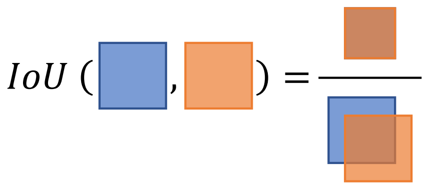
**Figure 4.6: *Screws* example images**. (*Top*) Objects are lying on a wooden background. Instances can be isolated but are also frequently overlapping or touching each other. (*Bottom*) The ground truth for *Screws* is given as OBBs.

the orientation is defined from the screw's head to its tail, parallel to the thread. The nut categories are symmetric. Thus, they have been labeled with AABBs to make their orientation unambiguous. In comparison to *Pill Bags*, the dataset is not labeled with pixel-wise mask annotations. Exemplary annotations are shown in Fig. 4.6.

Per class examples and detailed statistics about the number of objects within the *Screws* dataset are given in Table 4.2. Overall, the dataset is very balanced: each class is contained in approximately 60% to 70% of the images, and the total number of objects per class is uniformly distributed, as well. At least 44 instances are present for each class in the validation and test sets, such that a valid evaluation is possible.

## 4.2 Evaluation Metrics

In comparison to classification, where the predicted class can either be right or wrong, for detection we get a new dimension of the evaluation with respect to the localization. To measure how well a prediction $A$ fits a ground truth box $B$, we measure the intersection over union (IoU), i.e., the area of the intersection divided by the area of the union as depicted in Fig. 4.7:



$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{4.1}$$

**Figure 4.7: IoU.** Intersection over union.

For a prediction, the localization is correct if the IoU with a ground truth box is higher than a threshold $T_{\text{IoU}}$. Depending on the necessary precision, $T_{\text{IoU}}$ can be chosen higher or lower.

To be a true positive (TP), the prediction needs to have a correct localization and also the correct class, i.e., the same as the ground truth box that it overlaps (the most).

Using this definition, there are a number of different false positive (FP) cases, as depicted in Fig. 4.8. For example, there should be no other correct prediction that overlaps more, since in this case the prediction is a duplicate FP. Most of the *duplicate* FP cases can usually be filtered out using a non-maximum suppression algorithm. However, this requires that the score of the box is reliable in the sense that a box with better localization also obtains a higher score.

Note that except for the *duplicate* or the *wrong class* case, an FP prediction usually leads to a false negative (FN) GT instance.

For object detection, the case of a true negative (TN) would be background, i.e., no object recognized as no object. Since background parts of the image are all parts where no object is detected, this case is not explicitly present in the evaluation.

If we denote $\mathcal{G}$ as the set of all GT boxes, $\mathcal{FN}$ the set of FN boxes, $\mathcal{P}$ the set of all predictions, $\mathcal{TP}$ the set of all TP predictions, and $\mathcal{FP}$ the set of all FP predictions, we have the following well-known relations:

$$\text{Recall} = \frac{|\mathcal{TP}|}{|\mathcal{G}|} = \frac{|\mathcal{TP}|}{|\mathcal{TP}| + |\mathcal{FN}|}, \ \text{Precision} = \frac{|\mathcal{TP}|}{|\mathcal{P}|} = \frac{|\mathcal{TP}|}{|\mathcal{TP}| + |\mathcal{FP}|}. \tag{4.2}$$

**Mean Average Precision.** To compare different detection methods on a dataset, the authors of *VOC* [39] established the mean average precision (*mAP*) that summarizes the detector quality in a single measure. Therefore, first, the average precision (*AP*) for each category is calculated as the area under the interpolated precision–recall (PR) curve for a given IoU threshold. The *mAP* is then the mean over the per-class *AP* values. For convenience, we use the frequently used notation in the literature and write *AP* instead *mAP* also for the class-averaged *AP*.

While on *VOC* only the IoU threshold of 0.5 is used to account for inaccurate annotations, the predominant measure on *COCO* averages the area over 10 IoU thresholds [0.5 : 0.05 : 0.95] in order to give more weight to accurate localizations.[5] If multiple IoU thresholds are used, the *AP* is the average over the per-IoU threshold values. A value at a specific threshold, e.g., 0.5, is indicated as *AP*@0.5, or $AP_{50}$. Moreover, we show *AP* values as percentage numbers and sometimes neglect the %-sign within this thesis, for convenience. If we mention an "X% *AP*" improvement, we usually refer to an improvement by X pp. Only if we write about a "relative" improvement, we actually mean the relative improvement in percent.

To obtain the PR curve for a given class, all predictions for that class are first sorted

[5]Although this is the current standard measure in the research community, we doubt that the annotations of *COCO* are accurate and consistent enough to justify the evaluation with IoU thresholds above 0.85. This is further addressed in Chapter 6.

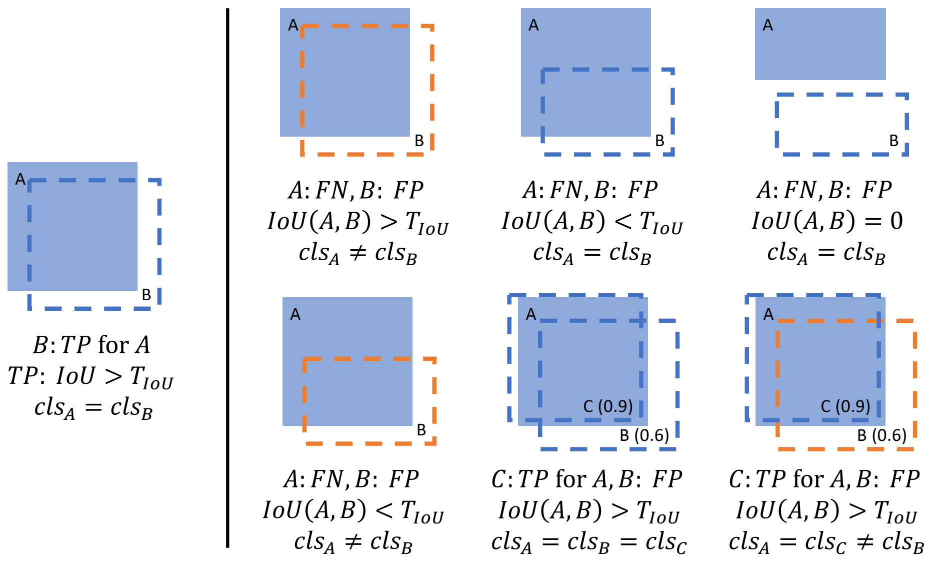


**Figure 4.8: Cases for true and false positives.** (*Left*) A TP occurs when the correct class is predicted and the IoU to the GT is high enough. (*Right, from left to right, top to bottom*) Different failure cases: B is a FP if the class is wrong (*cls*), the class is correct, but the IoU is too low (*localization*), or zero (*background*), the class is wrong and the localization is bad (*multiple*), or if there is another correct prediction that achieves a higher IoU (*duplicate*).

decreasingly by their score or confidence. A set of used predictions is initialized as empty set. Then, one-by-one the next highest scoring prediction is added to the used predictions and a new precision–recall pair (at the given IoU threshold) is calculated from the used predictions. Whenever a TP prediction is added, the recall increases and the precision increases (if it is not already 1.0); if a false positive is added the recall stays the same, but the precision decreases.

In the *COCO* evaluation protocol, the PR curve is interpolated as follows: For every calculated precision value $p_j, j = 1, \ldots, n_d$, where $n_d$ is the number of detections, the interpolated precision is given by

$$\hat{p}_j = \max_{i>j} p_i. \tag{4.3}$$

To approximate the area under the curve, the interpolated precision is evaluated and averaged at 101 (*VOC*: 11) equidistant recall values from 0.0 to 1.0.

Some examplary PR curves are shown in Fig. 4.9.

Today, *COCO AP* is the predominant measure for object detection and instance segmentation within the community. We will mostly use *AP* for model comparisons within this thesis, because this allows a comparison with other methods and benchmarks. Moreover, we will propose a refinement of *AP* in Chapter 6.

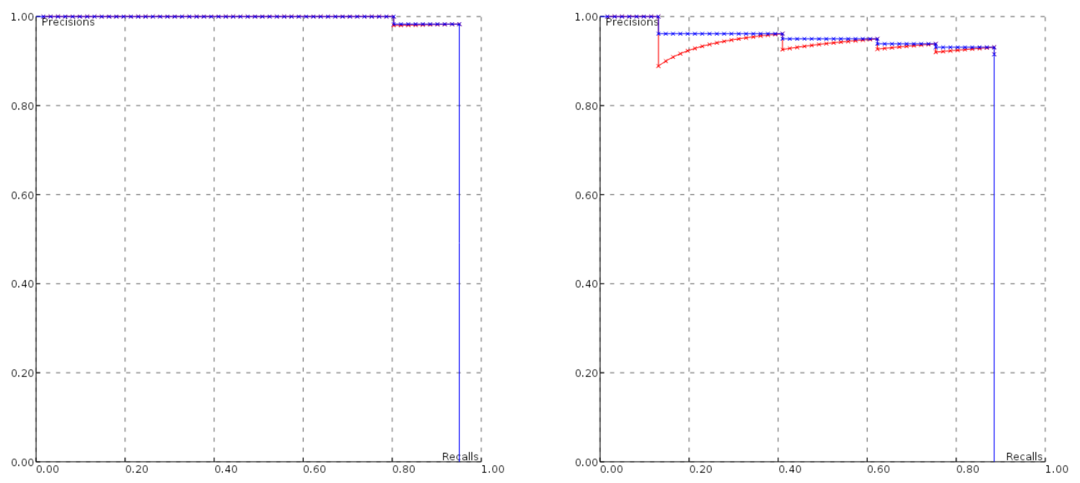


**Figure 4.9: Exemplary PR curves.** PR curves are shown for two different IoU thresholds, (*red*) without, and (*blue*) with interpolation. (*Left*) Evaluation with IoU threshold 0.75, (*right*) evaluation with IoU threshold 0.85. The figure shows that a slightly increased IoU threshold can have a large impact on the result. For better visibility of the differences, the non-interpolated PR curve is shown linearly instead of as a left-continuous step function.

## 4.3 Box Detection Methods

Object detection models generally follow the workflow depicted in Fig. 4.10:

1. From the input image, features are extracted.

2. From the input image, *region proposals*, i.e., regions where objects might be located, are extracted. Usually, the extracted features from the previous step are used as input for this task. Typically, this step is configured to favor recall over precision, such that at least one proposal is present for each object within the image. Regions are typically given as boxes because this is the desired output.

3. Region proposals are classified into one of the possible categories or assigned to the background (discarded). The extracted features (usually around/within the region proposal) are used as input to the classifier.

4. The (usually coarse) region proposals are refined by a box regression. Therefore, the difference between the proposal and the corresponding ground truth box parameters are the targets of a regression during training. Hence, during inference this module predicts individual parameter adaptions (deltas) to improve the box accuracy. Also here, the extracted features (around/within the region proposal) are input to the regression. The box regression can be done individually for each predicted class (class-specific) or independent of the class (class-agnostic).

5. The regressed boxes are deduplicated by a non-maximum suppression (NMS) to obtain the final result. As for the box regression, the non-maximum suppression can be done class-specifically or class-agnostically.

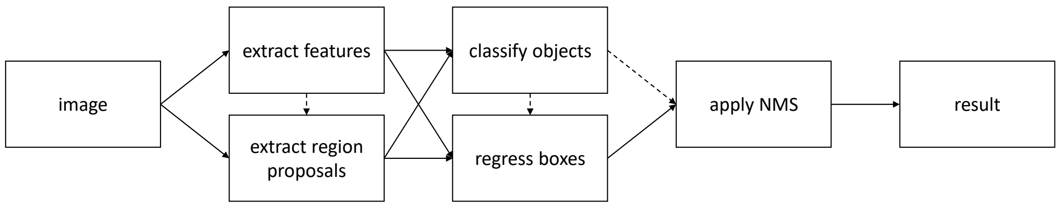


**Figure 4.10: General detection workflow.** Up to small variations, most detection methods follow the workflow shown here. Dashed connections indicate optional requirements. See text for a detailed explanation.

We will describe each of the different steps in more detail, especially those for which we propose changes in one of the subsequent chapters. Often, the differences between the proposed methods only lie in one step of the workflow.

As with many computer vision fields, object detection algorithms have made a massive progress at the beginning of the last decade, when deep CNNs, such as Alexnet [95], ZF net [206], and later Inception [183] and ResNet [68] have revolutionized the way of feature extraction in images. Today, all state-of-the-art object detection methods rely on deep features, but some of the pre-deep-learning age algorithms have led to very impressive results at the time they were published. Also some concepts of that time still remain until today and therefore it is worthwhile to have a brief overview of earlier methods.

### 4.3.1 Classic Methods

The two methods we present here are just representatives of a number of early object detection methods. They have in common that they follow a sliding window approach. That means that a kind of filter is slid densely across the image or across one or several feature maps that are calculated from the image in step one. The filter is supposed to *fire*, i.e., produce a high activation, whenever an object is present and should have a low answer to the input values when this is not the case. This dense sliding of the filter over the image can be seen as step two of the general workflow presented above in the sense that an object is proposed at every pixel of the feature map where the filter is centered. Thresholding the filter activation leads to a classification into object and not object, i.e., workflow step three. We will see later that modern deep-learning-based methods are still similar to such a sliding window approach, at least to some extent.

**Histogram of oriented gradients.** Similar to the SIFT keypoint descriptor [128], the histogram of oriented gradients for person detection (HOG) approach [27] is based on the gradient image. The image is therefor divided into small spatial regions where the orientation of the image gradient directions are accumulated into a one-dimensional histogram. Each such cell is then contrast-normalized over a block of neighboring cells. The idea of the HOG detector is to learn the weights of a filter that can be applied to the HOG feature map and should return a high response whenever the object is present and

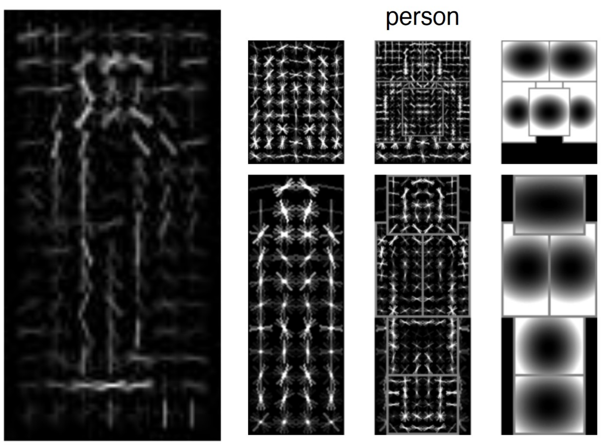


**Figure 4.11: HOG and DPM models.** (*Left*) The weighted filter response of the HOG person detector. (*Right*) The learned two component DPM model trained on *VOC*, for a sitting (*top*) and standing person (*bottom*). (*From left to right*) The root filter, the part filters, and the spatial model which indicates the costs for variations of part filter locations with respect to the root filter. Images are taken from [27] and [42], respectively.

a low response when not. The filter is of size $64 \times 128$ in width and height and can be applied to different levels of an image pyramid to address scale-changes.

To train the filter weights for person detection, a linear SVM is used. The input features are the accumulated HOG-features over the detection window. Cropped person images are used as positive examples, while randomly sampled crops from person-free images are used as negative set. Moreover, false positives from a preliminary detector within the person-free images are used as hard negative examples.

During inference, the filter is slid over the entire HOG-feature map, at all levels of the image pyramid to create weighted filter responses as shown in the left part of Fig. 4.11. The neighboring pixels within a low resolution pyramid level correspond to center locations with a higher stride in the input image. That means that large objects should be found in low-resolution feature maps, while small objects are supposed to be found in high-resolution feature maps. The HOG feature map calculation can be attributed to the first feature extraction step of the general workflow. The obtained score is thresholded to get the detections. Non-maximum suppression (cf. Subsection 4.3.2) is used to avoid duplicate detections. A box regression step is not done in the HOG-approach.

**Deformable Parts Model.** Variants of the *discriminatively trained deformable parts model* (DPM) [42] were the best performing models on *VOC* just before deep learning was used for object detection.

Similar to the HOG detection approach [27] from the previous paragraph, filters are applied within the HOG feature space. The features are computed efficiently by reducing their dimensionality with a principal component analysis. Also here, first, features are extracted from the input image and the filters are slid over the feature maps to obtain a score map. This means that in the beginning, the hypothesis for each pixel location is that there might be an object at the corresponding position of the input image. The DPM model for one particular class and view consists of one root and several part filters. High scores of the root filter indicate a high probability that the object is present at this location and the hypothesis is true. In the same manner, the part filters are slid over a feature map at twice the resolution to obtain hypotheses for detailed part locations and strengthen or weaken the root filter hypothesis. A spatial model that indicates the variability of part-locations with respect to the root filter location is used to calculate the total score of the root filter location from the root score and the weighted part scores. A model for one class can consist of several components, e.g., one for the front and one for the side view of an object. The components are handled independently and could also be seen as individual objects.

The root and part filters are learned during a training phase. They are instantiated using the ground truth boxes of each class. The ground truth boxes of each class are therefore sorted by their aspect ratio. One component is created for each group of aspect ratios. Moreover, an image pyramid is used to assign different models to different scales, depending on their size. During training a hypothesis is considered as a positive example if the IoU between the root filter and a ground truth box of this class is above 0.5. All other locations are considered as negative examples for this component. Training is done with a coordinate descent approach, where iteratively the highest scoring positive examples are chosen and for them the model, i.e., root filter, part filters, and spatial model, is updated. An example for class *person* is shown in the right part of Fig. 4.11.

Models for the individual categories are initialized and trained separately. That means that each category is only trained for the decision, yes, this category is present, or no, the category is not present. This is in contrast to modern deep-learning-based approaches, where the categories are trained all together such that the model can explicitly learn the features that distinguish one class from another. However, the authors propose to rescore the results of each proposed detection by a separate SVM classifier that takes as features the (normalized) detection score and box location as well as the highest scores from all other category classifiers within the image. Moreover, the DPM uses a linear regression to infer the bounding box from the model filter parts. Input to the bounding box regression are the root filter size and location in the input image as well as the location of all part filters.

A detailed explanation of DPM goes beyond the scope of this work. However, the interested reader is encouraged to read the paper and discover how many rules and human-intuitions are involved in the DPM. Compared to deep learning approaches the parameters of the model are comparably low, but the complexity of the model architecture is a lot higher and a lot more difficult to understand.

### 4.3.2 Non-Maximum Suppression

Per se, most object detection methods are predicting multiple results for one ground truth object. For example, consider the above mentioned classic methods. They are both using a sliding window approach with highly overlapping detection windows (i.e., a small stride). Therefore, it is likely that also the neighboring windows of the best possible location indicate a detection.

The idea of non-maximum suppression (NMS) is to suppress such suboptimal detections and only keep the best possible result. The underlying assumption is that the best prediction also obtains the highest score.

The non-maximum suppression works greedily: A list of already checked detections is initialized as the empty list $\mathcal{C} = \emptyset$ and all predicted detections are added to the list of remaining detections $\mathcal{R}$. Then, iteratively until $\mathcal{R} = \emptyset$, the detection with highest score $H$ is taken from $\mathcal{R}$ and added to $\mathcal{C}$. With a user-defined threshold $T_{NMS}$, for all other detections $B \in \mathcal{R}$, if $\text{IoU}(B, H) > T_{NMS}$, we update $\mathcal{R} = \mathcal{R} \setminus \{B\}$ (discard $B$).

**Fast NMS.** For close-packed objects, consider three objects diagonally aligned in a row, where for three predictions $A, B$, and $C$, $A$ significantly overlaps $B$ and $B$ significantly overlaps $C$, but $A$ does not ovlerap $C$ (significantly). With the above notation, $\text{IoU}(A, B) > T_{NMS}$ and $\text{IoU}(B, C) > T_{NMS}$, but $\text{IoU}(A, C) < T_{NMS}$. In the case that we have for the score of $A$, $S(A)$: $S(A) > S(B) > S(C)$, the normal NMS does not discard $C$ because $B$ is already discarded before it is chosen as the highest scoring detection. This is fine and exactly what the NMS is supposed to do. However, although such cases are rare in most datasets they prohibit a more efficient *fast* NMS: Suppose we have sorted all detections descendingly by their score into a list $L = (H_1, H_2, \ldots, H_n)$. The normal NMS is called sequentially, first checking all IoUs of $H_1$ with $H_i, i > 1$, before eventually continuing with $H_2$ to check the IoUs of $H_2$ with $H_j, j > 3$, and so on. Instead, if the situation from above is not to be expected, the IoU checks for $H_i, i = 1, \ldots, n-1$ can be done simultaneously. This also has been proposed in [9], where they found that for real-time detection methods the classic NMS with its quadratic cost can be a bottleneck concerning the overall runtime.

### 4.3.3 Bounding Box Regression

An axis-aligned box can be parameterized by its row and column center coordinates $(r, c)^T$,[6] and the box width and height $(h, w)^T$. Generally, a proposal $P = (r_p, c_p, h_p, w_p)^T$ that was classified as one of the object categories does not match the underlying ground truth box $G = (r_g, c_g, h_g, w_g)^T$. The idea of bounding box regression is to improve $P$ by transforming it with a function $\hat{d}(P)$ such that the final detection $D := \hat{d}(P)$ better matches the coordinates of $G$. Hence, $\hat{d}$ is an approximation of the real but unknown mapping $d(P)$ that converts each proposal to the assigned ground truth.

Obviously, a promising approach to get a good approximation $\hat{d}$ of the transformation $d$ is machine learning. Hence, we model the box regression as a function $\hat{d}$ that has learnable weights $\theta_d$ and takes some inputs $\mathbf{x}$ to predict $D = \hat{d}(\mathbf{x}(P), \theta_d, P)$. $\mathbf{x}$ depends on $P$ in the sense that usually the input features $\mathbf{x}$ are chosen depending on $P$.

[6]In some implementations, the coordinates of the top-left corner of the box are used instead of the box center coordinates.

Girshick et al. [58] propose to model $\hat{d}$ such that

$$\begin{aligned} G_r &\overset{!}{=} D_r = \hat{d}_r(P) = h_P \Delta_r(\mathbf{x}(P)) + r_P, \\ G_c &\overset{!}{=} D_c = \hat{d}_c(P) = w_P \Delta_c(\mathbf{x}(P)) + c_P, \\ G_h &\overset{!}{=} D_h = \hat{d}_h(P) = h_P \exp(\Delta_h(\mathbf{x}(P))), \\ G_w &\overset{!}{=} D_w = \hat{d}_w(P) = w_P \exp(\Delta_w(\mathbf{x}(P))). \end{aligned} \tag{4.4}$$

The function $\hat{d}$ can be trained by a least-squares linear regression for $(\Delta_r(\mathbf{x}(P)), \Delta_c(\mathbf{x}(P)), \Delta_h(\mathbf{x}(P)), \Delta_w(\mathbf{x}(P)))^T$. From (4.4), for a pair of proposal $P$ and assigned ground truth $G$, the targets $(t_r, t_c, t_h, t_w)^T$ for $(\Delta_r(\mathbf{x}(P)), \Delta_c(\mathbf{x}(P)), \Delta_h(\mathbf{x}(P)), \Delta_w(\mathbf{x}(P)))^T$ are inferred as

$$\begin{aligned} t_r &= \frac{r_G - r_P}{h_P}, \\ t_c &= \frac{c_G - c_P}{w_P}, \\ t_h &= \log(h_G / h_P), \\ t_w &= \log(w_G / w_P). \end{aligned} \tag{4.5}$$

To avoid large regression targets, only proposals that satisfy $\text{IoU}(P, G) > T_{\text{IoU}}$ are used to train $\hat{d}$. $T_{\text{IoU}}$ is often set to 0.5. Cai and Vasconcelos [14] propose an iterative box refinement trained with increasing thresholds $T_{\text{IoU}}$ to improve the box quality. Depending on what is considered as input features of the functions $\Delta_\star$ for $\star \in \{r, c, h, w\}$ are sometimes the identity, several layers of a CNN, or a linear function $W_\star \mathbf{x}$. Independent of the exact form of $\Delta_\star$, a box regression loss $L_{box}$ is formulated and summed over each pair of (overlapping) proposals and ground truth boxes $\mathcal{B}_{box} = \{(P, G) | \, \text{IoU}(P, G) > T_{\text{IoU}}\}$:

$$\mathbf{L}_{box}(\mathcal{B}_{box}) = \sum_{(P,G) \in \mathcal{B}_{box}} L_{box}(\mathbf{x}(P), P, G) = \sum_{(P,G) \in \mathcal{B}_{box}} \sum_{\star \in \{r,c,h,w\}} L_{box}\left(t_\star - d_\star(\mathbf{x}(P))\right). \tag{4.6}$$

In recent models, $L_{box}$ is usually chosen as a Huber loss [80] which in the context of detection was first proposed in [57] (named *Smooth L1* loss with $\beta$ set to 1):

$$L_{box}(y) = \begin{cases} 0.5y^2/\beta & \text{, if } |y| < \beta \\ \beta|y| - 0.5\beta^2 & \text{, otherwise.} \end{cases} \tag{4.7}$$

As mentioned before, the box regression can be done for each class independently (class-specific) or independent of the class (class-agnostic). The benefit of a class-specific box regression is that specific features can be learned that better characterize the object boundaries. This bears the potential of more accurate results. On the other hand, depending on the exact architecture, with a high number of classes the memory consumption and runtime can be increasing a lot compared to a class-agnostic box regression setup. Of course, in the class-specific case during training only proposals that fulfill both the IoU threshold criterium and have the same predicted class as the underlying ground truth box are considered. Because the batch size is often rather small for detection models ($< 16$), especially for rare categories a class-specific box regression can lead to very sparse training signals: a loss and gradients for the part of the model that predicts the box deltas for a specific class is only computed if this class is present within the batch and at least one proposal surpasses the IoU threshold $T_{\text{IoU}}$.

### 4.3.4 Deep-Learning-Based Object Detection

In comparison to the presented classic methods, the first step that was based on deep learning was the feature extraction part. Subsequently, with the progress in detection that has been made during the past years, methods also made use of deep CNNs to model some of the other steps within the general workflow. However, some parts, such as the NMS or the box-regression are still very similar to the DPM.

#### R-CNN

Shortly after Krizhevsky et al. [95] achieved a breakthrough for classification in *ILSVRC* with *AlexNet* in 2012, Girshick et al. [58] *"bridged the gap from image classification to object detection"* with deep CNNs and obtained the same relative progress on *VOC* (from 33% to 54% $AP_{50}$). In comparison to previous methods, in the regions with CNN features (R-CNN) approach the hand-crafted feature extraction part was exchanged by CNN-based *deep* features. The pipeline of R-CNN is as follows: First, around 2000 so-called region proposals are obtained by *Selective Search* [190] or any other method to generate category-independent boxes at positions where objects are likely (e.g., [3, 15, 18, 36, 216]). Second, each proposed box is used to crop the corresponding part out of the input image and warp it to a fixed input size of the CNN that is used for feature extraction. With fixed input size (e.g., $227 \times 227$ for AlexNet), also the obtained feature size is of fixed length. These features are then fed into $n_c$ independent SVMs that have been trained to classify the box into one of the $n_c$ possible classes of interest. Moreover, MLPs are used to regress the region proposal box parameters to better fit the underlying object. The box refinement is the same as in the DPM explained in more detail in the following Subsection 4.3.3.

This means that in R-CNN, the first and the second step of the general workflow (cf. Fig. 4.10) are interchanged. Moreover, the region proposal method, as a stand-alone part of the algorithm, relies on features that are independent of the object classification and box refinement. In comparison, the box refinement and classification SVMs use the same features that are generated by the CNN. However, all trainable steps, i.e., feature extraction, classification SVMs and box-regression SVMs are trained individually in a stage-wise manner. In particular, the feature extraction CNN is even trained on a separate, large-scale image classification dataset, such as ImageNet [28].

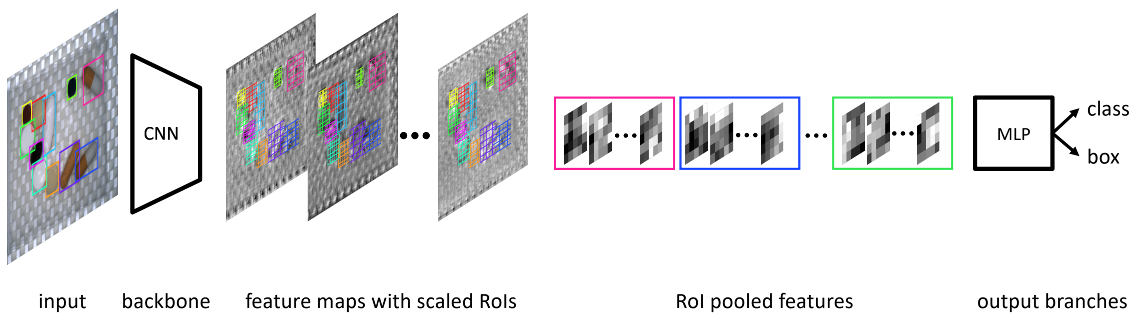


**Figure 4.12: Fast R-CNN pipeline with RoI pooling.** (*From left to right*) The whole input image is fed into a CNN to extract features. During RoI pooling, proposal boxes are scaled with respect to the feature map spatial dimension and a grid is overlayed to each channel of the feature map. For each proposal, and each bin of the grid a channel-wise max-pooling is performed to obtain a fixed feature-length for each proposal. The output batch-size of the RoI pooling operation is the number of proposals times the input batch-size. Fc layers are attached to use the RoI-pooled features for classification and box regression. Feature maps are zoomed for better visibility.

### Fast R-CNN

The results of R-CNN have improved previous baselines on benchmark datasets like *VOC* significantly. However, the method is far from being real-time capable. During inference, the region proposal method generates between 300 and 2000 proposal boxes, and each of them is used to crop and warp a part of the image that is subsequently fed as input to the CNN for feature extraction. After features have been extracted, they are used as input for the classifier and for the class-specific bounding box prediction, that are both also done sequentially proposal after proposal. At the time of publication, for inference the whole detection pipeline of R-CNN took $47s$ per image on a GPU.

**RoI pooling.** The main idea of Fast R-CNN [57] to improve the runtime is to parallelize the feature extraction, proposal classification, and box regression over all proposals within one image. Therefore, the core contribution of the approach is a *region of interest (RoI) pooling*. RoI pooling allows to process the whole input image with the feature extraction CNN (backbone), independent of the individual instantiations of region proposals. For classification and bounding box regression, two individual branches are attached to the backbone via the RoI pooling layer and two subsequent fc-layers. The RoI pooling layer pools the features of the backbone for each proposal (in parallel), as follows:

- Proposals are scaled to the last convolutional feature map before the (classification) fc-layers, by multiplying their parameters $(r, c, h, w)$ with the factor $h_f/h$ (where $h_f$ is the height of the feature map and $h$ the height of the input image).
- A grid with size $k \times k$ and depth $d_f$ is placed at each of the rescaled proposals within the feature map.
- A max-pooling is done within each of the grid bins to obtain the pooled features for each proposal. Thus, independent of the box proposal dimensions, a feature

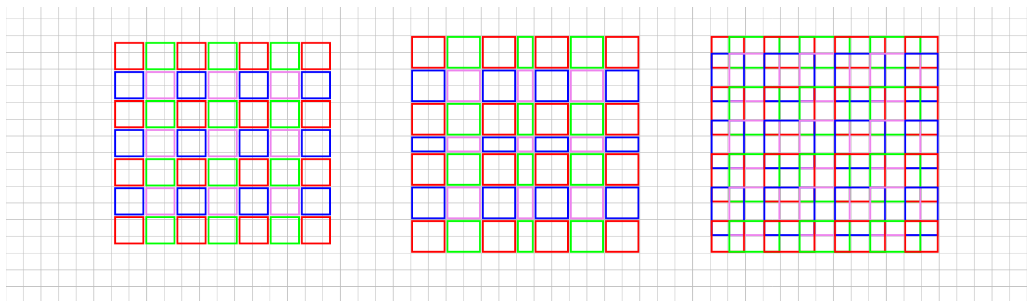


**Figure 4.13: RoI pooling types.** (*Left*) Subpixel-precise bins that are used in RoI Align. Within each bin, four (two in horizontal and vertical direction) equally spaced sampling points are distributed for which the features are bilinearly interpolated. Max- or average-pooling is done based on the sampled features. (*Middle*) One option for normal RoI pooling is to quantize the bin boundaries such that they match with the pixel-grid. (*Right*) Another option is to aim for equally-sized bins but still use rounding for the bin boundaries, which typically leads to widely overlapping bins.

map with fixed dimensions is obtained for further processing. The batch size of the RoI pooling output feature map is $n_p b$, where $n_p$ is the number of proposals within the current image and $b$ is the batch size.

Backpropagation for a RoI pooling layer is done as for a normal max-pooling layer: for each RoI bin, the gradient of the output is propagated to the position where the maximum value was present during the forward pass. The input gradient is the sum over all propagated gradients of each RoI and all RoI bins.

The Fast R-CNN pipeline without the proposal generation is visualized in Fig. 4.12. For classification and box regression of each proposal, two fc layers are connected to the RoI pooling output and from there, two individual fc layers are attached, one for classification with $n_c$ outputs and one for box regression with $4n_c$ outputs in the class-specific case. These last two branches can be seen as MLPs with two shared hidden layers. The training of the model is done with a multi-task loss that adds the gradients with respect to the *class* and *box* losses. Since the backpropagation for the RoI pooling operation is no problem, the feature extraction CNN can be finetuned together with the *class* and *box* branches.

The authors state that Fast R-CNN is up to 213 times faster during inference and 9 times faster during training than R-CNN (without the proposal extraction).

**RoI Align.** In the recent Mask R-CNN paper [70], the RoI pooling operation is enhanced to the *RoI Align* operation. A problem of RoI pooling is that the proposal boxes are given with subpixel-precise coordinates, but the RoI pooling operation requires pixel-precise bins. Therefore, as shown in the middle and right of Fig. 4.13, the bins have to be quantized to fit to the pixel grid. The approach in the middle has non-overlapping bins, but largely different bin sizes that vary with the position of the proposal box. The approach on the right has less variations in the bin size, but, especially for small proposals, widely overlapping bins.

In comparison, in RoI Align, the bins are constructed with subpixel-precise boundaries

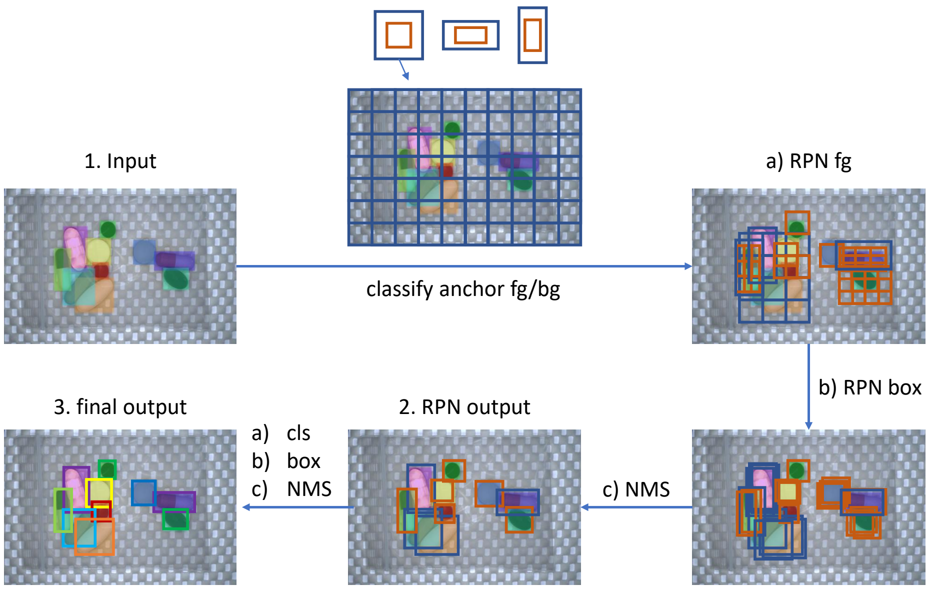


**Figure 4.14: FRCNN workflow.** See text for a detailed explanation. (*Top-middle*) Anchors are classified at each position of a feature map, leading to densely distributed and overlapping proposals across the whole image, here only visualized for one of the six indicated anchors.

and for each bin, four equally spaced sampling points, two in horizontal and two in vertical direction, are distributed. The features for the sampling points are calculated with bilinear interpolation and the max- or average-pooling operation is done based on the sampling point features.

### Faster R-CNN

The next important step towards an end-to-end trainable model was to incorporate the proposal generation into the CNN. This step was done in Faster R-CNN [159]. The benefit of replacing the proposal generation part lies not only within the end-to-end trainable network, but also within another significant reduction in runtime: With the introduced *region proposal network* (RPN), the proposal generation runtime is reduced from 2 s to approximately 10 ms, resulting in a total runtime (depending on the backbone) of approximately 100 ms/image.

With some modifications, Faster R-CNN (FRCNN) is still one of the most successful detection architectures today. We will explain the main concepts in more detail because they reappear during this thesis.

**Anchor boxes and region proposal network.** The main novelty of FRCNN is to incorporate the generation of region proposals directly into the CNN via a RPN. Therefore, box templates of various scales and aspect ratios, so-called *anchor boxes*, are proposed densely across the image. To judge whether a proposal covers an object or not, a mini-network is attached to the features of the last convolution layer of the backbone CNN: First, a convolution layer with kernel size $n \times n = 3 \times 3$, stride 1, and depth 256 is attached to the feature map to obtain intermediate features. Each position in the intermediate feature map corresponds to a set of anchor boxes at that position within the input image. The stride of the anchors, i.e., the distance between two neighboring anchors, depends on the spatial dimension of the feature map compared to the input image dimension. For example, in ZF-net [206], a stride of one in the intermediate feature map corresponds to a stride of 16 in the input image. Second, from the intermediate features two $1 \times 1$ convolutions are branching off, one for anchor classification into foreground and background (*fg/bg*), and one for anchor regression (*reg*). Hence, if $k$ is the number of anchors, the number of filters is $2k$ and $4k$ for the *fg/bg* and *reg* branches, respectively. Using a sigmoid activation instead of a softmax function for the *fg/bg* classification, the number of filters can be reduced to $k$.

In the original paper, for input images with the shorter size rescaled to 600, the number of anchors at each feature map position is set to nine: three scales $(128^2, 256^2, 512^2)$ and three aspect ratios $(0.5, 1.0, 2.0)$. If the input feature map of the RPN has a dimension of $37 \times 50$, this corresponds to a total number of anchors of 16 650.

An examplary workflow of FRCNN is shown in Fig. 4.14: From an input image (1), first each anchor at each position of the image/feature map is classified to get the foreground proposals (1a RPN fg). Second, the proposals are regressed (1b) and undergo a first, class-agnostic NMS (1c) to obtain the final proposals as the RPN output (2). Each proposal is then classified into one of the categories (or *background*) (2a), a second, optionally class-specific, box regression (2b) is done and a second NMS is done (2c) to obtain the final detections (3).

**Anchor and proposal assignment.** During training, the RPN with its *fg/bg* and *box* branches needs targets to learn which proposals should be assigned to the foreground or the background and to calculate the regression targets. In the same manner, for each proposal, a class and final box regression target has to be determined.

The concept for both assignments is the same: In principle, anchors that have a large overlap with a GT box should be assigned to this box, and hence be assigned to the foreground (or the respective class of the GT box). Accordingly, anchors, that do not have a large overlap with any GT box should be assigned to the background. Therefore, the first step of the anchor assignment is to measure the IoU of each anchor with each GT box. Then, all anchors that have an IoU $>$ FgPosThresh are assigned to the foreground. To avoid confusions for anchors for which the IoU is close to this threshold, usually only anchors with an IoU in between $[\text{BgLowThresh}, \text{BgHighThresh}]$, where BgHighThresh $<$ FgPosThresh, are assigned to the background.

During training, only those anchors or proposals that are assigned to the foreground contribute to the box regression loss. The box regression predictions of all other anchors or proposals are ignored.

The detailed decision rules for the anchor or proposal assignment are shown in Fig. 4.15 (for the case of anchors). In particular, to avoid that no anchor is assigned to a suboptimal GT box, it is always checked that there is no other GT box for which the anchor achieves a higher IoU.

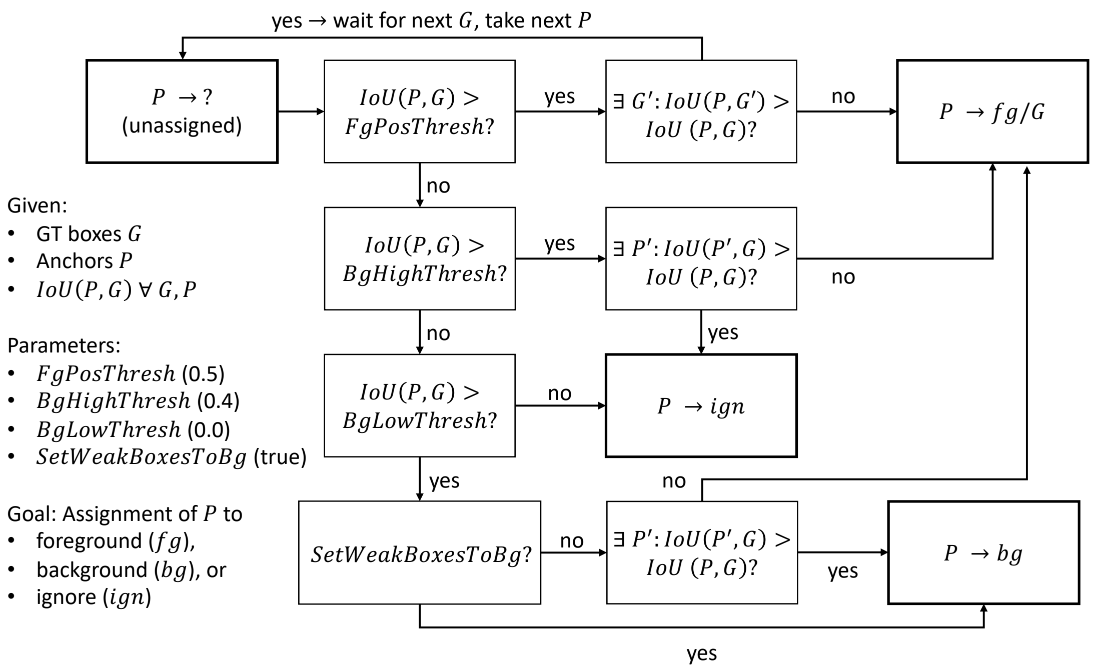

yes → wait for next $G$, take next $P$
$P \rightarrow ?$ (unassigned)
$IoU(P,G) > FgPosThresh?$
yes
$\exists\, G': IoU(P,G') > IoU(P,G)?$
no
$P \rightarrow fg/G$
no
Given:
• GT boxes $G$
• Anchors $P$
• $IoU(P,G)\ \forall\, G,P$
$IoU(P,G) > BgHighThresh?$
yes
$\exists\, P': IoU(P',G) > IoU(P,G)?$
no
no
yes
Parameters:
• $FgPosThresh$ (0.5)
• $BgHighThresh$ (0.4)
• $BgLowThresh$ (0.0)
• $SetWeakBoxesToBg$ (true)
$IoU(P,G) > BgLowThresh?$
no
$P \rightarrow ign$
yes
no
Goal: Assignment of $P$ to
• foreground ($fg$),
• background ($bg$), or
• ignore ($ign$)
$SetWeakBoxesToBg?$
no
$\exists\, P': IoU(P',G) > IoU(P,G)?$
yes
$P \rightarrow bg$
yes

**Figure 4.15: Anchor assignment rules.** The assignment of anchors to GT boxes is more difficult than it looks at first sight. The default parameters of thresholds are indicated on the left. If SetWeakBoxesToBg is set to *false*, also GT boxes with rare shape have a higher chance to get at least one assigned anchor.

In the original anchor assignment, the parameter SetWeakBoxesToBg is set to *true*. However, we noticed that especially for datasets where rare GT box shapes occur that have a low IoU with all anchors, it happens that none of the anchors is assigned to these GT boxes. Setting SetWeakBoxesToBg to *false*, the anchor $P$ that achieves the highest IoU is assigned to the GT box $G$, if $\mathrm{IoU}(P,G) > \mathrm{BgLowThresh}$.

During training, a majority of these proposals is assigned to be classified to the background class. This leads to a highly imbalanced class distribution and hinders a stable training, because assigning all proposals to belong to the background already yields a good local optimum.

One strategy to resolve this problem is to increase the lower threshold for background assignment $\mathrm{BgLowThresh} > 0.0$, such that all anchors that do not overlap with any GT box are ignored.

Another strategy is that during training a batch of $n = 256$ samples per image is chosen with a ratio of, e.g., $fg : bg = 1 : 3$. The loss of all other samples is neglected.

This random sampling can be improved by *online hard example mining* (OHEM) [171], where, during training, first the (classification) loss is computed for all proposals and then only the first $n = 256$ samples with highest loss, independent of their class, are chosen to contribute to the real loss, which is then recalculated. The idea is that with this strategy, the model concentrates on the *hard examples*, where the current model has difficulties to classify them correctly.

**Feature Pyramid Networks**

The exploitation of different scales of the input image in an image pyramid is a well known and widely used technique. Using multi-scale inputs to a detection network is a valid approach when the runtime and memory consumption is not of importance. However, for most real world applications this assumption does not hold and therefore running a costly model multiple times is not an option.

The idea of a *feature pyramid network* (FPN) is to incorporate the information from multiple scales directly into the CNN. However, this is done at the feature level and not at the input level. Common classification CNN architectures that are used as backbone for detection iteratively use strided convolutions or max-poolings to decrease the spatial dimension of the feature maps and increasing the receptive field at the same time. We can therefore assign a *level* $L$ to each feature map as

$$L = L(F^{(l)}) = \lceil \log_2(w^{(0)}/w^{(l)}) \rceil, \tag{4.8}$$

where $w^{(0)}$ is the width of the input image and $w^{(l)}$ is the width of the respective feature map. This means that the input image corresponds to level 0 and the feature map just before the third downsampling corresponds to level 2.

The idea behind FPNs is that the features from different levels have complementary information: Lower-level features contain lower-level semantics, such as edges, corners, or colors and contain information about small details, while higher-level features contain higher-level semantics, such as information about objects, textures, or context and have a wider receptive field. In order to fuse the information of both, the FPN-architecture, as depicted in Fig. 4.16, contains a decoder path from high-level features back to low-level features. The upsampling is done by transposed convolutions. To improve the information-flow from lower levels of the feature pyramid, the features from skip connections with $1 \times 1$ convolutions are added to the upsampled features of the higher level. Typically, the skip connections branch off from the last layer in a level, before the next downsampling is done by a strided convolution or a pooling operation. For shallow backbones with few levels, more levels are added by subsequent convolutions with stride 2 and usually a kernel size of $3 \times 3$. One global parameter determines the capacity of the FPN: the number of kernels of all involved convolution layers, also referred to as FPN dimension.

As shown in Fig. 4.16, the RPN can be based on the FPN features $\{P_j, j > 2\}$. Therefore, in each level of the FPN, one or several convolution layers are added to create the intermediate features of the RPN as described above. Typically, for these convolutions, the same number of kernels is used as within the FPN. Optionally, the weights of these layers can be shared across the different FPN levels, across the foreground/background prediction and the box regression branches, or both. In the FPN-based RPN, each level uses anchors of one base scale $s^{(L)}$ and $k$ *subscales* or *octaves* to get scales $\{s^{(L)} \cdot 2^{j/k}, j = 0, \ldots, k-1\}$. The base scale of the next level is given by $s^{(L)} = 2s^{(L)}$. This means that in high FPN levels large and in low levels small proposals are generated, respectively. To avoid an overly large number of anchors, usually a minimum level of

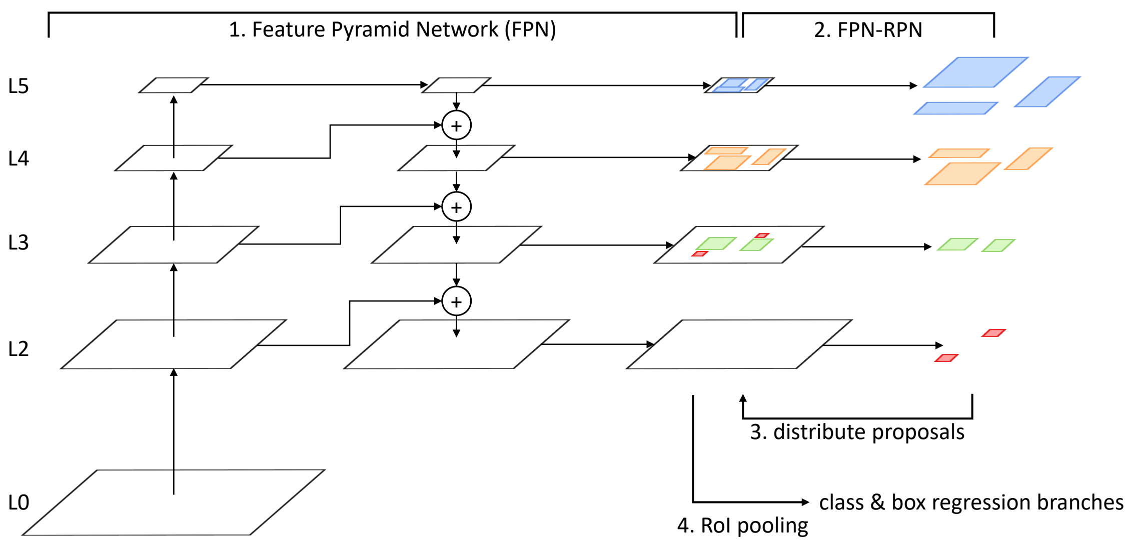

1. Feature Pyramid Network (FPN)
2. FPN-RPN
L5
L4
L3
L2
L0
3. distribute proposals
class & box regression branches
4. RoI pooling

**Figure 4.16: Detection with FPNs.** (1.) A feature pyramid is used to fuse the information from lower and higher level semantics. (2.) The RPN can be incorporated into the FPN by proposing small boxes based on low levels and large boxes based on higher levels of the FPN. (3.) For RoI pooling, the proposals are distributed to different levels of the FPN, such that their scaled size matches the resolution of the feature map (here $L_{\text{min}} = 3$ and $L_{\text{max}} = 5$). (4.) The pooled features can be used for further classification or box regression as in Faster R-CNN. The figure is inspired by [119].

two is chosen, such that the spatial resolution of the feature map is not too large. The base scale of the second level is chosen as 16, such that with the stride of four within the input image the anchors overlap and cover the whole image. Anchor aspect ratios are shared across all levels.

An FPN-based RPN usually provides a lot more proposals than the original RPN: For the orignal RPN, $12k$ proposals are generated if the level 5 feature map is used, or $47k$ when using the level 4 feature map. In comparison, using levels 2-5 an FPN–RPN generates $\approx 200k$ proposals and hence, the computation costs can increase significantly. Therefore, the necessary levels can be calculated based on the distribution of ground truth box scales within the training set. Especially avoiding low FPN levels for proposal generation can reduce the memory and runtime costs by a large factor.

When proposals have been generated, classified into foreground and background, regressed, and non-maximum-suppressed, they are distributed back to dedicated levels of the FPN features based on their scale:

$$L = \max\left(\min(L_c + \lfloor log_2(\sqrt{wh}/s_c)\rfloor, L_{\text{max}}), L_{\text{min}}\right), \tag{4.9}$$

where $(h, w)$ are the proposal height and width, $L_c$ is the canonical level, to which boxes of the canonical scale $s_c$ should be distributed, and $L_{\text{min}}$ and $L_{\text{max}}$ are the minimum and maximum levels for RoI pooling. The default setting is in accordance with the architecture of typical backbone networks such as ResNets [68]: $s_c$ is set to 224 and $L_c$ to 4 because an input image with spatial dimension of $224 \times 224$ leads to a feature map size of $7 \times 7$ in level 4, which corresponds to the default RoI pooling grid size.

According to the authors, the use of an FPN within FRCNN increases the single model *AP* on the *COCO* test-dev set to 36.2% *AP* with a ResNet-101 backbone. FRCNN using an image pyramid instead of an FPN only achieves 34.9% *AP*.

Recently, the FPN structure has been refined in various ways. *Path aggregation networks* (PANets) [123] propose to extend the FPN by another bottom-up path that iteratively decreases the intermediate feature maps from $N_2 = P_2$ to $N_5$. Here, $N_{i+1} = \tilde{N}_i + P_{i+1}$, where $\tilde{N}_i$ is generated from a $3 \times 3$ convolution with stride 2 acting on $N_i$. Moreover, the authors propose to pool the features from all FPN levels and fuse the pooled features by a max-pooling or summation to obtain an *adaptive feature pooling* mechanism.

Neural architecture search (NAS) FPN [55] propose to learn the network structure with a meta-learning approach of the FPN instead of hand-designing it. This leads to an unintuitive, but efficient architecture of connections within the FPN.

EfficientDet [187] uses a structure similar to PANet, but with skip connections between the top-down and bottom-up branches of the FPN. Moreover, they use several concatenated FPN structures. In the spirit of the used EfficientNet [186] backbone, the authors make massive use of depth-wise separable convolutions and compound scaling to keep the network efficient. Currently, this leads to the best single model result on *COCO* with up to 52.2% *AP* on the test-dev set (depending on the backbone and compound scaling factor). While PANet uses an architecture similar to FRCNN with some modifications, both NAS–FPN and EfficientDet use the single-stage architecture of RetinaNet (see below).

### Single-Stage Detectors

Detection methods similar to FRCNN contain two main stages: The first consists of the proposal generation; the second stage classifies and regresses the boxes. Because also the first stage contains a *fg/bg* classification, it is straightforward to extend the first stage to contain the classification into the final categories.

**SSD.** Similar to the FPN-architecture, the *single stage multi-box detector (SSD)* [124] uses feature maps of different scales to regress and classify anchors. However, unlike in a real FPN, there are no skip connections and there is no upscaling path. Instead, the intermediate features for box classification and regression are directly calculated from the backbone features.

By default, SSD uses five aspect ratios $\{1, 2, 3, 1/2, 1/3\}$ and only one scale per level, except for aspect ratio 1, where two scales are used (one subscale in between the current and the next level), which leads to six anchors per level. For some levels, the aspect ratios $\{3, 1/3\}$ are omitted, resulting in only four anchors. Overall, for an input image size of $300 \times 300$ (SSD300), the model generates 8732 proposals.

We can regard single-stage detectors as a variation of RPNs where the classification is done directly within the RPN and one of the two box regressions and non-maximum suppressions is omitted, respectively. One benefit of a class-specific RPN is that one anchor can lead to one detection for each category. If objects are positioned very close to each other and have a similar shape this can be an advantage. However, for the more general case where objects do not overlap frequently, the NMS parameters of one-stage detectors need to be carefully tuned in order to avoid many duplicate detections. In comparison to a two-stage detector, one-stage detectors such as SSD do not need any RoI pooling operations. The models are more light-weight with respect to memory consumption and runtime. However, due to the refinements that can be done in the second stage, two-stage detectors such as FRCNN are usually more accurate.

On the *VOC* 2012 test set, SSD300 achieves an $AP$@0.5 of 77.5% and outperforms the original FRCNN baseline (75.9% $AP$@0.5 without FPN) while running at 46fps. Increasing the input image size to $512 \times 512$, the $AP$@0.5 can be increased to 80.0% while increasing the runtime to 22fps. SSD512 increases the number of proposals to 24 564. Note that a major part of the runtime is dedicated to the backbone CNN. The numbers shown here are for the original version with a relatively heavy VGG-16 [177] backbone.

**RetinaNet.** RetinaNet [120] is a very similar architecture to SSD, but uses a full, but class-specific FPN–RPN as described above. It uses ResNet backbones with feature pyramid levels 3 to 7 by attaching two additional levels to the last ResNet feature map (level 5). As described above, this is done by two consecutive $3 \times 3$ convolutions with stride 2.

In comparison to FRCNN, the proposal classification and regression is not done with fc-layers but remains fully convolutional, where the class probabilities or regression predictions are encoded in the output depth of the last convolution layer. The intermediate features for the class and regression predictions are modeled as individual branches with $n = 4$ convolution layers each.

By default, RetinaNet uses three subscales and three aspect ratios ($\{1/2, 1, 2\}$) per level. The authors found that on *COCO*, sharing the weights of class and box regression branches among different FPN levels improved the accuracy.

**Focal loss.** Generating proposals at multiple feature levels usually leads to a very high number of proposals (e.g., $\approx 200k$ for levels 2–5 and images with the smaller side length scaled to 800 px). A majority of these proposals are not overlapping significantly with an object and should be predicted as belonging to the background. That means that the proposal classification problem is very unbalanced. Lin et al. [120] determined that most of the background proposals are correctly classified, but as their confidence for the background class is not exactly 1.0, the loss is still dominated by those *easy* samples.

The idea of the *focal loss* function [120] is to down-weight those easy samples by reshaping the cross-entropy (CE) loss. Starting from the CE loss for binary classification, where $y = 1$ means yes, the sample belongs to this class, $y = -1$, no it does not belong to this class,

$$CE(p,y) = \begin{cases} -\log(p) & \text{, if } y = 1, \\ -\log(1-p) & \text{, if } y = -1, \end{cases} \tag{4.10}$$

we define

$$p_t = \begin{cases} p & \text{, if } y = 1, \\ 1-p & \text{, if } y = -1. \end{cases} \tag{4.11}$$

Moreover, the CE loss can be *balanced* by down-weighting classes that occur often and up-weighting rarely occurring classes. Therefore, $\alpha_t \in [0,1]$ is introduced, such that $\sum_t \alpha_t = 1$.[7] Usually, $\alpha_t$ can be set to the inverse class frequency. The $\alpha$-balanced CE loss is thus written as

$$CE(p) = -\alpha_t \log(p_t). \tag{4.12}$$

Lin et al. [120] then define the $\alpha$-balanced focal loss (FL) as

$$FL(p) = -\alpha_t(1-p_t)^\gamma \log(p_t). \tag{4.13}$$

They claim that while the $\alpha$-balancing differentiates the importance of particular classes and their contribution to the loss, the reshaping with, e.g., $\gamma = 2$ shifts the attention away from easy examples towards difficult examples, independent of the class. This stabilizes the training, especially for detection models with thousands of potentially easy to classify background proposals. The authors propose to set $\alpha_t = 0.25, t = 1, \dots, n_c$ and $\gamma = 2$.

To further stabilize the training at the beginning, a prior probability is introduced that makes it more likely for the model to predict the background class. This is implemented by initializing the bias of the last convolution filters such that the class probabilities are initially only 0.01.

### Anchor-free Detection

Detection systems based on anchors like FRCNN or RetinaNet are still at the top of the *COCO* leaderboard today. However, their construction is complex and they require a relatively large amount of memory and runtime. That makes them hardly usable on embedded platforms, especially without a GPU. Therefore, *anchor-free* methods become more and more popular.

The first CNN-based anchor-free method that achieves reasonable results while being real-time capable, is YOLO [158]. YOLO subdivides the image into $7 \times 7$ tiles and for each tile, up to $B = 2$ boxes are predicted, as well as the probability of each out of $n_c$ classes to be present within the tile. Hence, for each tile, $B \cdot 5 + n_c$ values are predicted, the 5

[7] $\alpha_t$ is seen here for all classes separately because in RetinaNet, a sigmoid activation is used instead of a softmax. This means that it is only differentiated between the presence of the particular class or the background, but each class is handled independently of other classes.

values per box are $(d_r, d_c, d_w, d_h, \tilde{\text{IoU}})$. Four deltas, two for the offsets of the box center relative to the tile center and the box width and height relative to the image dimensions, and the predicted IoU of the box with the ground truth. Therefore, in comparison to the thousands of proposals that must be evaluated in an anchor-based detection model, in YOLO only 49 tiles are classified and an additional 98 box parameters are predicted. Already at the time the method was published, it reached impressive 45 fps while having an *AP* of 63.4% on *VOC 2012*. A faster, but less accurate variant even reaches 145 fps. YOLO uses an input size of $448 \times 448$ which corresponds to twice the input side lengths of the images that are used for pretraining the network on ImageNet.

The performance of YOLO has been improved in subsequent publications. In YOLO v2 [156], the authors incorporate several ideas to improve the localization accuracy on the *VOC 2007* test set: batch normalization (+2% *AP*@0.5); increased image size for classification pre-training (+4% *AP*@0.5); reduction of input size to $416 \times 416$, removal of fc layers, and use of anchors (-0.3% *AP*@0.5,+6% recall); determination of anchor box shapes with k-means on the training set to get better box priors, definition of anchor center offsets relative to the grid cells, such that they are constrained between zero and one (+5% *AP*@0.5); incorporating higher-resolution features (+1% *AP*@0.5). Overall this lifts the performance of YOLO v2 to 73.4% *AP*@0.5 on the *VOC 2012* test set, such that it is on the same level as SSD and FRCNN, but still significantly faster.

In a third version, YOLO v3 [157] proposes another set of changes: the anchor assignment only assigns the best matching anchor to each GT box and ignores all others. The network is an FPN architecture with three different levels and only 3 anchors per level. The backbone is changed to a heavier and modern architecture with skip connections.

Although from version two onwards the performance of YOLO has been increased significantly, it also became an anchor-based method, similar to SSD or RetinaNet.

**Modern anchor-free methods.** Recently, several other anchor-free methods have been proposed. We do not address them in this thesis, but we mention them here briefly for completeness.

CornerNet [101] proposes to predict heatmaps for the locations of top-left and bottom-right box corners. They also introduce a *corner pooling* method to address the problem that box corners are often lying outside of the object itself. For a single-model evaluation on *COCO test dev* the authors state 40.5% *AP*. The method is refined in [102] to improve the efficiency by using other backbone networks and a box-related zooming strategy. The latter makes it a two-stage approach and removes the simplicity of the original CornerNet.

The idea of ExtremeNet [212] is very similar to CornerNet, but instead of predicting the top-left and bottom-right points of the box, ExtremeNet predicts the top-, left-, bottom-, and right-most points of an object. The authors show that this can also be easily extended to predict octagons instead of axis-aligned bounding boxes. Moreover, the center of objects is predicted in order to assemble the keypoints together to form the correct bounding box.

CenterNet [32] is an extension of CornerNet that also predicts a heatmap for the center points of detections. As in ExtremeNet, this helps to reduce the number of false positives of the CornerNet results. Moreover, the corner pooling method is extended to a center pooling method and to a cascaded corner pooling method.

In *fully convolutional one-stage* (FCOS) object detection [188], the detection problem is framed as a regression from feature map pixel centers: For each point in a feature map, the offsets to the top, left, bottom, and right $(t, l, b, r)$ borders of the box the pixel is contained in are predicted. An FPN is used such that small and large boxes are predicted in different levels and overlapping boxes can be handled (at least to some extent). Moreover, similar as in CenterNet, a *centered-ness* score is predicted to help the NMS and avoid the prediction of poorly localizing boxes.

Very similar to FCOS, FoveaBox [93] predicts offsets to the box boundaries. In comparison, the classification targets are changed, such that only pixels within the inner part of the box are assigned to the GT box class, feature map pixels further away from the inner box area are assigned to the background, and pixels in between are ignored during training. Thus, the thresholded class probabilities lead to fewer detections, making the post processing more efficient.

The *feature selective anchor-free* (FSAF) [214] module is very similar to the anchor-free box prediction branches of FoveaBox. In comparison, during training the authors propose to predict all boxes within all feature levels, but only select the best level (i.e., the one with the smallest accumulated loss over the box center area) to contribute to the loss computation. The authors propose to use an IoU loss for the box regression. Moreover, they show that the FSAF module can be simply combined with an anchor-based prediction to improve the overall performance.

## 4.4 Instance Segmentation Methods

The task of instance-aware semantic segmentation (instance segmentation, for short) can be interpreted as the combination of semantic segmentation and object detection. While semantic segmentation methods predict a semantic category for each pixel [126], object detection focuses on generating bounding boxes for all object instances within an image [159]. As a combination of both, instance segmentation methods generate pixel-precise masks for all object instances in an image. While solving this task was considered a distant dream a few years ago, the recent advances in computer vision have made instance segmentation a key focus of current research [70, 116, 126]. This is especially due to the progress in deep convolutional networks [107] and the development of strong baseline frameworks such as FRCNN [159] and FCN [126].

Earlier instance segmentation methods did not propose category-specific instances, but only modeled a single category *object*. They were also used in the sense of an object-proposal algorithm, e.g., to replace the first step within an object detection method such as Fast R-CNN [57].

One of the first deep-learning-based instance mask proposal generation method is DeepMask [151]: In a sliding-window approach, a CNN is used to extract features from an image pyramid. Whenever the patch is centered on an object that is fully contained within the patch, one subnetwork branch is used to classify the patch to belong to an object. Another parallel branch is used to predict the instance mask for this patch. Both branches are attached to the features of the last convolution layer of a VGG [177] backbone. The branches consist of a $1 \times 1$ convolution followed by two subsequent fc layers. Since the different scales of the image pyramid are handled individually, DeepMask needs a merging strategy and a lot of effort to calculate the targets for training. Moreover, due to the many involved large fc layers and the heavy backbone, the model is computationally expensive, resulting in an average processing time of 1.5 s/image on *COCO*.

The model has been extended to SharpMask [152] by changing the segmentation branch of DeepMask to a fully convolutional decoder structure similar to the top-down path within an FPN. However, the method still uses the same patch-based training and inference strategy as DeepMask.

Dai et al. [24] also generate instance-aware mask proposals, but formulate the problem in a fully convolutional network [126]. Their network has one branch to predict the objectness of a sliding window on the feature map and one branch to predict position-sensitive score maps. The position-sensitive score maps are trained to be active at the relative locations of each instance. Therefore, the bounding boxes of the ground truth instance masks are divided into a $k \times k$ (e.g., $3 \times 3$) grid. Each score map is then trained to fire at one grid cell (e.g., the top-left, center, or right cell of the instance), but accumulated over all instances within the image. During inference, for each sliding window position, the $k^2$ score maps are assembled to obtain the instance mask at positions where the objectness score is high enough.

To obtain a proper instance segmentation model that predicts a class for each instance mask, Zagoruyko et al. [204] propose to feed box proposals obtained from DeepMask instance masks into a variant of Fast R-CNN [57]. The resulting boxes are then fed again into DeepMask to obtain a final classified instance mask. The authors state that the second DeepMask application is optional and could be avoided together with the box regression within Fast R-CNN.

At the same time, a significantly better result could be obtained by the *multi-task network-cascade* (MNC) method [25]. MNC is a multi-stage approach consisting of three main stages. The first stage generates box proposals, just as the RPN within FRCNN. In the second stage, the box proposals are used to RoI pool features from the backbone and regress a class-agnostic instance mask for each proposal using fc-layers. The third stage again RoI pools features, but now the features are masked using an element-wise multiplication with the obtained mask from the previous stage. These features are used for the classification of each instance. Optionally, in the third stage, a second box-regression can be added and stages two and three can again be applied to obtain a five-stage network. The authors also propose a RoI warping module to replace the original Fast R-CNN [57] RoI pooling operation. In comparison, during the backward pass, the RoI warping layer also calculates the gradients with respect to the box-coordinates of the used proposals and not only the gradients with respect to the feature map from which the features are pooled. MNC won the *COCO* 2015 instance segmentation challenge with an *AP* of 24.6% on the *test dev* set (using a single model with ResNet-101 backbone and images zoomed such that the shorter side-length is 600 px).

The idea of MNC to propose instance masks by an integrated branch within an end-to-end trainable box prediction network remains. Still today, most instance segmentation methods are similar to the FRCNN [159] two-stage object detection architecture. The first stage, i.e., the RPN stage, proposes axis-aligned class-agnostic boxes at all locations where an object is likely to be found. The second stage pools box-specific features with a region of interest (RoI) pooling and refines them to classify the proposals into one of the target classes or background. A second, parallel branch (sometimes with shared weights) is used to further improve the box coordinates via bounding-box regression. To predict instance masks, a third stage is used that pools features with respect to the box proposals. The methods mainly differ in the way the different branches are built, or where exactly they are attached, or how they are connected to the backbone or the other branches.

The *COCO* 2016 segmentation challenge winner FCIS [116] proposes to predict two position-sensitive score maps for each category, similar to the Instance FCN method [24]. One predicts an inside map that fires at positions where the object is present, the other predicts the outside map that is active outside of the object's boundary. Instead of using a sliding window approach, as was done by [24], here the score maps are only assembled at the best scoring RoIs that are proposed by the RPN. For each RoI, the class-specific instance mask is obtained by the softmax over the inside and outside map, evaluated at the inside map index. The class score is obtained by first taking the pixel-wise maximum over the inside and outside map and thereafter applying a spatial average pooling. FCIS achieved a single model result of 29.2% *AP* for instance segmentation on the *COCO test dev* set (using the same settings as MNC above).

In Mask R-CNN (MRCNN), He et al. [70] propose to RoI pool features using FPN–RPN proposals [119] (Section 4.3) to simultaneously predict and regress the class-specific box result, and to predict instance masks. Typically, for mask prediction a larger RoI pooling grid size of $14 \times 14$ is used. The architecture of the mask prediction branch is similar to the decoder of a fully convolutional network for semantic segmentation [126]. It consists of $k = 4$ intermediate $3 \times 3$ convolutions before a transposed convolution up-samples the features by a factor of 2 in each spatial dimension. Finally, mask probabilites are predicted using a sigmoid activation. The class score, which also reflects the model's confidence for each instance is predicted in the class head. In case of a class-specific mask branch, the predicted box category is also used to choose the correct mask prediction. Without the FPN, MRCNN already outperforms FCIS with 33.1% *AP* on *COCO test dev*. Using an FPN and a ResNeXt-101 backbone [201], the *AP* can be improved to 37.1%.

PA-Net [123] optimizes the information flow within MRCNN by fusing the features of several feature pyramid network (FPN) [119] levels to improve the quality, but at the cost of runtime. This improves the *AP* on *COCO test dev* only marginally to 38.2%.

Mask Scoring R-CNN [79] learns to predict the quality of the mask predictions, which improves the *AP* by re-calibrating the predicted scores such that low quality masks also receive a lower score compared to higher quality masks. Thus, for Mask Scoring R-CNN, the approximately 1% gain in *AP* on *COCO* is based on a better model uncertainty.

Among the more recent methods, YOLACT [8] improves the speed of instance segmentation by using a linear combination of prototypes for mask prediction and the fast variant of non-maximum suppression. YOLACT runs at more than 30 fps, with 30% *AP* on *COCO*. In a refined YOLACT++ version that incorporates deformable convolutions [26] and a fast version of the mask re-scoring of Mask Scoring R-CNN, the result *AP* is improved to 34%. Shape priors are also used in ShapeMask [97], mainly with the goal to improve the generalizability of the model and to reduce the amount of necessary annotated training data.

RetinaMask [50] extends the single-stage box detector RetinaNet [120] to instance segmentation by adding a mask branch similar to MRCNN. Thus the model is highly similar to MRCNN, with the only difference that the class prediction is already integrated into the RPN stage. In RetinaMask a second box regression that is typically used in FRCNN or MRCNN is omitted. RetinaMask mainly improves the box prediction of RetinaNet. The mask prediction results are comparable to MRCNN.

## 4.5 HALCON Implementation

In this thesis, we are mainly carrying out our experiments using FRCNN and RetinaNet for object detection, and MRCNN and RetinaMask for instance segmentation. For the results shown in Chapter 5, Chapter 7, and Chapter 11, we initially used the Detectron [56] framework to train and evaluate the networks. However, other ideas, such as the oriented detection of Chapter 8, were implemented in HALCON and even have been released as an official feature of the HALCON software [142]. Moreover, also the more recent results in Chapter 6 to Chapter 10 are implemented in HALCON. To allow a direct and fair comparison of results within this thesis, we have done new experiments using HALCON that achieve consistency across Chapters 5 to 10 in the remainder of this thesis. In the following, we first describe our used HALCON architectures in more detail and then we summarize their differences to the Detectron architectures used in parts of chapters 5, 7, and 11.[8]

### 4.5.1 Architectures

In our implementation, we basically use two different types of methods: The one-stage architecture RetinaNet [120], and the two-stage detector FRCNN [159] (cf. Section 4.3).

In theory, the benefit of a single-stage detector is that the architecture is simpler because the classification of proposals is already incorporated into the RPN. It is straightforward to reduce the model capacity and runtime by reducing the number of kernels or the number of intermediate convolution layers within the class prediction and box regression heads of the network. Since the proposal confidences are obtained by applying a sigmoid to the class activations of every anchor, RetinaNet can predict several classes for each of them. Therefore, typically the number of predictions is a bit higher and this can be reflected in a higher recall.

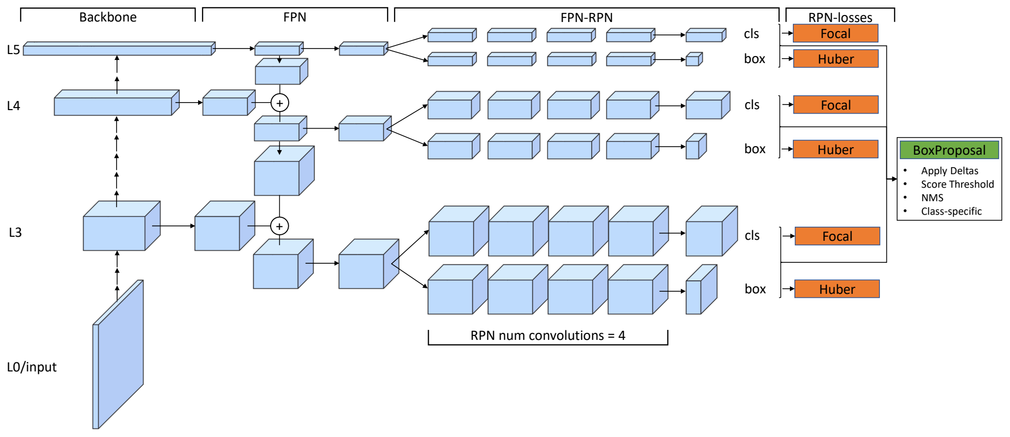
[8]For Chapter 5 and Chapter 7, we also provide HALCON results within a dedicated section. For the more advanced architecture and evaluation in Chapter 11, this goes beyond the scope of this thesis.

However, a two-stage detector has the benefit of a second refinement of the classification and localization: The RPN only generates class-agnostic proposals that are classified or sorted out (classified to the background) inside the class head using box-specific features. Also, a second box regression is added inside the box head, which allows to further refine the proposals. In comparison to RetinaNet, the class prediction in FRCNN is modeled via a softmax applied to the output of the class prediction fully-connected layer. Hence, each proposal is assigned to a unique class. Moreover, the number of anchors is lower because FRCNN only uses a single subscale (or octave) per FPN level. The anchor scale is larger by a factor of two, such that the anchors in level two of FRCNN have the same size as the anchors in level three of RetinaNet. However, the resolution of FPN level two is still higher than in FPN level three, which leads to a higher number of anchors with lower stride.

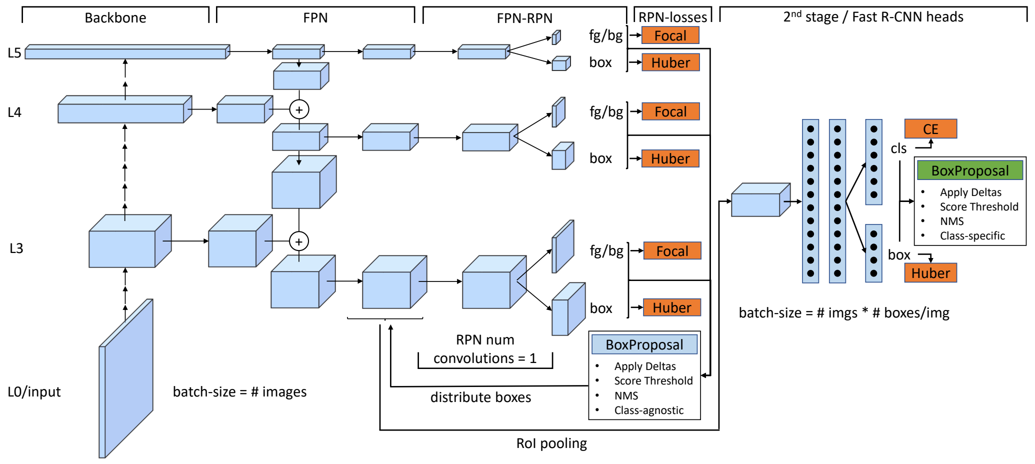
For both models, we use an FPN–RPN as depicted in the middle parts of Fig. 4.17: Inside the RPN, in our implementation, weights are not shared between the levels and only for FRCNN they are shared between the class and box branches. One and four intermediate convolutions are used within the RPN for FRCNN and RetinaNet, respectively. The convolutions have a kernel size of $3 \times 3$ and stride one. An FPN dimension of 256 filters per convolution is used. The box regression is done class-agnostically to reduce the computation time and memory consumption. Note that the depth of the box branch outputs is four times the number of anchor types. For RetinaNet, the depth of the class branches is the number of anchor types times the number of classes, while for FRCNN it equals the number of anchor types. To address the class imbalance between background and foreground anchors, both models use a focal loss for the class outputs instead of random box sampling within the RPN. We use FPN levels two to six for all FRCNN and MRCNN models. For RetinaNet and RetinaMask, we configure FPN levels three to seven because of the smaller anchor scale. Hence, all models have the same underlying anchor box dimensions and only their stride differs.

As can be seen in Fig. 4.17, the RetinaNet architecture can be seen as a class-specific FPN–RPN. While in RetinaNet the cls-outputs directly predict the class probability inside the RPN, in FRCNN the fg/bg-outputs are just deciding for each anchor whether it belongs to the foreground or to the background. The box proposals are then distributed to the FPN levels in order to pool features with a RoI-pooling operation. Those features are the input to an MLP with box regression and class prediction outputs. A softmax cross-entropy loss is used to train the class prediction, while a Huber loss is used to train the box regression. For FRCNN and MRCNN, we use 1024 neurons for each of the hidden MLP fc layers when not otherwise mentioned. The hidden layers are shared for the class and box prediction.

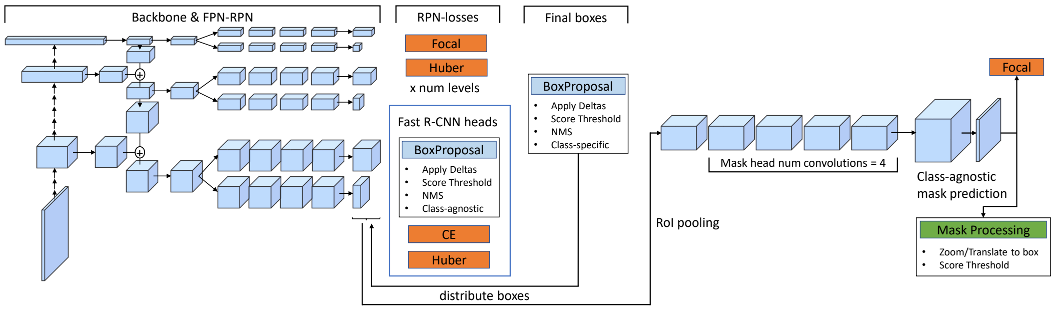
For both architectures, the final box proposal operation is only applied during inference because the losses are applied directly to the class and box prediction outputs. The box proposal operation is applying the predicted box-deltas in order to get the final box coordinates, thresholding the box scores to filter out box predictions with low confidence, applying an NMS (class-specific and class-agnostic), and finally restricting the number of predictions to the given maximum number of detections. For FRCNN and



(a) RetinaNet architecture



(b) FRCNN architecture



(c) Instance segmentation architecture

**Figure 4.17: HALCON detection and instance segmentation architectures.** (*a*) RetinaNet, (*b*) FRCNN, (*c*) RetinaMask based on (*a*), or MRCNN based on (*b*). For convenience, only FPN levels three to five are shown. Multiple arrows within the backbone indicate that there are a high number of intermediate convolutions. 3D boxes indicate the depth and spatial dimension of outputs, boxes with dots in FRCNN show fc-layers. Losses are indicated in orange and only need to be evaluated during training. On outputs that are fed into focal losses, sigmoid activations are applied, while on the class outputs of FRCNN, a softmax is applied. All blue layers are always evaluated, the green box proposal and mask processing operations are only evaluated during inference and can be omitted during training. See the text for a detailed explanation.

MRCNN, an intermediate box proposal operation is necessary to get the proposals for the consecutive RoI pooling operation. Since the RPN is class-agnostic for FRCNN, also the NMS can only be done class-agnostically. To decrease the memory consumption of the following layers and to reduce the runtime of the NMS within the intermediate box proposal operation, we set the parameters *PreNMSTopN* and *PostNMSTopN*. By default, *PreNMSTopN* is set to 1000, such that only the 1000 highest scoring proposals per FPN level are entering the consecutive NMS. Post NMS, only the *PostNMSTopN* = 512 highest scoring boxes per image are used within all consecutive operations. Within the RPN of FRCNN or MRCNN, we set the NMS parameter to 0.9 to filter out very similar duplicate proposals for the same object. The RoI pooling for the Fast R-CNN heads of FRCNN and MRCNN is done with a grid-size of $7 \times 7$.

Note that within the FRCNN and MRCNN architecture, the batch size changes from the first stage of the model until the first box proposal operation to the second stage of the model from the RoI pooling operation: Only *PostNMSTopN* boxes per image are used within the second stage, if so many exist after applying the RPN score threshold and the RPN–NMS. Typically, to ensure that enough boxes reach the second stage to train the corresponding layers, the RPN box proposal score threshold is set rather low during training, e.g., to 0.05. For proposal assignment in the Fast R-CNN heads of FRCNN, we set the parameters *SetWeakBoxesToBg* to *true*, *FgNegThresh* to 0.5, and *FgPosThresh* to 0.7, such that only good proposals are assigned to the foreground. During training, we sample 512 boxes per image randomly, where at most half of them have been assigned to foreground (if that many are available). A loss is applied only for the sampled boxes. To increase the *AP* but stay at reasonable evaluation times, we set the minimum confidence of the output box proposal operation to 0.05.[9] We will see later in Chapter 6 that the maximum *AP* would be achieved for a minimum confidence of 0.0, but in practice this leads to a high number of false postive predictions, which we want to avoid.

For the instance segmentation models MRCNN and RetinaMask, another RoI pooling operation with grid-size $14 \times 14$ is attached to the final box proposals that pools features again from the FPN levels. Since the grid size is twice as large as the grid size of the first RoI pooling operation in each dimension, we decrease the canonical level for FPN level assignment from four to three. Both for RetinaMask and MRCNN, four intermediate convolutions and a bilinear zooming to double the spatial dimensions follow to generate mask specific features. Finally, a $1 \times 1$ convolution predicts the mask logits that are activated with a sigmoid. We use a class-agnostic mask prediction.

To train the mask prediction branch, a focal loss is applied to each pixel of the $28 \times 28$ mask prediction map. Here the main reason to use the focal loss is not the class imbalance between foreground and background, because foreground pixels have usually approximately the same frequency as background pixels. Instead, the focal loss helps to emphasize the loss on those pixels that are difficult to predict.

Mask targets are generated online, with a modified RoI-align operation: For each ground truth instance, the mask is painted into one channel of a binary ground truth mask

[9]This is the score threshold of the first and only box proposal operation for RetinaNet and RetinaMask models, and the second box proposal operation for FRCNN and MRCNN models, respectively.

image. If a final box prediction is assigned to the foreground, the RoI-align operation is applied only to the channel that corresponds to the assigned instance. This has the benefit that the spatial dimension of the target fits exactly to the mask prediction. The mask targets are thresholded at 0.5 such that they are binary and can be used within the mask sigmoid focal loss. For final boxes that are assigned to the background, the mask prediction is ignored by setting the pixel-wise loss weights to zero. For proposal assignment in the mask head of MRCNN and RetinaMask, we use the same parameter settings as in the Fast R-CNN head proposal assignment, but increase *FgNegThresh* to 0.6 such that only good proposals contribute to the mask head training. To ensure, that the mask branch generates a meaningful loss just from the beginning of the training, we append the ground truth boxes to the predictions in the training phase. This ensures that at least the ground truth boxes are assigned to the foreground and a mask loss is calculated for them.

During inference, the mask predictions are translated and zoomed according to the corresponding box coordinates. Finally, a binary thresholding at 0.5 is applied to infer the mask region.

#### Differences to Detectron Models

Major parts of the HALCON architectures described above are similar to the implementation of Detectron [56], but some details and hyperparameters have been changed. The main differences to the architectures and hyperparameters of the Detectron framework are summarized again in the following:

- Focal losses in RPN class head and mask head: In HALCON, we use the more modern focal loss [120] (see above) instead of the sigmoid cross-entropy loss used in Detectron.

- Class-agnostic box regression and mask prediction: In HALCON, we did not find that a class-specific box regression or mask prediction improves the results. However, it leads to a significantly higher memory consumption and runtime. Detectron uses class-specific box regression and mask prediction per default.

- FPN–RPN weight sharing: In contrast to Detectron, in HALCON we do not share the weights between different FPN levels.

- Anchor assignment: Per default, we set *SetWeakBoxesToBg* to *false* (cf. Section 4.3).

- Foreground/background assignment in RPN, Fast R-CNN, and mask heads: We use different *FgNegThresh* and *FgPosThresh* values for assignement of proposals in the first, second, and third stage of the model (see above).

- Mask head RoI canonical level: We adjust the RoI canonical level to three to account for the larger grid size of the mask RoI pooling. Detectron uses the same canonical level (four) for the Fast R-CNN heads and the mask head.

- NMS parameters: We use lower *PreNMSTopN* (1000) and *PostNMSTopN* (512) values to speed up the NMS computations and reduce the batch size of subsequent layers. We also adjust the maximum number of detections, as well as the class-specific and class-agnostic NMS IoU thresholds. In comparison to Detectron, we use both a class-specific and a class-agnostic NMS sequentially (in this order).
- Image resolution: To highlight the influence of the image resolution, in HALCON we carry out experiments with images of size $512 \times 384$, $768 \times 512$, and $1024 \times 768$. We generally use an image size of $512 \times 384$ to reduce the runtime and allow to perform the experiments in a reasonable time. Using Detectron, we usually rescale the shorter image size to 800, leading to a resolution of $1066 \times 800$ for images with an aspect ratio of 4:3.
- Mirroring: While in the Detectron results, we only use horizontal reflections of the images, in HALCON, we use both horizontal and vertical reflections of the images. Therefore, during training, with 50% probability, the images are randomly flipped at either the column, row, or both axes.
- Batch size: In HALCON, we use a batch size of 2, unless otherwise mentioned. For the Detectron results in Chapter 5, we used a batch size of 8. On *D2S* (cf. Chapter 5), we did not find that an increased batch size helped to improve the results. With this low batch size the memory consumption is reduced, such that the models fit on a single GPU.

# 5 D2S: Densely Segmented Supermarket Dataset

In this chapter, we introduce the Densely Segmented Supermarket (*D2S*) dataset, a novel benchmark for instance segmentation and object detection in an industrial domain. It contains 21 000 high-resolution images with pixel-wise labels of all object instances. The objects comprise groceries and everyday products from 60 categories. The benchmark is designed such that it resembles the real-world setting of an automatic checkout, inventory, or warehouse system. The training images only contain objects of a single class on a homogeneous background, while the validation and test sets are much more complex and diverse. To further benchmark the robustness of detection methods, the scenes are acquired with different lightings, rotations, and backgrounds. We ensure that there are no ambiguities in the labels and that every instance is labeled comprehensively. The annotations are pixel-precise and allow using crops of single instances for articial data augmentation. The dataset covers several challenges highly relevant in the field, such as a limited amount of training data and a high diversity in the test and validation sets. The evaluation of state-of-the-art object detection and instance segmentation methods on D2S reveals significant room for improvement for current methods. The high-quality pixel-precise annotations allow a proper evaluation with high IoU thresholds.

*D2S* was published in [46]. In comparison to the publication, we add baselines with models implemented in HALCON [143] in Section 5.6. Moreover, we do an extensive evaluation with respect to failure cases in Section 5.7. These failure cases motivate the work of the following chapters.

## 5.1 Introduction

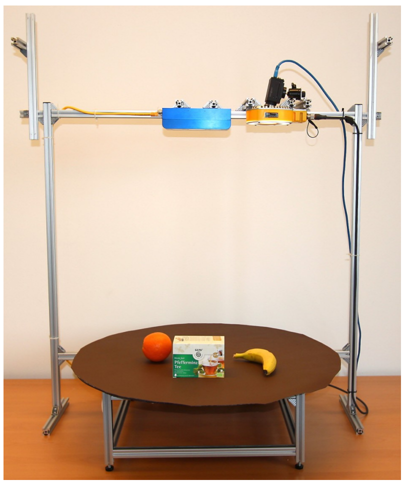
All top-performing methods in common instance segmentation and object detection challenges are based on deep learning and require a large amount of annotated training data. Accordingly, the availability of large-scale datasets, such as *ADE20K* [211], *Cityscapes* [22], *ImageNet* [168], *KITTI* [53], *COCO* [118], *Mapillary Vistas* [145], *VOC* [41], *Places* [210], *The Plant Phenotyping Datasets* [136], or *Youtube-8M* [1], is of paramount importance.

Most of the above datasets focus on everyday photography or urban street scenes, which makes them of limited use for many industrial applications. Furthermore, the amount and diversity of labeled training data is usually much lower in industrial settings. To train a visual warehouse system, for instance, the user typically only has a few images of each product in a fixed setting. Nevertheless, at runtime, the products must be robustly detected in very diverse settings. Only few datasets focus on industry-relevant challenges in the context of warehouses. The Freiburg Groceries Dataset [87], SOIL-47 [94], and the Supermarket Produce Dataset [160] contain images of supermarket products, but only provide class annotations on image level, and hence no segmentation. The Grocery Products Dataset [54] and GroZi-120 [135] include bounding box annotations that can be used for object detection. However, not all object instances in the images are labeled separately. To the best of our knowledge, none of the existing industrial datasets provides pixel-wise annotations on instance level.[1] In this chapter, we introduce the *Densely Segmented Supermarket (D2S) dataset*, which satisfies the industrial requirements described in Chapter 1. The training, validation, and test sets are explicitly designed to resemble the real-world applications of automatic checkout, inventory, or warehouse systems.

**Contributions.** We present a novel instance segmentation dataset with high-resolution images in a real-world, industrial setting. The annotations for the 60 different object categories were obtained in a meticulous labeling process and are of very high quality. Specific care was taken to ensure that every occurring instance is labeled comprehensively. We show that the high-quality region annotations of the training set can easily be used for artificial data augmentation. Using both the original training data and the augmented data leads to a significant improvement of the average precision ($AP$) on the test set by about 30 percentage points. In contrast to existing datasets, our setup and the choice of the objects ensures that there is no ambiguity in the labels and an $AP$ of 100% is achievable by an algorithm that performs flawlessly. To evaluate the generalizability of methods, the training set is considerably smaller than the validation and test sets and contains mainly images that show instances of a single category on a homogeneous background. Overall, the dataset serves as a demanding benchmark and resembles real-world applications and their challenges. The dataset is publicly available.[2]

We provide a thorough evaluation of the baseline result and, in particular, point out different types of failure cases for object detection and instance segmentation models on *D2S*. These failure cases are addressed in subsequent chapters.

[1]To be more precise, we refer to none of the published datasets before the *D2S* publication [46] in 2018.
[2]`https://www.mvtec.com/research`



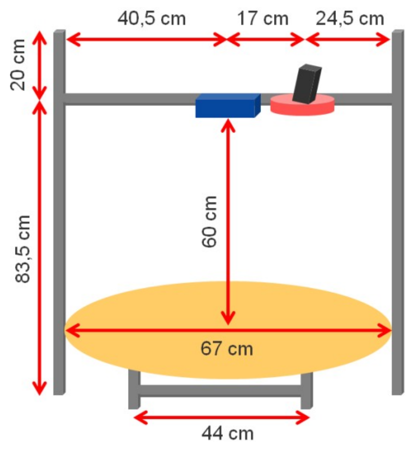


**Figure 5.1:** ***D2S* image acquisition setup.** Each scene was rotated ten times using a turntable. For each rotation, three images under different illuminations were acquired.

## 5.2 The Densely Segmented Supermarket Dataset

The overall target of the dataset is to realistically cover the real-world applications of an automatic checkout, inventory, or warehouse system. For example, existing automatic checkout systems in supermarkets identify isolated products that are conveyed on a belt through a scanning tunnel.[3][4] Even though such systems often provide a semi-controlled environment, external influences (e.g., lighting changes) cannot be completely avoided. Furthermore, the system's efficiency is higher if non-isolated products can be identified as well. Consequently, methods should be able to segment also partly occluded objects. Also, the background behind the products is not constant in many applications because of different types of storage racks in a warehouse system or because of dirt on the conveyer belt of a checkout system in the supermarket, for example.

For the *D2S* dataset, we acquired a total of 21 000 images in 700 different scenes with various backgrounds, clutter objects, and occlusion levels. In order to obtain systematic test settings and to reduce the amount of manual work, a part of the image acquisition process was automated. Therefore, each scene was rotated ten times with a fixed angle step and acquired under three different illuminations.

**Setup.** The setup used for the image acquisition is depicted in Fig. 5.1. A high-resolution industrial color camera with $1920 \times 1440$ pixels was mounted above a turntable. The camera was intentionally mounted off-center with respect to the rotation center of the turntable to introduce more variations in the rotated images.

**Objects.** An overview of the 60 different classes is shown in Fig. 5.2. The object categories cover a selection of common, everyday products such as fruits, vegetables,

[3]ECRS. RAPTOR. https://www.ecrs.com/retail-pos/hardware/accelerated-checkout/, accessed 2018-03-07.

[4]ITAB. HyperFLOW. https://itab.com/en/itab/checkout/self-checkouts/, accessed 2018-03-07.

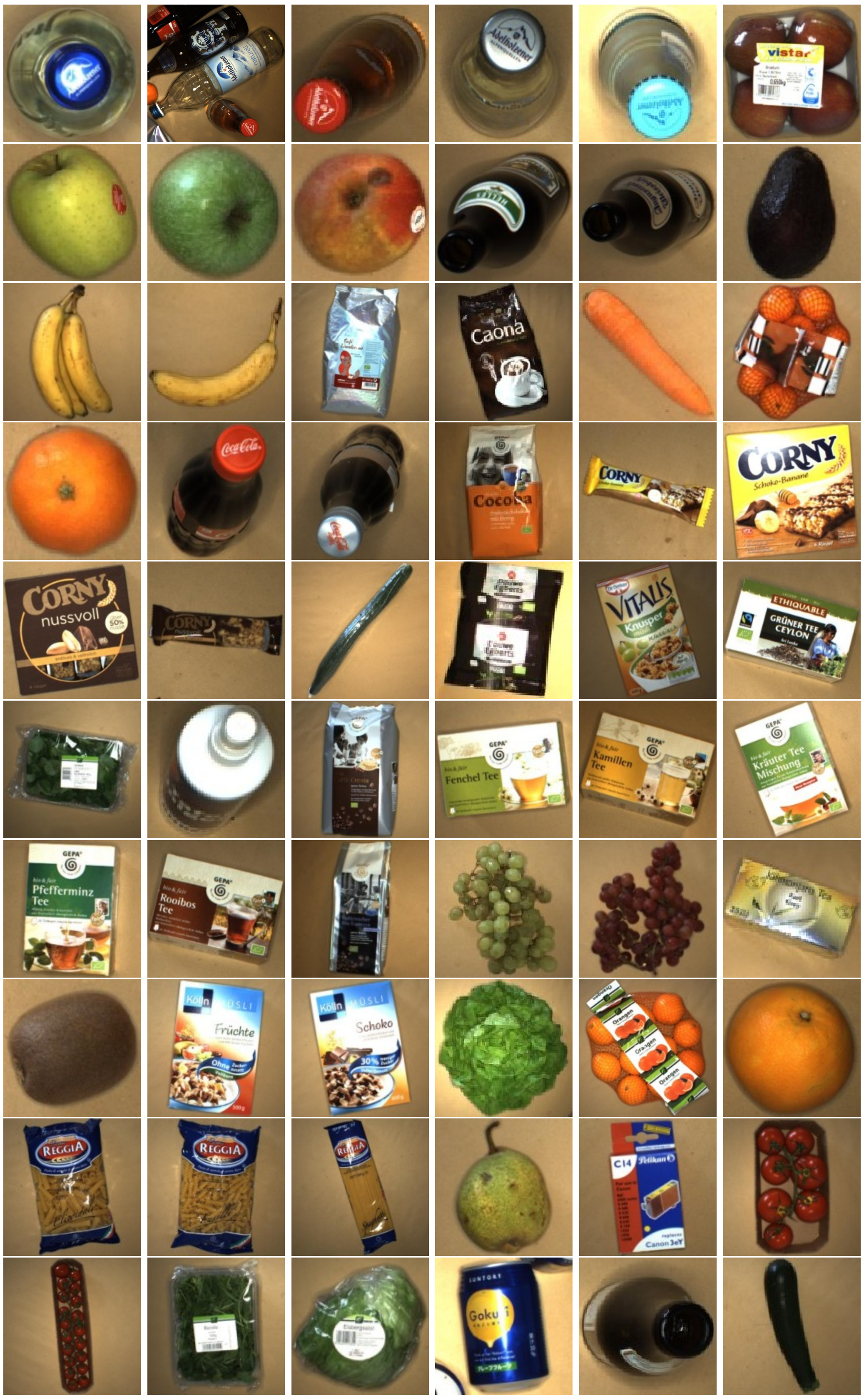


**Figure 5.2: *D2S* categories.** Overview of the 60 different classes within the *D2S* dataset. The second image in the first row shows an example of a lying bottle that must be stabilized by surrounding objects to prevent it from rolling off the turntable.

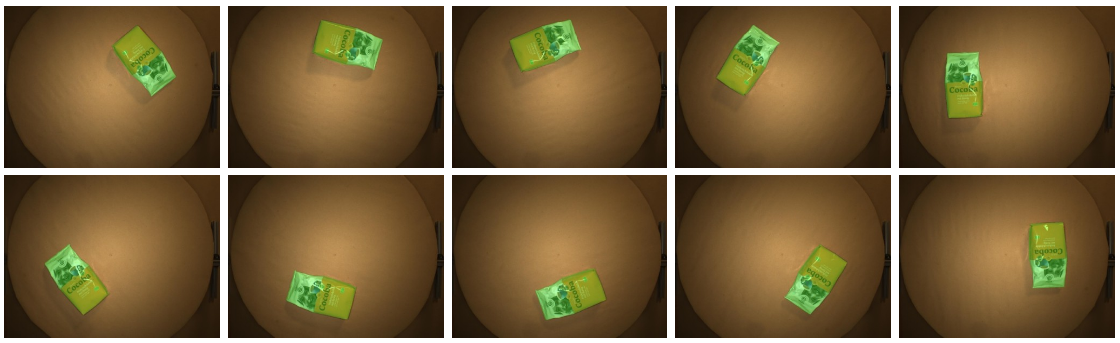


**Figure 5.3: Rotation example.** Each scene was acquired at ten different rotations in steps of 36°. The camera was mounted off-center in order to introduce more variation in the images.

cereal packets, pasta, and bottles. They are embedded into a class hierarchy tree. The branches of the tree split the classes into groups of different packaging. This results in neighboring leafs being visually very similar, while distant nodes are visually more different, even if they are semantically similar products, e.g., single apples in comparison to a bundle of apples in a cardboard tray. The class hierarchy can be used, for instance, for advanced training and evaluation strategies similar to those used by YOLO9000 [156]. However, it is not used within the scope of this work.

**Rotations.** Each scene was rotated ten times in increments of 36°. The turntable allowed to automate this process and to ensure that the rotation angles are precise. An example of the ten rotations for a scene from the training set is displayed in Fig. 5.3.

**Lighting.** To evaluate the robustness of methods to illumination changes and different amounts of reflection, each scene and rotation was acquired under three different lighting settings. For this purpose, an LED ring light was attached to the camera. The illumination was set to span a large spectrum of possible lightings. Hence, as displayed in Fig. 5.4 (*top*), the dark images were underexposed and the bright images overexposed.

**Background.** The validation and test scenes have a variety of different backgrounds that are shown in Fig. 5.4 (*bottom*). This allows to evaluate the generalizability of approaches. In contrast, the training set is restricted to images with a single homogeneous background. It is kept constant to imitate the settings of a warehouse system, where new products are mostly imaged within a fixed environment and not in the test scenario.

**Occlusion and clutter.** As indicated in Fig. 5.5, occlusions in the dataset may arise from objects of the same class, objects of a different class, or from clutter objects. Clutter objects have a category that is not present in the training images. They were added explicitly to the validation and test images to evaluate the robustness of methods to novel objects. Examples of the selected clutter objects are shown in Fig. 5.6.

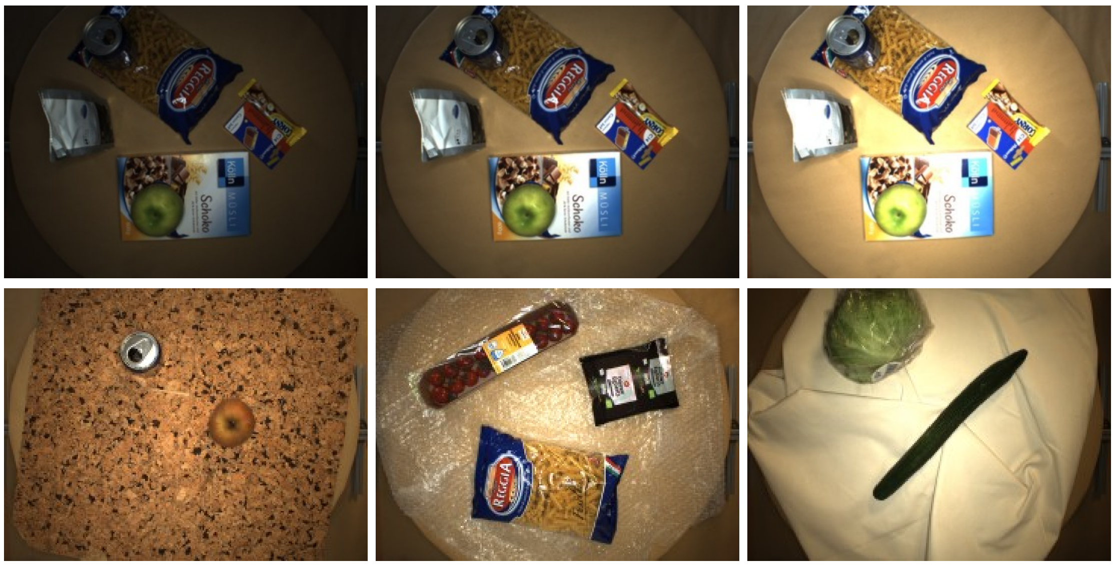


**Figure 5.4: Lighting and background examples.** (*Top*) Each scene was acquired under three different lightings. (*Bottom*) As opposed to the training set (where a single uniform background is used), the test and validation sets include three additional backgrounds. This allows for a detailed evaluation of the robustness of the methods.

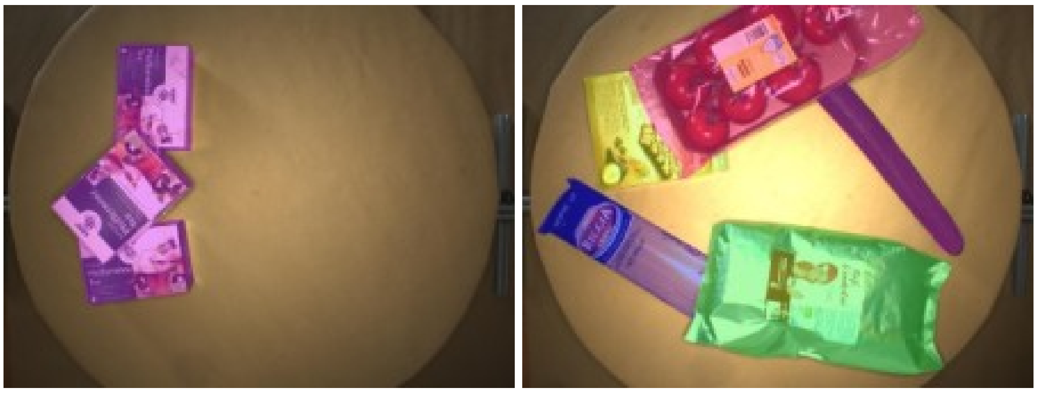


**Figure 5.5: Occlusion.** Objects appear with different amounts of occlusion. These may either be caused by objects of the same class, objects of a different class, or by clutter objects not within the training set.

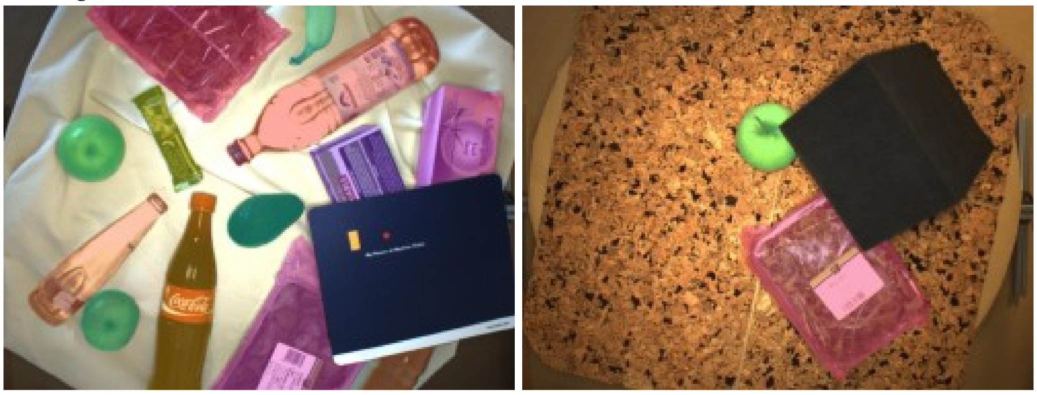


**Figure 5.6: Clutter.** To test the robustness of approaches to unseen clutter objects, objects not within the training set were added to the validation and test sets (e.g., a mouse pad and a black foam block).

## 5.3 Dataset Splitting

In contrast to existing datasets for instance-aware semantic segmentation, such as *VOC* [41] and *COCO* [118], the *D2S* training set has a different distribution with respect to image and class statistics than the validation and test sets. The complexity of the captured scenes as well as the average number of objects per image are substantially higher in the validation and test sets (see Table 5.1). The motivation for this choice of split is to follow common industrial requirements, such as: low labelling effort, low complexity of training set acquisition for easy replicability, and the possibility to easily add new classes to the system.

The split is performed on a per-scene basis: all 30 images of a scene, i.e., all combinations of the ten rotations and three lightings, are included in either the training, the validation, or the test set. In the following, we describe the rules for generating the splits.

**Training split.** To meet the industrial requirements mentioned above, the training scenes are selected to be as simple as possible. Hence, the scenes have a homogeneous background and mostly contain only one object. Furthermore, the amount of occlusions is reduced to a minimum. To summarize, we add scenes to the training split that

- contain only objects of one category,[5]
- provide new views of an object,
- only contain objects with no or marginal overlap, and
- have no clutter and a homogeneous background.

The total number of scenes in the training set is 147, resulting in 4380 images of 6900 objects. The rather small training set should encourage work towards the generation of augmented or synthetic training data, for instance, using generative adversarial networks [64, 78, 113, 172, 207].

**Validation and test splits.** The remaining scenes are split between the validation and the test set. They consist of scenes with

- single or multiple objects of different classes,
- touching or occluded objects,
- clutter objects, and
- varying background.

These scenes were chosen such that the generalization capabilities of approaches can be evaluated. Additionally, current methods struggle with heavy occlusion and clutter objects that are not present within the training set. These issues are addressed by this choice of splits as well. The split between validation and test set was performed on

[5]In order to provide similar views of each object class as they are visible in the validation and test set, four scenes were added to the training set that contain two distinct classes.

| split | *all* | *train* | *val* | *test* |
|---|---|---|---|---|
| # scenes | 700 | 146 | 120 | 434 |
| # images | 21000 | 4380 | 3600 | 13020 |
| # objects | 72447 | 6900 | 15654 | 49893 |
| # objects/image | 3.45 | 1.58 | 4.35 | 3.83 |
| # scenes w. occlusion | 393 | 10 | 84 | 299 |
| # scenes w. clutter | 86 | 0 | 18 | 68 |
| rotations | | ✓ | ✓ | ✓ |
| lighting variation | | ✓ | ✓ | ✓ |
| background variation | | | ✓ | ✓ |
| clutter | | | ✓ | ✓ |

**Table 5.1: Split statistics.** Due to our splitting strategy, the number of images and the number of instances per image is significantly lower for the training set. The complexity of validation and test scenes is approximately the same.

subgroups of images containing the same number of total and occluded objects. This ensures that both sets have approximately the same distribution. The ratio of the number of scenes in the validation and test set is chosen to be 1:4. The reasons for this decision are twofold: First, the evaluation of the model on a small validation set is faster. Second, we do not want to encourage training on the validation set, but stimulate work on approaches that require little training data or use augmentation techniques. The statistics of the number of images and objects in the splits are visualized in Table 5.1.

## 5.4 Statistics and Comparison

In this section, we compare our dataset to *VOC* [41] and *COCO* [118]. These datasets have encouraged many researchers to work on instance segmentation and are frequently used to benchmark state-of-the-art methods.

**Dataset statistics.** As summarized in Table 5.2, *D2S* contains significantly more object instances than *VOC*, but fewer than *COCO*. Specifically, although the *D2S* training set is larger than that of *VOC*, the number of training objects is less than 1% of those in *COCO*. This choice was made intentionally, as in many industrial applications it is desired to use as few training images as possible. In contrast, the proportion of validation images is significantly larger for *D2S* in order to enable a thorough evaluation of the generalization capabilities. On average, there are half as many objects per image in *D2S* as in *COCO*. However, for *COCO* this number is heavily biased since there is an disproportionately large number of instances from the class *person* throughout the whole dataset, which increases the number of objects per image dramatically (see also Fig. 5.7).

| Dataset | | *VOC* | *COCO* | *D2S* |
|---|---|---|---|---|
| # images | all | 4369 | 163957 | 21000 |
| | `train` | 1464 | 118287 | 4380 |
| | `val` | 1449 | 5000 | 3600 |
| | `test` | 1456 | 40670 | 13020 |
| # objects | all | - | - | 72447 |
| | `train` | 3507 | 849941 | 6900 |
| | `val` | 3422 | 36335 | 15654 |
| | `test` | - | - | 49893 |
| # obj/img | | 2.38* | 7.19* | 3.45 |
| # classes | | 20 | 80 | 60 |

**Table 5.2: Dataset statistics.** Number of images and objects per split, average number of objects per image and number of classes for *D2S* (ours), *VOC 2012*, and *COCO*. *For *VOC 2012* and *COCO*, the object numbers are only available for the training and validation set.

**Class statistics.** Since the images of *COCO* and *VOC* were taken from flickr, the distribution of object classes is not uniform. In both datasets, the class *person* dominates, as visualized in Fig. 5.7: 31% and 25% of all objects belong to this class for *COCO* and *VOC*, respectively. Moreover, 10% of the classes with the highest number of objects are represented by 51% and 33% of all objects, while only 5.4% and 13.5% of the objects belong to the 25% of classes with the lowest number of objects. This class imbalance is valid since both *COCO* and *VOC* represent the real world, where some classes naturally appear more often than others. In the evaluation, all classes are weighted uniformly. Therefore, the class imbalance inherently poses a challenge to learn all classes equally well, independent from the number of training samples. For example, the *COCO 2017* validation set contains nine instances of the class *toaster*, but 10 777 instances of *person*. Nevertheless, both categories are equally weighted in the calculation of the *AP*, which is the metric used for ranking the methods in the *COCO* segmentation challenge.

There is no such class imbalance in *D2S*. In the controlled environment of the supermarket scenario, all classes have the same probability to appear in an image. The class with the highest number of objects is represented by only 2.7% of all objects. Only 14% of the objects represent the 10% of classes with the highest number of objects, while 19% of the objects are from the 25% of classes with the lowest number of objects. The class distribution of *D2S* is visualized in Fig. 5.8, where the number of images per class is shown in total and for each split. As mentioned in the previous section, the number of images for each class is rather low in the training split. This is especially the case for classes that have a similar appearance for different views, such as *kiwi* and *orange_single*. Note that, although the split choice between validation and test set is not made on the class level, each class is well represented in both sets. The key challenge of the *D2S* dataset is thus not the handling of underrepresented classes, but the low amount of training data.

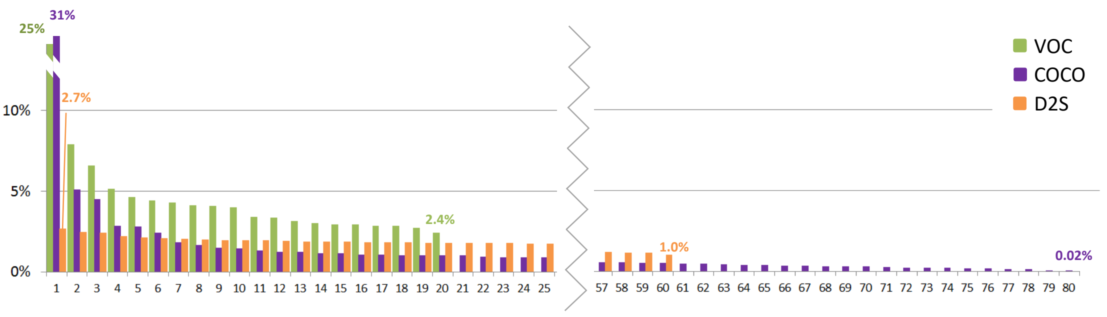


**Figure 5.7: Objects per class.** Ratio of objects per class for *D2S* (*orange*), *VOC* (*green*) and *COCO* (*violet*). In *COCO* and *VOC*, the class *person* is dominant and some classes are underrepresented. In *D2S*, the number of objects per class is uniformly distributed. Note that for *COCO* and *VOC* the diagram was calculated based on train and val splits.

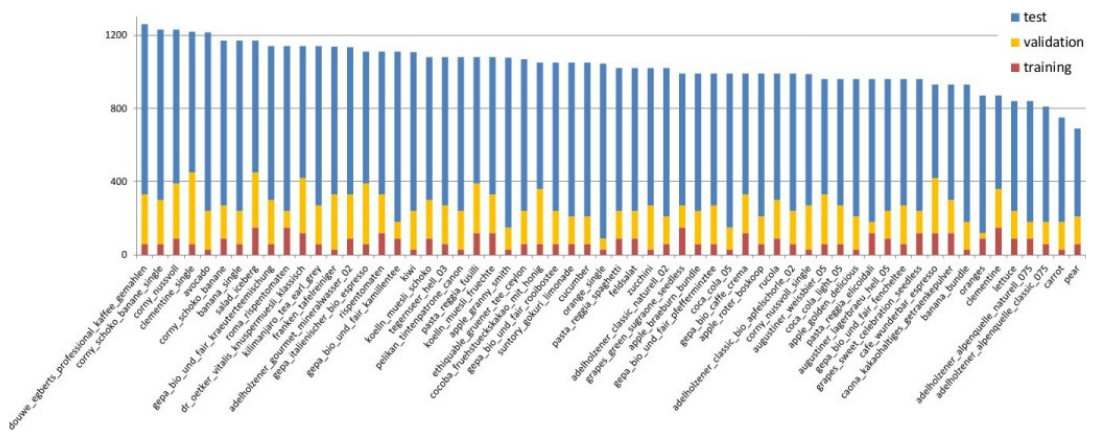


**Figure 5.8: *D2S* Images per class.** Number of images per class and split sorted by the total number of images per class for *D2S*. The number of images per class is almost uniformly distributed.

**Label consistency.** It is difficult to ensure that all object instances in large real-world datasets are labeled consistently. On the one hand, it is hard to establish a reliable review process for the labeling of large datasets, e.g., to avoid unlabeled objects. On the other hand, some labels are ambiguous by nature, for instance a painting of a person. Fig. 5.9 shows examples for label inconsistencies from *ADE20K* [211], *VOC*, and *COCO*.

In *D2S*, the object classes are unambiguous and have been labeled by six expert annotators. We ensured that all present objects are annotated with high quality labels. A perfect algorithm that flawlessly detects and segments every object in all images of the *D2S* dataset will achieve an *AP* of 100%. This is not the case for *COCO*, *VOC*, and *ADE20K*. In these datasets, if an algorithm correctly detects one of the objects that is not labeled, the missing ground truth leads to a false positive. Furthermore, if such an object is not found by an algorithm, the resulting false negative is not accounted for. As algorithms improve, this might prevent better algorithms from obtaining higher scores in the benchmarks. It should be noted that in *COCO*, this problem is addressed using *crowd annotations*, i.e., regions containing many objects of the same class that are ignored in the evaluation. However, crowd annotations are not present in all cases.

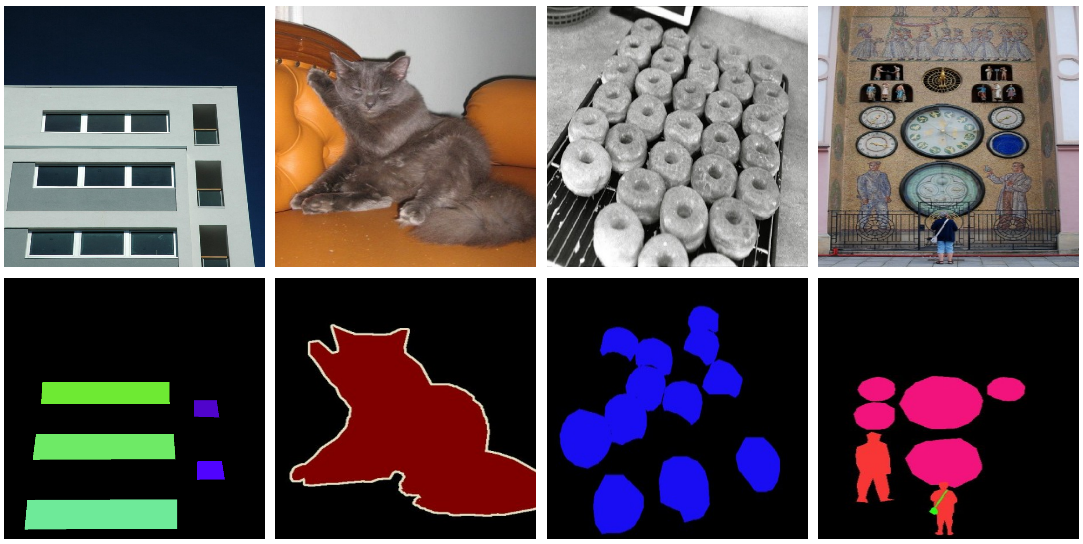


**Figure 5.9: Inconsistent labels in large-scale datasets.** Images (*top*) and labels (*bottom*). Large real-world datasets are extremely difficult to label consistently. In the examples from *ADE20K*, *VOC*, and *COCO*, some labels are missing (*from left to right*): a window, the sofa, some donuts, and the painting of a person.

## 5.5 Benchmark

In this section, we provide first benchmark results for our dataset. We evaluate the performance of state-of-the-art methods for object detection [159, 120] and instance segmentation [70, 116]. We experiment with various training sets, which differ in the number of rotations and the availability of under- and overexposed images. Furthermore, we evaluate a simple approach for augmenting the training data artificially.

### 5.5.1 Evaluated Methods

**Object detection.** For the object detection task, we evaluate the performance of Faster R-CNN (FRCNN) [159] and RetinaNet [120]. In this section, we use the official implementations of both methods, which are provided in the Detectron [56] framework. Both methods use a ResNet-101 [68] backbone with FPN [119].

**Instance segmentation.** For the instance segmentation task, we evaluate the performance of Mask R-CNN (MRCNN) [70] and FCIS [116]. In this section, we use the official implementation of MRCNN in the Detectron framework and the official implementation of FCIS provided by the authors.[6] MRCNN uses a ResNet-101 with FPN as backbone, while FCIS uses a plain ResNet-101. Since both methods output boxes in addition to the segmentation masks, we also include them in the object detection evaluation.

We provide more results of our own HALCON implementation for RetinaNet, FRCNN, and MRCNN, as well as RetinaMask [50] in Section 5.6.

[6]https://github.com/msracver/FCIS

**Training.** All methods are trained end-to-end. The network weights are initialized with the COCO-pretrained models provided by the respective authors. The input images are resized to have a shorter side of 800 pixels (600 pixels for FCIS, respectively). All methods use horizontal flipping of the images at training time. FCIS uses *online hard example mining* [171] during training.

### 5.5.2 Evaluation Metric

The standard metric used for object detection and instance segmentation is *mean average precision* ($mAP$) [41]. It is used, for instance, for the ranking of state-of-the-art methods in the *COCO* segmentation challenge [118]. We compute the $mAP$ exactly as in the official COCO evaluation tool[7] and give its value in percentage points. The $mAP$ is explained in Section 4.2. The $mAP$ is the mean over $AP$s of all classes in the dataset. In the following, we just use the abbreviation $AP$ for the value averaged over IoUs and classes. When referring to class-averaged $AP$ for a specific IoU threshold, e.g., 0.5, we write $AP_{50}$.

Due to the findings of this chapter, we will analyze the $AP$ measure in detail in Chapter 6. There, we also propose certain changes to the evaluation protocol that help to identify models with fewer false positive predictions.

### 5.5.3 Data Augmentation

In order to keep the labeling effort low and still achieve good results, it is crucial to artificially augment the existing training set such that it can be used to train deep neural networks. Hence, we experiment with a simple data augmentation technique, which serves as baseline for more sophisticated approaches. In particular, we simulate the distribution of validation and test set using only the annotations of the training set. For this purpose, we assemble 10 000 new artificial images that contain one to fifteen objects randomly picked from the training split. We denote the augmented data as *aug* in Table 5.3. For each generated image, we randomly sample the lighting and number of object instances. For each instance, we randomly sample its class, the orientation, and the location in the image. The background of these images is the plain turntable. We make sure that the instances' region centers lie on the turntable and that occluded objects have a visible area larger than 5000 pixels. Fig. 5.10 shows example images of the artificially augmented dataset for all three different lightings. Due to the high-quality annotations without margins around the object border, the artificially assembled images have an appearance that is very similar to the original test and validation images.

### 5.5.4 Results

When trained on the full training set *train* and evaluated on the *test* set, the instance segmentation methods provide solid baseline $AP$s of 49.5% (MRCNN) and 45.6% (FCIS). The object detection results are on a similar level, with $AP$s of 46.5% (MRCNN), 44.0% (FCIS), 46.1% (FRCNN), and 51.0% (RetinaNet). Tables 5.3 and 5.4 show the quantitative

[7] `https://github.com/cocodataset/cocoapi`

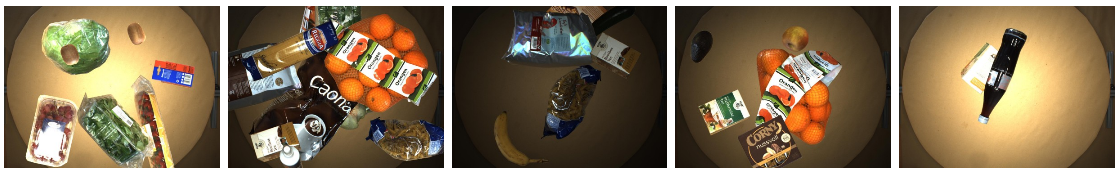


**Figure 5.10: Artificial images.** The artificial augmented training set is generated by randomly assembling objects from the basic training set.

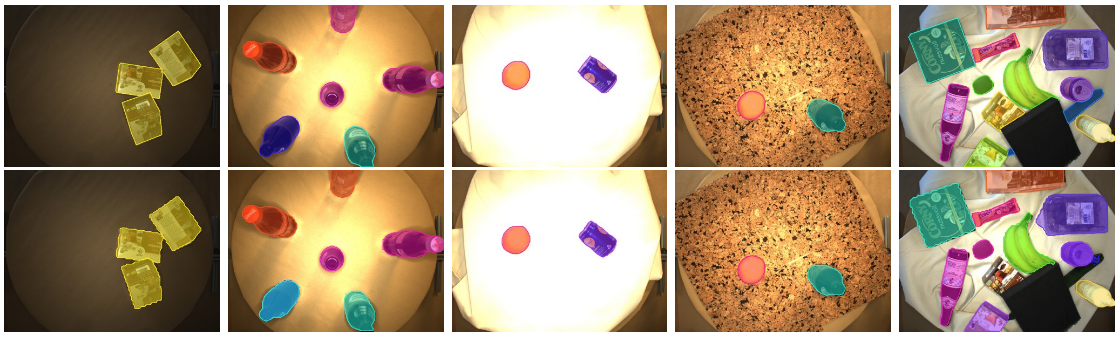


**Figure 5.11: Qualitative results.** (*Top*) Ground truth annotations from the *D2S val* and *test* sets. (*Bottom*) Results of MRCNN trained on the *train* set. The classes are indicated by colors.

results in full detail. We show qualitative results of the best performing MRCNN in Fig. 5.11. More qualitative results of the Detectron MRCNN implementation trained on the different *D2S* training splits are shown in Appendix B.

**Ablation study.** As mentioned previously, the *D2S* splits are based on scenes, i.e., all rotations and lightings for one placement of objects are included in the same split. In order to evaluate the importance of these variations and the ability of the instance segmentation methods to learn invariance with respect to rotations and illumination, we perform an ablation study. For this purpose, we create three subsets of the full training set *train*. The *train rot0* set contains all three lightings, but only the first rotation of each scene. The *train light0* set contains only the default lighting, but all ten rotations of each scene. The *train rot0 light0* set contains only the default lighting and the first rotation for each scene.

The resulting *AP* values of the instance segmentation methods MRCNN and FCIS are summarized in Table 5.3 (*top*). As expected, we obtain the best results when training on the full *train* set. Training only on the first rotation reduces the *AP* on the test set by 15.7% and 9.1% for MRCNN and FCIS, respectively. Training only with default lighting reduces the *AP* slightly by 3.4% for MRCNN and increases the *AP* by a neglible 0.4% for FCIS. Training on *train rot0 light0* reduces the *AP* by 13.2% and 12.9%, respectively. Overall, the results indicate that the models are more invariant to changes in lighting than to rotations of the objects. Why MRCNN performs better when trained on *train rot0 light0* than when trained on *train rot0* remains unclear.

**Data augmentation.** As shown in Table 5.3, training on the augmented dataset *aug* boosts the *AP* on the test set to 76.1% and 69.8% for MRCNN and FCIS, respectively.

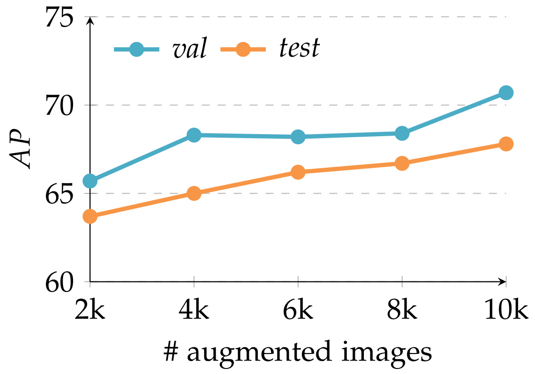


**Figure 5.12: Data augmentation.** Influence of number of artificially generated training images on $AP$ for validation and test set.

| | MRCNN | | | FCIS | | |
|---|---|---|---|---|---|---|
| | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP$ | $AP_{50}$ | $AP_{75}$ |
| *train* | 49.5 | 57.6 | 51.3 | 45.6 | 58.3 | 51.3 |
| *train rot0* | 33.8 | 41.6 | 35.6 | 36.5 | 47.5 | 41.8 |
| *train light0* | 46.1 | 54.8 | 48.0 | 46.0 | 59.3 | 52.0 |
| *train rot0 light0* | 36.3 | 45.1 | 38.6 | 32.7 | 43.4 | 38.1 |
| *aug* | 71.6 | 86.9 | 81.7 | 69.8 | 87.6 | 82.4 |
| *train+aug* | **79.9** | **89.1** | **85.3** | **72.5** | **88.1** | **83.5** |

**Table 5.3: Instance segmentation benchmark results on the test set.** (*Top*) Training on different subsets of the *train* set. (*Bottom*) Training on augmented data yields the highest $AP$.

This is significantly higher than the 49.5% and 45.6% achieved by training on the original *train* set. Combining the sets *train* and *aug* to *train+aug* further improves the $AP$ by 8.3% and 2.7%, respectively.

The *aug* split consists of 10 000 artificially generated images. In order to evaluate the influence of the number of training images, we used subsets of 2000, 4000, 6000, and 8000 images of *aug*, respectively, to train FCIS pretrained on ImageNet. The resulting $AP$ values for both validation and test set are shown in Fig. 5.12. It can be seen that using more images results in better performance. However, already with 2000 generated images the $AP$ increases by over 20 percentage points compared to using *train*.

**Object detection.** We conduct the same ablation study for the task of object detection. The resulting $AP$ values for all training splits of the methods FRCNN and RetinaNet, as well as the results of instance segmentation methods MRCNN and FCIS evaluated on bounding box level, are summarized in Table 5.4. It is interesting to note that these $AP$ values are not always better than the $AP$ values obtained for the more difficult task of instance segmentation. For all methods, the overall performance is very similar. Reducing the training set to only one rotation or only one lighting per scene results in worse performance. Analogously, augmenting the dataset by generating artificial training images results in a strong improvement.

| | MRCNN | | | FCIS | | | FRCNN | | | RetinaNet | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP$ | $AP_{50}$ | $AP_{75}$ | $AP$ | $AP_{50}$ | $AP_{75}$ |
| *train* | 46.5 | 58.3 | 53.5 | 44.0 | 59.4 | 51.7 | 46.1 | 55.2 | 49.7 | 51.0 | 61.0 | 52.8 |
| *train rot0* | 34.1 | 42.5 | 38.3 | 34.6 | 48.2 | 41.3 | 36.7 | 46.9 | 41.5 | 32.9 | 39.8 | 34.5 |
| *train light0* | 45.5 | 55.7 | 49.5 | 44.0 | 60.3 | 51.9 | 43.7 | 53.9 | 47.8 | 51.7 | 62.0 | 53.6 |
| *train rot0 light0* | 35.7 | 46.0 | 40.5 | 29.9 | 43.9 | 35.4 | 34.3 | 44.3 | 39.0 | 31.6 | 38.9 | 33.2 |
| *aug* | 72.9 | 87.9 | 82.0 | **69.9** | 88.1 | 80.7 | 73.5 | 88.4 | 82.2 | 74.2 | 86.9 | 81.4 |
| *train+aug* | **78.3** | **89.8** | **84.9** | 68.3 | **88.5** | **80.9** | **78.0** | **90.3** | **84.8** | **80.1** | **89.6** | **84.5** |

**Table 5.4: Object detection benchmark results on the test set.** Mean average precision values for models trained on different training sets.

## 5.6 HALCON Baselines

In this section, we re-evaluate the performance of object detection and instance segmentation models on *D2S* using our own implementations in HALCON [142] (cf. Section 4.5). In particular, this allows a fair comparison with more recent results in the following chapters that are also obtained with HALCON. Moreover, we compare the influence of using different backbone networks and the size of input images. The HALCON implementations outperform their Detectron counterparts of the previous section and thus can be used as a strong baseline. We provide a detailed analysis of the baseline results before we look at failure cases and future work in the next section.

To get a feeling for the influence of choosing one or the other architecture and the effects of some hyperparameters, we evaluate a number of models on D2S both for object detection and instance segmentation. For all models, we train on the *train + aug* split. Using the ground truth annotations of 1000 random samples from the validation set, we determine the maximum overlap between objects of any class as 0.42 and between objects of the same class as 0.41. In order to account for inaccurate results, we set the NMS parameters to 0.5, both for the class-agnostic and class-specific case. We set the maximum number of detections to 50 since there are at most 15 objects within one image and we want to avoid an overly high number of false positives. Setting the maximum number of detections too low can lead to the effect that some objects are found multiple times, but others not at all, especially for models that perform poorly.

### 5.6.1 Object Detection

Table 5.5 shows the results for RetinaNet and FRCNN object detection baselines using different weight initializations, backbone depths, and image resolutions. For a small (S) image size of $512 \times 384$ and initializing only the backbone of the models with the weights of a ResNet-50 classifier that has been pretrained on *ImageNet* [168], FRCNN clearly outperforms RetinaNet with respect to all measures, especially at the higher IoU threshold of 0.8. The advantage almost vanishes when the models are initialized with *COCO*-pretrained weights.[8] But what remains is the ten to eleven-fold number of false

[8]To be more precise, we can only initialize those weights that have exactly the same shape. This does not hold for the final prediction convolution layers where the output size depends on the number of classes.

| Model | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *AP* | *AR* | *AP* | *AR* | TP | FP | *AP* | *AR* | TP | FP |
| Image dimension 512 × 384 (S) | | | | | | | | | | |
| RetinaNet R50 * | 68.5 | 74.3 | 83.7 | 88.4 | 44016 | 161631 | 73.6 | 78.5 | 38782 | 166865 |
| RetinaNet R101 * | 72.1 | 77.4 | 86.0 | 90.0 | 44757 | 143939 | 77.5 | 82.0 | 40542 | 148154 |
| FRCNN R50 * | 74.1 | 79.8 | 84.1 | 87.2 | 43247 | 14024 | 77.9 | 82.6 | 40886 | 16385 |
| FRCNN R101 * | 76.2 | 82.0 | 86.1 | 88.9 | 44072 | 16590 | 80.1 | 84.8 | 41965 | 18697 |
| RetinaNet R50 | 76.8 | 81.5 | 86.3 | 89.7 | 44745 | 109185 | 80.2 | 84.1 | 41728 | 112202 |
| RetinaNet R101 | 78.3 | 82.8 | 87.2 | 90.5 | 45107 | 104633 | 81.6 | 85.3 | 42317 | 107423 |
| FRCNN R50 | 78.0 | 82.9 | 86.3 | 89.0 | 44197 | 10557 | 81.2 | 85.2 | 42221 | 12533 |
| FRCNN R101 | 79.0 | 83.8 | 86.8 | 89.4 | 44426 | 12134 | 81.9 | 85.7 | 42544 | 14016 |
| Image dimension 768 × 512 (M) | | | | | | | | | | |
| RetinaNet R101 | 80.8 | 84.9 | 88.6 | 91.5 | 45635 | 91371 | 83.6 | 87.1 | 43260 | 93746 |
| FRCNN R101 | 81.2 | 85.7 | 88.0 | 90.3 | 44890 | 9285 | 83.5 | 87.2 | 43314 | 10861 |
| Image dimension 1024 × 768 (L) | | | | | | | | | | |
| RetinaNet R101 | 81.6 | 85.6 | 88.9 | 91.7 | 45673 | 79014 | 83.9 | 87.1 | 43271 | 81416 |
| FRCNN R101 | 81.1 | 85.0 | 86.8 | 89.1 | 44263 | 9090 | 83.0 | 86.0 | 42733 | 10620 |

**Table 5.5: Box Detection: Influence of model choice, image size, backbone, and pretrained weights.** Overall *AP* and *AR* for IoU thresholds [0.5:0.05:0.95] are shown on the *D2S* test set, and together with the number of TPs and FPs at IoU thresholds 0.5 and 0.8, respectively. For models marked with * only the backbone has been initialized with *ImageNet* pretrained weights, all others have been initialized with weights from a model that has been pretrained on *COCO*. Increasing the spatial dimension of the input images from small to medium resolution leads to a large improvement of the *AP*, but there is no gain from medium to large resolution. At a low minimum confidence of 0.05 RetinaNet has a very high number of false positives in comparison to FRCNN, but this does not lead to a large difference in *AP*. A larger ResNet-101 backbone (R101) instead of a ResNet-50 backbone (R50) only slightly improves the result.

positives that RetinaNet predicts for a minimum confidence of 0.05 in comparison to FRCNN.

Initializing all weights of the model from a *COCO*-pretrained model, leads to a significant improvement of *AP* around between 2.8% (FRCNN R101) and 8.3% (RetinaNet R50). With *COCO*-initialization, at IoU 0.5 and minimum confidence of 0.05, the overall precision of RetinaNet R50 increases from 27% to a still very low value of 41% despite its relatively high *AP* value. For *COCO*-pretrained models, the use of a heavier ResNet-101 backbone increases the *AP* by 1-1.5% in comparison to a ResNet-50 backbone.

Increasing the image resolution to 768 × 512 (M) adds another 2.5% and 2.2% *AP* for RetinaNet R101 and FRCNN R101, respectively. While at IoU 0.5 RetinaNet and FRCNN are almost on the same *AP*-level, FRCNN predicts more accurate boxes such that it outperforms RetinaNet at higher IoU thresholds. For larger input images of 1024 × 768 (L) there is only a small increase in overall *AP* of 0.8% for RetinaNet and a marginal reduction of -0.1% *AP* for FRCNN, respectively. At IoU thresholds 0.5 and 0.8 the measures are on the same level or below their counterparts for the medium image size. However, for RetinaNet, there is a significant reduction of around 12 000 FPs when the image size is increased from M to L. Fig. 5.13 reveals that L models are stronger at very high IoU thresholds $> 0.8$. Especially for FRCNN, where for both IoU 0.5 and 0.8 only the number of false positives reduces and the *AP* even decreases by 1.2% and 0.5%,

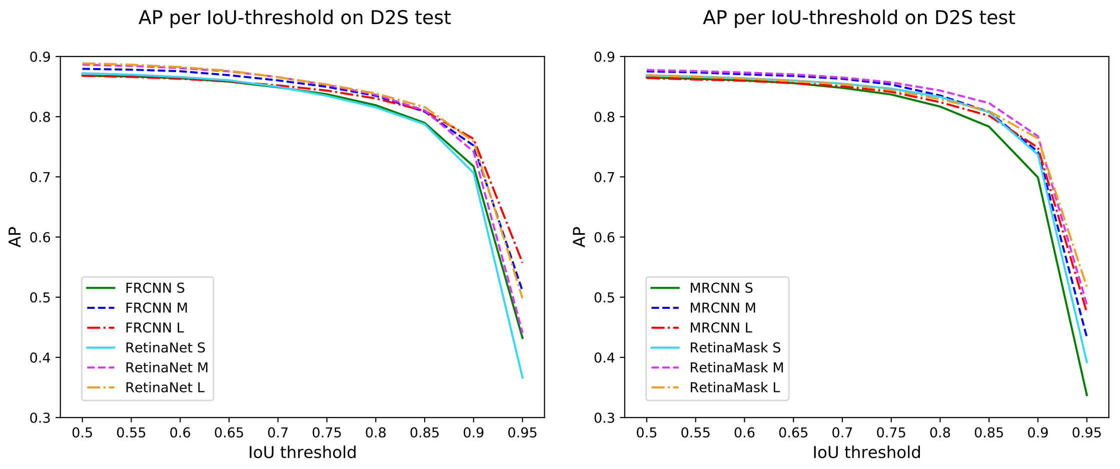


**Figure 5.13:** $AP$ **per** IoU **and image size on** ***D2S*** **test.** (*Left*) Box detection results of FRCNN and RetinaNet for different image resolutions. While a change from small to medium image size leads to a consistent improvement across all IoU thresholds, a change from medium to large image size only leads to more precise localization and improves only the $AP$ at high IoU thresholds. For lower IoU thresholds, higher $AP$ values are achieved using a medium image size. (*Right*) Instance segmentation results of MRCNN and RetinaMask. Similar to detection, the best results are achieved at medium image resolution. MRCNN S is by far the worst model. Generally, instance segmentation models have a slightly lower $AP$ then detection models already at IoU threshold 0.5. Note that the right plot shows *mask* $AP$, but the left plot *box* $AP$.

respectively. The figure also shows that the most potential for an improved overall $AP$ lies in the improvement of the $AP$ at high IoU thresholds. That means, only models with very precise localization are able to significantly improve the $AP$. On the other hand, also at IoU threshold 0.5 there is some room for improvement. A higher recall at this low IoU threshold will potentially also have positive effects for higher IoU thresholds. To find out what needs to be improved, we will look at qualitative results and a more detailed analysis of the baselines and their failure cases in Section 5.7. Overall, for RetinaNet the L model is the best choice, while for FRCNN, the M model yields the best trade-off between memory consumption, runtime, and accuracy of the results.

The HALCON implementation of RetinaNet and especially FRCNN achieves higher box $AP$ results than the Detectron implementations from the previous section, both for M and L models. Within Detectron, the used image resolution is slightly larger than the HALCON L models. In comparison to Detectron, HALCON has lower $AP_{50}$ values, but higher overall $AP$. This means that the HALCON box-predictions are more accurate, leading to higher $AP$ values at high IoU thresholds.

### 5.6.2 Instance Segmentation

Table 5.6 shows the results of our instance segmentation baselines. In general, the results are very similar to their detection counterparts in Table 5.5. This is no surprise because the instance segmentation results of both MRCNN and RetinaMask depend highly on the predicted boxes.

However, while RetinaNet R101 S performs worse than FRCNN R101 S, RetinaMask

| Model | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *AP* | *AR* | *AP* | *AR* | TP | FP | *AP* | *AR* | TP | FP |
| Image dimension 512 × 384 (S) | | | | | | | | | | |
| RetinaMask R50 | 78.1 | 82.3 | 85.9 | 89.1 | 44316 | 92680 | 81.7 | 85.2 | 42268 | 94728 |
| RetinaMask R101 | 79.3 | 83.5 | 86.9 | 90.0 | 44710 | 83192 | 83.3 | 86.7 | 43010 | 84892 |
| MRCNN R50 | 76.6 | 81.9 | 85.8 | 88.5 | 43961 | 10932 | 80.3 | 84.7 | 42047 | 12846 |
| MRCNN R101 | 77.7 | 83.5 | 86.5 | 89.7 | 44577 | 11289 | 81.7 | 86.5 | 42951 | 12915 |
| Image dimension 768 × 512 (M) | | | | | | | | | | |
| RetinaMask R101 | 81.4 | 85.3 | 87.8 | 90.6 | 45040 | 74100 | 84.3 | 87.6 | 43502 | 75638 |
| MRCNN R101 | 80.2 | 85.0 | 87.5 | 90.0 | 44721 | 9542 | 83.5 | 87.3 | 43369 | 10894 |
| Image dimension 1024 × 768 (L) | | | | | | | | | | |
| RetinaMask R101 | 80.8 | 84.6 | 86.9 | 89.9 | 44749 | 78759 | 82.9 | 86.2 | 42890 | 80618 |
| MRCNN R101 | 79.8 | 84.1 | 86.4 | 88.8 | 44112 | 8532 | 82.4 | 85.8 | 42672 | 9972 |

**Table 5.6: Instance segmentation: model choice, image size, and backbone depth.** Overall *AP* and *AR* are shown on the *D2S* test set, together with the number of TPs and FPs at IoU thresholds 0.5 and 0.8, respectively. All models have been initialized with weights from a model that has been pretrained on *COCO*.

R101 S is significantly stronger than MRCNN R101 S. Fig. 5.13 shows that here the overall *AP* of MRCNN R101 S is decreased by the bad results for high IoU thresholds. This means that a significant portion of the mask predictions is not accurate enough to count as TPs at high IoU thresholds.

For instance segmentation models, our baselines do not improve with an increase from medium (M) to large images (L). That is in contrast to the RetinaNet detection results, where RetinaNet L clearly outperforms its M counterpart at high IoU thresholds and achieves a higher overall *AP*. For MRCNN L and RetinaMask L, the overall *AP* drops by 0.4% and 0.6%, respectively. Hence, for both MRCNN and RetinaMask the M models have the best overall performance.

Also here, RetinaMask shows a very high number of FPs that does not affect the *AP* measure significantly. Interestingly, for IoU threshold 0.8, RetinaMask R50 S has a lower number of TPs (42 268) and a much higher number of FPs (94 728), but the same *AP* (81.7%) as MRCNN R101 S (42 951, 12 915, respectively). A possible reason is that there are specific categories where RetinaMask outperforms MRCNN significantly.

In comparison to the Detectron baselines from the previous section, here the mask *AP* of MRCNN is 1.0–1.3% lower than the box *AP* of FRCNN. Still, the overall achieved *AP* is on the same level or slightly higher than the Detectron result for M and L models (despite slightly smaller image size).

## 5.7 Failure Cases

The results that we have seen so far are remarkable and already on a quite high level with *AP* values above 80%. However, a closer look at the qualitative results reveals several shortcomings of the evaluated models. In this section, we provide a detailed failure case analysis and motivate the work of the following chapters.

### 5.7.1 Object Detection

Many of the instance segmentation failure cases are based on wrongly or inaccurately predicted boxes. Therefore, we start with an evaluation of box detection results. Table 5.5 shows that the overall $AP$ averaged over classes and IoU thresholds is not sufficient for the comparison of two models because the results may contain a high number of false positives that do not reduce the $AP$.

Moreover, for detection methods the accuracy is measured in two dimensions. One dimension concerns the classification accuracy and the other one the localization accuracy. To get more insight with respect to the reasons for false predictions, we look at the number of false positives (FPs) in different categories in the first row of Fig. 5.14: a false positive can occur due to a wrongly predicted class (*cls*); the localization (*loc*) can be bad, i.e., the prediction has an IoU below the given threshold with the ground truth annotation; there can be predictions within the background (*bg*) that have an IoU of 0.0 with any annotation; there can be duplicate predictions (*dup*), i.e., there is another TP that has higher score with the annotation; or there are multiple (*mult*) reasons for the prediction to be a false positive, mostly with wrong class and bad localization. We calculate these numbers for IoU threshold 0.5, since any prediction with bad localization at this threshold will also be an FP for higher IoU thresholds.

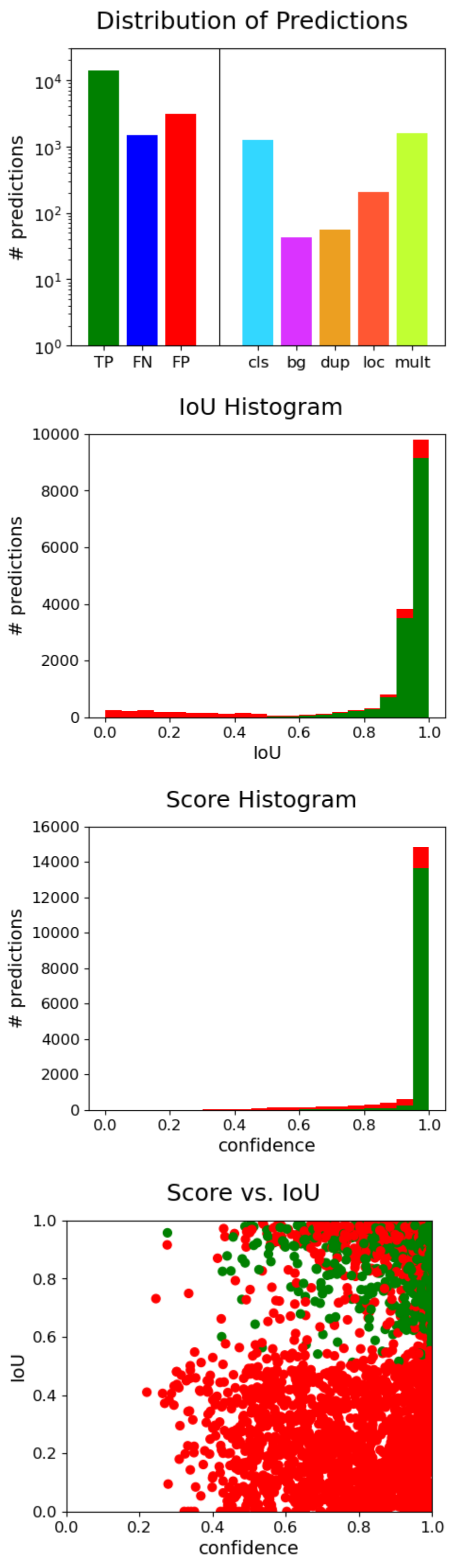
In order to analyze the localization performance in more detail, the second row of Fig. 5.14 shows the distribution of the obtained IoUs of the predictions with the annotations. Moreover, we show a histogram of the predicted scores and a score versus IoU scatterplot for FRCNN R101 S and RetinaNet R101 S in the third and forth row of Fig. 5.14. To generate this plot we choose a minimum confidence of 0.05, such that the $AP$ values for IoU threshold 0.5 get close to their maxima.

The $AP$-values of FRCNN and RetinaNet are almost equal (86.8% and 87.2% on test, 86.7% and 86.5% on val, respectively, Table 5.5). However, their distributions of predictions and numbers of false positives are significantly different.

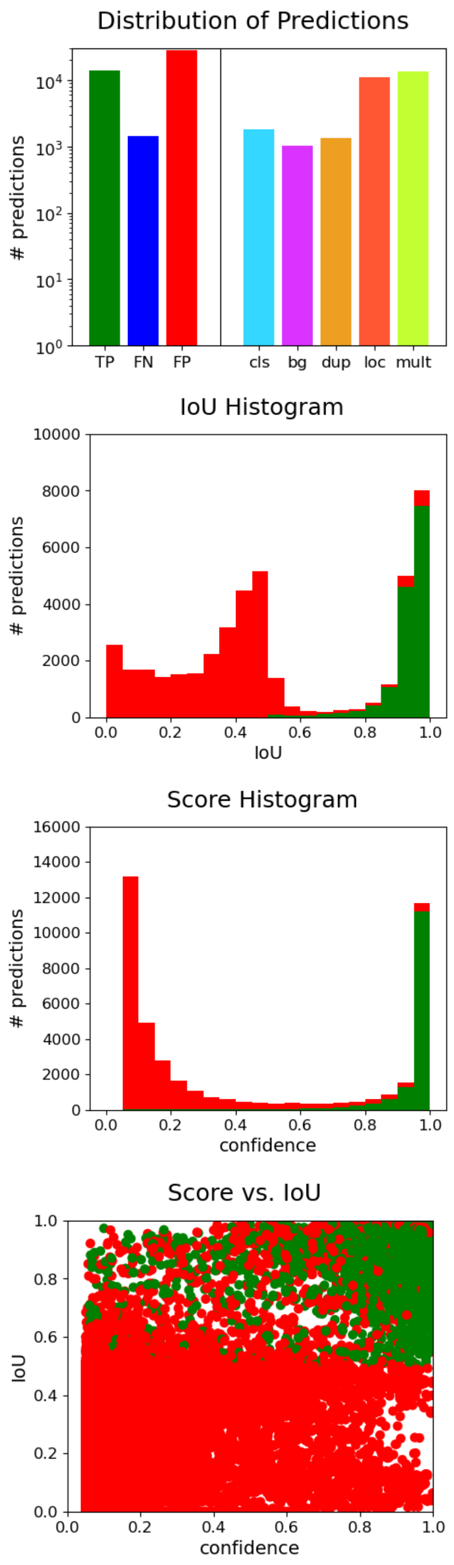
RetinaNet has a much higher number of FPs in general, but in particular for the categories *bg, dup, loc*, and *mult*. Since the average recall is higher for RetinaNet than for FRCNN, this means that these false positives are additional to the TPs and they are not penalized within the $AP$ measure. However, the scatterplot at the bottom of Fig. 5.14 reveals that just increasing the global minimum confidence threshold leads to a drop in recall and would thus reduce the $AP$. We will further analyze this issue in Chapter 6.

While the total number of TPs is slightly higher for RetinaNet (44 839) than for FRCNN (44 483), the histogram of IoUs with ground truth annotations reveals that FRCNN predicts more accurate boxes and thus has higher $AP$-values at higher IoU thresholds. The vast number of FPs of RetinaNet are mainly distributed over the IoU-range from 0.0 to 0.55, which can also be seen from the fact that the majority of FPs is in the categories *loc* and *mult*. FRCNN generally has a much lower number of FPs and they are mainly caused by a wrong class prediction or a bad localization (or both).

The score histograms and the score vs. IoU scatterplots show another significant difference between the two model types: While for RetinaNet the scores of predictions



(a) FRCNN R101



(b) RetinaNet R101

**Figure 5.14: Distribution of detection failure modes, scores, and** IoUs **on *D2S* val.** The figures reveal that despite a very similar $AP_{50}$ value of 86.9% vs. 86.8% for FRCNN and RetinaNet, respectively, the number of FPs and their score distribution is significantly different. See text for further interpretations.

start close to the minimum confidence, for FRCNN the majority of confidences are close to 1.0. This can be explained by a different calculation of scores: FRCNN uses a softmax on the output of the class prediction fc-layer and RetinaNet uses a sigmoid activation on the activations of the class prediction convolution. This might also be a reason for the significantly higher number of predictions and FPs of RetinaNet, because the sigmoid activation is not exclusive, i.e., for each anchor within the RPN each class can be predicted. In the architecture of FRCNN only the class with the highest activation is predicted and all other categories are suppressed. Hence, in cases where the model is unsure whether the underlying object belongs to class A or B, FRCNN will guess and by some chance be right. If it is right, FRCNN predicts one TP, if it is wrong FRCNN predicts one FP and gets one FN. In comparison, RetinaNet will most likely predict a box for both classes A and B, leading to one TP and one FP, but no FN. This could also be the reason why RetinaNet has a slightly higher $AR$ than FRCNN (90.5% vs. 89.4% at IoU 0.5).

Fig. 5.15 shows some qualitative results of failure cases. Clearly, there are many more FPs for RetinaNet than for FRCNN, but a closer look reveals that also a true positive is present for almost all objects. Both models fail mainly in cases that have not explicitly been part of the training data:

- FPs occur in the background when the background is different to the brown paper background from the training set (*6th and 7th row*).
- The models have difficulties with touching or overlapping objects (*1st and 2nd row*).
- Both models fail if the object appearance is slightly different than known from the training set. For example, for the Coca Cola bottle in the first or fourth row of Fig. 5.15, the classification or the localization fails if the bottle appears in a side view and empty instead of standing and filled. The same holds for some tea boxes, where some of the categories are only captured from a specific view within the training set. If the box is rotated and the printing is only slightly different to the training set, the classification typically fails or the localization is wrong (*5th row*).
- Within the training set the objects are usually positioned close to the center of the turntable. Objects that reach out of the image boundary are often not found at all or wrongly classified (*7th row*).
- Within the validation and test set, oversaturated images occur due to reflections, a long exposure time, or a light background. This can lead to a very low contrast for some objects. In turn, this often leads to a bad localization or objects that are not found at all (*8th row*).

### 5.7.2 Instance Segmentation

Since the instance segmentation models basically just add a mask prediction head to the detection architectures, it is no surprise that the distributions of the instance mask predictions of MRCNN and RetinaMask are close to the box prediction distributions

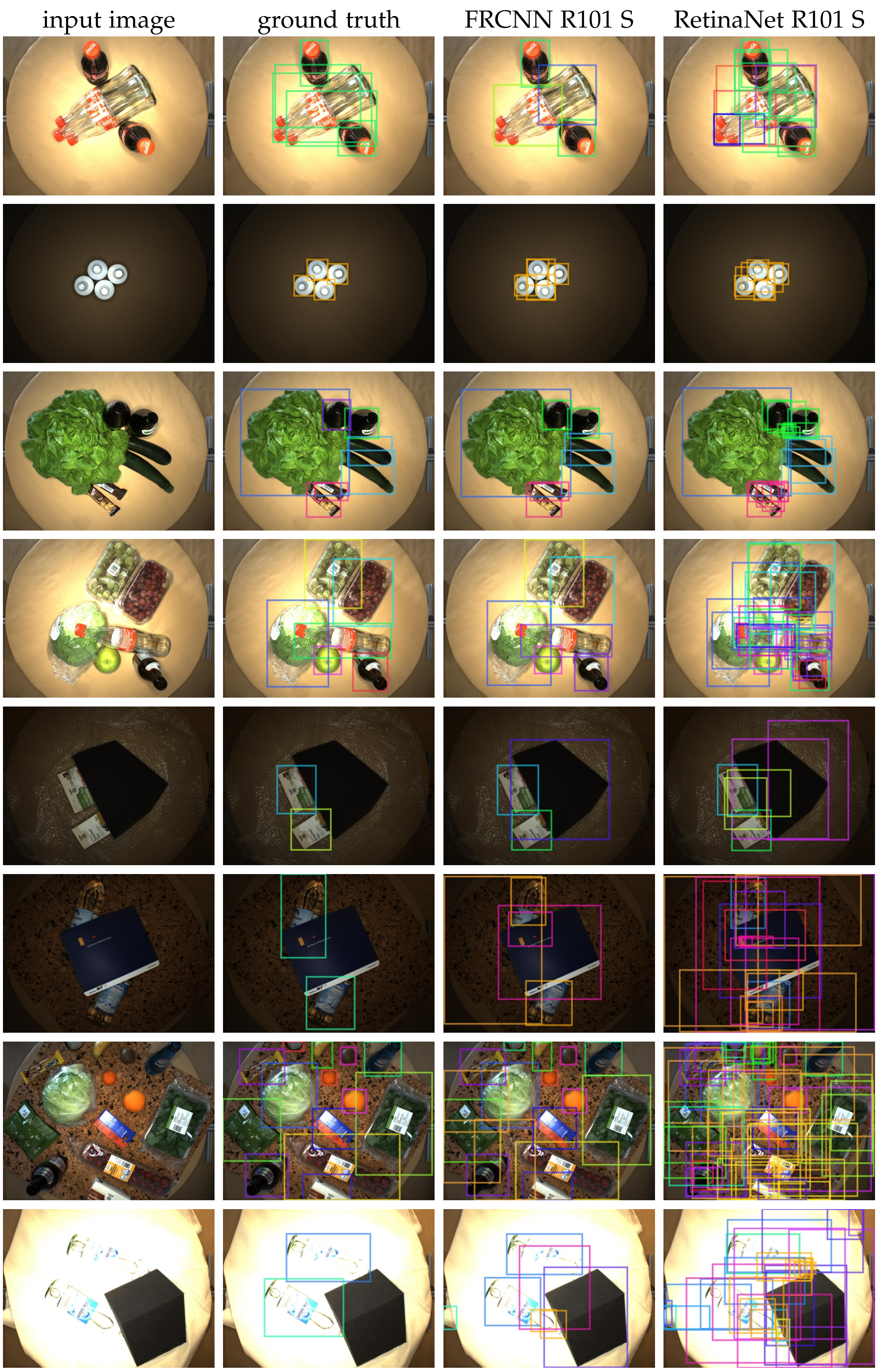
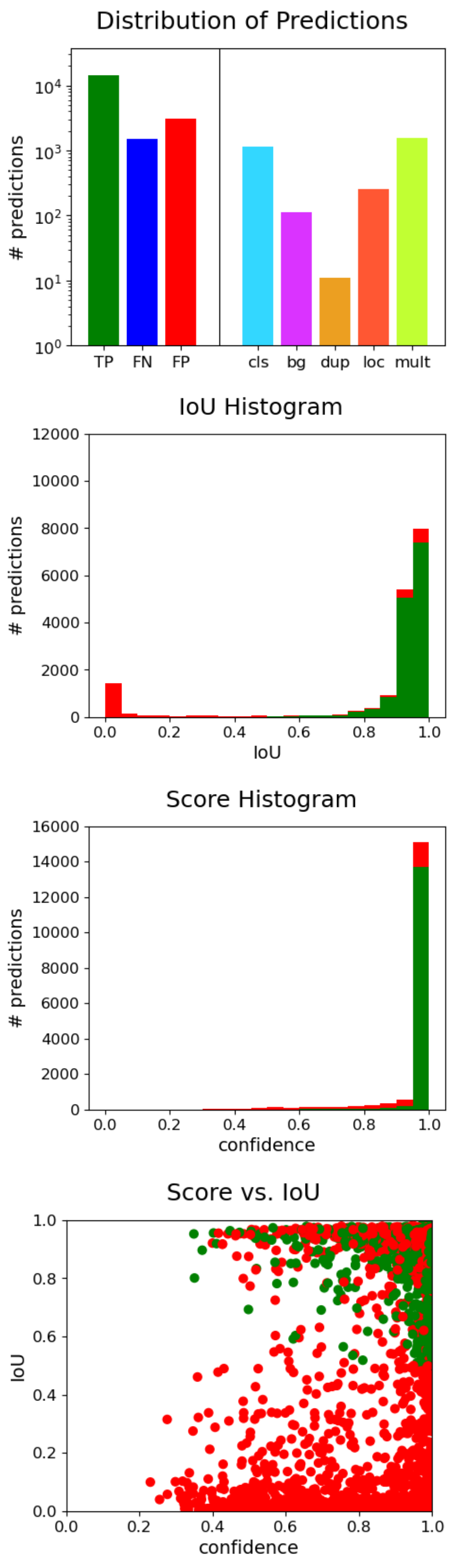
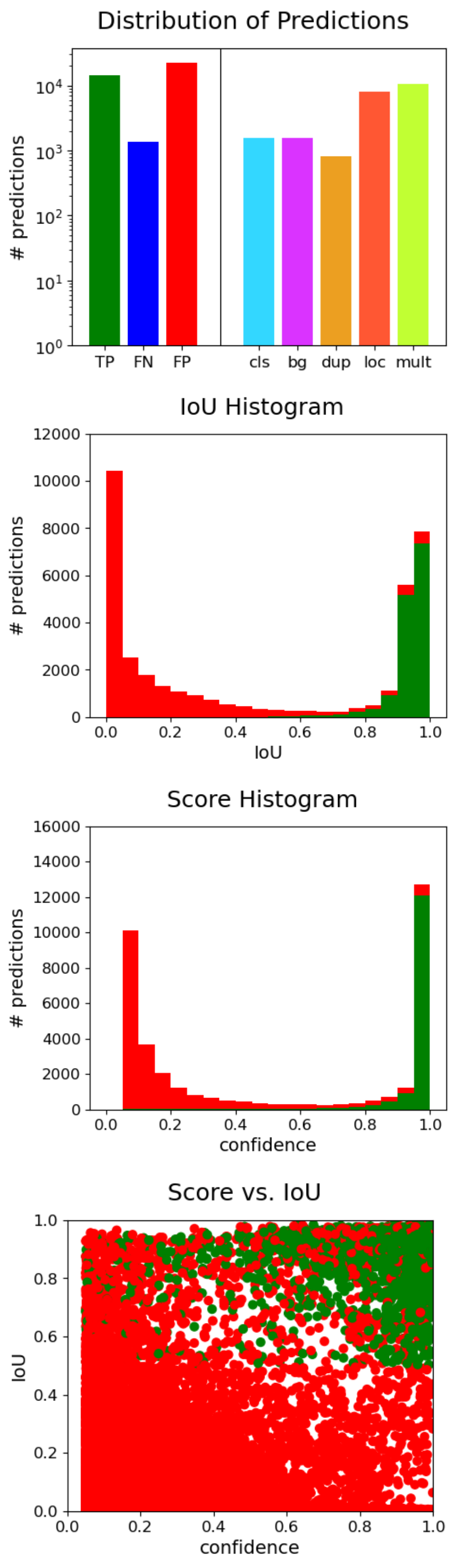


**Figure 5.15: Detection Failure Cases on *D2S* test.** Failures occur due to occluding or touching objects, wrong classifications, especially fine-grained classification problems, clutter objects, unknown backgrounds, or inaccurate localizations. RetinaNet shows significantly more false positives (*best viewed digitally and with zoom*).



(a) MRCNN R101



(b) RetinaMask R101

**Figure 5.16: Distribution of instance segmentation failure modes, scores, and** IoUs **on *D2S* val.** RetinaMask predicts many more FPs than MRCNN, especially with very low IoU to any annotation. Scores of RetinaMask are spread more uniformly across the interval from the minimum confidence to 1.0. See text for further interpretations.

of their detection counterparts FRCNN and RetinaNet, respectively. Fig. 5.16 shows that RetinaMask has a much higher number of FPs than MRCNN, and scores are more uniformly distributed over the whole range from zero to one. However, a comparison of Fig. 5.16 with Fig. 5.14 reveals that there are some differences between the box detection models and the instance segmentation ones:

For both MRCNN and RetinaMask, a majority of FPs have a maximum IoU with any GT annotation in the range [0, 0.05]. This means that there are many mask predictions solely within the background or there are predictions where no mask is present at all. Moreover, for FRCNN a majority of TPs have an IoU in the range [0.95, 1]. For MRCNN, still the largest fraction of TPs has a mask IoU within this range, but a significant portion of TPs has moved to the [0.9, 0.95) IoU interval. This can have a large impact on the overall $AP$ since $AP_{95}$ is dropping significantly with this change. The score vs. IoU scatter plot of RetinaMask shows that in comparison to the RetinaNet detection model it predicts more FPs that have a very high IoU but a low score. Presumably, these must be either duplicate or wrong class predictions.

The qualitative results in Fig. 5.17 underline that when a box prediction has a bad localization and mainly overlaps with the background, the mask prediction is either very bad or it is surrounding the objects that it contains (*6th to 8th row*). It is interesting that the mask head predicts a high mask probability within the background although it has never been trained for that. A reason could be that the training set only contains a non-textured background and hence any difference to the brown uniform paper background within the training images indicates that there must be an object.

If a box prediction does not perfectly fit one of several neighboring objects, then the predicted mask mostly covers parts of all of these objects instead of predicting a good mask for only one of them (*2nd and 6th row for FRCNN, almost all rows for RetinaMask*).

Neighboring or touching objects are a challenge also in terms of mask prediction, but in the case that the box is predicted correctly, both models can learn to predict an instance mask that only covers the object of interest. However, for overlapping objects the mask is often inaccurate at the locations of overlap (*1st and 2nd row*).

For large objects that are diagonally aligned (*the pasta bag in the 4th row*) there are subtle artifacts that result from upsampling the low resolution mask predictions to the original box dimension within the input image.

A very common artifact is that whenever there is a clutter object that has not been part of the training set, the models predict an almost perfect instance mask for these (*the mouse pad in the 7th and 8th row*). Theoretically, a class-specific mask head could help to reduce this problem, but we did not find that there is a significant improvement compared to the used class-agnostic prediction. Moreover, the class-specific mask prediction is computationally more expensive, especially during the training phase.

Of course, also for instance segmentation, a low contrast due to a light background and a too long exposure time leads to inaccurate masks (*3rd and 8th row*).

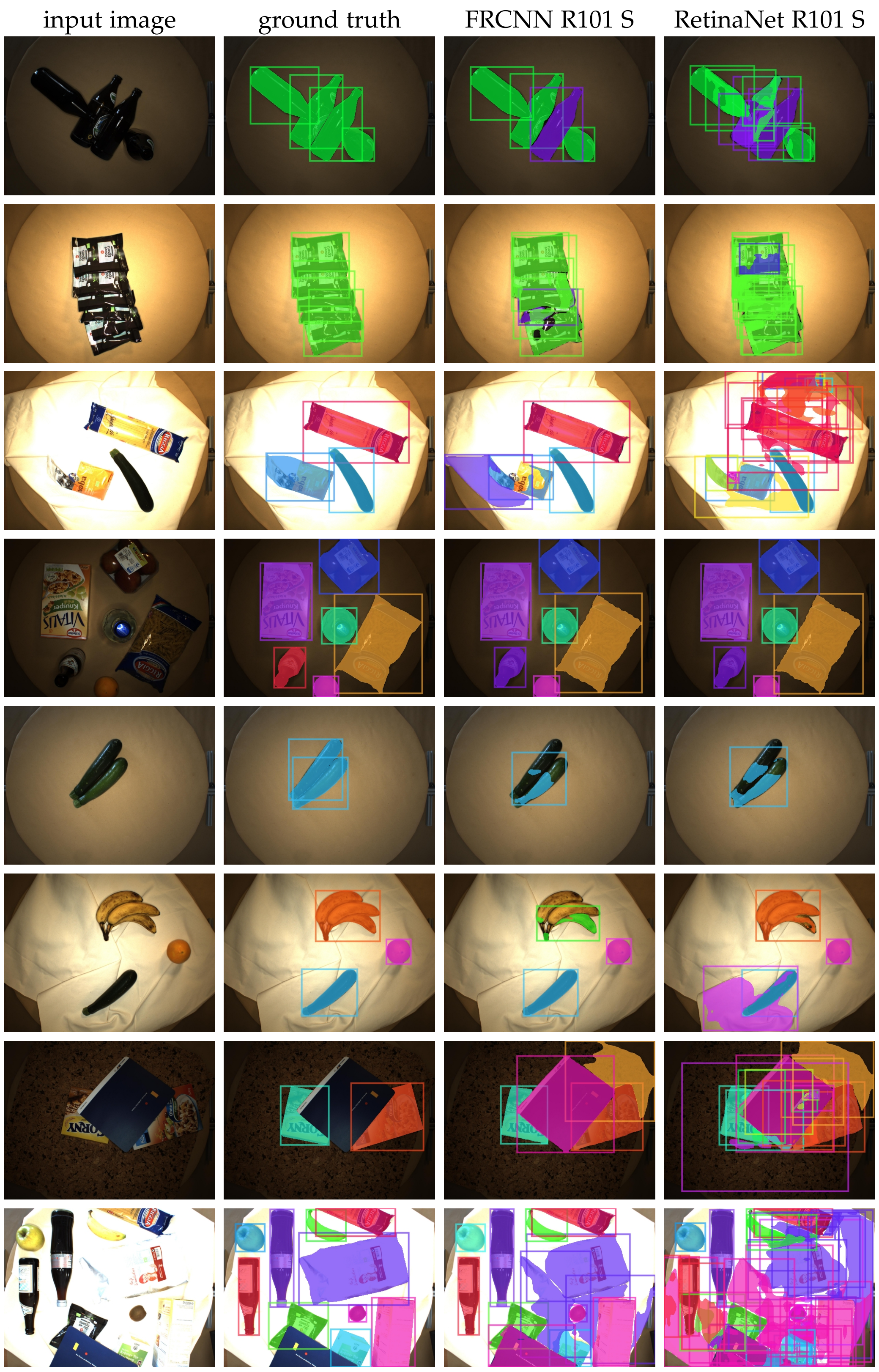


**Figure 5.17: Instance Segmentation Failure Cases on *D2S* test.** Failure cases have similar reasons as in the detection case. Additionally, if the box localization is bad this leads to cropped masks. For boxes that reach over multiple objects or boxes within the background, usually the mask predictions are not reasonable (*best viewed digitally and with zoom*).

### 5.7.3 Summary of Issues and Further Work

In this and the previous section, we have seen several failure cases and issues of both the detection and instance segmentation methods on *D2S*. We summarize them here once again and refer to the following chapters, in which we address them:

- High *AP* values do not necessarily lead to a low number of false positives. Especially for RetinaNet and RetinaMask, despite high *AP* values, we have seen a very low precision and many false positives with a low IoU to any ground truth annotation. If we think about the typical application of the *D2S* scenario, i.e., an automatic warehouse or supermarket checkout system, these predictions are items that are on the customers invoice — either as additional items that are not really present, or as wrongly classified objects. This issue will be addressed in Chapter 6.
- A large number of false positives occurs due to a domain gap between the training data and the validation and test data. This is the case for the failure cases due to neighboring objects, backgrounds that have not been part of the training data, and failures due to reflections. One way to solve this issue is by using data augmentation as shown in Chapter 7.
- Axis-aligned box detections often cover a large part of the background. Thus, when multiple, close-packed objects are within the image, for a human it is sometimes difficult to visually assign the detections to the underlying objects. Moreover, for two-stage detection methods such as FRCNN, MRCNN, or RetinaMask, a significant portion of the features that are used within the second or a consecutive stage are pooled from the background or from neighboring objects. Hence, in Chapter 8, we extend the box detection to oriented boxes and analyze whether this helps to improve the class prediction.
- The *AP* is calculated as an average over different IoU thresholds in order to highlight accurate models. So far, at high IoU thresholds, the models are weak. That means, the predictions need to be more accurate to improve the overall *AP*.
  - If applicable, matching algorithms can lead to very accurate subpixel-precise detections. In Chapter 9, we compare a matching algorithm with deep-learning-based oriented box detection and present a hybrid approach that fuses both.
  - Unfortunately, the matching approach is not suitable in the *D2S* case due to perspective distortions and it also does not predict instance masks. Hence, we present another method that fuses the oriented box detection with instance segmentation for accurate mask prediction in Chapter 10.
- We have seen that many of the failure cases arise from touching or overlapping objects. A method to predict instance masks beyond their visible parts is presented in Chapter 11.

## 5.8 Conclusion

We have introduced *D2S*, a novel dataset for instance-aware semantic segmentation that focuses on real-world industrial applications. The dataset addresses several challenges that are highly relevant in the field, such as dealing with very limited training data. The training set is intentionally small and simple, while the validation and test sets are much more complex and diverse. As opposed to existing datasets, *D2S* has a very uniform distribution of the samples per class. Furthermore, the fixed acquisition setup prevents ambiguities in the labels, which in turn allows flawless algorithms to achieve an AP of 100%. We further showed how the high-quality annotations can easily be utilized for artificial data augmentation to significantly boost the performance of the evaluated instance segmentation methods from an AP of 49.5% and 45.6% to 79.9% and 72.5%, respectively.

Moreover, we have done an extensive quantitative and qualitative result and failure case analysis of our baselines. Our analysis reveals severe issues of current instance segmentation methods that predict a high number of false positives. These are not all taken into account by the current evaluation protocol *AP*, which makes it difficult to rank different models in a meaningful way.

Overall, the benchmark results indicate a significant room for improvement of the current state-of-the-art. We believe the dataset will help to develop more accurate instance-aware segmentation methods and leverage new approaches for artificial data augmentation as will be shown in the following chapters.

# 6
# $AP^\star$ — Fixing the Recall Bias of Average Precision

We have seen in Chapter 5 that comparing two models solely based on the overall mean average precision (AP) measure in many cases is not sufficient. The models' predictions can have widely different distributions of scores and IoUs with the ground truth annotations although their *AP* is almost equal. In particular, our analysis has shown that using the current form of *AP* as the selection criterion favors models with a potentially very high number of false positives. In order to promote algorithms that are reliable and usable in industrial applications (cf. Chapter 1), we propose to improve the evaluation protocol that is used for the evaluation on datasets like *COCO* [118], *LVIS* [62], and many other detection datasets and challenges. In this chapter, we provide additional insights to potential problems of *AP*, especially for a low number of per-category ground truth instances. We further propose certain changes to the way *AP* is calculated and introduce the novel $AP^\star$ measure that incorporates the actual precision of a method. This makes $AP^\star$ more reliable than *AP* and avoids selecting models that show a high number of false positives. Moreover, in comparison to *AP*, $AP^\star$ allows to calculate optimal class-specific or class-agnostic score thresholds to be used in the application.

In our experiments on *D2S*, we show that if the score thresholds of RetinaNet are optimized for $AP^\star$ instead of *AP*, the actual average precision of the model can be increased from 32.9% to 82.7%, while the *AR* is kept at a high value of 79.8%. The same holds for FRCNN, where the precision increases from 78.5% to 86.4% with an *AR* of 80.4%. On the one hand, with the proposed changes to *AP*, we strengthen its expressiveness while we keep the benefits of the measure. On the other hand, the extension of *AP* to $AP^\star$ leads to more reliable models and thus $AP^\star$ is a good alternative to *AP* as an evaluation measure for detection and instance segmentation methods.

## 6.1 Introduction

Evaluation is potentially the most fundamental step of the model selection process in every industrial application. Evaluation is mainly used to select the best model architecture among several possibilities, to tune hyperparameters, or to select the best model state during the training process. The higher goal of an evaluation is usually to predict how well the given model will perform on new, unseen data, e.g., when being used in a machine during the online phase. As described in Chapter 2, the selection of the best model state or hyperparameter setting is typically done with the help of a validation set, and for the final model selection or comparison within a benchmark, another disjoint test set is used. However, when looking at the evaluation results on the validation and test sets, one should always keep in mind that these sets just contain a finite number of samples. Those samples have (hopefully) been drawn from the same image distribution as the images that the model will be faced with during the online phase, but we do not always have a guarantee for that. Hence, with an evaluation we just get a prediction of how well the model might perform in the underlying application and this prediction might be wrong or at least contain a bias.

As introduced in Section 4.2, the evaluation of detection methods needs to incorporate two dimensions at the same time: one is measuring the quality of localization, and the other is measuring the accuracy of classification. Today, the predominant evaluation measure in detection challenges, like *COCO* [118], *LVIS* [62], or the Robust Vision Challenge,[1] is *AP*, as introduced in *VOC* [40] and adapted by *COCO*.

*AP* handles the localization by introducing IoU thresholds that define when a prediction is accurate enough to count as a true positive (TP). With respect to the classification dimension, at the same time the predicted class needs to be equal to the class of the underlying ground truth annotation. Setting an appropriate IoU threshold allows to define how strict the localization should be judged. Choosing lower IoU thresholds can be appropriate if the underlying data is difficult or ambiguous to annotate or if a very precise localization is not important to solve the task at hand.

Generally, the benefit of using a single evaluation value like *AP* is that the comparison and ranking of models is straightforward. But at the same time, all different aspects of the model's result are compressed into this single value and some aspects might be lost or not be represented as much as others. For example, during the calculation of *AP*, the *AP* values for individual IoU thresholds are averaged and they in turn are averages of the per class *AP*s. This means that two models where one has *AP* 1.0 for half of the classes and *AP* 0.0 for the remaining classes has the same overall *AP* as another model that gets an *AP* of 0.5 for all other categories. However, it is very difficult to reduce the evaluation result to a single number and at the same time keep such information. Still, when looking at the evaluation result, the user wants to see a difference between two models with very different distributions of predictions.

Therefore, in this work, we take a closer look at the *AP*. In particular, we make the following contributions:

- We explain why the *AP* in its current form has a bias towards recall and is not penalizing many false positives with low scores.

[1] `http://www.robustvision.net/index.php`

- We show that $AP$ cannot be used to infer an optimal global nor class-specific score thresholds because it always favors models with score thresholds set to 0.0.
- We introduce $AP^{\star}$ as a refinement of $AP$ that fixes the recall bias of $AP$ and allows to compute optimal global or class-specific score thresholds.
- We revisit the $AP$ in detail and have a closer look at the PR-curve interpolation and the subsampling that is done within the $AP$ calculation.
- In our experiments, we show that using $AP^{\star}$ instead of $AP$ as the model selection criterion leads to a drastically reduced number of FPs and hence to models with a much better precision–recall trade-off.
- Finally, $AP^{\star}$ allows a better comparison of different models without the need to set score thresholds manually.

To showcase that the mentioned issues are not just theoretical, in Section 6.5, we look at some examplary cases where the overall $AP$ is very similar, but the result distribution is very different. In particular, we show how the results from the previous chapter on *D2S* can be significantly improved. For both RetinaNet and FRCNN, a majority of FPs within the background or due to clutter objects can be suppressed by the use of optimal score thresholds based on $AP^{\star}$. For example, with this modification, the mean precision on the *D2S* test set can be increased from 78.5% to 86.4% for FRCNN and from 32.9% to 82.7% for RetinaNet, respectively. At the same time, the average recall is only slightly reduced from 83.8% to 80.4% for FRCNN and from 82.8% to 79.8% for RetinaNet, respectively. $AP^{\star}$ also helps to obtain a more meaningful ranking of detection models because FPs with low scores are not ignored in its calculation. For example, RetinaNet with a low score threshold and many FPs only achieves an $AP^{\star}$ of 53.2%, but if FPs are suppressed with an appropriate score threshold, we can increase the $AP^{\star}$ value to 72.1%.

## 6.2 Related Work

With the introduction of Pascal *VOC*, Everingham et al. [39, 40] first defined the mean average precision ($AP$) as an evaluation measure for box detection methods. They proposed to use a single and relatively low IoU threshold of 0.5 in order to account for imprecise labels. Due to the popularity of the challenge, $AP$ is still used today. During the period of the *VOC* challenge, the way how the $AP$ was computed as the area under the precision–recall (PR) curve was changed: from 2010 onwards, the PR curve subsampling has been removed (cf. Section 6.3.1). Lin et al. [118] slightly adapted the $AP$ again when they published the *COCO* dataset and detection challenge. We describe the $AP$ calculation of both *VOC* and *COCO* in Section 6.3.

Hoiem et al. [74] provide visualization tools to extend the evaluation of a detector summarized in a single $AP$ value. The tools do not change the evaluation measure, but reveal the causes of errors that are made by a detector. Thus, this work can be seen as a complementary contribution to the development of new detection measures.

Oksuz et al. [148] propose the Localization Recall Precision ($LRP$) error metric as an alternative to $AP$. The authors motivate their new measure by some of the issues that we list in Section 6.4, e.g., non-penalized FPs (Section 6.4.1), interpolation of PR curves (Section 6.4.2), and they claim that $AP$ does not propose a (class-specific) score threshold. To address these deficiencies, $LRP$ does not average over all possible precision and recall pairs. For a given IoU threshold $T_{\text{IoU}}$, the $LRP$ at an optimal score threshold $s$ per class can be calculated. It represents a trade-off between localization, precision, and recall. For a given threshold $s$, a set of $n_{GT}$ GT annotations $G$ and the set of detections with score of at least $s$ $D_s$, $LRP$ error is computed as

$$LRP(G, D_s) = \frac{1}{Z}\left(w_{\text{IoU}} LRP_{\text{IoU}} + w_{FP} LRP_{FP} + w_{FN} LRP_{FN}\right), \tag{6.1}$$

where $Z = n_{TP} + n_{FP} + n_{GT}$, $LRP_{\text{IoU}}$ is $1-$ the mean IoU of true positives, $LRP_{FP}$ is $1-$ precision, and $LRP_{FN}$ is $1-$ recall. The weights are set to $w_{\text{IoU}} = n_{TP}/(1 - T_{\text{IoU}})$, $w_{FP} = |D_s|$, and $w_{FN} = n_{GT}$. If the IoU threshold is fixed, the optimal score threshold $s$ for a detector is given by the optimal $LRP$:

$$oLRP(G, D) = \min_s LRP(G, D_s), \tag{6.2}$$

$$T_s^{LRP} = \arg\min_s LRP(G, D_s). \tag{6.3}$$

Moreover, the authors claim that *"[a]nother deficiency of AP is that it does not explicitly include localization accuracy"* [148, p. 505]. However, this is done by averaging over multiple IoU thresholds. $LRP$ uses the $LRP_{\text{IoU}}$ term to favor more accurate localizations, but the IoU threshold parameter $T_{\text{IoU}}$ remains.

$LRP$ is a valid alternative for $AP$, but it is not widely used in the community. A certain drawback of $LRP$ is that it does not distinguish between FPs with high confidence and those with low confidence, while this is incorporated in $AP$ via the PR curve. Oksuz et al. claim that *"LRP represents the shape of the PR curve via its components"* [148, p. 506] and that *"AP cannot distinguish between very different PR curves"* [148, p. 515]. We believe the opposite is true: by using the $AP$ correctly (with proper sampling and no interpolation), the $AP$ is indeed different for different PR curves. Moreover, we do not see how $LRP$ does represent the shape of the PR curve, since it is just defined over the number of TPs, false negatives (FN), and the mean IoU of TPs. We will show in Section 6.5 in a simple example that $LRP$ is not always intuitive to interpret.

Another aspect is that $LRP$ is strictly penalizing detections with IoU $< 1.0$. We claim that for datasets with low annotation reproducibility this should be avoided. In comparison, for $AP$ the preferred level of localization accuracy can be set via the used IoU thresholds.

As stated in the survey papers of Liu et al. [122] and Zou et al. [217], apart from the $LRP$ and $AP$, there are no commonly used evaluation measures for detection methods. Also in the recently introduced benchmark dataset and challenge of *LVIS* [62], $AP$ is used as defined by the *COCO* authors. Of course, also for detection methods we can compute general measures, such as the precision or recall (cf. Section 6.3.1).

## 6.3 Average Precision as Detection Measure

In this section, we explain *AP* as it is used in the current benchmark datasets like *VOC*, *COCO*, or *LVIS*. We also go through the changes that have been made between the definition in *VOC* and *COCO*. All definitions can be used both for detection methods that predict boxes or instance masks — just interpret the box as a box-shaped mask.

### 6.3.1 Basics

For convenience, we repeat some of the definitions from Section 4.2 here:

In comparison to classification, where the predicted class can either be right or wrong, for detection we get a new dimension of the evaluation with respect to the localization. To measure how well a prediction $A$ fits a ground truth box $B$, we measure the intersection over union (IoU), i.e., the area of the intersection divided by the area of the union:

$$\mathrm{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{6.4}$$

For a prediction, the localization is correct if the IoU with a ground truth box is higher than a threshold $T_{\mathrm{IoU}}$. Depending on the necessary precision, $T_{\mathrm{IoU}}$ can be chosen higher or lower.

To be a true positive (TP), the prediction must have a correct localization and also the correct class, i.e., the same as the ground truth box that it has been matched to. We have seen in Section 4.2 and the previous chapter that there are various reasons for a prediction to be a false positive (FP) and that usually an FP that has either a wrong class or is a duplicate also leads to a false negative (FN).

If we denote $G$ as the set of all $n_{GT}$ GT boxes, $n_{FN}$ as the number of FN boxes, $D$ as the set of all $n_D$ predictions, $n_{TP}$ as the number TP predictions, and $n_{FP}$ as the number of FP predictions, we can calculate the following well-known relations:

$$\text{Recall} = r(G, D) = \frac{n_{TP}}{n_{GT}} = \frac{n_{TP}}{n_{TP} + n_{FN}}, \tag{6.5}$$

$$\text{Precision} = p(G, D) = \frac{n_{TP}}{n_D} = \frac{n_{TP}}{n_{TP} + n_{FP}}. \tag{6.6}$$

### 6.3.2 PR Curve

To form a precision–recall (PR) curve, all detections $d$ in $D$ are sorted by their score $s = s(d)$. We denote $d_i$ as the detection with the $i$-th highest score in $D$, and $D_i := \{d_j \in D | j \leq i\}$ as the set of all highest scoring detections up to the $i$-th highest scoring one.

Iteratively, for $i = 1, \dots, n_D$, we can compute the precision–recall pairs $(p_i, r_i) = (p(G, D_i), r(G, D_i))$. The PR curve is usually a left-continuous step function: we have $n_{GT} + 1$ recall-steps $r_j = j/n_{GT}, j = 0, \dots, n_{GT}$. For $r \in (r_{j-1}, r_j], j = 1, \dots, n_{GT}$, the corresponding precision is then given by

$$p(r) = \max_i \{p_i | (p_i, r_i) \text{ s.t. } r_i = r_j,\ r_{j-1} < r <= r_j\}. \tag{6.7}$$

An example of a PR curve is given on the left side of Fig. 6.1.

To the best of our knowledge, the intention of the left-continous step function is the following: the PR curve is used to find a good precision–recall trade-off. This is done by choosing a score threshold $T_s$ and suppressing all detections with $s(d) < T_s$. Instead of choosing $s(d_{i-1}) < T_s < s(d_i)$, based on the given evaluation the user should always choose either $T_s = s(d_{i-1})$ or $T_s = s(d_i)$ (compare with the left plot in Fig. 6.1):

- If $d_{i-1}$ is a TP and $d_i$ is a TP, we choose $T_s = s(d_i)$, since $p_i \geq p_{i-1}$ and $r_i > r_{i-1}$,
- if $d_{i-1}$ is a FP and $d_i$ is a TP, we choose $T_s = s(d_i)$, since $p_i > p_{i-1}$ and $r_i > r_{i-1}$,
- if $d_i$ is a FP, we choose $T_s = s(d_{i-1})$, since $p_i < p_{i-1}$ and $r_i = r_{i-1}$, and
- if $d_{i-1}$ is also an FP, we can iteratively continue to increase $T_s$ until we come to the next TP and the maximum precision for this recall is reached.

However, if we see our evaluation set as a predictor for the unknown online phase, we question whether the use of a step function makes sense, especially for rare categories where the evaluation set contains only a few GT annotations (e.g., $n_{GT} < 20$). In this case, we actually do not know much about the distribution of our predictions with respect to their scores and their assignment to TPs and FPs. Hence, we also do not know how the shape of the PR curve between the recall steps would look like if we had a larger evaluation set. For increasing $n_{GT}$, the differences between different kinds of interpolation become smaller. Thus, we will stay with a left-continuous step function for the PR curve in this work.

The value at $r_0 = 0.0$ of the PR curve is undefined if the first prediction is a TP because precision is not defined if no detection is present, otherwise it is 0.0.

Another way to think of the PR curve computation is the following: It is built from $n_D$ precision–recall pairs $(p, r)$. Whenever we add a new detection $d_i \cup D_{i-1} = D_i$, we make a step. If $d_i$ is a TP, recall is increased and, therefore, we call this a recall step. In total, for $D$, there are $n_{TP} \leq n_{GT}$ of these. At the same time, for a TP, the precision increases if it is not already at 1.0. In case of an FP, the precision decreases and the recall remains constant. While the recall increments always have the same size of $1/n_{GT}$, the precision steps become smaller and smaller as the overall number of predictions increases. More precisely, the $i$-th precision step is $-1/(|D_{i-1}| + |D_i|)$ if $p_{i-1} > 0$ and $d_i$ is an FP, and it has the size of $(1 - p_{i-1})/|D_i|$ if $d_i$ is a TP. This means that those detections with the highest scores have the largest influence on the precision, but only TPs have an influence on the recall and it is always the same, independent of their score.

We refer to those FPs with high scores that appear on the left side of the PR curve as *early* FPs and to those with low scores on the right side of the PR curve as *late* FPs. In particular, early FPs are harmful during the use of our model because their scores are

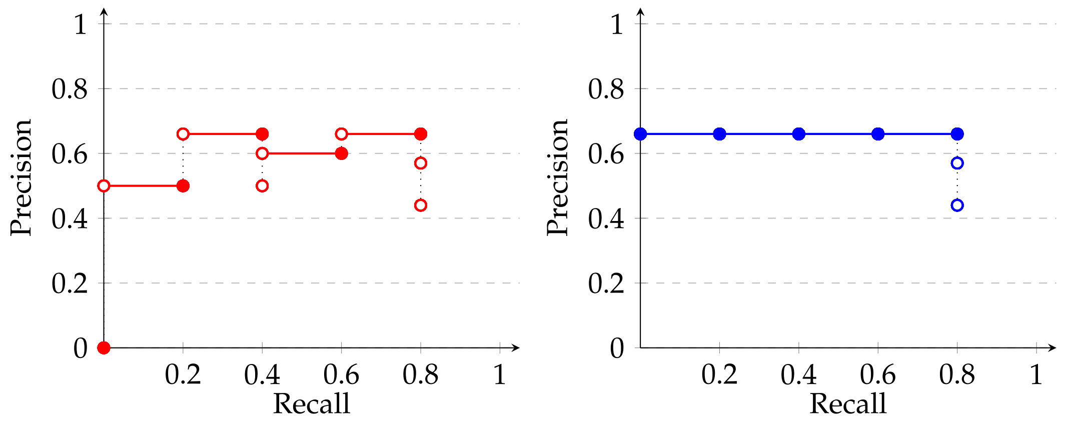


**Figure 6.1: Exemplary PR curves.** $n_{GT} = 5$, the order of detections is (FP, TP, TP, FP, TP, TP, FP, FP). (*Left, red*) A non-interpolated PR curve, leading to $AP_{\text{real}} = 0.484$. (*Right, blue*) The interpolated counterpart with $\overline{AP}_{101} = 0.528$, i.e., 0.044 increase of $AP$.

higher than most of the scores of TPs and thus they cannot be filtered out by increasing the score threshold $T_s$ without significantly reducing the recall.

### 6.3.3 Mean Average Precision

To compare different detection methods on a dataset, the authors of *VOC* [40] established the mean average precision ($mAP$) that summarizes the detector quality in a single measure. Therefore, first, the average precision ($AP$) for each category is calculated as an approximation of the area under the interpolated PR curve for a given IoU threshold. The $mAP$ is the mean over the per-class $AP$ values.

**PR curve interpolation.** To interpolate the PR curve for every calculated precision value $p_j, j = 1, \ldots, n_D$, the interpolated precision is defined as

$$\hat{p}_j = \max_{i>j} p_i. \tag{6.8}$$

This makes the PR curve monotonically decreasing. We will explain in Section 6.4 why we believe that PR curve interpolation should be avoided. In this and the following section, to emphasize their differences, we denote the area under the non-interpolated PR curve as $AP$ and write $\overline{AP}$ for the area under the interpolated PR curve.

**PR curve subsampling.** Sometimes, instead of computing the real area under the interpolated PR curve, the PR curve is subsampled at $n_s$ equally distant recall values $r_j^{(s)} = j/(n_s - 1)$. The subsampled and interpolated average precision is then defined as the mean over the precision values at the sampling points:

$$\overline{AP}_{n_s} = \frac{1}{n_s} \sum_{j=0}^{n_s-1} p(r_j^{(s)}). \tag{6.9}$$

In the Pascal *VOC* challenge up to 2010, the interpolated and subsampled $AP$ with $n_s = 11$ was used. From 2010 to 2012, the subsampling was dropped *"to improve precision and ability to measure differences between methods with low AP"* [38, Section 3.4.1, p. 12] and, instead, the precision was evaluated at every recall step of the PR curve. This relates to the real area under the interpolated PR curve, and we refer to it as $\overline{AP}_{\text{real}}$.

### 6.3.4 COCO $AP$

In comparison to the 2012 Pascal *VOC* $mAP$, *COCO* introduced the following changes:

1. Rename $mAP$ to $AP$, and speak of a class-specific $AP$ whenever it is unclear whether the per-class $AP$ or mean $AP$ is meant. We will stick to the convention of *COCO* and only write $mAP$ if we explicitly want to make clear that the mean $AP$ in comparison to the the per-class $AP$ is meant.

2. While on *VOC* the single IoU threshold of 0.5 is used to account for inaccurate annotations, the measure on *COCO* takes again an average over the individual $mAP$ values for ten IoU thresholds $T_{\text{IoU}} \in [0.5 : 0.05 : 0.95]$ in order to highlight models with accurate localizations.

3. Since the *COCO* validation and test sets are huge, for some of the frequently occuring categories (e.g., *person*), it is computationally expensive to evaluate the PR curve at every recall step. This might be a reason why *COCO* goes back to the pre-*VOC*2010 subsampling of the PR curve at uniformly-spaced recall values. However, the number of sampling points $n_s$ is increased from 11 to 101 ($[0.0 : 0.01 : 1.0]$). We will discuss in Section 6.4.3 why subsampling leads to inaccurate approximations, especially for rare categories.

Both *COCO* and *LVIS* use the subsampled and interpolated average precision $\overline{AP}_{101}$. Note that within the *COCO* evaluation protocol,[2] the matching of predictions to GT annotations is done greedily with respect to the score of predictions. This means that if there are multiple correct predictions for one specific GT annotation, the one with the highest score is matched and all others are counted as FPs (if they do not also match with another GT annotation). Hence, the matching TP prediction is not always the one with the highest IoU to the GT annotation. Furthermore, it means that for two IoU thresholds $T^{(1)}_{\text{IoU}} > T^{(2)}_{\text{IoU}}$, the TP predictions with respect to $T^{(1)}_{\text{IoU}}$ are not always a subset of the TP predictions with respect to $T^{(2)}_{\text{IoU}}$. In particular, the minimum score of TPs for $T^{(1)}_{\text{IoU}}$ can be lower than the minimum score of TPs for $T^{(2)}_{\text{IoU}}$.

## 6.4 $AP^{\star}$ — Fixing Issues of $AP$ in its Current Form

In this section, we introduce a refined version of $AP$ that resolves some of its issues: $AP^{\star}$. In each of the following subsections, we explain the issue and our proposed solution. Some problems have a large effect on the measure and the corresponding model selection,

[2] https://github.com/cocodataset/cocoapi

while others do not change the values much in practice, but still have a theoretical justification.

### 6.4.1 Non-Penalized FPs and Bias Towards Recall

An excellent example for a case where FPs are not penalized is given in the appendix of the YOLOv3 preprint [157]. Non-penalized FPs are those *late* FPs that have a score that is smaller than the minimum score of all TPs for the given class. By adding those FP predictions, the corresponding recall remains the same and only the precision is decreasing. This does not change the area under the PR curve and thus the $AP$ is not affected by their presence.

With respect to a given validation set, in theory, non-penalized FPs could be easily filtered out by introducing a class-specific score threshold $T_s$: For each class, starting from 0.0 the score threshold is increased to remove detections until the maximum recall that is achieved with all available detections $D$, $r^{\star} = \max_{(p,r)\in D} r$, starts to decrease. Since this threshold corresponds to the score range between the lowest scoring TP and the previous FP with lower score, and it is chosen such that no TPs are suppressed, we denote it by $T_s^{TP}$. We write $T_s^{TP}(val)$ to emphasize that these class-specific score thresholds are hyperparameters and need to be calculated on the validation set.

There are two reasons why such a class-specific score threshold is not used in any of the current state-of-the-art detection methods:

1. $AP$ does not increase if we suppress late FPs.

2. Eventually, on another dataset (such as the private test set), there might be TPs with a lower score than $T_s^{TP}(val)$ that are suppressed. Hence, the usage of $T_s^{TP}$ could decrease the recall and, in turn, this will decrease $AP$s (especially interpolated $\overline{AP}$s, cf. Section 6.4.2).

An example from practice is given in Appendix B.1 of [62]: for the Mask R-CNN baseline on *LVIS*, increasing the maximum number of detections and successively reducing the score threshold (from an already low 0.05[3]) down to 0.0 leads to a remarkable improvement of the $AP$ from 15.7% to 20.9%. This finding is to be expected because $\overline{AP}_{101}(0.0) \geq \overline{AP}_{101}(T_s)$ for every $T_s > 0$. In other words, a lower score threshold will always result in an equal or higher $\overline{AP}_{101}$. However, the downside of using low score thresholds is that the number of predictions and especially the number of FPs is increasing substantially. For models like RetinaNet [120] that use a sigmoid activation for all categories and each class-specific anchor, this can lead to huge numbers of false positives (and possibly a much higher model runtime). A practical example for this issue on *D2S* is given in Sections 5.7 and 6.5. To summarize, the current definition of $AP$ has a bias towards recall because $AP$ will always reach its maximum for the maximum possible recall that the model can achieve, no matter how low the actual precision is sinking.

[3]Is a score threshold of 0.05 reasonable for an industrial application? How can we explain to the customer that we believe that a 5% confidence is trust worthy?

How can we incorporate the late FPs into the existing $AP$ measure and remove the recall bias? The goal is to reduce the number of late FPs, while keeping a good balance between precision and recall. Therefore, we propose to include the real final precision using all detections of the method $p(D)$ into the $AP$. In order to avoid a bias towards precision, we multiply it by the model's recall $r(D)$ and compute $AP^\star$ as:

$$AP^\star(\lambda) = \lambda \cdot AP + (1-\lambda) p_{n_D} r_{n_D}, \tag{6.10}$$

where we set $p_{n_D}$ to zero if no detection is present for a class. Although we could tune $\lambda$ according to our preference towards $AP$ or the obtained $(p, r)$ pair, we use $\lambda = 1/2$ and weight both components equally. We will thus omit $\lambda$ in our notation.

Geometrically, $p_{n_D} r_{n_D}$ is the area of the box that is enclosed by the actual precision and recall and thus both components of $AP^\star$ have the same dimension of an area with respect to precision and recall.

In practice, $AP^\star$ encourages to increase the score threshold $T_s$ until a good trade-off between precision and recall is found. The $p_{n_D} r_{n_D}$ term thus leads to a balance of both dimensions and penalizes methods for which either the precision or the recall is very low. The $AP$ term allows to reward models that have a good uncertainty estimation in the sense that for two models, for which the $p_{n_D} r_{n_D}$-term is equal, $AP^\star$ favors the one with a better ordering of TPs and FPs with respect to their score: the model where most TPs have higher scores than most FPs will have a larger area under the PR curve. Hence, models with fewer early FPs achieve higher $AP^\star$ values. Examples are given in the next sections.

Moreover, note that $AP^\star$ is suitable to compute an optimal score threshold $T_s^\star$

$$T_s^\star \in \left[s(d_{j^\star}), s(d_{j^\star+1})\right), \text{ where} \tag{6.11}$$

$$j^\star = \arg\max_j AP^\star(D_j). \tag{6.12}$$

This means that we can iteratively add predictions (with decreasing score) and compute $AP^\star(D_j)$ for each $D_j, j = 1, \ldots, |D|$. We can then choose $T_s$ somewhere between the score of the prediction for which we achieved the highest $AP^\star_{\max} = AP^\star(D_{j^\star})$ and the score of the next prediction. This is a clear benefit over $AP$, for which we always get the maximum value if we use all predictions.

### 6.4.2 Interpolation of the PR Curves

A possible intention behind the interpolation of the PR curve could be the following: Suppose you have a trained, final detector. To use it inside a machine, you still have to decide about your preference concerning the precision–recall trade-off. This can be tuned by setting a score threshold $T_s$ as explained in section 6.3. Effectively, each point on the PR curve belongs to one such threshold and yields one possible precision and recall pair $(p, r)$ on the evaluation set. In practice, if we had the choice, we would never opt for a sub-optimal PR pair. Instead, for a given recall $r$, one would increase the score threshold to increase the precision $p$ until $r$ decreases. Likewise, for a given precision $p$, one would decrease the score threshold to increase the recall $r$ until $p$ decreases. In other words, for two pairs of precision and recall, $(p, r)$ and $(p', r')$, if $p \geq p'$ and $r \geq r'$, clearly we choose the option $(p, r)$. For example, given the PR curve in Fig. 6.1, one would always choose $T_s$, such that the last two FPs with lowest scores are avoided, leading to $(p, r) = (0.667, 0.8)$.

But is this a reasonable explanation for PR curve interpolation? No. Because once we have decided for a pair $(p, r)$, our model will have certain numbers $n_{TP}$ and $n_{FP}$ that lead to precision $p$ and recall $r$, no matter what other $(p, r)$ trade-offs are possible. Moreover, within the $AP$ calculation, we use the area under the PR curve to summarize all $(p, r)$ pairs to obtain a measure that is less affected by the exact choice of $T_s$, but should emphasize the ordering of TPs and FPs with respect to their score.

Instead, what PR curve interpolation does is that it reduces the penalty of early FPs. This can be seen in the example in Fig. 6.1 or in the example in Fig. 6.2.

However, exactly those early FPs with high scores are the most painful and dangerous ones in an application. With high confidence, the model tells us something that is wrong. For example, the detector in an autonomously driving car is absolutely sure that the traffic lights show "green", but they don't. These early FPs are also one of the reasons that prevent people from using deep learning (DL) based systems. Today, DL-based models are often very bad when it comes to uncertainty estimation. The use of softmax and sigmoid as classifiers or activation functions often leads to overconfident models. For example, Ulrich et al. [194] (cf. Chapter 9) have shown that the score distribution of a DL-based detection method such as RetinaNet [120] is almost binomial, independent of the localization accuracy of the prediction.

In order to penalize early FPs more, we propose not to interpolate PR curves. In particular, the $AP$-term in $AP^\star$ is based on a non-interpolated PR curve.

In practice, the difference of $AP$ based on an interpolated or a non-interpolated PR curve becomes smaller for better models and where fewer early FPs are present.

### 6.4.3 Subsampling of the PR Curve

The statement of Pascal *VOC* authors [41] about subsampling of PR curves is somewhat ambiguous: *"The intention in interpolating [they mean what we call subsampling] the precision-recall curve was to reduce the impact of the 'wiggles' in the precision-recall curve, caused by small variations in the ranking of examples. However, the downside of this interpolation was that the evaluation was too crude to discriminate between the methods at low AP."* [41, p. 313]. We are not sure what exactly is meant by *wiggles*, since the PR curve is a step-function. Using the PR curve interpolation as proposed by *VOC*, the curve is even monotonically decreasing.

We claim that PR curve subsampling with $n_s$ sampling points is not advisable for a small evaluation set with $n_{GT} < 1000$.[4]

Let us consider the example in Fig. 6.3. The left plot shows the interpolated PR curve subsampled at 11 sampling points $[0.0 : 0.1 : 1.0]$ like in the first years of the *VOC*

[4]The threshold 1000 is arbitrarily chosen, but from 1000 onwards the error that is done by subsampling at least only leads to changes of $AP$ below 0.001

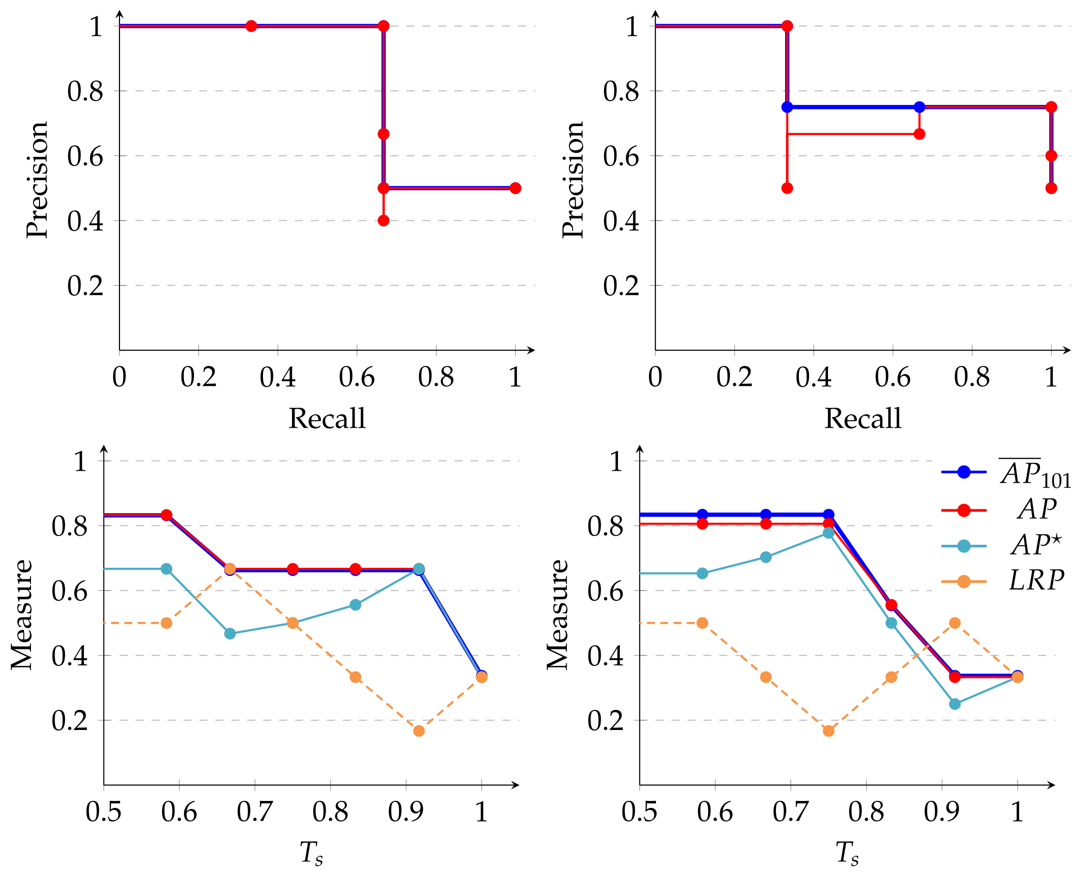


**Figure 6.2: PR Curve Interpolation and non-penalized FPs.** $n_{GT} = 3$, the order of detections is (TP, TP, FP, FP, FP, TP) (*left*) and (TP, FP, TP, TP, FP, FP) (*right*), both with scores (1.0, 0.92, 0.83, 0.75, 0.66, 0.58).
(*Top*) PR curves, with all $(p, r)$ pairs indicated by the markers. The non-interpolated PR curve is shown in red and the interpolated PR curve is shown in blue. For the left detector, interpolation does not change the PR curve and we get $\max \overline{AP}_{101} = 83.2\%$ and $\max AP = 83.3\%$. The difference between $\overline{AP}_{101}$ and $AP$ is due to PR curve subsampling. For the right detector, interpolation reduces the penalty of the first FP, and we get $\max \overline{AP}_{101} = 83.4\%$ and $\max AP = 80.6\%$. This shows that without PR curve interpolation $AP$ becomes a better measure for the model's uncertainty estimation. $AP$ is lower because for the right detector the first FP occurs earlier and with a higher score than most TPs.
(*Bottom*) the $\overline{AP}_{101}, AP, AP^\star$, and $LRP$ measure is plotted against the score threshold $T_s$. Note that for LRP the values are not normalized to $[0, 1]$ and minimum values are optimal. While both $\overline{AP}_{101}$ and $AP$ indicate that a low threshold — $T_s \in [0, 0.58]$ for the left example and $T_s \in [0, 0.75]$ for the right example — is optimal, $LRP$ and $AP^\star$ reach their optima if late FPs are suppressed. Moreover, $AP$ and $\overline{AP}_{101}$ remain constant although FPs are suppressed with increasing $T_s$.
In comparison to $LRP$, where $LRP_{\min} = 16.7\%$ for both detectors, $AP^\star$ clearly favors the right detector with $AP^\star_{\max} = 77.8\%$ vs. $AP^\star_{\max} = 66.7\%$.
Overall, the right detector yields a better result since late FPs can be filtered out by setting $T^\star_s$ as score threshold which leads to a well balance between precision and recall, as indicated by $AP^\star_{\max}$.

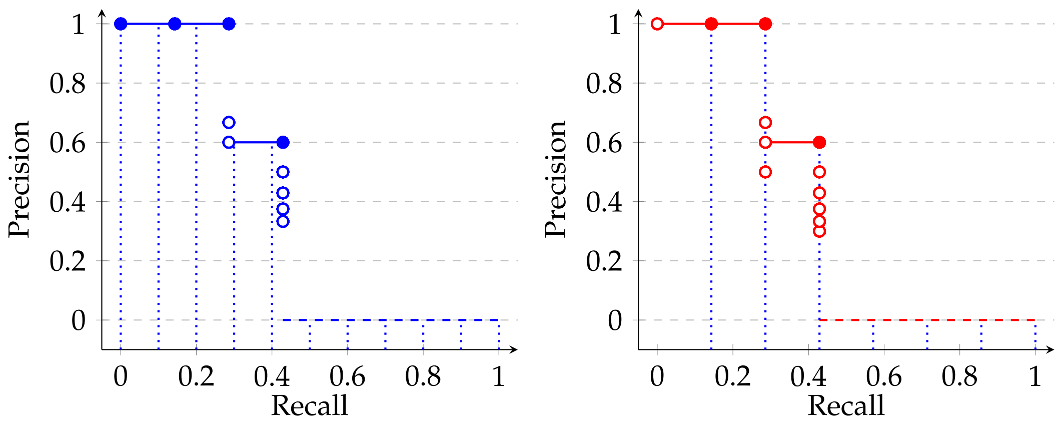


**Figure 6.3: Implications of PR curve subsampling.** $n_{GT} = 7$; the order of detections is (TP, TP, FP, FP, TP, FP, FP, FP, FP, FP). The PR curve is shown interpolated (*left, blue*), and without interpolation (*right, red*), a dashed line is used for recalls higher than $r_{n_D}$ to indicate that the precision is set to zero for those sampling points. Subsampling positions are indicated with blue dotted lines. The axes are positioned at (-.05, -.1) for better visibility. The real area under the PR curve is $AP_{\text{real}} = 2/7 + 3/5 \cdot 1/7 = 0.371$.
(*Left, blue*) *VOC*-style subsampling with $n_s = 11$, leading to $\overline{AP}_{11} = 1/11 \cdot (3 \cdot 1 + 2 \cdot 3/5 + 5 \cdot 0) = 0.382$; $\overline{AP}_{101} = 0.370$.
(*Right, red*) The proposed non-interpolated and precisely subsampled PR Curve with $AP_{n_{GT}} = 1/7 \cdot (2 \cdot 1 + 3/5 + 4 \cdot 0) = AP_{\text{real}}$.

challenge. Interpolation leads to the fact that the precision value at the first recall step is evaluated twice, if the first detection is a TP. Additionally, since the sampling points do not coincide with the recall steps, different parts of the PR curve get more or less weight, just by chance.

Subsampling at $r_0 = 0.0$ can be avoided because this doubles the impact of the first predictions with highest scores. Moreover, if interpolation of the PR Curve is switched off, the precision $p(r_0)$ might be undefined if $d_1$ is a TP. Otherwise, if the very first predictions are FPs, they are already included in the precision at the first recall step $p(r_1)$ that becomes lower the more FPs occur in the beginning.

We propose to either increase the number of subsampling points to $n_s \geq 1000$ or to dynamically set the number of subsampling points to $n_s = n_{GT}$ and use the recall steps as sampling points $r_i^{(s)} = i/n_{GT}, i = 1, \ldots, n_s$, to obtain $AP_{n_{GT}}$. It is shown in the right plot, that this leads to the real area under the curve $AP_{n_{GT}} = AP_{\text{real}}$. A compromise for efficient evaluation could be to use $n_s = \min(1000, n_{GT})$. In practice, the effects of this change are often marginal, especially if $n_{GT} > 100$. However, summing them up for many categories and different IoU thresholds can make a difference.

As we have shown, the computational cost is not increased. In fact, for categories with small evaluation sets it can even be reduced. Therefore, from a theoretical point of view, the area under the PR curve should be calculated accurately using $AP = AP_{n_{GT}}$. Another example where $AP$ differs from $\overline{AP}_{101}$ is shown in Fig. 6.2.

### 6.4.4 Too High IoU Thresholds

When *AP* was introduced as an evaluation metric in the Pascal *VOC* challenge, the authors decided to use only one low IoU threshold of 0.5 *"set deliberately low to account for inaccuracies in bounding boxes in the ground truth data"* [40, p. 314]. In the *COCO* challenge, the *AP* was made more strict by averaging over 10 IoU thresholds $[0.5 : 0.05 : 0.95]$, but the authors do not explicitly explain this choice (as also pointed out in [157]).

The intention of averaging the *AP* over several iteratively increasing IoU thresholds emphasizes the importance of very accurate mask predictions. However, we claim that this is only meaningful if the given GT annotations are as accurate as the used IoU thresholds, in the sense of reproducibility. If annotations are not accurate, meaning that the labels of two different annotators do not exceed an IoU of, e.g., 0.7, it is hard to tell which of the two annotations are correct. Hence, it is likely that the prediction of a model is different from the chosen "ground truth" and will also not exceed the 0.7 IoU threshold, which does not necessarily mean that the model was wrong. However, the model is penalized by the current definition of *COCO AP* because the prediction counts as FP for all IoU thresholds greater than or equal to 0.7.

In the *COCO* publication [118], an analysis of the precision and recall of annotators was carried out on a subset of the images: A high recall could be confirmed, but the maximum precision of workers did not exceed an IoU of 0.85, while the mean precision of workers was only around 0.75 IoU. Therefore, the choice of high thresholds of 0.8 and above still seeks for an explanation.

On *LVIS* [62], a reproducibility study was carried out by running all 5$k$ validation images through the annotation process twice. The mean IoU between the two runs is 0.85, and only for approximately 5% of instances the (multiple) human annotators could exceed the 0.95 IoU threshold. Thus, also on *LVIS*, it is not clear why such high IoU thresholds should help to identify the best among several detection methods.

Therefore, the maximum IoU threshold used for the evaluation should be lowered.

If this is not desired, another approach is the following: The same reproducibility analysis that was done for the validation set should also be done for the test set annotations. The (mean) achieved IoUs for each annotation between different runs could then be stored. A low IoU shows that the object is somewhat ambiguous or difficult to annotate and we should not expect from any method to do "better" than the achieved IoU. During the evaluation, each individual annotation could then only be evaluated up to the stored IoU threshold and ignored for all higher IoU thresholds. The analysis of such a change in the evaluation protocol goes beyond the scope of this work, but is an interesting topic for future research. In particular, Gupta et al. [62] empirically found a bias of *AP* with respect to the evaluation set size. This has to be kept in mind if evaluation sets become smaller and smaller with increasing IoU thresholds.

### 6.4.5 *AP* has a Bias for Instance Size

Generally, IoU has the advantage that it is invariant to the scale of objects. This holds as long as both ground-truth annotations as well as predictions are given in a subpixel-

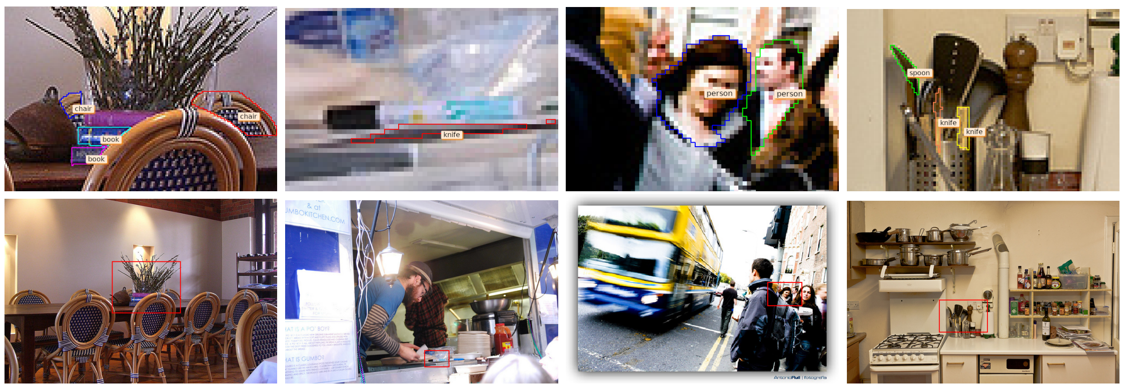


**Figure 6.4: Small instance examples from *COCO*.** (*Top*) Zoomed images with very small instances below $32^2$px. Instance masks are indicated by colored region boundaries, a class label is shown at the center of each instance. (*Bottom*) The corresponding full images with the zoomed image part indicated by a red box. Very small instances are difficult and often ambiguous to annotate. Moreover, a small number of false mask pixel predictions has a large influence on the IoU. Hence high IoU thresholds should be avoided for small instances (*Best viewed digitally and with zoom*).

precise format. While this might be true for the ground-truth annotations, for current methods such as Mask R-CNN [70], at least the intermediate predictions are predicted and trained in a pixel-precise grid. The final mask is obtained by translation and scaling to the subpixel-precise final box prediction but still is a pixel-precise region.

Especially for pixel-precise predictions and labels, high IoU thresholds are biasing categories with smaller instances towards lower $AP$ values. The same threshold is acting more strictly on small instances, compared to large instances. For example, consider two boxes $A$ and $B$ of size $32^2$ and $16^2$, respectively. Adding a one-pixel boundary to both boxes to obtain $\hat{A}$ and $\hat{B}$, we can calculate $\text{IoU}(A, \hat{A}) = 0.94$ and $\text{IoU}(B, \hat{B}) = 0.89$.

Very small instances have a much higher ambiguity when being annotated with masks or boxes. Including several pixels close to the object's boundary to the annotation or not can make a large difference. Hence, for these objects the reproducibility is inherently lower. We claim that when users are trying to detect very small objects in rather low-resolution images, they should not expect to get a localization precision above, e.g., 0.75 IoU. Some examples of small instances with an area below $32^2$ px are shown in Fig. 6.4.

To compensate this problem, we propose to reduce the maximum evaluation IoU threshold for small objects of size $\leq 32^2$ px. Another possibility would be to use the same approach that has been indicated in the previous subsection: One could relabel the images of the evaluation set and check the annotation reproducibility for each instance by computing the IoU of the initial annotations with the new annotations. The maximal IoU threshold that is used during the evaluation could then be set to the mean IoU of the annotations of different annotators. However, this approach is very costly and the ambiguity of mask labels remains.

To avoid effects of very small instances, our experiments are carried out on *D2S*, where the annotations are very accurate and very small instances are not present.

## 6.5 Experiments

We do our experiments on *D2S* (cf. Chapter 5). *D2S* is well suited since the annotations are accurate such that an evaluation with respect to high IoU thresholds is meaningful. Moreover, *D2S* is challenging and far from being solved.

We mainly look at box detection methods here, but it is straightforward to generalize the results to instance segmentation methods because both are structurally similar.

To see the reason why $\overline{AP}_{101}$, $LRP$, or $AP^{\star}$ might be low or high, we also compute the average recall $AR$ and mean precision $mPr$

$$AR = \frac{1}{|\{T_{\text{IoU}}\}|} \sum_{\{T_{\text{IoU}}\}} AR_{T_{\text{IoU}}} = \frac{1}{|\{T_{\text{IoU}}\}|} \sum_{\{T_{\text{IoU}}\}} \frac{1}{C} \sum_{c} r_c, \tag{6.13}$$

$$mPr = \frac{1}{|\{T_{\text{IoU}}\}|} \sum_{\{T_{\text{IoU}}\}} mPr_{T_{\text{IoU}}} = \frac{1}{|\{T_{\text{IoU}}\}|} \sum_{\{T_{\text{IoU}}\}} \frac{1}{C} \sum_{c} p_c. \tag{6.14}$$

If we look at a single IoU threshold, we will omit the $T_{\text{IoU}}$-threshold subscript in $AR$ and $mPr$ for convenience. Moreover, in this section we will write $AP$ for the *COCO* $AP = \overline{AP}_{101}$ to avoid confusions with other chapters and publications.

When not explicitly mentioned otherwise, the IoU threshold for TPs is set to 0.5, and the global minimum confidence to 0.05 in all plots independent of other stricter class-specific score thresholds that might be applied. All models have been trained and evaluated on images with resolution $512 \times 384$.

### 6.5.1 Global Confidence Threshold

Setting a global minimum confidence threshold $T_s^G$ is widely used when models are used within an application. It is one of the main reasons why we actually build models that return a confidence for each result. By looking at the score, we can suppress results where the model is unsure. Unfortunately, deep-learning-based detectors or classification models are often over-confident. That means, they predict wrong results with high confidence values. One of the reasons is the use of softmax predictions or sigmoid activations that are trained to predict a confidence of 1.0 for the ground truth class. Hence, although the model might already predict the correct class with high confidence, the loss is still positive and drives the model towards larger and larger activations. This makes it difficult to interpret the scores of DL-based models and also makes it hard to set a proper score threshold. In other words, DL-based models are often bad in estimating their uncertainty. This emphasizes the need to incorporate properly tuned score thresholds into the evaluation protocol. At the time of writing this work, the opposite happens: some of the authors of the best methods in the 2020 *LVIS* challenge[5] report that they reduce the score threshold from an already low 0.05 to 0.0.

In the previous sections, we have seen that $AP$ does not encourage to set any score threshold. $AP$ is always maximal if a global score threshold $T_s^G$ is set to 0.0. Fig. 6.5 confirms that $AP$ is monotonically decreasing with increasing $T_s^G$. Moreover, as shown

[5] https://www.lvisdataset.org/challenge_2020

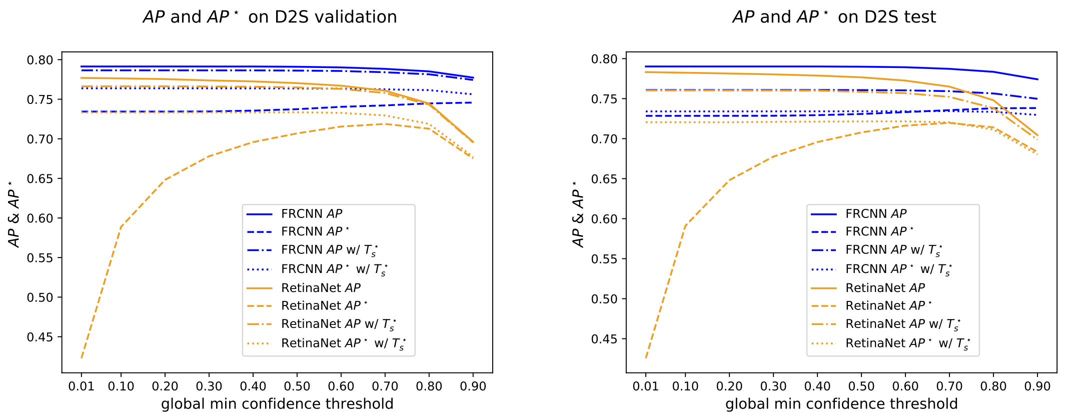


**Figure 6.5: Influence of $T_s^G$ for different measures on *D2S*.** Box detection results of FRCNN and RetinaNet on the validation and test set. Note that here the $AP$ and $AP^\star$ values, averaged over all IoU thresholds, are shown, but $T_s^\star(val)$ was computed at IoU 0.5.

in Table 6.1, the $AP$ values of FRCNN (F) and RetinaNet (R), with 79.0% and 78.3%, respectively, are almost on the same level on the test set. However, their mean precision at IoU 0.5 is very different, with 83.8% vs. 35.1%, respectively. At IoU threshold 0.5, RetinaNet even has a higher $AP$ value than FRCNN. While overall RetinaNet has 681 more TPs than FRCNN on all the test images, it also has 92 499 FPs more than FRCNN. With this low confidence threshold of $T_s = 0.05$, for every TP of RetinaNet, the model also predicts almost two FPs, on average. In contrast to $AP$, $AP^\star$ reveals this difference with values of 72.8% and 53.2% overall, and 81.0% and 59.9% at IoU 0.5, for FRCNN and RetinaNet, respectively.

Intuitively, the easiest way to filter out weak predictions is to suppress them by setting a global score threshold $T_s^G$. This works if the scores are well calibrated in the sense that for predictions with higher quality also their respective score is higher and for wrong predictions, either due to a bad localization or due to the wrong class, the score should be lower. We also see in the score histograms for RetinaNet and FRCNN in Fig. 6.8 that this holds for most of the false positives, especially in the case of RetinaNet.

The left plot in Fig. 6.5 shows that for both FRCNN and RetinaNet, we can determine an optimal global score threshold on the validation set if we use $AP^\star$ as the underlying quality measure. Evaluating $AP^\star$ (as average over all IoU thresholds) at the global score thresholds $T_s^G \in \{0.01, 0.1, 0.2, \ldots, 0.9\}$, we see that it is maximal if $T_s^G$ is set to 0.7 and 0.9 for RetinaNet and FRCNN, respectively. For example, for RetinaNet, on the validation set this leads to an increase of $AP^\star$ from 52.9% to 71.9% and of $mPr$ from 33.1% to 81.9%, respectively. At $T_{\text{IoU}} = 0.5$, the number of false positives can be reduced from 28 563 to 1799, while $T_s^G$ only leads to a small reduction of true positives from 14 256 to 13 795. Although this yields a much better precision–recall trade-off, $AP$ is significantly reduced from 77.7% to 76.1% on the validation set. For FRCNN, the changes are less pronounced, but still, using $T_s^G$, at $T_{\text{IoU}} = 0.5$, $mPr$ is increased from 84.5% to 90.4% while $AR$ is only slightly reduced from 89.9% to 87.8%. Overall, $AP^\star$ increases from 73.4% to 74.6%.

The right plot in Fig. 6.5 shows the impact of setting a global score threshold on the test set. With this, we evaluate how well the thresholds that we have computed on the validation set generalize. The figures confirm that the evaluation on the validation set is just an approximation of the goodness of the model on unseen data. Therefore, it is crucial that we always use the validation set to compute optimal thresholds and compare the models with fixed hyperparameters on the test set. Of course, the optimum thresholds on the validation set might differ from the generally unknown optimal thresholds on the test set. Generally, a larger validation set should lead to optimal hyperparameters that are close to the optimal hyperparameters on unseen data, but only if there is no bias between the validation and test set. Another approach is to use cross-validation on the validation set to determine the best score thresholds, but this goes beyond the scope of this work.

### 6.5.2 $T_s^\star$ Per-Class Thresholds Based on $AP^\star$

We have seen that to optimize $AP^\star$, setting a score threshold is crucial. We have also seen that using a global score threshold, the number of FPs can be reduced significantly. But is one global threshold meaningful for a large number of different categories? Some classes might be easier to find than others, inter-class variations might be small for a few categories, but large for others, some classes might only appear in a specific context which might be helpful for the classification of these objects. That is why the score distributions of different categories are very different as depicted in Fig. 6.6.

There are many possible heuristics that could be used to define per class score thresholds. In the following, we will compare the thresholds $T_s^{TP}, T_s^\star$, and $T_s^{LRP}$ (6.3) that all can be easily computed from a given validation set.

With respect to their underlying measure, i.e., recall for $T_s^{TP}$, $LRP$ for $T_s^{LRP}$ and $AP^\star$ for $T_s^\star$, all three thresholds are optimal. However, this only holds for the evaluation set and IoU threshold where $T_s$ has been determined. For example, for a particular class, it might hold that $T_s^{TP}(val) > T_s^{TP}(test)$, such that some of the lowest scoring TPs in the test set will be filtered out. Also for the other measures, it might be the case that they are not optimal on the unseen test set. Hence, we also want to evaluate how good class-specific score thresholds generalize and if they are useful in practice.

Fig. 6.6 reveals that the per-class thresholds are indeed very different for different categories, both for FRCNN and RetinaNet. While $T_s^{TP}$ is the most conservative threshold that keeps almost all TPs, of course at the same time many FPs remain. On the contrary, $T_s^{LRP}$ is always the highest threshold of the three thresholds. In theory, one could also implement $T_s^{FP}$ as the highest score of FPs such that all FPs are suppressed and maximum precision is obtained. The figure shows however, that for almost all categories this would reduce the recall significantly. Looking at the plots, a compromise between $T_s^{TP}$ and $T_s^{FP}$, e.g., selecting the 5% quantile of TPs or the 90% quantile of FPs (sorted increasingly by scores), respectively, could be a reasonable choice. However, we aim for models that produce a score distribution where FPs and TPs are separable, and in this case both choices lead to suboptimal results and thus they generally are not meaningful. $T_s^\star$ typically resides between $T_s^{TP}$ and $T_s^{LRP}$, and sometimes is equal to either of them. Clearly, $T_s^{TP} \leq T_s^\star$ always holds because to optimize $AP^\star$ we need to filter out at least all

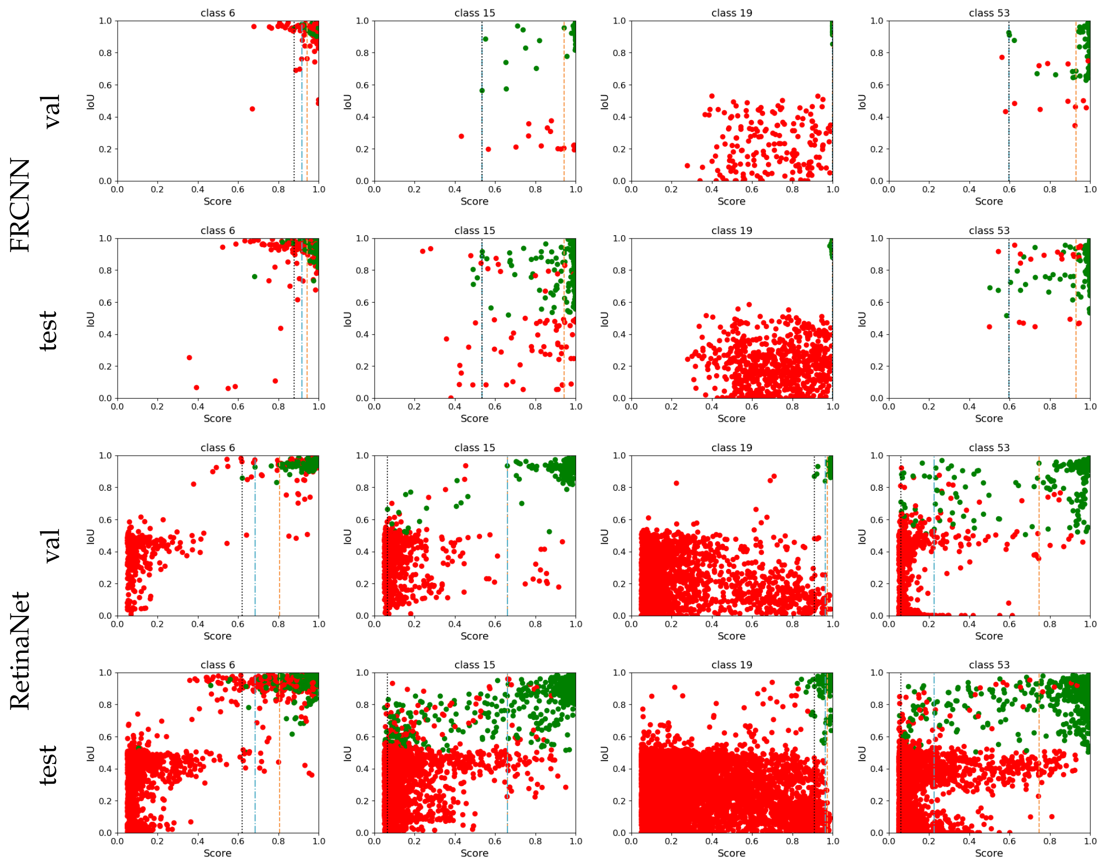


**Figure 6.6: Score vs.** IoU **distribution of predictions with class-specific score thresholds on *D2S*.** Predictions for some selected classes on the validation and test set of FRCNN and RetinaNet, respectively. The optimal score thresholds $T_s^{TP}(val)$, $T_s^{\star}(val)$ and $T_s^{LRP}(val)$ are indicated by black-dotted, blue-dash-dotted and orange-dashed vertical lines, respectively. The plots for the test set show that they do not always generalize well. RetinaNet scores are spread a lot more across the interval $[0, 1]$ than for FRCNN. (*Best viewed digitally and with zoom*).

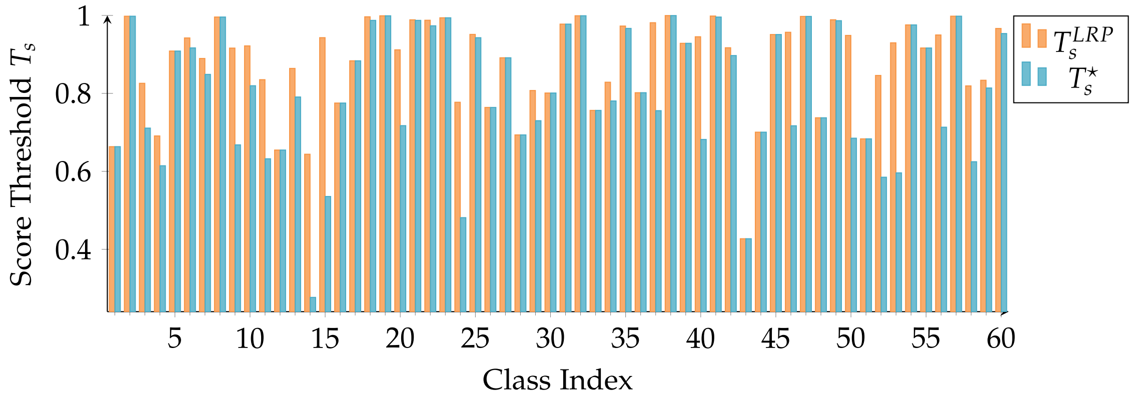


**Figure 6.7: Comparison of $T_s^{\mathrm{LRP}}(val)$ and $T_s^{\star}(val)$ for FRCNN R101 S.** For many categories the computed optimal scores thresholds based on LRP and $AP^{\star}$ are equal. For others the difference is large due to the influence of the $AP$ and the $\mathrm{LRP}_{\mathrm{IoU}}$ terms. Note that the score thresholds are mainly determined by the lowest scoring TPs and their scores are outliers of a relatively dense score distribution. Very high score thresholds close to 1.0 do not generalize well to the test set. Also see Fig. 6.6.

| Model | IoU[0.5:0.95] | | | | | IoU[0.5] | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $AP$ | $AP^\star$ | $AR$ | $mPr$ | $moLRP$ | $AP$ | $AP^\star$ | $AR$ | $mPr$ | $moLRP$ | $TP$ | $FP$ |
| Validation set | | | | | | | | | | | | |
| R | 77.7 | 52.9 | 82.9 | 33.1 | 0.77 | 86.5 | 59.4 | 90.3 | 35.9 | 0.70 | 14256 | 28563 |
| R $T_s^G$ | 76.1 | 71.9 | 80.8 | 81.9 | 0.47 | 84.2 | 80.9 | 87.4 | 88.7 | 0.32 | 13795 | 1799 |
| R $T_s^{TP}$ | 77.6 | 66.9 | 82.8 | 65.0 | 0.55 | 86.5 | 75.5 | 90.3 | 70.5 | 0.42 | 14256 | 8602 |
| R $T_s^{LRP}$ | 75.7 | 73.2 | 80.2 | 85.7 | 0.45 | 83.7 | 82.2 | 86.7 | 92.7 | 0.29 | 13721 | 957 |
| R $T_s^\star$ | 76.6 | 73.3 | 81.5 | 83.6 | 0.45 | 85.0 | 82.7 | 88.3 | 90.7 | 0.29 | 13958 | 1386 |
| F | 79.1 | 73.4 | 84.4 | 79.2 | 0.45 | 86.7 | 81.4 | 89.8 | 84.5 | 0.32 | 14179 | 3142 |
| F $T_s^G$ | 77.7 | 74.6 | 82.8 | 85.0 | 0.42 | 85.0 | 82.3 | 87.8 | 90.4 | 0.28 | 13879 | 1558 |
| F $T_s^{TP}$ | 79.1 | 75.7 | 84.4 | 84.2 | 0.41 | 86.7 | 83.8 | 89.8 | 89.7 | 0.27 | 14179 | 1696 |
| F $T_s^{LRP}$ | 78.1 | 76.3 | 83.1 | 88.1 | 0.40 | 85.4 | 84.2 | 88.1 | 93.6 | 0.25 | 13948 | 844 |
| F $T_s^\star$ | 78.7 | 76.4 | 83.7 | 87.0 | 0.40 | 86.1 | 84.4 | 88.9 | 92.6 | 0.26 | 14071 | 1040 |
| Test set (using $T_s(val)$) | | | | | | | | | | | | |
| R | 78.3 | 53.2 | 82.8 | 32.9 | 0.77 | 87.2 | 59.9 | 90.5 | 35.8 | 0.70 | 45107 | 104633 |
| R $T_s^G$ | 76.5 | 72.0 | 80.5 | 81.5 | 0.47 | 84.6 | 81.0 | 87.1 | 88.3 | 0.32 | 43238 | 6472 |
| R $T_s^{TP}$ | 77.8 | 66.8 | 82.1 | 65.1 | 0.56 | 86.5 | 75.3 | 89.4 | 70.5 | 0.43 | 44576 | 27341 |
| R $T_s^{LRP}$ | 74.7 | 71.7 | 78.2 | 85.1 | 0.46 | 82.4 | 80.4 | 84.4 | 91.9 | 0.31 | 41986 | 3715 |
| R $T_s^\star$ | 76.0 | 72.1 | 79.8 | 82.7 | 0.47 | 84.1 | 81.0 | 86.4 | 89.4 | 0.31 | 43085 | 5567 |
| F | 79.0 | 72.8 | 83.8 | 78.5 | 0.46 | 86.8 | 81.0 | 89.4 | 83.8 | 0.33 | 44426 | 12134 |
| F $T_s^G$ | 77.4 | 73.8 | 82.0 | 84.4 | 0.43 | 84.8 | 81.7 | 87.1 | 89.8 | 0.29 | 43244 | 5834 |
| F $T_s^{TP}$ | 77.4 | 73.6 | 82.1 | 83.7 | 0.43 | 84.9 | 81.6 | 87.3 | 89.1 | 0.30 | 43516 | 6209 |
| F $T_s^{LRP}$ | 75.3 | 73.2 | 79.5 | 87.8 | 0.42 | 82.4 | 80.9 | 84.4 | 93.1 | 0.29 | 42080 | 3159 |
| F $T_s^\star$ | 76.1 | 73.4 | 80.4 | 86.4 | 0.43 | 83.3 | 81.2 | 85.4 | 91.8 | 0.29 | 42556 | 3967 |

**Table 6.1: Results for different class-specific score thresholds.** $AP$ reaches its maximum without class-specific or global score thresholds, however with low $mPr$ for RetinaNet (R) and FRCNN (F), respectively. This is indicated by low $AP^\star$ and high $moLRP$. Models with class-specific score thresholds lead to large improvements of the precision. $T_s^\star$ leads to the best balance between $AP$, $AR$ and $mPr$. For FRCNN, $T_s^\star$ does not generalize well from the validation to the test set, as $AR$ is reduced too much. However, the model has a good balance between $AR$ and $mPr$ and obtains roughly the same $moLRP$ as for $T_s^{LRP}$.

FPs with smaller score than the smallest score of all TPs. Otherwise, the precision would be unnecessary low without increased recall. On the other hand, $T_s^{LRP} \geq T_s^\star$ does not necessarily hold. $LRP$ contains components that reflect precision, recall, and localization quality of TPs and hence also depends on the IoU distribution of TPs. As shown in Fig. 6.7, for FRCNN R101 S on $D2S$, $T_s^{LRP}(val) \geq T_s^\star(val)$ holds for all classes.

All three thresholds are mainly influenced by the lowest scores of TPs. For some categories these scores are quite sparse and hence the difference between $T_s^{TP}$, $T_s^{LRP}$, and $T_s^\star$ can be large. Moreover this leads to remarkable differences between $T_s(val)$ and $T_s(test)$ for all three measures.

Table 6.1 shows a comparison of different evaluation measures for the global and all three class-specific thresholds. All score thresholds lead to a remarkable reduction of FPs without a large reduction of $AR$. We see that for FRCNN the class-specific score thresholds do not generalize as well as the global threshold for which we obtain the best $AP^\star$ result of 73.8%, which is a 1% improvement over the baseline result at $T_s^G = 0.05$. Although $T_s^\star$ does not generalize very well, it still leads to overall $AP^\star = 73.4\%$, an improved $mPr$ of 86.4%, with 688 less TPs and 1867 fewer FPs than the model using optimal $T_s^G(val)$ at IoU 0.5. $T_s^{LRP}$ has a slightly lower $moLRP$ value with higher $mPr$ but lower $AR$ and $AP^\star = 73.2\%$. Generally, it is hard to tell which threshold is giving the best compromise of recall and precision, but surely for FRCNN it is a problem that the class-specific score thresholds do not generalize well to the test set.

For RetinaNet, scores are a bit more spread over the range from zero to one. This leads to the fact that $T_s^\star$ clearly leads to the best compromise between $AP$, $AR$ and $mPr$ and overall $AP^\star$ of 72.1%. At IoU 0.5, the number of FPs can be reduced from 104 633 to 5567 on the test set. This means that while with the initial low score threshold $T_s^G = 0.05$ more than two thirds of the predictions were FPs, now only roughly every tenth prediction is a FP. As we have seen, $T_s^{LRP}$ thresholds are higher and thus increase the precision even a bit further, while $AR$ is reduced. As for FRCNN, $T_s^{LRP}$ leads to a slightly lower $moLRP$. For RetinaNet, all thresholds generalize somewhat better than for FRCNN. However, RetinaNet performs slightly worse than FRCNN independent of the chosen score threshold.

Looking at the results on the validation set, we get an impression how good the score thresholds could work if they were better predictors for the score distributions on the test set. Further, we believe that with better calibrated scores more FPs could be filtered out without a reduction of recall.

The confidence vs. IoU plots in the first row of Fig. 6.8 show the benefit of a class-specific score threshold like $T_s^\star$ above a global score threshold $T_s^G$. $T_s^\star$ is able to filter out a significant portion of FPs in the bottom right part of the plot, while preserving most of the TPs above. $T_s^G$ would just be a vertical line within this plot that filters out all predictions on the left. FPs in the bottom right of the line would thus remain. The confidence and IoU histograms in the second and third row underline that mostly FPs are filtered out and the reduction in TPs is almost negligible. They also show the different score distributions of RetinaNet and FRCNN. A major difference between the two models is that RetinaNet uses a sigmoid to predict the score, while FRCNN uses a softmax layer.

The barplots in the bottom row of Fig. 6.8 reveal that $T_s^\star$ mostly filters out FPs of type background, duplicate, localization, or multiple, but that most of the FPs with respect to a wrong class prediction remain. For FRCNN, a majority of these FPs have a score in the interval $(0.95, 1.0]$. Thus, it is very difficult to filter them out solely based on an improved score threshold.

### 6.5.3 Qualitative Results

A number of qualitative examples from the *D2S* test set are shown in Fig. 6.9. The application of $T_s^\star$ clearly helps to suppress many FPs within the background, especially for backgrounds that are not part of the training set (*1st, 8th, and 9th row for FRCNN, almost all rows for RetinaNet*). Additionally, FPs due to clutter objects can be removed (*1st and 7th row*). Generally, FRCNN is the better model with fewer FPs, but using $T_s^\star$ the results of FRCNN and RetinaNet are very close to each other.

Sometimes, the application of $T_s^\star$ removes a TP instead of a FP (*6th row*). Moreover, if no TP is present for an object, of course, the introduction of a score threshold does not

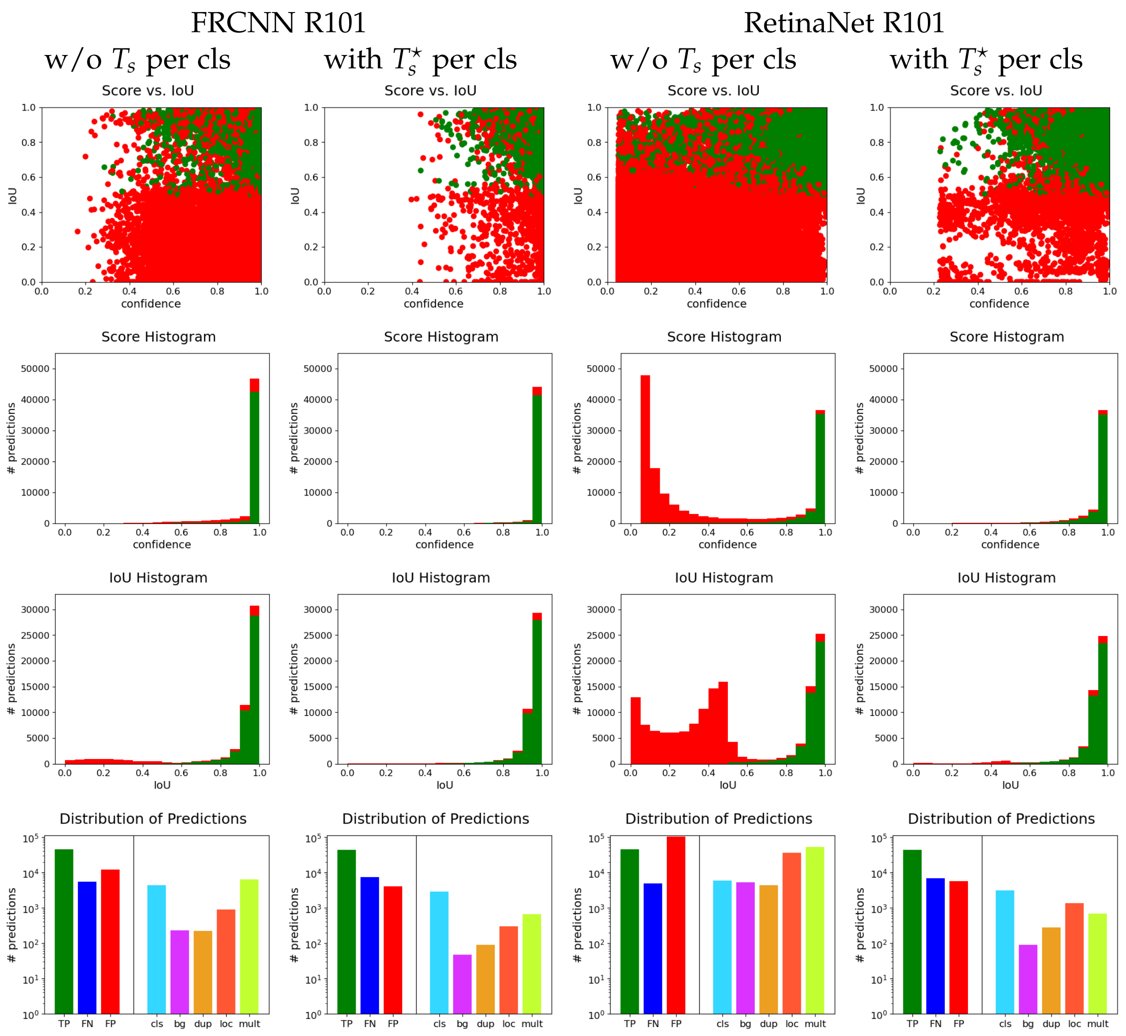


**Figure 6.8: Distribution of predictions with and without $T_s^\star(val)$ on *D2S* test.** The figure shows a comparison of the distribution of detection model results, with and without optimal per class min score thresholds $T_s^\star$. (*Top row*) Confidence vs. IoU: With class-specific score thresholds $T_s^\star$ many FPs are filtered out that have a larger score than the lowest scores of TPs. (*2nd and 3rd row*) Score and IoU histograms. Applying $T_s^\star$ only leads to a negligible reduction of TPs, but to a remarkable reduction of FPs, also for those with high IoUs and scores. The difference is more severe for RetinaNet. (*Bottom row*) Failure type distribution of FPs. Using $T_s^\star$ leads to a significant reduction of FPs in the categories background, duplicates, localization and multiple. Note the log-scale of the axis. For some categories the reduction is more than an order of magnitude.

help to predict another class (*2nd, 4th, 5th, or 7th row*). Hence, in future work one could incorporate the class-specific score threshold already within an earlier stage of the model, such that the model can predict the second best alternative, which might be correct.

Failure cases that remain are often due to a wrong class with high score (*bottles in 7th row*) or due to inaccurate box predictions (*4th or 6th row*). Overall, we believe that a better uncertainty estimation with more reliable confidences could help a lot to improve the results. This should also lead to a better generalization of calculated score thresholds from the validation to the test set.

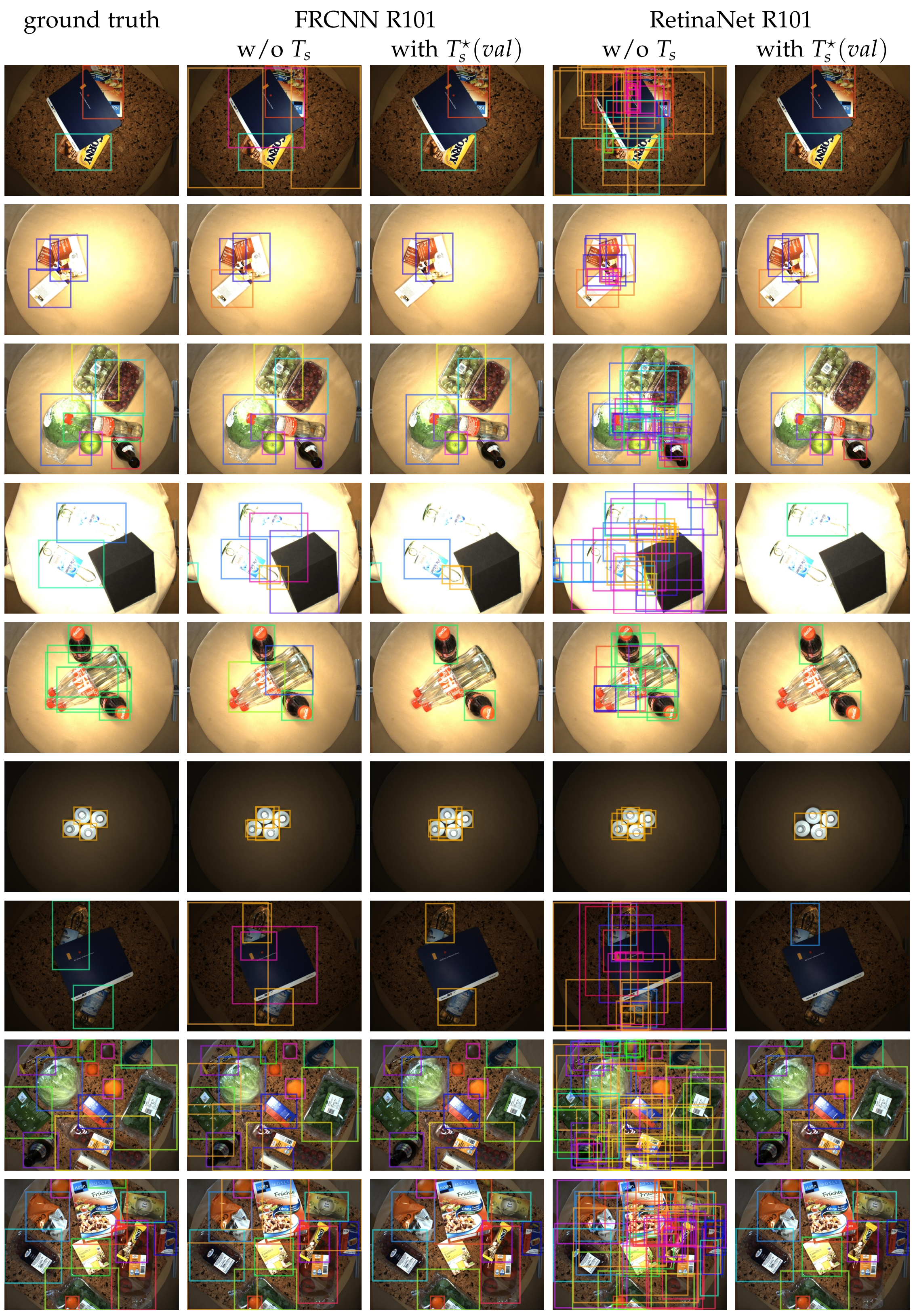


**Figure 6.9: Qualitative results on *D2S* test.** $T_s^\star$ helps to filter out many FPs within the background or with bad localizations. Mainly FPs due to wrong class or due to touching objects remain (*best viewed digitally and with zoom*).

| Model | IoU[0.5:0.95] | | | | | IoU[0.5] | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $AP$ | $AP^{\star}$ | $AR$ | $mPr$ | $moLRP$ | $AP$ | $AP^{\star}$ | $AR$ | $mPr$ | $moLRP$ | $TP$ | $FP$ |
| Validation set | | | | | | | | | | | | |
| R | 80.4 | 57.8 | 85.0 | 40.8 | 0.71 | 87.2 | 63.2 | 90.7 | 43.4 | 0.64 | 14306 | 20749 |
| R $T_s^{TP}$ | 80.4 | 71.0 | 84.9 | 71.0 | 0.51 | 87.2 | 77.8 | 90.7 | 75.6 | 0.38 | 14306 | 6497 |
| R $T_s^{LRP}$ | 78.7 | 76.3 | 82.7 | 87.4 | 0.42 | 84.8 | 83.3 | 87.7 | 92.7 | 0.27 | 13873 | 1027 |
| R $T_s^{\star}$ | 79.5 | 76.5 | 83.8 | 85.6 | 0.43 | 85.9 | 83.7 | 89.1 | 91.1 | 0.27 | 14065 | 1345 |
| M | 80.0 | 74.7 | 85.0 | 80.7 | 0.45 | 87.4 | 82.5 | 90.5 | 86.1 | 0.31 | 14268 | 2609 |
| M $T_s^{TP}$ | 80.0 | 76.7 | 85.0 | 84.9 | 0.42 | 87.4 | 84.7 | 90.5 | 90.6 | 0.27 | 14268 | 1582 |
| M $T_s^{LRP}$ | 78.9 | 77.3 | 83.6 | 88.9 | 0.40 | 86.0 | 85.1 | 88.8 | 94.6 | 0.25 | 14036 | 713 |
| M $T_s^{\star}$ | 79.6 | 77.5 | 84.5 | 87.5 | 0.41 | 86.9 | 85.4 | 89.9 | 93.2 | 0.25 | 14192 | 990 |
| Test set (using $T_s(val)$) | | | | | | | | | | | | |
| R | 81.4 | 58.6 | 85.3 | 41.3 | 0.71 | 87.8 | 63.7 | 90.6 | 43.8 | 0.63 | 45040 | 74100 |
| R $T_s^{TP}$ | 80.9 | 71.2 | 84.6 | 71.1 | 0.51 | 87.0 | 77.4 | 89.7 | 75.1 | 0.39 | 44596 | 22586 |
| R $T_s^{LRP}$ | 78.2 | 75.4 | 81.4 | 87.2 | 0.43 | 83.8 | 81.7 | 85.9 | 91.9 | 0.29 | 42836 | 4217 |
| R $T_s^{\star}$ | 79.5 | 75.8 | 82.9 | 85.1 | 0.43 | 85.3 | 82.4 | 87.6 | 89.9 | 0.29 | 43659 | 5513 |
| M | 80.2 | 74.8 | 85.0 | 80.9 | 0.45 | 87.5 | 82.3 | 90.0 | 85.7 | 0.31 | 44721 | 9542 |
| M $T_s^{TP}$ | 79.4 | 75.8 | 84.1 | 84.9 | 0.43 | 86.5 | 83.3 | 88.9 | 89.8 | 0.28 | 44271 | 5775 |
| M $T_s^{LRP}$ | 77.4 | 75.6 | 81.7 | 89.0 | 0.42 | 84.2 | 82.9 | 86.3 | 94.0 | 0.27 | 42927 | 2805 |
| M $T_s^{\star}$ | 78.7 | 76.3 | 83.3 | 87.6 | 0.42 | 85.8 | 83.8 | 88.0 | 92.6 | 0.27 | 43754 | 3715 |

**Table 6.2: Instance segmentation results for different class-specific score thresholds.** The RetinaMask R101 M (R) and MRCNN R101 M (M) models behave very similar to the detection results of Table 6.1. Using $AP^{\star}$, the ranking of the models changes, even when most FPs of RetinaMask are filtered out.

### 6.5.4 Instance Segmentation

For completeness, we also show results with class-specific score thresholds and $AP^{\star}$ evaluation for RetinaMask R101 M and MRCNN R101 M in Table 6.2. Those have been the best instance segmentation baselines on *D2S* in Chapter 5.

Without using score thresholds, RetinaMask has a 1.2% higher $AP$ on the test set than FRCNN. However, again due to the high number of FPs, $AP^{\star}$ is 16.2% lower. Here, it is remarkable that even with the use of class-specific score thresholds, the $AP^{\star}$ values remain lower than those of FRCNN. For example, using $T_s^{\star}$ RetinaMask and MRCNN achieve an $AP^{\star}$ of 75.8% and 76.3%, respectively.

However, and maybe even more importantly, when $AP$ is replaced by $AP^{\star}$, the ranking of the two models changes. Thus, $AP^{\star}$ is able to show that FRCNN yields the better trade-off between precision and recall. The $AP^{\star}$ difference is large with a naive global score threshold $T_s^G = 0.05$ and can be reduced with the use of properly chosen global or class-specific score thresholds.

## 6.6 Conclusion

We revisited the mean average precision, the predominantly used evaluation measure for object detection. We have shown a number of issues that occur when using $AP$ as objective for model optimization and propose corresponding changes to the evaluation protocol. In particular, we introduced an adapted version of average precision, $AP^\star$, that leads to a better balance between precision and recall. Moreover, $AP^\star$ can be used to compute optimal class-specific score thresholds $T_s^\star$. In our experiments, we have seen that they reduce the number of false positives remarkably, e.g., from 104 633 to 5567 for RetinaNet on *D2S*, while preserving a high average recall. $AP^\star$ fixes $AP$'s bias towards recall and will encourage the design of models with properly adjusted score thresholds and meaningful uncertainty estimation. Our experiments for instance segmentation show that $AP^\star$ leads to a different ranking of models that better incorporate their precision.

# 7 Data Generation for Few-Shot Detection

Despite the great success that deep-learning-based object detection has brought to many challenging application areas, one substantial hurdle still remains: the tedious annotation process. In comparison to image classification, where a single label per image is sufficient, for object detection each object instance within the image has to be annotated with a class label and with a bounding box. For some applications, e.g., in a medical or biological context, or for specific types of defects on an industrial product, the labeling requires expert knowledge. This results in significant costs that occur directly at the beginning of the method's implementation. For the user, this means that the annotation investment has to be made before it is clear how well or if at all the method will be successful for the task at hand.

But even if everything works out and the user can achieve his goal, what if the task changes? For example, the number of categories to be detected changes from 13 to 15 because two new products are being manufactured, or the biologist is interested in new cell types. Then, the process starts all over again and images have to be relabeled or they have to be acquired and annotated from scratch, again without any guarantee of success.

Hence, we are interested in methods that work without much annotation effort. On the one hand, this means that we want to train our algorithm with only a small number of images, but on the other hand, and more importantly, with only a few labeled object instances per class. Methods that work under these circumstances are termed few-shot detection methods. They certainly improve the ease-of-use aspect of DL-based algorithms.

In this chapter, we neither change the algorithm, nor do we restrict the total number of training images. However, we aim to reduce the number of manually annotated instances and, instead, we artificially *augment* our data to increase the number of labeled training images. In particular, we try to use only a few instances or a single instance per category as a basis. Using these samples, we generate artificial images that are close to the images we expect in the test set and use them to train and validate our model. Because only a fraction of the training data has been annotated manually and the annotations might be incomplete, these approaches are referred to as *weakly supervised* methods.

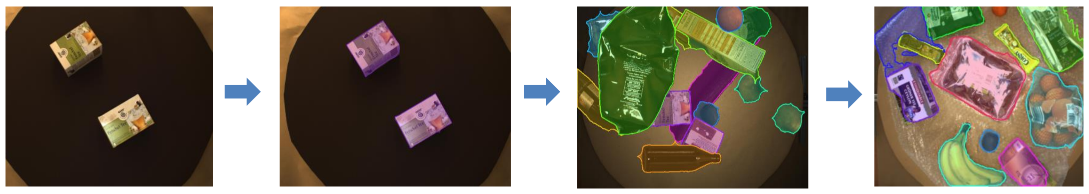


**Figure 7.1: Few-shot data generation pipeline.** (*From left to right*) Acquisition of template images for automatic labeling, generation of artificial training images, and application on real test images. The training step is not visualized.

Of course, there is a domain gap between the real test images and the artificially generated training images. Hence, the goal is to reach the performance of the results that we achieve when using real training data.

The proposed pipeline consists of the following steps as depicted in Fig. 7.1:

- Acquisition of template images for each object category.
- Generation of artificial training images.
- Training of the model using the generated data.
- Application of the model on real images.

We will only describe the first two steps in more detail because the third and fourth step are the same as when training on real images except that the training set is changed.

In this chapter, we evaluate the weakly generated annotations against the baseline trained with fully-supervised training data on *D2S* (cf. Chapter 5). Due to partially different objects, a slightly different acquisition setup, and lighting changes, there is a domain shift to the validation images. Nevertheless, we find that the proposed method achieves 68.9% *AP*, compared to 80.1% *AP* of a fully-supervised baseline without domain-shift. Hence, we show that it is possible to produce competitive segmentation results with a very simple acquisition setup, virtually no label effort, and suitable data augmentation.

The content of this chapter has been partly published in [46], [47], and [194]. For the comparison of the results to other chapters within this work, we have added Section 7.5.

## 7.1 Related Work

**Weakly supervised instance segmentation.** Solving computer vision tasks with weakly annotated data has become a major topic in recent years. It can significantly reduce the amount of manual labeling effort and thus make solving new tasks feasible. For example, Deselaers et al. [29] learn *generic knowledge* about object bounding boxes from a subset of object classes. This is used to support learning new classes without any location annotation. This allows them to train a functional object detector from weakly supervised images only. Similarly, Vezhnevets et al. [195] attempt to learn a supervised semantic segmentation method from weak labels that can compete with a fully supervised one. Recently, there has been work that attempts to train instance segmentation models from bounding box labels or by assuming only parts of the data is labeled with pixel-precise annotations. For example, the work by Hu et al. [77] attempts to train instance segmentation models over a large set of categories, where most instances only have box annotations. They merely assume a small fraction of the instances have mask annotations that have been manually acquired. Khoreva et al. [91] train an instance segmentation model by using GrabCut [163] foreground segmentation on bounding box ground truth labels.

In contrast to the above works, our weak supervision only assumes the object class of each training image to be known and does not require bounding boxes of the single objects or their pixel-precise annotations. We use basic image processing techniques and a simple acquisition setup to learn competitive instance segmentation models from weak image annotations.

**Data augmentation.** Since we restrict the training images to objects of a single class on a homogeneous background, it is essential to augment the training data with more complex, artificial images. Otherwise, state-of-the-art instance segmentation methods fail to generalize to more complex scenes, different backgrounds, or varying lighting conditions. This is often the case for industrial applications, where a huge effort is necessary to obtain a large amount of annotated data. Hence, extending the training dataset by data augmentation is common practice [176]. Augmentation is often restricted to global image transformations such as random crops, translations, rotations, horizontal reflections, or color augmentations [95]. However, for instance-level segmentations, it is possible to extend these techniques to generate completely new images. For example, in [48, 114, 215], new artificial training data is generated by randomly sampling objects from the training split of COCO [118] and pasting them into new training images. However, since the COCO segmentations are coarse and the intra-class variation is extremely high, the augmentation brings limited gain. On the other hand, in the *D2S* dataset [46] (Chapter 5), it is difficult to obtain reasonable segmentation results without any data augmentation. The training set is designed to mimic the restrictions of industrial applications, which include little training data and potentially much more complex test images than training images.

Analogously to the above works, we perform various types of data augmentation to increase complexity and the amount of the training data. However, we go a step further and address specific weaknesses of the state of the art on *D2S* by explicitly generating artificial scenes with neighboring and touching objects. To further gain robustness to changing illumination, we also render the artificial scenes under different lighting conditions by exploiting depth information.

## 7.2 Acquisition of Template Images

Independent of the algorithm, some input to the model is necessary. In our setting, we are dealing with end-to-end trainable models and thus the input to the model are images and the corresponding labels, consisting of bounding boxes.

When training on real images, we require that

- all different combinations of neighboring objects from different categories should be captured such that the distribution of the test set is captured as well as possible;
- the background and environment is as close as possible to the expected setting in the test images. If different backgrounds are present in the test setting, also the training images should capture these differences; and
- objects should be captured in various different orientations and scales as present in the test set.

A valid and frequently used strategy to fulfill these requirements is to use a running system and acquire the images from exactly the setup for which the algorithms should be applied later.

For our approach, the first step is to acquire a set of template images for each of the object classes that we want to detect. In comparison, to the use of real images, we capture the templates with following rules:

- Only objects of a single category should be present within an image.
- The background should be of homogeneous color, texture-less, and such that the objects can be clearly distinguished from it.
- Objects should be captured without touching or overlapping each other.
- Objects should be captured with all different poses that are present in the test set. Different rotations or scales are only required if they add information about the object that is not present in other views. For example, a box with two different prints on each side should be captured from both sides, while a banana that looks the same from both directions only needs to be captured once.

The first rule is important for the annotation process. If users only capture template images containing objects of a single category, they just need to label the whole image with the category. Each instance within the image can than be assigned to this class. Because users usually know which category of objects they are currently capturing, this labeling only adds a negligible cost. For example, in our acquisition script the user just has to press one button if he changes to the next class of objects that are listed according to the order the objects are captured.

In this case, if the test environment is used to acquire the templates, one needs to make sure that a homogeneous background is present according to the second rule. However, in this work, we use a specific acquisition setup similar to the one shown in Fig. 5.1. To be able to reduce the label and acquisition effort to a bare minimum, the image acquisition setup is constructed in a very basic manner. A high-resolution industrial RGB camera with $1920 \times 1440$ pixels is mounted above a turntable. The turntable allows to acquire multiple views of each scene without any manual interaction. To increase the perspective variation, the camera is mounted off-center with respect to the rotation center of the turntable. Additionally, a stereo camera that works with projected texture is fixed centrally above the turntable. In Section 7.3, we show how the depth images may be used to extend the capabilities of data augmentation.

To make the automatic label generation as simple and robust as possible, the background of the turntable is set to a plain colored brown surface. In an initial step, we keep the background color fixed for every training image. In a next step, we further use a lighter brown background to improve the automatic segmentation of dark or transparent objects such as avocados or bottles. The corresponding generated datasets are denoted with the prefix *weakly* and *weakly cleaned*, respectively.

Note that, although the acquisition setup is very similar to that of the original *D2S* setup, there is a domain shift between the new training images and original *D2S* images. In particular, the camera pose relative to the turntable is not the same and the background and lighting differ slightly. Maybe the most significant difference is, however, that some of the captured objects are different from those in original *D2S*. For example the vegetables and fruit categories have a slightly different appearance and some packaging, e.g., for the classes *clementine* or *apple_braeburn_bundle* are not the same as in *D2S*. The reason is that we had to buy these fruits again for the experiments in this chapter and their packaging had changed. The two classes *oranges* and *pasta_reggia_fusilli* were not available for purchase with a similar packaging anymore and therefore, the respective *D2S* training scenes (without labels) are used.

In the following, we will see that the second and third rule allow to retrieve pixel-precise segmentations of each object. One option is to use a very basic image processing pipeline that mainly consists of background subtraction from the captured template images. To evaluate this simple algorithm, we also compare it to a more sophisticated saliency detection method.

### 7.2.1 Background Subtraction

We utilize the simple setting of acquired images and automatically generate pixel-precise segmentations by background subtraction. To account for changing illumination of the surrounding environment, an individual background image is acquired for each scene. By subtracting the background image from each image, a foreground region can be generated automatically with an adaptive binary threshold [149]. Depending on the object and its attributes, we either use the V channel from the HSV color space, or the summed absolute difference of each of the RGB channels. The results for both color spaces can be computed with negligible cost. Therefore, they can already be shown during the acquisition process and the user can choose the better region online. To ensure the object is not split into multiple small parts, we perform morphological closing with a circular structuring element on the foreground region. The instance segmentations can then be computed as the connected components of the foreground. The automatic segmentation method assumes that the objects are not touching or occluding each other, and generally works for images with an arbitrary number of objects. A schematic overview of the region generation is shown in Fig. 7.2.

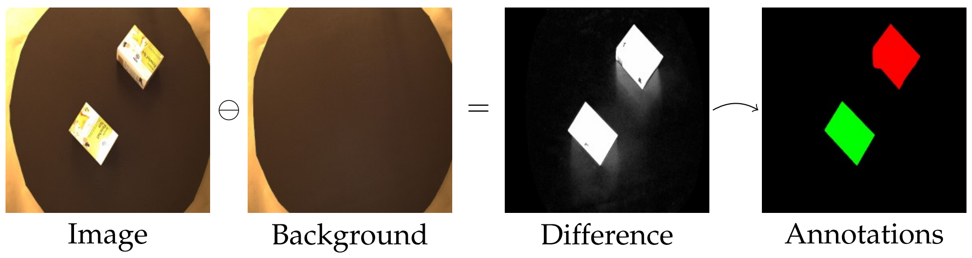


**Figure 7.2: Background subtraction.** Schematic overview of the generation of pixel-precise segmentations for objects in the template images.

### 7.2.2 Saliency Detection

As an alternative to the algorithmically simple background subtraction, the characteristics of the template images also invite to use saliency detection methods to identify the instances. Currently, the best methods are based on deep learning and require fine-tuning to the target domain [10, 112]. Hence, they require manually labeling at least for a subset of the data. A more generic approach is that of the *Saliency Tree* [125]. It is constructed in a multi-step process. In a first step, the image is simplified into a set of primitive regions. The partitions are merged into a saliency tree based on their saliency measure. The tree is then traversed and the salient regions are identified. The salient region generation requires no fine-tuning to the target domain and achieves top ranks in recent benchmarks [10, 112].

We use the Saliency Tree to generate saliency images for each of the template images. The foreground region is then generated from the saliency image by a simple thresholding and an intersection with the region of the turntable in the acquisition setup. Also here, morphological closing and opening with a small circular structuring element is used to close small holes and smooth the boundary. Analogously to the background subtraction, the single instances are computed as the connected components of the foreground region. Qualitatively, we found that a threshold of 40 was a good compromise between too large and too small generated regions. For some rather small objects, using this threshold results in regions that almost fill the whole turntable. To prevent these artifacts, we iteratively increase the threshold by ten until the obtained total area of the regions is smaller than 30% of the turntable area. However, even with this precaution, in some cases the obtained instances may be degenerated. A few examples and failure cases of both the background subtraction and the saliency detection are displayed in Fig. 7.3. We denote the annotations obtained with the saliency detection method by *saliency cleaned*.

A few example classes where the lighter background of the *cleaned* setup significantly improved the automatic labels are displayed in Fig. 7.4.

## 7.3 Generation of Training Images

One of the challenges of applying deep-learning-based CNNs is the large amount of training data that is required to obtain competitive results. In the real-world applications discussed in this work, the acquisition and labeling of training data can be expensive

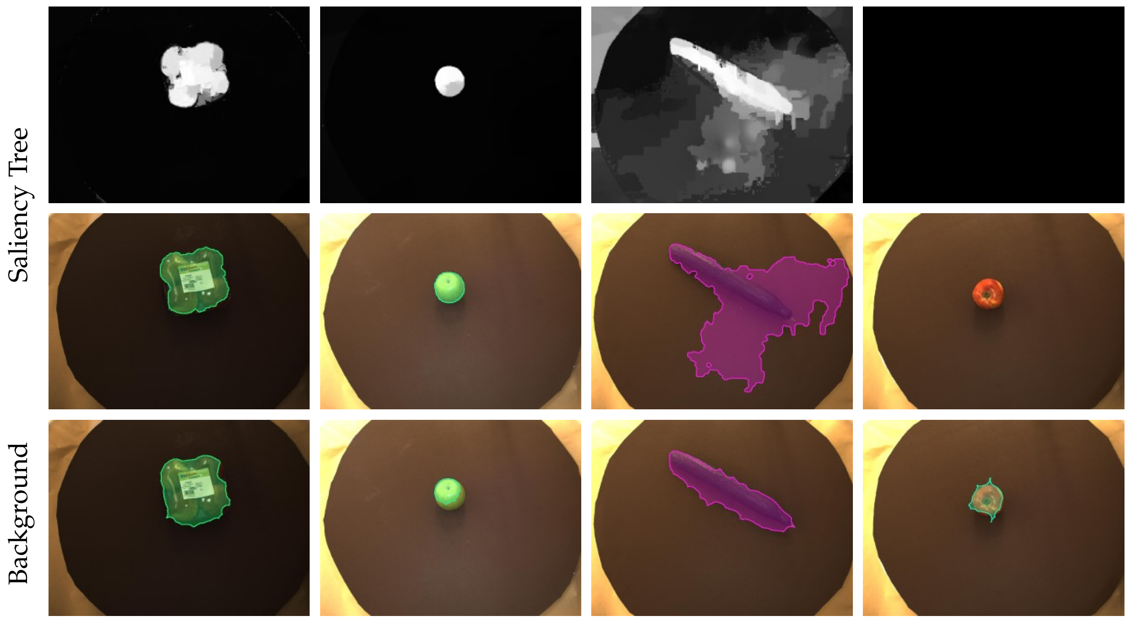


**Figure 7.3: Saliency Tree vs. background subtraction.** The primitive regions of the Saliency Tree method not always correspond to semantically different regions but can also occur due to shadows or reflections (*1st row*). The automatic segmentations for saliency detection (*2nd row*) and background subtraction (*3rd row*) are displayed. In general, the saliency detection and the background subtraction return similar results (*1st column*). In rare cases, the saliency detection outperforms the background subtraction and returns more complete regions. In the third and fourth column some typical failure cases of the saliency detection scheme are displayed. Either it fails completely (*4th column*) or it is hard to find a reasonable threshold (*3rd column*).

because it requires many manual tasks. To mitigate this issue, we use data augmentation, where additional training data is created automatically based on a few manually acquired simple images. The generated images simulate the variations and complexity that commonly occur when applying the trained network.

Mostly, we are using data generation techniques here because we synthesize new images. In comparison to the term *data generation*, in the literature, *augmentation* mostly denotes applying transformations to an image without changing the scene. Because we are not synthesizing new images from scratch, but use realistic backgrounds and reuse parts of existing images, we are somewhere in between data generation and augmentation. Hence, we use the terms generation and augmentation as if they were synonyms throughout this chapter.

To augment the training data, we randomly select between 3 and 15 objects from the training set, crop them out of the training image utilizing the generated annotation, and paste them onto a background image similar to the one from the *D2S* training images. The objects are placed at a random location with a random rotation. This generates complex scenes, where multiple objects of different instances may be overlapping each other. Exemplary images using the manual annotations for cropping are shown in Fig. 5.10. However, since certain failure cases in the *D2S* validation and test set remain (cf. Section 5.7), we also introduce two new augmentation techniques to specifically address these challenges, namely *touching objects* and *reflections*.

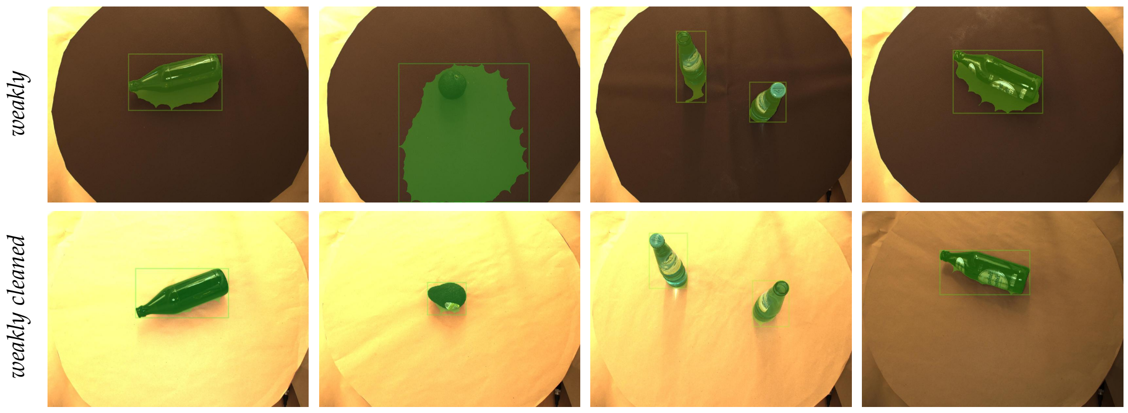


**Figure 7.4: Weakly vs. weakly cleaned.** Examples where the light background used for *weakly cleaned* significantly improves the automatic segmentations over those from the dark background used for *weakly*. In these examples the background subtraction method is used to derive the segmentations.

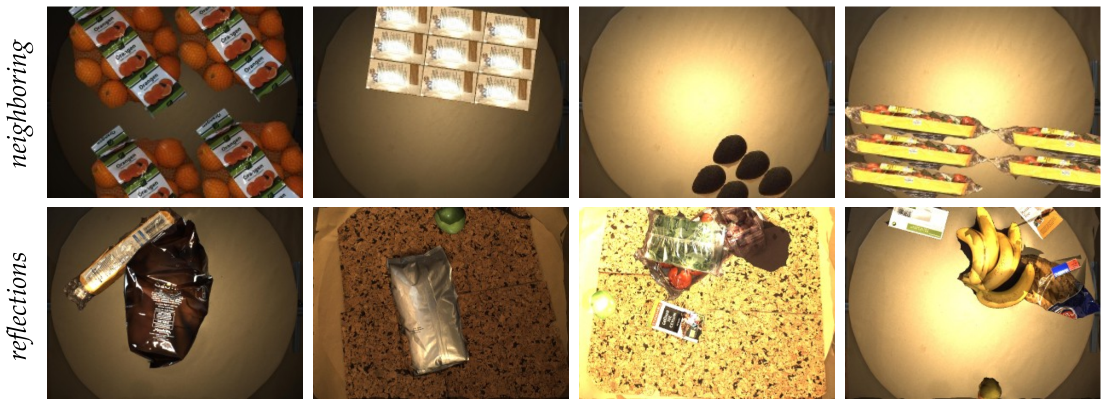


**Figure 7.5: Touching objects and reflections.** Examples for generated images with touching objects (*top*) and reflections using a simulation of a spotlight (*bottom*).

### 7.3.1 Touching Objects

In the validation and test set of *D2S*, there are many instances of the same class that touch each other. The existing instance segmentation schemes have difficulties to find the instance boundaries and often return unsatisfactory results. Frequently, if objects are close, the methods only predict a single instance or the instances extend into the neighboring object. Hence, we specifically augment the training set by generating new images where instances of the same class are very close to or even touching each other. We denote the respective dataset with the suffix *neighboring*. Fig. 7.5 shows some examples of augmented touching objects.

### 7.3.2 Reflections

To create even more training data, we augment the original data by rendering artificial scenes under different lighting conditions. For this, we use the registered 3D sensor and RGB camera to build textured 3D models of the different object instances. Random subsets of these instances are then placed at random locations to create new, artificial scenes. Since we do not know the surface characteristics of the individual objects, we use a generic Phong shader [150] with varying spotlight location and spotlight and ambient light intensity to simulate real-world lighting. We use this approach also because in real-world scenarios, lighting can vary drastically compared to the training setup. For example, different checkout counters can have different light placements, while others might be close to a window such that the time of day and the weather influence the local lighting conditions. We denote the respective dataset with the suffix *reflections*. Example images of the *reflections* set are shown in Fig. 7.5.

## 7.4 Experiments

All of the experiments are carried out on *D2S* [46], as the datasets' splits are explicitly designed for the use of data generation and augmentation. In comparison to the validation and test sets, the complexity of the scenes in the training set are a lot lower in terms of object count, occlusions, and background variations. Moreover, the techniques introduced in Section 7.3 are well-suited for an industrial setting, where the intra-class variations are mainly restricted to rigid and deformable transformations of the objects and background or lighting changes.

Our focus is the analysis of the weakly supervised setting and the different augmentation techniques. Therefore, we use the competitive instance segmentation method Mask R-CNN [70] (cf. Chapter 4) for our experiments. We choose the original Detectron [56] implementation in order to make the results easy to reproduce. For the comparison with results from the previous and the following chapters, we additionally show HALCON results in Section 7.5.

**Setup.** To speed up the training and evaluation process, we downsized *D2S* by a factor of four, which results in an image resolution of $480 \times 360$ px. Note that we also adapted the image size parameters of Mask R-CNN accordingly, such that no scaling has to be carried out during training or evaluation.

All models were trained using two NVIDIA GTX1080Ti GPUs. We used the original hyperparameters from Detectron except for the following: a batch size of five images per GPU (resulting in ten images per iteration) and a base learning rate of 0.02 (reduced to 0.002 after 12k iterations). We trained for 15 000 iterations and initialized with weights pretrained on *COCO*.

For the evaluation of our results, we used the *D2S* validation set. The *COCO AP* is used as the performance measure (cf. Section 4.2) and we also show some comparisons using $AP^\star$ (cf. Chapter 6).

**Baseline.** As a baseline for the weakly supervised setting, we use the high quality, manually generated annotations from the *D2S* training set using the original split as provided in [46] (Chapter 5). To better compare and separate the effect of the data

| Training Set | baseline | weakly | weakly cleaned | saliency cleaned |
|---|---|---|---|---|
| *train* | 48.3 | 8.5 | 15.9 | 16.5 |
| *train + augm 2500* | 77.0 | 62.8 | 64.8 | 55.7 |
| *train + augm 5000** | 77.8 | 62.2 | 61.9 | 55.4 |
| *train + augm 10000* | 78.4 | 65.0 | 63.0 | 55.8 |
| *train + augm 20000* | 78.4 | 65.0 | 62.7 | 59.7 |
| * + NB | 78.5 | 63.7 | 62.5 | 59.2 |
| * + RB | 78.5 | 64.9 | 68.0 | 58.7 |
| * + RE | - | - | 66.9 | 59.7 |
| * + NB + RB | **80.1** | **66.8** | **68.9** | 60.2 |
| * + NB + RB + RE | - | - | 68.5 | **61.9** |

**Table 7.1: Results.** Ablation study for different ways of generating the annotations and different augmentations. The performance is given in terms of *AP* percentage values. * indicates the set that gave the best results in combination with specific augmentations. Abbreviations for augmentation types are as follows: *neighboring* (NB), *random background* (RB), *reflections* (RE).

generation from the effect of having better training data, we also augment this high-quality training set. Except for reflections (which requires depth information), the generation can be done analogously to the weakly supervised setting. Because the annotations fit almost perfectly, the object crops contain only a very small amount of background surroundings compared to the crops from the weak annotations. Therefore, one can expect the best results for the baseline.

**Results.** For all types of underlying annotations, *baseline*, *weakly*, *weakly cleaned*, and *saliency cleaned*, we made similar experiments. First, we trained the model only on the training images, denoted as *train*. Second, we augmented 2500, 5000, 10 000, or 20 000 images as described at the beginning of Section 7.3 and added them to *train*, respectively (*augm*). Third, we generated 2000 images both with touching objects (*neighboring*) or on a random background (*random background*). Additionally, for *weakly cleaned* and *saliency cleaned*, we generated 2000 images and augmented them with *reflections* (on random background). The corresponding *AP* values on the *D2S* validation set (in quartersize) are shown in Table 7.1.

The images obtained for our weakly supervised training are significantly less complex than the *D2S* validation images; there are no touching or occluding objects and always only one category per image. This large domain shift results in a very poor performance of the models trained only on *train* compared to the *baseline* (cf. row 1 of Table 7.1). Normal augmentation by cropping and randomly pasting the objects strongly improves the results, e.g., from 8.5% to 65.0% for *weakly*. Note that for the normal augmentation, the annotation quality seems to be less important, as *weakly cleaned* is on the same *AP* level as *weakly*. Only *saliency cleaned* performs significantly worse, probably due to some corrupt automatically generated annotations. The specific augmentation types *neighboring* (NB), *random background* (RB), and *reflections* (RE) further help to improve the result to

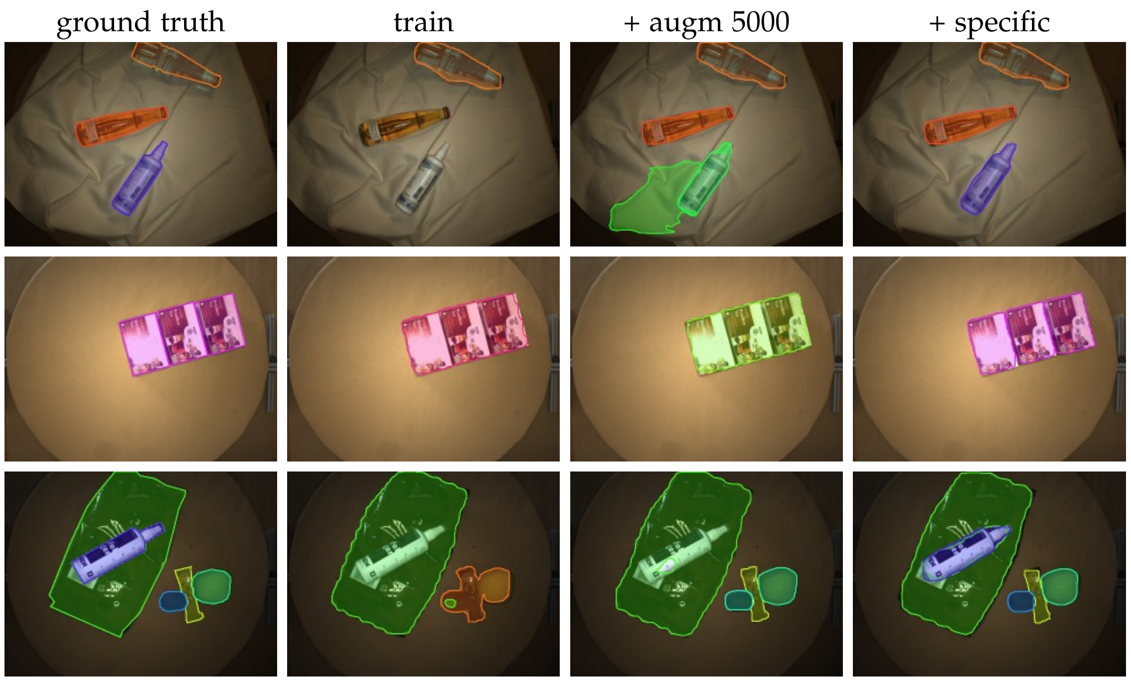


**Figure 7.6: Qualitative Results.** Improvements for *weakly cleaned* using specific augmentations: (*top*) *random backgrounds*, (*middle*) *neighboring* objects, (*bottom*) *reflections*.

68.9% for *weakly cleaned*, which is more than four times better than the *train* result. NB, RB, and RE are indeed complementary augmentation types as each of them consistently helps to improve upon *train + augm 5000*. In Fig. 7.6, some qualitative results are displayed. They show that using the specific augmentations indeed helps to improve on the typical failure cases that they address. Also note that the relative improvement using specific augmentations is higher in the weakly supervised setting than for the *baseline* (e.g., 7% for *weakly cleaned* vs. 2.3% for *baseline*).

Usually with a higher number of training data, the results of models with a high number of parameters are improved. However, we found that the best results are obtained if the specific augmentation sets of step three and four are added to *train + augm 5000*. A possible reason is the domain shift between *D2S* validation and the augmented images. For completeness, we show results for *augm 2500* and *augm 10000* in Appendix C.

## 7.5 Improving the D2S Baselines

In Section 5.7, we saw that a high number of false positives (FPs) arise from the domain gap between the training set and the validation and test sets of *D2S*. In this section, we want to use the augmentation techniques for touching objects and with different backgrounds to improve our baselines on *D2S*. Therefore, we add the *baseline* NB and BG sets to the training images. For a fair comparison, we reduce the number of epochs, such that the baseline models from Chapter 5 and the new models are trained with approximately the same number of iterations.

In Table 7.2, we compare the results for the two box detection methods RetinaNet and

| Model | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $AP$ | $AR$ | $AP$ | $AR$ | $TP$ | $FP$ | $AP$ | $AR$ | $TP$ | $FP$ |
| validation set | | | | | | | | | | |
| RetinaNet | 77.7 | 82.9 | 86.5 | 90.3 | 14256 | 28563 | 81.3 | 85.8 | 13547 | 29272 |
| RetinaNet + RB + NB | 80.1 | 85.2 | 88.2 | 91.8 | 14476 | 18602 | 84.0 | 88.4 | 13943 | 19135 |
| FRCNN | 79.1 | 84.4 | 86.7 | 89.8 | 14179 | 3142 | 81.7 | 86.3 | 13643 | 3678 |
| FRCNN + RB + NB | 81.5 | 86.6 | 88.4 | 91.4 | 14427 | 2038 | 84.6 | 88.7 | 14018 | 2447 |
| RetinaMask | 79.5 | 84.1 | 87.3 | 90.8 | 14300 | 22542 | 83.5 | 87.5 | 13773 | 23069 |
| RetinaMask + RB + NB | 81.0 | 85.5 | 88.3 | 91.7 | 14454 | 15738 | 85.1 | 89.0 | 14032 | 16160 |
| MRCNN | 77.7 | 83.2 | 86.5 | 89.7 | 14169 | 3124 | 82.0 | 86.5 | 13674 | 3619 |
| MRCNN + RB + NB | 80.6 | 85.6 | 88.8 | 91.7 | 14459 | 1938 | 85.1 | 88.9 | 14039 | 2358 |
| test set | | | | | | | | | | |
| RetinaNet | 78.3 | 82.8 | 87.2 | 90.5 | 45107 | 104633 | 81.6 | 85.3 | 42317 | 107423 |
| RetinaNet + RB + NB | 80.5 | 84.8 | 88.6 | 91.2 | 45461 | 66203 | 84.1 | 87.5 | 43470 | 68194 |
| FRCNN | 79.0 | 83.8 | 86.8 | 89.4 | 44426 | 12134 | 81.9 | 85.7 | 42544 | 14016 |
| FRCNN + RB + NB | 81.4 | 86.0 | 88.5 | 90.8 | 45171 | 7642 | 84.4 | 88.1 | 43759 | 9054 |
| RetinaMask | 79.3 | 83.5 | 86.9 | 90.0 | 44710 | 83192 | 83.3 | 86.7 | 43010 | 84892 |
| RetinaMask + RB + NB | 81.3 | 85.1 | 88.2 | 90.8 | 45178 | 55403 | 85.3 | 88.4 | 43867 | 56714 |
| MRCNN | 77.7 | 83.5 | 86.5 | 89.7 | 44577 | 11289 | 81.7 | 86.5 | 42951 | 12915 |
| MRCNN + RB + NB | 80.1 | 85.0 | 88.2 | 90.6 | 44998 | 7510 | 84.6 | 88.3 | 43791 | 8717 |

**Table 7.2: Baseline improvements with data augmentation.** All measures are calculated with respect to box IoU for detection models RetinaNet and FRCNN and mask IoU for instance segmentation models RetinaMask and MRCNN, respectively. Using the introduced augmentation techniques for touching objects or random backgrounds we can increase the $AP$ and significantly reduce the number of FPs.

| Model | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $AP$ | $AP^\star$ | $AP$ | $AP^\star$ | $TP$ | $FP$ | $AP$ | $AP^\star$ | $TP$ | $FP$ |
| RetinaNet | 76.0 | 72.1 | 84.1 | 81.0 | 43085 | 5567 | 79.5 | 75.4 | 41065 | 7587 |
| RetinaNet + RB + NB | 78.6 | 75.4 | 86.0 | 83.6 | 44019 | 4778 | 82.2 | 79.1 | 42455 | 6342 |
| FRCNN | 76.1 | 73.4 | 83.3 | 81.2 | 42556 | 3967 | 78.9 | 76.1 | 40965 | 5558 |
| FRCNN + RB + NB | 80.3 | 77.7 | 87.2 | 85.0 | 44562 | 4220 | 83.3 | 80.7 | 43210 | 5572 |
| RetinaMask | 77.4 | 73.5 | 84.4 | 81.2 | 43265 | 5807 | 81.5 | 77.5 | 41906 | 7166 |
| RetinaMask + RB + NB | 79.8 | 76.6 | 86.3 | 83.7 | 44123 | 5047 | 83.8 | 80.7 | 43075 | 6095 |
| MRCNN | 75.4 | 73.0 | 83.8 | 81.9 | 43119 | 4158 | 79.3 | 76.8 | 41684 | 5593 |
| MRCNN + RB + NB | 78.6 | 76.4 | 86.5 | 84.6 | 44101 | 3824 | 83.1 | 80.8 | 42980 | 4945 |

**Table 7.3: $AP^\star$ results on *D2S* test using $T_s^\star(val)$.** Gains from the specific *neighboring* and *random backgrounds* augmentations are remarkable and consistent across IoU thresholds and different model types.

FRCNN as well as for the two instance segmentation models RetinaMask and MRCNN. Here, the baseline results without specific augmentations are from the same models as in Table 5.5 for detection and Table 5.6 for instance segmentation. All models use a ResNet 101 backbone and are trained and evaluated on images of size $512 \times 384$ px.

For all models, the $AP$ can be increased by 1.5% (RetinaMask) to 2.9% (MRCNN). At the same time, the number of FPs is reduced by around 30% for all models without the use of score thresholds. The gains are even higher at IoU threshold 0.8 than at IoU threshold 0.5. This indicates that the localization accuracy of results is improved.

To allow a comparison with the results of Chapter 6, we show $AP^\star$ results in Table 7.3. Note that for all models, we used class-specific score thresholds $T_s^\star$ computed to optimize

$AP^{\star}_{50}$. Here, the baseline results without specific augmentations are from the same models as shown in Table 6.1 for detection and Table 6.2 for instance segmentation. Since $AP^{\star}$ includes the actual model precision directly, the FP reduction and higher recall leads to a significant increase in $AP^{\star}$ between 3.1% (RetinaMask) and 4.3% (FRCNN).

## 7.6 Conclusion

We have presented a system that allows to train competitive instance segmentation methods with virtually no label effort. By acquiring very simple training images, we were able to automatically generate reasonable object annotations for the *D2S* dataset. To tackle the complex validation and test scenes, we propose to use different types of data augmentation to generate artificial scenes that mimic the expected validation and test sequences. We present new augmentation ideas to help improve scenes where touching objects and changing illumination is a problem. The results indicate that weakly supervised models yield a very good trade-off between annotation effort and performance. This paves the way for cost-effective implementations of instance segmentation approaches by lifting the requirement of acquiring large amounts of training images.

Using imperfect annotations, we also found that increasing the number of augmented images does not always improve the result. We believe that reducing the domain shift to the test set by generating more realistic augmentations is an open topic that could resolve this problem. Additionally, we found that data augmentation can be beneficial even if the number of labeled training images is already large.

# 8 Oriented Box Detection

While so far we have built models that are invariant to the objects' orientation, now we explicitly want to predict the orientation of each object within an image. In this chapter, we show how current object detection methods, such as RetinaNet or FRCNN, can be adapted to predict oriented boxes instead of axis-aligned ones. In many applications, such as bin-picking or where the found objects need to be aligned for further processing, this can be a requirement for the detection method. Moreover, we will see that the prediction with oriented boxes contains some more benefits that make it more usable than axis-aligned box detection for some tasks.

Oriented box detection as described within this chapter has been implemented and released as a major new feature of HALCON 19.05 [142].

## 8.1 Introduction

Generally, an axis-aligned bounding box (AABB) is a very coarse 2D approximation of an object within an image. For example, think of an articulated person or animal the body might be in the center of the AABB and only the legs and arms are stretched out and touching the boundary of the box. But also for box-shaped objects that are not aligned with the image axes, AABBs do not fit well due to their orientation. While for the first case of articulated objects, an oriented bounding box (OBB) might not fit much better than an AABB, for box-shaped objects or thin and elongated parts like a screw or pencil this is the case. Moreover, predicting the orientation can be of interest, e.g., when the object should be grasped by a robot. This can also be the case for objects with a circular shape — think of a circle that has a small printing on one side that defines the orientation.

Today, in most detection datasets, such as *VOC* [41], *COCO* [118], *OpenImages* [98], or *D2S* (Chapter 5), the ground truth annotations are either given as instance masks or as AABBs. Compared to an OBB, the instance mask captures the object more precisely, but is much more expensive to annotate. AABBs can be labeled faster than OBBs and they can be uniquely derived from an instance mask. This is not always the case for OBBs, as shown by Böttger et al. [12]. The increased annotation effort and the sometimes ambiguous orientation of OBBs might be the main reasons why detection with OBBs has not gained much popularity so far.

However, oriented box detection (*OD*) certainly has some advantages over axis-aligned box detection (*AAD*): The best possible intersection over union (IoU) an oriented bounding box can achieve for an arbitrary mask is typically much higher than the IoU of the axis-aligned bounding box [12]. This means that, independent of an object's orientation, most of the bounding box area overlaps with the instance of interest. This avoids large overlaps with neighboring objects. In theory, for two-stage detectors like Faster R-CNN [159], features that are pooled for the second stage should contain less background influences. Since the box is tighter around the object, the given resolution of the pooling grid is used more effectively. In this work, we analyze if this helps to improve the classification results. We have seen in Chapter 5 that a wrong classification is one of the main reasons of detection failure cases.

Bounding box aspect ratios become invariant to an object's rotation, which makes it easier to set the parameters of anchor boxes. For example, for a dataset with few object categories and where perspective distortions are negligible, the anchor aspect ratios can be set according to the object's dimensions. For non-overlapping objects, OBBs overlap significantly less than AABBs. This avoids erroneously filtering out candidate boxes by NMS and makes it more intuitive to tune NMS hyperparameters. Moreover, the detection results of OBBs are visually more appealing as it is easier to assign the box to the underlying object.

Therefore, in this chapter we propose using oriented as opposed to axis-aligned bounding boxes for object detection. The main contributions of this chapter are: We show how current *AAD* methods need to be adapted for *OD*. Our approach can easily be applied to existing models based on AABBs. We describe how to adapt architectures for *OD* and explain necessary changes to different parts of the model compared to baseline *AAD* models. Further, we introduce $\mathrm{rIoU}_{\mathrm{BB}}$, an adapted version of the well-known IoU that allows to compare *AAD* and *OD* models on a fair basis. In our extensive experiments section we evaluate *OD* models on *Screws*, *Pill Bags*, and *D2S*. We provide ablation studies for our design choices and show that *OD* predictions are in many cases a much better abstraction of the underlying objects than *AAD* results. Finally, with our experiments on *D2S*, we show that also for datasets where many categories do not have a clear orientation, *OD* still leads to comparable results as the *AAD* baseline. Moreover, for some categories, such as bottles, zucchinis, bananas, or oranges, the results can be significantly improved.

## 8.2 Related Work

The idea of oriented box detection has been introduced in the context of scene text detection. Existing methods are all based on Faster RCNN [159]. Jiang et al. [86] still use an axis-aligned box RPN but infer an oriented final box output by regressing the orientation. In [129], Ma et al. extend the RPN to use oriented anchors and the RoI pooling to pool features with respect to oriented boxes. Oriented box detection has been demonstrated to perform well on other domains than in OCR. For example, Ding et al. [31] have set a new benchmark on the oriented object detection dataset DOTA [200]. In

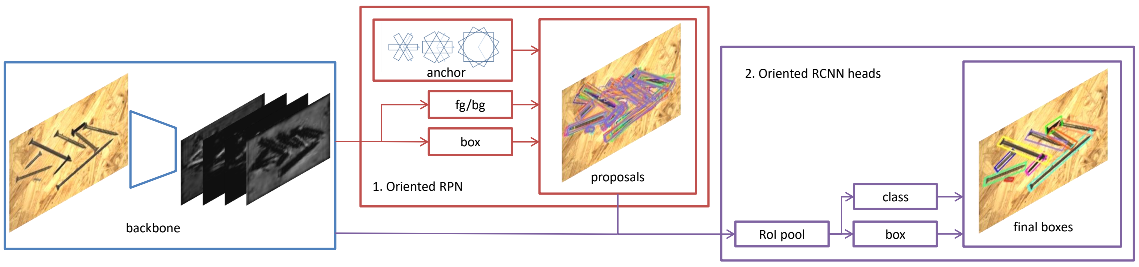


**Figure 8.1: Exemplary *OD* architecture.** (*Blue, left*) Input image and backbone for feature extraction. (*Red, middle*) Oriented RPN. (*Violet, right*) Features are pooled with respect to oriented proposals to feed RCNN heads for final oriented box output. While the overall architecture is very similar to the *AAD* counterpart, it is important to change several details such that the method works as expected.

contrast to our work and [129], they do not use oriented anchors but propose a RoI Transformer module before pooling with respect to the oriented box. Another application that benefits from the use of oriented boxes is ship detection in satellite images [202], where the authors propose a dense FPN with oriented anchors.

The most similar approach to ours is that of [129], but we use the angle-related IoU [121] for the anchor target generation and certain other small but logic changes to their approach. We incorporate the idea of [31] to pool features with respect to the oriented box proposals. Moreover, we explain our design both for a two-stage detector, such as FRCNN [159], and a one-stage detector, such as RetinaNet [120].

In comparison to previous oriented object detection methods, we explicitly allow to set *classes without orientation*: Since not all classes might have a well-defined orientation, we include the option that the model falls back to predict an AABB for some user-defined classes, which stabilizes the training.

## 8.3 Oriented Box Detection

In the following, we present the key components to make the idea of *OD* work efficiently. Our approach is applicable to all *AAD* methods. Exemplary *OD* architectures based on FRCNN and RetinaNet are depicted in Fig. 8.1: In a first step, the backbone is applied to the input image to extract features that are used for the one or two following stages: For FRCNN, the first stage is the RPN, which predicts for each of a number of template *anchor boxes* whether it is likely that an object with similar bounding box is present or not (fg/bg branch) and if so, how the anchor should be refined to better match the underlying object (box branch). The second stage are the RCNN heads, where the oriented box proposal outputs of the RPN are used to RoI pool the features for class prediction (class branch) and further box refinement (box branch). In the one-stage RetinaNet version, the fg/bg branch is exchanged by a class branch that directly predicts the category for each of the anchor boxes. This makes the second stage obsolete.

For FRCNN, the RoI pooling layer pools from the oriented grid aligned with the input boxes. Since the output feature maps are upright, usual convolutions can be used in the subsequent layers.

In the following, we explain all adaptions and design choices in more detail.

**Box representation.** We use a five-parameter representation for boxes $b = (r, c, l_1, l_2, \phi)$, where $(r, c)$ denote the subpixel-precise center point row and column coordinates, $(l_1, l_2)$ the semi-axes lengths of the oriented box, and $\phi$ the orientation pointing in the direction of the major axis. $\phi$ is given as an angle in radians between the positive horizontal axis and the $l_1$-axis in mathematically positive sense. In contrast to the parametrization used in [129], we generally do not enforce that $l_1 \geq l_2$ such that sudden flips of the orientation for boxes with aspect ratio close to one are avoided. More precisely, only when we are not interested in the exact orientation (*IgnoreDirection* is true, see below), we use the longer box-side as $l_1$. In this case, we also swap the axes of output boxes such that $l_1 \geq l_2$ holds.

**Ignore direction.** We differentiate between two scenarios: The first scenario is that we are interested in the exact orientation of boxes, i.e., $\phi \in (-\pi, \pi]$. In the second scenario, $\phi$ is limited to the range $(-\frac{\pi}{2}, \frac{\pi}{2}]$, which is suitable if the actual orientation of the box is not important, but only a tight oriented bounding box of the instance (mask) is the goal. In this case, predictions with $\phi$ outside of the range can be corrected by subtracting or adding $\pi$. To differentiate the two scenarios, we use a parameter *IgnoreDirection* that is set to *true* in the latter case. Note that it is important that also the ground truth given in the dataset at hand is annotated in accordance with *IgnoreDirection*. In particular, if the oriented ground-truth boxes are generated as the smallest bounding boxes (SBB) of an instance mask, we do not know which of the four options is the correct direction of the orientation. Hence, in this case, we always use *IgnoreDirection* set to true.

## 8.3.1 RPN and Anchor Assignment

Because we want to predict oriented boxes, the region proposal network is fed with oriented anchors with different orientations, aspect ratios, and subscales. Ding et al. [31] argue that this leads to a massively higher number of anchors compared to axis-aligned boxes. However, in many cases, we can choose the orientation of the ground-truth (GT) boxes such that it is aligned with the longer box side length.[1] Therefore, in comparison to *AAD*, only aspect ratios smaller or equal to one are necessary, which almost halves the number of anchors. Moreover, if one is not interested in the exact orientation of the box (*IgnoreDirection* is true), three orientations such as $(-\frac{\pi}{3}, 0, \frac{\pi}{3})$ are sufficient in most situations. We did experiments using only three orientations in the case that the full orientation range is of interest. We found that using $(-\frac{2}{3}\pi, 0, \frac{2}{3}\pi)$ is often also sufficient when *IgnoreDirection* is set to false.

As usual, the anchor target assignment is based on the IoU between anchors and GT, as explained in Section 4.3. With a naive implementation, we noticed that for *OD* the

[1] However, this is not the case if the object's orientation is defined by a printing or other mark on the object and not given by its shape.

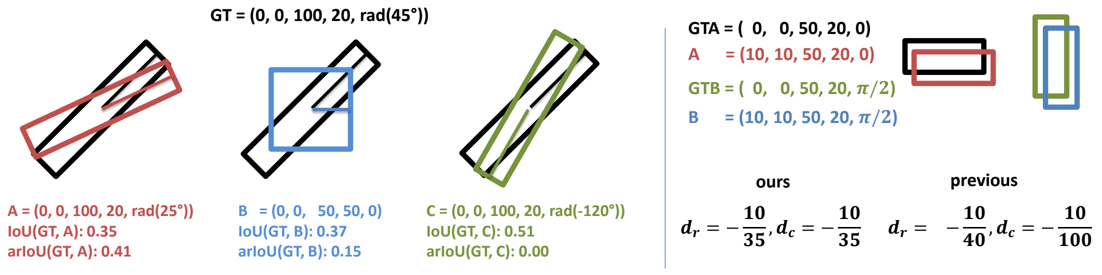

GT = (0, 0, 100, 20, rad(45°))
A = (0, 0, 100, 20, rad(25°))
IoU(GT, A): 0.35
arIoU(GT, A): 0.41
B = (0, 0, 50, 50, 0)
IoU(GT, B): 0.37
arIoU(GT, B): 0.15
C = (0, 0, 100, 20, rad(-120°))
IoU(GT, C): 0.51
arIoU(GT, C): 0.00
GTA = ( 0, 0, 50, 20, 0)
A = (10, 10, 50, 20, 0)
GTB = ( 0, 0, 50, 20, π/2)
B = (10, 10, 50, 20, π/2)
ours
previous
$d_r = -\frac{10}{35}, d_c = -\frac{10}{35}$
$d_r = -\frac{10}{40}, d_c = -\frac{10}{100}$

**Figure 8.2: arIoU and box target example.** (*Left*) GT box (*black*) and oriented anchors (*red, blue, green*). Although for the blue anchor the coordinates $l_1, l_2$, and $\phi$ differ substantially, the exact IoU is higher than for the red anchor where only $\phi$ differs by 20°. Moreover, the exact IoU for the green anchor is the highest, but its direction is almost opposite to the GT direction. These issues are solved by the use of the arIoU for anchor assignment. (*Right*) Our box center targets ensure that the targets are consistent for anchors with different orientation but the same center coordinates.

training is very unstable or converging to a bad local minimum due to the following reasons: On the one hand, the number of foreground anchors can be very low or in some cases there is not a single anchor assigned to a particular GT instance. On the other hand, the exact oriented box IoU sometimes leads to very inefficient assignments (cf. Fig. 8.2).

**Angle-related IoU.** To fix the latter problem, we replace the exact IoU by the angle-related IoU (arIoU [121]) for anchor assignment:

$$\text{arIoU}(A, B) = \max(0, \cos(\phi_A - \phi_B))\,\text{IoU}(\hat{A}, B), \tag{8.1}$$

where $\hat{A}$ is the box $(r_A, c_A, l_{1,A}, l_{2,A}, \phi_B)$. First, we see that anchors that point into the opposite direction of the GT box obtain an arIoU of zero although their naïve IoU might be large. This helps to avoid very large targets $d_\phi$ for the orientation regression. Moreover, the right part of (8.1) can be evaluated as an IoU between axis-aligned boxes (since the axis can be chosen parallel to $l_{1,B}$) and is therefore much more efficient to compute than the exact IoU of two oriented boxes.

**Set weak boxes to background.** Although, the arIoU can fix the problem of wrong anchor assignment, the second problem of a low number of foreground instances remains because its values can still become small for anchors that have only slightly different parameters than the GT box. To fix this problem and increase the number of foreground anchors, we evaluate the influence of *setting weak boxes to background (swb2bg)*: If *swb2bg* is set to *false*, anchors that achieve the highest arIoU with a GT box, but where the arIoU is below *fgNegThresh*, are still assigned to the foreground (cf. Fig. 4.15). By changing *swb2bg*, we ensure that all GT instances contribute to the training, even if none of the anchors fits very well. Hence, instead of increasing the number of anchors, this change offers an efficient way to address all available GT box shapes. Effectively, this leads to a more stable training because enough foreground examples are present. Moreover, if for some GT boxes no anchor is assigned to the foreground, the model is told to learn that no object is present at that position although there obviously is one and the image features are indicating it. This can lead to an unstable training.

In theory, more anchors could also be assigned to the foreground by relaxing the negative and positive thresholds *fgNegThresh* and *fgPosThresh*. However, we found that these hyperparameters are much more difficult to tune and we could not improve the overall results by adapting them.

The relaxation of *swb2bg* is only used within the RPN. For the RCNN heads (if present), we only want good box proposals to generate a loss. This is in line with the findings of Cascade R-CNN [14], where the assignment thresholds are increased in several iterative box refinements.

**Oriented box targets.** The correct definition of training targets is a crucial ingredient for oriented box detection, as minor modifications can have a large impact on the training convergence. We use the following box target calculations for the row and column coordinate deltas:

$$d_r = \frac{r^* - r}{\bar{l}}, \; d_c = \frac{c^* - c}{\bar{l}}, \tag{8.2}$$

where $(r^*, c^*)$ are coordinates of the GT box, $(r, c)$ are anchor or box-proposal coordinates, and $\bar{l} = (l_1 + l_2)/2$ is the mean axis length of the anchor or box-proposal for which the deltas should be calculated. By using $\bar{l}$, the normalization does not depend on the orientation of the box. In previous work, such as [31, 129], $l_2$ is used to normalize $d_r$ and $l_1$ to normalize $d_c$ instead of the mean value $\bar{l}$. However, if a box is oriented with $\phi \geq \frac{\pi}{2}$ the meaning of $l_1$ and $l_2$ with respect to $r$ and $c$ is flipped. This is avoided by the use of $\bar{l}$.

For the target delta of the orientation $\phi$, we use the angular difference, but we make sure that the smaller of both possible difference angles is used:

$$d_\phi = \begin{cases} \phi^* - \phi, & |\phi^* - \phi| < T_\phi \\ \phi^* - \phi + 2\,\mathrm{sgn}(\phi) T_\phi, & \text{otherwise,} \end{cases} \tag{8.3}$$

where $T_\phi = \pi/2$ if *IgnoreDirection* is true and otherwise $T_\phi = \pi$. Note that (8.3) can lead to predictions that have an orientation outside of the possible range and hence we need to correct them by adding or subtracting $2T_\phi$ as long as $\phi$ is not within the angle range ($(-\pi, \pi]$ or $(-2\pi, 2\pi]$ if *IgnoreDirection* is true or false, respectively).

Moreover, in the calculation of $d_\phi$, we avoid large regression targets by ensuring that $d_\phi$ is in the range $(-\frac{\pi}{2}, \frac{\pi}{2})$ due to the use of the arIoU during the assignment phase.[2] In practice, the targets are often smaller, depending on the number of anchor orientations.

**Classes without orientation.** In many applications, we have objects without a uniquely defined orientation. If we use the smallest oriented bounding box as GT for symmetric or round objects, their GT orientations are varying between different instances without a

[2]The interval boundaries are excluded here because an angular difference with $|d_\phi| = \frac{\pi}{2}$ leads to an arIoU of zero (see (8.1)).

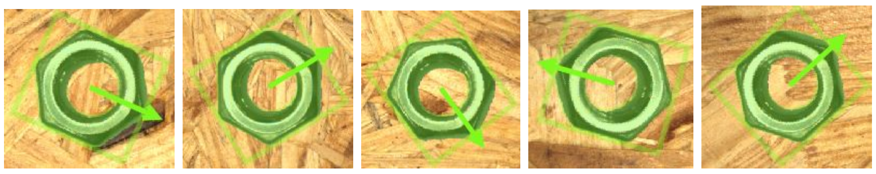


**Figure 8.3: Example for *classes without orientation.*** For a symmetric object like the shown nut, the correct orientation is ambiguous to annotate. Therefore, we assign it to be a *class without orientation*.

hint for the model why this is the case. This can lead to a destabilization of the training. Therefore, we propose to assign these classes to a set of *classes without orientation*.

For these categories, we annotate the GT boxes as the smallest AABB of the GT instance masks with $\phi = 0$. An example is shown in Fig. 8.3.

### 8.3.2 Evaluation of Oriented Box Detection Methods

Of course, generally, *OD* methods can also be evaluated by *AP* or $AP^\star$ (cf. Chapter 4, Chapter 6). However, the additional component of the box orientation brings a new challenge. Moreover, the direct comparison of *OD* and *AAD* approaches only by looking at *AP* or $AP^\star$ is often not very meaningful, as described in the following.

**Score of angular precision — *SoAP*.** The normal evaluation with *AP* or $AP^\star$ might in some cases be misleading for oriented boxes. Also here, a true positive (TP) is defined as a prediction that exceeds the given IoU threshold with a GT box and has the correct class. However, the IoU can be large or even one although the two boxes point into opposite directions. Therefore, we can additionally measure the score of angular precision (SoAP):

$$SoAP = \sum_{B \in \mathcal{B}_{\text{IoU}>0}} \min(|\phi_B - \phi_G|, 2\pi - |\phi_B - \phi_G|), \tag{8.4}$$

$$SoAP_{TP} = \sum_{B \in \mathcal{B}_{TP}} \min(|\phi_B - \phi_G|, 2\pi - |\phi_B - \phi_G|), \tag{8.5}$$

where $\phi_G$ is the orientation of the matched GT box $G$ if the prediction $B$ is a TP and otherwise it is the GT box $G$ with which $B$ overlaps the most. $\mathcal{B}_{\text{IoU}>0}$ is the set of predictions that overlap with at least one GT box and hence *SoAP* excludes background boxes that do not overlap with any GT box. $SoAP_{TP}$ only includes TP predictions for the given IoU threshold. In the same way as in the definitions of *AP* or $AP^\star$, *SoAP* or $SoAP_{TP}$ can be an average over the respective measure at multiple IoU thresholds.

**Relative IoU for bounding boxes.** As we will see later in Chapter 10, oriented box detection might be only a subpart of a larger network. In this particular case, we are not interested in the orientation of boxes, but want to make use of *OD* because of the tighter fit of the oriented boxes around the object mask compared to *AAD*. However, it is not straight forward to select a good model using the normal *AP* or $AP^\star$ metric based on the usual IoU computations. In some cases, the box that leads to a maximum IoU with the instance's mask has no unique orientation. There might also be bounding boxes with different orientations that lead to almost the same IoU between the box and the mask.

To this end, Böttger et al. [12] introduced the relative IoU for axis-aligned and oriented boxes that compares the actual IoU of a box $B$ and a mask $M$ to the best possible IoU that can be achieved for $M$ by any box $B'$, $\mathrm{IoU}_{opt}(M)$:

$$\mathrm{IoU}_{opt}(M) = \max_{B' \in \mathcal{B}} \mathrm{IoU}(B', M), \tag{8.6}$$

$$\mathrm{rIoU}(B, M) = \frac{\mathrm{IoU}(B, M)}{\mathrm{IoU}_{opt}(M)}, \tag{8.7}$$

where $\mathcal{B}$ is the set of all possible boxes with any parameters $b$ and are not necessarily bounding boxes of the mask $M$. In particular, independent of the shape of the mask $M$, rIoU leads to a value between zero and one, where one is always achievable (e.g., by $B_{opt}$ that solves (8.6)). Hence, if instance masks are available, we can replace IoU by rIoU during the evaluation and avoid penalizing boxes disproportionally that have a different orientation than the ground truth box but still achieve a high IoU with the ground-truth instance mask.

However, using rIoU as in (8.7) as the objective for model optimization leads to box predictions that are generally no bounding boxes, but cut away thin parts of the mask boundary that do not contribute much to the overall mask area. Further, $\mathrm{IoU}_{opt}(M)$ cannot be computed in closed form and needs a multi-start iterative optimization scheme. Most instance-segmentation methods, like MRCNN or RetinaMask, require bounding boxes of the mask because the mask prediction is only carried out within the predicted box. Therefore, we propose the usage of the relative bounding box IoU:

$$\mathrm{SBB}(M, \phi) = \mathrm{argmax}_{B' \in \mathcal{B}(M, \mathrm{BB}, \phi)} \mathrm{IoU}(B', M), \tag{8.8}$$

$$\mathrm{SBB}(M) = \mathrm{argmax}_{B' \in \mathcal{B}(M, \mathrm{BB})} \mathrm{IoU}(B', M), \tag{8.9}$$

$$\mathrm{rIoU}_{\mathrm{BB}}(B, M) = \min\left(\mathrm{IoU}\left(B, \mathrm{SBB}(M, \phi_B)\right), \min\left(1, \frac{\mathrm{IoU}(B, M)}{\mathrm{IoU}(\mathrm{SBB}(M), M)}\right)\right), \tag{8.10}$$

where $\mathcal{B}(M, \mathrm{BB}, \phi)$ in (8.8) is the set of bounding boxes of $M$ with orientation $\phi$. Hence, $\mathrm{SBB}(M, \phi)$ is the smallest bounding box of $M$ with orientation $\phi$. It can be computed by rotating $M$ by $-\phi$ and computing the smallest axis-aligned box around the rotated $M$ before transforming the coordinates back with the inverse rotation. $\mathrm{SBB}(M)$ from (8.9) is the bounding box with any orientation that achieves the maximum IoU for $M$. It is hence the *smallest bounding box* (SBB) of the mask $M$. Finally, $\mathrm{rIoU}_{\mathrm{BB}}(B, M)$ is computed as the minimum of two components: The first component on the left reflects how well the box $B$ is a bounding box for the mask $M$. Here, we compute the axis-aligned IoU between the box $B$ and the smallest bounding box of $M$ with the orientation $\phi_B$ of $B$. The second component on the right reflects the goodness of the IoU between the box $B$ and the mask $M$. Since we want to obtain an optimum for bounding boxes, we set the value to one whenever $\mathrm{IoU}(B, M)$ is higher than the IoU of $\mathrm{SBB}(M)$ with $M$. Also here, $\mathrm{SBB}(M)$ can be computed efficiently, e.g., using the *rotating calipers* algorithm [170, 189].

## 8.4 Experiments

In this section, we show results on different datasets that have been labeled for oriented detection (e.g., *Screws*) or that have accurate mask annotations from which the OBBs can be inferred (e.g., *Pill Bags* or *D2S*). In particular, we provide ablation studies for the design choices that we made and explained in the previous section. For all experiments within this section, we show the results of the *best* model state that is selected by early stopping (cf. Section 2.3.4). For each model, the results of three experiments (Exp) that only differ in the random seed for weight initialization and SGD-mini-batch selection are shown.

### 8.4.1 Screws

As described in Section 4.1.2, *Screws* contains thirteen different categories of screws and nuts on a wooden background. The four nut categories are symmetric such that their orientation is ambiguous to annotate. Hence, they are assigned to be *classes without orientation*. In contrast, for all screw categories, the orientation can be uniquely defined to point from the head along the thread to the tail of the screw.

The dataset is split in the ratio 70% : 15% : 15% to the training, validation, and test sets. All models are trained on images of size $512 \times 384$ for 60 epochs with batch size two. During the training the images are mirrored vertically or horizontally (or both) with a probability of 50%. The initial learning rate is set to 0.001 and reduced by a factor of 0.1 after 30 and 50 epochs. We use a warmup (cf. Section 2.3.1) with a factor of 0.3333 for 500 iterations. The momentum and weight decay are set to 0.9 and $10^{-5}$, respectively. Unless mentioned otherwise, we use a maximum number of detections of 60 and a minimum confidence threshold of 0.05. During NMS, we filter boxes of the same class if they have a lower score and exceed the IoU threshold of 0.4 and of any class if they exceed the IoU threshold of 0.5. For RetinaNet, we use FPN levels three to seven and generally use anchors with three subscales, aspect ratios $(0.1, 0.3, 0.9)$, and angles $(-2\pi/3, -\pi/3, 0, \pi/3, 2\pi/3, \pi)$. This results in a quite high number of $3 \cdot 3 \cdot 6 = 54$ anchors per pixel in the FPN feature maps. With the given image size and chosen levels, this leads to an overall number of $(64 \cdot 48 + 32 \cdot 24 + 16 \cdot 12 + 8 \cdot 6 + 4 \cdot 3) \cdot 54 = 220\,968$ anchor boxes per image.

**Influence of backbone and pretrained weights.** Also for *OD* models, the choice of backbone architecture and pretraining of model weights can have a large influence on the quality of results as depicted in Table 8.1. The models in the top part of the table have class-specific score thresholds $T_s^\star$ to optimize $AP_{50}^\star$ on the validation set (cf. Chapter 6).

A memory and runtime efficient choice is to use a SqueezeNet [81] backbone. On *Screws*, this leads to comparable results as when a ResNet 50 [68] is used. The use of an even larger ResNet 101 backbone slightly increases $AP^\star$, especially at higher IoU thresholds, mainly due to a reduced number of FPs. However, at IoU threshold 0.5, the recall is lower than for the ResNet 50 counterparts.

| Model | Exp | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *AP* | *AP*$^\star$ | *AP* | *AP*$^\star$ | *TP* | *FP* | *AP* | *AP*$^\star$ | *TP* | *FP* |
| With $T_s^\star(val)$ | | | | | | | | | | | |
| SqueezeNet *OpenImages* | 1 | 65.6 | 63.2 | 91.8 | 90.0 | 618 | 39 | 65.7 | 62.4 | 494 | 163 |
| | 2 | 65.3 | 62.9 | 92.8 | 91.1 | 626 | 44 | 63.7 | 59.8 | 478 | 192 |
| | 3 | 65.4 | 63.1 | 92.7 | 91.3 | 626 | 35 | 62.4 | 58.6 | 475 | 186 |
| ResNet50 *OpenImages* | 1 | 66.3 | 63.7 | 91.6 | 89.5 | 616 | 43 | 66.1 | 62.8 | 492 | 167 |
| | 2 | 65.9 | 63.1 | 92.3 | 90.1 | 617 | 42 | 63.2 | 59.4 | 474 | 185 |
| | 3 | 65.4 | 63.6 | 90.8 | 89.8 | 608 | 22 | 63.7 | 60.9 | 480 | 150 |
| ResNet50 *ImageNet* | 1 | 69.0 | 66.0 | 94.0 | 91.8 | 629 | 41 | 69.7 | 65.5 | 510 | 160 |
| | 2 | 68.1 | 65.4 | 91.9 | 89.8 | 615 | 37 | 68.6 | 65.0 | 502 | 150 |
| | 3 | 67.9 | 65.1 | 92.5 | 90.4 | 617 | 35 | 68.8 | 65.5 | 509 | 143 |
| ResNet50 *COCO* | 1 | 74.8 | 72.6 | 96.2 | 95.3 | 640 | 18 | 79.0 | 75.3 | 559 | 99 |
| | 2 | 74.7 | 72.2 | 95.9 | 94.3 | 638 | 27 | 79.2 | 75.2 | 559 | 106 |
| | 3 | 74.4 | 72.6 | 95.4 | 94.7 | 635 | 14 | 80.0 | 76.9 | 564 | 85 |
| ResNet101 *ImageNet* | 1 | 68.6 | 66.1 | 90.9 | 89.2 | 607 | 35 | 72.5 | 68.6 | 520 | 122 |
| | 2 | 68.7 | 66.8 | 92.2 | 91.3 | 615 | 18 | 71.1 | 67.9 | 511 | 122 |
| | 3 | 67.7 | 65.7 | 90.8 | 89.9 | 606 | 19 | 70.2 | 66.9 | 507 | 118 |
| ResNet101 *COCO* | 1 | 74.9 | 73.3 | 94.5 | 94.2 | 629 | 11 | 80.5 | 77.5 | 562 | 78 |
| | 2 | 75.8 | 73.7 | 95.0 | 93.8 | 631 | 22 | 81.4 | 78.0 | 566 | 87 |
| | 3 | 75.3 | 73.7 | 94.5 | 94.0 | 628 | 12 | 82.1 | 79.6 | 572 | 68 |
| Without $T_s^\star(val)$ | | | | | | | | | | | |
| SqueezeNet *OpenImages* | 1 | 67.9 | 44.9 | 96.0 | 63.2 | 651 | 2388 | 67.1 | 44.7 | 505 | 2534 |
| | 2 | 67.0 | 44.0 | 96.2 | 62.9 | 650 | 2409 | 64.7 | 42.4 | 486 | 2573 |
| | 3 | 67.2 | 43.5 | 95.8 | 61.9 | 651 | 2302 | 63.6 | 41.0 | 484 | 2469 |
| ResNet101 *COCO* | 1 | 76.8 | 48.2 | 98.0 | 61.9 | 651 | 2069 | 81.9 | 51.1 | 572 | 2148 |
| | 2 | 77.3 | 48.7 | 97.6 | 61.9 | 649 | 1990 | 82.4 | 51.7 | 574 | 2065 |
| | 3 | 77.2 | 48.5 | 97.9 | 62.0 | 651 | 2012 | 83.5 | 52.3 | 583 | 2080 |

**Table 8.1: Influence of backbones and pretrained weights on *Screws* test.** All models are RetinaNet. Also for *OD* models, the chosen backbone and pretrained model weights can have a large influence on the results. Only for *COCO*-pretrained models, the weights of the FPN and the class and box regression heads are pretrained; for all others they are initialized randomly.

Pretraining of weights has a much larger impact on the model's performance than the backbone choice. *AP* and *AP*$^\star$ can be improved by approximately three percentage points (pp) when the ResNet 50 backbone is pretrained on *ImageNet* [168] instead of *OpenImages* [98]. Moreover, a remarkable additional gain of approximately six pp is achieved when the whole model has been pretrained on *COCO* [118]. In this case, the main advantage might not come from better features extracted by the backbone, but from the fact that the FPN, class-head, and box-regression-head layers are also initialized with pretrained weights from a model that has already learned to predict boxes and classify them. Note that in this case, due to the different number of anchors and classes used on *COCO* and *Screws*, the last prediction convolution layers of both the class and box heads are still initialized randomly due to the mismatch of the number of kernels. A large advantage of *COCO*-pretrained weights is visible at IoU threshold 0.8, while the number of TPs is only slightly higher at IoU threshold 0.5. Hence, the boxes are predicted with a better localization accuracy than, e.g., for *ImageNet*-pretrained weights.

| Model | Exp | IoU[0.5:0.95] | | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $AP$ | $AP^\star$ | $SoAP$ | $AP$ | $AP^\star$ | $TP$ | $FP$ | $AP$ | $AP^\star$ | $TP$ | $FP$ |
| RNet R50 *COCO* | 1 | 74.8 | 72.6 | 84.7 | 96.2 | 95.3 | 640 | 18 | 79.0 | 75.3 | 559 | 99 |
| | 2 | 74.7 | 72.2 | 86.2 | 95.9 | 94.3 | 638 | 27 | 79.2 | 75.2 | 559 | 106 |
| | 3 | 74.4 | 72.6 | 84.8 | 95.4 | 94.7 | 635 | 14 | 80.0 | 76.9 | 564 | 85 |
| RNet R50 *COCO* FA | 1 | 74.4 | 71.2 | 82.6 | 96.0 | 94.0 | 638 | 37 | 78.1 | 73.1 | 547 | 128 |
| | 2 | 74.2 | 71.2 | 85.1 | 95.3 | 93.1 | 634 | 40 | 79.6 | 74.9 | 556 | 118 |
| | 3 | 74.9 | 71.6 | 85.5 | 96.2 | 94.0 | 640 | 40 | 78.2 | 73.1 | 551 | 129 |
| RNet R101 *COCO* | 1 | 74.9 | 73.3 | 89.6 | 94.5 | 94.2 | 629 | 11 | 80.5 | 77.5 | 562 | 78 |
| | 2 | 75.8 | 73.7 | 91.0 | 95.0 | 93.8 | 631 | 22 | 81.4 | 78.0 | 566 | 87 |
| | 3 | 75.3 | 73.7 | 86.6 | 94.5 | 94.0 | 628 | 12 | 82.1 | 79.6 | 572 | 68 |
| RNet R101 *COCO* FA | 1 | 73.6 | 71.0 | 89.1 | 94.2 | 92.7 | 627 | 29 | 77.0 | 73.4 | 543 | 113 |
| | 2 | 74.5 | 71.8 | 88.3 | 94.9 | 93.4 | 631 | 30 | 79.6 | 75.8 | 556 | 105 |
| | 3 | 74.3 | 71.2 | 85.8 | 94.9 | 93.0 | 633 | 37 | 78.6 | 74.2 | 552 | 118 |

**Table 8.2: *OD* with few anchors (FA).** FA models use only 18 instead of 54 anchor boxes. FA models have only slightly lower $AP^\star$, but more FPs, especially at higher IoU thresholds.

| Model | Exp | NVIDIA GTX 1080Ti | | NVIDIA RTX 2080Ti | |
|---|---|---|---|---|---|
| | | mem[MB] | time[ms] | mem[MB] | time[ms] |
| RNet R50 *COCO* | 1 | 2981 | 66 | 3235 | 55 |
| RNet R50 *COCO* FA | 1 | 2779 | 45 | 3025 | 32 |
| RNet R101 *COCO* | 1 | 4099 | 77 | 4455 | 63 |
| RNet R101 *COCO* FA | 1 | 3897 | 56 | 4245 | 38 |

**Table 8.3: Performance of *OD* with few anchors (FA).** FA models need less memory on the GPU and can be significantly faster. Shown are the average runtimes per image across the *Screws* test set. The batch size has been set to one. Both GPUs are built into the same server and hence use the same CPUs (Intel Xeon Silver 4114 CPU@2.2GHz).

For completeness, we also show results of the worst and the best configuration where $T_s^\star(val)$ is replaced by a low global confidence threshold of 0.05 in the bottom part of Table 8.1. The $AP$ values are increased with a slightly higher overall recall, but the number of FPs is increased up to a factor of almost 200.

***OD* with fewer anchors.** Because for *OD*, anchor boxes have to be setup with different orientations, generally, more anchors are necessary than for *AAD*. When low FPN levels with high feature map resolutions are used, the number of anchors can have a large impact on the training and inference runtime. Therefore, in this paragraph, we compare our baseline models against models with significantly fewer anchors (FA). Instead of six orientations $(-2\pi/3, -\pi/3, 0, \pi/3, 2\pi/3, \pi)$ and three aspect ratios $(0.1, 0.3, 0.9)$, for FA models, we only use three orientations $(-2\pi/3, 0, 2\pi/3)$ and two aspect ratios $(0.25, 0.9)$. This leads to only 18 instead of 54 different anchor boxes per feature map pixel and 73 656 instead of 220 968 anchors per image.

Table 8.2 shows that by using fewer anchors, the quality of the results is only marginally lower. Generally, FA models predict a bit more FPs, but have a comparable recall to the models with a higer number of box templates. However, as shown in Table 8.3, reducing the number of anchors leads to approximately 200MB less memory

| Model | Exp | IoU[0.5:0.95] | | | | IoU[0.5] | | | | | train |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $AP$ | $AP^{\star}$ | $SoAP$ | $SoAP_{TP}$ | $AP$ | $AP^{\star}$ | $TP$ | $FP$ | $SoAP_{TP}$ | time [s] |
| | 1 | 74.8 | 72.6 | 84.7 | 98.3 | 96.2 | 95.3 | 640 | 18 | 97.3 | 4234 |
| RNet R50 arIoU | 2 | 74.7 | 72.2 | 86.2 | 98.5 | 95.9 | 94.3 | 638 | 27 | 97.4 | 4024 |
| | 3 | 74.4 | 72.6 | 84.8 | 98.5 | 95.4 | 94.7 | 635 | 14 | 97.5 | 4590 |
| | 1 | 49.8 | 47.5 | 77.6 | 92.3 | 73.5 | 70.4 | 530 | 192 | 89.7 | 8508 |
| RNet R50 exact IoU | 2 | 50.5 | 47.9 | 74.7 | 92.0 | 73.8 | 70.4 | 537 | 279 | 90.1 | 8349 |
| | 3 | 49.1 | 46.9 | 78.0 | 94.7 | 72.5 | 69.7 | 534 | 188 | 89.4 | 8679 |

**Table 8.4: Comparison of** arIoU **and exact** IoU **for anchor assignment.** RNet R50 arIoU is the same as ResNet 50 *COCO* from Table 8.1. The exact IoU model uses the same initialization and hyperparameters, but arIoU is replaced by the exact OBB IoU during anchor assignment.

consumption on the GPU.[3] Moreover, a speedup of 30% on the older NVIDIA GTX 1080Ti GPU, and of 40% on the newer NVIDIA RTX 2080Ti GPU can be achieved. Hence, the overall performance of a model with fewer anchors that includes accuracy, runtime, and memory consumption, might even be better than for a model with many anchor boxes.

**Angle-related IoU vs. exact IoU.** To evaluate the influence of using arIoU during anchor assignment, we take the exact same model as ResNet 50 *COCO* from Table 8.1 and replace the arIoU computation within the anchor assignment by the exact OBB IoU. All other hyperparameters are the same. The results in Table 8.4 show a remarkable improvement of the accuracy with respect to $AP^{\star}$ and $SoAP$ for the arIoU model: The orientation of the predictions is much more accurate and also the recall is significantly higher.

The usage of the arIoU for anchor assignment does not only lead to higher $AP^{\star}$ values, but is also much more efficient. As shown in the last column of Table 8.4, the total training time can be halved if the arIoU is used. The runtime comparison has been done on the same machine, where for both methods the anchor assignment is computed on the CPU (Intel Xeon E5-2637 v4 @3.5GHz), but the remaining parts of the CNN are run on the GPU (NVIDIA GTX 1080 Ti). Moreover, with a GPU implementation of the box target generation, the arIoU training time can be further reduced to approximately 1600 seconds ($\approx$ 27min). Note that this difference is present even though for both IoU types we directly set the IoU to zero if an anchor has no chance to overlap a GT box. Therefore, we first check the distance of each anchor center to each GT box center within the IoU computation and only compute the box IoU or arIoU if there is any chance that the two boxes overlap.

**Influence of *SetWeakBoxesToBg* during anchor assignment.** Generally, only anchors with high IoU to a GT box are assigned to the foreground (cf. Fig. 4.15). As we have seen, in comparison to AABBs, slight differences in orientations of OBBs can reduce the IoU to a GT box significantly. This can lead to a low number of foreground anchors per

[3]We explicitly choose the same type of cuDNN convolution algorithm throughout the network for this comparison. In practice, if the convolution algorithm is not set deterministically, the internal cuDNN optimization might lead to varying memory consumptions across models and GPUs that do not always reflect the theoretical number of weights and activations of a model.

| SWB | Neg | Pos | Exp | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | $AP$ | $AP^\star$ | $AP$ | $AP^\star$ | $TP$ | $FP$ | $AP$ | $AP^\star$ | $TP$ | $FP$ |
| RetinaNet R50 *ImageNet*-pretrained | | | | | | | | | | | | | |
| ✓ | | | 1 | 58.5 | 56.7 | 78.6 | 77.7 | 528 | 61 | 59.4 | 56.9 | 434 | 155 |
| | | | 2 | 59.2 | 57.2 | 79.0 | 77.9 | 530 | 45 | 60.1 | 57.1 | 435 | 140 |
| | | | 3 | 59.0 | 57.1 | 79.3 | 78.3 | 532 | 58 | 61.4 | 58.7 | 442 | 148 |
| | | | 1 | 69.0 | 66.0 | 94.0 | 91.8 | 629 | 41 | 69.7 | 65.5 | 510 | 160 |
| | | | 2 | 68.1 | 65.4 | 91.9 | 89.8 | 615 | 37 | 68.6 | 65.0 | 502 | 150 |
| | | | 3 | 67.9 | 65.1 | 92.5 | 90.4 | 617 | 35 | 68.8 | 65.5 | 509 | 143 |
| RetinaNet R50 *COCO*-pretrained | | | | | | | | | | | | | |
| ✓ | | | 1 | 64.9 | 63.6 | 81.0 | 80.5 | 541 | 233 | 70.4 | 68.3 | 491 | 283 |
| | | | 2 | 65.1 | 63.5 | 81.2 | 80.4 | 542 | 227 | 70.9 | 68.4 | 492 | 277 |
| | | | 3 | 64.8 | 63.4 | 81.0 | 80.6 | 541 | 219 | 70.4 | 68.2 | 492 | 268 |
| ✓ | ✓ | | 1 | 73.1 | 71.3 | 94.1 | 93.5 | 627 | 17 | 77.8 | 74.9 | 554 | 90 |
| | | | 2 | 72.3 | 70.6 | 93.0 | 92.2 | 618 | 16 | 76.7 | 73.8 | 547 | 87 |
| | | | 3 | 72.3 | 70.3 | 93.8 | 92.7 | 625 | 22 | 77.1 | 74.0 | 550 | 97 |
| ✓ | | ✓ | 1 | 63.1 | 61.7 | 81.3 | 80.8 | 543 | 141 | 67.4 | 65.3 | 480 | 204 |
| | | | 2 | 62.9 | 61.5 | 81.2 | 80.7 | 543 | 128 | 67.6 | 65.6 | 483 | 188 |
| | | | 3 | 63.0 | 61.3 | 81.3 | 80.3 | 543 | 134 | 67.9 | 65.5 | 484 | 193 |
| ✓ | ✓ | ✓ | 1 | 71.3 | 69.3 | 94.4 | 93.6 | 627 | 15 | 75.3 | 72.1 | 535 | 107 |
| | | | 2 | 72.5 | 70.7 | 95.0 | 94.7 | 631 | 10 | 75.7 | 73.0 | 536 | 105 |
| | | | 3 | 71.2 | 69.2 | 93.8 | 93.0 | 625 | 19 | 73.8 | 70.6 | 527 | 117 |
| | | | 1 | 74.8 | 72.6 | 96.2 | 95.3 | 640 | 18 | 79.0 | 75.3 | 559 | 99 |
| | | | 2 | 74.7 | 72.2 | 95.9 | 94.3 | 638 | 27 | 79.2 | 75.2 | 559 | 106 |
| | | | 3 | 74.4 | 72.6 | 95.4 | 94.7 | 635 | 14 | 80.0 | 76.9 | 564 | 85 |
| | ✓ | | 1 | 74.0 | 72.5 | 94.3 | 93.9 | 628 | 11 | 80.0 | 77.4 | 562 | 77 |
| | | | 2 | 73.6 | 72.0 | 94.6 | 94.2 | 630 | 11 | 78.0 | 75.5 | 557 | 84 |
| | | | 3 | 74.4 | 72.8 | 95.5 | 94.9 | 635 | 12 | 78.6 | 76.0 | 555 | 92 |

**Table 8.5: Influence of *SetWeakBoxesToBg* and anchor assignment thresholds.** A checkmark (✓) indicates that *SetWeakBoxesToBg* (SWB) is set to true, that *FgNegThresh* (Neg) is reduced to 0.3, or that *FgPosThresh* (Pos) is reduced to 0.4. Using our default *SetWeakBoxesToBg* set to *false* clearly gives the best results. Reducing *FgNegThresh* or *FgPosThresh* does not match the performance of *SetWeakBoxesToBg* set to *false*.

image and can lead to GT boxes where not a single anchor is assigned to the foreground. To compensate this effect, especially for elongated objects such as screws, we would need a large set of anchors, where for all GT boxes at least one anchor has an IoU above *FgNegThresh*. However, we have seen above that this increases the memory consumption and runtime. A workaround is to set *SetWeakBoxesToBg* to *false* such that at least the best fitting anchor for each GT box is set to the foreground. Alternatively, to increase the number of foreground anchors, we can reduce the thresholds *FgNegThresh* and *FgPosThresh*, respectively, or both at the same time. In Table 8.5, we investigate the influence of each parameter on *Screws*.

For RetinaNet with a ResNet 50 backbone and the given anchor configuration, the best performance is reached when *SetWeakBoxesToBg* is set to *false*. When *ImageNet*-pretrained weights are used, setting *SetWeakBoxesToBg* to *true* lowers *AP* and *AP*$^\star$ by approximately 9 pp. For *COCO*-pretrained weights, the effect is even slightly larger.

Further, by reducing *FgNegThresh* from 0.4 to 0.3, the difference can be reduced to approximately 1 pp *AP*$^\star$, but still the recall is lower. Setting *FgPosThresh* from 0.5 to 0.4 in order to increase the number of anchors that are assigned to the foreground leads to significantly worse results. A combination of setting *SetWeakBoxesToBg* to *false* together with the reduction of *FgNegThresh* leads to similar *AP*$^\star$ values with a higher precision but slightly lower recall.

Overall, this shows that setting *SetWeakBoxesToBg* to *false* is a good alternative to the use of an overly large set of anchor boxes. Since the reduction of *FgNegThresh* did not significantly improve the results, we leave it at its default value of 0.4 in all remaining experiments (also for other datasets).

**Qualitative results.** Fig. 8.4 shows qualitative results of *OD* on *Screws* for a selection of the above mentioned models. In the tables from the previous paragraphs, these are the models of the first experiment (Exp 1), respectively.

The exact IoU model fails to predict a correct orientation for many objects, even within the easier images where the screws and nuts are clearly separated. Often, for one screw there are several predictions with varying orientations where most of them hardly overlap the underlying screw.

On easier images, the SqueezeNet model predicts similar results to the ResNet 101 model, but in some cases the boxes are less accurate, in particular for the longer screws. In some cases, the direction of the box is opposite to the ground truth, i.e., pointing from tail to head instead of from head to tail.

When screws are overlapping and the overlapped screw is separated into two connected components, the models often make two separate wrong predictions for each visible screw part. Also when the objects are touching or are lying very close to each other, all models have difficulties.

Similar to the *AAD* detection results on *D2S* in Chapter 6, also here $T_s^\star$ does not generalize very well from the validation to the test set such that some correct predictions might be filtered out due to a confidence below the class-specific thresholds.

### 8.4.2 Pill Bags

*Pill Bags* (cf. Section 4.1.2) is a rather easy dataset with low intra-class variations in which all images have a similar appearance. However, also here we can see differences between *AAD* and *OD* methods. Some pills have an elliptic or elongated shape and thus, if they are diagonally aligned, OBBs certainly have a better fit than AABBs. Since *Pill Bags* is labeled with high-quality instance masks, we can compare the *OD* and *AAD* methods using $\mathrm{rIoU}_{\mathrm{BB}}$. Moreover, we showcase the advantage of *OD* when tuning NMS thresholds.

On *Pill Bags* we set *IgnoreDirection* to true because for most of the pills, the exact direction of the pill is only uniquely defined modulo $\pi$. Moreover, we set round pill

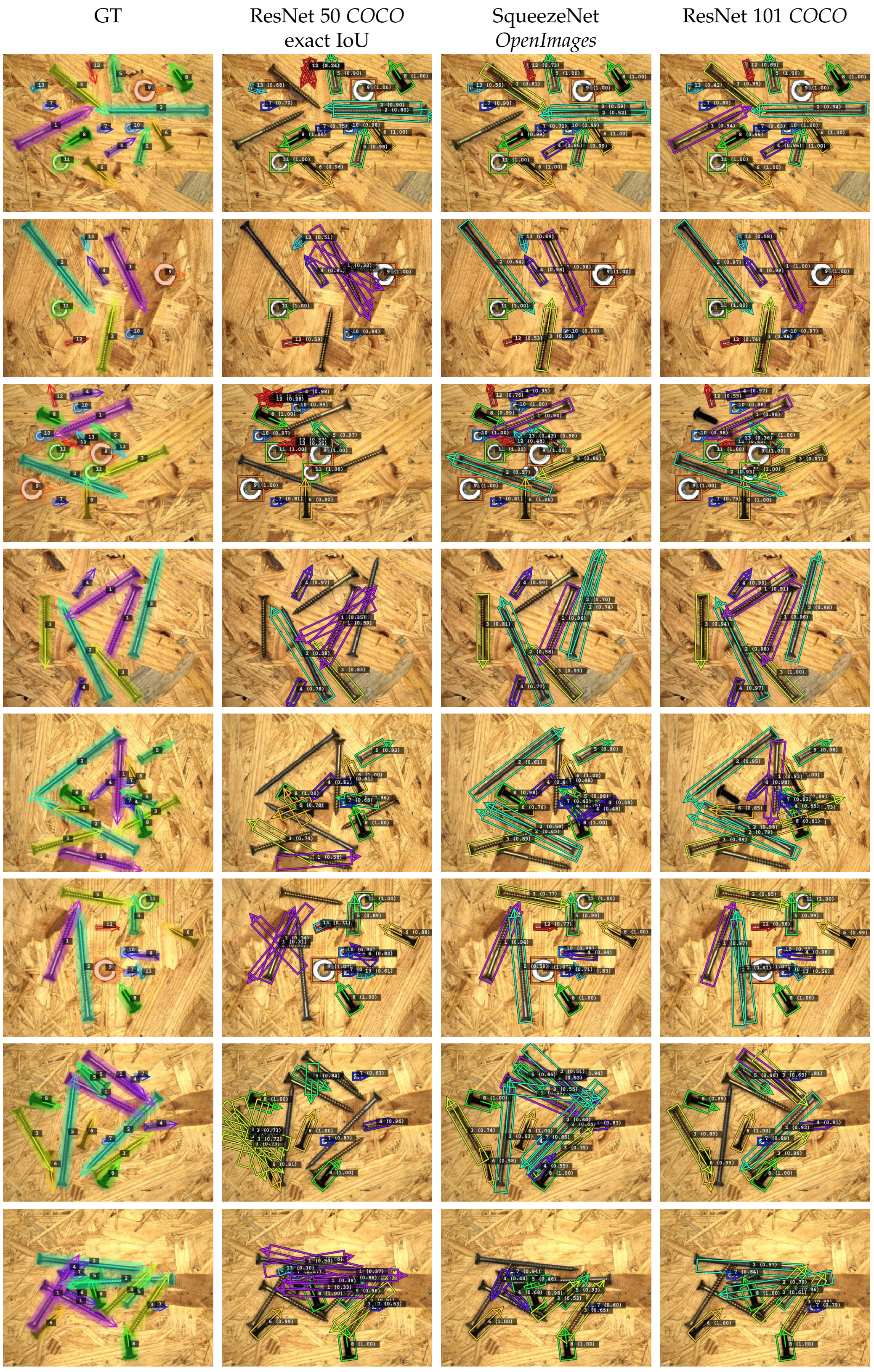


**Figure 8.4: Qualitative results on *Screws* test with $T_s^\star(val)$.** ResNet 50 *COCO* trained with exact IoU for anchor assignment shows significantly worse results. For most images the ResNet 101 *COCO* model shows more accurate results than the SqueezeNet *OpenImages* model. All models have difficulties with occluding and touching screws.

types to *classes without orientation* and for them we use the box parameters of the smallest axis-aligned bounding box of the instance mask as GT. For all other categories, the GT orientations and box parameters are obtained as those of the smallest oriented bounding box of the instance mask.

Unless mentioned otherwise, all models are trained with the following hyperparameters: Initial learning rate 0.001, momentum 0.9, weight decay $10^{-5}$; we train with batch size two for 60 epochs and reduce the learning rate after 30 and 50 epochs by a factor of 0.1, respectively; we use linear warmup with a factor of 0.1 for 200 iterations. In this subsection, we use RetinaNet with FPN levels three and four. We use a maximum number of 50 detections per image. We use vertical and horizontal mirroring during training. The class-agnostic NMS threshold is set to 0.5 and class-specific NMS is switched off during training. For *AAD*, we use anchors of three subscales and with aspect ratios (0.5, 1.0, 2.0). For *OD*, we only use two anchor subscales; the aspect ratios are set to (0.5, 1.0) and the orientations to $(-\frac{\pi}{3}, 0, \frac{\pi}{3})$. Hence, with nine and twelve anchors for *AAD* and *OD*, respectively, the total number of anchors is comparable.

**Quantitative comparison of *AAD* and *OD*.** Table 8.6 shows a comparison of RetinaNet setup for *AAD* and *OD* and with a SqueezeNet or ResNet 50 backbone, respectively. For SqueezeNet models, we load only weights of the backbone that have been pretrained for classification on *OpenImages*. For the ResNet 50 models, we use *COCO*-pretrained weights of RetinaNet. In this case, also the convolution weights of RetinaNet's class- and bounding-box-prediction heads as well as the FPN convolutions are initialized. Since the pretrained model was trained for *AAD* on *COCO*, the *AAD* models might have a small advantage at the beginning of the training. For each model, we run three different experiments to evaluate how reliable the results and differences between the models are. We can see that differences are present, but they are smaller than the differences between different backbones or the choice of model type. For all models, we use the optimal NMS thresholds that we compute on the validation set: 0.1 for both class-agnostic and class-specific thresholds in case of *OD* and 0.2 and 0.1 in case of *AAD*, respectively (see next paragraph). We use a global minimum score of 0.05 and no class-specific score thresholds.

*OD* models generally achieve higher *AP* than *AAD* models at low IoU thresholds. At very high IoU thresholds the opposite must hold because the overall *AP* is slighly lower for the SqueezeNet backbone and three to four percentage points lower for the *COCO*-pretrained ResNet-50 model. *OD* models mostly predict fewer FPs and thus obtain higher $AP^{\star}$.

However, the comparison of *AAD* and *OD* models with *AP* or $AP^{\star}$ values based on the exact IoU might be misleading: In Table 8.6, we compare AABB measures with OBB measures. A slightly wrong prediction of the box orientation can lead to a relatively large impact on the IoU of prediction and GT box. At the same time, the predicted box can still be a good fit for the underlying instance mask or at least a fit that is comparable to the *AAD* result or the OBB GT. For this reason, we also compute the *AP* and $AP^{\star}$ measures based on $\text{rIoU}_{\text{BB}}$ as shown in Table 8.7. In comparison to the measures based

| Model | Exp | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *AP* | *AP*$^\star$ | *AP* | *AP*$^\star$ | *TP* | *FP* | *AP* | *AP*$^\star$ | *TP* | *FP* |
| Validation set | | | | | | | | | | | |
| *AAD* SqueezeNet *OpenImages* | 1 | 86.2 | 81.6 | 97.8 | 93.4 | 611 | 76 | 97.2 | 92.7 | 607 | 80 |
| | 2 | 85.8 | 80.1 | 97.2 | 91.7 | 608 | 98 | 96.5 | 90.8 | 603 | 103 |
| | 3 | 85.7 | 80.5 | 97.2 | 92.0 | 608 | 95 | 96.6 | 91.4 | 604 | 99 |
| *AAD* R50 *COCO* | 1 | 95.5 | 94.7 | 97.8 | 97.3 | 611 | 12 | 97.8 | 97.3 | 611 | 12 |
| | 2 | 95.4 | 94.5 | 97.8 | 97.3 | 611 | 12 | 97.8 | 97.3 | 611 | 12 |
| | 3 | 95.4 | 94.5 | 97.8 | 97.2 | 611 | 13 | 97.8 | 97.2 | 611 | 13 |
| *OD* SqueezeNet *OpenImages* | 1 | 85.1 | 83.3 | 99.6 | 98.4 | 620 | 18 | 97.7 | 96.1 | 611 | 27 |
| | 2 | 85.3 | 83.6 | 99.6 | 98.4 | 620 | 17 | 97.6 | 96.1 | 611 | 26 |
| | 3 | 85.2 | 83.4 | 99.8 | 98.7 | 621 | 16 | 95.4 | 93.8 | 603 | 34 |
| *OD* R50 *COCO* | 1 | 91.4 | 90.7 | 100.0 | 99.8 | 622 | 3 | 99.4 | 99.2 | 619 | 6 |
| | 2 | 91.5 | 90.8 | 100.0 | 99.7 | 622 | 4 | 99.6 | 99.3 | 620 | 6 |
| | 3 | 91.4 | 90.7 | 100.0 | 99.8 | 622 | 3 | 99.3 | 99.1 | 619 | 6 |
| Test set | | | | | | | | | | | |
| *AAD* SqueezeNet *OpenImages* | 1 | 83.9 | 78.6 | 95.3 | 90.3 | 578 | 82 | 94.4 | 89.1 | 573 | 87 |
| | 2 | 83.9 | 78.3 | 95.3 | 89.9 | 577 | 89 | 94.5 | 88.9 | 573 | 93 |
| | 3 | 83.5 | 77.7 | 95.5 | 89.7 | 579 | 96 | 93.4 | 87.4 | 570 | 105 |
| *AAD* R50 *COCO* | 1 | 92.9 | 92.0 | 95.3 | 94.7 | 577 | 11 | 95.3 | 94.7 | 577 | 11 |
| | 2 | 93.0 | 92.0 | 95.3 | 94.6 | 577 | 13 | 95.3 | 94.6 | 577 | 13 |
| | 3 | 93.0 | 92.2 | 95.3 | 94.8 | 577 | 10 | 95.3 | 94.8 | 577 | 10 |
| *OD* SqueezeNet *OpenImages* | 1 | 83.6 | 81.7 | 98.0 | 96.5 | 591 | 24 | 95.4 | 93.6 | 581 | 34 |
| | 2 | 83.8 | 82.2 | 98.4 | 97.2 | 593 | 18 | 95.0 | 93.4 | 580 | 31 |
| | 3 | 84.2 | 82.4 | 98.4 | 97.1 | 593 | 19 | 95.7 | 94.0 | 582 | 30 |
| *OD* R50 *COCO* | 1 | 89.5 | 88.7 | 98.2 | 97.9 | 592 | 7 | 97.7 | 97.2 | 589 | 10 |
| | 2 | 89.5 | 88.6 | 98.2 | 97.9 | 592 | 8 | 97.7 | 97.2 | 589 | 11 |
| | 3 | 89.4 | 88.5 | 98.2 | 97.9 | 592 | 7 | 97.7 | 97.2 | 589 | 10 |

**Table 8.6: *Pill Bags AAD* and *OD* detection results.** *OD* models clearly outperform *AAD* at low IoU thresholds. For large IoU thresholds *OD* models obtain lower *AP* and *AP*$^\star$. Note that here we compare the AABB measures of *AAD* models with OBB measures of *OD* models. Generally, very high BB IoUs are more difficult to obtain in the *OD* setting due to the high influence of the orientation on the IoU.

on IoU, the overall results are slightly reduced. Moreover, now the results of *OD* models at very high rIoU$_{\text{BB}}$ thresholds are better than those of *AAD* models, which indicates that the predicted OBBs are a better fit to the underlying instance masks than the AABBs.

**Tuning NMS thresholds.** For detection methods, it is crucial to tune NMS thresholds carefully for two reasons: First, most approaches predict many duplicate predictions for each object within the image. Second, in many cases, there are objects that are easy to find and objects that are difficult to find within the same image. Moreover, there is a maximum number of detections to avoid large runtimes and memory consumption. Hence, if we do not suppress duplicates for the easy objects, it can be the case that the easier objects are found very often with high scores and the detection limit is reached before the results for more difficult objects with lower scores are returned.

Fig. 8.5 shows the influence of setting class-specific and class-agnostic NMS thresholds. Note that in our implementation the class-specific and class-agnostic NMS are done

| Model | Exp | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *AP* | *AP*$^\star$ | *AP* | *AP*$^\star$ | *TP* | *FP* | *AP* | *AP*$^\star$ | *TP* | *FP* |
| Validation set | | | | | | | | | | | |
| *AAD* | | | | | | | | | | | |
| SqueezeNet *OpenImages* | 1 | 81.4 | 77.1 | 97.8 | 93.4 | 611 | 76 | 96.8 | 92.3 | 605 | 82 |
| R50 *COCO* | 1 | 86.6 | 85.6 | 97.8 | 97.3 | 611 | 12 | 97.6 | 97.1 | 610 | 13 |
| *OD* | | | | | | | | | | | |
| SqueezeNet *OpenImages* | 1 | 82.9 | 81.2 | 99.6 | 98.4 | 620 | 18 | 96.4 | 94.8 | 606 | 32 |
| R50 *COCO* | 1 | 88.2 | 87.5 | 100.0 | 99.8 | 622 | 3 | 98.6 | 98.2 | 615 | 10 |
| Test set | | | | | | | | | | | |
| *AAD* | | | | | | | | | | | |
| SqueezeNet *OpenImages* | 1 | 79.2 | 74.2 | 95.3 | 90.3 | 578 | 82 | 94.0 | 88.4 | 571 | 89 |
| R50 *COCO* | 1 | 84.6 | 83.6 | 95.3 | 94.7 | 577 | 11 | 95.1 | 94.5 | 576 | 12 |
| *OD* | | | | | | | | | | | |
| SqueezeNet *OpenImages* | 1 | 81.8 | 79.8 | 98.0 | 96.5 | 591 | 24 | 95.2 | 93.1 | 580 | 35 |
| R50 *COCO* | 1 | 86.6 | 85.7 | 98.2 | 97.9 | 592 | 7 | 97.3 | 96.7 | 588 | 11 |

**Table 8.7: rIoU$_{\text{BB}}$ *Pill Bags* detection results.** In comparison to the IoU-results of Table 8.6, if we evaluate with rIoU$_{\text{BB}}$, the results of *OD* do not fall short of *AAD* results even at high IoU thresholds. Since we observed only minor fluctuations between different experiments, we only show the results of the first experiment here.

sequentially and in this order. Here, we compare an *OD* RetinaNet with *OpenImages*-pretrained SqueezeNet backbone against the corresponding *AAD* method. Evaluations have been carried out based on rIoU$_{\text{BB}}$, and with *AP* and *AP*$^\star$, respectively. In all cases, if no NMS is used, i.e., both NMS thresholds are set to 1.0, the *AP* and *AP*$^\star$ values drop to values between 19% and 41% for both *AAD* and *OD* models. This indicates that the recall has dropped. Without NMS, both *AP* and *AP*$^\star$ are approximately ten pp higher for *OD* than for *AAD*. A possible reason is that, on average, less anchors per object are assigned to the foreground and thus there are also fewer predictions per object during the inference. Hence, because both approaches have the same maximum number of detections, *OD* can get a higher recall.

Moreover, as the main goal of NMS thresholds is the suppression of FPs, *AP* is not a useful measure to select these hyperparameters because it does not take low-scoring FPs into account (cf. Chapter 6). Instead, we use *AP*$^\star$ and see a much more differentiated image in columns two and three of Fig. 8.5 than for *AP* in the first column. When most duplicates are filtered out due to a low NMS threshold, the *AP*$^\star$ value gets close to its maximum. For *OD*, the NMS parameters are easier to tune in the sense that they better refer to the overlap of objects that we actually see within the images: The pills are often touching, but overlap rarely. Hence, the NMS thresholds should be low. Moreover, if we increase or decrease each of the class-agnostic or class-specific thresholds by 0.1, the effect on *AP*$^\star$ is not as high as it is for *AAD*. For example, for ADD, *AP*$^\star$ on the validation set drops from 77% to 68% if the class-agnostic IoU threshold is lowered from 0.2 to 0.1. The respective changes for *OD* are less pronounced.

For both *OD* and *AAD* methods, the optimal NMS thresholds generalize well to

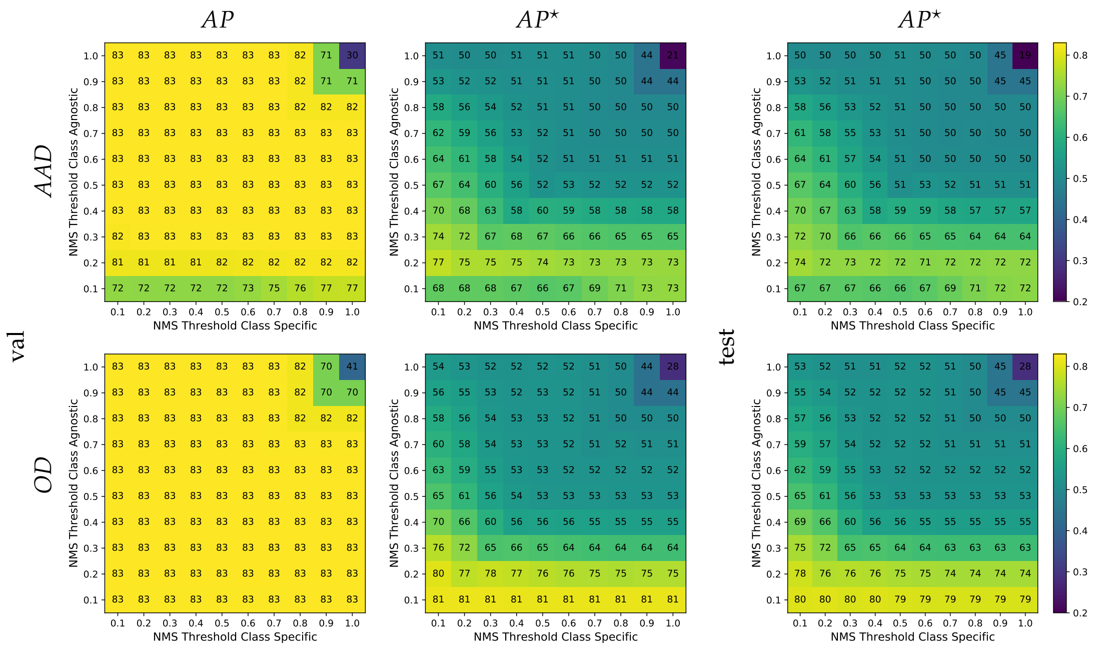


**Figure 8.5: Influence of NMS thresholds on *Pill Bags*.** *AP* and *AP*$^{\star}$ results based on rIoU$_{\mathrm{BB}}$ for different class-specific and class-agnostic NMS thresholds. Both models are RetinaNet with SqueezeNet backbone pretrained on *OpenImages*. The use of *AP* is not suitable to choose NMS thresholds as FPs are not thoroughly taken into account (*1st column*). For *OD*, the best result is obtained with both thresholds set to 0.1; for *AAD*, the class-agnostic threshold should be set to 0.2, while the class-specific threshold should be 0.1 (*2nd column*). For both models, the found thresholds generalize well to the test set (*3rd column*) (*best viewed digitally and with zoom*).

the test set. The main reason for this might be the low variance of images and object alignment within *Pill Bags* — most images look more or less the same. Moreover, the comparison of the second and third column in Fig. 8.5 reveals that for *OD*, the obtained *AP*$^{\star}$ values on the validation set are almost matched on the test set. This is not the case for the *AAD* model.

**Qualitative results.** At first sight, the results on *Pill Bags* in Fig. 8.6 look very good for both *AAD* and *OD* models. The boxes are very accurate and the class prediction is mostly correct. A closer look reveals that some pills are not found in the *AAD* case if they are touching and diagonally aligned. Both models have problems with pills that overlap, such as in the third or fourth row. These cases are rarely found within the training set.

For humans, *OD* results are visually more pleasant because it is easier to assign the box predictions to the underlying pills: The boxes are tighter and usually only their corners overlap. Moreover, if a robot was used to grasp the objects, the orientation could be used to align the gripper with the pills which could lead to a higher success rate.

### 8.4.3 D2S

Of course, we also evaluate if *OD* is helpful on a larger dataset, such as *D2S*. For this we have used the instance masks to generate the OBB GT as smallest enclosing OBB.

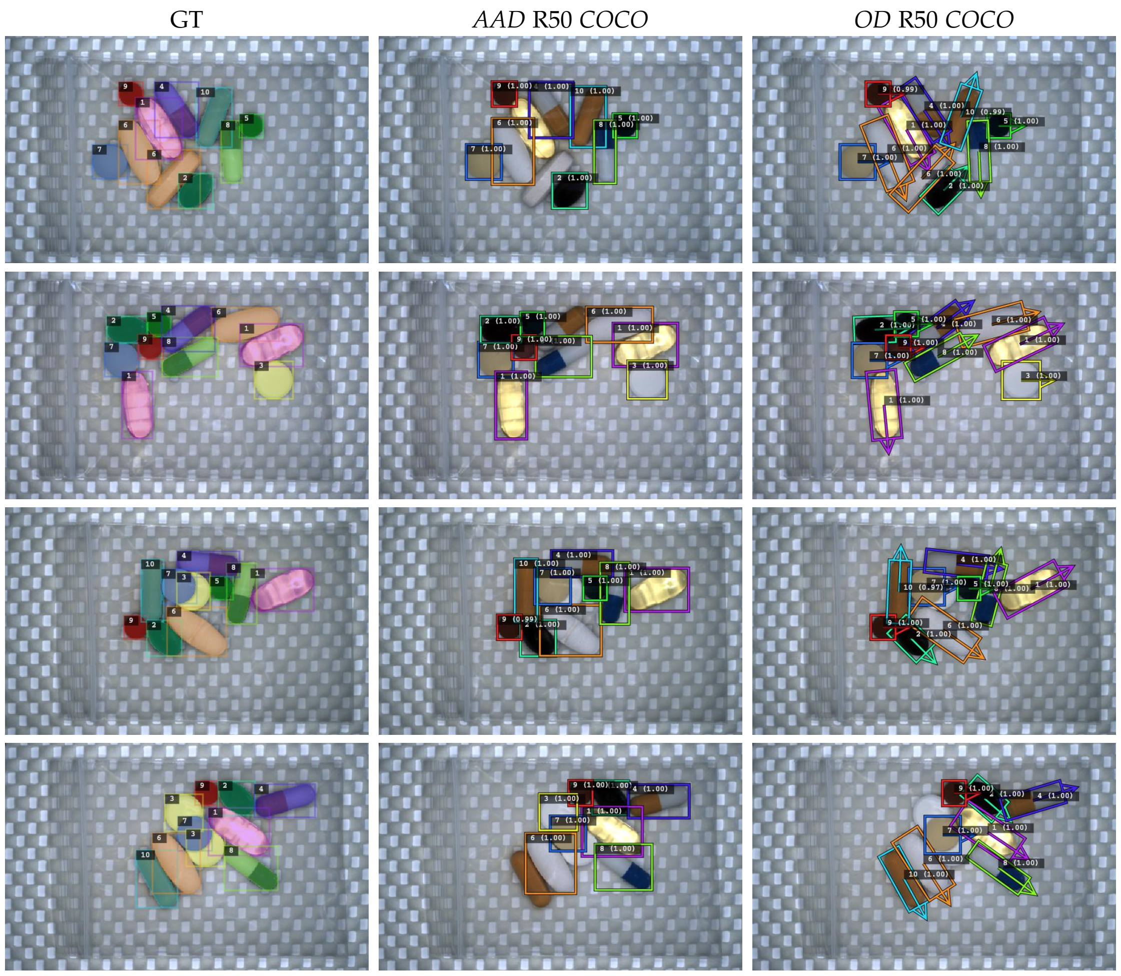


**Figure 8.6: Qualitative results on *Pill Bags* test.** *OD* boxes are tighter and fewer TPs are mistakenly filtered out by too strict NMS thresholds. Overlapping pills are challenging for both model types.

For our experiments, we use the same training and model hyperparameters as those described for object detection in Section 5.6. For *OD*, we use anchor boxes with aspect ratios $(0.4, 0.9)$ and three orientations $(-\pi/3, 0, \pi/3)$. We set *IgnoreDirection* to *true* and use no classes without orientation. We use images of size $512 \times 384$ in this subsection.

**Backbone and model choice.** Table 8.8 shows a comparison of different *AAD* and *OD* models and backbones: Averaged over all IoU thresholds, the RetinaNet and FRCNN *AAD* models achieve 15 pp and 13 pp higher *AP* values, respectively. $AP^{\star}$ values are approximately 11–12 pp higher. However, at IoU 0.5, *OD* results are almost on the level of *AAD* and *OD* FRCNN R101 achieves the highest $AP^{\star}_{50}$. *OD* FRCNN has fewer FPs than *AAD* FRCNN at IoU 0.5, but this does not hold for RetinaNet. The reason could be that the *OD* model has twice the number of anchors and because of the low minimum confidence there are more FPs. In contrast, FRCNN can use its second stage classification to filter out wrong proposals and with the used softmax low scores are less likely as we have already seen in Chapter 6. For the higher IoU threshold 0.8, the *AAD* models are clearly better than *OD* models, which means that *AAD* models predict results that have higher IoU to the GT boxes. One major reason for this behavior is that several object categories of *D2S* have no clear orientation because they are round (e.g., apples, single oranges, or clementines) or quadratic (e.g., corny boxes). Hence, the predicted orientation might differ from the GT, which can lead to a low IoU. Therefore, we evaluate $\mathrm{rIoU}_{\mathrm{BB}}$ below.

In the middle part of Table 8.8, we use $T_s^\star(val)$ to filter out FPs. This mainly improves the $AP^\star$ results of the RetinaNet models but does not affect the overall ranking significantly. *OD* FRCNN R101 is still the best model at IoU 0.5, but at higher IoU thresholds, for *OD* models, the ambiguous orientation for round or quadratic objects leads to an advantage for the *AAD* models.

The evaluation with $\mathrm{rIoU}_{\mathrm{BB}}$, shown in the bottom part of the table, closes the gap between *AAD* and *OD* models further. Since the ambiguity of the GT orientation for symmetric objects is now taken into account, $AP$ and $AP^\star$ of *OD* FRCNN R101 are now almost on the same level as for the *AAD* models. Still, overall the *OD* models lag behind, in particular for the RetinaNet architecture.

**Anchor assignment parameters.** In *D2S*, the majority of objects is not elongated and thus an AABB in most cases leads to a good fit. Surely, this also holds for the anchor boxes, where for most GT objects at least one anchor gets assigned to the foreground since its IoU is above *FgNegThresh*. However, we want to avoid an extensive hyperparameter search for every new dataset that we use. Hence, we investigate whether using *SetWeakBoxesToBg* set to *false* does any harm if uncommon box dimensions are rare as in *D2S*.

For both FRCNN and RetinaNet with ResNet 101 backbone, the results in Table 8.9 confirm that *SetWeakBoxesToBg* has no significant influence on the results on *D2S*. We also try to lower *FgPosThresh* to 0.465, such that the average number of foreground anchors per GT box on the training set approximately equals this statistic for the *AAD* model. However, for FRCNN this has almost no influence, while for RetinaNet this only leads to a slight improvement of $AP^\star_{50}$ but worsens the result at the higher IoU threshold 0.8.

**Per-class comparison.** The overall $AP^\star$ value is just an average of per class $AP^\star$ values. To get further insights, we compare the per-class $AP^\star$ values based on $\mathrm{rIoU}_{\mathrm{BB}}$ in Fig. 8.7. The plots show that *OD* is better especially for elongated objects such as bottle categories, *banana_bundle*, and *zucchini*. But also the results for *oranges* is much better than for *AAD*. In some cases, *OD* results are on the same level or slightly better than *AAD* for the lower IoU threshold 0.5, but the opposite holds for the higher IoU threshold 0.8. Hence, the *OD* results are not as accurate as *AAD* results.

Note that still the $AP^\star$ values are highly influenced by a correct class prediction. Therefore, we also run an evaluation where for all predictions that have at least IoU 0.5 with a GT box, we set the class prediction to the class of the GT box. Hence, in this evaluation, all FPs only arise from a bad localization, duplicates, or predictions within the background. The results are shown in Fig. 8.8. The comparison with Fig. 8.7 reveals that indeed some of the bottle categories, *banana_single*, and *oranges* are better found with

| Model | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $AP$ | $AP^\star$ | $AP$ | $AP^\star$ | $TP$ | $FP$ | $AP$ | $AP^\star$ | $TP$ | $FP$ |
| IoU without $T_s^\star$ | | | | | | | | | | |
| *AAD* FRCNN R50 | 78.0 | 71.9 | 86.3 | 80.6 | 44197 | 10557 | 81.2 | 75.0 | 42221 | 12533 |
| *AAD* FRCNN R101 | 79.0 | 72.8 | 86.8 | 81.0 | 44426 | 12134 | 81.9 | 75.5 | 42544 | 14016 |
| *AAD* RetinaNet R50 | 76.8 | 51.7 | 86.3 | 58.7 | 44745 | 109185 | 80.2 | 53.9 | 41728 | 112202 |
| *AAD* RetinaNet R101 | 78.3 | 53.2 | 87.2 | 59.9 | 45107 | 104633 | 81.6 | 55.5 | 42317 | 107423 |
| *OD* FRCNN R50 | 65.3 | 60.2 | 84.4 | 79.5 | 43331 | 9913 | 66.6 | 60.3 | 36685 | 16559 |
| *OD* FRCNN R101 | 67.3 | 62.0 | 86.4 | 81.3 | 44178 | 10237 | 68.6 | 62.0 | 37588 | 16827 |
| *OD* RetinaNet R50 | 61.6 | 39.2 | 85.7 | 55.4 | 44541 | 179012 | 59.3 | 36.9 | 33368 | 190185 |
| *OD* RetinaNet R101 | 63.3 | 40.9 | 86.3 | 56.8 | 44913 | 150411 | 62.0 | 39.1 | 34688 | 160636 |
| IoU with $T_s^\star(val)$ | | | | | | | | | | |
| *AAD* FRCNN R50 | 76.4 | 73.1 | 84.3 | 81.6 | 43122 | 4781 | 79.6 | 76.3 | 41345 | 6558 |
| *AAD* FRCNN R101 | 76.1 | 73.4 | 83.3 | 81.2 | 42556 | 3967 | 78.9 | 76.1 | 40965 | 5558 |
| *AAD* RetinaNet R50 | 74.4 | 70.0 | 82.9 | 79.4 | 42535 | 6086 | 77.9 | 73.4 | 40300 | 8321 |
| *AAD* RetinaNet R101 | 76.0 | 72.1 | 84.1 | 81.0 | 43085 | 5567 | 79.5 | 75.4 | 41065 | 7587 |
| *OD* FRCNN R50 | 64.2 | 61.2 | 82.6 | 80.2 | 42322 | 4651 | 65.8 | 61.8 | 36140 | 10833 |
| *OD* FRCNN R101 | 66.6 | 63.6 | 85.1 | 82.8 | 43507 | 4552 | 67.9 | 64.0 | 37216 | 10843 |
| *OD* RetinaNet R50 | 60.1 | 56.1 | 82.8 | 79.4 | 42660 | 6612 | 58.5 | 53.0 | 32853 | 16419 |
| *OD* RetinaNet R101 | 61.4 | 57.8 | 82.9 | 80.1 | 42671 | 5573 | 60.8 | 55.7 | 33858 | 14386 |
| rIoU$_{\mathrm{BB}}$ with $T_s^\star(val)$ | | | | | | | | | | |
| *AAD* FRCNN R50 | 74.5 | 71.3 | 84.3 | 81.6 | 43110 | 4793 | 79.4 | 75.9 | 41218 | 6685 |
| *AAD* FRCNN R101 | 74.1 | 71.5 | 83.3 | 81.2 | 42549 | 3974 | 78.7 | 75.9 | 40867 | 5656 |
| *AAD* RetinaNet R50 | 72.4 | 68.2 | 82.9 | 79.4 | 42526 | 6041 | 77.6 | 73.2 | 40196 | 8371 |
| *AAD* RetinaNet R101 | 73.7 | 69.9 | 84.1 | 81.0 | 43077 | 5575 | 79.3 | 75.1 | 40959 | 7693 |
| *OD* FRCNN R50 | 68.4 | 65.6 | 82.6 | 80.3 | 42354 | 4608 | 74.9 | 71.5 | 39486 | 7476 |
| *OD* FRCNN R101 | 70.9 | 68.1 | 85.2 | 82.9 | 43539 | 4510 | 77.4 | 74.1 | 40736 | 7313 |
| *OD* RetinaNet R50 | 64.1 | 60.1 | 82.9 | 79.4 | 42712 | 6663 | 69.5 | 64.3 | 36974 | 12401 |
| *OD* RetinaNet R101 | 65.2 | 61.8 | 83.0 | 80.1 | 42740 | 5710 | 71.3 | 66.8 | 37913 | 10537 |

**Table 8.8: Model and backbone comparison on *D2S* test.** On average over all IoU thresholds, *AAD* models outperform *OD* models. The difference is smaller for rIoU$_{\mathrm{BB}}$. *OD* FRCNN R101 achieves the best $AP^\star$ at IoU threshold 0.5. At higher IoU thresholds, e.g., 0.8, *AAD* has better results, which indicates that the predicted OBBs of *OD* models are less accurate.

| Model | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $AP$ | $AP^\star$ | $AP$ | $AP^\star$ | $TP$ | $FP$ | $AP$ | $AP^\star$ | $TP$ | $FP$ |
| FRCNN R101 SWB | 66.2 | 63.4 | 85.0 | 82.8 | 43268 | 4027 | 67.4 | 63.5 | 36917 | 10378 |
| RetinaNet R101 SWB | 61.5 | 57.5 | 83.2 | 79.9 | 42725 | 6070 | 60.6 | 55.0 | 33626 | 15169 |
| FRCNN R101 FgPos | 66.6 | 63.6 | 85.3 | 82.8 | 43518 | 4507 | 67.9 | 63.9 | 37241 | 10784 |
| RetinaNet R101 FgPos | 61.5 | 57.6 | 83.6 | 80.4 | 42944 | 6071 | 60.7 | 55.2 | 33803 | 15212 |
| FRCNN R101 | 66.6 | 63.6 | 85.1 | 82.8 | 43507 | 4552 | 67.9 | 64.0 | 37216 | 10843 |
| RetinaNet R101 | 61.4 | 57.8 | 82.9 | 80.1 | 42671 | 5573 | 60.8 | 55.7 | 33858 | 14386 |

**Table 8.9: Influence of anchor assignment hyperparameters on *D2S*.** All models are evaluated based on IoU and with $T_s^\star(val)$. SWB: *SetWeakBoxesToBg* $\rightarrow$ *true*; FgPos: *FgPosThresh*$\searrow$ 0.465. On *D2S*, different anchor assignment hyperparameters do not lead to significant changes.

*OD*. However, some of the very elongated objects, such as *cucumber* or *roma_vine_tomatoes*, where one would hope for a much better fit by *OD*, are not found very well in practice. For these long objects, the orientation and also the box coordinates seem to be more difficult to predict very accurately.

The comparison of Fig. 8.7 and Fig. 8.8 also shows that for many categories, such as apples, bottles, or tea-boxes, the fine-grained classification is the major challenge that remains. On the other hand, for categories like *banana_bundle*, *banana_single*, and *(roma_)vine_tomatoes* none of the methods reaches a good $AP^{\star}$ at higher IoU thresholds.

**Qualitative results.** Some exemplary results are shown in Fig. 8.9. Overall, as the quantitative numbers from previous paragraphs indicate, the results are approximately on the same level. Of course, in most images, the *OD* predictions have less overlap than the *AAD* results. In some cases, the orientation of the GT boxes and the predictions are differing (*2nd, 3rd, and 4th row*). This reduces the IoU drastically, but as the predictions still lead to a good bounding box for the underlying objects, $\text{rIoU}_{\text{BB}}$ should stay high. In many cases, the *OD* boxes for long objects, such as the *cucumber* in the second row or the *vine_tomatoes* in the fifth row, do not match the object mask accurately. In the latter case, this can be attributed to the plastic packaging that is hard to differentiate from the background.

Both models have difficulties with objects that reach out of the image boundary. Especially for *OD* predictions, this leads to many inaccurate results because the orientation of the GT box highly depends on how the instance mask intersects with the image boundary: Sometimes the GT orientation is parallel to the image boundary, but in other cases the orientation is parallel to the longest edge of the object within the image (*6th row, compare pasta on the bottom and grapes on the top*). It is difficult for the model to learn how the orientation actually should be.

Interestingly, the remaining failure cases are often the same for *AAD* and *OD*: Clutter objects often lead to FPs (when not filtered out using $T_s^{\star}$). Both models struggle with fine-grained classification problems, e.g., the correct class of tea box or type of apple is often not found. Overlapping objects, such as shown in the bottom two rows, cannot be solved satisfactory by either of the models.

## 8.5 Conclusion

In this chapter, we introduced oriented box detection. Compared to axis-aligned bounding boxes, in particular for elongated objects that are not aligned with the image boundaries, oriented boxes describe the objects' location much better. We showed how current models need to be adapted such that they can be used for oriented detection. Additionally, we presented the concept of *classes without orientation*, such that a single model can be used to describe objects that have a clear orientation and objects for which the orientation is ambiguous.

Moreover, with $\text{rIoU}_{\text{BB}}$ we introduced a measure to compare axis-aligned box detection with oriented box detection more fairly. In our extensive experiments, we evaluated

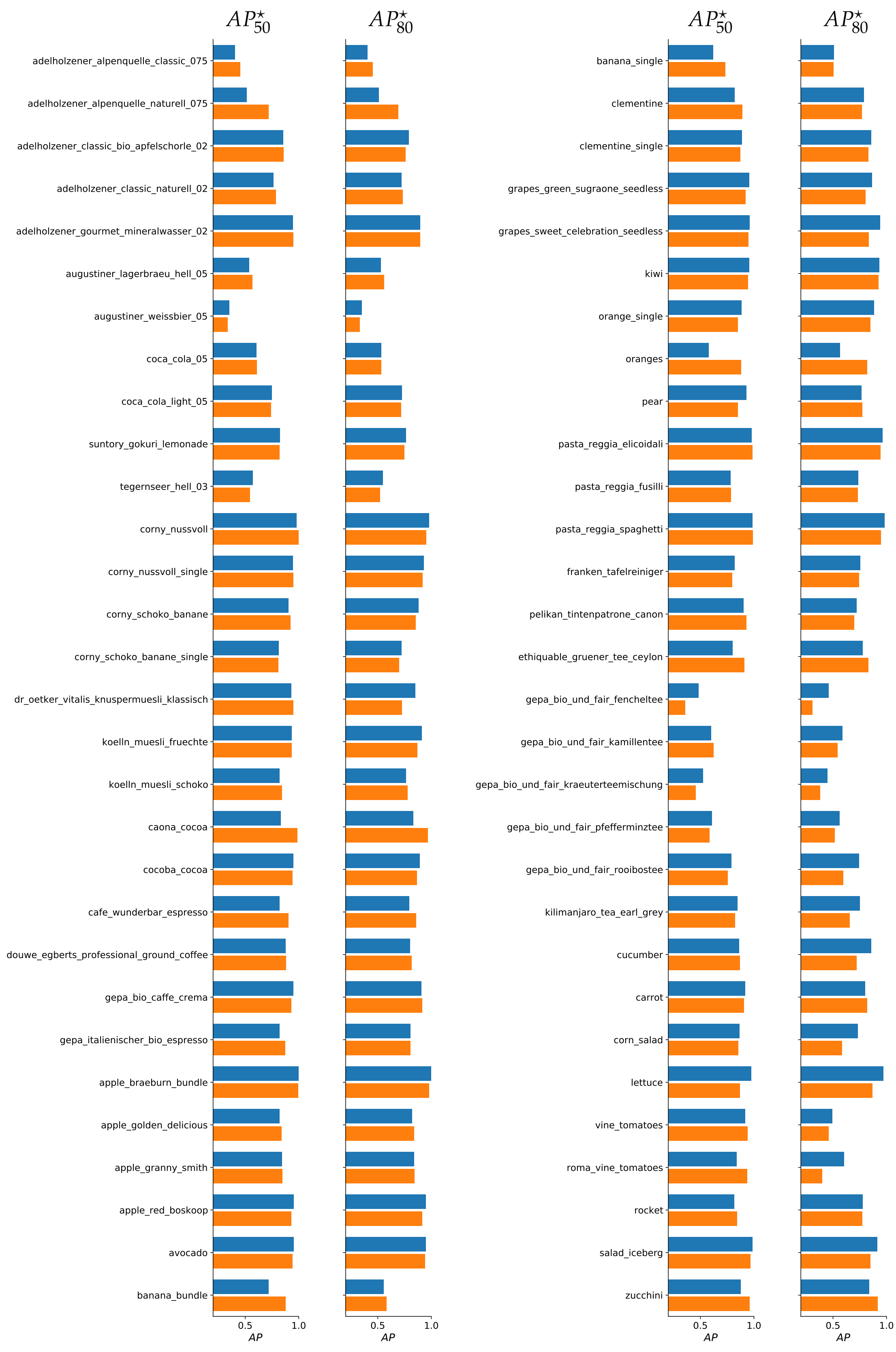


**Figure 8.7: Per class $AP^{\star}$ rIoU$_{BB}$ comparison on *D2S* test.** *AAD* (*blue*) vs. *OD* (*orange*). For some categories, e.g., bottles, *banana*, or *zucchini*, *OD* is significantly better than *AAD*. In comparison to *AAD*, *OD* results are sometimes better at IoU threshold 0.5, but worse at IoU threshold 0.8. Rarely, the opposite is true, e.g., for class *pear* or *gepa_bio_caffe_crema*.

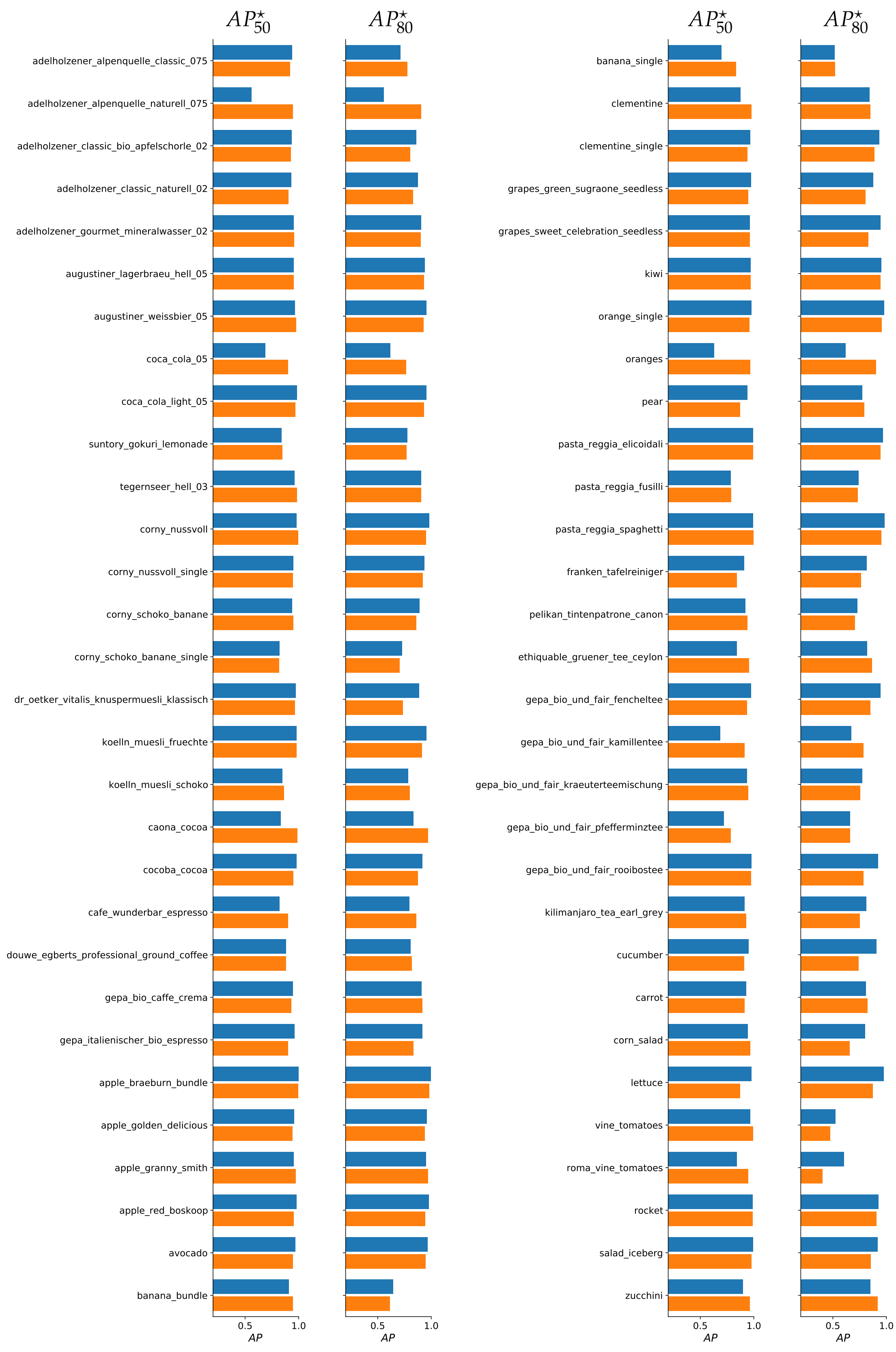


**Figure 8.8: Per class $AP^{\star}$ rIoU$_{\text{BB}}$ comparison on *D2S* test — class corrected.** For a majority of the classes, the localization accuracy of *AAD* (*blue*) and *OD* (*orange*) models is on par. For some categories, e.g., *adelholzener_alpenquelle_naturelle_075*, *coca_cola_05*, and *oranges*, *OD* clearly performs better.

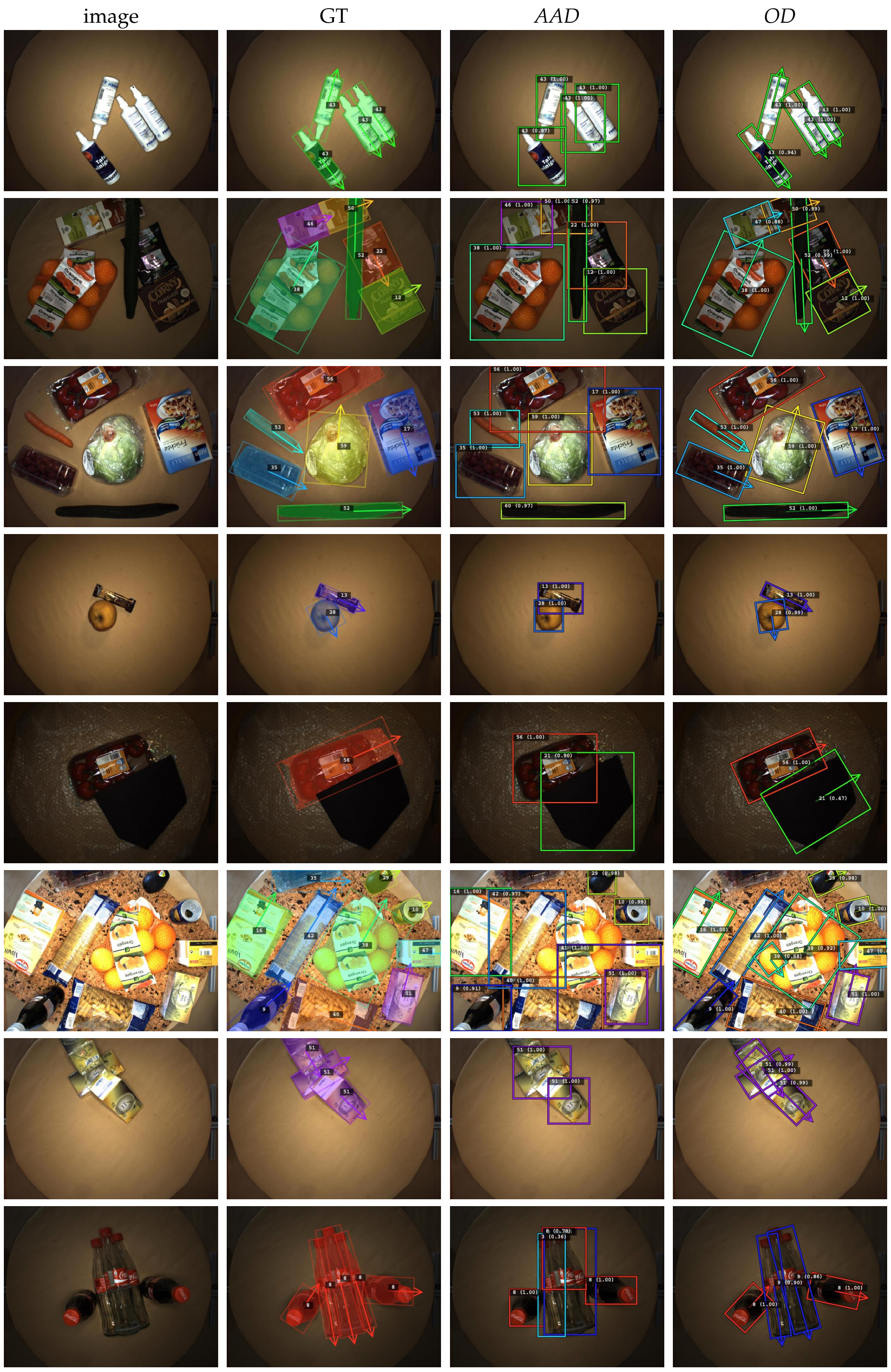


**Figure 8.9: *AAD* vs. *OD* results on *D2S* test.** Both models are without $T_s^\star(val)$, except for the results in the 6th row. *OD* certainly leads to less overlapping predictions. Generally, results are comparable; often both approaches struggle with the same images. See text for further interpretations.

our modeling choices and gave further insights when oriented detection is to be preferred.

On *Screws*, the results indicate that the orientation of the objects can be inferred with a high precision, even if the model uses only a relatively small number of orientations for the anchor boxes.

On *Pill Bags*, oriented detection leads to very promising results that are not only visually more appealing than the axis-aligned box detection results, but also leads to boxes that better fit to the underlying instance masks. For example, $AP^{\star}(\text{rIoU}_{\text{BB}})$ of the oriented box detection model is 2 pp higher than for the axis-aligned detection model.

The evaluations on *D2S* show that also for larger datasets, where many categories do not have an explicit orientation, the oriented detection approach is applicable and leads to results that are at least on the same level as the axis-aligned baseline. Further, the per-class comparison reveals that for some categories, large gains can be made and $AP^{\star}_{50}$ can be increased from 81.2% to 82.9% for an FRCNN R101 model. However, the hypothesis that oriented detection models also improve the classification accuracy must be rejected on this dataset.

# 9 A Comparison to Shape-Based Matching

Matching, that means determining the exact 2D pose (i.e., position and orientation) of objects, is still one of the key tasks in machine vision applications like robot navigation, measuring, or grasping an object. There are many classic approaches for matching, based on edges or on the pure gray values of the template. In recent years, deep learning has been utilized mainly for more difficult tasks where the objects of interest are from many different categories with high intra-class variations and classic algorithms are failing. In this chapter, we compare one of the latest deep-learning-based object detectors with classic shape-based matching. We evaluate the methods both on a matching dataset as well as an object detection dataset that contains rigid objects and is thus also suitable for shape-based matching. We show that for datasets of this type, where rigid objects appear with rigid transformations, shape-based matching still outperforms recent object detectors regarding runtime, robustness, and precision if only a single template image per object is used. On the other hand, we show that for the application of object detection, the deep-learning-based approach outperforms the classic approach if annotated data is used for training. Ultimately, the choice of the best suited approach depends on the conditions and requirements of the application. Finally, for the case where both kinds of algorithms are applicable, we show that their combination can exploit the advantages of both, leading to an improved accuracy while the runtime can be reduced.

Parts of this chapter have been published in [194]. For this thesis, the publication has been extended by the hybrid approach in 9.6.

## 9.1 Introduction

CNNs have led to a breakthrough in object classification [95], detection [58], and semantic segmentation [126, 161]. Recently, deep neural networks also became part of industrial machine vision algorithms [180, Chapter 3.15.3.4]. However, this development was driven by benchmarks on large datasets like ILSVRC [168], COCO [118], or Cityscapes [22], and by more and more powerful hardware as these models typically are trained on one or several GPUs. Moreover, the improvements are achieved on datasets with categories that are mostly deformable or exhibit large intra-class variations. The main challenge of benchmarks is often to predict the correct category, whereas, for example, in object detection tasks, the approximate axis-aligned bounding box of the object is sufficient to count a prediction as true positive.

In comparison, in industrial problems, e.g., in bin picking or measuring applications, we are interested in the exact, possibly subpixel-precise, pose of the object. Objects are often rigid and intra-class variations are small. Exact object annotations for these types of problems are time-consuming and expensive to obtain. Thus, the algorithms should be trainable from only a few example images. Due to the constraints given in a production pipeline, possibly using an embedded platform, models should be lightweight, run in real-time, be very precise, and robust to occlusions or illumination changes.

Recently, Siegfarth et al. [174] have compared a CNN-based regression and classification method to a shape-based matching (SBM) [180, Chapter 3.11.5.6] approach, which does not include a machine learning component. In their experiments, they show that of the CNN-based methods, only the regression works reasonably well, but by far worse than SBM.

In this chapter, we compare SBM against the modern deep-learning-based (DL-based) object detection method from Chapter 8 that is capable of predicting 2D rotated bounding boxes. On the one hand, we evaluate the robustness and the pose accuracy on a matching dataset. On the other hand, we compare the methods on a typical object detection problem, but where the objects and their transformations are mainly rigid. We show that a DL-based matching approach based on a single training image can locate the object of interest with subpixel precision. However, SBM still outperforms the DL-based method in terms of robustness, precision, and runtime. Regarding the detection application, we show that one can achieve very good results using SBM and that the DL-based approach only outperforms SBM in case a labeled dataset is used for training the model.

Moreover, we show that the best result can be obtained by combining both methods. Therefore, first the DL-based method is applied. The results are then used in a second step to restrict the search space of SBM and refine the box-positions obtained in the first step. Overall, this leads to a better average precision, while at the same time the runtime of SBM can be reduced significantly.

## 9.2 Methods

For our comparison of classic and DL-based methods, we use SBM [180, Chapter 3.11.5.6] as the classic representative and the one-stage detection CNN architecture RetinaNet [120] extended for oriented box detection (cf. Chapter 8) as the DL representative. Both methods are implemented in the HALCON software [142], which we use for our evaluations. We introduce SBM briefly within this section. The RetinaNet architecture is described in Section 4.3 and Chapter 8 and we only explain certain parameter choices here.

### 9.2.1 Object Detection: RetinaNet

For our purposes of comparing the algorithm against a classic approach, we choose RetinaNet for the following reasons: On the one hand, we want to have a model with a small memory footprint such that running the model is possible also on an embedded device. On the other hand, we want to achieve runtimes that are as small as possible since the longer runtime of DL algorithms compared to classic algorithms is usually one of the major drawbacks. To get a similar 2D pose output as for SBM, the bounding box prediction of RetinaNet is enhanced by predicting rotated instead of axis-aligned bounding boxes, as explained in Chapter 8.

As backbone, we use a SqueezeNet architecture [81] (referred to as the DL model in the following) and append further $3 \times 3$ convolutions with stride 2 for FPN levels higher than four. It is beyond the scope of this work to do a full benchmark of all state-of-the-art detection architectures. For DL-based object detectors, there is typically a tradeoff between accuracy on the one hand and speed and memory consumption on the other hand. With RetinaNet, we choose one representative architecture that has shown to be working well on large-scale object detection datasets such as COCO [118] and at the same time allows to be configured such that it has a low memory footprint and small runtimes. To give an impression of the influence of using a heavier architecture, we additionally run our experiments with a ResNet-50 [68] backbone (referred to as DLR in the following).

### 9.2.2 Shape-Based Matching

Robustly and efficiently finding objects in an image is one of the most important machine vision tasks. SBM [180, Chapter 3.11.5.6] is a template matching approach that uses a similarity measure that is robust to occlusions, clutter, and illumination changes [178, 179]. It can be used to recognize and locate instances of planar objects in an image. Because of its accuracy, efficiency, and robustness [191, 192, 193], it is one of the most widely used matching methods for industrial applications.

In SBM, the model of an object is defined by a set of points and associated direction vectors. The model is generated from an image of the object (model image) in which the object is specified by an arbitrary region of interest (RoI). The model points and their associated direction vectors are obtained by extracting subpixel-precise edges in the model image within the RoI and computing the gray value gradients at these points. The model is used to efficiently find instances of the object in a runtime image, where the gray value gradient is computed at each pixel. In the matching process, transformed models are compared to the image at a particular location by computing the dot product of the normalized direction vectors of the model and the runtime image at the transformed model points. Depending on the application, these transformations may cover translations, rigid transformations, similarity transformations, or even arbitrary affine transformations. By using image pyramids and applying sophisticated search heuristics, matching becomes extremely efficient. As result, SBM returns transformation parameters and a score value ($[0, 1]$) for all found instances of the object in the image.

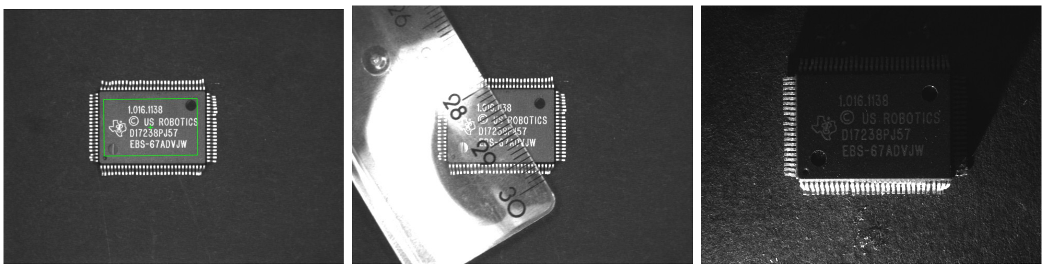


**Figure 9.1: Matching application.** Training image of the chip that is used to generate or train a model (*left*). The user specifies the object of interest by drawing a rectangular region of interest into the training image (*shown in green*). The robustness of the model is measured on images with occlusions (*middle*) or different illuminations (*right*).

## 9.3 Application 1: Matching

The first application to compare SBM against DL-based object detection is a matching scenario that is typical for industrial applications. As shown in Figure 9.1, an IC must be found within an image and its position and orientation must be determined. All images in this applications are 8-bit single-channel images of size $652 \times 494$ pixels. The user must define a RoI within a single training image (model image) and the task of the algorithm is to generate a model from this template in order to precisely and robustly find the RoI within other test images. We describe the RoI using the subpixel-precise center coordinates $(r, c)$ as well as the RoI's orientation $(p)$ in degrees. We are interested both in the accuracy of the matching result with respect to $r, c$, and $p$ as well as in the robustness of the algorithm with respect to partial occlusions of the object or illumination changes. Therefore, we evaluate both methods on four different test sets: *move_vert*, *move_rot*, *overlap*, and *illu* [192, 193]. For each of the sets, a single image without any disturbances is chosen as training image where a ground truth rectangle is drawn. For further information on the test sets, refer to Table 9.1.

For the test sets *move_vert* and *move_rot*, the IC was mounted on a table that can be shifted with an accuracy of $1\,\mu m$ and rotated with an accuracy of $0.0117\,°$. For the test set *move_vert*, the IC was shifted 50 times in $10\,\mu m$ steps in the vertical direction. For the test set *move_rot*, the IC was rotated 50 times in $0.117\,°$ steps. At each position and orientation, 10 images were acquired to be able to measure the precision of the methods in addition to the accuracy. For the test set *overlap*, the IC was occluded by various objects to different degree, while for the test set *illu*, the IC was illuminated by multiple diffuse and directed light sources of various intensities.

### 9.3.1 DL-Based Matching

Training a machine learning algorithm on a single example is very prone to overfitting. However, a comparison to the SBM approach would be unfair if more training images were used to train RetinaNet for the matching task. Therefore, we apply multiple data augmentation techniques to increase the training set without having to annotate any further images:

| category | description | # test imgs | evaluation | dl-augm |
|---|---|---|---|---|
| *move_vert* | vertical movement by 6px | 500 | accuracy and precision | c, l, n |
| *move_rot* | rotation by 6° | 510 | accuracy and precision | c, l, n, r |
| *overlap* | occlusion with various objects | 500 | robustness | c, n, o |
| *illu* | illumination changes | 200 | robustness | c, l, n |

**Table 9.1: Matching test sets.** # test imgs shows the number of images of the test set (including the training image). dl-augm shows the types of augmentation techniques that were used for the different categories: *cropping* (c), *lighting* (l), *noise* (n), *occlusion* (o), *rotation* (r).

- *lighting*: Multiply gray values by a factor between 0.1 and 6.0 in order to simulate different illuminations.
- *occlusion*: Overlay random crops from other gray value images. The cropping domains consist of the union of randomly generated rings, ellipses, and rectangles.
- *noise*: Globally add Gaussian white noise to the image to simulate sensor noise.
- *cropping*: Randomly crop the image such that the object appears at various different positions.
- *rotation*: Randomly rotate the image such that the object appears with various different orientations.

For cropping and rotation, we mirror the training image at its borders such that fewer artificial edges occur. All of the augmentation techniques can be combined depending on what is to be expected in the test images. The constellations of augmentations for different test set categories are summarized in the right column of Table 9.1.

For each of the categories, we generate 700 training and 300 validation images and train RetinaNet using stochastic gradient descent with the following optimization hyperparameters: Initial learning rate 0.0001, batch size 2, momentum 0.9, no weight prior. We train for a maximum of 70 epochs and multiply the learning rate at epoch 60 and 67 with a factor of 0.1. We evaluate the $AP$ on the validation set after every epoch and always choose the best model on the validation set for the evaluation on the test set.[1] Note that also the validation images are generated from the single training image and therefore no additional annotations are necessary.

Concerning the model hyperparameters, we reduce the number of convolutions in the box prediction and classification heads to one (default: four) and the number of filters of these convolutions to 64 (default: 256) to reduce memory footprint and runtime. We can infer the single used FPN level and the anchor aspect ratio directly from the single ground truth annotation of the IC. All other model hyperparameters are left at their default values as described in [120].

[1]We did not use $AP^\star$ to select the best model because the experiments of this chapter were done before we developed $AP^\star$. Moreover, in some experiments we tried to use $AP^\star$ instead of $AP$ as the selection criterion: although the $AP$ and $AP^\star$ values differed significantly, the same model state (best epoch) was selected by $AP^\star$ as for $AP$. This finding should be confirmed with more evaluations in future work.

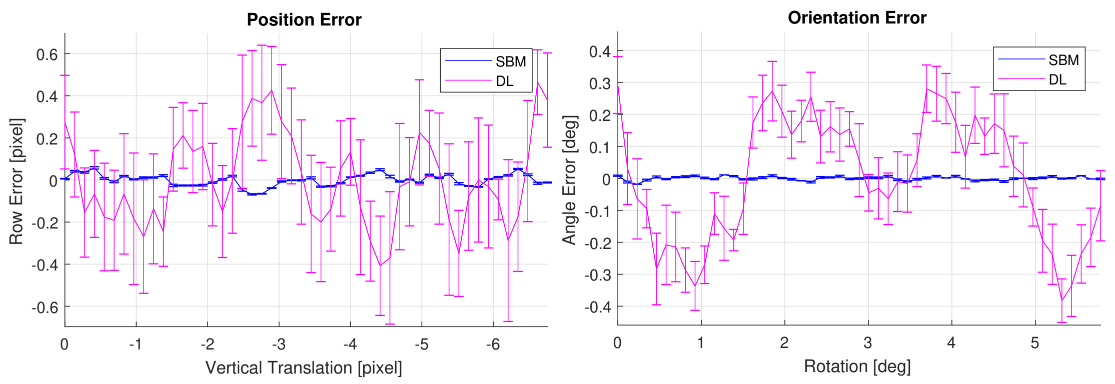


**Figure 9.2: Accuracy and precision.** Position error (*left*) and orientation error (*right*) over the test sets *move_vert* and *move_rot*, respectively. The error bars display the standard deviations over 10 images. Note that for the DL-based approach, only the results of the SqueezeNet backbone (DL) are visualized for clarity reasons.

### 9.3.2 SBM

For SBM, a model of the print on the IC was generated for each of the four datasets as described in Section 9.2.2. Because the IC does not change its orientation in the test sets *move_vert*, *overlap*, and *illu*, we restrict the transformations during the matching to translations. In contrast, rigid transformations are selected for the test set *move_rot*, while the range of searched orientations is restricted to 15 degrees. Because this application represents a typical industrial setup for which SBM is optimized, the parameters of SBM can be left at their default values.

### 9.3.3 Results

#### Accuracy and Precision

The results of the accuracy and the precision test of both methods are shown in Fig. 9.2 and Table 9.2. Although the DL-based approach (DL) achieves remarkable accuracy values, the root mean square (RMS) error of SBM is still lower by a factor of 8 for the position accuracy and by a factor of 32 for the orientation accuracy. Similarly, the precision of SBM is significantly higher compared to that of the DL-based approach: the standard deviation of SBM is lower approximately by a factor of 80 for the position and by a factor of 40 for the orientation. By using the ResNet-50 backbone (DLR), both accuracy and precision can be slightly improved, however, at the cost of an approximately five times higher runtime. Still, the results are far from those obtained by SBM. Consequently, for applications with high demands on accuracy, SBM is the first choice.

#### Implementation Effort and Computation Time

Table 9.2 also compares the working hours that were necessary to create an appropriate image processing solution for this application. Because for this application, the default parameters of SBM worked sufficiently well, it only took approximately 1 hour to create the solution with HALCON [142]. The main working time for the DL-based approach

| | | SBM | DL | DLR |
|---|---|---|---|---|
| effort for solution creation [h] | | 1 | 4 | 4 |
| position accuracy | model creation time [s] | 0.1 | ∼1000 | ∼5000 |
| (*move_vert*) | mean recognition time CPU [ms] | 1.8 | 66.4 | 338.6 |
| | mean recognition time GPU [ms] | - | 5.9 | 39.4 |
| | RMS [pixel] | 0.028 | 0.220 | 0.172 |
| | mean standard deviation [pixel] | 0.003 | 0.240 | 0.167 |
| orientation accuracy | model creation time [s] | 0.1 | ∼1000 | ∼5000 |
| (*move_rot*) | mean recognition time CPU [ms] | 2.3 | 53.9 | 340.9 |
| | mean recognition time GPU [ms] | - | 5.9 | 39.5 |
| | RMS [deg] | 0.006 | 0.191 | 0.128 |
| | mean standard deviation [deg] | 0.002 | 0.086 | 0.061 |

**Table 9.2: Result of the accuracy test of application 1.** The root mean square (RMS) error is a measure of accuracy, the mean standard deviation is a measure of precision. The model creation times refer to a CPU implementation of SBM and a GPU implementation of DL (SqueezeNet backbone) and DLR (ResNet-50 backbone).

had to be invested to write an automated script to augment the input training image with the different types of variations (*cropping, lighting, noise, occlusions, rotation*) to generate a training and validation set (∼3 h). Setting the hyperparameters of the object detection model can be mostly done automatically based on the RoI parameters within the training image. Approximately 1 h was invested to adjust solver parameters (maximum number of epochs, when to reduce the learning rate) and to optimize the augmentation script.

Because SBM does not contain a learning component, the runtime to create the model is only 0.1 s compared to 1000 s or 5000 s for the DL-based approaches. For the DL-based approaches, this includes the automatic augmentation of the training and validation images (DL: ∼1.5 min) as well as the training time (DL: ∼15 min). The average runtime for recognizing the object in an image is approximately 2 ms for SBM. For the DL-based approaches, it is 6 ms for DL and 40 ms for DLR, both on a GPU. On the CPU, the inference runtime of the DL-based approaches increases to about 60 ms for DL and 340 ms for DLR. All time measurements for the SBM were performed on an Intel Core i7-6820HQ with 2.7 GHz. For the CPU implementation of the DL-based approaches, an Intel Xeon Silver 4114 CPU with 2.2 GHz was used while the GPU implementations were run on an NVIDIA GTX 1080 Ti GPU.

### Robustness

Fig. 9.3 shows the robustness with respect to partial object occlusions and illumination changes. Here, the proportion of true and false positives are plotted for different values of the minimum score (see below). A result was interpreted as a true positive if the returned object pose was wrong by at most 8 pixels in position and 4 degrees in orientation.

For SBM, the minimum score is a parameter that determines the score a potential match must at least have to be regarded as an instance of the model in the image. It can be set by the user very intuitively as it approximately corresponds to the degree of

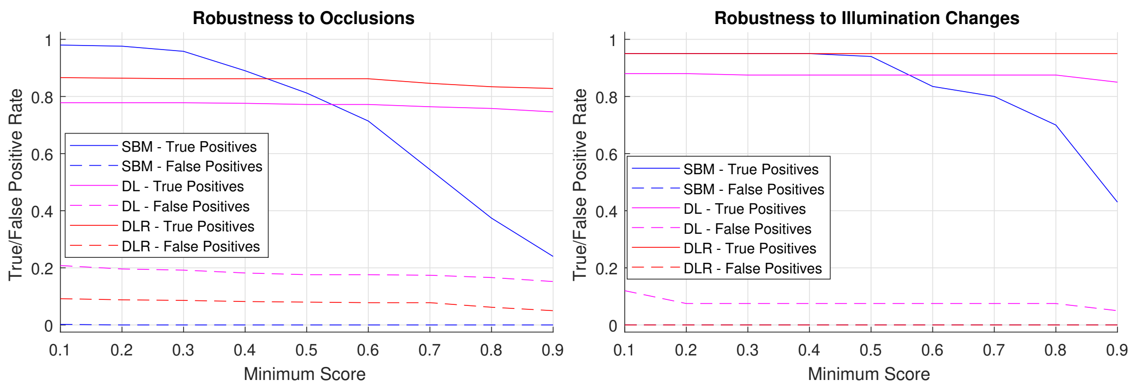


**Figure 9.3: Robustness to occlusions (*left*) and illumination changes (*right*).** The robustness is represented by the proportion of true positives and false positives. The values are plotted for different values of the minimum score. Note that SBM and DLR do not return any false positives when testing the robustness to illumination changes, and hence have identical curves.

occlusion that must be expected in the application. For example, if at most 30 % of the object is expected to be occluded, the minimum score should be set to a value of at most 0.7. The minimum score should not be chosen too small, as this increases the runtime and the risk of false positive matches. For the DL-based approaches, the minimum score is a threshold on the returned confidence of the result.

Both plots of Fig. 9.3 show that for SBM, the number of true positives decreases with increasing values of minimum score. For the occlusion case, this is immediately clear from the above definition of the minimum score. In the illumination case, parts of the object are under- or overexposed in some test images. This has a similar effect as partial occlusions, and hence also results in a decreasing number of true positives for higher minimum score values. It should be noted that SBM did not return any false positives in the two test sequences.

In contrast to SBM, the number of true positives of the DL-based approaches only marginally changes with the minimum score. For lower values of the minimum score, the DL-based results are worse compared to SBM, while for higher minimum score values, the DL-based approaches perform better. It should be noted, however, that the DL-based approaches also return a significant proportion of false positives in all cases. For illumination changes, the use of the heavier ResNet-50 backbone definitely improves the results: Independent of the minimum score, the true positive rate is the same as the maximum value of SBM and there are no false positives.

To investigate the reason for the different behaviors with respect to a changing minimum score, we plotted the score values (i.e., the confidence value for the DL-based approaches) with respect to the estimated object visibility for all images in the test set *overlap* (see Fig. 9.4). For all approaches, the minimum score was set to 0.3. The score value of false negatives is set to 0 in the plots, while false positives are not displayed. For SBM, there is a significant correlation between the object visibility and the returned score value (the correlation coefficient is 0.95). Furthermore, objects with a high visibility are recognized with a high robustness by SBM. In contrast, there is hardly any such

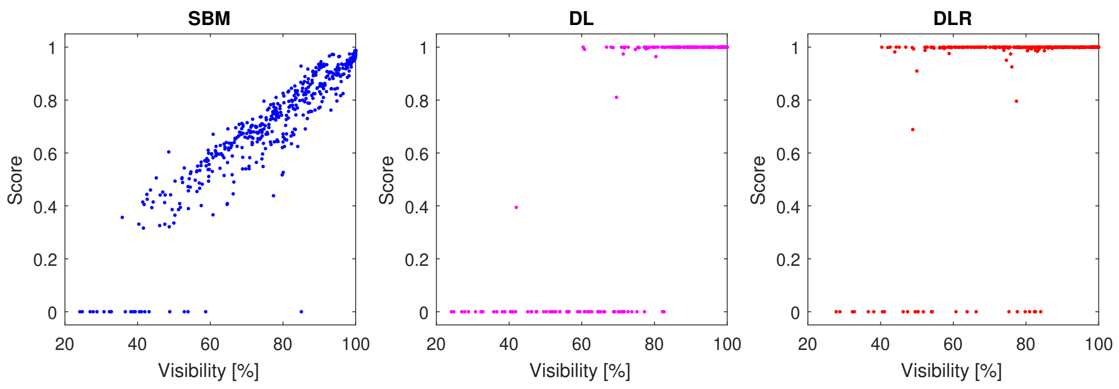


**Figure 9.4: Correlation between object visibility and score.** While for SBM a significant correlation is evident (correlation coefficient of 0.95), there is almost no correlation for the DL-based approaches (correlation coefficient of 0.41 (DL) and 0.17 (DLR)).

correlation for the DL-based approaches (the correlation coefficients are 0.41 (DL) and 0.17 (DLR)). For RetinaNet, the confidence is only taking the classification of the object into account (i.e., object vs. background) and does not explicitly model the confidence with respect to the object localization or the quality of the match. The internally used sigmoid function often results in confidence values that are either 1 or 0. Also, several objects with a high visibility could not be recognized. Compared to DL, DLR is detecting many more objects with low visibility. However, also here the score is very close to 1, independent of the visibility. In general, it is very difficult to interpret the score value that is returned by the DL-based approaches. Consequently, it is very difficult to choose a meaningful value for the minimum score. Also, while the minimum score in SBM can be used to balance the number of false positives versus the number of false negatives, this is not possible for the DL-based approaches. However, balancing this ratio is important in many practical applications. In bin picking applications, for example, the number of false positives must not exceed a certain predetermined threshold while a higher number of false negatives is less critical.

## 9.4 Application 2: Object Detection

For the second application, we use the *Screws* dataset. Exemplary images are shown in Fig. 4.6: The goal is to detect and localize different kinds of screws and nuts on a structured wooden background. All images in this applications are 8-bit three-channel RGB color images resized to 960 × 720 pixels. The dataset consists of 384 images in total. Of these, 15% are used as test set. Only to create a DL-based baseline (supervised models, Section 9.4.3), we use 70% for training and the remaining 15 % for validation.

Each object must be classified into one of the 13 possible object types and located by its rotated bounding box. The categories differ in the length and width of the screw or the diameter of the nut, in the color of the metal, and in the shape of the screw's head, tip, or thread. Objects might be touching or overlapping each other and the number of objects varies. For screws, the orientation is defined from the center to the tip. Because the orientation of a nut is not well defined due to their symmetries, the orientation is

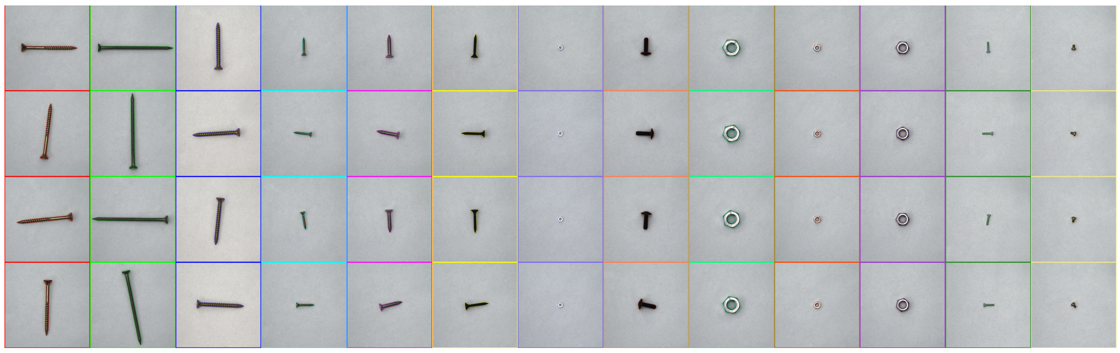


**Figure 9.5: Screw and nut template images.** Each column contains four cropped images of one object type that are used for model training or data augmentation. The borders of the automatically generated regions are shown in color (*best viewed digitally and with zoom*).

labeled parallel to the positive horizontal axis (cf. *classes without orientation*, Chapter 8). For the evaluation, we are using the *COCO AP* (cf. Section 4.2) and $AP^{\star}$ (cf. Chapter 6).

### 9.4.1 DL: Training with Weak Annotations

A drawback of DL-based systems is that they typically require a large number of annotations. Depending on the application, these can be tedious and time-consuming to obtain. As shown, e.g., in Chapter 7, one can circumvent this problem using data generation in combination with weak annotations: Instead of labeling the objects in images of the test set, each object is acquired separately on a homogeneous white background (Fig. 9.5). The object region can thus be extracted by thresholding and the class can be documented with minimal additional cost when the object is positioned for acquisition. In the experiments, for the training images five to twenty objects are sampled from thirteen object types and placed with random orientation on a random empty background similar to those in the test set. To prevent smaller objects from being fully concealed by larger regions, a maximum IoU of 0.3 is introduced. By employing this method, an arbitrarily large number of synthetic images can be generated nearly free of cost. In the following, the effect of the amount of generated training samples in the training set is evaluated (Fig. 9.6). Furthermore, we investigate whether using four template images of the same object taken from different viewing angles improves the model performance compared to the use of a single template per object type. Cropping the objects and pasting them onto the background causes sharp and visually unpleasant edges. A promising approach is to apply Gaussian smoothing to the edges of the objects to generate more natural object boundaries and to reduce artifacts.

Note that in case of using weak annotations, we do not use any images from the original training or validation set. The validation set that is used to choose the best model is generated in the same manner as the training set. Thus, we do not require any hand-labeled images to train the model. Generally, the number of synthetically generated training images is unlimited. To see how much information gain is achieved by adding more images, we do experiments using 1500, 3000, 6000, and 10 000 training images. We always use the same generated validation set consisting of 400 images.

Further experiments are carried out, in which we add the original annotated training set of 269 images to the generated training set. We also evaluate how much it changes the performance if the original annotated validation set of 55 images is merged to the generated validation set.

To cover the different scales, shapes, and lengths of the screw types, we use FPN levels three and four and aspect ratios of (0.1, 0.16, 0.35, 1.0) in this application. We also use the default value of four convolutions in the box prediction and classification heads and use the default number of filters for these convolutions (256). The anchor orientations are set to $(-2\pi/3, -\pi/3, 0, \pi/3, 2\pi/3, \pi)$ to ensure that the full range of possible orientations is covered. We set the initial learning rate to 0.0075 and multiply it by factors of 0.5 and 0.01 after 35 and 50 epochs, respectively, as the loss stagnates at these epochs. Momentum was set to 0.99 in this application.

All DL-based models are trained and evaluated using images of size $512 \times 384$ to avoid overly long training times. The models are evaluated after each epoch and the model with the best $AP$ on the validation set is chosen as the final model (early stopping).

### 9.4.2 SBM

Because in this application the objects may appear at arbitrary positions and orientations, we choose rigid transformations with a full orientation range of 360 degrees as the transformation class. SBM is intended for (approximately) planar objects. However, the distinct 3D shapes of the objects cause perspective distortions of the imaged objects depending on their position and orientation in the image, which cannot be covered by the 2D transformations of SBM. Furthermore, because of the screw threads, the shape of the screws in the image varies slightly depending on the screw's rotation around its longitudinal axis. Therefore, in this application, extensive parameter tuning was necessary to obtain satisfying results.

To apply SBM, all images are transformed into gray-scale images. We test two different cases: In the first case, we create a single model for each object type from one of the images shown in Fig. 9.5. In the second case, we create four models for each object type using each of the images in Fig. 9.5 as a model image. The second case is intended to better cover the small perspective variations of the 3D objects.

The shapes of two object types (screws in columns 5 and 6 in Fig. 9.5) are almost identical. Because the screws only differ in brightness, it is impossible to distinguish them solely based on the SBM result. To solve this problem, we apply a simple post-processing to all found instances of the two object types: We set the result class based on the mean gray value within a small central region of the found instance. This is feasible because the illumination of the scene does not change too much in this application.

### 9.4.3 Results

#### Evaluation Measurement

For the comparison of the methods on the object detection application, we evaluate them on the unseen test images and calculate *AP* and *AP*$^\star$ values for different IoU thresholds. A low IoU threshold means that the evaluation is not as strict regarding the localization, whereas for a high threshold the models prediction has to overlap almost perfectly with the ground truth box. Additionally, independent of the localization accuracy, the predicted class has to be correct if the prediction should count as a true positive.

#### Average Precision

As depicted in Fig. 9.6 and Table 9.3, SBM has the highest *AP* values when only a single template image is used to generate the model (SBM 1) and the minimum score is set to a relatively low value of 0.5 (SBM 1–0.5). SBM returns a very exact match in case the prediction is correct and, therefore, the performance is almost independent of the IoU threshold. This comes at the cost of a relatively large number of false positives as multiple screws are detected on a single screw head or predictions are made within the textured background. The *AP* measure suppresses these false positives if their score is lower than the score of most of the true positives. However, using the lower minimum score of 0.5, these false positives will also be present during the online use of the model, which is not desirable in practical applications. Therefore, in another parameter set (SBM 1–0.6), we increase the minimum score to 0.6 such that the number of false positives could be reduced from 192 to 12. At the same time, the *AP* slightly dropped from 82% to 78% and the number of true positives from 646 to 614. However, with respect to *AP*$^\star$, SBM 1–0.6 outperforms SBM 1–0.5 since it achieves a better compromise between recall and precision. This confirms the result of Chapter 6 that *AP*$^\star$ indeed can change the ranking of models.

Increasing the number of models per object type to four (SBM 4–0.6) only marginally improves the *AP* value (over SBM 1–0.6) but increases the runtime by a factor of three. The reason is that the runtime of SBM for a larger number of models increases almost linearly with the number of models to search. Also here, the slightly lower *AP*$^\star$ value compared to SBM 1–0.6 indicates that the increased recall comes together with a higher number of false positives.

Training RetinaNet in the weakly supervised setting as described in Section 9.4.1 leads to a significantly lower performance. Training on 6000 generated images from one template per object type or four templates per object type leads to an overall *AP* value of 69% and 73%, respectively. Generally, the weakly supervised results are comparable, but slightly worse than training in the supervised setting (DL SV, *AP* 75%).

In the left plot of Figure 9.6 we show that using four instead of one template per object type increases the performance. Also Gaussian smoothing to prevent sharp edges when pasting the objects consistently improves the results. Furthermore, we show that increasing the number of generated images also increases the *AP*, but the value

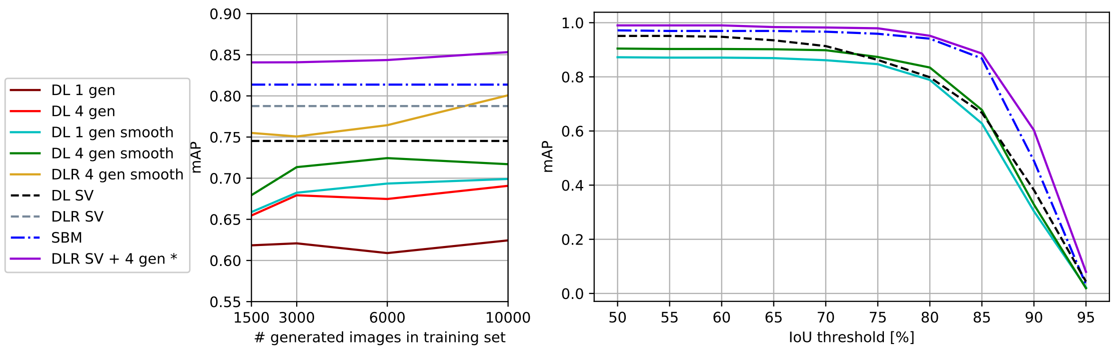


**Figure 9.6: Detection evaluation.** *AP* (averaged over all IoU thesholds) depending on the number of generated images and the number of templates per object type that were used for generation (*left*). *AP* for each IoU threshold separately for different models, training, and validation methods (*right*). For SBM, a single template per object type was used and the minimum score was set to 0.5. This corresponds to SBM 1–0.5 in Table 9.3. In the right figure, results are shown for DL and DLR (with ResNet-50 backbone) trained on 6000 and 10 000 generated images, respectively.

approximately saturates at 6000 images for the SqueezeNet backbone. With the larger ResNet-50 backbone, the use of 10 000 generated images for training helps to increase the *AP* to 80% (DLR 4 gen 10000), which is even higher than training the same model in the supervised setting (DLR SV, *AP* 79%). The overall $AP^{\star}$ value for DLR 4 gen 10000 of 77% is equal to that of SBM 4–0.6 and only 1 pp lower than for SBM 1–0.6.

The DL-based approaches only outperform SBM if annotated images from the original training set are added to the training set. This leads to an increased *AP* value of 83% (DL SV + 4 gen 6000) with a lower number of false positives compared to SBM. The reduced number of false positives also reflects in a significantly higher $AP^{\star}$ value of 82% compared to 78% for SBM 1–0.6 and 77% for DLR 4 gen 10000. Using annotated images also in the validation set does not improve the result significantly (DL SV + 4 gen 6000*). Our best result is achieved with the use of a ResNet-50 backbone (DLR SV + 4 gen 10000*, 86% *AP*). This model clearly outperforms all others with an $AP^{\star}$ of 85%. Note that the runtime of the best DL-based models using the SqueezeNet backbone is comparable to SBM on the CPU. Better models generally tend to be slightly faster as the postprocessing (including non-maximum-suppression of results) has to filter out fewer predictions.

### Implementation Effort and Computation Time

Regarding the implementation time, the main effort for the DL-based approach was to acquire the template image and some empty background images similar to those of the test images (~0.5 h). The generation of training images could be done using an automated script that only has to be written once and can be reused for different applications with different templates and backgrounds. The model training times vary substantially depending on the number of training images, the number of training iterations and the used hardware (GPU). For our setting with a NVIDIA GTX 1080 Ti GPU, it took between 15 min for the DL SV model to 7 h for DL SV + 4 gen 10000. The

| Model | all IoU | | IoU@0.75 | | | IoU@0.9 | | | Runtime [ ms] | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *AP* | *AP*⋆ | *AP* | TP | FP | *AP* | TP | FP | GPU | CPU |
| SBM models | | | | | | | | | | |
| SBM 1–0.5 | 82 | 75 | 96 | 646 | 192 | 50 | 422 | 416 | - | 110 |
| SBM 1–0.6 | 79 | 78 | 92 | 614 | 12 | 48 | 406 | 220 | - | 78 |
| SBM 4–0.6 | 81 | 77 | 95 | 633 | 65 | 52 | 431 | 267 | - | 331 |
| DL-based supervised models | | | | | | | | | | |
| DL SV | 75 | 73 | 87 | 592 | 66 | 39 | 335 | 323 | 14 | 130 |
| DLR SV | 79 | 77 | 91 | 615 | 50 | 50 | 404 | 261 | 20 | 170 |
| DL-based weakly supervised models | | | | | | | | | | |
| DL 1 gen 6000 | 69 | 66 | 85 | 578 | 109 | 32 | 297 | 390 | 12 | 122 |
| DL 4 gen 6000 | 73 | 70 | 88 | 592 | 85 | 35 | 313 | 364 | 12 | 122 |
| DLR 4 gen 10000 | 80 | 77 | 95 | 633 | 53 | 49 | 400 | 286 | 19 | 175 |
| DL-based weakly supervised + supervised models | | | | | | | | | | |
| DL SV + 4 gen 6000 | 83 | 82 | 97 | 646 | 14 | 52 | 439 | 221 | 12 | 108 |
| DL SV + 4 gen 6000* | 83 | 82 | 97 | 646 | 15 | 58 | 462 | 199 | 11 | 105 |
| DLR SV + 4 gen 10000* | 86 | 85 | 98 | 652 | 10 | 63 | 496 | 166 | 19 | 170 |

**Table 9.3: Evaluation results of application 2.** Overall *AP* and *AP*⋆ values, *AP* values, number of true positive (TP) and false positive (FP) predictions are given for IoU thresholds 0.75 and 0.9 on the test set, respectively. Overall, 660 object instances are contained in the test set. The runtime is averaged over all test samples. Note that SBM is only running on CPU. For DL, <k> gen <n> means that n images have been generated from k templates for training. Models marked with * used the annotated validation set to select the best model while training. All DL models in this table were trained on generated training data with Gaussian smoothing to reduce artifacts. DLR denotes a model with ResNet-50 backbone instead of SqueezeNet. For SBM, we applied three different setups: in two of them, we used a single model image per object type (SBM 1), while in a third setup, we used four model images per object type (SBM 4). Furthermore, we applied a minimum score of 0.5 in one setup (–0.5) and a minimum score of 0.6 (–0.6) in the two other setups.

mean recognition time is about 12 ms for the SqueezeNet backbone and 19 ms for the ResNet-50 backbone. Using the CPU, these runtimes increase to approximately 120 ms and 170 ms, respectively (cf. Table 9.3).

SBM can be optimized to a specific application by adjusting several parameters [142]. For typical applications, no manual parameter tuning is necessary. The parameters are either automatically estimated by SBM based on the model image or can be left at their default values. In this challenging application, however, a manual tuning of the parameters was indeed beneficial. Furthermore, some special cases had to be addressed and an appropriate solution had to be implemented. The overall implementation time including several optimizations was about 10 hours for a computer vision expert. The model creation took about 100 ms for each of the 13 or 52 models, respectively.

Note that for each new application that is similarly challenging as the present application, a similar implementation effort for the SBM must be expected again. In contrast, the implementation effort for the DL-based approaches is minimal once the above described template is available. The training time of the DL-based approaches, of course, must be spent for each new application again.

| | SBM | DL-based |
|---|---|---|
| Pose accuracy | high | low |
| Pose precision | high | low |
| Model creation/training | fast | slow |
| Recognition | very fast | fast (GPU) / slow (CPU) |
| GPU required? | no | yes (for reasonable runtimes) |
| Suited for embedded vision? | yes | limited |
| Memory requirement | low | high |
| Can be applied to | mostly rigid planar objects | rigid and deformable objects |
| Returns meaningful score? | yes | no |
| Used features | object shape | features are trained |
| Geometric transformations | up to projective (2D) | arbitrary |
| Runtime scales with #transformations | linearly | sublinearly |
| Runtime scales with #objects | linearly | sublinearly |
| Required user knowledge | moderate | little |
| Effort for solution development | moderate | low |
| Labeled training data | not needed | advantageous |

**Table 9.4: Summary of the comparison between SBM and a DL-based approach.**

## 9.5 Summary

Table 9.4 summarizes the results of our comparison. When it comes to selecting the appropriate algorithm in an application, our general advice is to prefer SBM over the DL-based approaches whenever the prerequisites for SBM are fulfilled. In this case, the application benefits from the advantages of SBM: high accuracy and precision, high speed, low memory and power consumption, short training time, an intuitive meaningful score value, and the possibility to easily adjust the performance (i.e., the ratio between false negatives and false positives) by choosing an appropriate score threshold.

In all other cases, the DL-based approaches are a promising alternative: applications in which a large number of different objects needs to be searched simultaneously, applications in which the objects are non-rigid or might undergo transformations in the image that are more general than a projective 2D transformation, and applications in which the object shape is not a suitable feature for recognition (e.g., to distinguish an apple from an orange, which both have a circular shape). Moreover, the DL-based approaches are to be preferred in cases where SBM will not work, for example, when detecting highly deformable or articulated objects such as animals in wildlife [45].

Another aspect that should be considered when choosing between a classic approach and a DL-based approach is the question whether a GPU can be used in the application. The use of GPUs in the machine vision industry might sometimes be difficult because the cooling system of most GPUs is not suited for dirty environments and because many GPU models on the market are available only for a short period of time, typically less than one year. Furthermore, the current trend in the machine vision industry towards embedded vision (e.g., smart devices) asks for algorithms with a high power efficiency. Although the inference of deep neural networks is possible on embedded platforms, runtimes that are acceptable in practice can often only be reached with small networks. Furthermore, GPUs are still necessary to train the networks at reasonable time.

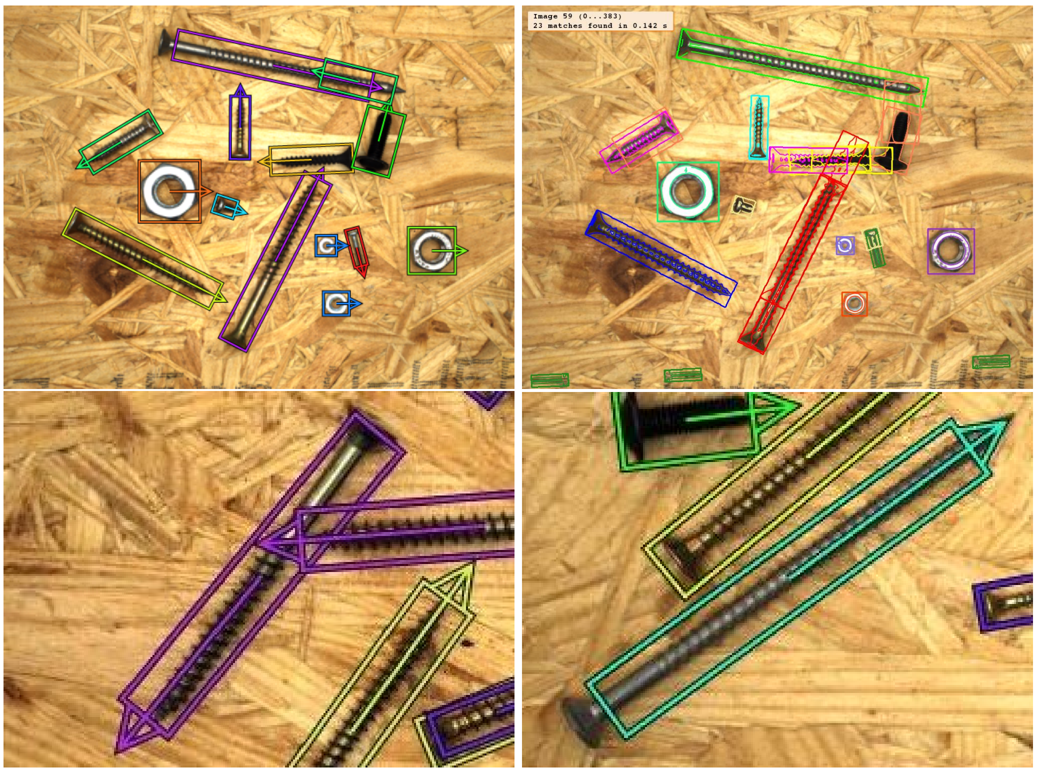


**Figure 9.7: DL and SBM result examples**. DL-based result (*top-left*) in comparison to a SBM result (*top-right*). The correctly found matches of SBM are usually very accurate. However, there are duplicate predictions with wrong orientation or textures within the background that lead to additional false positives. In case of DL, there are some wrong classifications and some inaccurate results with slightly wrong orientations or the wrong box size (*bottom row*).

## 9.6 SBM and DL-Based Detection — A Hybrid Approach

In the previous sections, we have seen that both approaches have their strengths and weaknesses (compare Table 9.3). On the one hand, for SBM the true positive predictions are usually very accurate, but the model also predicts a rather high number of false positives within the background (e.g., for model SBM 1–0.5). In particular, SBM has difficulties with the very small screws because edges or textures within the background frequently lead to false positives. On the other hand, the DL-based detection only leads to very good results when we add fully annotated data (supervised models). Often, false positives occur because the orientation is not predicted exactly along the thread of the screw or because the classification fails.

Concerning the runtime, when using a GPU, the DL-based approach has advantages over SBM. The main reason for the comparably long runtime of SBM is that all different classes have to be searched within the whole image and with all possible orientations.

However, the strengths of both approaches are complementary to each other and thus here we combine both models in a hybrid approach. In the following, we describe the combination of DL-based detection with SBM step by step as visualized in Fig. 9.8. We denote the hybrid model combining the DL model (with a SqueezeNet backbone) and SBM 1–0.5 as Hybrid, and the combination of DLR and SBM 1–0.5 as HybridR.

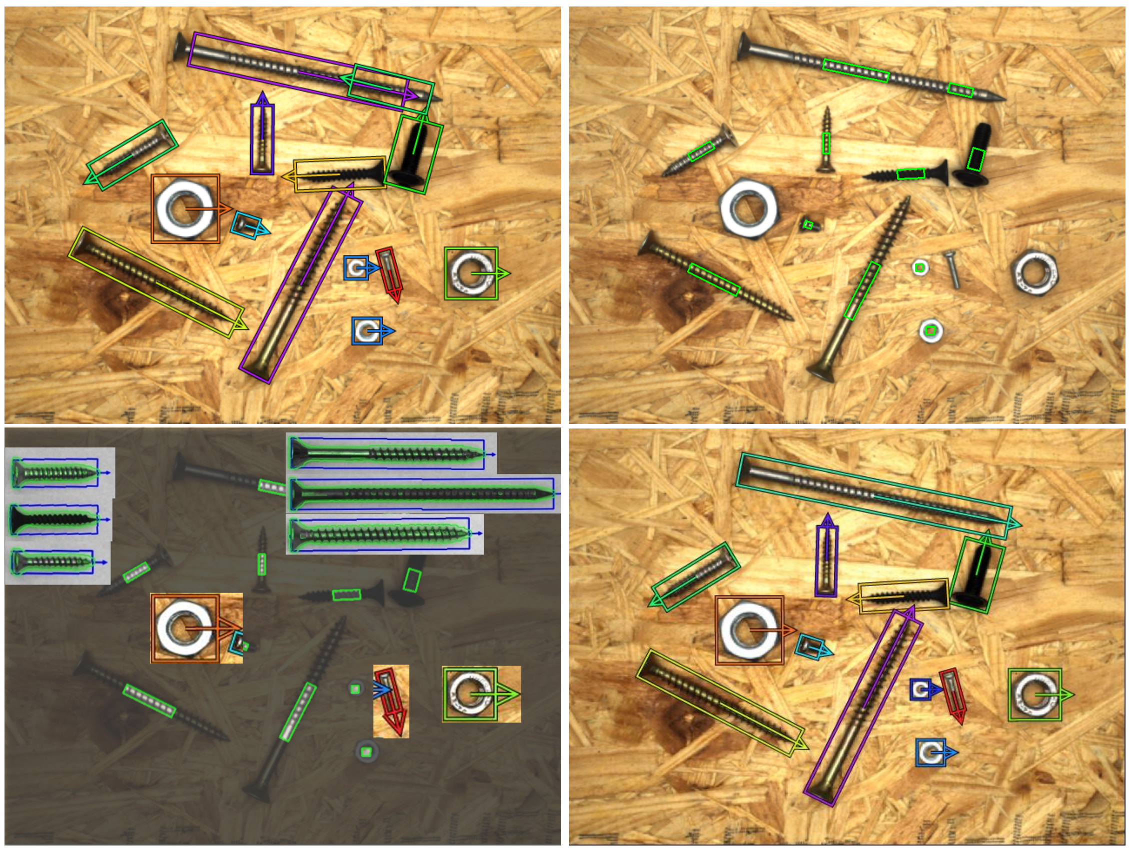


**Figure 9.8: Steps of the hybrid approach.** (*From left to right, top to bottom*) Step 1: run the DL-based detection. Step 2: restrict the search space of SBM. Step 3: Search for all similar categories and use the DL results whenever they are good. Step 4: Enjoy the final results.

### 9.6.1 Step 1: DL-Based Detection

The first step is to run the DL-based detection model. Since SBM is applied in a later step, here a high recall is of higher interest than a high precision. Inaccurate results can still be corrected in a later step.

### 9.6.2 Step 2: Restricting the Search Space of SBM

In order to avoid a large number of false positives in the background or duplicate results with wrong orientation, we restrict the search space of SBM using the DL predictions of step 1. The restricted search space for each result of the DL-based detection is given by an oriented box with 0.3 times the side lengths and with the center and orientation of the result. In order to account for inaccurate predicted orientations, the search range for the orientation is set to $\pm 7.5°$ for screw and $\pm 45°$ for nut categories. For each predicted category, we take the union of these search boxes for all predictions of this class as the final search space for the corresponding shape model.

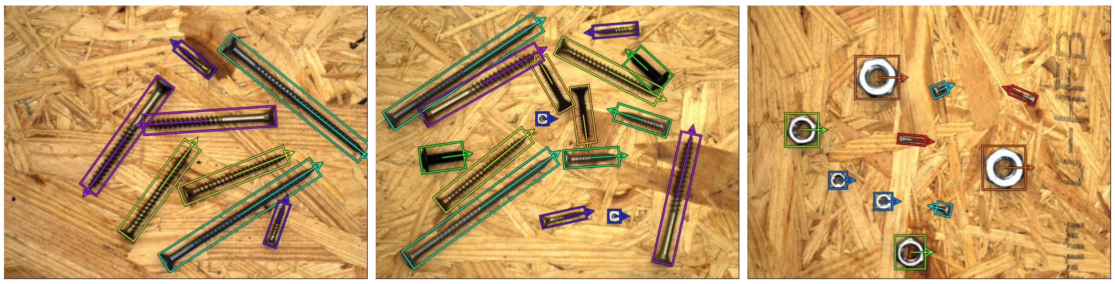


**Figure 9.9: Results of the hybrid approach**. The results contain almost no false positives, the predictions are accurate and have the correct class.

### 9.6.3 Step 3: Correct DL and SBM Mistakes

In case of the *Screws* dataset, we found that for some of the easier categories, like the larger nut classes, the DL results are already very accurate and hence we use them as the final result. Moreover, SBM struggles to find the very small screws of type 12 correctly and thus the DL predictions are also used as final result for this class.

However, for other categories, we have seen that the DL-based detection sometimes misclassifies them to another class with similar appearance. Hence, instead of just looking for the single predicted categories, we use all shape models of the categories with similar appearance. In case of *Screws*, those search clusters are given by the three largest screws, the three medium-sized screws, and the two smallest nut categories. This allows to correct classification mistakes of the DL-based detection.

### 9.6.4 Step 4: Enjoy the Results

Both quantitatively and qualitatively, the proposed hybrid approach leads to excellent results. Fig. 9.9 shows some example images. The number of false positives can be significantly reduced, predictions usually have the correct class label, and the orientation is along the thread of the screw.

Table 9.5 also confirms this quantitatively, where here also the $AP^\star$ values are shown (cf. Chapter 6). Using the combination of a weakly supervised DL-based detection model with SBM, the results are on the same level as the weakly and fully supervised models from Table 9.3. Moreover, the precision of the hybrid models is very high at IoU 0.75, which leads to very high $AP^\star_{75}$ values of 98% for HybridR in comparison to 89% for the SBM model. Also, the overall $AP^\star$ can be significantly improved to 82% for the HybridR model from 75% for SBM 1–0.5. However, the combination of SBM with a fully supervised DL-based detection model does not increase the $AP$ further and only slightly improves the $AP^\star_{90}$. There are two theoretical reasons for this: First, SBM only uses a single template image to create the shape models. The viewpoint of the template screw does not always match the viewpoint of the screws in the test images. Therefore, sometimes the predicted boxes are too short or too long. Second, it might be the case that the ground truth labels are less accurate than the predictions. This could be the case for long objects, since it is very difficult to annotate the exact center of the bounding box and the correct orientation. In those cases, the prediction might be better than the ground

| Model | all IoU | | IoU@0.75 | | | | IoU@0.9 | | | | Runtime [ ms] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *AP* | *AP*⋆ | *AP* | *AP*⋆ | TP | FP | *AP* | *AP*⋆ | TP | FP | combined |
| SBM model | | | | | | | | | | | |
| SBM 1–0.5 | 82 | 75 | 96 | 89 | 646 | 192 | 50 | 42 | 422 | 416 | 110 |
| Hybrid weakly supervised models | | | | | | | | | | | |
| Hybrid 4 gen 6000 | 81 | 80 | 96 | 96 | 639 | 3 | 49 | 46 | 409 | 233 | 23 |
| HybridR 4 gen 10000 | 83 | 82 | 97 | 98 | 646 | 3 | 52 | 49 | 429 | 220 | 32 |
| Hybrid weakly supervised + supervised models | | | | | | | | | | | |
| Hybrid SV + 4 gen 6000 | 83 | 82 | 97 | 97 | 645 | 3 | 52 | 49 | 433 | 215 | 22 |
| Hybrid SV + 4 gen 6000* | 82 | 82 | 97 | 97 | 643 | 4 | 53 | 50 | 432 | 215 | 22 |
| HybridR SV + 4 gen 10000* | 83 | 83 | 98 | 98 | 647 | 2 | 54 | 50 | 436 | 213 | 33 |

**Table 9.5: Evaluation results of hybrid models.** The evaluations are explained in Table 9.3. All hybrid models used the SBM 1–0.5 settings. Hybrid models that only use weakly annotated data are on the same level as models trained with manually labeled data. The hybrid approach significantly reduces the number of false positives.

truth, which could count as false positives for very high IoU thresholds ($> 0.85$).

Another benefit of the hybrid models is that, due to the restricted search space, the runtime of SBM can be reduced approximately by a factor of 10 such that the total runtime of 22-33 ms is realtime-capable.

## 9.7 Conclusion

In this chapter, we have compared SBM, a classic method without a machine learning component, against a modern DL-based object detector. Both methods have been evaluated on two typical industrial datasets: One for matching and another one for object detection. The results show that for rigid objects that appear with rigid transformations, SBM outperforms the DL-based object detectors in terms of runtime, accuracy, precision, and robustness.

On the other hand, we have shown that for datasets with low intra-class variations, a DL-based object detector can locate objects with subpixel precision when trained only using annotations of a single training image. The same holds for a detection application, where four templates of each object type in combination with data generation leads to very promising results. However, adding more annotated training images from the target domain improves the results even further.

With the combination of a DL-based and a classic approach in a hybrid model, the advantages of both worlds can be combined: The DL-based approach is suited for a larger variety of object types and scales better with the number of objects and with more general object transformations. The classic approach provides higher accuracy and precision and returns more meaningful score values. Moreover, in the hybrid model, the search space of the shape-based matching can be reduced such that the runtime is significantly faster. Overall, this leads to very accurate results with few FPs. The hybrid model can be trained only from augmented images in a weakly supervised manner and achieves similar results to the supervised setting.

# 10 Oriented Boxes for Few-Shot Instance Segmentation

To be useful in industrial applications, our methods should be reliable. Reliability in the context of instance segmentation means that methods should predict a low number of false positives and that the true positives should be accurate. To evaluate the latter, accurate ground truth labels are required. In Chapter 5, we presented *D2S* as a new dataset with high-quality instance mask annotations that allow to measure the instance mask accuracy properly. We saw that current approaches have a relatively high recall but a rather low precision. In the following chapters, we worked towards models that require fewer annotated training images but still achieve the same result quality.

In this chapter, we present a new method that aims to predict more accurate instance masks and at the same time fixes some of the remaining failure cases that we found in the baseline evaluations of Chapter 5. While issues like FPs within the textured backgrounds of the validation and test splits could already be addressed by changing the training data, in this part of the thesis, we change the model itself to improve the accuracy of results. At the same time, the proposed model requires fewer mask annotations which saves the user a substantial amount of tedious labeling time and costs.

State-of-the-art instance segmentation algorithms use axis-aligned bounding boxes as an intermediate processing step to infer the final instance mask output. This often leads to coarse and inaccurate mask proposals due to the following reasons: Axis-aligned boxes have a high background-to-foreground pixel-ratio, there is a strong variation of mask targets with respect to the underlying box, and neighboring instances frequently reach into the axis-aligned bounding box of the instance mask of interest.

In this chapter, we overcome these problems by proposing to use oriented boxes as the basis to infer instance masks. All current instance segmentation methods, such as the ones discussed in Section 4.4, are based on axis-aligned bounding box predictions. We show that oriented instance segmentation improves the mask predictions, especially when objects are diagonally aligned, touching, or overlapping each other. We evaluate our model on the *Screws*, *Pill Bags*, and *D2S* datasets and show that we can significantly improve the mask accuracy compared to instance segmentation using axis-aligned bound-

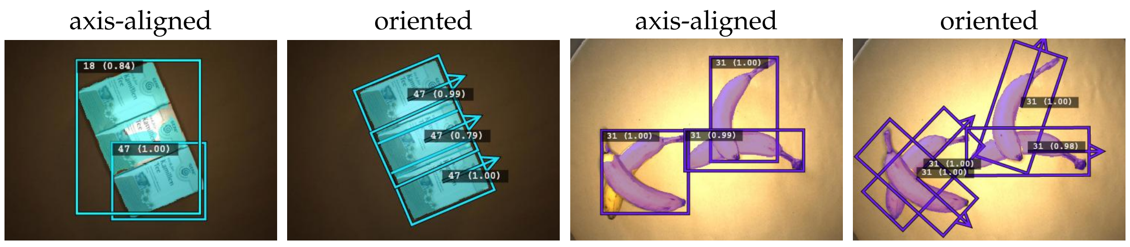


**Figure 10.1: Benefits of oriented instance segmentation.** Qualitative comparison of our proposed method based on oriented boxes and the baseline based on axis-aligned boxes. Oriented boxes contain fewer background pixels and do not reach far into neighboring instances. The resulting instance masks use the mask prediction resolution more effectively and are more accurate.

ing boxes. Further, on *Pill Bags*, we outperform the baseline using only 30% of the mask annotations and on *Screws*, we perform on the same level as the baseline using only half the amount of training images with instance mask labels.

## 10.1 Introduction

The accurate localization of objects in natural images is fundamental for many industrial tasks, such as bin-picking or object counting. The detection is often done using axis-aligned bounding boxes [120, 159]. However, if objects are deformable, articulated, or diagonally oriented, axis-aligned bounding boxes are often only a very coarse approximation of the objects' locations. Instance segmentation tries to overcome this limitation by predicting a pixel-precise mask for each of the instances. However, recent instance segmentation methods that are at the top of challenge leaderboards, such as FCIS [116] or variants of Mask R-CNN [70], rely on axis-aligned box proposals to infer the instance mask per box.

This intermediate axis-aligned box detection step introduces several limitations to the final instance mask output. Some of them are similar to the problems that we discussed in Chapter 8, but some only apply to instance segmentation methods.

Depending on an object's orientation, a majority of the box covers the background or another instance that is not of interest for the mask prediction. As features are pooled with respect to the box, this can lead to false classifications or mask predictions reaching into neighboring objects, as shown in Fig. 10.1.

If an object is rotated, the bounding box aspect ratio can vary significantly. For asymmetric objects, this leads to highly varying mask targets with respect to the bounding box, even for very accurate box predictions, which are generally not given. As a consequence, more pixel-precise annotated instances are necessary to learn these variations.

For large and diagonally oriented instances, the relatively coarse resolution often used for mask prediction can lead to inaccurate mask boundaries because of interpolation artifacts. The use of finer grids can resolve this issue to some extent, but comes at the cost of higher computation time and memory consumption.

Therefore, in this work, we propose using oriented as opposed to axis-aligned bounding boxes to infer instance masks. Oriented instance segmentation (*OIS*) solves the aforementioned issues of axis-aligned instance segmentation (*AAIS*), which are visualized by the examples in Fig. 10.1:

- The best possible IoU an oriented bounding box can achieve for an arbitrary mask is typically much higher than the IoU of the axis-aligned bounding box ([12], Chapter 8). Hence, independent of an object's orientation, most of the bounding box area overlaps with the instance of interest. This increases the mask-to-background ratio for mask targets and avoids large overlaps with neighboring objects.
- Bounding box aspect ratios become invariant to an object's rotation and the mask exhibits significantly smaller variations with respect to the pooling grid. This leads to more consistent mask targets and a better conditioned training.
- For objects with non-overlapping masks, oriented bounding boxes overlap significantly less than axis-aligned bounding boxes. This prevents false positive mask predictions within neighboring objects.
- Especially for large, elongated and diagonally aligned objects, the accuracy of mask predictions at their boundaries is increased because long edges are aligned with the oriented box.

The main contributions of this chapter are: We propose to predict accurate instance masks based on oriented box detections (cf. Chapter 8). Our approach can be easily applied to existing models based on axis-aligned boxes to enhance their performance. In particular, when trained with partial mask supervision, the models require significantly fewer pixel-precise instance mask annotations. We describe how to adapt architectures for *OIS* and explain necessary changes to different parts of the model compared to the baseline *AAIS* models. Our evaluation on the four datasets *D2S* (Chapter 5), *Screws* (Chapter 4), *Screws Gen* (Section 10.3.1), and *Pill Bags* (Chapter 4) shows that the mask accuracy is improved significantly. This leads to an increase in overall $AP^{\star}$ from 77.5% to 79.6% on *D2S* val with a Mask RCNN architecture [70] and from 79.6% to 81.5% with a RetinaMask architecture [50]. On *Screws Gen*, $AP^{\star}$ is improved from 83.5% to 86.0%. Moreover, we show that on *Screws* we obtain reasonable mask predictions when only 20 generated images with mask annotations are added to the training set. This is in line with the results on *Pill Bags*, where for *OIS* only 10% of the mask annotations are required to perform as well as the *AAIS* baseline and with 30% of the mask annotations, it can clearly be outperformed.

## 10.2 Oriented Instance Segmentation

The major change to transform an *AAIS* model to an *OIS* is to replace all axis-aligned bounding box prediction modules by oriented bounding box prediction modules, as described in detail in Chapter 8. This makes our approach applicable to all *AAIS* methods. Before we describe further key components of *OIS* models below, we give an overview

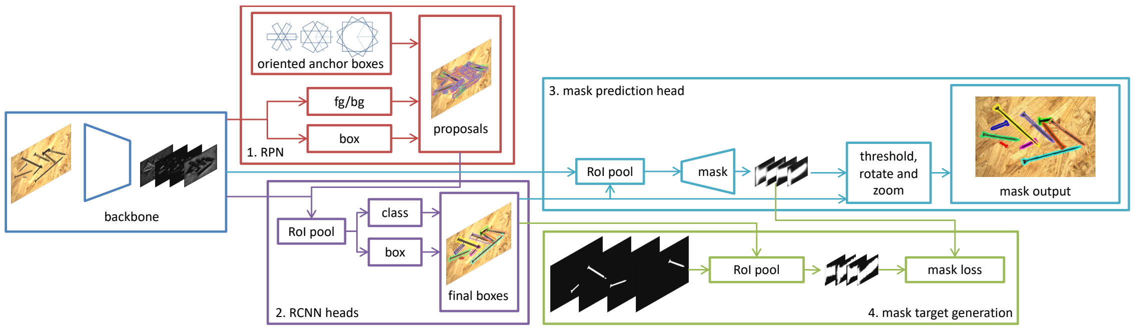
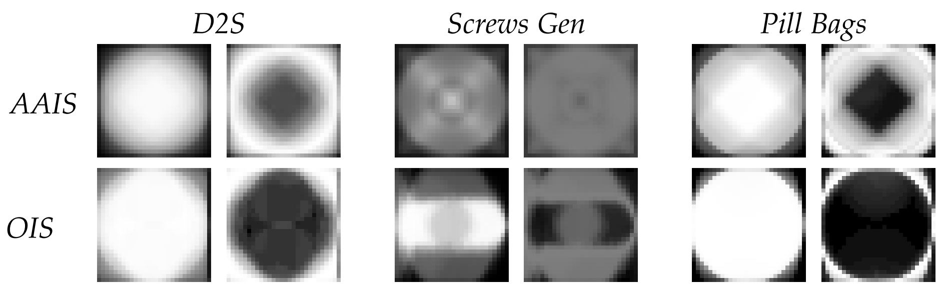


**Figure 10.2: Exemplary *OIS* architecture.** (*Blue, center-left*) Input image and backbone for feature extraction; (*red, top*) oriented RPN. (*Violet, bottom*) Features are pooled with respect to oriented proposals to feed RCNN heads for final oriented box output. (*Cyan, middle*) Features are pooled with respect to final boxes to feed the branch for mask prediction. Finally, mask probabilites are fit to the oriented box output and thresholded. (*Green, bottom-right*) During training, mask targets are calculated by RoI pooling the GT masks.

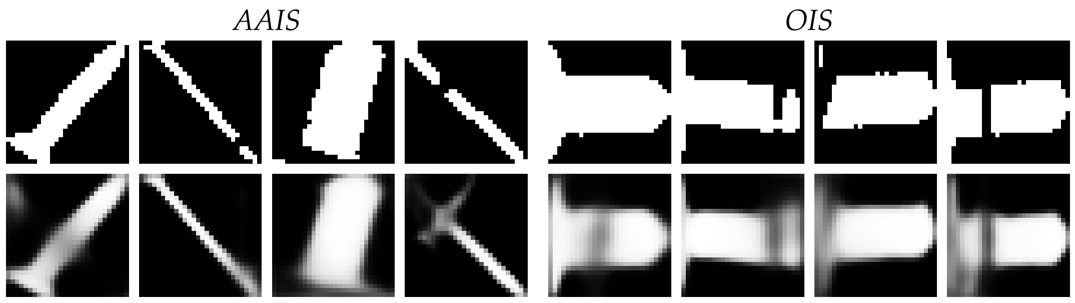
of a typical *OIS* architecture based on MRCNN [70], as depicted in Fig. 10.2: In a first step, the backbone is applied to the input image to extract features that are used for the three following stages: The first stage is the RPN, which predicts for each of a number of oriented template *anchor boxes* whether it is likely that an object with similar bounding box is present or not (fg/bg branch) and, if so, how the anchor should be refined to better match the underlying object (box branch). The second stage is the RCNN heads, where the oriented box proposal outputs of the RPN are used to RoI pool the features for class prediction (class branch) and further box refinement (box branch). The third stage uses the final RCNN head oriented box output to again RoI pool the features to feed the mask prediction head that outputs a pixel-precise mask for each of the final boxes. All RoI pooling layers pool from the oriented grids that are aligned with their input boxes. Since the output feature maps are upright, usual convolutions can be used in the subsequent layers. We use oriented RoI Align operations for bilinear interpolation during pooling.

For an *OIS* version of RetinaMask [50], the same strategy can be applied. The only difference is that the RCNN heads of the second stage are fused with the RPN stage that directly outputs class probabilities and the final boxes.

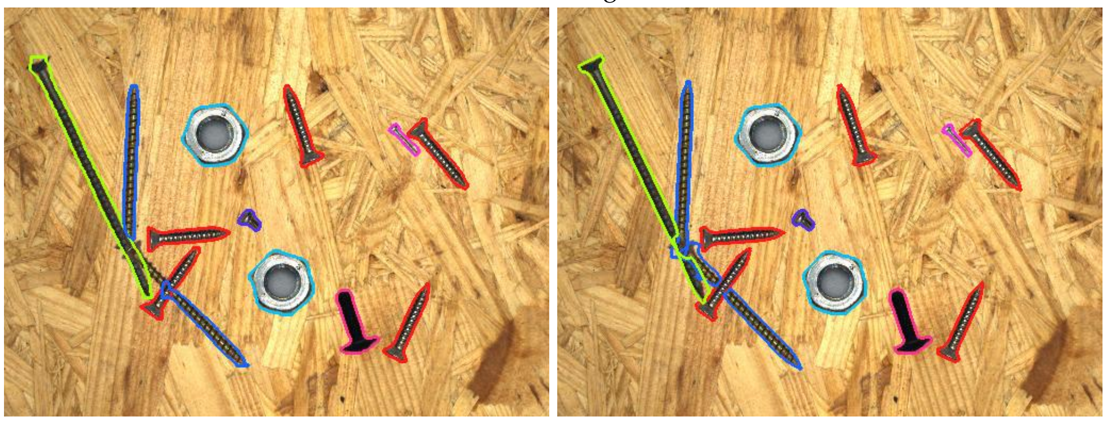
**Oriented mask prediction.** For each predicted oriented box, features are pooled with an oriented RoI pooling layer, such that the batch size of the mask branch equals the input batch size times the maximal number of predictions. The pooled feature maps with size $m \times m = 14 \times 14$ are fed into a sequence of $n$ convolutions with ReLU activation followed by an upsampling transposed convolution. Finally, a sigmoid activation is applied to get the mask probability of each pixel within the warped box. Mask prediction is done class-agnostically since it has already been shown in [70] that a class-specific mask prediction does not improve the results significantly but comes at a higher computational cost. In our experiments on *D2S*, we show that for a mask prediction head that is initialized randomly using batch-normalization (BN) layers between the ReLU and the following convolution layer is crucial for the model to converge. Those BN-layers can be omitted if *COCO* pretrained weights are used to initialize the mask head convolution weights.



**(a)** Mean (*left*) and deviation (*right*) of GT masks for *D2S*, *Screws Gen* and *Pill Bags*.



**(b)** Mask targets.



**(c)** Outputs: (*left*) *AAIS*, (*right*) *OIS*

**Figure 10.3: Mean and deviation of GT masks and comparison of mask targets and results.** (a) The mean and standard deviation of the first 3000 instance masks that are pooled to a size of 28 × 28 with respect to the GT boxes. (b) Mask targets (*top*) and mask probabilities (*bottom*) for *AAIS* (*left*) and for *OIS* (*right*), respectively. The low resolution (28 × 28) of mask targets and predictions is used much more efficiently for oriented boxes, where mask targets are more detailed. Moreover, the targets are much more consistent even for totally different instances. This simplifies both the training and the prediction. Shown are the bottom-left four screws of the image below. (c) Final mask output for *AAIS* (*left*) and *OIS* (*right*) on a *Screws Gen* image (*best viewed digitally and with zoom*).

For target generation, the GT masks are painted into a multi-channel binary image, where each channel contains the GT mask of one instance. Given the final box predictions, the assignment to the GT instances is done in the same way as in the oriented RPN based on the arIoU (cf. Chapter 8). In this separate assignment step, we increase both IoU thresholds such that only good final boxes contribute to the training of the mask branch. If a final box is not assigned to any GT instance, the corresponding weights in the mask prediction loss are set to zero such that the predicted mask is ignored. To obtain the mask probability targets for the predicted boxes during training, we use a modified oriented RoI Align operation (cf. Chapter 4) on the multi-channel GT mask image that only pools within the channel indices of the assigned GT masks (cf. bottom-right of Fig. 10.2). A sigmoid cross-entropy loss is used to train the mask branch. Therefore, the pooled mask targets are thresholded at 0.5, such that the targets are binary.

As in [70], a final grid size of $28 \times 28$ pixels for mask targets and prediction is used. This low resolution limits the detailedness of the output masks. However, since the oriented boxes generally contain much less background than the axis-aligned boxes, the given resolution is used much more efficiently. This is visualized in Fig. 10.3a and Fig. 10.3b, where for *OIS* the targets' and predictions' variance between different instances is significantly lower than for *AAIS*. On the one hand, *OIS* models can learn a stronger prior for the mask prediction since especially in the middle of the object it is very likely that the object is present. The capacity of the model can be focused on the boundary of the instances, where they differ the most. On the other hand, the mask prediction is done based on a higher resolution relative to the instance size. This gives the model more flexibility to capture fine details of the instance shape.

During inference, the upright output mask probabilities are rotated, translated and zoomed (with bilinear interpolation) according to the corresponding predicted box. Finally, the transformed mask probabilities are thresholded at 0.5 to obtain the output mask regions.

**Enlarged GT boxes.** The oriented box detection results tend to be very tight around the instance masks. On the one hand, this is exactly what we are aiming for because we want to use the resolution of the mask predictions as effectively as possible. On the other hand, as we have seen in Chapter 8, oriented box detection is more difficult than axis-aligned detection. In many cases, the oriented boxes do not fit accurately and clip parts at the boundary of the underlying objects. Since the boxes are aligned with the object orientation, this often means that a relatively large fraction of the object mask reaches outside of the predicted box. For *AAIS*, these artifacts are less severe, because in most cases only a small tip or a corner of the object is clipped by an imprecise box prediction. We hypothesize that enlarging the GT boxes by a few pixels has several advantages: First, the model learns to predict oriented boxes that do not clip the object mask. Second, the model learns that the mask is not always connected to the box, although in most cases it is really close. Third, we believe that *AAIS* models have one major advantage during the mask prediction. The boundary of the objects is in most cases given by sharp edges that are clearly within the box for *AAIS*. For *OIS*, these edges are either directly at the box boundary or they are very close. By enlarging the GT boxes, these edges are again within the box proposals and can hence be used within box regression or mask prediction heads of the model. Therefore, we evaluate models that are trained on models with enlarged boxes (box++). We enlarge the GT boxes by 10 px for small images of resolution $512 \times 384$ px, by 15 px for medium images of resolution $768 \times 512$ px, and by 20 px for images of resolution $1024 \times 768$ px. For the small images, we also tried an

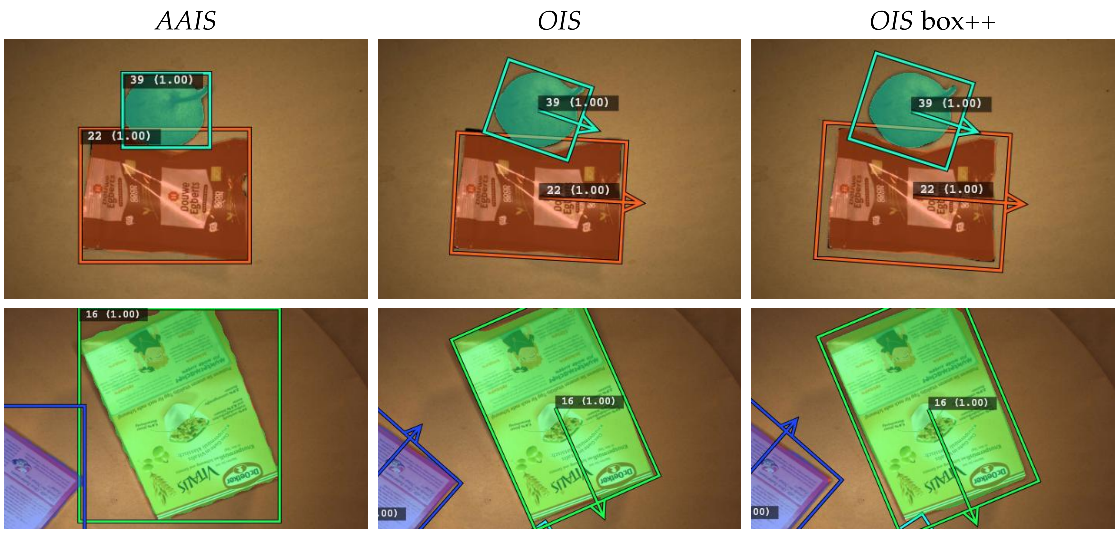


**Figure 10.4: Examples where enlarging the GT boxes helps for *OIS*.** If boxes are very tight, there is a higher chance to clip parts of the object boundary (*top, top-left corner of the coffee package*). Using enlarged boxes, the model avoids clipping the object boundaries and learns that the mask is not necessarily connected to the box. (*bottom*). Note the upsampling artifacts in the *AAIS* case: the waved mask boundary is visible but does not lower the IoU significantly (*bottom-left*).

enlargement by 20 px, but this did not improve the results. A detailed study to find the optimal enlargement, which may be relative to the box size, should be done in future work.

Fig. 10.4 shows two examples where the box++ model has an advantage. At first sight, the improvement of the box++ model compared to the *OIS* model might seem marginal, but sometimes a well predicted corner of a small instance mask might tip the scales for a prediction to be a TP at high IoU thresholds. A quantitative analysis of the effects of enlarging the GT boxes is carried out below in Section 10.3.3.

**Partial mask annotations.** Box-based instance segmentation models can be easily modified to be trainable only with mask GT available for a fraction of the training instances: Whenever the mask GT is not available for an instance, we set the loss weights of this instance within the mask branch to zero. Therefore, the prediction for this instance will be ignored during the loss computation.

*OIS* models require fewer pixel-precise mask annotations to obtain the same result quality as an *AAIS* method because the mask targets are more consistent between different instances and there is less noise from neighboring objects within the pooled features for mask prediction. Therefore, we show experiments with fewer GT mask annotations on *Pill Bags* and on *Screws* in Section 10.3.

We present two approaches to label instance masks only partially. The first is to label all instance masks within an image but only for a subset of the training images. The second is to label only a fraction of the instance masks but at least for some objects within each image. This second approach is similar to the *class-agnostic* setting in [77]. Of course, a combination of both approaches is possible, where only a fraction of the objects have mask annotations and only a subset of all training images contain mask labels at all.

Generally, partial mask annotations lead to sparse training signals for the mask prediction head of the network. If these signals are too seldom and the other losses for the box regression and classification are still high, the model might do well without predicting masks at all. Hence, at least with a decent probability, the batch should contain at least some mask annotations such that a mask loss is generated regularly. For both the *OIS* and *AAIS* models, we left the mask loss weight at its normal value, although it could help to increase it if fewer mask annotations are contained in the training set.

It is important to mention that still each object needs a box GT annotation. Otherwise, during training, the model would be told to assign the area containing the object to the background. This could destabilize the training and reduce the model performance.

## 10.3 Experiments

In the following subsections, we evaluate our *OIS* model on three different datasets: *D2S* (Chapter 5), *Screws*, and *Pill Bags* (Chapter 4). *Screws* does not have manually labeled instance masks, but we use the data generation technique of Chapter 7 to generate training and evaluation images with pixel-precise masks (*Screws Gen*, see below). Hence, all datasets have high-quality annotations such that a potentially improved mask accuracy can be measured.

An indicator for the potential benefit of *OIS* compared to *AAIS* is the comparison of the average IoU of the instance masks with the smallest AABB and smallest OBB, respectively. The mBMIoU averages these IoUs for the whole dataset:

$$\text{mBMIoU} = \frac{1}{C}\sum_{c=1}^{C}\frac{1}{N_c}\sum_{i=1}^{N_c}\text{IoU}(B_i, M_i), \tag{10.1}$$

where $C$ is the number of classes, $N_c$ is the number of instances per class and $\text{IoU}(B_i, M_i)$ is the IoU of the bounding box $B_i$ with the ground truth mask $M_i$ of instance $i$. The absolute mBMIoU value can also be interpreted as the average mask-to-background ratio of all pixels within the bounding box. Hence, a high or low mBMIoU does not automatically yield an easy or difficult mask prediction. However, the relative comparison of mBMIoU values in Table 10.1 reveals that on average, oriented bounding boxes are indeed tighter around the instances. For all datasets, the difference is significant and this indicates that the resolution of the mask prediction feature maps could be used more effectively for *OIS*. In the following, we show that this potential can indeed be utilized.

| dataset | axis-aligned | oriented |
|---|---|---|
| *Screws Gen* | 45% | 53% |
| *Pill Bags* | 72% | 82% |
| *D2S* | 61% | 80% |

**Table 10.1: Mean box mask** IoU**.** mBMIoU values computed on the validation set. For all three datasets, oriented bounding boxes have a much higher foreground-to-background ratio during mask prediction than axis-aligned bounding boxes.

### 10.3.1 Screws and Screws Gen

Since in *Screws*, the nut classes are symmetric and their orientation is not well-defined, these four classes are assigned to *classes without orientation*. In this dataset for screw classes, the exact orientation pointing from the screws head to its tail is of interest. Therefore, *IgnoreDirection* is set to *false* such that box orientations are in the range $(-\pi, \pi]$. Unless mentioned otherwise, on *Screws* and *Screws Gen*, all training hyperparameters of RetinaMask (RMask) and MRCNN are used similar to the RetinaNet and FRCNN models used on *Screws* in Chapter 8: we train the models for 60 epochs with initial learning rate 0.001 that is dropped by a factor of 0.1 after 30 and 50 epochs, respectively. We use horizontal and vertical mirroring. We use RPN levels three to six for RMask and two to five for MRCNN due to the higher anchor scale of MRCNN (8 instead of 4). For *OIS*, we use anchors with aspect ratios$(0.1, 0.3, 0.9)$ and orientations $(-2\pi/3, -\pi/3, 0, \pi/3, 2\pi/3, \pi)$. *AAIS* models use the default anchors with three aspect ratios $(0.5, 1.0, 2.0)$. For RMask, both model types use three anchor subscales, for MRCNN, only one is used. The class-specific NMS IoU threshold is set to 0.4 and the class-agnostic to 0.5.

**Generated data — *Screws Gen*.** Because the pixel-precise annotation of instance masks is tedious and time-consuming, we follow the approach of Chapter 7 and Chapter 9 to generate artificial training images. Therefore, each category is captured on a homogeneous white background where a relatively precise instance mask can be obtained by thresholding. In this weakly-supervised setting, we use a single template image per category and generate 1000 images (600 train, 200 validation, 200 test) by cropping and pasting random instances onto empty wooden backgrounds similar to those of the original dataset. Images are generated with the resolution $512 \times 384$ px. Example images of this generated dataset *Screws Gen* are shown in Fig. 10.3 and Fig. 10.6.

To train and evaluate *AAIS*, we obtain the box ground truth annotations as the smallest axis-aligned bounding box of instance masks. The oriented box ground truth for categories with orientation is generated as follows: The orientation of the box is calculated based on the second moments of the instance mask (cf. Appendix D.1). To get the orientation pointing from the screws head to its tail, the orientation is corrected by adding or subtracting $\pi$ if the orientation from the center of gravity to the most distant point on the mask boundary is pointing into the opposite direction. To have consistent box dimensions for objects that reach out of the image (long screws), we use the amodal mask (cf. Chapter 11) to obtain the initial coordinates of the amodal OBB. Then, we keep the orientation and only adjust the center and one of the semi-axes lengths, such that at least three of the OBB corner points are within the image. Thus, the final OBB is a tight bounding box for the cropped mask, but with the orientation of the amodal mask.

**Quantitative results.** The top part of Table 10.2 shows box detection results of *OIS* and *AAIS* models with *COCO* pretrained ResNet 101 backbones on the *Screws Gen* test set based on $\text{rIoU}_{\text{BB}}$. The *AAIS* models consistently predict better bounding boxes for the instances than the *OIS* models, in particular at higher $\text{rIoU}_{\text{BB}}$ thresholds. At $\text{rIoU}_{\text{BB}}$

| Model | Exp | IoU[0.5:0.95] | | IoU[0.5] | | | | IoU[0.8] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $AP$ | $AP^{\star}$ | $AP$ | $AP^{\star}$ | $TP$ | $FP$ | $AP$ | $AP^{\star}$ | $TP$ | $FP$ |
| Box rIoU$_{\text{BB}}$ | | | | | | | | | | | |
| AAIS RMask | 1 | 76.6 | 75.2 | 97.5 | 97.4 | 2160 | 25 | 79.1 | 77.1 | 1862 | 323 |
| AAIS MRCNN | 1 | 78.6 | 78.2 | 98.1 | 98.7 | 2172 | 0 | 82.2 | 81.3 | 1931 | 241 |
| OIS RMask | 1 | 71.0 | 68.8 | 97.6 | 96.2 | 2165 | 88 | 72.6 | 69.2 | 1749 | 504 |
| OIS MRCNN | 1 | 75.4 | 74.7 | 98.6 | 98.6 | 2179 | 15 | 80.7 | 79.0 | 1934 | 260 |
| Mask IoU | | | | | | | | | | | |
| *AAIS* RMask | 1 | 82.5 | 81.5 | 97.0 | 96.7 | 2151 | 34 | 89.7 | 88.2 | 2044 | 141 |
| | 2 | 82.4 | 81.5 | 96.8 | 96.7 | 2151 | 32 | 89.2 | 87.9 | 2039 | 144 |
| | 3 | 82.5 | 81.5 | 96.7 | 96.4 | 2147 | 33 | 89.4 | 88.0 | 2040 | 140 |
| *AAIS* MRCNN | 1 | 83.8 | 83.2 | 97.7 | 98.0 | 2164 | 8 | 92.8 | 92.0 | 2097 | 75 |
| | 2 | 84.2 | 83.5 | 98.2 | 98.4 | 2170 | 6 | 92.9 | 91.8 | 2097 | 79 |
| | 3 | 83.9 | 83.4 | 97.8 | 98.1 | 2163 | 5 | 93.3 | 92.3 | 2100 | 68 |
| *OIS* RMask | 1 | 84.8 | 82.7 | 98.2 | 96.8 | 2172 | 81 | 92.3 | 89.6 | 2070 | 183 |
| | 2 | 84.4 | 82.8 | 97.5 | 96.8 | 2157 | 49 | 92.2 | 90.3 | 2067 | 139 |
| | 3 | 84.7 | 83.1 | 97.8 | 97.1 | 2166 | 50 | 92.5 | 90.5 | 2075 | 141 |
| *OIS* MRCNN | 1 | 86.4 | 85.8 | 98.8 | 98.9 | 2183 | 11 | 96.0 | 95.5 | 2141 | 53 |
| | 2 | 86.2 | 85.9 | 98.5 | 98.9 | 2179 | 3 | 96.1 | 95.8 | 2142 | 40 |
| | 3 | 86.6 | 86.0 | 99.2 | 99.3 | 2189 | 7 | 96.4 | 95.9 | 2146 | 50 |
| Without *classes without orientation* | | | | | | | | | | | |
| *OIS* RMask | 1 | 83.8 | 81.9 | 97.5 | 96.5 | 2162 | 66 | 91.3 | 88.8 | 2058 | 170 |
| | 2 | 83.4 | 81.7 | 97.4 | 96.5 | 2157 | 54 | 91.0 | 88.8 | 2052 | 159 |
| | 3 | 83.6 | 81.8 | 97.3 | 96.4 | 2154 | 54 | 90.8 | 88.7 | 2050 | 158 |
| *OIS* MRCNN | 1 | 85.9 | 85.4 | 98.6 | 98.9 | 2179 | 6 | 95.3 | 94.8 | 2127 | 58 |
| | 2 | 86.0 | 85.4 | 98.9 | 99.0 | 2183 | 9 | 95.4 | 94.8 | 2129 | 63 |
| | 3 | 85.7 | 85.3 | 98.3 | 98.7 | 2174 | 2 | 95.5 | 95.0 | 2126 | 50 |

**Table 10.2: Box and mask results on *Screws Gen* test with $T_s^{\star}$.** *AAIS* models predict more accurate bounding boxes than *OIS* models (*top*). However, the *OIS* models clearly outperform the *AAIS* counterparts when the mask predictions are evaluated (*middle*). Setting nut categories to *classes without orientation* further improves the results (*bottom*).

threshold 0.5, *AAIS* MRCNN has 100% precision (no FPs), but *OIS* MRCNN has a slightly higher recall. *OIS* RMask predicts significantly more FPs than all other models, probably because of the large number of anchor boxes and the used class-specific score thresholds $T_s^{\star}(val)$ that do not generalize very well to the test set. Hence, these results are in line with the findings of Chapter 8.

However, the situation is turned around with the evaluation of the mask IoU in the lower part of Table 10.2. Here, the *OIS* models clearly outperform their *AAIS* counterparts and the mean $AP^{\star}$ is increased between 1.2 pp and 2.5 pp. For all models, the mask IoU result at threshold 0.8 is much higher than the corresponding rIoU$_{\text{BB}}$ results. This means that if the boxes are inaccurate, often only small parts of the objects are clipped and the models still predict most of the mask area correctly.

**Classes without orientation.** In Chapter 8, we have seen that setting *classes without orientation* can help to get consistent oriented box detection results for classes where the orientation is ambiguous. Therefore, on *Screws* and *Screws Gen*, we assign the nut categories to *classes without orientation*. However, for instance segmentation models, the IoU of the predicted masks to the given GT is of interest. Hence, in theory, *classes without orientation* are not necessary to set because independent of the predicted bounding box orientation, the model can still predict a correct mask.

For the experiments shown at the bottom of Table 10.2, we do not use *classes without orientation*. Instead, for all categories the orientation of the bounding box is inferred from the GT mask. The results indicate that on *Screws Gen*, we get competitive instance mask results even without using *classes without orientation*. However, at higher IoU thresholds, our proposed model that uses the AABBs as GT for the nut categories achieves slightly better results both for the RMask and MRCNN architecture.

**Partial mask annotations.** To evaluate a model with partial mask annotations, we randomly sample a number of generated training images with mask labels from the *Screws Gen* training set and add them to the *Screws* training images that do not have any instance mask labels. This means that in these experiments some images contain mask annotations for all objects within the image and some images do not have any mask annotations. We use the *Screws Gen* validation and test sets for model selection (early stopping) and the quantitative analysis. Moreover, we successively add more generated images with mask annotations to the training set to see the dependence of the results on the number of images with mask supervision. We add 20, 50, 100, 300, and all 600 training images from *Screws Gen* to the 269 training images of *Screws*. To have a fair comparison, all experiments run for approximately 34 700 iterations, which is the equivalent of 40 epochs for the dataset with all 600 *Screws Gen* training images. The learning rate is initialized at 0.001 and dropped to 0.0001 after 26 000 iterations (30 epochs for the largest training set). We perform three runs for each experiment, where each of the runs uses a different random selection of generated images. To ensure a fair comparison, the selection of images is the same for *AAIS* and *OIS* models.

We show the mean $AP^{\star}$ and standard deviation over the runs in Fig. 10.5: *OIS* needs significantly fewer generated images to obtain the same results as *AAIS*. On average, the *AAIS* model needs approximately twice the number of images with masks in the training set to reach the same $AP^{\star}$ as the *OIS* model. Hence, due to less variation in the mask targets and predictions, the *OIS* model needs significantly fewer mask annotations. This is a great benefit for the user, who can reduce the labeling time and potentially use a smaller dataset for training.

The comparison between *COCO* and *ImageNet* pretrained weights shows that a pretrained mask head is mainly useful if only few images contain labeled instance masks. If more generated images are added (>100), the *ImageNet* BN model consistently outperforms the *COCO* pretrained model. *ImageNet* BN model uses batch normalization layers within the mask head; also see the analysis on *D2S* below.

**Qualitative results.** Results for MRCNN R101 *COCO* on *Screws Gen* are shown in Fig. 10.6. Due to the low variance of object appearance, for all models the quality of

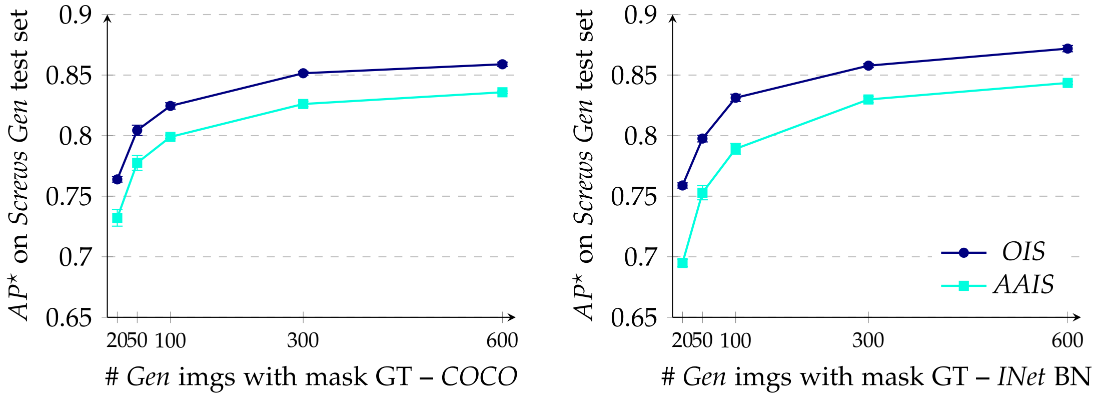


**Figure 10.5: *Screws (Gen)* partial mask supervision.** The results are for MRCNN R101 *COCO* (*left*) and *ImageNet* BN (*right*) using $T_s^\star(val)$. Markers show the mean and error bars (hardly visible) the standard deviation for three runs of each experiment; see text for further explanation. We successively add more generated training images with mask annotations from *Screws Gen* to the *Screws* training set. For *OIS*, with only half the number of additional generated images, *AP** is on the same level as for *AAIS*.

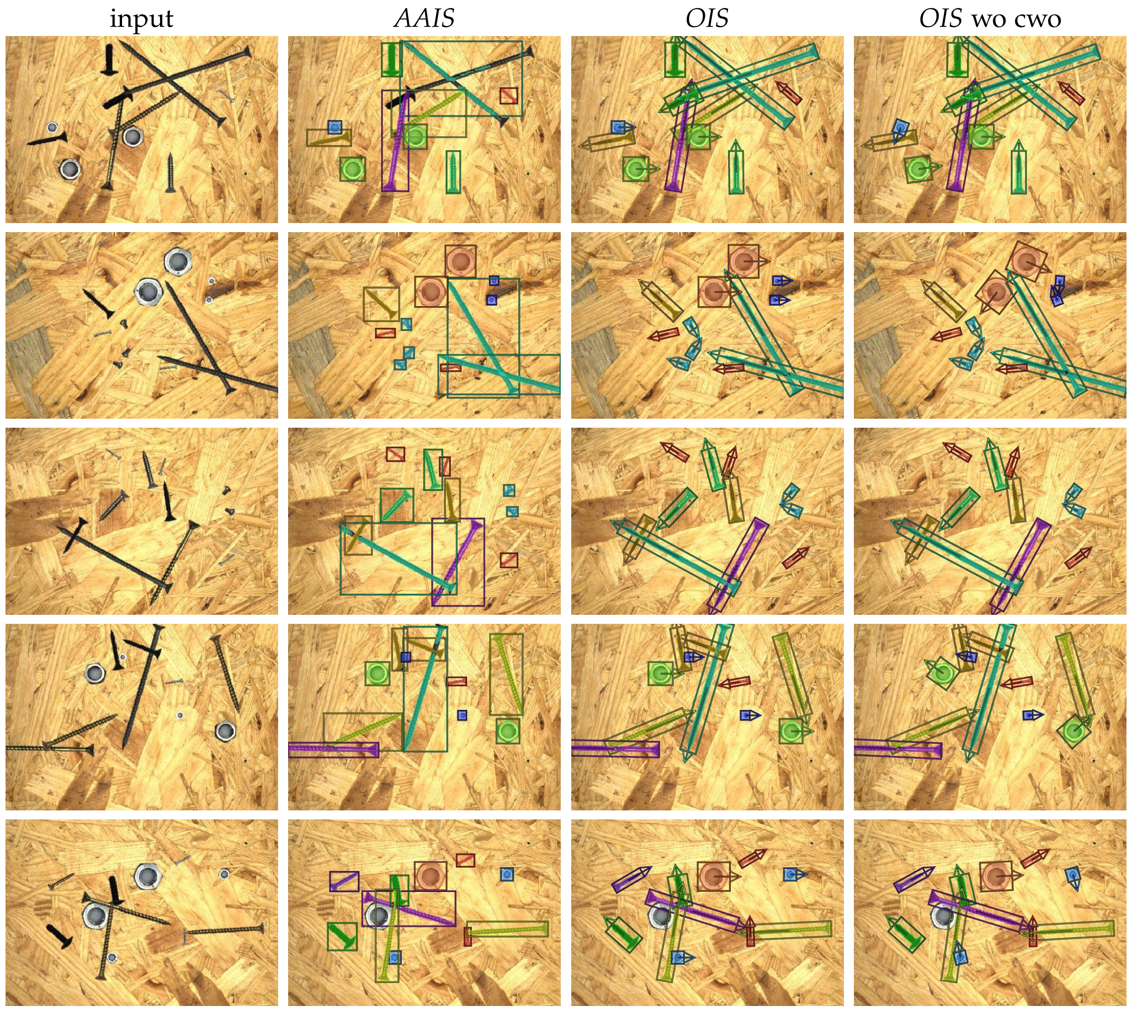


**Figure 10.6: *Screws Gen* qualitative results.** Generated input image, results of *AAIS*, results of *OIS*, results of *OIS* without *classes without orientation (from top to bottom)*. *OIS* improves the mask accuracy, especially at fine details such as the screw heads. Using *classes without orientation* leads to consistent box orientations, but improves the mask precision only slightly.

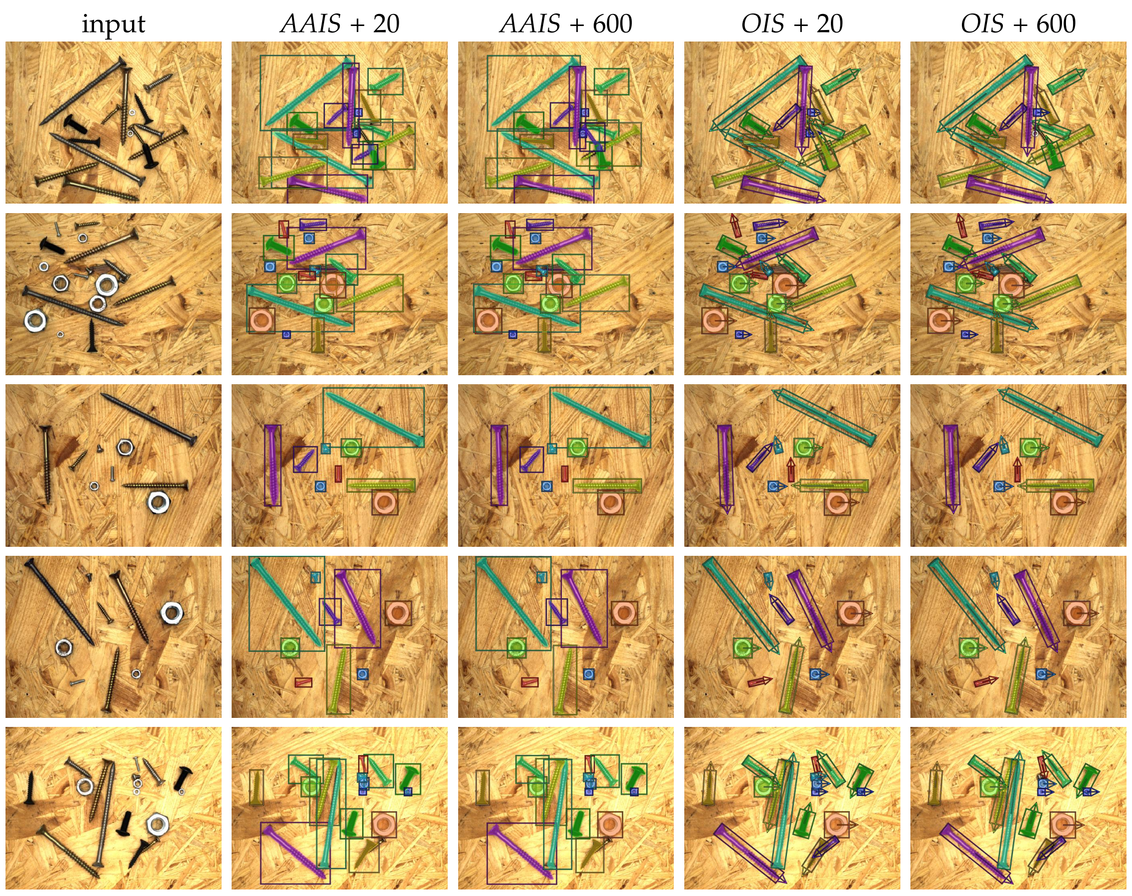


**Figure 10.7: *Screws* qualitative results.** *AAIS* predicts too wide masks for the thin screws that are diagonally aligned. Already with 20 additional generated images, *OIS* predicts very accurate masks (*best viewed digitally and with zoom*).

predicted masks is high. In some cases, *AAIS* struggles if objects are very close or overlapping and predicts the overlap only very coarsely. In those cases *OIS* is usually better. However, both models often confuse which object is lying on top of the other. Sometimes, objects are not found by all models, maybe due to a high $T_s^\star(val)$ threshold (*the large nut in the bottom row*).

Additionally, we show results on *Screws* for models that have been trained with partial mask annotations in Fig. 10.7. For these models, the mask prediction head has only been trained on the generated images that have been added to the training set. Although there is a domain gap between *Screws* and *Screws Gen*, the results are accurate. If the *AAIS* model is trained with few generated images, it cannot predict the screw heads properly and the masks for diagonally aligned screws are too wide. This is improved if more generated images are added to the training set. In comparison, the *OIS* model already predicts accurate masks with only 20 generated images added to the training set. There is hardly any visual difference between the results of *OIS* + 20 and *OIS* + 600. Also here, sometimes results might be filtered out due to high $T_s^\star(val)$ values that are calculated on the *Screws Gen* validation set.

### 10.3.2 Pill Bags

Also on *Pill Bags*, we use the same training setup as in Chapter 8. For mask prediction, we use a RetinaMask architecture with a SqueezeNet backbone that has been pretrained on *OpenImages*. In this case, we use a lightweight mask head with only two intermediate convolution layers (instead of four) and a feature map depth of 128 (instead of 256). Moreover, we reduce the training to 25 epochs and reduce the initial learning rate by a factor of 0.1 after 10 and 20 epochs, respectively. We will see that due to the small variations between the samples, this suffices to obtain strong results on this dataset. At the same time, using an NVIDIA RTX 2080 Ti GPU leads to training times of one model well below 10 minutes.

**Partial mask supervision.** Like in a typical industrial application, all images of *Pill Bags* have more or less the same appearance. Moreover, the pills do not have many deformations and their shape is close to that of an ellipse. Therefore, it is not surprising that the instance segmentation task can already be solved quite well with *AAIS*. However, also on *Pill Bags*, it is tedious to annotate the full training set with pixel-precise masks. Hence, we analyze how many of the instance mask labels are necessary to obtain a good result. Therefore, we delete all but a fraction of the annotated GT instance masks and whenever the mask is not available during training, we set the corresponding mask loss weight to zero. This allows to train with a fraction of the expensive instance mask labels. Note that we delete the GT masks randomly from the training set, i.e., it could happen that by chance for some categories only a few or even no mask annotations are present within the training set. However, due to the relatively high number of instances per class in the training images (at least 230 per class for *Pill Bags*, cf. Section 4.1.2), this is very unlikely.

Fig. 10.8 shows that for *OIS* we can already successfully train instance segmentation if only 10% of the masks are annotated in the training set — without a large reduction in mask $AP^{\star}$ on the test set. Using 30% of the annotations, *OIS* already outperforms the *AAIS* model with all annotations by 3 pp. In comparison, the *AAIS* method requires more annotations to obtain a reasonable result. Even when using all annotations, the *AAIS* $AP^{\star}$ is only 1 pp higher than the 10% counterpart of *OIS*. This can be explained by the higher variance of the mask within the axis-aligned bounding boxes. The comparison to the $AP$ values reveals that both model types predict only few FPs on *Pill Bags*.

**Qualitative results.** Some qualitative results on *Pill Bags* are shown in Fig. 10.9. When the *AAIS* model is trained with only 10% of the instance mask annotations, the predictions are very weak: Masks reach into neighboring objects and often contain holes (*2nd column*). The results are significantly better when all mask labels are used during training, but still for some predictions, the mask reaches out of the object boundary (*3rd column*). This is not the case for the *OIS* 10% model, where the mask predictions are bounded by tight oriented bounding boxes. Sometimes the predicted mask is too narrow, in particular when objects are touching (*4th column*). This issue is less severe for the *OIS* model trained

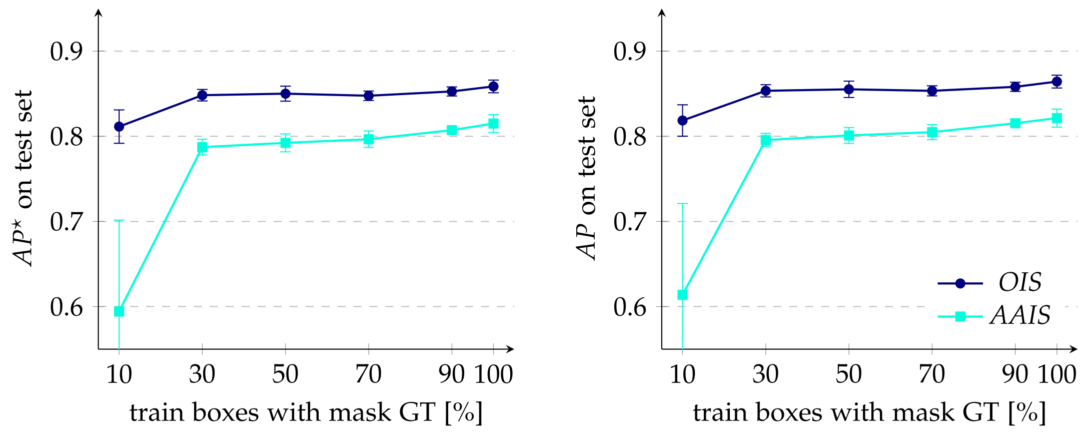


**Figure 10.8: *Pill bags* partial mask supervision.** For *OIS*, if only 10% of the boxes contain a mask ground truth, the *AP* is on the same level as with all mask annotations for *AAIS*. *AAIS* needs at least 30% of the boxes labeled with instance masks to obtain reasonable results.

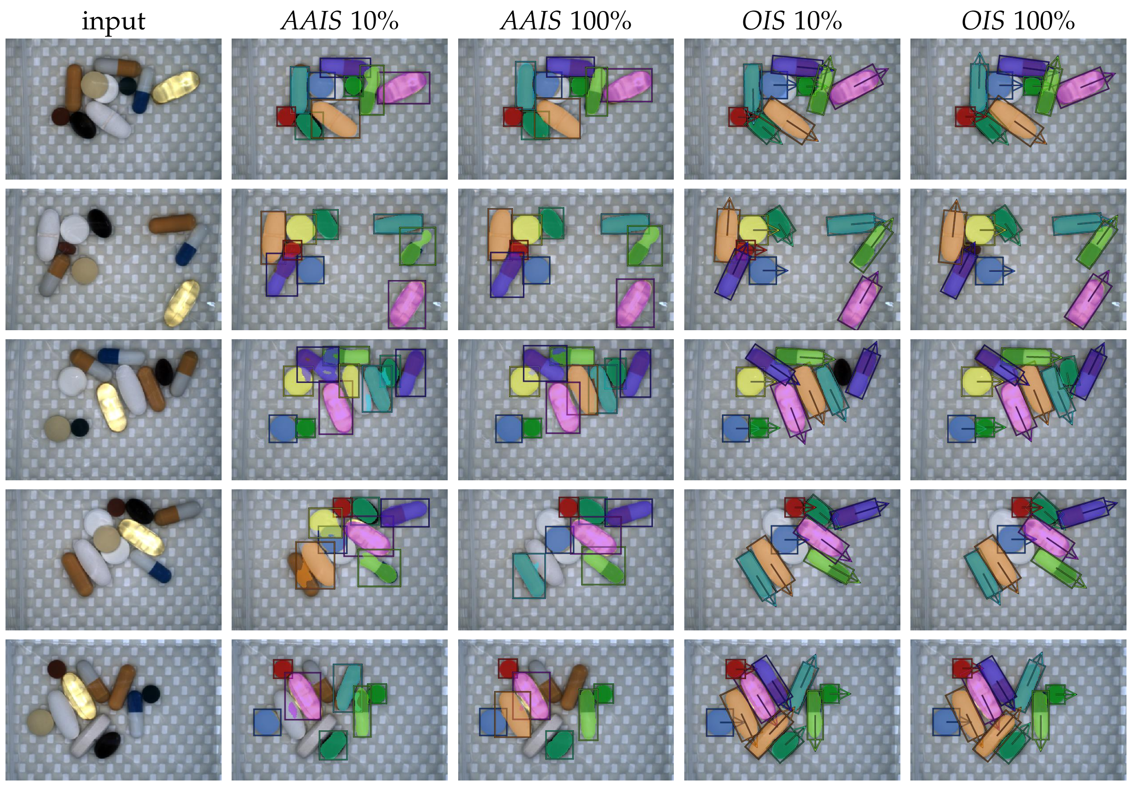


**Figure 10.9: *Pill Bags* qualitative results.** Images are cropped to show result details. The models in the second and fourth column have been trained with only 10% of the GT instance masks available. *OIS* models predict clearly more accurate instance masks. See text for further explanations.

with all mask labels that clearly outperforms all other models. However, even this model struggles with occluded pills that are rarely occuring within the training set (*5th column*).

### 10.3.3 D2S

In *D2S* (Chapter 5), there are elongated objects such as lying bottles, boxes, or certain vegetables like carrots or zucchinis, but also round or highly deformable objects, such as standing bottles, apples, or nets filled with oranges. We have seen in Chapter 8 that the orientation of some objects is ambiguous and oriented box detection results are slightly worse than axis-aligned box detection results on *D2S*. Hence, in comparison to the datasets above, it is unclear if *OIS* can lift its potential also on this dataset.

**Data preparation and model settings.** Unless mentioned otherwise, to have a fair comparison, we use exactly the same settings as in our baseline experiments in Section 5.6. For *D2S*, we do not use *classes without orientation* and set *IgnoreDirection* to *true*. For *OIS* and *AAIS*, we use the same anchor boxes that we have used in Section 8.4 for *OD* and *AAD* on *D2S*, respectively.

**Mask prediction only.** To get a first impression how large the potential of *OIS* is on *D2S*, we train only the mask prediction head of both model types (*AAIS* and *OIS*). Therefore, we use a RetinaMask architecture where we replace the RPN by feeding the GT boxes into the model and only predict the mask for the given GT boxes.

Since training only the mask prediction is an easier task than training the whole box detection and mask prediction pipeline, we shorten the training time from 30 to 15 epochs and reduce the initial learning rate by a factor of 0.1 after 8 and 13 epochs, respectively. We train both models with *ImageNet* and *COCO* pretrained weights. For *ImageNet*-pretrained weights, we had to change the convolution+ReLU blocks within the mask head to convolution+ReLU+BN blocks. Without this change, the mask predictions were either empty or covered the whole GT box. We also evaluate if the additional batch normalization layers within the mask head help for *COCO*-pretrained weights.

Table 10.3 shows that using GT boxes *OIS* improves the mask prediction consistently. However, the gain on *D2S* is not as large as the advantage with respect to mBMIoU in Table 10.1 suggests. For *ImageNet*-pretrained weights both model types require additional BN layers within the mask head. If those are used, the mask prediction is on the same level as with *COCO*-pretrained weights. Using BN layers with *COCO*-pretrained weights does not further improve the results. Increasing the oriented GT boxes by 10 px (box++) further improves the *OIS* model despite a slightly worse mask-to-background ratio.

**Mask and class prediction.** The results in Chapter 8 have shown that oriented detection is more difficult than axis-aligned detection. However, in theory, the pooled features from oriented boxes contain less noise from neighboring objects than for axis-aligned boxes. Hence, *OIS* could also lead to better classification results. To investigate this in practice, we again run experiments based on the GT boxes. This time, we use an MRCNN-like

| | | R50 | | | R101 | | | |
|---|---|---|---|---|---|---|---|---|
| | Weights | *INet* | *INet* | *COCO* | *INet* | *COCO* | *COCO* | *COCO* |
| Model | Exp | | BN | | BN | | BN | box++ |
| *AAIS* | 1 | 31.6 | 92.6 | 93.0 | 93.2 | 93.6 | 93.5 | |
| | 2 | 31.6 | 92.7 | 93.1 | 93.2 | 93.4 | 93.4 | |
| | 3 | 31.6 | 92.4 | 93.0 | 93.2 | 93.4 | 93.5 | |
| *OIS* | 1 | 68.9 | **94.1** | 94.1 | **94.4** | **94.4** | 94.4 | 94.5 |
| | 2 | 68.9 | 94.0 | 94.1 | **94.4** | 94.2 | 94.2 | **95.1** |
| | 3 | 68.9 | 94.0 | **94.2** | **94.4** | 94.3 | **94.5** | 95.0 |

**Table 10.3: *AAIS* vs. *OIS* mask prediction.** The table shows *AR* values in % on the *D2S* validation set with small images. *OIS* consistently improves the mask prediction by 1 pp *AR*.

| Model | Exp | IoU[0.5:0.95] | | | IoU[0.5] | | | IoU[0.95] | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | *AP* | *AP*$^\star$ | *AR* | *AP* | *AP*$^\star$ | *AR* | *AP* | *AP*$^\star$ | *AR* |
| *AAIS* R101 *ImageNet* BN | 1 | 78.5 | 76.0 | 82.1 | 86.3 | 84.5 | 88.0 | 39.0 | 34.8 | 51.5 |
| | 2 | 79.0 | 76.5 | 82.7 | 87.2 | 85.3 | 88.8 | 38.2 | 34.4 | 51.6 |
| | 3 | 77.9 | 75.2 | 81.8 | 85.6 | 83.5 | 87.6 | 38.3 | 34.3 | 51.7 |
| *AAIS* R101 *COCO* | 1 | 80.3 | 77.9 | 83.9 | 87.3 | 85.6 | 89.1 | 42.5 | 38.8 | 55.5 |
| | 2 | 78.6 | 76.1 | 82.3 | 85.5 | 83.5 | 87.5 | 41.7 | 37.7 | 54.4 |
| | 3 | 79.3 | 76.7 | 83.0 | 86.3 | 84.3 | 88.2 | 42.0 | 38.1 | 55.2 |
| *OIS* R101 *ImageNet* BN | 1 | 78.5 | 76.0 | 82.0 | 85.7 | 83.9 | 87.6 | 35.3 | 31.4 | 48.5 |
| | 2 | 79.0 | 76.7 | 82.5 | 85.7 | 84.0 | 87.5 | 40.3 | 36.2 | 53.0 |
| | 3 | 79.6 | 77.1 | 83.0 | 86.5 | 84.6 | 88.2 | 40.4 | 36.2 | 53.1 |
| *OIS* R101 *COCO* | 1 | 80.6 | 78.2 | 83.8 | 86.7 | 84.9 | 88.4 | 44.5 | 40.5 | 56.8 |
| | 2 | 79.6 | 77.3 | 83.4 | 85.9 | 84.1 | 88.1 | 43.2 | 39.1 | 56.1 |
| | 3 | 80.4 | 77.9 | 83.9 | 86.5 | 84.6 | 88.4 | 44.1 | 40.0 | 56.6 |
| *OIS* R101 *COCO* box++ | 1 | 80.9 | 78.4 | 84.0 | 86.9 | 84.9 | 88.5 | 47.0 | 42.7 | 58.8 |
| | 2 | 80.9 | 78.5 | 84.3 | 86.8 | 84.9 | 88.7 | 47.1 | 42.9 | 59.0 |
| | 3 | 80.7 | 78.3 | 83.9 | 86.7 | 84.9 | 88.4 | 47.1 | 42.9 | 59.1 |

**Table 10.4: *AAIS* vs. *OIS* mask and class prediction.** The table shows results using $T_s^\star(val)$ on the *D2S* test set (S). On average, *OIS* improves the accuracy of the instance segmentation results based on GT boxes somewhat, but obviously the class prediction is not significantly better than for *AAIS* models. The model with enlarged GT boxes clearly outperforms the *AAIS* baseline.

architecture where the RPN is replaced by using the GT boxes as proposals directly. In this case, we only use the GT box parameters and use a Fast R-CNN classification head to predict the box category and an MRCNN mask head to predict the pixel-precise instance masks.

Mask and class prediction results are shown in Table 10.4. Since the evaluated models predict a score (the confidence of the class prediction), we can calculate meaningful *AP* and *AP*$^\star$ values. This allows to compare the results to the full model results below.

Both for *AAIS* and *OIS* models, the different training runs lead to more variance in the results than for the only mask prediction models. Moreover, in contrast to the models that predict only masks of Table 10.3, now the *OIS* models are not consistently outperforming their *AAIS* counterparts. On average, the *OIS* models are still performing better, but now the advantage shrinks to below one percentage point. This indicates that

| Model | 512 × 384 px (S) | | | | 768 × 512 px (M) | | | | 1024 × 768 px (L) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | val | | test | | val | | test | | val | | test | |
| | *AP* | *AP*⋆ | *AP* | *AP*⋆ | *AP* | *AP*⋆ | *AP* | *AP*⋆ | *AP* | *AP*⋆ | *AP* | *AP*⋆ |
| *AAIS* RMask | 78.4 | 75.4 | 77.4 | 73.5 | 79.5 | 76.5 | 79.5 | 75.8 | 78.3 | 75.1 | 78.3 | 74.2 |
| *AAIS* MRCNN | 77.1 | 74.8 | 75.4 | 73.0 | 79.6 | 77.5 | 78.7 | 76.3 | 79.9 | 77.7 | 78.2 | 75.6 |
| *AAIS* RMask* | 80.4 | 77.7 | 79.8 | 76.6 | 82.3 | 79.8 | 82.2 | 78.9 | 82.6 | 80.3 | **82.6** | 79.9 |
| *AAIS* MRCNN* | 80.3 | 78.2 | 78.6 | 76.4 | 82.7 | 80.9 | 81.4 | 79.4 | 82.0 | 80.3 | 82.0 | **80.0** |
| *OIS* RMask | 77.3 | 74.1 | 74.8 | 71.2 | 78.5 | 75.5 | 78.3 | 74.5 | 78.6 | 75.1 | 77.4 | 73.3 |
| *OIS* MRCNN | 78.2 | 75.7 | 76.0 | 73.2 | 80.7 | 78.8 | 78.4 | 76.0 | 79.7 | 77.8 | 77.7 | 75.3 |
| *OIS* RMask box++ | 79.0 | 75.6 | 76.9 | 73.0 | 79.3 | 76.3 | 78.7 | 74.9 | 80.2 | 76.9 | 78.8 | 74.9 |
| *OIS* MRCNN box++ | 79.1 | 76.9 | 76.3 | 73.7 | 81.5 | 79.6 | 79.0 | 76.6 | 81.5 | 79.5 | 78.7 | 76.4 |
| *OIS* RMask* box++ | **81.3** | 78.3 | **80.0** | **76.8** | 82.2 | 79.9 | 81.8 | 79.2 | 83.2 | 80.6 | 82.1 | 79.3 |
| *OIS* MRCNN* box++ | 81.1 | **79.0** | 78.9 | 76.7 | **83.4** | **81.5** | **82.4** | **80.2** | **83.6** | **82.0** | 81.0 | 79.1 |

**Table 10.5: *OIS* vs. *AAIS* results on *D2S* val and test.** All models use a ResNet 101 backbone and $T_s^\star(val)$ score thresholds; models marked with * use the *random background* and *neighboring* augmentation from Chapter 7. *OIS* is clearly better on the validation set, but the benefit does not transfer in the same amount to the test set. Still, with enlarged GT boxes, the *OIS* MRCNN model achieves the highest $AP^\star$ values.

*OIS* models are not better classifiers than the *AAIS* models. A possible reason might be that the oriented RoI Align operation introduces artifacts into the features that are used for the classification head and thus the theoretical advantage due to a more effective use of the given feature resolution does not translate to any gains in practice.

However, for the highest IoU threshold 0.95, *OIS* R101 *COCO* still yields a 1–2 pp $AP$ and $AP^\star$ improvement above the *AAIS* model. The *OIS* box++ model predicts even more accurate masks and further improves the results at the 0.95 IoU threshold by approximately 3 pp. Moreover, the *OIS* box++ model has much less variance over the three different runs and gives a significant improvement above the *AAIS* baseline.

**Results of the full *OIS* model on *D2S*.** Table 10.5 shows a quantitative comparison between the RetinaMask (RMask) and MRCNN *AAIS* baselines and our *OIS* models on *D2S*. For the box++ models, the GT boxes have been enlarged by 10 px, 15 px, and 20 px for the S, M, and L image sizes, respectively. The table reveals that using enlarged GT boxes improves the results of the *OIS* RMask and MRCNN models across all image sizes.

Overall, *OIS* MRCNN box++ is the best model with the highest $AP$ and $AP^\star$ values in most cases. For small and medium image size, the clear advantage over *AAIS* MRCNN does not generalize to the test set. For RMask, the *OIS* version is better for the large image size, but not on small or medium images. Here, both model types are on the same level on the validation set. However, on the test set, *AAIS* RMask is better. To get further insights for the possible reasons of this behavior, we carry out a per-class analysis below.

In comparison to the experiments using GT boxes in Table 10.4, *OIS* MRCNN box++ (S) only falls short around 5 pp $AP^\star$ and 4 pp $AP$. On the one hand, this means that the model's localization accuracy is very high. On the other hand, unless the predicted bounding boxes clip a large fraction of the object area, we do not need perfectly accurate

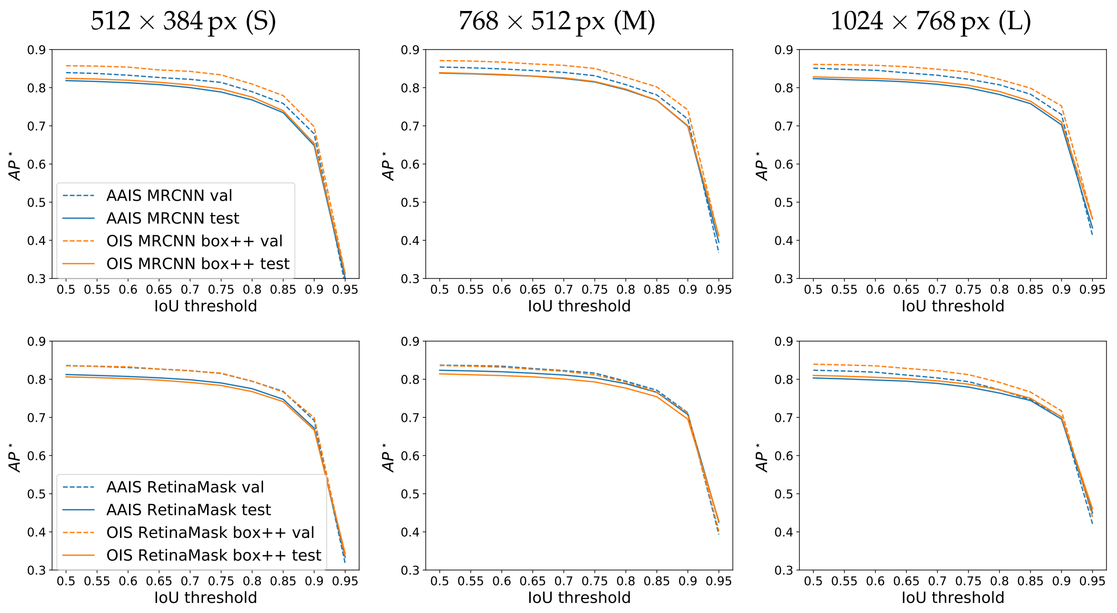


**Figure 10.10: Comparison per** IoU **threshold on *D2S*.** *OIS* MRCNN box++ clearly outperforms *AAIS* MRCNN on the validation set. On the test set, the benefit is less pronounced but still present for all image sizes (*top*). For RetinaMask, the *OIS* box++ model is only significantly better for a large image size (*bottom*). Note that an increased image size mainly improves *AP*⋆ at higher IoU thresholds (*from left to right*).

boxes to predict accurate instance masks.

If the images with specific *neighboring* or *random background* augmentations from Chapter 7 are added to the training set, the models improve by up to 3.4 pp *AP*⋆. *OIS* MRCNN* box++ (M) in the bottom row of Table 10.5 achieves the highest *AP*⋆ value of 80.2%. This is a 3.9 pp improvement to the baseline model *AAIS* MRCNN (M) from Chapter 5. However, for image size L, the *AAIS* models are again better on the test set.

Fig. 10.10 shows that in most cases *OIS* consistently improves the *AP*⋆ across all IoU thresholds. In other cases, *AAIS* is better for the lower IoU thresholds but slightly worse at IoU 0.95. Increasing the image size mainly improves *AP*⋆ at high IoU thresholds.

**Per-class comparison.** It is unclear why the improvement of *OIS* MRCNN box++ over *AAIS* MRCNN on the validation set does not transfer directly to the test set. Hence, we do a per-class analysis to see if the effect occurs for all categories in the same manner or only for some of them. Fig. 10.11 reveals that indeed the *AP*⋆ gains of *OIS* MRCNN box++ (M) on the validation set are due to small improvements for the majority of the categories and large improvements for some classes (e.g., *augustiner_weissbier, coca_cola_05, banana_bundle, banana_single, orange_single*). Only for a few of the bottle or tea categories, *AAIS* MRCNN is significantly better.

Still, on the test set, for the majority of the categories *OIS* and *AAIS* are on the same *AP*⋆ level, with a slight advantage for *OIS*, which also reflects the overall *AP*⋆ improvement of 0.3 pp. However, for some of the categories, the situation turns around on the test set: While the *OIS* model is clearly better on the validation set, it performs only on

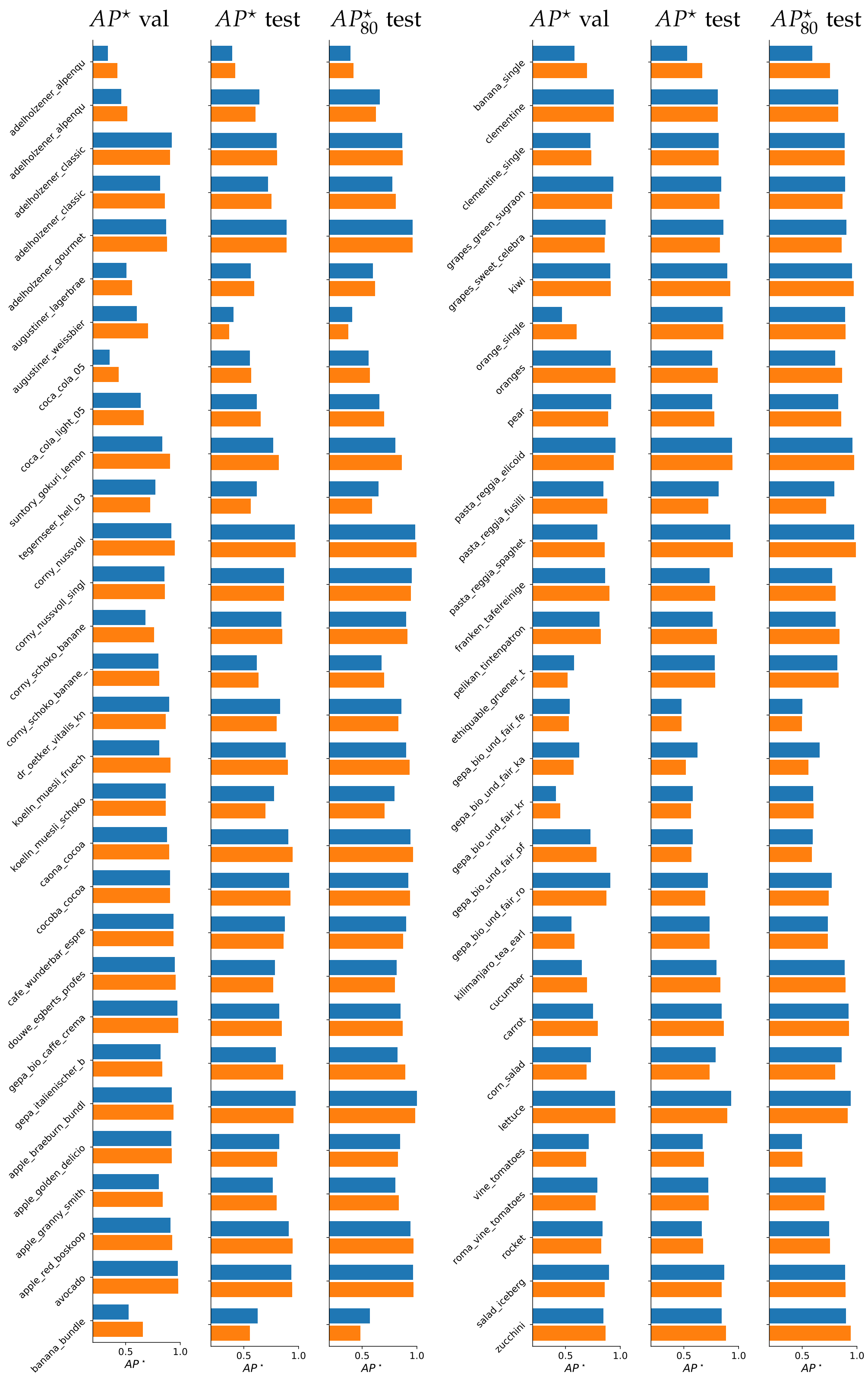


**Figure 10.11: Per-class $AP^{\star}$ comparison on *D2S*.** $AP^{\star}$ values on the validation and test set of the *AAIS* MRCNN (M) and *OIS* MRCNN box++ (M) models (*blue and orange*). We further provide test set results at IoU 0.8 to see if the differences are related to more accurate localizations. For some categories, there is a clear improvement on the validation set that turns into a disadvantage on the test set (e.g., *adelholzener_alpenquelle, augustiner_weissbier*).

the same level as the *AAIS* model on the test set (e.g., *orange_single*) or is even significantly worse (e.g., *augustiner_weissbier, banana_bundle*). There are also some classes where an *OIS* disadvantage on the validation set turns into an advantage on the test set, but usually the test set advantage is less pronounced (e.g., *adelholzener_classic, ethiquable_gruener_tee*) There could be several reasons for these effects: Classification errors, Localization errors, a generally higher number of additional FPs that reduce the precision, or a bias within the validation and test sets that favors one or the other method. For the latter reason it is hard to find out if such a bias exists. Clearly, it would be less pronounced if both evaluation sets and, in particular, the relatively small validation set of *D2S* were larger.

In Fig. 10.12, we analyze the distribution of predictions for categories with large $AP^\star$ differences between the validation and test set: For *adelholzener_alpenquelle_naturell_075* in the first two rows, it is hard to see any difference between the validation set distributions of the *AAIS* and *OIS* models. The number of FNs and FPs is slightly lower, the number of TPs somewhat higher. This means that the relatively large $AP^\star$ advantage of *OIS* arises from only a few instances that the model predicted correctly. On the test set, *OIS* still has slightly fewer FPs, but also a lower recall than *AAIS*, which explains the disadvantage with respect to $AP^\star$. We see that probably a lower class-specific score threshold would lead to a better recall for *OIS* and an improved overall result. Also for *augustiner_weissbier* in the third and fourth row, $T_s^\star(val)$ fits better for *AAIS* than *OIS* on the test set. Moreover, the precision of *OIS* is much worse on the test set. This class is also very often confused with the very similar class *augustiner_lagerbraeu_hell*. As above, it seems that if a handful of instances were predicted with the correct class, both the precision and recall would increase significantly. On the one hand, this means that *OIS* does not improve the fine-grained classification on *D2S*. On the other hand, larger evaluation and test sets could help to get a clear judgement which of the two methods is better or whether they are just equally good for this category. The class *banana_bundle* in row five and six is another category that belongs to one of these twin categories because it is frequently confused with *banana_single* (and vice-versa). Here, *OIS* clearly has a better recall and precision on the validation set. However, on the test set there are certainly more FPs with a bad localization that just fall under the 0.5 IoU threshold. We show an example of such a case in the qualitative results below. The class *gepa_bio_und_fair_kraeuter_tee* shown in the bottom two rows is one of the tea box categories that are often confused among each other. Thus, the number of TPs and FNs is approximately on the same level for both *AAIS* and *OIS* on the validation set. Here, *OIS* has a small advantage, but both approaches perform weakly. On the test set, we get the same picture, but this time with the better end for *AAIS*. Also, this shows that *OIS* does not perform significantly better with respect to fine-grained classification on *D2S*.

The per-class comparison in Fig. 10.11 also shows that for approximately half of the categories the results are already at a very high level and that one has to get rid of these poor-performing classes within the bottles and tea boxes group in order to significantly improve the overall result.

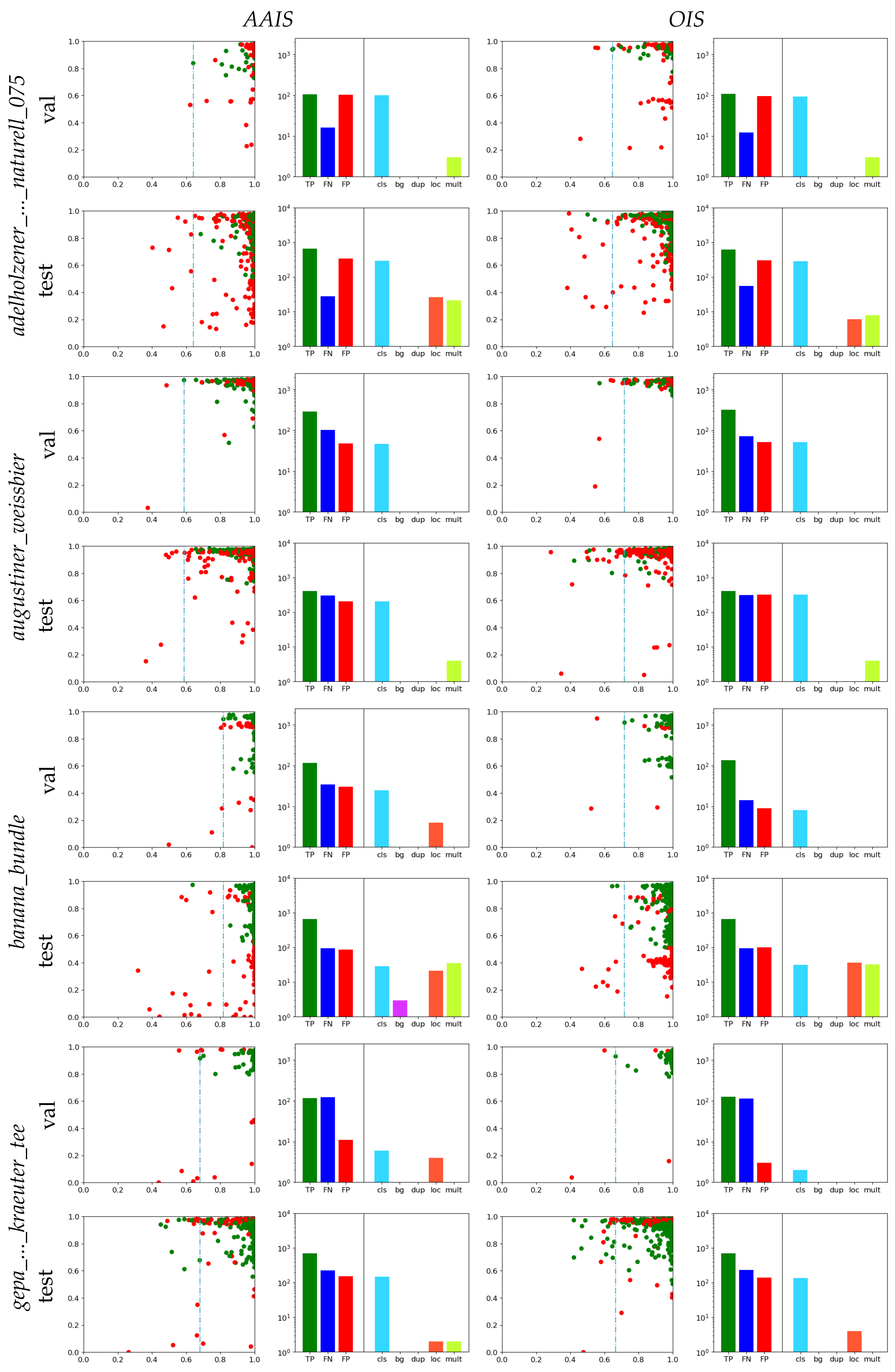


**Figure 10.12:** ***D2S* examples where validation results do not match the test set.** Score vs. IoU plots (*1st and 3rd columns*) and categories of predictions (*2nd and 4th column*). For some categories, the *OIS* advantage does not generalize to the test set.

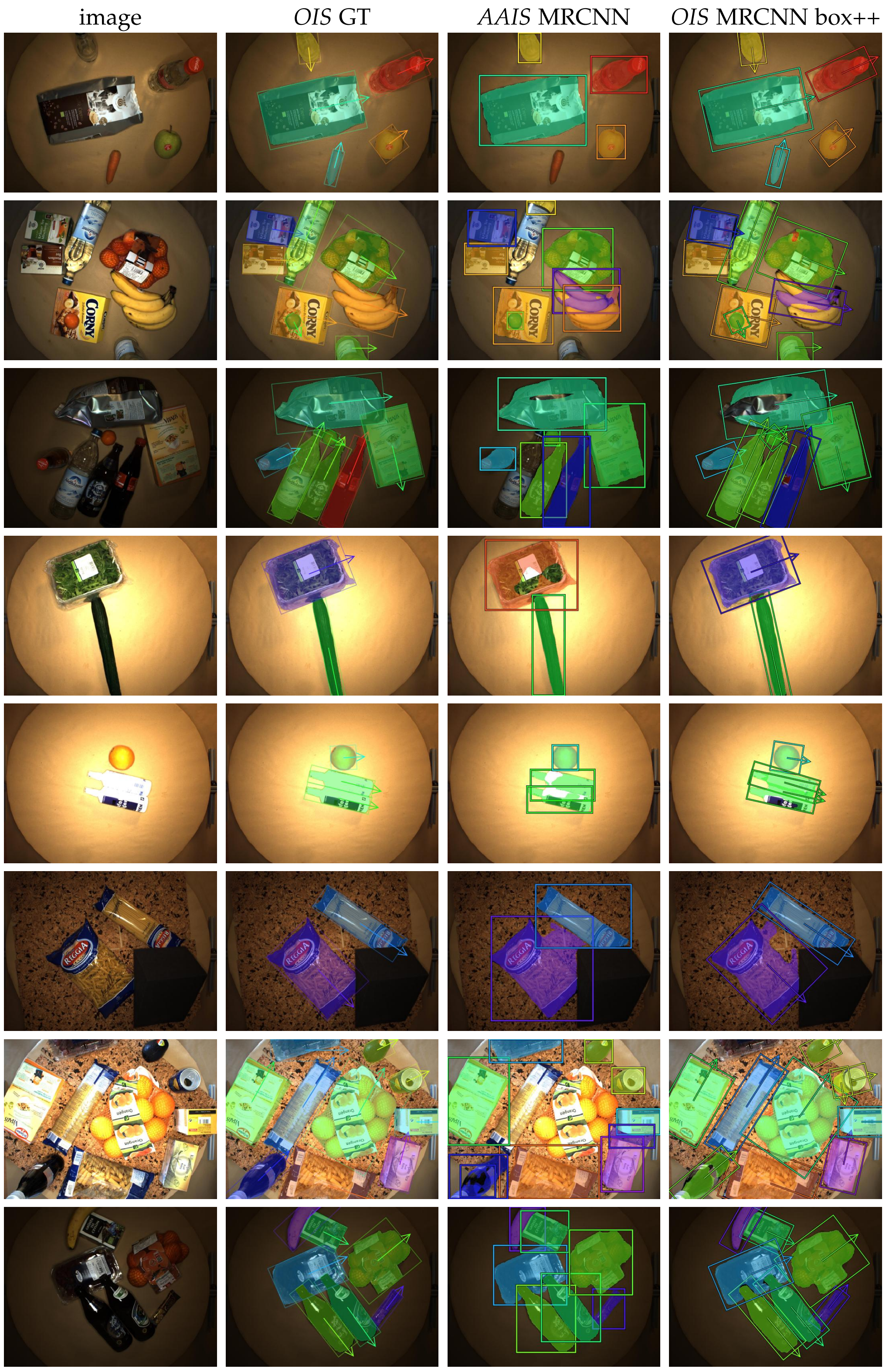


**Figure 10.13: Qualitative results on *D2S*.** Results for small images of the test set and with $T_s^\star(val)$. Generally, the model predictions of *AAIS* and *OIS* box++ are approximately on the same level. The models often struggle exactly with the same items due to reflections or oversaturated images (*3rd and 5th row*). The class *banana_bundle* is challenging because it is visually almost identical to the class *banana_single* (*2nd row*). Interestingly, in some cases, objects that seem to be very easy to find are not detected or the prediction is filtered out by $T_s^\star$ (*7th row*).

**Qualitative results.** We show some exemplary qualitative results from the *D2S* test set for *AAIS* MRCNN S and *OIS* MRCNN box++ S in Fig. 10.13. Of course, with very similar *AP* and *AP*$^\star$ values, also the mask results are very similar for most instances. There are objects that are only found by *OIS* and not by *AAIS* and vice-versa (*e.g., carrot in the 1st row, bottles in the 2nd and 3rd row, pasta, oranges, and grapes in the 7th row*). At first sight, the classification of the *OIS* model is slightly better than for *AAIS* (*rocket in the 4th row*), but this impression is not confirmed in the overall quantitative evaluation results. There are also many images where all objects are almost perfectly found (*8th row*). In those images, there are only subtle differences between the *OIS* and *AAIS* results, in particular for overlapping objects. Theoretically, *OIS* should have an advantage in these cases because the mask prediction resolution is higher to predict small holes and notches within the mask correctly. However, it is sometimes surprising how well the *AAIS* model can predict touching masks with the given resolution (*package of grapes in the 8th row*). Still, for both models oversaturated images (*5th row*) and objects that are clipped at the image boundary (*7th row*) are challenging. Moreover, if the box prediction fails, also the mask prediction is not meaningful (*bananas in the 2nd row or bottles in the 4th row*).

Both models are confused if there are reflections or sharp edges within the images (*coffee package at the top of the 3rd row*). Since the augmented training images of *D2S* contain many overlapping objects the models learn that holes within the masks are likely. To address this issue, in Chapter 11 we introduce a model that additionally predicts amodal masks and the occluded areas simultaneously with the modal masks shown here.

## 10.4 Conclusion

In this chapter, we presented an instance segmentation model that predicts instance masks based on oriented bounding boxes. The use of oriented boxes leads to more consistent mask targets than for axis-aligned bounding boxes with a higher foreground-to-background ratio. Moreover, the resolution of the mask prediction feature maps can be used more effectively, which results in very accurate mask predictions. We have shown that the overall mask *AP* and *AP*$^\star$ on *Pill Bags* and *Screws* can be improved significantly even if the underlying boxes are not very accurate. The predicted instance masks achieve better results, in particular at high IoU thresholds.

Moreover, with the use of oriented boxes, the instance segmentation task is better conditioned, such that fewer annotations are required. On *Pill bags*, only 10% of the instance mask annotations can be used to perform on the same level as the baseline that is based on axis-aligned boxes. Also on *Screws*, we have shown that the oriented instance segmentation model needs only approximately half the amount of generated training images with mask annotations to obtain competitive results with the axis-aligned baseline.

Also on *D2S*, where many categories do not have a unique orientation, the models based on oriented boxes are slightly better than the baseline. However, the significant improvement on the validation set does not transfer to the test set. In our extensive result analysis, we have shown that on *D2S*, for some categories a remarkable improvement can be achieved on the validation set, while on the test set the baseline is slightly better than our proposed approach. We believe that there are three main reasons for this issue: First, we need better fine-grained classification models that generalize well to samples at the boundary of the training data distribution. In particular, for various fine-grained bottle and tea categories, the predicted class sometimes seems to be randomly chosen for both the axis-aligned and oriented instance segmentation models, which makes it difficult to compare them. Second, more reliable confidences of the predictions would allow to calculate score thresholds that better generalize to the test set. And third, there is still much room for improvement concerning the oriented box prediction quality that serves as the basis of current instance segmentation models.

# 11 Amodal Instance Segmentation

The results of previous chapters have shown that current methods have large difficulties to predict accurate object masks whenever the objects are touching or overlapping each other. The idea of this chapter is to predict the instance mask beyond its visible parts and add awareness of the occluded parts of objects to the model. On the one hand, the occlusion prediction improves the localization accuracy. For example, it helps a grasping robot to decide which of the objects to take first. On the other hand, we believe that the prediction of occluded object parts allows to visualize better how the model is reasoning. Therefore, it also addresses the reliability of the model.

Semantic amodal segmentation is a recently proposed extension to instance segmentation that includes the prediction of the invisible region of each object instance. We present the first all-in-one end-to-end trainable model for semantic amodal segmentation that predicts the amodal instance masks as well as their visible and invisible part in a single forward pass.

In a detailed analysis, we provide experiments to show which architecture choices are beneficial for an all-in-one amodal segmentation model. On the *COCO amodal* dataset, our model outperforms the current baseline for amodal segmentation by a large margin. To further evaluate our model, we provide two new datasets with ground truth for semantic amodal segmentation, *D2S amodal* and *COCOA cls*. For both datasets, our model provides a strong baseline performance. Using special data augmentation techniques, we show that amodal segmentation is possible with reasonable performance, even without providing amodal training data. Parts of this chapter have been published in [48].

## 11.1 Introduction

Humans, with their strong visual system, have no difficulties reasoning about foreground and background objects in a two-dimensional image. At the same time, humans have the ability of *amodal perception*, i.e., to reason about the invisible, occluded parts of objects [89, 108]. Robots that should navigate in their environment and pick or place objects need to know if the objects are occluded or hidden by one or several other instances. This problem leads to the task of semantic amodal segmentation, i.e., the

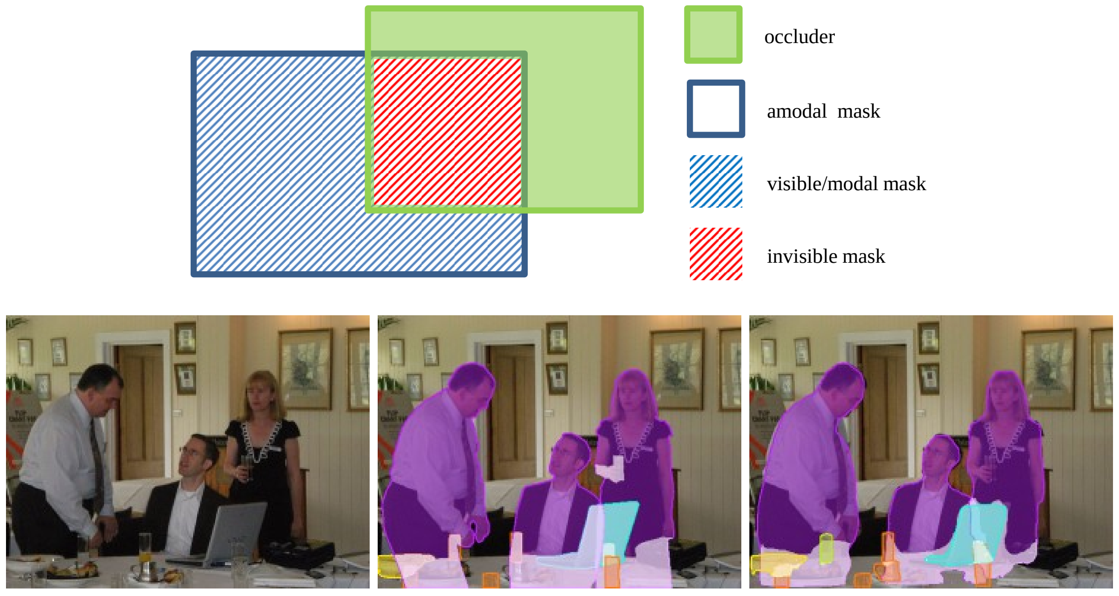


**Figure 11.1: Learning the invisible.** (*Top*) Explanation of different mask types; (*bottom, from left to right*) input image, ground-truth amodal instance annotations, predictions of our model: amodal instance masks and occlusion masks. The mask color encodes the object class and occlusion masks are highlighted in light color.

combination of segmenting each instance within an image by predicting its amodal mask and determining which parts of the segmented instances are occluded and what the corresponding occluder is. A typical example is shown in Fig. 11.1.

The amodal mask is defined as the union of the visible mask (which we will also refer to as modal mask) and the invisible occlusion mask of the object (cf. Fig. 11.1). Predicting amodal and visible masks simultaneously provides a deeper understanding of the scene. For example, it allows to calculate regions of occlusion and lets the robot know which objects have to be removed or in which direction to move in order to get free access to the object of interest.

Predicting the invisible part of an object is difficult: If the object is occluded by an object from another category, the model has no visual cues how to extend the visible mask into the occluded object part. There are generally no edges or other visual features that indicate the contour of the occluded object. In contrast, if the object is occluded by another instance of the same category, it is very hard for the model to judge where to stop expanding the mask into the occlusion part as the category-specific features are present all around.

We propose a model that can predict the visible, invisible, and amodal masks for each instance simultaneously without much additional computational effort. In summary, this chapter contains the following contributions:

- To the best of our knowledge, we are the first to propose an all-in-one end-to-end trainable multi-task model for semantic segmentation that simultaneously predicts amodal masks, visible masks, and occlusion masks for each object instance in a single forward pass.

- We provide the new semantic amodal segmentation dataset *D2S amodal*, which is based on *D2S* [46], with guaranteed annotation completeness and high-quality annotations. In comparison to the class-agnostic *COCO amodal* dataset [215], *D2S amodal* contains 60 different object categories and allows to predict amodal and occlusion masks class-specifically.

- By merging the categories of the modal *COCO* dataset with the instances of *COCO amodal*, we obtain the new amodal dataset *COCO amodal cls* with class labels.

- Our architecture ORCNN outperforms the current baseline on *COCO amodal* [215] and sets a strong baseline on *D2S amodal*. We provide extensive evaluations in order to compare different architectural choices.

- The training set of *D2S* allows to apply extensive data augmentation. This allows to train a semantic amodal method without any amodally annotated data. The model achieves competitive results on *D2S amodal*.

Note that throughout the chapter, we will call annotations containing only visible masks and models predicting visible masks *modal*, in contrast to *amodal* annotations and methods. We will also use the terms occlusion masks and invisible masks as synonyms.

## 11.2 Related Work

The topic of amodal perception has already been addressed in various fields of computer vision research.

**Semantic segmentation and 3D scene reconstruction.** Two tasks, for which amodal completion has already been used for some years, are semantic segmentation and 3D scene reconstruction. The task of semantic segmentation is to predict a category label for each pixel in an image. Semantic segmentation does not take different object instances into account, but returns a single region for each of the possible classes. Classes are often related to background or *stuff*, such as *sky, water, ground, wall, etc.* In [61], Guo and Hoiem describe a method to infer the entire region of occluded background surfaces. Their algorithm detects the occluding objects and fills their regions with the underlying or surrounding surface.

In 3D reconstruction, parts of the scene often cannot be reconstructed because of occlusions. Gupta et al. [63] combine depth information, superpixels, and hierarchical segmentations for amodal completion of semantic surfaces. Also Silberman et al. [175] address the problem of surface completion in the setting of a 2.5D sensor. They use a conditional random field in order to complete contours. The completed contours are subsequently used for surface completion.

In contrast to the above mentioned semantic segmentation methods, our work does not deal with the amodal completion of background regions or 3D object surfaces, but focuses on object instances in 2D images.

**Object detection.** In the context of object detection, Kar et al. [88] use a CNN to predict amodal bounding boxes of objects. By additionally estimating the depth of the bounding boxes and the focal length of the camera, object dimensions can be derived. However, neither the object mask nor the occluded part of the object is predicted.

**Instance segmentation.** More recent methods extend the object detection task to the more challenging instance segmentation task to predict the category and visible segmentation mask of each object instance in an image [116, 70, 115]. Yang et al. [203] propose a probabilistic model that uses the output of object detectors to predict instance shapes and their depth ordering. However, no occlusion regions are predicted. In [17], Chen et al. propose a graph-cut algorithm with occlusion handling in order to improve the quality of visible masks. However, they neither predict occlusion nor amodal masks.

**Amodal instance segmentation.** Research on amodal instance segmentation or semantic amodal segmentation has just started to emerge. Li and Malik [114] were the first to provide a method for amodal instance segmentation. They extend their instance segmentation approach [115] by iteratively enlarging the modal bounding box of an object into the directions of high heatmap values and recomputing the heatmap. Due to the lack of amodal instance segmentation ground truth, they use modally annotated data and data augmentation in order to train and evaluate their model.

In [215], Zhu et al. provide a new and pioneering dataset *COCO amodal* for amodal instance segmentation based on images from the original *COCO* [118] dataset. The authors did not restrict the annotations to the usual *COCO* classes and annotators could assign arbitrary names to the objects. Therefore, all objects in the dataset belong to a single class *object* and the variety of objects in this class is very large. Additionally, the authors provide annotations of background regions, which are sometimes extending to the full image domain, labeled as *stuff*. In order to provide a baseline, Zhu et al. use AmodalMask, which is the SharpMask [152] model trained on the amodal ground truth. The model suggests object candidates with a relatively high recall. However, the predictions of the model are class-agnostic. They also trained a ResNet-50 [68] to predict the foreground object given two input object masks and the corresponding image-patches.

In contrast to [114] and [215], our model is class-specific, end-to-end trainable, lightweight, and can predict amodal, visible, and invisible instance masks in a single forward pass.

## 11.3 Prediction of Amodal, Visible, and Invisible Masks

### 11.3.1 Architecture

We name our method *Occlusion R-CNN* (*ORCNN*) because the architecture is based on Mask R-CNN [70] (MRCNN). In the ORCNN architecture we extend MRCNN with additional heads for the prediction of amodal masks (amodal mask head) and the

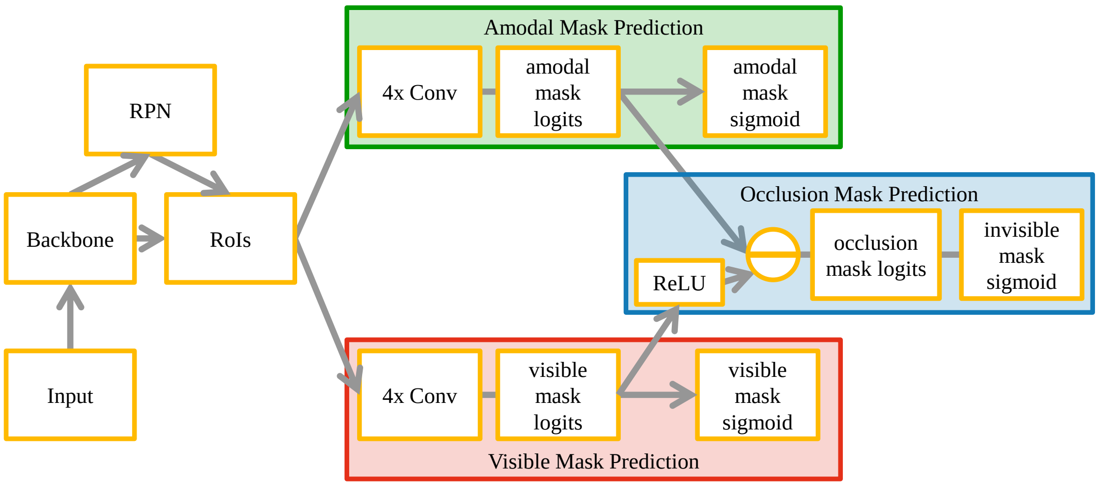


**Figure 11.2: ORCNN architecture.** ORCNN predicts the visible, amodal, and invisible masks simultaneously with its three respective branches. See text for a detailed explanation.

occlusion mask (occlusion mask head). An overview of the ORCNN architecture is shown in Fig. 11.2.

The visible mask head and the amodal mask head share the same architecture and use four $3 \times 3$ convolutions and ReLU layers to generate meaningful features for mask prediction. Their inputs are the extracted features from the RoI Align [70] layer. Note that during training and inference, the amodal and visible mask prediction heads of ORCNN share the same box proposals generated by the region proposal network (RPN). The target ground-truth masks of the RPN are the bounding boxes of the amodal instances. Therefore, the visible mask prediction head has to predict the visible mask of an instance from the amodal bounding box. This is a major difference to a modal model that is trained using the bounding boxes of the modal, i.e., visible masks of the instances.

A key component is that we link the modal and amodal mask heads with the occlusion mask head. The occlusion mask head essentially subtracts the visible from the amodal mask logits in order to obtain the occlusion mask logits. It is crucial to apply a ReLU operation on the visible mask logits before subtraction to avoid occlusion mask prediction for pixels where neither the amodal nor the modal mask are predicted.

Mounting both the modal as well as the amodal head on the same RoI feature extraction module leads to several advantages: First, this makes the additional amodal and occlusion mask prediction light-weight as only five additional convolution modules and two sigmoid layers are necessary. Second, using the same RoIs for the amodal and visible mask prediction guarantees that both predicted masks correspond to the same object prediction. In comparison, if one uses an ensemble of two separate models for amodal and visible mask predictions, it is not straightforward to fuse the results of these models. And third, by sharing the same architecture for the amodal and visible mask prediction head, we can initialize both heads with the same weights that have been pretrained on a large modal instance segmentation dataset, such as *COCO*.

### 11.3.2 Training

In order to obtain meaningful predictions for the visible, invisible, and occlusion masks, we have to formulate the corresponding losses for each of the tasks. As the tasks are similar and only the ground truth differs, we use a similar sigmoid-cross-entropy loss for all three mask types, first applying a per-pixel sigmoid and thereafter an average binary cross-entropy loss like in [70]. In combination with the losses for the class (*cls*) and bounding box (*box*), we obtain the total loss $L$:

$$L = L_{cls} + L_{box} + L_{AM} + L_{VM} + L_{IVM}, \tag{11.1}$$

where $AM$, $VM$, and $IVM$ are abbreviations for amodal, visible, and invisible mask, respectively. In theory, one of the three losses $L_{AM}, L_{VM}$, and $L_{IVM}$ is redundant because for the ground-truth masks it holds that $IVM = AM - VM$. Nevertheless, adding an additional loss for occlusion masks leads to amodal mask logits and visible mask logits that are on the same scale. Otherwise, consider the case that for a pixel we have high probability for the amodal and the visible mask. For example, let the logit activations be 14 and 10, respectively. This leads to a probability of $1/(1+e^{-(14-10)}) = 0.982$ for the occlusion mask to be present at this pixel, although occlusion should not be predicted.

In order to test the influence of adding the visible and invisible mask losses, we experiment with four different model variants: standard ORCNN is using a loss for each of the three mask types (amodal, visible, and invisible). For ORCNN (w/o $L_{IV}$) and ORCNN (w/o $L_V$), the loss for invisible or visible mask prediction is switched off. ORCNN (independent) is a model including $L_{IV}$ and $L_V$, but where the gradients with respect to $L_{IV}$ and $L_V$ are not propagated to the amodal mask head nor to the RoI feature extraction part.

As an alternative to including the invisible mask prediction into the model, we will also show results where the invisible masks are computed as the difference between the amodal and visible mask outputs. However, for ORCNN, the direct prediction of invisible masks comes at negligible cost.

### 11.3.3 Evaluation

To judge which model is the best for the task of amodal instance segmentation, we propose to extend the mean average precision ($AP$) and mean average recall ($AR$) evaluation measures commonly used for instance segmentation, e.g., on *Pascal VOC* [41] and *COCO* [118] benchmarks.[1] For brevity, in the following we will describe the extension of the measure only for the case of $AP$. The extension for $AR$ is straightforward. As in the *COCO* benchmark, we compute the final $AP$ by taking the mean of the per-category $AP$s.

In order to evaluate the individual tasks, we calculate the $AP$ values independently for amodal and visible masks to obtain $AP_A$ and $AP_V$, respectively. We can then include both of the masks into the definition of a true positive instance to obtain a combined $AP$

[1]Becaue $AP^\star$ was developed after the experiments of this chapter were done, we did not evaluate with $AP^\star$ here. The integration of $AP^\star$ into the evaluation of amodal instance segmentation models is left for future research.

measure $AP_{AV}$ (amodal-visible $AP$). For example, in order to obtain a true positive result in the $AP_{AV}$ setting, for a given IoU threshold $T_{\mathrm{IoU}}$, we need the correct predicted class and additionally, $\mathrm{IoU}(AM^G, AM^P) > T_{\mathrm{IoU}}$ and $\mathrm{IoU}(VM^G, VM^P) > T_{\mathrm{IoU}}$ both need to be satisfied. Here, $AM^G$ and $AM^P$ denote the amodal mask ground truth and the amodal mask prediction, respectively.

The invisible masks are included only indirectly into the overall measure $AP_{AV}$ due to the following issues: First, for non-occluded objects, the invisible mask is not present and it is not straightforward to define recall on something that is not present.[2] Second, for most objects in *COCO amodal* or *D2S amodal*, the invisible mask areas are rather small compared to the amodal or visible masks. Hence, small differences in invisible mask predictions have a large influence on the IoU.

To measure the quality of predicted occlusions, we do separate evaluations, where we ignore all non-occluded ground-truth objects. For these evaluations, we also calculate the average precision of invisible masks $AP_{IV}^{0.5}$ at IoU threshold 0.5. We use a low IoU threshold since the invisible masks are often very small and to take the difficulty of the task into account. For the calculation of $AP_{IV}$, we only use results where the amodal segmentation has an IoU higher than 0.5 with a ground-truth amodal mask of an occluded object. Hence, if the predicted amodal mask is a false positive, we ignore the corresponding invisible mask.

For models that do not predict any visible or invisible masks, we calculate the measures $AP_{AV}$ and $AP_V$ by using amodal masks also as predictions for the visible masks. When invisible masks are calculated as the difference of amodal and visible masks, we denote the corresponding measure $AP_{IV}$ *diff*.

## 11.4 Experiments

In the following, we compare our models to previous results on *COCO amodal* (*COCOA*) and set new benchmarks for the new semantic amodal datasets *COCO amodal cls* and *D2S amodal*. All models were trained using the Detectron [56] framework. More information on the settings can be found in Appendix E.

### 11.4.1 COCOA

*COCOA* [215] is the first dataset with ground truth for semantic amodal segmentation. The dataset consists of 2500 training, 1323 validation, and 1250 test images. In each image, most objects and background *stuff* areas are annotated with amodal masks. Occluded objects are additionally annotated with visible and invisible masks. All objects belong to a single category *object* and have an additional *stuff* label.

**Amodal mask prediction.** As a baseline result for amodal semantic segmentation, we train MRCNN with a ResNet-50 or ResNet-101 backbone on the amodal annotations.

[2] Generally, recall is increasing with more proposals. In case of an empty invisible mask, this is not the case, which prevents the usual way of computing the $AP$ measure.

| | **all** | | **things** | | **stuff** | |
|---|---|---|---|---|---|---|
| | $AP_A$ | $AR_A$ | $AP_A$ | $AR_A$ | $AP_A$ | $AR_A$ |
| AmodalMask [215] | 5.7 | 43.4 | 5.9 | 45.8 | 0.8 | **36.7** |
| ARCNN-50 | **29.9** | **45.8** | **33.2** | **50.4** | **5.8** | 33.0 |

**Table 11.1: Baseline results for *COCOA*.** Amodal mean average precision and average recall values for AmodalMask [215] compared to ARCNN.

We call these models *ARCNN-50/ARCNN*. Table 11.1 compares ARCNN to the baseline AmodalMask of [215] using their evaluation tool. ARCNN outperforms AmodalMask by a large margin in terms of average precision. AmodalMask achieves a high recall, since it always predicts 1000 regions for each image. Nevertheless, ARCNN achieves even higher recall while predicting only 30 results per image on average. An exception is the category *stuff*, since *stuff* regions are often very large areas in the background of the image and ARCNN predicts no object proposals for these areas.

In the following, our focus is on *things* because masks for *stuff* are hard to define and, therefore, the variance of annotations between different annotators is high. Thus, in *COCOA no stuff*, we exclude *stuff* annotations during training and evaluation. We found that for ARCNN and ORCNN, $AR$ is generally in line with $AP$ since recall is already captured within the $AP$ measure. Therefore, for the following evaluations we will just show $AP$ values. In order to highlight the performance on occluded objects, we also evaluate the architectures ignoring all non-occluded instances.

**Occlusion prediction.** As a baseline for a model like ORCNN that can predict amodal, visible, and invisible masks at the same time, we use ARCNN and standard MRCNN. Since both models only predict amodal or visible masks, respectively, in the evaluation, we use the amodal masks also as visible mask predictions and vice-versa.

To combine the benefits of MRCNN and ARCNN, we also use an ensemble approach by applying both models and merging the results (mergedAMRCNN). Therefore, we match the modal results to the amodal results greedily: We merge the modal result with highest IoU to the amodal result (if IoU $> 0.5$) if the predicted classes match. The score of a match is set to the mean of amodal and visible mask scores. If an amodal result is not matched, the amodal mask is used as visible mask and the invisible mask is set to an empty region. The best results are obtained if the unmatched modal results are ignored.

The results on *COCOA* are summarized in the top two sections of Table 11.2. The multi-task model ORCNN improves the quality of visible masks compared to ARCNN and sometimes even compared to MRCNN, while at the same time predicting occlusion masks. The best result for the combined measure $AP_{AV}$ is obtained using the variant ORCNN (independent). This variant combines the benefits of the ensemble mergedAM-RCNN into a single model with only a slight performance decrease for some of the measures. Standard ORCNN is the best choice for the prediction of invisible masks. Qualitative results for *COCOA* are shown in Appendix E.2.1.

| | all | | | occluded | | | | |
|---|---|---|---|---|---|---|---|---|
| dataset & model | $AP_{AV}$ | $AP_A$ | $AP_V$ | $AP_{AV}$ | $AP_A$ | $AP_V$ | $AP_{IV}^{0.5}$ | $AP_{IV}^{0.5}$ diff |
| *COCOA* | | | | | | | | |
| AmodalMask [215] | 3.7 | 5.7 | 4.8 | 1.2 | 3.2 | 2.6 | - | - |
| MRCNN [70] | 17.0 | 18.6 | 21.7 | 8.6 | 10.6 | 15.3 | - | - |
| ARCNN | 24.1 | **31.3** | 26.1 | 11.8 | **21.6** | 16.2 | - | - |
| mergedAMRCNN | 22.5 | 28.5 | 27.0 | 12.5 | 19.5 | 18.5 | - | 0.3 |
| ORCNN | 21.2 | 25.7 | 26.7 | 11.5 | 17.0 | 18.4 | **1.3** | **1.7** |
| ORCNN (independent) | **25.0** | 31.1 | **28.8** | **13.7** | **21.6** | **19.9** | 1.0 | 0.5 |
| *COCOA no stuff* | | | | | | | | |
| MRCNN [70] | 22.0 | 23.9 | 27.9 | 12.4 | 15.0 | 21.6 | - | - |
| ARCNN | 27.8 | **35.6** | 29.4 | 14.7 | **25.4** | 18.5 | - | - |
| mergedAMRCNN | 28.1 | 33.6 | **33.6** | **16.8** | 23.5 | **24.4** | - | 0.4 |
| ORCNN | 25.1 | 30.1 | 30.0 | 14.3 | 20.8 | 21.4 | **3.0** | **1.9** |
| ORCNN (independent) | **29.0** | 35.1 | 32.8 | 16.6 | 25.0 | 23.0 | 1.0 | 0.5 |
| *COCOA cls* | | | | | | | | |
| MRCNN [70] | 39.0 | 39.8 | 44.9 | 25.4 | 26.5 | 34.9 | - | - |
| ARCNN (agn) | 37.1 | 40.4 | 38.6 | 23.9 | 28.7 | 26.8 | - | - |
| ARCNN | 38.8 | 41.7 | 40.5 | 24.9 | 29.2 | 28.0 | - | - |
| mergedAMRCNN | **40.1** | **42.5** | **45.7** | **27.7** | **30.0** | **34.9** | - | 1.0 |
| ORCNN (agn) | 34.3 | 36.2 | 39.3 | 23.1 | 25.0 | 29.5 | 1.8 | 1.4 |
| ORCNN | 35.1 | 37.6 | 39.4 | 23.7 | 25.8 | 29.9 | **2.0** | 1.4 |
| ORCNN (independent) | 38.0 | 40.7 | 41.0 | 26.0 | 28.9 | 30.5 | 0.2 | 1.0 |

**Table 11.2: *COCOA* results.** Note that only ORCNN is predicting visible and invisible masks in addition to the amodal masks. For all other models, the predicted amodal mask was used for the evaluation of *AV* and *V* measures. Models marked with (agn) are using class-agnostic mask prediction heads.

### 11.4.2 COCOA cls

For amodal completion, the model has to get some intuition about the common shape of objects. We evaluate whether the prediction of amodal masks is a class-specific task. Therefore, we generated a new dataset *COCOA cls* by merging the usual *COCO* 2014 annotations with the *COCOA* dataset. *COCOA* contains many objects of categories (e.g., sandals, sneakers, or stuff categories) that are not part of *COCO*. Although each object in *COCOA* has a name tag, the annotators were free to choose a name. Furthermore, not all objects present in the ground truth of *COCO* have been annotated in *COCOA*. Therefore, to assign a class label to the objects in *COCOA*, we calculate the IoUs of the visible masks with the masks given for the corresponding image-id in *COCO*. Only annotations, for which the IoU between visible mask and any *COCO* annotation exceeds a threshold of 0.75 (and not labeled as stuff or crowd), were kept for *COCOA cls*. Overall, *COCOA cls* has 3501 images with 10 592 objects compared to the 3823 images and 34 916 objects of *COCOA*. Note that using this merging scheme, some *COCO* classes, e.g., *hairdryer*, are not present in the training set of *COCOA cls*. Furthermore, for many images not all *COCO* annotations can be matched to a corresponding *COCOA* label.

As shown in Table 11.2, models perform much better on *COCOA cls* than on *COCOA*.

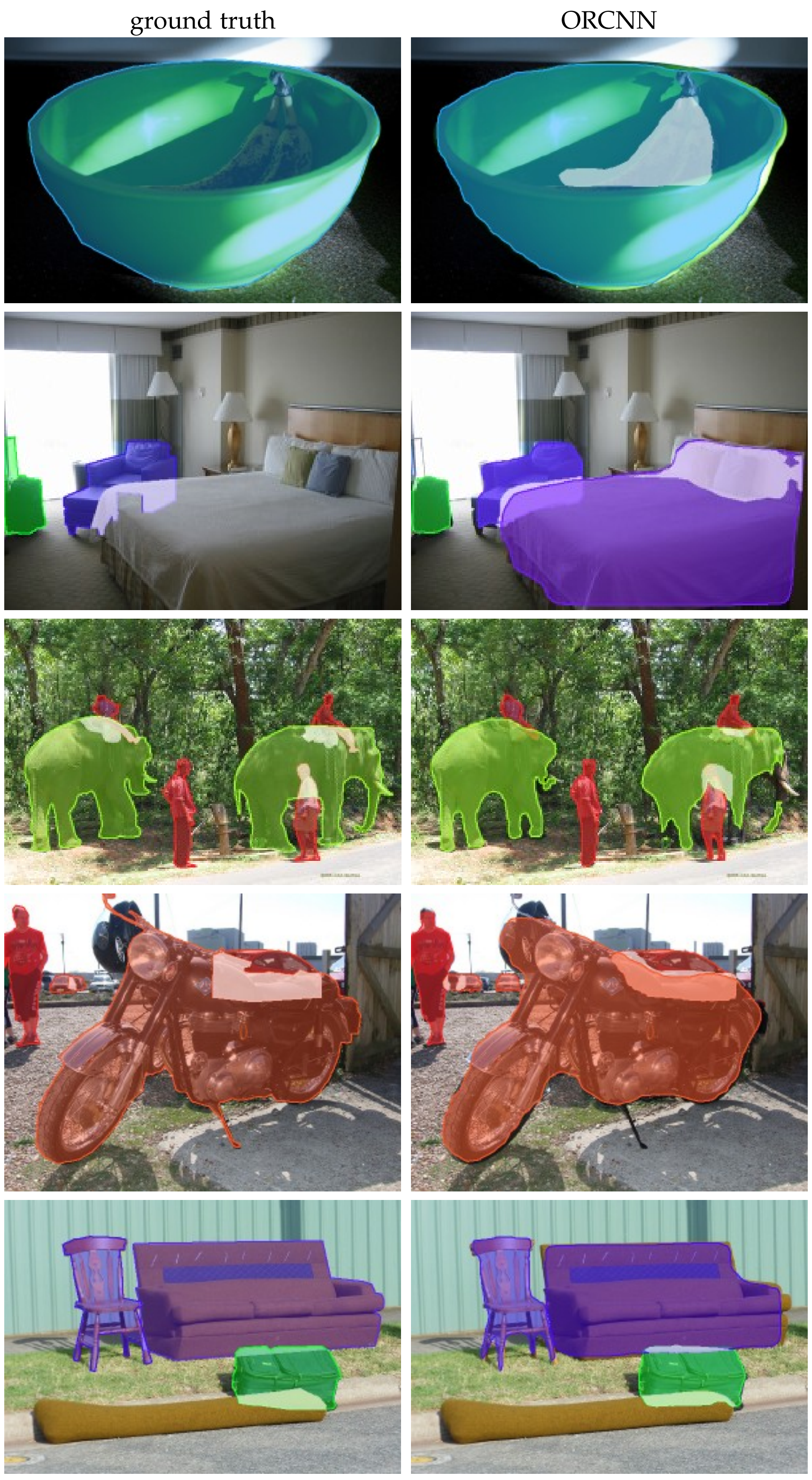


**Figure 11.3: *COCOA cls* results.** (*Left*) Ground truth annotations for images of the validation set; (*right*) exemplary qualitative results of our ORCNN model trained on the *train* split. Predicted invisible masks are indicated by a white overlay. Note that ORCNN sometimes predicts instances or invisible masks correctly that are not part of the ground truth. These count as false positives in the evaluation or at least lead to reduced $AP_{\text{AV}}$ and $AP_{\text{V}}$ values.

On *COCOA cls*, the ensemble mergedAMRCNN is the best choice, slightly increasing the performance compared to the multi-task model ORCNN (independent). The ensemble is in favor, as it averages the scores of both predictions: Only if both models are confident, the final score is high. Also, false positive predictions of MRCNN are filtered out in mergedAMRCNN if there is no corresponding false positive of ARCNN. This reduces the total number of false positives.

In order to see if class-specific mask prediction improves the results, ARCNN and ORCNN were also trained using class-agnostic mask proposals. On *COCOA cls*, using class-specific mask proposals helps for almost all measures both in the case of ARCNN and ORCNN. For occluded objects, class-agnostic ORCNN achieves the best average precisions in terms of invisible masks.

Generally, the interpretation of *AP* values for *COCOA* and *COCOA cls* results is difficult because for both datasets the annotations are incomplete. For example, as shown in Fig. 11.3, there are some cases where ORCNN makes correct predictions but the ground truth does not contain the corresponding annotations.

### 11.4.3 D2S amodal

Since for *D2S* (Chapter 5), only minor occlusions are present in the training set, the generation of reasonable artificial images can be done by cropping instances from the training set and pasting them onto an empty background (cf. Chapters 5 and 7). We annotated all images of *D2S* amodally to obtain *D2S amodal*. The annotations contain the category, amodal mask, and additional visible and invisible masks for occluded objects. For images where the amodal masks are reaching out of the image boundary, a zero padding is used such that all amodal masks are fully contained in the image.

**Splits.** Because the *D2S amodal* training split only contains minor occlusions, it is not suitable to train semantic amodal models that predict occlusion masks also for moderately to heavily occluded objects. Hence, we use data augmentation similar to the data augmentation described in Chapters 5 and 7. *D2S amodal augmented* consists of 1562 augmented images, where only objects from *D2S train* that do not reach out of the image boundary have been used for augmentation. The *D2S amodal train* set is then the combination of *D2S amodal train rot0* with *D2S amodal augmented* splits resulting in a total of 2000 images.

To evaluate if an amodal model can be trained only from modal annotated data, we augmented another 2000 images in the same way, but using the modal annotations from *D2S train* to obtain *D2S amodal modal augmented*. For all splits of *D2S amodal*, the statistics are given in Table 11.3. As is common for *D2S*, the number of images containing occlusion is much higher for the *val* set compared to the *train rot0* set. By adding the *augmented* set to *train rot0*, the resulting *amodal train* split contains many objects which have a higher frequency of occlusion and a higher average per-object occlusion rate than for the *val* and *test* splits. Exemplary ground truth annotations are shown in Fig. 11.4.

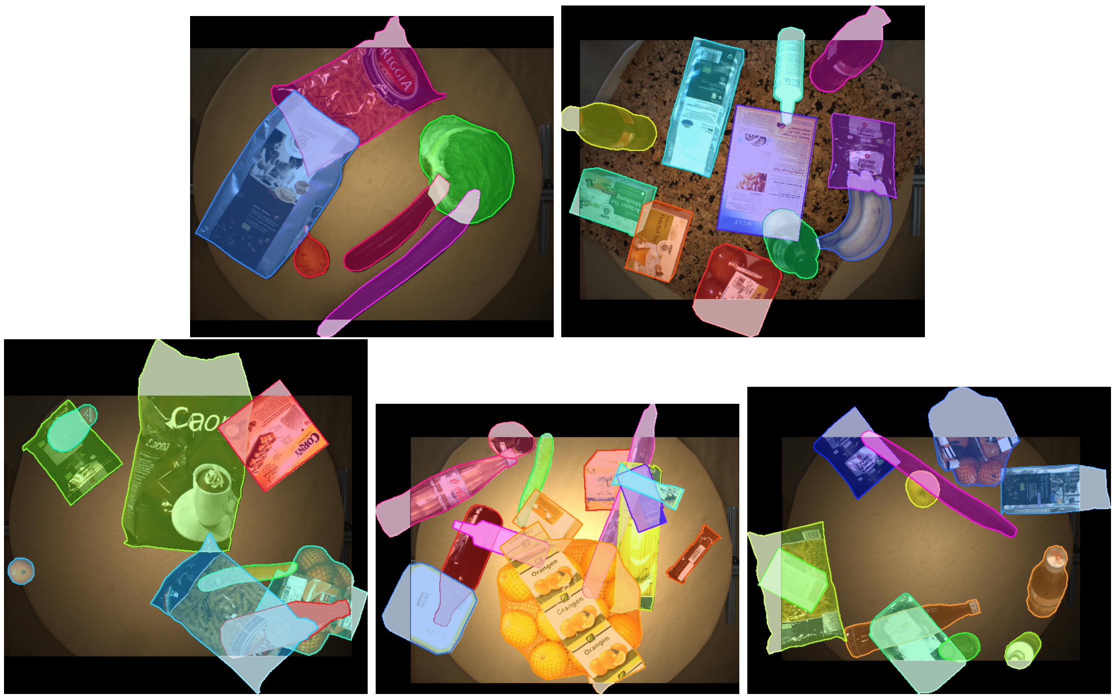


**Figure 11.4: *D2S amodal* ground truth.** Exemplary ground truth annotations for different splits of the *D2S amodal* dataset. Two images with different backgrounds from *val* (*top*), two generated images from *augmented*, and an image generated from modal annotations *modal augmented* (*bottom, from left to right*).

**Results.** The qualitative results of ORCNN shown in Fig. 11.5 are very promising. The model predicts occlusions correctly in many cases, especially for objects lying completely on top of another one or objects reaching out of the image boundary. This shows that the model is able to learn the common shape of object classes.

The quantitative analysis given in Table 11.4 shows that the prediction of invisible masks is very difficult. On the one hand, if the ground-truth invisible masks are small, small differences of the invisible mask proposal already lead to a low IoU value. On the other hand, if the ground-truth invisible mask is large, it is very difficult for the model to generate the correct shape of the invisible mask prediciton, again leading to a low IoU value. In both cases, if the IoU value is below 0.5, this leads to a false positive prediction and in most cases to a false negative as the corresponding ground truth is not matched.

On *D2S amodal*, ORCNN (independent) achieves comparable results to mergedAMR-CNN and even outperforms the ensemble with respect to $AP_{AV}$ on occluded objects. For invisible mask prediction, standard ORCNN is the best choice. Interestingly, ORCNN achieves a higher $AP_{IV}^{0.5}$ when the invisible mask is predicted directly instead of using the difference between amodal and visible mask ($AP_{IV}^{0.5}$diff).

Some failure cases of ORCNN on *D2S amodal* are shown in Fig. 11.6. False positive occlusion predictions are often caused by reflections or lighting changes. When objects are lying next to each other and touching, the amodal and invisible masks are sometimes extended into neighboring instances. For other cases, occlusions are not predicted at all.

In summary, ORCNN (independent) yields the best compromise for the prediction

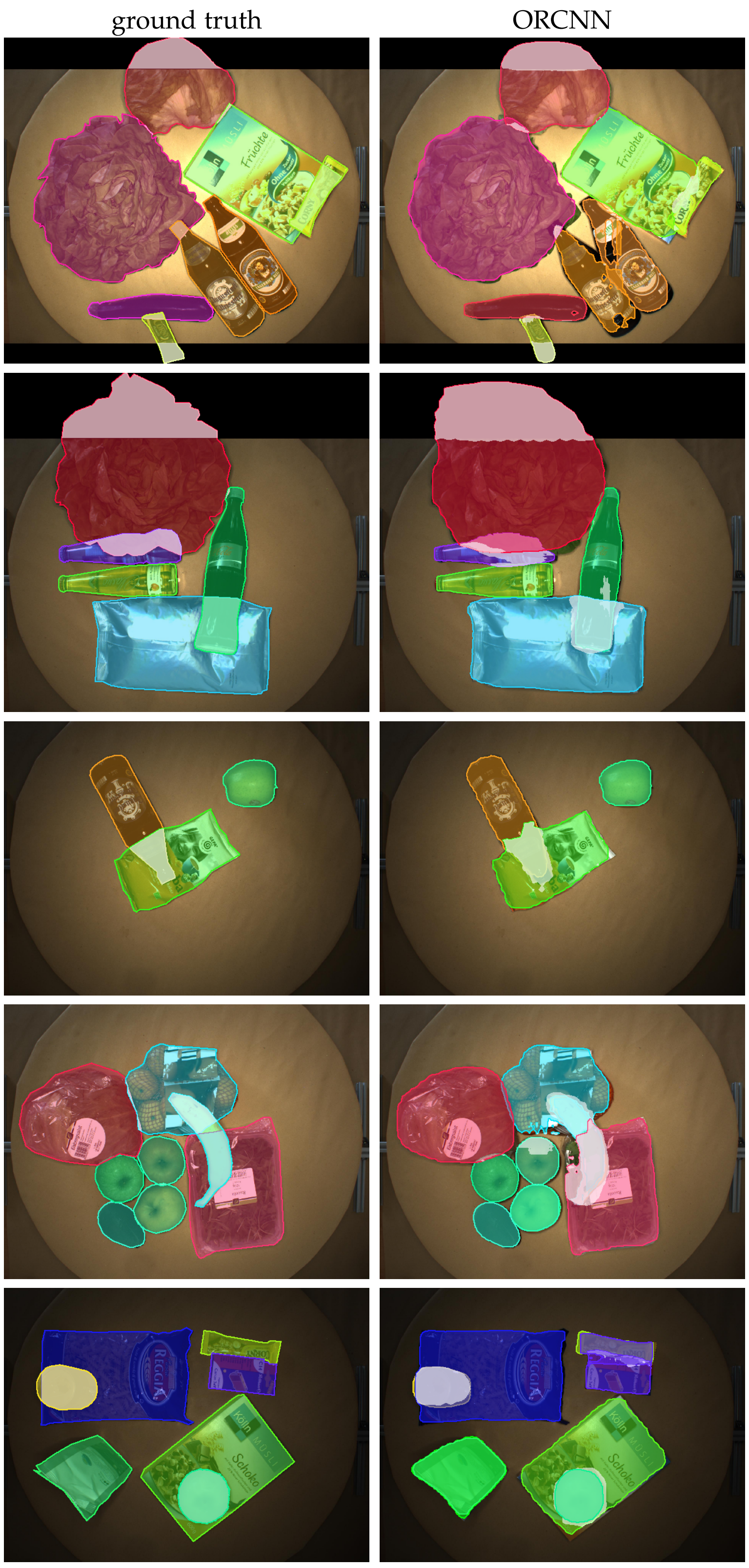


**Figure 11.5: *D2S amodal* results.** (*Left*) Ground truth annotations for images of the *val* set; (*right*) exemplary qualitative results of our ORCNN model trained on the *amodal train* split. Predicted invisible masks are indicated by a white overlap. In most cases, the results are promising, especially for objects lying on top of each other or reaching beyond the original image boundary. More results are in Appendix E.2.2.

| split name | *augmented* | *train rot0* | *train* | *val* | *modal augmented* |
|---|---|---|---|---|---|
| num imgs | 1562 | 438 | 2000 | 3600 | 2000 |
| num imgs w/ occl | 1507 | 57 | 1564 | 2520 | 1930 |
| img OR [%] | 96 | 13 | 78 | 70 | 96 |
| num objs | 12 376 | 690 | 13 066 | 15 654 | 15 851 |
| num objs occl | 8798 | 66 | 8864 | 7473 | 11 302 |
| obj OR [%] | 71 | 9 | 68 | 47 | 71 |
| avg OR / reg (all) [%] | 23 | 0 | 22 | 8 | 23 |
| avg OR / reg (occl) [%] | 33 | 4 | 33 | 18 | 33 |

**Table 11.3: *D2S amodal* splits.** Image and occlusion statistics for the splits used with *D2S amodal* (*from top to bottom*): number of images, number of images with at least one occluded object, rate of images that contain any occlusion, total number of objects, number of occluded objects, rate of objects that are occluded, average occlusion rate per object region for all objects, average occlusion rate per object only for occluded objects.

| | all | | | occluded | | | | |
|---|---|---|---|---|---|---|---|---|
| dataset & model | $AP_{AV}$ | $AP_A$ | $AP_V$ | $AP_{AV}$ | $AP_A$ | $AP_V$ | $AP_{IV}^{0.5}$ | $AP_{IV}^{0.5}$ diff |
| ARCNN (agn) | 63.4 | 72.2 | 64.8 | 48.0 | 62.2 | 50.8 | - | - |
| ARCNN | 63.8 | 72.6 | 65.3 | 48.7 | **63.0** | 51.6 | - | - |
| MRCNN (agn) | 64.7 | 65.2 | **77.7** | 48.7 | 49.3 | **71.3** | - | - |
| mergedAMRCNN (agn) | **69.2** | **72.7** | 75.5 | 58.2 | 62.8 | 68.2 | - | 13.2 |
| ORCNN (w/o $L_{IV}$) | 34.0 | 68.6 | 34.7 | 26.8 | 58.9 | 28.0 | 0.1 | 0.0 |
| ORCNN (w/o $L_V$) | 11.0 | 66.1 | 11.1 | 7.5 | 56.7 | 7.6 | 5.8 | 0.0 |
| ORCNN (agn) | 62.3 | 65.2 | 71.0 | 52.5 | 55.2 | 65.3 | **14.7** | **13.8** |
| ORCNN | 58.9 | 62.1 | 66.2 | 48.1 | 51.2 | 58.9 | 8.7 | 8.9 |
| ORCNN (independent, agn) | **69.2** | 72.3 | 74.3 | **59.1** | 62.5 | 67.6 | 0.8 | 9.2 |
| ORCNN (modal aug) | 56.3 | 59.9 | 62.6 | 47.9 | 51.0 | 57.6 | 7.8 | - |

**Table 11.4: *D2S amodal* results.** Models marked with (agn) are using class-agnostic mask prediction heads. ORCNN (modal aug) is a ORCNN only trained on *D2S amodal modal augmented*. All values are calculated for the *D2S amodal val* set.

of visible and invisible masks at the same time (highest $AP_{AV}$ and $AP_V$). This comes at the cost of a slightly lower $AP_V$ value compared to a model that only predicts visible masks, like MRCNN. Note that compared to ORCNN, mergedAMRCNN has a much higher memory consumption ($2 \times 7600$ MB vs. 8300 MB), as well as significantly longer inference runtime ($2 \times 170$ ms vs. 180 ms, nvidia GTX1080Ti GPU). Furthermore, there is an additional overhead for the merging strategy ($\approx 10$ ms), which depends on the number of output results. In contrast to *COCOA*, for *D2S amodal*, ORCNN (agn) outperforms the ORCNN model that predicts a class-specific mask.

Table 11.5 also shows the result of ORCNN when training only on artificially augmented data obtained from *D2S train*, *D2S amodal modal augmented*. The model performs only slightly worse than ORCNN trained on *D2S amodal train*. Thus, it is possible to train a competitive model without the need of amodally annotated data.

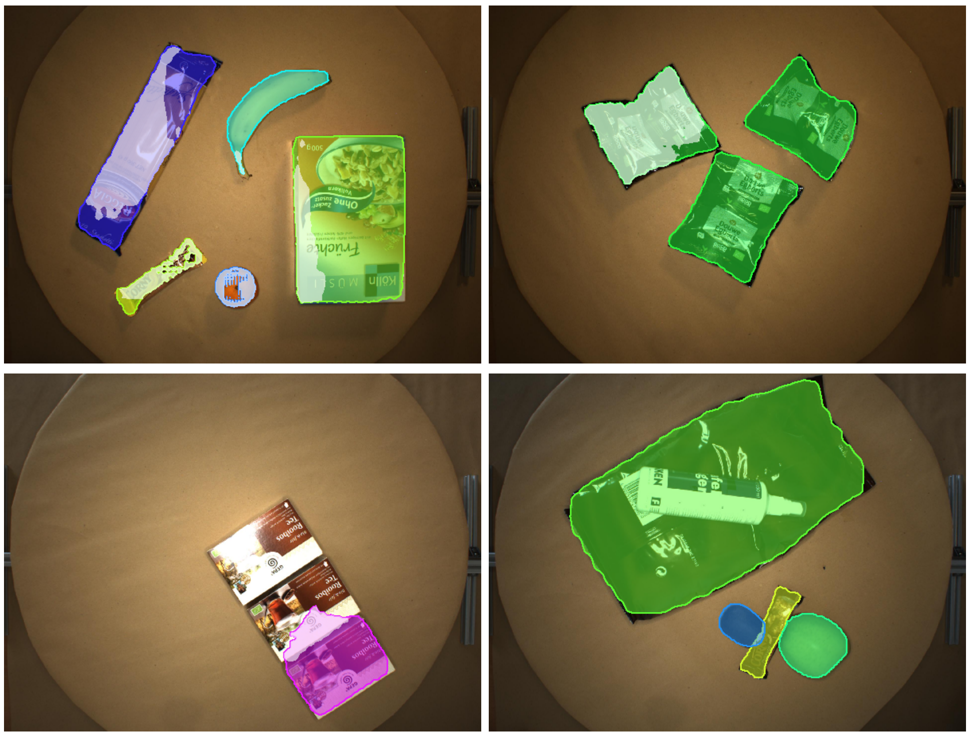


**Figure 11.6: *D2S* failure cases.** Exemplary qualitative results where our ORCNN model fails. Predicted invisible masks are indicated by a white overlap. *(Top)* occlusion is falsely predicted, possibly due to reflections or lighting changes; *(bottom-left)* the amodal and invisible mask of this instance is extended into the neighboring object of the same class (only this instance is visualized); *(bottom-right)* the occlusion caused by the plastic bottle is not detected.

## 11.5 Conclusion

We proposed an end-to-end trainable, instance-aware model for semantic amodal segmentation. Our model, ORCNN, simultaneously predicts amodal, visible, and invisible masks, and the category of each instance in a single forward pass. By merging annotations of *COCO* with *COCOA*, we obtain a category-specific semantic amodal dataset based on *COCO* images: *COCOA cls*. We provide semantic amodal ground truth for *D2S* splits resulting in *D2S amodal*. ORCNN (independent) outperforms previous work on *COCOA* and sets strong benchmarks on *COCOA cls* as well as *D2S amodal*. The light-weight, all-in-one model is able to achieve comparable performance to an ensemble approach. Furthermore, we show that it is possible to train a competitive amodal model only using modal annotations and data augmentation. Our experiments and results show that it is possible to predict the invisible masks of occluded objects even in areas without any visual cue. Thus, our model can indeed *learn to see the invisible*.

The results show that the prediction of amodal and in particular invisible masks is a difficult task that needs further research to reduce the number of false positive predictions of occlusions. Once the occlusion predictions are reliable, they can help a grasping robot to navigate first to the objects without occlusions which leads to a more efficient application.

# 12 Conclusion

This chapter summarizes the contributions of this work and potential directions for future research in this area.

## 12.1 Summary

In this work, we looked at the application of DL-based methods and, in particular, CNNs to solve industrial classification, object detection, and instance segmentation problems. In this regard, we evaluated our proposed ideas with respect to the requirements of successful industrial algorithms as stated in Chapter 1: localization accuracy, fine-grained classification, robustness, reliability and uncertainty estimation, ease of use, and scalability. The focus was on the ease of use, localization accuracy, and fine-grained classification requirements as the central research question of this thesis was: Is it possible to build methods that learn from few annotated samples, but localize and classify objects accurately?

To answer this question, first we presented a thorough introduction to the use of machine learning methods in general and to modern classification CNNs in Chapter 2.

To reduce the number of necessary training samples, in Chapter 3 we built a novel rotational convolution and pooling module for classification CNNs. We showed that the module extracts approximately rotationally invariant features from images and thus, it allows to train the network only with a single orientation of each class. Still, the network is capable to classify the classes correctly if they occur in an arbitrarily rotated version. On this task, we could significantly outperform related work that relies on training data that contains all possible orientations of the objects as they are shown within the validation and test sets. Hence, our model certainly improves the ease of use of CNNs, as the user does not have to think about the possible orientations in which the different categories might occur.

However, we saw that creating features that are invariant to certain transformations reduces their expressiveness. For example, our proposed model leads to inferior results compared to rotationally equivariant models on MNIST-rot36 when being trained on rotated inputs. We have to admit that in practice, if rotated variants of the inputs are to be expected within the application, the easier and better performing choice is to simply rotate the input data.

From Chapter 4 onwards, we turned our attention towards the more complex task of object detection and instance segmentation. To get a better understanding of the models that we extended in subsequent chapters, we investigated and compared the design of recent object detection and instance segmentation models in detail in Chapter 4.

To be able to evaluate the above mentioned criteria for algorithms in industrial applications, we started by introducing the novel *D2S* dataset on which industrial methods can be evaluated properly. The dataset's design encourages the use and development of few-shot detection methods that meet the ease of use requirement. Further, with its high-quality annotations, clutter objects, and severe occlusions, it is suitable to evaluate the localization accuracy and robustness of methods. The small inter-class variations between some of the bottle and tea categories pose a challenge with respect to fine-grained classification.

Our extensive experiments and failure case analysis of the current models revealed that the localization accuracy requirement can only be met for rather easy images where objects are positioned on a familiar background and without overlaps. But even in this setting, fine-grained classification is a major challenge for the models because they frequently confuse classes with similar appearance, such as the different sorts of tea boxes in *D2S*. The robustness requirement is also not met satisfyingly: unknown backgrounds, clutter objects, reflections, or objects in unknown views lead to a high number of false positives. This is directly related to the fact that current models estimate their uncertainty badly and are not reliable. A huge portion of the false positives is predicted with a high confidence such that it is not possible to filter out false positives based on their score. Moreover, it is hard to tell in which cases the model is unsure about its predictions. The failure cases that we identified in Chapter 5 were addressed in the following chapters.

In Chapter 6, we investigated the predominantly used evaluation measure for object detection and instance segmentation methods — the average precision ($AP$). We found several issues in the current computation of the measure and, more importantly, that a majority of the false positives that a model might predict is not penalized by $AP$. In order to better include the reliability and uncertainty estimation into the evaluation and ranking of models, we proposed a novel $AP^\star$ measure that includes the actual model precision into the evaluation. In our extensive experiments that re-evaluated the *D2S* baselines from Chapter 5, we showed that using $AP^\star$, indeed the ranking of the models changes. Moreover, $AP^\star$ allows to compute optimal class-specific score thresholds that suppress a large amount of false positives. For example, with these, the number of false positives on the *D2S* test set could be drastically reduced from 104 633 to 5567 and from 74 100 to 5513 for RetinaNet and RetinaMask, respectively.

We came back to the ease of use requirement and the reduction of the amount of hand-labeled images in Chapter 7. Instead of changing the model, as was done in Chapter 3, here the idea was to change the training data. We proposed to use artificially generated training images instead of real and manually annotated images. Moreover, we introduced a simple, time- and cost-efficient pipeline for the training of instance segmentation methods in a weakly supervised setting. This allowed to quickly generate large training sets for instance segmentation methods without a tedious annotation process and obtain competitive results. Moreover, we showed that specific augmentations of the generated images can be used to further address certain failure cases, such as touching objects or reflections. With these specific augmentations, the baselines from Chapter 5 could be improved by over 3 pp $AP^{\star}$.

Chapter 8 presented the concept of oriented box detection. Generally, oriented boxes better approximate the shape of objects in comparison to axis-aligned boxes. For example, an oriented box prediction allows to grasp an object in the correct direction and align it with others. Since oriented boxes are generally tighter around the object than axis-aligned boxes and thus, they contain less noise from the background or neighboring objects, we proposed that oriented boxes could also improve the fine-grained classification. We showed the necessary steps to change detection models to predict oriented boxes instead of axis-aligned ones. Moreover, we introduced the idea of classes without orientation that allows to apply oriented box detection in applications where only a subset of the classes has a clearly defined orientation. Further, we provided insights how axis-aligned and oriented boxes can be compared using an evaluation based on the relative bounding box IoU — $\text{rIoU}_{\text{BB}}$. Our experiments on *Screws*, *Pill Bags*, and *D2S* revealed that accurate oriented box detection is challenging and mainly improves the localization of objects with a well defined orientation. However, on *D2S*, the hypothesis that oriented detection also improves the classification had to be rejected.

In Chapter 9, we compared the DL-based oriented detection method of Chapter 8 to Shape-Based Matching, a classic, rule-based approach that is known to be accurate and efficient. Moreover, we explored the limits of reducing the manually labeled annotations and only used a single template image to generate the training set of the DL-based method. The comparison revealed that both methods have their strength and weaknesses. While Shape-Based Matching predicts much more accurate results, it also has more false positives in the background and requires some manual tuning of the model hyperparameters. On the contrary, the DL-based model is easy to setup — at least if enough annotations exist — and its runtime scales better with an increased number of object classes. To get the best of both worlds, we presented a novel method that fuses both approaches in a hybrid model: the DL-based model is used to restrict the search space of the Shape-Based Matching and for some categories, where it performs better or equally well, we can directly use the DL results. In the second stage, we apply Shape-Based Matching that is now significantly faster due to the restricted search space, but still predicts very accurate results and returns more reliable scores.

In Chapter 10, we presented a novel instance segmentation method based on oriented box prediction that we had developed in Chapter 8. We saw that the model has the advantage that the comparably low resolution of the mask prediction head is used more effectively and that mask targets become invariant to the orientations of the objects within the image. In our experiments on *Screws* and *Pill Bags*, we showed that this makes the instance mask prediction better conditioned and thus, fewer of the expensive mask annotations are necessary to obtain very accurate results. For example, on *Pill Bags*, the oriented instance segmentation model could outperform the axis-aligned baseline with only 30% of the objects labeled with instance masks. Moreover, on *Screws*, with only half the amount of generated images with mask annotations, the oriented instance segmentation method performed on the same level as its axis-aligned baseline. Also on *D2S*, where many of the categories do not have a uniquely defined orientation, our proposed method could outperform the baseline.

Next to the difficult fine-grained classification problems, severe occlusions are one of the main remaining sources of failure on *D2S*. Hence, in Chapter 11, we built a novel model that directly predicts the visible, invisible, and the amodal instance mask of each object within an image simultaneously. In our experiments on *COCO* amodal (*COCOA*) and *COCOA cls*, our model could significantly outperform the previous baselines on this very challenging task. To this end, we also presented novel measures to evaluate such methods. Further, we extended *D2S* with amodal annotations and setup first baselines for amodal instance segmentation. While previous work only evaluated the recall of their methods, our evaluations revealed that there is much room for improvement, in particular, the precision of existing models must be increased.

Overall, our proposed models and data generation methods make object detection and instance segmentation better usable in industrial applications. The presented algorithms certainly improve the ease of use aspect of current methods since they significantly reduce the amount of necessary hand-labeled instance mask annotations. This saves the user time directly at the beginning of the model setup and reduces costs. Moreover, the presented data augmentation techniques allow to adjust the training set very quickly for new classes or classes that have a different appearance due to a product line change. We showed that our models based on oriented boxes can be either used in a hybrid approach with, e.g., a Shape-Based Matching algorithm, or can be used to improve the accuracy of mask predictions with few annotated data. Hence, we can say *yes*, it is indeed possible to get high-quality results with only a few annotated training samples.

## 12.2 Future Work

Our work can be considered as one of many steps that need to be taken such that current object detection and instance segmentation can be seamlessly integrated into industrial applications.

With *D2S*, we provide a high-quality dataset for one application scenario in the context of a supermarket or warehouse setting, but there are many more application areas that could be addressed. For example, there are cases where the product portfolio consists of several thousands of categories and where the images are not colored, but in grayscale. Other application areas include agriculture, electric components, or pharmaceuticals, where different categories are often only defined based on subtle differences to others. Generally, in quality inspection systems defect detection and segmentation play an important role. The segmentations allows to compute the area of the erroneous region and thus, it can be decided if the defect is severe or not. There is always a need for high-quality datasets in industrial domains that are still underrepresented in today's computer vision research.

In our extensive evaluations within this thesis, we saw that a major challenge for current methods is fine-grained classification. For example, in an object counting task the results are only of limited use if we cannot rely on the class predictions because they are frequently confused with similar categories. Therefore, in future work, ideas for fine-grained classification should be brought to detection methods, such as hierarchical classification approaches that make use of supercategories.

In our opinion, the robustness requirement is directly related to the uncertainty estimation. Of course, in a perfect world, our models should also predict correct results for objects with an uncommon appearance. However, as we have seen in many of the failure cases throughout this work, mostly for clutter objects or objects with an uncommon appearance, the models predict wrong results. Hence, for the state of current models it would already be a large gain if those predictions could be clearly identified as 'unsure' predictions, indicated by lower confidences.

Another topic, where research is still in its infancy, is occlusion prediction or amodal instance segmentation. Although for some cases the results of our proposed model in Chapter 11 are impressive, there is still a huge room for improvement. Predicting the occlusions reliably and accurately would be a large improvement, e.g., for bin-picking applications: the robot could grasp objects in a meaningful order and thus, avoid losing the objects if they are covered by others. This would lead to a much more efficient application and save costs.

Within this thesis, we proposed two different ways to reduce the amount of labeled images that is necessary to train a model successfully. The first approach was to change the models themselves such that they require fewer training samples. The second way changed the data that was artificially generated including pixel-precise instance segmentation masks. Concerning the first, it remains an open challenge to build few-shot models that can learn from few data, but are still powerful enough to generalize to a large variety of different test cases. Concerning the second, our experiments on *D2S* have shown that training in a weakly-supervised setting, without a single hand-labeled segmentation, the models are competitive, but still lag behind the fully supervised baselines by about 11 percentage points $AP$. Possible reasons are that the automatically generated instance mask annotations are not as accurate as the manually annotate ones and that there is a domain gap between the artificially generated training images and the original images of the validation and test sets. Hence, a topic for future research is to generate artificial training data with a more realistic appearance, such that the domain gap between the artificial and real data is closed. Moreover, currently, the data generation leads to a random assembly of objects within an image. While the statistics of object categories that are present within the images could be considered if they are known, the positioning of objects, such that they reflect a realistic scenario, is a more difficult task that should be addressed in future work.

Finally, one of the requirements of industrial algorithms that certainly should be addressed in future work is reliability and uncertainty estimation. Today, for most users, DL-based models are still mysterious black boxes. Only if the predictions of these methods are meaningful and the models are able to show that they are unsure in certain cases, users will start to trust them. For example, if physicians see that a CNN only predicts a very high confidence if it is absolutely sure about its diagnosis, they might be willing to take the CNN's prediction as a second opinion. In contrast, if the CNN also returns high scores for false predictions, the physicians are not able to rely on the model's outputs at all, even if statistically the CNN is outperforming humans.

# Appendices

# A Rotational Invariance for CNNs

In this supplementary material for Chapter 3, we add the descriptions of all networks that we used in our ablation study. For some examples, we show in more detail how we calculate the number of learnable parameters. Additionally, we also provide results of different CNN architectures for all models that we trained on MNIST, CIFAR-10, and CIFAR-100.

## A.1 CNN architectures

The architectures of different CNNs we used for MNIST are summarized in Table A.1. Note that in Table 1 of the submitted paper version there is a mistake for model `RP_RF_1`, where for conv2, conv3, and conv4, the number of rotations $n_r$ should be 1 instead of 8.

The architecture for the `RP_RF_1 MNIST` model is as follows: We use a rotational convolution and pooling module with $n_f = 40$ filters and, depending on the setting, $n_r = 1, 2, 4, 8, 12, 16$, or 32 rotations. The second convolution layer is a conventional convolution with $n_f = 120$ filters. Both convolution layers have filter size $5 \times 5$ and stride 1 and are followed each by a $2 \times 2$ max-pooling with stride 2. The classifier consists of a fully-connected layer with 500 neurons that is followed by a ReLU nonlinearity and by a second fully-connected layer with 10 neurons. Finally, a softmax is applied.

Exemplary calculations of the number of learnable parameters $n_p$ are shown in Table A.2.

## A.2 Results

**MNIST** Results of different models on the MNIST dataset are shown in Table A.3.

**CIFAR** Results of different models on the CIFAR-10 and CIFAR-100 datasets are shown in Table A.4. Note that for CIFAR, we did not train `RC` or `RC_RF` models, as those were not performing well on MNIST.

CNN architectures

| | | conv1 | | | conv2 | | | conv3 | | | conv4 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | RF | $n_f$ | $n_r$ | RP | $n_f$ | $n_r$ | RP | $n_f$ | $n_r$ | RP | $n_f$ | $n_r$ | RP | $n_p$ |
| reference | | 10 | 1 | | 20 | 1 | | 40 | 1 | | 80 | 1 | | 130 050 |
| reference_lrg | | 80 | 1 | | 160 | 1 | | 320 | 1 | | 80 | 1 | | 382 320 |
| RC_RF_1 | ✓ | 10 | 8 | | 20 | 1 | | 40 | 1 | | 80 | 1 | | 142 650 |
| RC_RF_2 | ✓ | 10 | 1 | | 20 | 8 | | 40 | 1 | | 80 | 1 | | 180 450 |
| RC_RF_3 | ✓ | 10 | 1 | | 20 | 1 | | 40 | 8 | | 80 | 1 | | 331 650 |
| RC_RF_4 | ✓ | 10 | 1 | | 20 | 1 | | 40 | 1 | | 80 | 8 | | 703 490 |
| RC_RF_12 | ✓ | 10 | 8 | | 20 | 8 | | 40 | 1 | | 80 | 1 | | 193 050 |
| RC_RF_123 | ✓ | 10 | 8 | | 20 | 8 | | 40 | 8 | | 80 | 1 | | 394 650 |
| RC_RF_1234 | ✓ | 10 | 8 | | 20 | 8 | | 40 | 8 | | 80 | 8 | | 968 090 |
| RP_RF_1 | ✓ | 10 | 8 | ✓ | 20 | 1 | | 40 | 1 | | 80 | 1 | | 130 050 |
| RP_RF_2 | ✓ | 10 | 1 | | 20 | 8 | ✓ | 40 | 1 | | 80 | 1 | | 130 050 |
| RP_RF_3 | ✓ | 10 | 1 | | 20 | 1 | | 40 | 8 | ✓ | 80 | 1 | | 130 050 |
| RP_RF_4 | ✓ | 10 | 1 | | 20 | 1 | | 40 | 1 | | 80 | 8 | ✓ | 130 050 |
| RP_RF_12 | ✓ | 10 | 8 | ✓ | 20 | 8 | ✓ | 40 | 1 | | 80 | 1 | | 130 050 |
| RP_RF_123 | ✓ | 10 | 8 | ✓ | 20 | 8 | ✓ | 40 | 8 | ✓ | 80 | 1 | | 130 050 |
| RP_RF_1234 | ✓ | 10 | 8 | ✓ | 20 | 8 | ✓ | 40 | 8 | ✓ | 80 | 8 | ✓ | 130 050 |
| RC_1 [20] | | 10 | 8 | | 20 | 1 | | 40 | 1 | | 80 | 1 | | 142 650 |
| RC_2 [20] | | 10 | 1 | | 20 | 8 | | 40 | 1 | | 80 | 1 | | 180 450 |
| RC_3 [20] | | 10 | 1 | | 20 | 8 | | 40 | 8 | | 80 | 1 | | 331 650 |
| RC_4 [20] | | 10 | 1 | | 20 | 8 | | 40 | 1 | | 80 | 8 | | 703 490 |
| RC_12 [20] | | 10 | 8 | | 20 | 8 | | 40 | 1 | | 80 | 1 | | 193 050 |
| RC_123 [20] | | 10 | 8 | | 20 | 8 | | 40 | 8 | | 80 | 1 | | 394 650 |
| RC_1234 [20] | | 10 | 8 | | 20 | 8 | | 40 | 8 | | 80 | 8 | | 968 090 |
| RP_1 [20] | | 10 | 8 | ✓ | 20 | 1 | ✓ | 40 | 1 | ✓ | 80 | 1 | | 130 050 |
| RP_2 [20] | | 10 | 1 | ✓ | 20 | 8 | ✓ | 40 | 1 | ✓ | 80 | 1 | | 130 050 |
| RP_3 [20] | | 10 | 1 | ✓ | 20 | 1 | ✓ | 40 | 8 | ✓ | 80 | 1 | | 130 050 |
| RP_4 [20] | | 10 | 1 | ✓ | 20 | 1 | ✓ | 40 | 1 | ✓ | 80 | 8 | ✓ | 130 050 |
| RP_12 [20] | | 10 | 8 | ✓ | 20 | 8 | ✓ | 40 | 1 | | 80 | 1 | | 130 050 |
| RP_123 [20] | | 10 | 8 | ✓ | 20 | 8 | ✓ | 40 | 8 | ✓ | 80 | 1 | | 130 050 |
| RP_1234 [20] | | 10 | 8 | ✓ | 20 | 8 | ✓ | 40 | 8 | ✓ | 80 | 8 | ✓ | 130 050 |
| ORN [213] | | 10 | 8 | | 20 | 8 | | 40 | 8 | | 80 | 8 | ✓ | 382 320 |

**Table A.1: CNN architectures.**

Number of parameters and activations.

| layer | | reference | | | RP_RF_1 | | | ORN | | | RC_1234 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $n_f$ | size | $n_p$ | $n_f$ | size | $n_p$ | $n_f$ | size | $n_p$ | $n_f$ | size | $n_p$ |
| data | raw | | 1x28x28 | | | 1x28x28 | | | 1x28x28 | | | 1x28x28 | |
| | extend | | - | | | - | | | 8x28x28 | | | - | |
| conv1 | filter | 10 | 1x3x3 | 90 | 10 | 1x3x3 | 90 | 10 | 8x3x3 | 720 | 10 | 1x3x3 | 90 |
| | act | | 10x26x26 | | | 80x26x26 | | | 80x26x26 | | | 80x26x26 | |
| | RP | | | | | 10x26x26 | | | | | | | |
| pool1 | act | | 10x13x13 | | | 10x13x13 | | | 80x13x13 | | | 80x13x13 | |
| conv2 | filter | 20 | 10x3x3 | 1 800 | 20 | 10x3x3 | 1 800 | 20 | 80x3x3 | 14 400 | 20 | 80x3x3 | 14 400 |
| | act | | 20x11x11 | | | 160x11x11 | | | 160x11x11 | | | 160x11x11 | |
| | RP | | | | | 20x11x11 | | | | | | | |
| pool2 | act | | 20x6x6 | | | 20x6x6 | | | 160x6x6 | | | 160x6x6 | |
| conv3 | filter | 40 | 20x3x3 | 7 200 | 40 | 20x3x3 | 7 200 | 40 | 160x3x3 | 57 600 | 40 | 160x3x3 | 57 600 |
| | act | | 40x6x6 | | | 320x6x6 | | | 320x6x6 | | | 320x6x6 | |
| | RP | | | | | 40x6x6 | | | | | | | |
| pool3 | act | | 40x3x3 | | | 40x3x3 | | | 320x3x3 | | | 320x3x3 | |
| conv4 | filter | 80 | 40x3x3 | 28 800 | 80 | 40x3x3 | 28 800 | 80 | 320x3x3 | 230 400 | 80 | 320x3x3 | 230 400 |
| | act | | 80x1x1 | | | 640x1x1 | | | 640x1x1 | | | 640x1x1 | |
| | RP | | | | | 80x1x1 | | | 80x1x1 | | | | |
| fc1 | wghts | | 80x1024 | 81 920 | | 80x1024 | 81 920 | | 80x1024 | 81 920 | | 640x1024 | 655 360 |
| | nrons | | 1024 | | | 1024 | | | 1024 | | | 1024 | |
| fc2 | wghts | | 1024x10 | 10 240 | | 1024x10 | 10 240 | | 1024x10 | 10 240 | | 1024x10 | 10 240 |
| | nrons | | 10 | | | 10 | | | 10 | | | 10 | |
| sum | $n_p$ | | | **130 050** | | | **130 050** | | | **382 320** | | | **968 090** |

**Table A.2: Exemplary calculation of number of learnable parameters $n_p$ for some CNN architectures.** The abbreviations are as follows: dimension of activations (act), dimension of weights (wghts), number of neurons (nrons).

| MNIST-rot36 | |
|---|---|
| model | min error test [%] |
| reference | 58.20 |
| RC_1 [20] | 58.92 |
| RC_2 [20] | 58.20 |
| RC_3 [20] | 56.81 |
| RC_4 [20] | 58.05 |
| RC_12 [20] | 58.12 |
| RC_123 [20] | 56.85 |
| RC_1234 [20] | 57.21 |
| RP_1 [20] | 58.17 |
| RP_2 [20] | 57.90 |
| RP_3 [20] | 57.81 |
| RP_4 [20] | 58.61 |
| RP_12 [20] | 58.99 |
| RP_123 [20] | 58.54 |
| RP_1234 [20] | 48.24 |
| ORN (ORPooling) [213] | 42.59 |
| RC_RF_1 | 60.64 |
| RC_RF_2 | 58.72 |
| RC_RF_3 | 57.66 |
| RC_RF_4 | 58.05 |
| RC_RF_12 | 60.41 |
| RC_RF_123 | 59.81 |
| RC_RF_1234 | 61.43 |
| RP_RF_1 | **19.85** |
| RP_RF_2 | 43.46 |
| RP_RF_3 | 59.94 |
| RP_RF_4 | 58.61 |
| RP_RF_12 | 24.00 |
| RP_RF_123 | 22.44 |
| RP_RF_1234 | 23.71 |
| RP_RF_1 MNIST $n_r = 1$ | 59.61 |
| RP_RF_1 MNIST $n_r = 2$ | 46.24 |
| RP_RF_1 MNIST $n_r = 4$ | 16.59 |
| RP_RF_1 MNIST $n_r = 8$ | 14.11 |
| RP_RF_1 MNIST $n_r = 12$ | 12.84 |
| RP_RF_1 MNIST $n_r = 16$ | 12.37 |
| RP_RF_1 MNIST $n_r = 32$ | **12.20** |

**Table A.3: Results for MNIST.** Models are trained on MNIST, but evaluated on MNIST-rot36.

| mboxCIFAR-rot36 | | |
|---|---|---|
| model | CIFAR-10 | CIFAR-100 |
| | min error test [%] | |
| reference | 67.55 | 83.65 |
| RP_1 [20] | 66.91 | 83.40 |
| RP_2 [20] | 66.72 | 82.53 |
| RP_3 [20] | 67.23 | 83.02 |
| RP_4 [20] | 67.41 | 82.94 |
| RP_12 [20] | 65.90 | 82.23 |
| RP_123 [20] | 66.41 | 82.03 |
| RP_1234 [20] | 62.55 | 79.03 |
| ORN (ORPooling) [213] | 59.31 | 78.36 |
| RP_RF_1 | **55.88** | **77.06** |
| RP_RF_2 | 65.45 | 78.98 |
| RP_RF_3 | 67.37 | 81.80 |
| RP_RF_4 | 67.41 | 82.94 |
| RP_RF_12 | 57.69 | 79.23 |
| RP_RF_123 | 58.43 | 79.49 |
| RP_RF_1234 | 58.62 | 80.25 |

**Table A.4: Results for CIFAR-10/100.** Models are trained on CIFAR-10/100, but evaluated on CIFAR-10/100-rot36.

# B
# D2S: Densely Segmented Supermarket Dataset

We provide the following supplementary material:

- Qualitative results of the instance segmentation models.
- Per-class results.
- The class tree.
- A video showing the quality of annotations and the variation of the dataset is available at https://www.mvtec.com/company/research/datasets/mvtec-d2s.

## B.1 Qualitative Results

Fig. B.1 shows qualitative instance segmentation results of MRCNN (Detectron [56]). The model has been trained on different splits of the *D2S* dataset. Note that the model trained on *train+aug* performs significantly better than the model trained only on *train*, especially for objects with occlusion. All models struggle with textured background and reflections.

## B.2 Per-Class Results

Table B.1 shows the per-class *AP* values for MRCNN (Detectron [56]), for all 60 classes and all training splits. In general, the highest *AP*s are achieved for classes with large objects and no reflections, such as *lettuce, salad_iceberg* or *oranges*. The data augmentation boosts particularly the worst-performing classes, such as *banana_single, cucumber, zucchini* and all *gepa_bio_und_fair_**-classes.

## B.3 Class Tree

The class tree assigns superclasses to classes according to their appearance or packaging in a hierarchical manner. It is visualized in Fig. B.2.

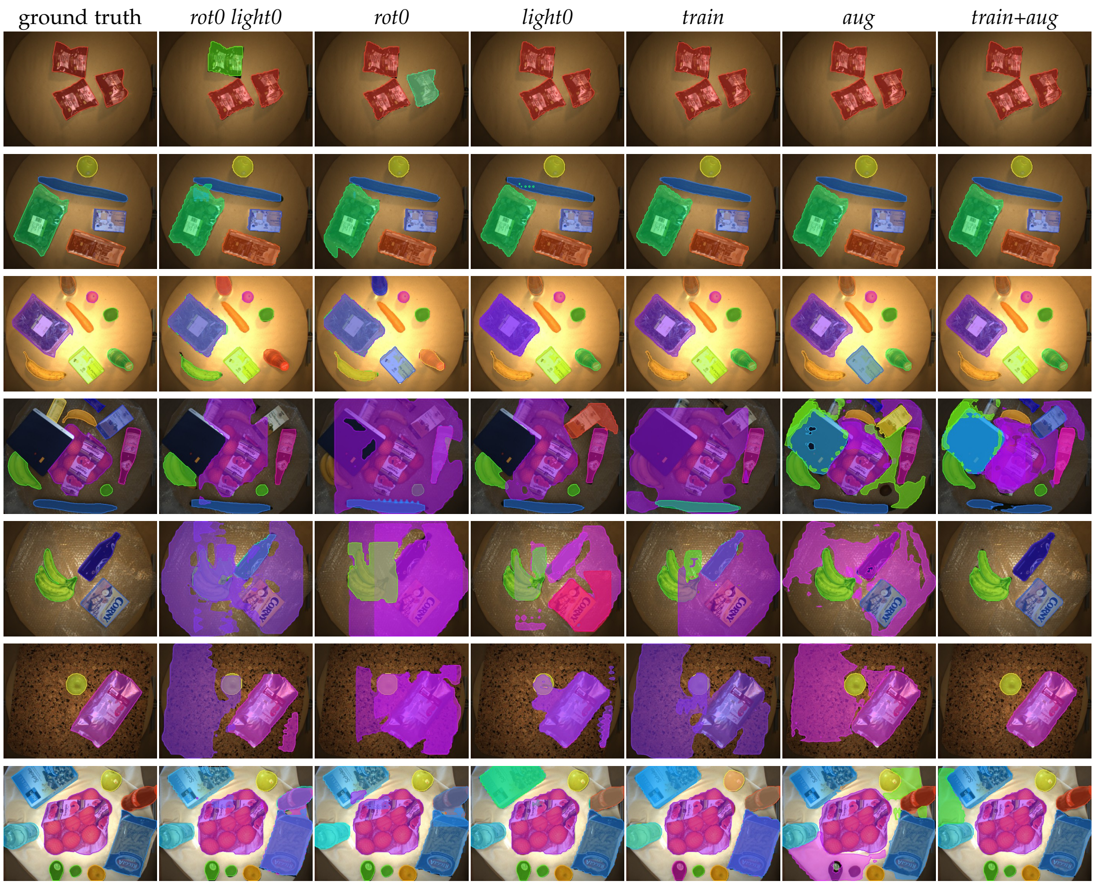


**Figure B.1: MRCNN (Detectron [56]) qualitative results.** A collection of exemplary results of MRCNN trained on different splits. Note that complex backgrounds often lead to false detections. The object classes are color-coded.

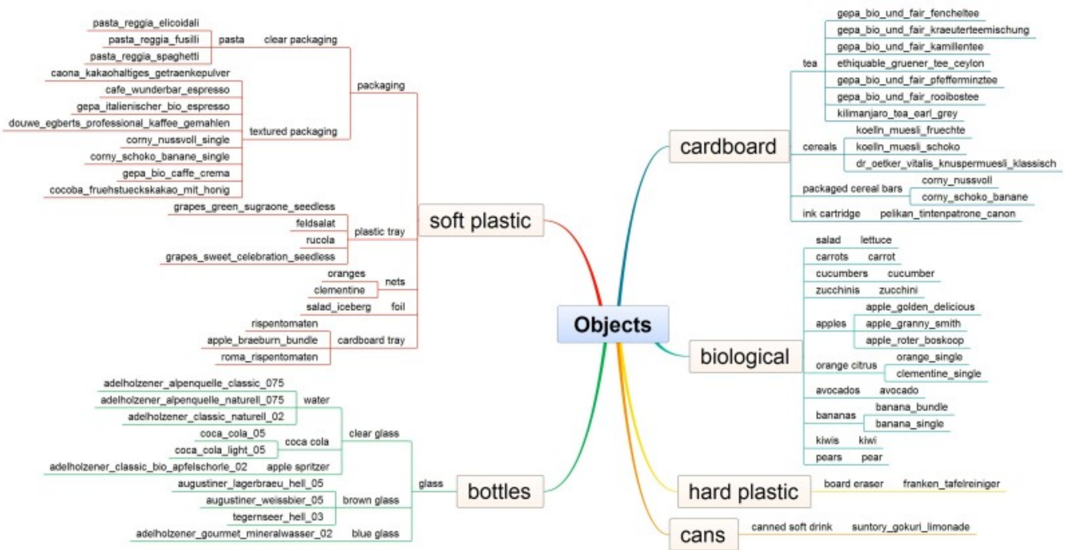


**Figure B.2: Class Tree.** Objects are sorted based mainly on their appearance and packaging rather than on their content. For example, different kinds of coffee are close to each other since they all have a textured, soft plastic packaging. But they are far from all sorts of tea, which is packaged in cardboard.

| class | *train* | *rot0* | *light0* | *rot0_light0* | *aug* | *train+aug* |
|---|---|---|---|---|---|---|
| adelholzener_alpenquelle_classic_075 | 40.6 | 16.4 | 28.2 | 9.4 | 43.1 | 44.8 |
| adelholzener_alpenquelle_naturell_075 | 53.8 | 23.8 | 37.1 | 23.1 | 72.1 | 76.3 |
| adelholzener_classic_bio_apfelschorle_02 | 57.7 | 12.1 | 56.7 | 4.7 | 65.8 | 75.9 |
| adelholzener_classic_naturell_02 | 48.5 | 11.3 | 59.3 | 36.3 | 68.4 | 79.0 |
| adelholzener_gourmet_mineralwasser_02 | 69.9 | 36.9 | 61.4 | 45.7 | 76.2 | 86.8 |
| augustiner_lagerbraeu_hell_05 | 48.6 | 17.1 | 57.0 | 17.9 | 49.8 | 70.1 |
| augustiner_weissbier_05 | 15.6 | 12.7 | 21.9 | 13.0 | 41.2 | 50.5 |
| coca_cola_05 | 45.8 | 23.7 | 48.8 | 16.4 | 53.2 | 61.2 |
| coca_cola_light_05 | 38.9 | 16.7 | 35.1 | 9.1 | 55.3 | 70.8 |
| suntory_gokuri_limonade | 67.1 | 51.6 | 67.2 | 54.0 | 59.4 | 83.0 |
| tegernseer_hell_03 | 44.8 | 11.7 | 38.5 | 6.0 | 47.7 | 67.8 |
| corny_nussvoll | 53.6 | 45.1 | 49.2 | 55.3 | 93.0 | **96.6** |
| corny_nussvoll_single | 70.9 | 40.8 | 64.2 | 32.9 | 77.6 | 84.2 |
| corny_schoko_banane | 45.2 | 12.5 | 37.9 | 29.8 | 84.1 | 90.1 |
| corny_schoko_banane_single | 23.6 | 14.1 | 24.0 | 19.2 | 59.7 | 64.0 |
| dr_oetker_vitalis_knuspermuesli_klassisch | 48.9 | 28.7 | 24.8 | 24.6 | 83.4 | 91.0 |
| koelln_muesli_fruechte | 51.1 | 24.3 | 53.8 | 26.7 | 88.8 | 93.3 |
| koelln_muesli_schoko | 57.8 | 4.4 | 38.6 | 8.1 | 85.3 | 90.2 |
| caona_kakaohaltiges_getraenkepulver | 56.9 | 45.3 | 44.9 | 56.0 | 82.0 | 88.0 |
| cocoba_fruehstueckskakao_mit_honig | 51.0 | 43.0 | 44.9 | 36.4 | 76.1 | 86.7 |
| cafe_wunderbar_espresso | 39.3 | 12.8 | 44.5 | 37.1 | 83.0 | 88.2 |
| douwe_egberts_professional_kaffee_gemahlen | 45.8 | 38.2 | 42.2 | 38.7 | 71.5 | 75.1 |
| gepa_bio_caffe_crema | 40.4 | 31.5 | 41.5 | 41.9 | 81.9 | 82.1 |
| gepa_italienischer_bio_espresso | 59.3 | 30.9 | 52.5 | 19.6 | 61.9 | 65.5 |
| apple_braeburn_bundle | 59.8 | 56.8 | 60.7 | 66.2 | 82.3 | 91.7 |
| apple_golden_delicious | 78.2 | **72.8** | 66.4 | 73.0 | 76.1 | 91.3 |
| apple_granny_smith | 72.5 | 71.4 | 65.9 | 66.4 | 75.5 | 87.4 |
| apple_roter_boskoop | **79.4** | 71.2 | **76.6** | **79.8** | 78.1 | 94.1 |
| avocado | 73.0 | 60.6 | 76.2 | 74.2 | 84.2 | 93.4 |
| banana_bundle | 51.4 | 47.5 | 55.8 | 46.9 | 76.8 | 86.6 |
| banana_single | 27.4 | 18.0 | 28.6 | 20.6 | 55.9 | 57.9 |
| clementine | 54.6 | 57.5 | 58.4 | 59.8 | 79.7 | 83.0 |
| clementine_single | 59.5 | 65.0 | 60.4 | 59.9 | 81.2 | 89.7 |
| grapes_green_sugraone_seedless | 44.0 | 29.4 | 27.8 | 34.2 | 68.9 | 77.2 |
| grapes_sweet_celebration_seedless | 61.5 | 54.9 | 51.4 | 46.8 | 75.7 | 80.7 |
| kiwi | 45.5 | 60.0 | 55.6 | 54.0 | 68.9 | 88.7 |
| orange_single | 69.6 | 70.5 | 66.5 | 72.7 | 81.5 | 91.2 |
| oranges | 61.1 | 51.1 | 61.2 | 65.8 | 81.3 | 86.3 |
| pear | 56.4 | 40.7 | 74.8 | 70.5 | 76.5 | 85.1 |
| pasta_reggia_elicoidali | 43.1 | 23.3 | 48.2 | 38.8 | 88.5 | 90.5 |
| pasta_reggia_fusilli | 48.8 | 23.7 | 51.6 | 38.4 | 76.7 | 76.5 |
| pasta_reggia_spaghetti | 55.4 | 28.3 | 45.0 | 28.5 | 85.0 | 90.5 |
| franken_tafelreiniger | 48.2 | 33.4 | 39.8 | 34.1 | 52.8 | 67.7 |
| pelikan_tintenpatrone_canon | 30.5 | 25.6 | 27.2 | 32.1 | 69.1 | 75.8 |
| ethiquable_gruener_tee_ceylon | 52.3 | 35.9 | 48.1 | 18.6 | 77.8 | 86.4 |
| gepa_bio_und_fair_fencheltee | 19.3 | 5.0 | 14.7 | 10.9 | 50.2 | 55.9 |
| gepa_bio_und_fair_kamillentee | 32.7 | 17.9 | 23.3 | 18.4 | 64.4 | 71.1 |
| gepa_bio_und_fair_kraeuterteemischung | 27.0 | 10.1 | 17.8 | 9.4 | 58.1 | 72.6 |
| gepa_bio_und_fair_pfefferminztee | 37.5 | 5.2 | 24.5 | 7.9 | 69.3 | 77.2 |
| gepa_bio_und_fair_rooibostee | 47.3 | 29.1 | 45.0 | 21.9 | 70.4 | 77.3 |
| kilimanjaro_tea_earl_grey | 46.1 | 31.6 | 38.4 | 25.4 | 64.1 | 75.6 |
| cucumber | 29.9 | 23.9 | 33.0 | 27.6 | 71.4 | 77.6 |
| carrot | 54.1 | 44.5 | 42.8 | 44.5 | 62.7 | 72.1 |
| feldsalat | 49.5 | 33.2 | 40.5 | 34.8 | 77.4 | 84.0 |
| lettuce | 51.6 | 55.0 | 34.7 | 38.4 | **93.4** | 96.2 |
| rispentomaten | 45.0 | 39.4 | 45.2 | 52.3 | 71.9 | 78.0 |
| roma_rispentomaten | 38.8 | 33.7 | 37.5 | 40.2 | 71.6 | 73.7 |
| rucola | 38.9 | 15.8 | 41.0 | 12.8 | 71.7 | 76.1 |
| salad_iceberg | 55.7 | 53.4 | 52.9 | 66.8 | 77.9 | 86.4 |
| zucchini | 49.4 | 23.9 | 51.0 | 24.8 | 68.7 | 84.0 |

**Table B.1: MRCNN (Detectron [56])** $AP$ **values per class.** The $AP$ values of the MRCNN models trained on different training sets are calculated on the *test* set. The mean $AP$ is calculated over all 60 classes. The highest class $AP$ values are highlighted in bold.

# C Data Generation for Few-Shot Detection

This is the appendix for Chapter 7. In Tables C.1, C.2, and C.3 we show the influence of augmenting a different amount of images and adding specific augmentations for *baseline*, *weakly*, and *weakly cleaned*, respectively. The performance is given in terms of *AP* percentage values. Abbreviations for augmentation types are as follows: *neighboring* (NB), *random background* (RB), and *reflections* (RE). The models have been trained using the same settings as in Section 7.4.

| Training Set | *augm* 2500 | *augm* 5000 | *augm* 10000 |
|---|---|---|---|
| *train* | 48.3 | 48.3 | 48.3 |
| *train + augm* | 77.0 | 77.8 | 78.4 |
| + NB | 77.3 | 78.5 | 78.3 |
| + RB | 78.2 | 78.5 | 79.1 |
| + NB + RB | 79.3 | **80.1** | 79.9 |

**Table C.1: Baseline results**

| Training Set | *augm* 2500 | *augm* 5000 | *augm* 10000 |
|---|---|---|---|
| *train* | 8.5 | 8.5 | 8.5 |
| *train + augm* | 62.8 | 62.2 | 65.0 |
| + NB | 64.0 | 63.7 | 65.8 |
| + RB | 64.0 | 64.9 | 63.5 |
| + NB + RB | 64.0 | **66.8** | 65.3 |

**Table C.2: Weakly results.**

| Training Set | *augm* 2500 | *augm* 5000 | *augm* 10000 |
|---|---|---|---|
| *train* | 15.9 | 15.9 | 15.9 |
| *train + augm* | 64.8 | 61.9 | 63.0 |
| + NB | 65.0 | 62.5 | 64.7 |
| + RB | 65.9 | 68.0 | 65.8 |
| + RE | 68.1 | 66.9 | 65.3 |
| + NB + RB | 65.9 | **68.9** | 66.9 |
| + NB + RB + RE | 68.4 | 68.5 | 66.9 |

**Table C.3: Weakly cleaned results**

# D Oriented Boxes for Few-Shot Instance Segmentation

## D.1 Calculation of orientation

In this section, we show the calculation of a region's orientation as used in Section 10.3 and provided by the HALCON operator *orientation_region* [143]. Consider a region $R$ consisting of $n$ pixels with row coordinates $r_i$ and column coordinates $c_i$, $i = 1, \ldots, n$. We denote the center of gravity by $g = (r_0, c_0)$ (see HALCON operator *elliptic_axis* [143]):

$$(r_0, c_0) = \frac{1}{n} \sum_{i=1}^{n} (r_i, c_i). \tag{D.1}$$

Further, the second moments of $R$, $M_{ij}$ are given by:

$$M_{ij} = \sum_{(r,c) \in R} (r_0 - r)^i (c_0 - c)^j. \tag{D.2}$$

We calculate the orientation $\phi$ of $R$ by fitting an ellipse to the region that has the same aspect ratio and orientation and get:

$$\phi = -\frac{1}{2} \operatorname{atan2}(2M_{11}, M_{02} - M_{20}) \tag{D.3}$$

An example where this method to extract the orientation has benefits is given in Fig. D.1. To obtain the bounding box with orientation $\phi$ calculated as above, we rotate the mask to be axis-aligned, get the axis-aligned bounding box, and rotate it back.

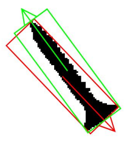


**Figure D.1: Orientation of a region.** Region $R$ of a screw (*black*), smallest OBB (*red*), and OBB with the region's orientation (*green*). It points from the screw's head to its tail and is more consistent over different screw instances.

# E
# Amodal Instance Segmentation

The following is provided as supplementary material for Chapter 11:

- Description of training configuration.
- Qualitative Results *COCOA*.
- Qualitative Results *D2S amodal*.

## E.1 Training Setup

For the training of ARCNN as well as ORCNN we used the Detectron [56] framework built on caffe2[1] / caffe [85].

**Batch size and image size.** For all trainings, two NVIDIA GTX 1080 Ti GPUs were used. On *D2S amodal* we used a batch size of one image per GPU, while on *COCOA* and *COCOA cls*, two images per GPU were used in a batch. The batch size of regions of interest (RoIs) per image was set to 256. The maximum number of RoIs per image before the non-maximum suppression in the region proposal network was set to 1000 per feature pyramid level. The images were scaled such that the shorter side of each image was set to a maximum of 800 pixels while the longer side was not exceeding 1111 pixels.

**Finetuning.** Each training was using the weights from a *COCO*-pretrained model as provided by Detectron. For the case of *COCOA* and *D2S amodal*, the final output layers that are class-specific had to be initialized randomly since the number of classes and their semantic meaning did not fit to the number of classes of *COCO*.

**Solver settings.** Training was performed for 10 000 iterations with a base learning rate of 0.0025 and a weight decay of 0.0001. The same warm-up period and settings as described in the Mask R-CNN paper [70] were used for the learning rate. The learning rate was multiplied by $\gamma = 0.1$ at 6000 and 8000 iterations, respectively.

[1]https://github.com/caffe2/caffe2

## E.2 Qualitative Results

In this section, qualitative results are shown on *COCOA no stuff*, *COCOA cls*, and *D2S amodal*. ORCNN, a model that can predict amodal, invisible, and visible masks simultaneously, is compared against ARCNN, which is only capable to predict amodal masks.

### E.2.1 COCOA

Qualitative results for *COCOA no stuff* and *COCOA cls* are shown in Fig. E.1 and Fig. E.2, respectively. Please note that the ground truth annotations are sometimes wrong or incomplete. Hence, the models sometimes find annotations correctly that have not been annotated in the ground truth. In the case of *COCOA cls*, missing annotations are also generated by our merging strategy. We do not show qualitative results for *COCOA* because the ground truth and results for *stuff* regions make it very difficult to visualize and interpret the results.

Generally, we observe that the predicted amodal regions are slightly better for ARCNN compared to ORCNN. This is also confirmed by the quantitative evaluation in the chapter. However, the occlusion predictions of ORCNN are often very promising and better than the relatively low $AP_{\text{IV}}$ values indicate.

### E.2.2 D2S

For *D2S amodal*, we show qualitative examples for the class-agnostic mask versions of ARCNN and ORCNN in Fig. E.3. It is difficult to see a qualitative difference between the amodal masks of ARCNN and ORCNN. The predicted invisible masks of ORCNN are often at the correct location but their IoU with the ground truth invisible masks is rather low. This explains the low $AP_{\text{IV}}$ values.

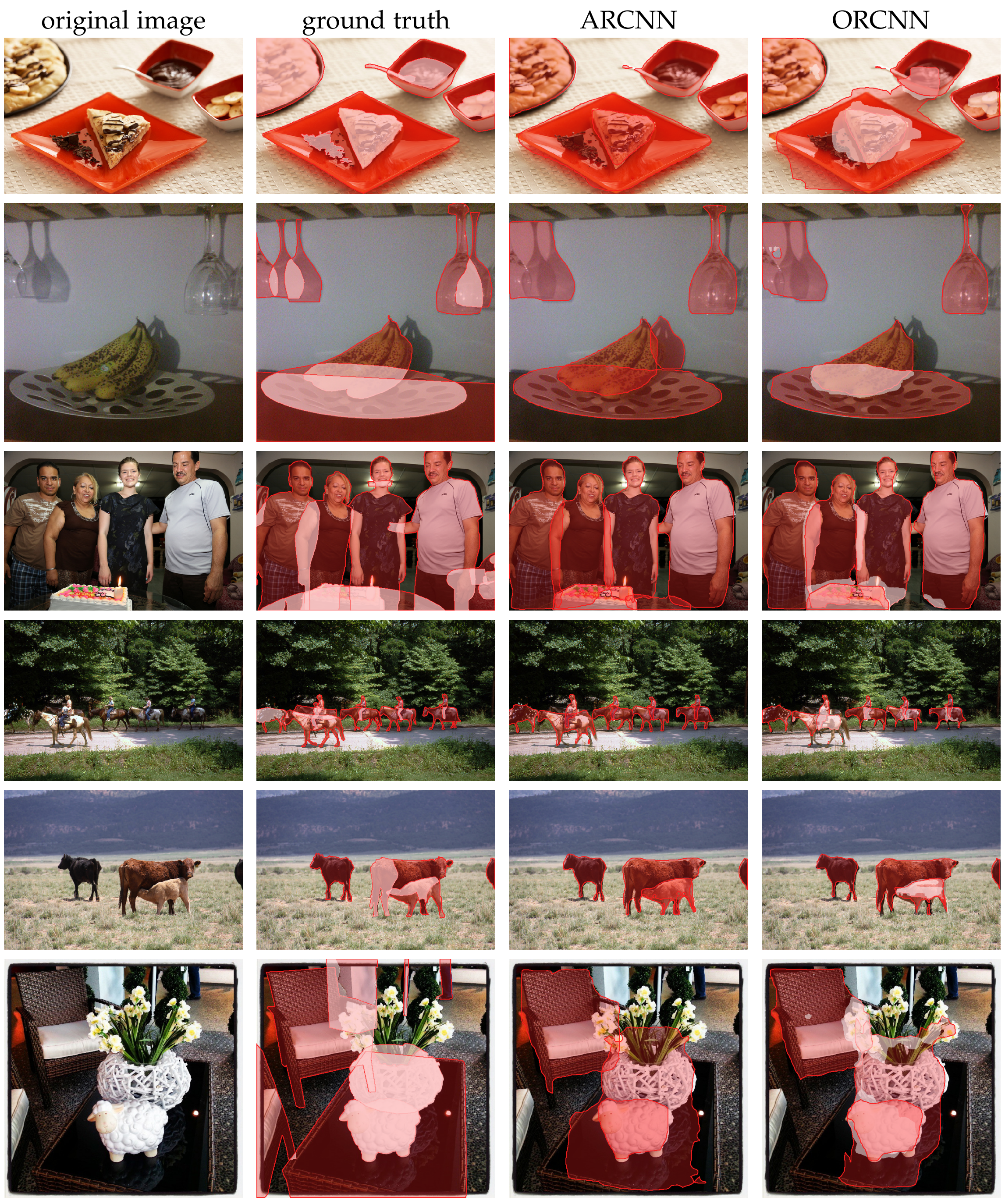


**Figure E.1: COCOA no stuff results.** (*From left to right*) Input image, ground truth annotations, results of ARCNN, results of ORCNN. Note that for all images, the minimal score of results was set to 0.8. The ordering of the result overlays might be different to the ordering of the ground truth annotation overlays.

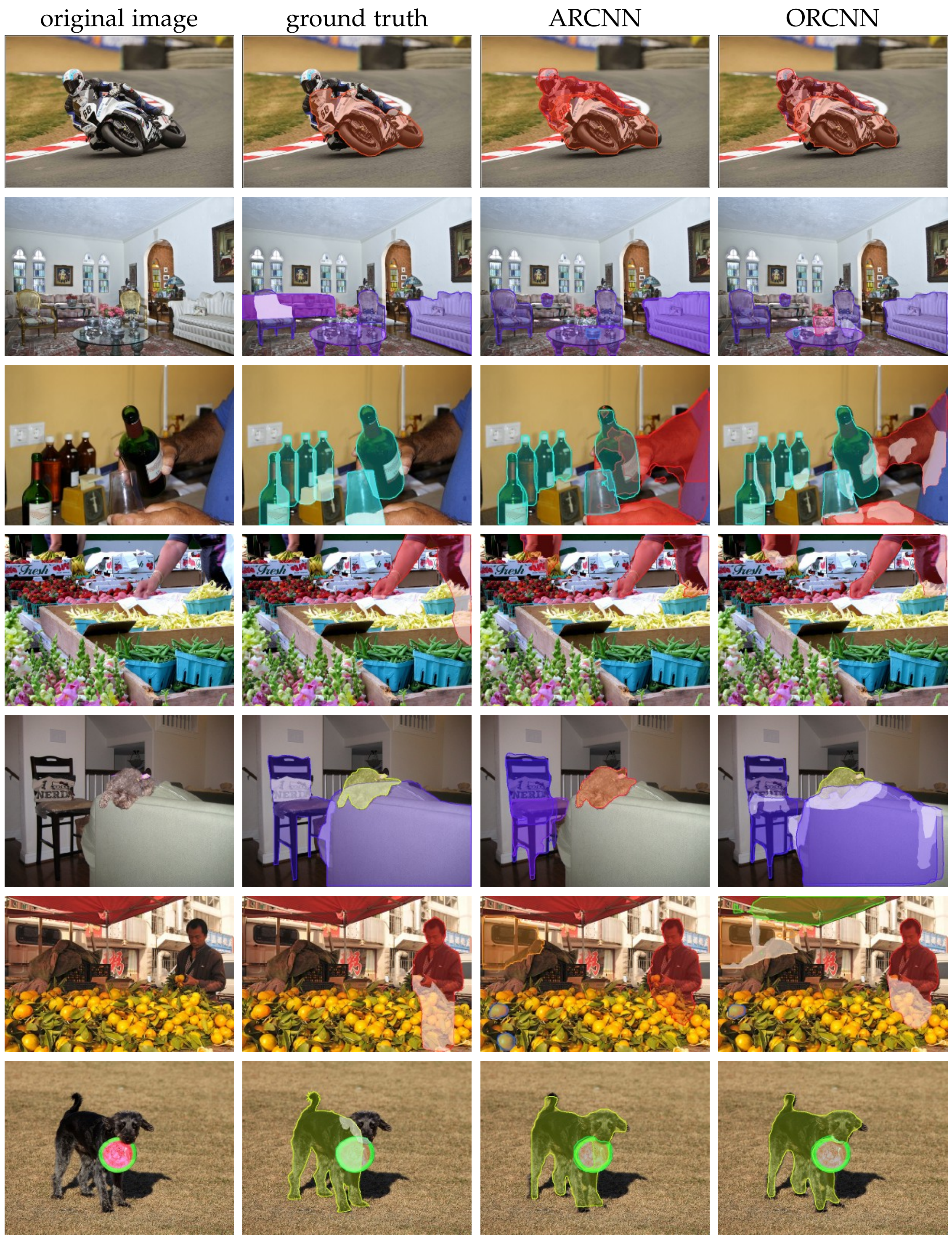


**Figure E.2: COCOA cls results.** (*From left to right*) Input image, ground truth annotations, results of ARCNN, results of ORCNN. Note that for all images, the minimal score of results was set to 0.5. The ordering of the result overlays might be different to the ordering of the ground truth annotation overlays.

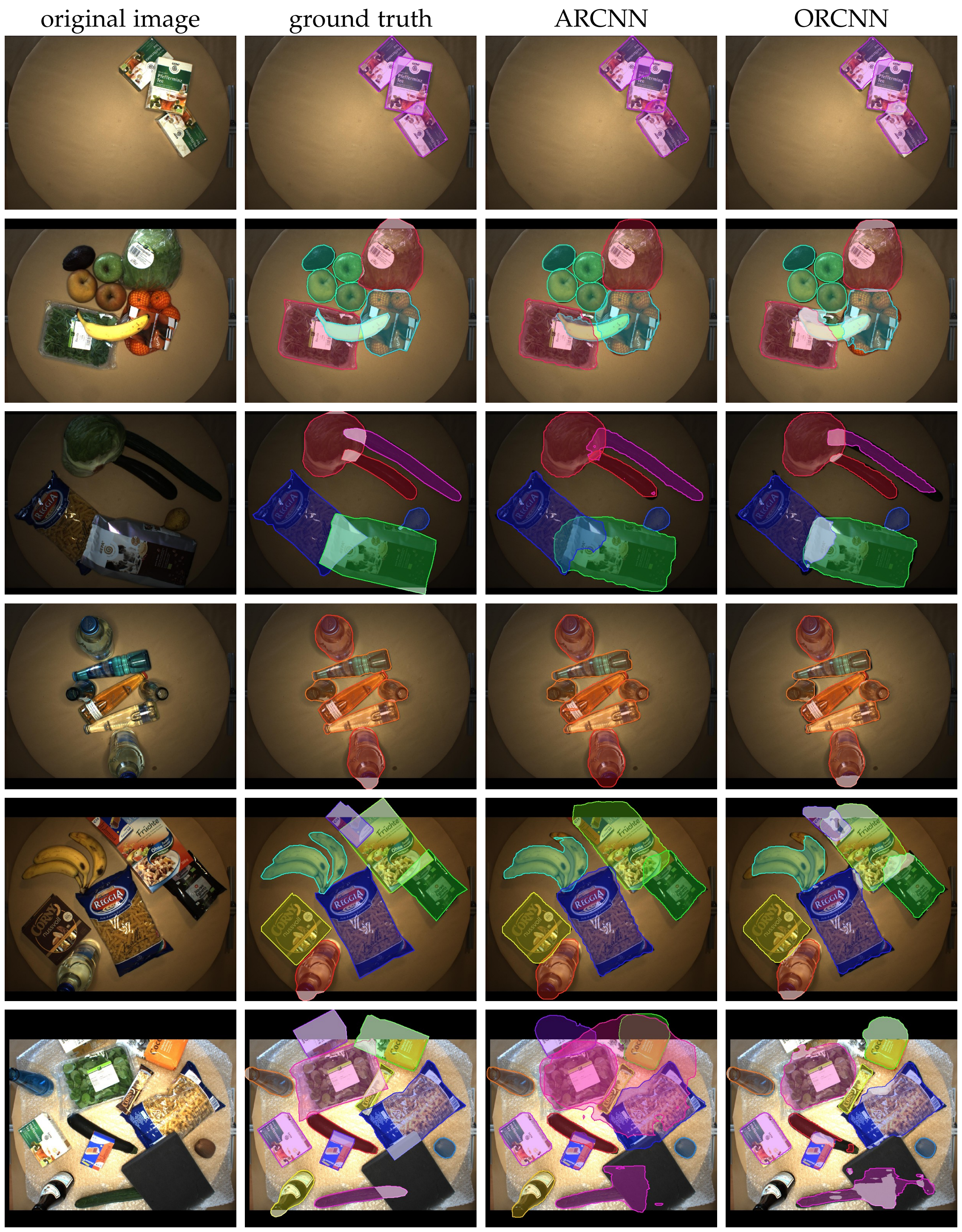


**Figure E.3: D2S amodal results.** (*From left to right*) Input image, ground truth annotations, results of ARCNN, results of ORCNN. Note that for all images, the minimal score of results was set to 0.8. The ordering of the result overlays might be different to the ordering of the ground truth annotation overlays.

# Bibliography

[1] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan. YouTube-8M: A Large-Scale Video Classification Benchmark. *CoRR*, abs/1609.08675, 2016. URL http://arxiv.org/abs/1609.08675.

[2] S. Agarwal, A. Awan, and D. Roth. Learning to Detect Objects in Images via a Sparse, Part-Based Representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(11): 1475–1490, 2004. doi:10.1109/TPAMI.2004.108.

[3] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the Objectness of Image Windows. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2189–2202, 2012. doi:10.1109/TPAMI.2012.28.

[4] C. Arteta, V. S. Lempitsky, and A. Zisserman. Counting in the Wild. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Proceedings, Part VII*, volume 9911 of *Lecture Notes in Computer Science*, pages 483–498. Springer, 2016. doi:10.1007/978-3-319-46478-7_30.

[5] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network Dissection: Quantifying Interpretability of Deep Visual Representations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3319–3327, 2017. doi:10.1109/CVPR.2017.354.

[6] H. Bay, T. Tuytelaars, and L. V. Gool. SURF: Speeded Up Robust Features. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision - ECCV 2006, 9th European Conference on Computer Vision, Proceedings, Part I*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer, 2006. doi:10.1007/11744023_32.

[7] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[8] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee. YOLACT: Real-Time Instance Segmentation. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV*, pages 9156–9165, 2019. doi:10.1109/ICCV.2019.00925.

[9] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee. YOLACT++: Better Real-time Instance Segmentation. *CoRR*, abs/1912.06218, 2019. URL http://arxiv.org/abs/1912.06218.

[10] A. Borji, M. Cheng, H. Jiang, and J. Li. Salient Object Detection: A Benchmark. *IEEE Trans. Image Process.*, 24(12):5706–5722, 2015. doi:10.1109/TIP.2015.2487833.

[11] T. Böttger and P. Follmann. The Benefits of Evaluating Tracker Performance Using Pixel-Wise Segmentations. In *2017 IEEE International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1983–1991, 2017. doi:10.1109/ICCVW.2017.232.

[12] T. Böttger, P. Follmann, and M. Fauser. Measuring the Accuracy of Object Detectors and Trackers. In V. Roth and T. Vetter, editors, *Pattern Recognition - 39th German Conference, GCPR 2017, Proceedings*, volume 10496 of *Lecture Notes in Computer Science*, pages 415–426. Springer, 2017. doi:10.1007/978-3-319-66709-6_33.

[13] T. J. Brinker, A. Hekler, A. H. Enk, J. Klode, A. Hauschild, C. Berking, B. Schilling, S. Haferkamp, D. Schadendorf, S. Fröhling, et al. A convolutional neural network trained with dermoscopic images performed on par with 145 dermatologists in a clinical melanoma image classification task. *European Journal of Cancer*, 111:148–154, 2019. doi:10.1016/j.ejca.2019.02.005.

[14] Z. Cai and N. Vasconcelos. Cascade R-CNN: Delving Into High Quality Object Detection. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 6154–6162, 2018. doi:10.1109/CVPR.2018.00644.

[15] J. Carreira and C. Sminchisescu. CPMC: Automatic Object Segmentation Using Constrained Parametric Min-Cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(7): 1312–1328, 2012. doi:10.1109/TPAMI.2011.231.

[16] A. Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(OC I, 10 (383), 399–402):536–538, 1847. URL https://cs.uwaterloo.ca/~y328yu/classics/cauchy-en.pdf. Accessed: 2021-02-07.

[17] Y. Chen, X. Liu, and M. Yang. Multi-instance Object Segmentation with Occlusion Handling. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3470–3478, 2015. doi:10.1109/CVPR.2015.7298969.

[18] M. Cheng, Z. Zhang, W. Lin, and P. H. S. Torr. BING: Binarized Normed Gradients for Objectness Estimation at 300fps. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3286–3293, 2014. doi:10.1109/CVPR.2014.414.

[19] D. C. Cireşan, U. Meier, and J. Schmidhuber. Multi-column Deep Neural Networks for Image Classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3642–3649, 2012. doi:10.1109/CVPR.2012.6248110.

[20] T. Cohen and M. Welling. Group Equivariant Convolutional Networks. In M. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages

2990–2999. PMLR, 2016. URL http://proceedings.mlr.press/v48/cohenc16.html. Accessed: 2021-02-07.

[21] T. S. Cohen and M. Welling. Steerable CNNs. In *5th International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2017. URL https://openreview.net/forum?id=rJQKYt5ll. Accessed: 2021-02-07.

[22] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3213–3223, 2016. doi:10.1109/CVPR.2016.350.

[23] C. Cortes and V. Vapnik. Support-Vector Networks. *Mach. Learn.*, 20(3):273–297, 1995. doi:10.1007/BF00994018.

[24] J. Dai, K. He, Y. Li, S. Ren, and J. Sun. Instance-Sensitive Fully Convolutional Networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Proceedings, Part VI*, volume 9910 of *Lecture Notes in Computer Science*, pages 534–549. Springer, 2016. doi:10.1007/978-3-319-46466-4_32.

[25] J. Dai, K. He, and J. Sun. Instance-Aware Semantic Segmentation via Multi-task Network Cascades. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3150–3158, 2016. doi:10.1109/CVPR.2016.343.

[26] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable Convolutional Networks. In *IEEE International Conference on Computer Vision, ICCV*, pages 764–773, 2017. doi:10.1109/ICCV.2017.89.

[27] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, pages 886–893, 2005. doi:10.1109/CVPR.2005.177.

[28] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. ImageNet: A Large-Scale Hierarchical Image Database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, pages 248–255, 2009. doi:10.1109/CVPR.2009.5206848.

[29] T. Deselaers, B. Alexe, and V. Ferrari. Weakly Supervised Localization and Learning with Generic Knowledge. *Int. J. Comput. Vis.*, 100(3):275–293, 2012. doi:10.1007/s11263-012-0538-3.

[30] S. Dieleman, K. W. Willett, and J. Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly Notices of the Royal Astronomical Society*, 450(2):1441–1459, 2015. doi:10.1093/mnras/stv632.

[31] J. Ding, N. Xue, Y. Long, G. Xia, and Q. Lu. Learning RoI Transformer for Oriented Object Detection in Aerial Images. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 2849–2858, 2019. doi:10.1109/CVPR.2019.00296.

[32] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian. CenterNet: Keypoint Triplets for Object Detection. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV*, pages 6568–6577, 2019. doi:10.1109/ICCV.2019.00667.

[33] J. C. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011. URL https://dl.acm.org/doi/10.5555/1953048.2021068. Accessed: 2021-02-07.

[34] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *CoRR*, abs/1603.07285, 2016. URL http://arxiv.org/abs/1603.07285.

[35] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107: 3–11, 2018. doi:10.1016/j.neunet.2017.12.012.

[36] I. Endres and D. Hoiem. Category-Independent Object Proposals with Diverse Ranking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(2):222–234, 2014. doi:10.1109/TPAMI.2013.122.

[37] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017. doi:10.1038/nature21056.

[38] M. Everingham and J. Winn. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit. *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, 8, 2011. URL http://host.robots.ox.ac.uk/pascal/VOC/voc2012/devkit_doc.pdf. Accessed: 2021-02-07.

[39] M. Everingham, A. Zisserman, C. K. I. Williams, L. V. Gool, M. Allan, C. M. Bishop, O. Chapelle, N. Dalal, T. Deselaers, G. Dorkó, S. Duffner, J. Eichhorn, J. D. R. Farquhar, M. Fritz, C. Garcia, T. L. Griffiths, F. Jurie, D. Keysers, M. Koskela, J. Laaksonen, D. Larlus, B. Leibe, H. Meng, H. Ney, B. Schiele, C. Schmid, E. Seemann, J. Shawe-Taylor, A. J. Storkey, S. Szedmák, B. Triggs, I. Ulusoy, V. Viitaniemi, and J. Zhang. The 2005 PASCAL Visual Object Classes Challenge. In J. Q. Candela, I. Dagan, B. Magnini, and F. d'Alché-Buc, editors, *Machine Learning Challenges, Evaluating Predictive Uncertainty, Visual Object Classification and Recognizing Textual Entailment, First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, volume 3944 of *Lecture Notes in Computer Science*, pages 117–176. Springer, 2005. doi:10.1007/11736790_8.

[40] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vis.*, 88(2):303–338, 2010. doi:10.1007/s11263-009-0275-4.

[41] M. Everingham, S. M. A. Eslami, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman. The Pascal Visual Object Classes Challenge: A Retrospective. *Int. J. Comput. Vis.*, 111(1):98–136, 2015. doi:10.1007/s11263-014-0733-5.

[42] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, 2010. doi:10.1109/TPAMI.2009.167.

[43] R. Fergus, P. Perona, and A. Zisserman. Object Class Recognition by Unsupervised Scale-Invariant Learning. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, pages 264–271, 2003. doi:10.1109/CVPR.2003.1211479.

[44] P. Follmann and T. Böttger. A Rotationally-Invariant Convolution Module by Feature Map Back-Rotation. In *2018 IEEE Winter Conference on Applications of Computer Vision, WACV*, pages 784–792, 2018. doi:10.1109/WACV.2018.00091.

[45] P. Follmann and B. Radig. Detecting Animals in Infrared Images from Camera-Traps. *Pattern Recognit. Image Anal.*, 28(4):605–611, 2018. doi:10.1134/S1054661818040107.

[46] P. Follmann, T. Böttger, P. Härtinger, R. König, and M. Ulrich. MVTec D2S: Densely Segmented Supermarket Dataset. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Proceedings, Part X*, volume 11214 of *Lecture Notes in Computer Science*, pages 581–597. Springer, 2018. doi:10.1007/978-3-030-01249-6_35.

[47] P. Follmann, B. Drost, and T. Böttger. Acquire, Augment, Segment and Enjoy: Weakly Supervised Instance Segmentation of Supermarket Products. In T. Brox, A. Bruhn, and M. Fritz, editors, *Pattern Recognition - 40th German Conference, GCPR, Proceedings*, volume 11269 of *Lecture Notes in Computer Science*, pages 363–376. Springer, 2018. doi:10.1007/978-3-030-12939-2_25.

[48] P. Follmann, R. König, P. Härtinger, M. Klostermann, and T. Böttger. Learning to See the Invisible: End-to-End Trainable Amodal Instance Segmentation. In *IEEE Winter Conference on Applications of Computer Vision, WACV*, pages 1328–1336, 2019. doi:10.1109/WACV.2019.00146.

[49] W. Förstner and E. Gülch. A fast operator for detection and precise location of distinct points, corners and centres of circular features. In *Proceedings of the ISPRS Intercommission Conference on Fast Processing of Photogrammetric Data*, pages 281–305, 1987. URL https://www.ipb.uni-bonn.de/pdfs/Forstner1987Fast.pdf. Accessed: 2021-02-07.

[50] C. Fu, M. Shvets, and A. C. Berg. RetinaMask: Learning to predict masks improves state-of-the-art single-shot detection for free. *CoRR*, abs/1901.03353, 2019. URL http://arxiv.org/abs/1901.03353.

[51] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybernetics*, 36(4):193–202, 1980. doi:10.1007/BF00344251.

[52] Y. Gal and Z. Ghahramani. Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference. *CoRR*, abs/1506.02158, 2015. URL http://arxiv.org/abs/1506.02158.

[53] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *Int. J. Robotics Res.*, 32(11):1231–1237, 2013. doi:10.1177/0278364913491297.

[54] M. George and C. Floerkemeier. Recognizing Products: A Per-exemplar Multi-label Image Classification Approach. In D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Proceedings, Part II*, volume 8690 of *Lecture Notes in Computer Science*, pages 440–455. Springer, 2014. doi:10.1007/978-3-319-10605-2_29.

[55] G. Ghiasi, T. Lin, and Q. V. Le. NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 7036–7045, 2019. doi:10.1109/CVPR.2019.00720.

[56] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He. Detectron. https://github.com/facebookresearch/detectron, 2018.

[57] R. B. Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision, ICCV*, pages 1440–1448, 2015. doi:10.1109/ICCV.2015.169.

[58] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 580–587, 2014. doi:10.1109/CVPR.2014.81.

[59] I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org/. Accessed: 2021-02-07.

[60] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

[61] R. Guo and D. Hoiem. Beyond the Line of Sight: Labeling the Underlying Surfaces. In A. W. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, editors, *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Proceedings, Part V*, volume 7576 of *Lecture Notes in Computer Science*, pages 761–774. Springer, 2012. doi:10.1007/978-3-642-33715-4_55.

[62] A. Gupta, P. Dollár, and R. B. Girshick. LVIS: A Dataset for Large Vocabulary Instance Segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 5356–5364, 2019. doi:10.1109/CVPR.2019.00550.

[63] S. Gupta, P. Arbelaez, and J. Malik. Perceptual Organization and Recognition of Indoor Scenes from RGB-D Images. In *2013 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 564–571, 2013. doi:10.1109/CVPR.2013.79.

[64] S. Gurumurthy, R. K. Sarvadevabhatla, and R. V. Babu. DeLiGAN: Generative Adversarial Networks for Diverse and Limited Data. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4941–4949, 2017. doi:10.1109/CVPR.2017.525.

[65] H. A. Haenssle, C. Fink, R. Schneiderbauer, F. Toberer, T. Buhl, A. Blum, A. Kalloo, A. B. H. Hassen, L. Thomas, A. Enk, et al. Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. *Annals of Oncology*, 29 (8):1836–1842, 2018. doi:10.1093/annonc/mdy166.

[66] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000. doi:10.1038/35016072.

[67] C. G. Harris and M. Stephens. A Combined Corner and Edge Detector. In C. J. Taylor, editor, *Proceedings of the Alvey Vision Conference, AVC*, pages 1–6. Alvey Vision Club, 1988. doi:10.5244/C.2.23.

[68] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 770–778, 2016. doi:10.1109/CVPR.2016.90.

[69] K. He, X. Zhang, S. Ren, and J. Sun. Identity Mappings in Deep Residual Networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 630–645. Springer, 2016. doi:10.1007/978-3-319-46493-0_38.

[70] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision, ICCV*, pages 2980–2988, 2017. doi:10.1109/ICCV.2017.322.

[71] G. E. Hinton. Learning translation invariant recognition in a massively parallel networks. In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, editors, *PARLE Parallel Architectures and Languages Europe*, pages 1–13. Springer, 1987. doi:10.1007/3-540-17943-7_117.

[72] G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming Auto-Encoders. In T. Honkela, W. Duch, M. A. Girolami, and S. Kaski, editors, *Artificial Neural Networks and Machine Learning - ICANN 2011 - 21st International Conference on Artificial Neural Networks, Proceedings, Part I*, volume 6791 of *Lecture Notes in Computer Science*, pages 44–51. Springer, 2011. doi:10.1007/978-3-642-21735-7_6.

[73] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

[74] D. Hoiem, Y. Chodpathumwan, and Q. Dai. Diagnosing Error in Object Detectors. In A. W. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, editors, *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Proceedings, Part III*, volume 7574 of *Lecture Notes in Computer Science*, pages 340–353. Springer, 2012. doi:10.1007/978-3-642-33712-3_25.

[75] K. Hornik, M. B. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. doi:10.1016/0893-6080(89)90020-8.

[76] J. Hu, L. Shen, and G. Sun. Squeeze-and-Excitation Networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 7132–7141, 2018. doi:10.1109/CVPR.2018.00745.

[77] R. Hu, P. Dollár, K. He, T. Darrell, and R. B. Girshick. Learning to Segment Every Thing. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4233–4241, 2018. doi:10.1109/CVPR.2018.00445.

[78] X. Huang, Y. Li, O. Poursaeed, J. E. Hopcroft, and S. J. Belongie. Stacked Generative Adversarial Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 1866–1875, 2017. doi:10.1109/CVPR.2017.202.

[79] Z. Huang, L. Huang, Y. Gong, C. Huang, and X. Wang. Mask Scoring R-CNN. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 6409–6418, 2019. doi:10.1109/CVPR.2019.00657.

[80] P. J. Huber. Robust Estimation of a Location Parameter. In S. Kotz and N. L. Johnson, editors, *Breakthroughs in Statistics: Methodology and Distribution*, pages 492–518. Springer, 1992. doi:10.1007/978-1-4612-4380-9_35.

[81] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR*, abs/1602.07360, 2016. URL http://arxiv.org/abs/1602.07360.

[82] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. PMLR, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

[83] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial Transformer Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015*, pages 2017–2025, 2015. URL http://papers.nips.cc/paper/5854-spatial-transformer-networks. Accessed: 2021-02-07.

[84] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *IEEE 12th International Conference on Computer Vision, ICCV*, pages 2146–2153, 2009. doi:10.1109/ICCV.2009.5459469.

[85] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In K. A. Hua, Y. Rui, R. Steinmetz, A. Hanjalic, A. Natsev, and W. Zhu, editors, *Proceedings of the ACM International Conference on Multimedia, MM '14*, pages 675–678. ACM, 2014. doi:10.1145/2647868.2654889.

[86] Y. Jiang, X. Zhu, X. Wang, S. Yang, W. Li, H. Wang, P. Fu, and Z. Luo. R2CNN: Rotational Region CNN for Orientation Robust Scene Text Detection. *CoRR*, abs/1706.09579, 2017. URL http://arxiv.org/abs/1706.09579.

[87] P. Jund, N. Abdo, A. Eitel, and W. Burgard. The Freiburg Groceries Dataset. *CoRR*, abs/1611.05799, 2016. URL http://arxiv.org/abs/1611.05799.

[88] A. Kar, S. Tulsiani, J. Carreira, and J. Malik. Amodal Completion and Size Constancy in Natural Scenes. In *2015 IEEE International Conference on Computer Vision, ICCV*, pages 127–135, 2015. doi:10.1109/ICCV.2015.23.

[89] P. J. Kellman and C. M. Massey. Perceptual learning, cognition, and expertise. In B. H. Ross, editor, *Psychology of Learning and Motivation*, volume 58, pages 117 – 165. Academic Press, 2013. doi:10.1016/B978-0-12-407237-4.00004-9.

[90] A. Kendall and Y. Gal. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 5574–5584, 2017. URL http://papers.nips.cc/paper/7141-what-uncertainties-do-we-need-in-bayesian-deep-learning-for-computer-vision. Accessed: 2021-02-07.

[91] A. Khoreva, R. Benenson, J. H. Hosang, M. Hein, and B. Schiele. Simple Does It: Weakly Supervised Instance and Semantic Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 1665–1674, 2017. doi:10.1109/CVPR.2017.181.

[92] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[93] T. Kong, F. Sun, H. Liu, Y. Jiang, and J. Shi. FoveaBox: Beyond Anchor-based Object Detector. *CoRR*, abs/1904.03797, 2019. URL http://arxiv.org/abs/1904.03797.

[94] D. Koubaroulis, J. Matas, and J. Kittler. Evaluating Colour-Based Object Recognition Algorithms Using the SOIL-47 Database. In *Asian Conference on Computer Vision, ACCV*, pages 840–845, 2002.

[95] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012*, pages 1106–1114, 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[96] A. Krogh and J. A. Hertz. A Simple Weight Decay Can Improve Generalization. In J. E. Moody, S. J. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems, NIPS*, volume 4, pages 950–957, 1991. URL http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.

[97] W. Kuo, A. Angelova, J. Malik, and T. Lin. ShapeMask: Learning to Segment Novel Objects by Refining Shape Priors. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV*, pages 9206–9215, 2019. doi:10.1109/ICCV.2019.00930.

[98] A. Kuznetsova, H. Rom, N. Alldrin, J. R. R. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig, and V. Ferrari. The open images dataset V4: unified image classification, object detection, and visual relationship detection at scale. *CoRR*, abs/1811.00982, 2018. URL http://arxiv.org/abs/1811.00982.

[99] D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys. TI-POOLING: Transformation-Invariant Pooling for Feature Learning in Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 289–297, 2016. doi:10.1109/CVPR.2016.38.

[100] H. Larochelle, D. Erhan, A. C. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In Z. Ghahramani, editor, *Machine Learning, Proceedings of the Twenty-Fourth International Conference, ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 473–480. ACM, 2007. doi:10.1145/1273496.1273556.

[101] H. Law and J. Deng. CornerNet: Detecting Objects as Paired Keypoints. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Proceedings, Part XIV*, volume 11218 of *Lecture Notes in Computer Science*, pages 765–781. Springer, 2018. doi:10.1007/978-3-030-01264-9_45.

[102] H. Law, Y. Teng, O. Russakovsky, and J. Deng. Cornernet-lite: Efficient keypoint based object detection. In *31st British Machine Vision Conference 2020, BMVC*, 2020. URL https://www.bmvc2020-conference.com/assets/papers/0016.pdf. Accessed: 2021-02-07.

[103] Y. LeCun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in perspective*, volume 19, pages 143–155. Elsevier Amsterdam, 1989.

[104] Y. LeCun and Y. Bengio. Convolutional Networks for Images, Speech, and Time Series. In *The Handbook of Brain Theory and Neural Networks*, pages 255–258. MIT Press, 1998.

[105] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi:10.1109/5.726791.

[106] Y. LeCun, F. J. Huang, and L. Bottou. Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting. In *2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR*, pages 97–104, 2004. doi:10.1109/CVPR.2004.144.

[107] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi:10.1038/nature14539.

[108] S. Lehar. Gestalt isomorphism and the quantification of spatial perception. *Gestalt theory*, 21:122–139, 1999. URL http://gestalttheory.net/archive/lehar.pdf. Accessed: 2021-02-07.

[109] B. Leibe, A. Leonardis, and B. Schiele. An Implicit Shape Model for Combined Object Categorization and Segmentation. In J. Ponce, M. Hebert, C. Schmid, and A. Zisserman, editors, *Toward Category-Level Object Recognition*, pages 508–524. Springer, 2006. doi:10.1007/11957959_26.

[110] K. Lenc and A. Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 991–999, 2015. doi:10.1109/CVPR.2015.7298701.

[111] V. Lepetit, P. Lagger, and P. Fua. Randomized Trees for Real-Time Keypoint Recognition. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, pages 775–781, 2005. doi:10.1109/CVPR.2005.288.

[112] H. Li, H. Lu, Z. L. Lin, X. Shen, and B. L. Price. Inner and Inter Label Propagation: Salient Object Detection in the Wild. *IEEE Trans. Image Process.*, 24(10):3176–3186, 2015. doi:10.1109/TIP.2015.2440174.

[113] J. Li, X. Liang, Y. Wei, T. Xu, J. Feng, and S. Yan. Perceptual Generative Adversarial Networks for Small Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 1951–1959, 2017. doi:10.1109/CVPR.2017.211.

[114] K. Li and J. Malik. Amodal Instance Segmentation. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Proceedings, Part II*, volume 9906 of *Lecture Notes in Computer Science*, pages 677–693. Springer, 2016. doi:10.1007/978-3-319-46475-6_42.

[115] K. Li, B. Hariharan, and J. Malik. Iterative Instance Segmentation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3659–3667, 2016. doi:10.1109/CVPR.2016.398.

[116] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei. Fully Convolutional Instance-Aware Semantic Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4438–4446, 2017. doi:10.1109/CVPR.2017.472.

[117] M. Lin, Q. Chen, and S. Yan. Network In Network. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.4400.

[118] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common Objects in Context. In D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Proceedings, Part V*, volume 8693 of *Lecture Notes in Computer Science*, pages 740–755. Springer, 2014. doi:10.1007/978-3-319-10602-1_48.

[119] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature Pyramid Networks for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 936–944, 2017. doi:10.1109/CVPR.2017.106.

[120] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal Loss for Dense Object Detection. In *IEEE International Conference on Computer Vision, ICCV 2017*, pages 2999–3007, 2017. doi:10.1109/ICCV.2017.324.

[121] L. Liu, Z. Pan, and B. Lei. Learning a Rotation Invariant Detector with Rotatable Bounding Box. *CoRR*, abs/1711.09405, 2017. URL http://arxiv.org/abs/1711.09405.

[122] L. Liu, W. Ouyang, X. Wang, P. W. Fieguth, J. Chen, X. Liu, and M. Pietikäinen. Deep Learning for Generic Object Detection: A Survey. *Int. J. Comput. Vis.*, 128(2): 261–318, 2020. doi:10.1007/s11263-019-01247-4.

[123] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path Aggregation Network for Instance Segmentation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 8759–8768, 2018. doi:10.1109/CVPR.2018.00913.

[124] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: Single Shot MultiBox Detector. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Proceedings, Part I*, volume 9905 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2016. doi:10.1007/978-3-319-46448-0_2.

[125] Z. Liu, W. Zou, and O. L. Meur. Saliency Tree: A Novel Saliency Detection Framework. *IEEE Trans. Image Process.*, 23(5):1937–1952, 2014. doi:10.1109/TIP.2014.2307434.

[126] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3431–3440, 2015. doi:10.1109/CVPR.2015.7298965.

[127] D. G. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision, ICCV*, pages 1150–1157, 1999. doi:10.1109/ICCV.1999.790410.

[128] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vis.*, 60(2):91–110, 2004. doi:10.1023/B:VISI.0000029664.99615.94.

[129] J. Ma, W. Shao, H. Ye, L. Wang, H. Wang, Y. Zheng, and X. Xue. Arbitrary-Oriented Scene Text Detection via Rotation Proposals. *IEEE Trans. Multim.*, 20(11):3111–3122, 2018. doi:10.1109/TMM.2018.2818020.

[130] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013. URL https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf. Accessed: 2021-02-07.

[131] D. R. Magee and R. D. Boyle. Detecting lameness using 'Re-sampling Condensation' and 'multi-stream cyclic hidden Markov models'. *Image Vis. Comput.*, 20(8):581–594, 2002. doi:10.1016/S0262-8856(02)00047-1.

[132] D. Marcos, M. Volpi, N. Komodakis, and D. Tuia. Rotation Equivariant Vector Field Networks. In *IEEE International Conference on Computer Vision, ICCV*, pages 5058–5067, 2017. doi:10.1109/ICCV.2017.540.

[133] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon. A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955. *AI Mag.*, 27(4):12–14, 2006. doi:10.1609/aimag.v27i4.1904.

[134] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943. doi:10.1007/BF02478259.

[135] M. Merler, C. Galleguillos, and S. J. Belongie. Recognizing Groceries in situ Using in vitro Training Data. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR*, 2007. doi:10.1109/CVPR.2007.383486.

[136] M. Minervini, A. Fischbach, H. Scharr, and S. A. Tsaftaris. Finely-grained annotated datasets for image-based plant phenotyping. *Pattern Recognit. Lett.*, 81:80–89, 2016. doi:10.1016/j.patrec.2015.10.013.

[137] D. Misra. Mish: A Self Regularized Non-Monotonic Neural Activation Function. *CoRR*, abs/1908.08681, 2019. URL http://arxiv.org/abs/1908.08681.

[138] J. E. Moody. Note on generalization, regularization and architecture selection in nonlinear learning systems. In *Neural Networks for Signal Processing, Proceedings of the 1991 IEEE Workshop*, pages 1–10, 1991. doi:10.1109/NNSP.1991.239541.

[139] C. Müller and N. Kutzbach. *World Robotics 2020 — Industrial Robots*. IFR Statistical Department, VDMA Services GmbH, Frankfurt am Main, Germany, 2020.

[140] C. Müller, B. Graf, K. Pfeiffer, S. Bieller, N. Kutzbach, and K. Röhricht. *World Robotics 2020 — Service Robots*. IFR Statistical Department, VDMA Services GmbH, Frankfurt am Main, Germany, 2020.

[141] MVTec Software GmbH. HALCON/HDevelop Operator Reference, Version 18.11, 2018. URL https://www.mvtec.com/products/halcon.

[142] MVTec Software GmbH. HALCON/HDevelop Operator Reference, Version 19.05, 2019. URL https://www.mvtec.com/products/halcon.

[143] MVTec Software GmbH. HALCON/HDevelop Operator Reference, Version 20.11, 2020. URL https://www.mvtec.com/products/halcon.

[144] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning, ICML*, pages 807–814. Omnipress, 2010. URL https://icml.cc/Conferences/2010/papers/432.pdf. Accessed: 2021-02-07.

[145] G. Neuhold, T. Ollmann, S. R. Bulò, and P. Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *IEEE International Conference on Computer Vision, ICCV*, pages 5000–5009, 2017. doi:10.1109/ICCV.2017.534.

[146] D. Nistér and H. Stewénius. Scalable Recognition with a Vocabulary Tree. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR*, pages 2161–2168, 2006. doi:10.1109/CVPR.2006.264.

[147] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):971–987, 2002. doi:10.1109/TPAMI.2002.1017623.

[148] K. Oksuz, B. C. Cam, E. Akbas, and S. Kalkan. Localization Recall Precision (LRP): A New Performance Metric for Object Detection. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Proceedings, Part VII*, volume 11211 of *Lecture Notes in Computer Science*, pages 521–537. Springer, 2018. doi:10.1007/978-3-030-01234-2_31.

[149] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. doi:10.1109/TSMC.1979.4310076.

[150] B. T. Phong. Illumination for Computer Generated Pictures. *Commun. ACM*, 18(6): 311–317, 1975. doi:10.1145/360825.360839.

[151] P. H. O. Pinheiro, R. Collobert, and P. Dollár. Learning to Segment Object Candidates. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems*, pages 1990–1998, 2015. URL http://papers.nips.cc/paper/5852-learning-to-segment-object-candidates. Accessed: 2021-02-07.

[152] P. O. Pinheiro, T. Lin, R. Collobert, and P. Dollár. Learning to Refine Object Segments. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Proceedings, Part I*, volume 9905 of *Lecture Notes in Computer Science*, pages 75–91. Springer, 2016. doi:10.1007/978-3-319-46448-0_5.

[153] B. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. doi:10.1016/0041-5553(64)90137-5.

[154] B. Radig and P. Follmann. Training a Classifier for Automatic Flash Detection in Million Images from Camera–Traps. In *International Conference on Pattern Recognition and Artificial Intelligence, (ICPRAI)*, volume 12, pages 589–591, 2018.

[155] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for Activation Functions. In *6th International Conference on Learning Representations, ICLR, Workshop Track Proceedings*, 2018. URL https://openreview.net/forum?id=Hkuq2EkPf. Accessed: 2021-02-07.

[156] J. Redmon and A. Farhadi. YOLO9000: Better, Faster, Stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 6517–6525, 2017. doi:10.1109/CVPR.2017.690.

[157] J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement. *CoRR*, abs/1804.02767, 2018. URL http://arxiv.org/abs/1804.02767.

[158] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 779–788, 2016. doi:10.1109/CVPR.2016.91.

[159] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149, 2017. doi:10.1109/TPAMI.2016.2577031.

[160] A. Rocha, D. C. Hauagge, J. Wainer, and S. Goldenstein. Automatic fruit and vegetable classification from images. *Computers and Electronics in Agriculture*, 70(1): 96–104, 2010. doi:10.1016/j.compag.2009.09.002.

[161] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W. M. W. III, and A. F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference, Proceedings, Part III*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015. doi:10.1007/978-3-319-24574-4_28.

[162] F. Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[163] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004. doi:10.1145/1015706.1015720.

[164] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski. ORB: An efficient alternative to SIFT or SURF. In D. N. Metaxas, L. Quan, A. Sanfeliu, and L. V. Gool, editors, *IEEE International Conference on Computer Vision, ICCV*, pages 2564–2571, 2011. doi:10.1109/ICCV.2011.6126544.

[165] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL http://arxiv.org/abs/1609.04747.

[166] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi:10.1038/323533a0.

[167] O. Russakovsky, J. Deng, Z. Huang, A. C. Berg, and F. Li. Detecting Avocados to Zucchinis: What Have We Done, and Where Are We Going? In *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pages 2064–2071, 2013. doi:10.1109/ICCV.2013.258.

[168] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015. doi:10.1007/s11263-015-0816-y.

[169] B. Schölkopf and A. J. Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning series. MIT Press, 2002.

[170] M. I. Shamos. Computational geometry. *Ph. D. thesis, Yale University*, 1978. URL http://euro.ecom.cmu.edu/people/faculty/mshamos/1978ShamosThesis.pdf. Accessed: 2021-02-07.

[171] A. Shrivastava, A. Gupta, and R. B. Girshick. Training Region-Based Object Detectors with Online Hard Example Mining. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 761–769, 2016. doi:10.1109/CVPR.2016.89.

[172] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from Simulated and Unsupervised Images through Adversarial Training. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 2242–2251, 2017. doi:10.1109/CVPR.2017.241.

[173] B. Siciliano and O. Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008. doi:10.1007/978-3-540-30301-5.

[174] C. Siegfarth, T. Voegtle, and C. Fabinski. Comparison of two methods for 2D pose estimation of industrial workpieces in images - CNN vs. Classical image processing system. *International Archives of the Photogrammetry, Remote Sensing &*

*Spatial Information Sciences*, 42(1):401–405, 2018. doi:10.5194/isprs-archives-XLII-1-401-2018.

[175] N. Silberman, L. Shapira, R. Gal, and P. Kohli. A Contour Completion Model for Augmenting Surface Reconstructions. In D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Proceedings, Part III*, volume 8691 of *Lecture Notes in Computer Science*, pages 488–503. Springer, 2014. doi:10.1007/978-3-319-10578-9_32.

[176] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *7th International Conference on Document Analysis and Recognition ICDAR, 2-Volume Set*, pages 958–962, 2003. doi:10.1109/ICDAR.2003.1227801.

[177] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

[178] C. Steger. Similarity measures for occlusion, clutter, and illumination invariant object recognition. In B. Radig and S. Florczyk, editors, *Pattern Recognition, 23rd DAGM-Symposium, Proceedings*, volume 2191 of *Lecture Notes in Computer Science*, pages 148–154. Springer, 2001. doi:10.1007/3-540-45404-7_20.

[179] C. Steger. Occlusion, clutter, and illumination invariant object recognition. In *International Archives of Photogrammetry and Remote Sensing*, volume XXXIV, part 3A, pages 345–350, 2002.

[180] C. Steger, M. Ulrich, and C. Wiedemann. *Machine Vision Algorithms and Applications*. Wiley-VCH, Weinheim, 2nd edition, 2018. ISBN 978-3-527-41365-2.

[181] P. Stone, R. Brooks, E. Brynjolfsson, R. Calo, O. Etzioni, G. Hager, J. Hirschberg, S. Kalyanakrishnan, E. Kamar, S. Kraus, et al. Artificial Intelligence and Life in 2030. *One Hundred Year Study on Artificial Intelligence: Report of the 2015-2016 Study Panel*, page 52, 2016. URL https://ai100.stanford.edu/2016-report. Accessed: 2021-02-07.

[182] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning, ICML*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1139–1147. PMLR, 2013. URL http://proceedings.mlr.press/v28/sutskever13.html. Accessed: 2021-02-07.

[183] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 1–9, 2015. doi:10.1109/CVPR.2015.7298594.

[184] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 2818–2826, 2016. doi:10.1109/CVPR.2016.308.

[185] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In S. P. Singh and S. Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 4278–4284. AAAI Press, 2017. URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14806. Accessed: 2021-02-07.

[186] M. Tan and Q. V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019. URL http://proceedings.mlr.press/v97/tan19a.html. Accessed: 2021-02-07.

[187] M. Tan, R. Pang, and Q. V. Le. EfficientDet: Scalable and Efficient Object Detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 10778–10787. IEEE, 2020. doi:10.1109/CVPR42600.2020.01079.

[188] Z. Tian, C. Shen, H. Chen, and T. He. FCOS: Fully Convolutional One-Stage Object Detection. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV*, pages 9626–9635, 2019. doi:10.1109/ICCV.2019.00972.

[189] G. T. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE MELECON*, volume 83, page A10, 1983. URL http://www-cgrl.cs.mcgill.ca/~godfried/research/calipers.html. Accessed: 2021-02-07.

[190] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective Search for Object Recognition. *Int. J. Comput. Vis.*, 104(2):154–171, 2013. doi:10.1007/s11263-013-0620-5.

[191] M. Ulrich. *Hierarchical Real-Time Recognition of Compound Objects in Images*, volume 569 of *Reihe C*. Deutsche Geodätische Kommission bei der Bayerischen Akademie der Wissenschaften, München, 2003.

[192] M. Ulrich and C. Steger. Empirical performance evaluation of object recognition methods. In H. I. Christensen and P. J. Phillips, editors, *Empirical Evaluation Methods in Computer Vision*, pages 62–76, Los Alamitos, CA, 2001. IEEE Computer Society Press.

[193] M. Ulrich and C. Steger. Performance comparison of 2D object recognition techniques. In *International Archives of Photogrammetry and Remote Sensing*, volume XXXIV, part 3A, pages 368–374, 2002.

[194] M. Ulrich, P. Follmann, and J.-H. Neudeck. A comparison of shape-based matching with deep-learning-based object detection. *tm-Technisches Messen*, 86(11):685–698, 2019. doi:10.1515/teme-2019-0076.

[195] A. Vezhnevets, V. Ferrari, and J. M. Buhmann. Weakly supervised semantic segmentation with a multi-image model. In *IEEE International Conference on Computer Vision, ICCV*, pages 643–650, 2011. doi:10.1109/ICCV.2011.6126299.

[196] M. Weber, M. Welling, and P. Perona. Towards Automatic Discovery of Object Categories. In *2000 Conference on Computer Vision and Pattern Recognition, CVPR*, page 2101, 2000. doi:10.1109/CVPR.2000.854754.

[197] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1974.

[198] B. Widrow and M. E. Hoff. Adaptive Switching Circuits. Technical report, Stanford University California, Stanford Electronics Laboratories, 1960. URL https://apps.dtic.mil/dtic/tr/fulltext/u2/241531.pdf. Accessed: 2021-02-07.

[199] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic Networks: Deep Translation and Rotation Equivariance. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 7168–7177, 2017. doi:10.1109/CVPR.2017.758.

[200] G. Xia, X. Bai, J. Ding, Z. Zhu, S. J. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang. DOTA: A Large-Scale Dataset for Object Detection in Aerial Images. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3974–3983, 2018. doi:10.1109/CVPR.2018.00418.

[201] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated Residual Transformations for Deep Neural Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 5987–5995, 2017. doi:10.1109/CVPR.2017.634.

[202] X. Yang, H. Sun, K. Fu, J. Yang, X. Sun, M. Yan, and Z. Guo. Automatic Ship Detection in Remote Sensing Images from Google Earth of Complex Scenes Based on Multiscale Rotation Dense Feature Pyramid Networks. *Remote. Sens.*, 10(1):132, 2018. doi:10.3390/rs10010132.

[203] Y. Yang, S. Hallman, D. Ramanan, and C. C. Fowlkes. Layered Object Models for Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(9):1731–1743, 2012. doi:10.1109/TPAMI.2011.208.

[204] S. Zagoruyko, A. Lerer, T. Lin, P. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár. A MultiPath Network for Object Detection. In R. C. Wilson, E. R. Hancock, and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference 2016, BMVC*. BMVA Press, 2016. URL http://www.bmva.org/bmvc/2016/papers/paper015/index.html.

[205] M. D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701, 2012. URL http://arxiv.org/abs/1212.5701.

[206] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Proceedings, Part I*, volume 8689 of *Lecture Notes in Computer Science*, pages 818–833. Springer, 2014. doi:10.1007/978-3-319-10590-1_53.

[207] H. Zhang, T. Xu, and H. Li. StackGAN: Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks. In *IEEE International Conference on Computer Vision, ICCV*, pages 5908–5916, 2017. doi:10.1109/ICCV.2017.629.

[208] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid Scene Parsing Network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 6230–6239, 2017. doi:10.1109/CVPR.2017.660.

[209] B. Zhou, À. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning Deep Features for Scene Recognition using Places Database. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*, pages 487–495, 2014. URL http://papers.nips.cc/paper/5349-learning-deep-features-for-scene-recognition-using-places-database.

[210] B. Zhou, A. Khosla, À. Lapedriza, A. Torralba, and A. Oliva. Places: An image database for deep scene understanding. *CoRR*, abs/1610.02055, 2016. URL http://arxiv.org/abs/1610.02055.

[211] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene Parsing through ADE20K Dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 5122–5130, 2017. doi:10.1109/CVPR.2017.544.

[212] X. Zhou, J. Zhuo, and P. Krähenbühl. Bottom-Up Object Detection by Grouping Extreme and Center Points. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 850–859, 2019. doi:10.1109/CVPR.2019.00094.

[213] Y. Zhou, Q. Ye, Q. Qiu, and J. Jiao. Oriented response networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4961–4970, 2017. doi:10.1109/CVPR.2017.527.

[214] C. Zhu, Y. He, and M. Savvides. Feature Selective Anchor-Free Module for Single-Shot Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 840–849, 2019. doi:10.1109/CVPR.2019.00093.

[215] Y. Zhu, Y. Tian, D. N. Metaxas, and P. Dollár. Semantic amodal segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3001–3009, 2017. doi:10.1109/CVPR.2017.320.

[216] C. L. Zitnick and P. Dollár. Edge Boxes: Locating Object Proposals from Edges. In D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision - ECCV*

*2014 - 13th European Conference, Proceedings, Part V*, volume 8693 of *Lecture Notes in Computer Science*, pages 391–405. Springer, 2014. doi:10.1007/978-3-319-10602-1_26.

[217] Z. Zou, Z. Shi, Y. Guo, and J. Ye. Object Detection in 20 Years: A Survey. *CoRR*, abs/1905.05055, 2019. URL http://arxiv.org/abs/1905.05055.