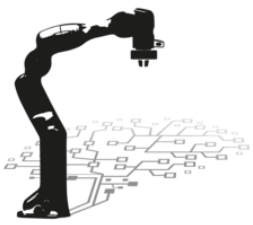
ELECTRICAL ENGINEERING AND
INFORMATION TECHNOLOGY

TECHNICAL UNIVERSITY OF MUNICH
MUNICH SCHOOL OF ROBOTICS AND MACHINE INTELLIGENCE

Master Thesis

Investigating the Use of Depth in Deep Visual Object Trackers

Maximilian Zimmer



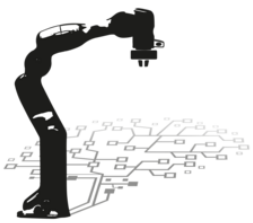
ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

TECHNICAL UNIVERSITY OF MUNICH
MUNICH SCHOOL OF ROBOTICS AND MACHINE INTELLIGENCE

Master Thesis

Investigating the Use of Depth in Deep Visual Object Trackers

Author:	Maximilian Zimmer
Supervisor:	Prof. Dr.-Ing. Sami Haddadin
Advisor:	M.Sc. Elie Aljalbout
Submission Date:	15.09.2020



I confirm that this master thesis is my own work and I have documented all sources and material used.

Munich, 15.09.2020 Maximilian Zimmer

Acknowledgments

I want to thank the Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften (LRZ) for giving me access to the NVIDIA DGX-1 supercomputer. Specifically, I want to express my thanks to Dr. Ludwig Maier, who enabled me to use the system efficiently by taking care of my requirements. In addition, I want to thank Elie Aljalbout for the valuable input and feedback during our meetings.

Contents

1. Introduction

Generic visual object tracking is among the most challenging tasks in the field of computer vision. Given the location of an object in the first video frame, the task of generic object tracking is to estimate its position in all subsequent frames and encapsulate it with a rectangular 2D bounding box. Thereby, trackers should not be susceptible to illumination variation, occlusion, out-of-view, rotation, deformation, scale variation, fast motion, background clutters, and other changes in object appearance [88]. It is related to a wide range of applications such as video surveillance, autonomous driving, pilotless drones that perform flight manoeuvres around persons or objects, robot manipulation etc. The focus of this work is generic object tracking, where the objects that will be tracked are not known in advance. While early methods in visual object tracking made use of hand-engineered features, the recent development has shown a clear trend towards deep Convolutional Neural Networks (CNNs), which is also the focus of this work. Visual object tracking generally operates on RGB data. Today, several depth sensors which capture the distance of pixels from the camera have become available, e.g. Microsoft Kinect, Intel RealSense or LIDAR sensors. Also, modern computer vision libraries allow to calculate the distances from multi-camera setups. As depth information by definition provides spatial information, incorporating depth into deep visual object trackers might have even more potential than standard RGB tracking. As there have not been thorough analyses of how to profit from depth information, this thesis aims to get a deeper understanding about what might work, serving as a base for the tracking community for further work on RGBD (4 channels: red (R), green (G), blue (B), and depth (D)) tracking. The recent publication of the VOT2019-RGBD [48] dataset provides a corresponding dataset and is an important milestone to attract the attention of the tracking community towards tracking with depth information. This work focuses on long-term single object tracking to comply with the videos of the VOT dataset. Generally, tracking splits into single and multi-object tracking, with single object tracking splitting further into short-term and long-term tracking. The major difference is that in long-term tracking an object may leave the field of view.

Following this introduction, developments in the field of generic visual object tracking with and without depth information are reviewed in chapter 2. The algorithm that serves as a base for the investigations with depth information is presented in chapter 3. Chapter 4 thoroughly explains the conducted experiments and the outcomes. In addition to the classic supervised learning methods, section 4.8 analyses the potential of unsupervised training methods. Results are summarized in chapter 5, before concluding the thesis and proposing ideas for future work in chapter 6.

2. Related Work

As the objective of this thesis is to investigate the use of depth information in deep visual object trackers, a good knowledge about the state of the art methods in tracking as well as approaches to use depth information in other computer vision tasks is mandatory. The following sections cover the latest developments in both research areas, providing a broad overview of existing algorithms. This work demands to combine these findings and propose a novel framework to tackle the challenge of RGBD tracking. The selection of the presented RGB trackers tries to outline the milestones in the field of tracking on the one hand and present modern approaches on the other hand. The focus is hereby mostly on real-time trackers as a secondary objective is to obtain a real-time RGBD tracker. Many of them participated in the VOT2018-RT [45] challenge, which explicitly demands trackers to process frames with a rate of at least 20 frames per second. At the time of writing, this has been the latest real-time tracking challenge with all papers describing the trackers being available. Furthermore, existing RGBD trackers at the time of writing are discussed. The scope of this thesis is generic object tracking, thus the class of objects is not known beforehand and tracking architectures have to extract features that are beneficial for tracking of many objects on the one hand, while discriminating against other objects at the same time.

2.1. Visual Object Tracking

Visual object tracking is a fundamental and challenging task in computer vision, which has gained much attention in research in the last years. The approaches that modern trackers in the domain of generic object tracking build on roughly divide into template-matching methods and tracking-by-detection [54].

The idea of template-matching methods is to match a template of the target object to different regions in the new image frame. Then, some similarity measure defines the new position of the object. template-matching methods have very recently gained much attention with the trend to deep CNNs in other computer vision tasks. While the early approaches tried to manually develop feature extractors that are optimized for a specific tracking environment, a shift towards using neural networks as feature extractors is recognizable. Due to the mentioned challenges in tracking, the definition of a similarity function is not apparent. Therefore, training a network to learn a good matching function based on training data seems to be a reasonable approach. The key concept of all template-matching trackers is that they do not adapt the similarity measure online, resulting in high tracking speed. Many benchmarks such as Visual Object Tracking Challenge (VOT) (VOT2015 [44], VOT2016 [47], VOT2017 [46], VOT2018 [45], VOT2019 [48]), Object Tracking benchmark (OTB50 [89], OTB100 [88]) or GOT-10k [35] have been developed to address the problem of tracking. In addition to the datasets

coherent with the challenges, large-scale tracking datasets as LaSOT [21] recently became available and even better trackers might be developed with the growth of training data. The results of VOT2018 show a clear trend towards template-matching methods when it comes to real-time requirements.

Tracking-by-detection, on the other hand, performs model updates while tracking. Most of these trackers are based on Discriminative Correlation Filters (DCFs). They improve computational efficiency and speed by transforming convolutional operations from time-domain to frequency-domain. Many of these filters use handcrafted features as used in the beginnings of template-matching methods. Again, this makes finding appropriate features challenging, as they should be invariant to occlusion etc. Some solutions make use of CNNs to extract high-dimensional features. However, as this goes along with a reasonable higher computational effort and a radical increase in the model parameters, most of these trackers cannot meet real-time requirements.

2.1.1. Template-Matching Methods

To achieve fast speed and online operability, most template-matching trackers rely on Siamese network architectures [46, 88]. Siamese Instance Search Tracker (SINT) [82] achieved the first remarkable success in this field. It was able to deliver outstanding performance by using a rather simple tracker built on an offline-trained Siamese architecture. Until then, tracking-by-detection algorithms have been state-of-the-art. The significant difference in template-matching is the idea of learning object features that are invariant to disturbing factors without explicitly modeling them. In this work, the focus is generic object tracking. Therefore the matching function does not get trained on a specific object category but on a dataset of videos that generally does not contain the target object. Thus, the target appearance is not learned offline.

SINT. Tao et al. [82] proposed two different networks that were suited to learn the similarity measure. These are variations of AlexNet [50] and VGGNet [77], which are both initialized with the corresponding network parameters from the unmodified ImageNet-pretrained models. According to their results, using the deeper VGGNet significantly improved the tracking results. Figure 2.1 shows the architecture of SINT on a high level. Fewer max-pooling layers shall help to maintain robustness against noise. As the complexity of the target object is unknown, the architecture tries to combine CNN features from different layers. Feature combination takes advantage of the property that CNNs capture simple features in early layers, but build more complex features when the layer depth increases [93]. All features contribute to the margin contrastive loss that Tao et al. adapted in this work. It is contrastive in the way that it assigns low loss for samples from the same category while at the same time making sure to obtain high loss for falsely assuming that samples belong to other categories. Chopra et al. [14] originally introduced it in a face verification setup. For training, sampled boxes are labeled positive if they are overlapping with the ground truth box to some extent, otherwise negative. The concept of this loss summarises as follows: If a search frame contains the target object (is labeled positive), the distance between the corresponding feature representations is minimized. If the search frame is labeled negative, then a variation of the support vector machine (SVM) loss is used, thus accumulating loss if the

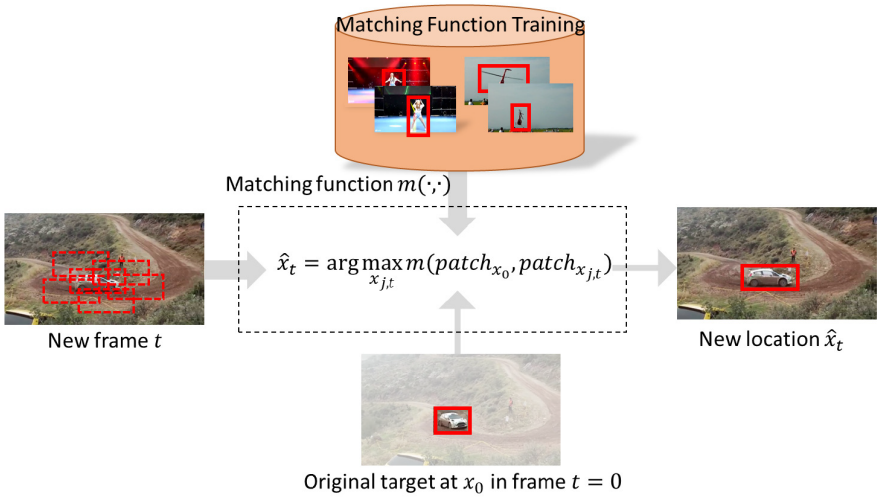


Figure 2.1.: Overview of the SINT architecture. The algorithm starts with a bounding box centered at the target object. Then a pretrained matching function predicts the new location of the object at the next image frame by evaluating the best match on different candidate boxes (dashed boxes in the left image) [82].

distance of the two representations is smaller than some margin.

SiamFC. Another milestone was the publication of the Siamese Fully-Convolutional (SiamFC) tracker [3]. This tracker differs from SINT as the siamese architecture is fully convolutional and therefore has a higher throughput. It uses the template as correlation filter for a candidate image, which has a higher computational efficiency. Besides, the network is trained from scratch and does not finetune a pretrained classification network. Bertinetto et al. [3] applied a logistic loss to train the network. It has been the only tracker that achieved frame-rate speed at the VOT2015 [44] challenge. According to the authors, such a learning approach from scratch only became possible because of the emergence of the ILSVRC (ImageNet Video) dataset [74] for object detection in video. Before that, datasets that allowed training a full network have not been available. The network processes two input patches:

- exemplar image z: first appearance of the target object in a $127 \times 127 \times 3$ patch
- candidate image x: patch of size $255 \times 255 \times 3$ in which similarities with z are checked

The fully convolutional neural network φ extracts characteristic image features. It is important to note that both images get processed by the same network with the same weights, which is the definition of a siamese network. This produces feature vectors of size $6 \times 6 \times 128$ and $22 \times 22 \times 128$, respectively. The central idea with SiamFC is adding a cross-correlation layer, which applies a convolution of the two feature vectors. The output is a score map of size $17 \times 17 \times 1$ with a single convolution, being simple and efficient. Figure 2.2 shows the full architecture of the SiamFC tracker . φ is similar to the convolutional stage of AlexNet [50]. Noticeably, the CNN does not use padding. Because of that, the network is fully-convolutional, stating that it commutes with translation. This

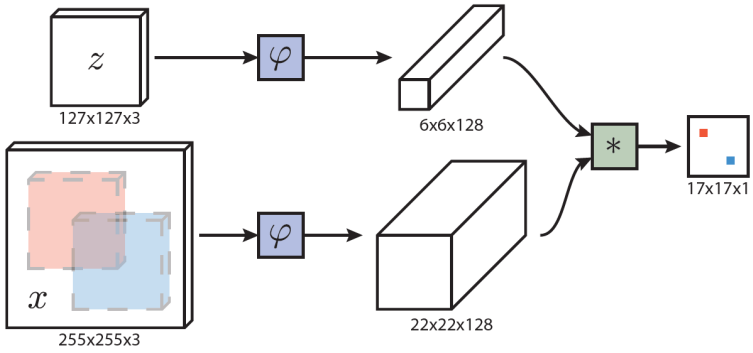


Figure 2.2.: SiamFC architecture [3].

property allows the search image to be larger than the candidate image, which leads to a similarity score at all translated sub-windows on a dense grid (score map). SiamFC does only search for the object in a region of approximately four times its previous size. A cosine window is applied to the score map to penalize large displacements. In practice, several scaled versions of the search image are used to cope with scale variation, while any change in scale is penalized.

SiamVGG, SA-Siam. SiamVGG [57] and SA-Siam [28] are based on SiamFC. SiamVGG modifies the baseline SiamFC method by using a variant of VGGNet [77] as a backbone network for feature extraction. The deeper network shall tackle weak discrimination capability. The general architecture is identical to SiamFC (see Figure 2.2), with φ being the modified VGGNet. SiamVGG considers the finding in Tao et al. [82], which states that the deeper network improves tracking performance over AlexNet-based architectures. Thus, by using a modified version of the convolutional part of VGG-16, more detailed semantic features shall improve tracking performance. The final tracker outperforms SiamFC. However, it has been trained on ImageNet Video [74] as well as Youtube-BB [72], which was not available when SiamFC was published. Furthermore, the network was partly pretrained on ImageNet. Using even deeper architectures as ResNet [29] or GoogLeNet [81] has not been possible as these architectures need to use padding, which violates the property of being fully-convolutional. SA-Siam [28] takes the SiamFC architecture but considers two feature extractors for each path in Figure 2.2. Thus, the candidate image and exemplar image have two feature representations each. The idea is that one feature extractor is focused on semantic (S) features, while the other one is specialized to appearance (A). Both branches use AlexNet-like networks. The semantic branch is pretrained on ImageNet, while the appearance branch is trained from scratch. This learning scheme shall encourage the networks to extract complementary image features, which shall improve tracking metrics when combined correctly.

SiamRPN, DaSiamRPN. Siamese Region Proposal Network (SiamRPN) [54] gets rid of testing multi-scale images by extending SiamFC with a Region Proposal Network (RPN). SiamRPN transfers the idea of Faster R-CNN [73] to the area of object tracking. This further improves the speed and online-capability of siamese trackers. The architecture consists of a siamese subnetwork for feature extraction and a region proposal subnetwork, which includes a classification and a bounding box regression branch. Figure 2.3 shows

the framework. The RPN uses the loss function of Faster R-CNN, which is composed of a cross-entropy classification loss as well as a smooth L_1 regression loss. For real-time tracking, the framework performs one-shot learning, which refers to just taking the first frame as input. That allows precomputing the detection kernel after training the network. The training was performed on the ImageNet Video and Youtube-BB datasets. Li et al. found out that their tracker performance was increasing when considering more training videos. Hence, they concluded that even bigger datasets might further improve the tracker. The feature extraction network is again based on AlexNet [50], pretrained on ImageNet. The first three convolutional layers are fixed; only the deeper layers are fine-tuned. The Distractor-aware Siamese Region Proposal

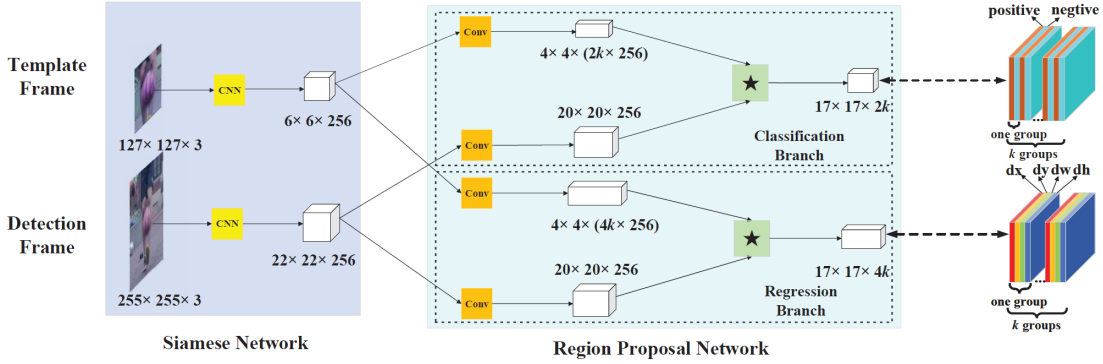


Figure 2.3.: The framework of SiamRPN uses a siamese subnetwork for feature extraction as well as RPN. The RPN consists of two branches, thus handling foreground/background classification and bounding box regression. [54]

Network (DaSiamRPN) [97] tracker is mainly a modification of SiamRPN. It improves the discrimination power of SiamRPN, thus tweaking the ability to discriminate foreground from semantic background. This is achieved by enriching the number of training samples. More precisely, the object detection datasets large-scale ImageNet Detection [74] and COCO Detection [58] are utilized to generate image pairs for training the tracker. Apart from that, negative pairs from the same category, as well as negative pairs from different categories are generated, which makes the tracker more robust. Data augmentation additionally includes motion blur. The tracker uses a new distractor-aware objective function, which further improves online performance. When choosing the output bounding box from the response map, distractors influence the final choice. Compared to SiamRPN, the results of Zhu et al. show that their network has better discrimination behavior against distractors of the same category as the target object.

SiamPRN++. SiamPRN++ [53] is an improvement over SiamRPN by Li et al. It investigates the problem of siamese networks not being able to take deep networks such as ResNet [29] as a backbone. The motivation behind this study is the success of deeper networks in other computer vision tasks. SiamRPN++ aims to carry these milestones over to the tracking domain. So far, template-matching based trackers did not use them due to the violation of the fully-convolutional property [3, 28, 54, 57]. SiamRPN++ has been the first architecture that achieved end-to-end learning on a deep siamese network for visual tracking while still performing real-time. It relies on a modified

ResNet as feature extractor. Figure 2.4 shows the architecture. Li et al. figured out that the restriction to networks that fulfill the strict translation invariance property could be repealed by using a spatial aware sampling strategy. Thus, their tracker is shifting patches in data augmentation before feeding them to the network. Shifting is performed by a uniform distribution with a maximum shift value. Higher shift values increased the performance on the VOT2016 [47] and VOT2018 [45] benchmark and allowed using deep architectures. According to the authors, the other major problem that exists in SiamRPN is the parameter imbalance of the feature extraction and the RPN path, which impeds the training process. The reason for this is the use of an up-channel cross-correlation layer in SiamRPN. Therefore, SiamRPN++ introduces a depth-wise cross-correlation layer. It calculates a one-dimensional response map for every channel, reducing the number of parameters by a factor of ten. Figure 2.5 visualizes the different cross-correlation layers.

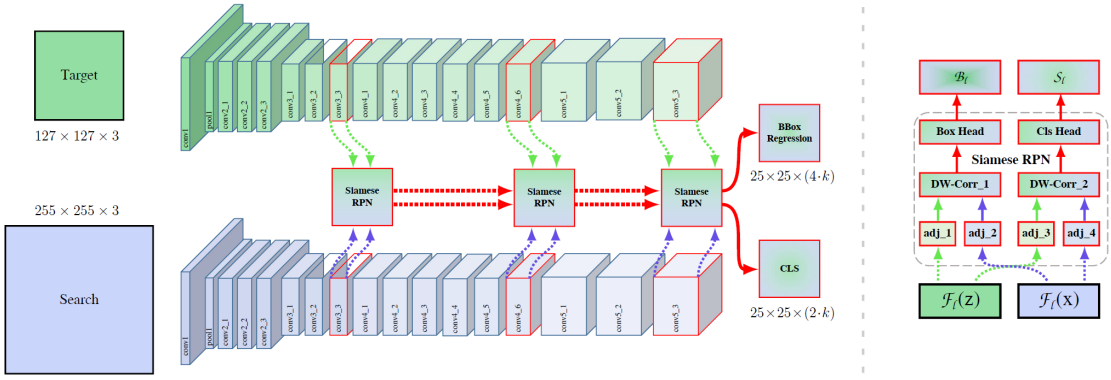
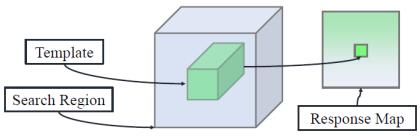
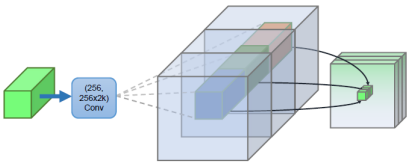


Figure 2.4.: SiamRPN++ architecture which uses a modified ResNet as backbone feature extractor. The network outputs a classification score as well as a bounding box by combining outputs from multiple SiamRPN blocks. The structure of each SiamRPN block is shown on the right. It is similar to the RPN in Figure 2.3 but uses another type of correlation function [53].

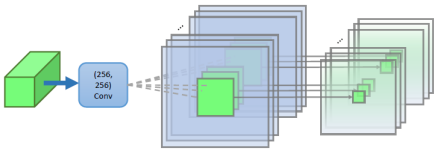
SiamMask, SiamMask_E. In 2018 Wang et al. proposed a framework that combines the tasks of tracking and segmentation, namely SiamMask [85]. This was motivated by the success of siamese networks in visual object tracking and the availability of the Video Object Segmentation dataset (YouTube-VOS) [91], which is a video set with pixel-wise annotations. The tracker also relies on a single bounding-box initialization but produces object segmentation masks when evaluated online. SiamMask has a siamese network structure and can perform real-time tracking. The algorithm has been evaluated with ResNet-like as well as AlexNet [50] architectures. Again, it turned out that deeper networks allow better results. Identical to SiamRPN++, the ResNet network is optimized with a random translation training strategy to overcome the problem of not being fully-convolutional. The best-performing architecture builds upon SiamRPN, which has been extended with a new segmentation branch (see Figure 2.6). The overall loss function is composed of the SiamRPN loss, together with a binary logistic regression loss over all segmentation masks. When applied to real-time tracking, a rotated-minimum bounding rectangle approach generates bounding boxes from the



(a) Cross Correlation Layer



(b) Up-Channel Cross Correlation Layer



(c) Depth-wise Cross Correlation Layer

Figure 2.5.: Visualization of different cross-correlation layers. (a) predicts a single similarity map between a template and a search patch, as in SiamFC. (b) predicts one similarity map for each anchor, as in SiamRPN. (c) predicts depth-wise similarity maps between a template and a search patch, as in SiamRPN++ [53].

segmentation masks. Specific refinement modules are used in the backbone network to generate a more accurate object mask by combining features of different layers. This work revealed that the rotated-minimum bounding rectangle strategy offers a significant advantage over popular strategies that report axis-aligned bounding boxes. Of course, this only applies to challenges that provide rotated ground truth bounding boxes in their datasets. Based on this finding, Chen et al. developed an ellipse fitting method to estimate the bounding box rotation angle and size given a segmentation mask of an object. As SiamMask produces the mask, the tracker is referred to as SiamMask_E [11]. Applying this method is fast and further improved the performance metric of SiamMask by proposing better-matching bounding boxes when evaluated on the VOT challenges.

THOR. Tracking Holistic Object Representations (THOR) [75] is a novel framework published in 2019. It operates on top of any template-matching based tracker, which uses an inner product operation for similarity computation, thus all siamese trackers. THOR improves the robustness and performance of underlying trackers by providing a mechanism to gather representative template representations of the target object, capturing the object's appearance over time. The general concept is not to improve the similarity measure for single templates as the case for other presented methods but instead generating new templates during inference. The framework captures a fixed amount of templates while trying to keep them as diverse as possible. Diversity is achieved by approaching the problem similar to capturing the similarity between an exemplar and a candidate image, thus convolving two templates with each other. Only templates that maximize the diversity of information about the tracked object will be

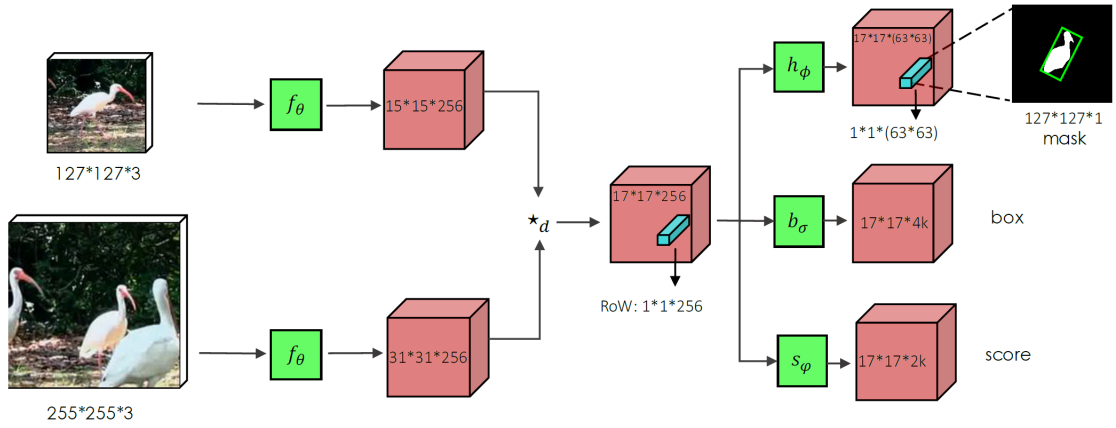


Figure 2.6.: Schematic visualization of the SiamMask architecture based on SiamRPN. $\star_d$ denotes depth-wise cross-correlation. After the features have been correlated, the network splits into three branches. As in SiamRPN, one branch performs foreground/background classification (score), while another one performs bounding box regression (box). In SiamMask, a third branch is introduced for segmentation mask generation (mask) [85].

kept in memory. Specifically, THOR uses two different modules to store templates, a Short-term module (STM) and a Long-term module (LTM). The STM handles abrupt movements and partial occlusion, while the LTM captures variations of the object on a longer time horizon. During inference, a modulation module implements the idea of model ensembles by combining all templates to get one bounding box for STM and LTM, respectively. An ST-LT Switch usually takes the predicted bounding box that builds on the STM but can switch to the LTM prediction if the drift between both predictions is too high. Figure 2.7 summarizes the architecture.

ATOM. A promising milestone in template-matching based trackers was set by Danelljan et al. with their tracker Accurate Tracking by Overlap Maximization (ATOM) [18]. It revisits the main problems of siamese trackers. According to the authors, one problem is not to explicitly include distractors in the framework as existing approaches do not perform online learning. The other problem states that bounding box estimation cannot completely be solved with multi-scale testing, for example, in scenarios like out-of-plane rotation or deformation. Thus, this paper concentrates on distractor handling and bounding box estimation. Specifically, ATOM introduces a classification component that is learned online to guarantee high discriminative power in the presence of distractors. The use of a two-layer CNN obtains real-time performance and prevents overfitting. A conjugate gradient method performs online parameter updates, as this leads to faster convergence. A novel target estimation technique in the field of tracking tackles the second problem. Danelljan et al. utilize the IoU-Net [39], which has been introduced for bounding box estimation in the object detection domain and adapt it to make it suitable for generic object tracking. The original paper [39] proposed to learn one IoU-Net per object category. As not all object categories might be present in the training data, such an approach with object specific components cannot be realized for generic

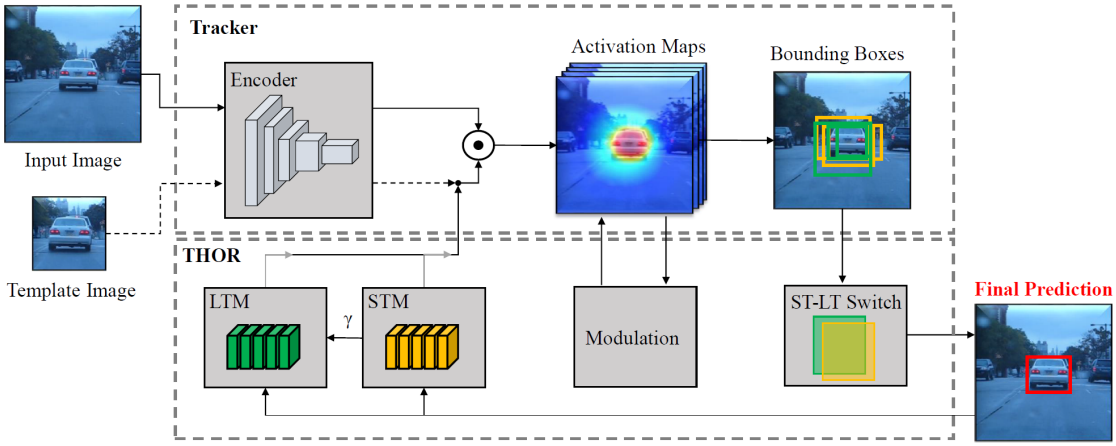


Figure 2.7.: Representation of THOR architecture. A neural network extracts features from the input image and the initial template image. Activation maps are produced by a dot product, which is a convolution operation for siamese networks. THOR accumulates templates in the STM and LTM over time. The modulation module combines the activation maps of both modules. The ST-LT Switch takes the two predictions based on STM and LTM templates and selects which bounding box to use for prediction. Depending on the final prediction, both modules decide whether to capture the new template. γ is a measure of diversity and controls the threshold of the LTM to capture new templates [75].

tracking. A modulation-based network component is introduced in the original network to consider target appearance and therefore making it suitable for generic object tracking. This allows target-specific intersection-over-union (IoU) estimates. During tracking, the bounding box estimate that maximizes the predicted IoU with the target bounding box determines the tracking output. Figure 2.8 shows an overview of the discussed architecture. Although it performs some online learning to incorporate background information, as common for tracking-by-detection frameworks, this architecture belongs to the template-matching methods in the scope of this work. The reason is that ATOM still uses an offline trained similarity function that does not get updated online to extract image features. The feature extracting ResNet uses the same weights for test and template images.

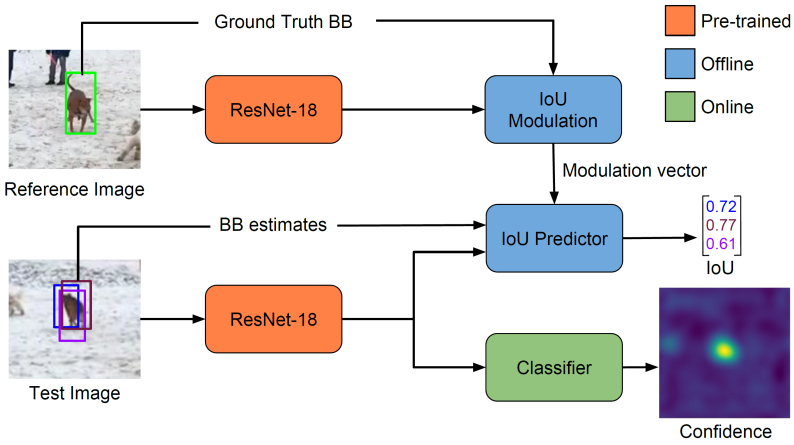


Figure 2.8.: The architecture of ATOM. The framework consists of a feature extraction network (orange), a target estimation module (blue), and a target classification module (green). The feature extraction network is a ResNet architecture pretrained on ImageNet. The target estimation network is trained offline on large-scale datasets, while the target classification module is completely trained online. The classifier outputs confidence scores for the estimated target position. The target estimation network consists of an IoU modulation component and an IoU predictor. The modulator incorporates target-specific information in the offline trained model by performing feature modulation, while the IoU predictor predicts the overlap of estimated bounding boxes with the target [18].

In the following, the presentation of the newly available winners of the VOT2019 [48] challenges complete the survey on template-matching based trackers. All of the presented algorithms until this point have been short-term trackers, stating that they do not need to perform re-detection after losing the target. In the VOT2019-ST and VOT2019-RT challenges, the trackers are reset after drifting. The VOT2019-LT and VOT2019-RGBD challenges require long-term trackers. Thus re-detection is mandatory and trackers do not reset. As the VOT2019-RGBD challenge will be used for the final evaluation of the tracker developed in this work, the requirement is to adopt a long-term tracking approach, ideally with real-time capability. Table 2.1 and Table 2.2 show the top-

performers in the VOT2019-RT and VOT2019-LT challenges, respectively. The Expected Average Overlap (EAO) is used as a primary performance measure for short-term trackers in the VOT challenges. The tracker with the highest EAO score is defined as the best tracker. F-score is the analogous measure for long-term trackers. See chapter 3 for more details. Interestingly, the top three performing trackers in the long-term challenge utilize short-term trackers and extend them with a long-term detector approach [48]. This development shows that advances in short-term tracking are meaningful and have impact on the long-term tracking domain to some extent. All of the trackers in the tables belong to the domain of template-matching methods. Therefore, using template-matching methods as a base for RGBD tracking seems to be appealing. Figure 2.9 visualizes the performance of the discussed trackers on the VOT2018-RT challenge. It is obvious that the trackers have significantly improved within the last years. With background knowledge about the mentioned trackers, there is also a trend of deeper backbone networks obtaining better EAO on this challenge. In addition, the tracking-by-detection architectures that will be discussed in subsection 2.1.2 perform inferior compared to the others, again underlining the importance of template-matching methods in modern visual tracker development. As not all of these reported scores on the VOT2018-RT challenge, Figure 2.9 only contains a selection. Finally, it should be noted that all trackers that participate in the VOT2018-RT challenge meet real-time requirements.

Table 2.1.: Winners of the VOT2019-RT challenge [48].

Ranking	Tracker	EAO
1.	SiamMargin	0.366
2.	SiamFCOT	0.350
3.	DiMP [4]	0.321

Table 2.2.: Winners of the VOT2019-LT challenge [48].

Ranking	Tracker	F-score
1.	LT-DSE	0.695
2.	CLGS	0.674
3.	SiamDW_LT	0.665

DiMP. Discriminative Model Prediction for Tracking (DiMP) [4] focuses on learning a robust target-specific model that incorporates background information while tracking. It builds on the assumption that some target-specific information is only available at inference. Thus, the target model cannot be learned offline. In traditional siamese architectures, the target appearance only affects the model during initialization, more precisely at target patch extraction. Therefore, background information during tracking is not incorporated. DiMP also tackles the problem of extracted features during tracking not being applicable to object categories that have not been present in the training data. The approach of Bhat et al. consists of two branches: a classification branch and a bounding box estimation branch. DiMP reuses the IoU predictor architecture that is used

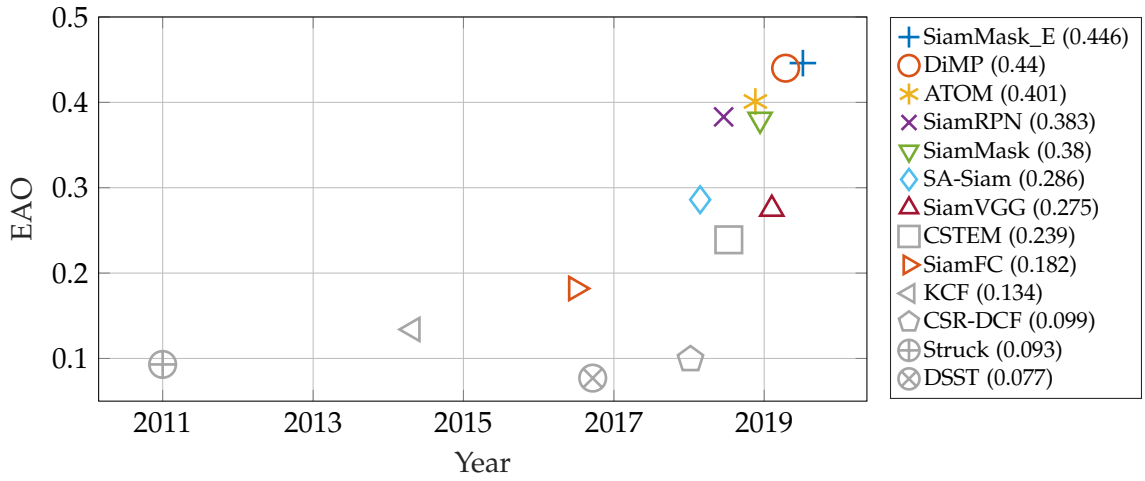


Figure 2.9.: Performance of all discussed trackers on the VOT2018-RT [45] challenge, together with the publishing date of the corresponding paper. Only trackers where results have been published are visualized. The scores are taken from the official challenge results and papers, respectively. Whenever there have been different results for a particular tracker, the better score was taken to consider bugfixes and code optimizations. Trackers that belong to tracking-by-detection are displayed in gray, template-matching architectures in color. Best viewed in color.

in ATOM [18] as the bounding box estimation branch. The classification branch contains a novel model predictor module. The predictor outputs the filter which gets convolved over the extracted features of the test frame. The convolution produces a confidence map, with its maximum value indicating the estimated position of the target object. Figure 2.10 shows the structure, omitting the bounding box branch for clarity. The model optimizer consists of a discriminative loss function and an optimization strategy that allows fast convergence by applying steepest descent, enabling online optimization. The loss function itself consists of a hinge loss in the background region and a least-squares loss in the foreground. In that way, the optimizer does not force the model to learn to predict zero scores in background regions but can also output negative values. The authors conclude that this saves model capacity and encourages the model to be more discriminative. Spatial weighting, target mask, and regularization strength parameters are learned from the data during inference, which is a protruding property of the loss function.

The papers of SiamMargin, Siamese Fully Convolutional Object Tracking (SiamFCOT), long-term tracking by diving videos into successive short episodes (LT-DSE), Complementary Local-Global Search for Robust Long-term Tracking (CLGS) and SiamDW_LT have not been published by the time of writing. However, the VOT2019 [48] results present short descriptions of these trackers. The following shall only give a rough overview of which approach was taken.

SiamMargin. SiamMargin (Chen et al.) builds on SiamRPN++. It improves the handling of distractors by introducing a discrimination loss which is minimized during

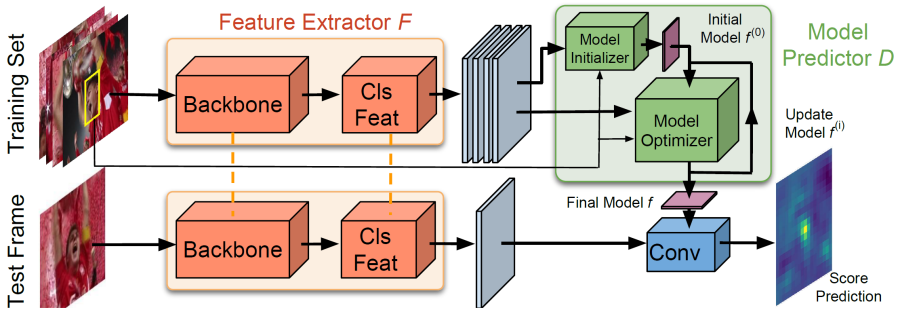


Figure 2.10.: Visualization of the DiMP tracker. The feature extraction network processes a training set and a test frame, respectively. The novel model predictor module consists of an initialization module and the optimizer itself. The model predictor outputs the weights of the convolutional layer, which produces a score prediction map with the extracted features from the test frame. The bounding box branch, which is taken from ATOM [18], is not shown for clarity [4].

offline training. The modified loss function shall put a margin to the decision boundary similar to an SVM embedding. During tracking, SiamMargin updates the template image with a moving average strategy.

SiamFCOT. Different to using anchor-based RPNs, SiamFCOT (Wang et al.) utilizes a bounding box estimation approach from the object detection domain that does not build on anchors. The overall architecture relies on a standard Siamese correlation network with an AlexNet [50] as a backbone architecture. On top of that, Wang et al. adapt Fully Convolutional One-Stage Object Detector (FCOS) [83] such that it can be applied to object tracking. It refines the bounding box obtained by the Siamese correlation network. FCOS is taking pixel locations as training samples rather than anchor boxes. After applying penalizations, the tracker uses the bounding box with the highest score. Again, this tracker utilizes a template-update technique. It is striking that SiamMargin and SiamFCOT both perform template updating, yielding that this might be important for improving performance on real-time trackers.

CLGS. The following trackers belong to the domain of long-term tracking (see Table 2.2). CLGS (Zhao et al.) combines SiamMask [85], R-CNN [8], and an online verifier based on Real-time MDNet (RT-MDNet) [41]. During regular tracking, the short-term tracker SiamMask performs target localization and bounding box estimation. The online verifier detects target loss and activates a global R-CNN detector to generate region proposals on the whole image. Based on these, the verifier will re-detect the target and re-initialize SiamMask.

LT-DSE. The components of LT-DSE (Dai et al.) are somewhat similar to the CLGS architecture as it also uses SiamMask [85] and an online verifier based on RT-MDNet [41]. The short-term tracking part is based on ATOM [18] for target localization and SiamMask for bounding box refinement. The feature extractor network consists of ResNet-18 [29], with two more convolutional layers added on top. The verifier switches to a long-term component whenever the target is detected as disappeared. The long-term component

consists of an RPN with MobileNet [34] as feature extractor network. The network architecture of the RPN is designed based on the MobileNet based tracking by detection algorithm (MBMD) tracker of Zhang et al. [94], which was the winner of the VOT2018-LT [45] challenge. Thus, the main contribution of this paper compared to [94] is a new short-term tracking component. The RPN outputs possible target locations with bounding boxes, while the online verifier processes the proposals and can re-initialize the short-term component if the score of a candidate exceeds a certain threshold. LT-DSE follows the short-term tracker long-term detector approach, dividing tracking sequences into shorter episodes and performing short-term tracking on these.

SiamDW_LT. The SiamDW_LT (Du et al.) architecture is a follow-up work of Zhang et al. [96], which investigated the use of deeper and wider backbone networks in siamese trackers. [96] analyzed the problems that rise with deep networks in object trackers and proposed guidelines to design and adapt deep CNNs to the tracking domain. This brings the correlation of deeper networks and better performance in the object classification domain to generic object tracking. SiamDW_LT consists of a short-term tracker, an online correction module, and a global re-detection module. The short-term tracker implements the design guidelines in [96] but uses an even deeper MobileNetV2 [34] backbone. The short-term module also consists of a proposal refinement module which is similar to the IoU architecture used in ATOM [18]. The proposal refinement module is appended after the RPN to improve the bounding box generation. Finally, the short-term tracker outputs a confidence score to indicate the reliability of the tracking estimate. If the short-term tracker fails, the framework switches to the re-detection module. Finally, model ensemble is utilized to leverage tracking accuracy and robustness.

To sum up the results of the VOT2019-LT challenge, CLGS, LT-DSE and SiamDW_LT all have a very similar architecture. They implement a short-term tracker, a verifier network and a re-detection module. CLGS utilizes SiamMask [85] as short-term tracker. LT-DSE predicts bounding boxes by using ATOM [18] and then performs SiamMask to refine those boxes. SiamDW_LT uses a RPN with a deep architecture and ATOM to refine the results. All of these trackers have a module to detect target loss. CLGS and LT-DSE are using a verifier based on RT-MDNet [41]. For SiamDW_LT the verifier network is unknown by the time of writing. Each of these approaches has a global re-detection module.

2.1.2. Tracking-By-Detection

tracking-by-detection intends to make use of additional information about the object to be tracked that can be gathered online in subsequent frames, assuming correct target identification. The task can be formulated as repeated detection problem in a local search window with an online parameter updating scheme of the classifier [64]. Specifically, correlation filters generate correlation peaks for an object of interest while yielding low responses for background. They have attracted much attention in the beginning of visual object tracking. This trend started with the work of Bolme et al. by introducing the Minimum Output Sum of Squared Error (MOSSE) [7] filter. The most straightforward approach for modeling an object's appearance with a correlation filter is to crop a template from the image. While generating strong peaks for the target, this approach

is prone to respond to the background as well. Generally, correlation filters make use of the convolution theorem. It states that circulant convolution equals to element-wise multiplication in the frequency domain

$$x \otimes h = \mathcal{F}^{-1}(X \odot H^*) \tag{2.1}$$

where $\otimes$ is circulant convolution, $\mathcal{F}^{-1}$ denotes inverse fourier transform operation, uppercase variables indicate variables in the frequency domain and $\odot$ represents element-wise multiplication. Applied to object tracking, the input x would correspond to preprocessed extracted features or a raw image patch while h is the correlation filter.

MOSSE. In the case of MOSSE [7], the input is a raw image patch that is multiplied by a cosine window. Thus, the tracker emphasizes the region near the center of the tracked object while at the same time smoothing the discontinuities at the window boundaries. The target is tracked by correlating the filter over a search region in the next frame, yielding to the new object's position where the correlation output is the highest. To train such a filter, outputs g have to be defined. In MOSSE, g is generated from the ground truth such that it obtains a 2D Gaussian-shaped peak centered on the target in the training image. The difference between the actual output of the convolution and the desired output g forms the error. The sum of squared errors is minimized to obtain the best correlation filter. Formulated in the Fourier domain, this leads to

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2 \tag{2.2}$$

where F_i corresponds to a training image transformed to the Fourier domain with G_i being its desired output counterpart. By solving this optimization problem and formulating it as the closed-form expression

$$H^* = \frac{1}{N} \sum_i \frac{G_i \odot F_i^*}{F_i \odot F_i^*} \tag{2.3}$$

Bolme et al. presented the first computationally efficient DCF-based real-time tracker. Performing the calculations in the Fourier domain reduced the computational complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n \, log \, n)$, considering an n-dimensional feature representation of an input image. For every input frame, evaluation of Equation 2.3 results in a correlation filter that minimizes the error for that frame. For online tracking, MOSSE uses separate moving average filters for the numerator and denominator of Equation 2.3, thus exponentially decaying the effect of previous frames over time. Figure 2.11 summarizes the general workflow of correlation filter based tracking methods.

KCF, DualCF. The continuous work of Henriques et al. [31, 32] improved the MOSSE filter by extending the framework with kernel methods, utilizing the properties of circulant matrices to enhance the processing speed. In order to model translations of an object, they use cyclic shifts of the base sample. Figure 2.12 shows an example of cyclic shifting applied to an image. The effect of wrapped-around edges is not a big problem in practice, as this effect can be mitigated by preprocessing techniques such as applying a cosine window. When looking at cyclic shifts of an image in matrix form, it turns out that the representation is a circulant matrix. Henriques et al. make use of the

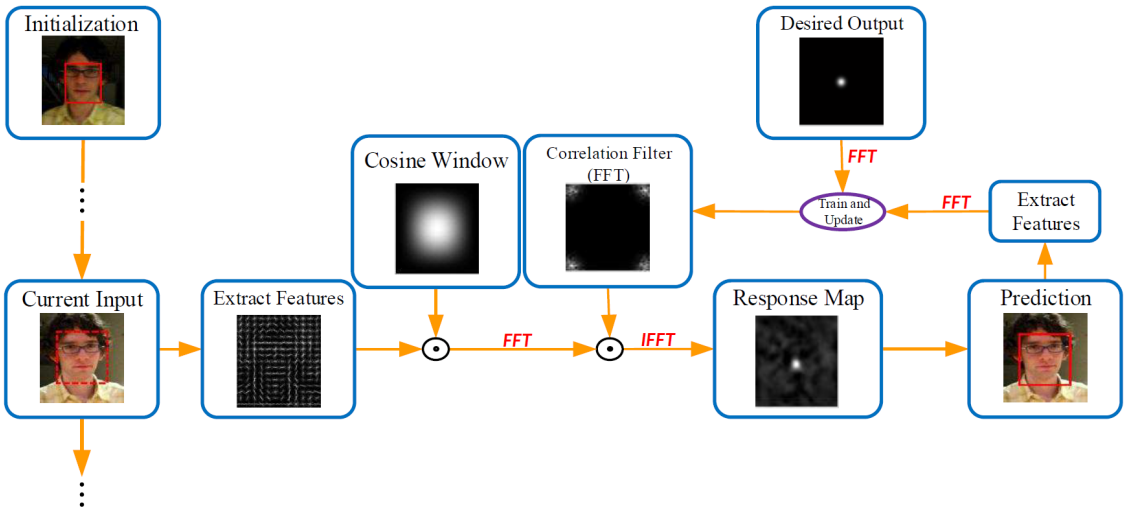


Figure 2.11.: Workflow of correlation filter based tracking methods. For each frame after initialization, the current output is cropped at the previously predicted object position. Depending on the tracker, raw pixel values or more advanced features to better represent the object function as image features. Usually, a cosine window is applied to the features to smoothen the discontinuities at the boundaries before being transformed to the Fourier domain, where the filter is trained and updated. The symbol $\odot$ denotes element-wise multiplication, FFT means Fast Fourier Transform operation while IFFT, namely Inverse Fast Fourier Transformation, is the inverse operation. After updating the filter, the correlation between the current image and the filter is calculated in the Fourier domain. Applying IFFT outputs the response map, whose peak estimates the new position of the target [13].

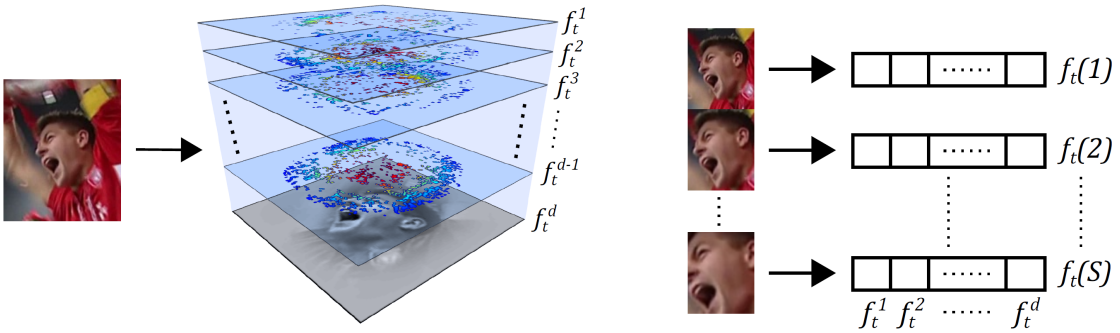
property that all circulant matrices are made diagonal by the Discrete Fourier Transform (DFT). Operations on diagonal matrices can directly be performed element-wise on their diagonal elements. This relation allows effectively training a tracker in the Fourier domain with all possible cyclic shifts of a sample, horizontally and vertically, without iterating over them explicitly. Building on this, they proposed two trackers, namely Dual Correlation Filter (DualCF) and Kernelized Correlation Filter (KCF). KCF makes use of the kernel trick to obtain a more powerful non-linear regression function at the same computational complexity as linear correlation filters. DualCF, on the other hand, uses a linear kernel. Both trackers deliver better performance when using more complex Histogram of Oriented Gradients (HOG) [16] features as image representations rather than raw pixel values.

Figure 2.12.: Example of vertical cyclic shifting of an image. Effects from the wrapped-around edges can be seen; however they are reduced by the multiplication with a cosine window and padding [32].

As feature representation methods greatly influence the performance of trackers [13], several works concentrated on finding better representations than raw pixel values or HOG. Danelljan et al. [17] investigated the use of color features in visual object tracking, which have effectively been used in image classification and detection tasks as well as action recognition. They concluded that using color features does improve performance when combined with luminance. However, a careful choice of which color features to use is crucial to obtain a reasonable performance gain when including it in a tracker. Color Names (CNs) [84] obtained the best results. These are linguistic color labels assigned by humans to represent colors. Using CNs, colors split into eleven color bins, which deal as image features.

SA, MF, SAMF, DSST, fDSST. Li et al. [55] presented the Scale Adaptive with Multiple Features Tracker (SAMF), which uses KCF as a starting point for their investigations. SAMF tries to cope with two problems of DCF trackers: handling scale variations and solely relying on a single feature representation. Thus, SAMF combines raw pixel features with HOG and CN. Li et al. assume that HOG and CN are complementary to each other as HOG concentrates on the image gradient while CN focuses on color information. By combining these, they were able to obtain a performance gain. To be able to handle scale variations of the target object, a set of candidate templates with different scales is resized using bilinear interpolation. Correlating these templates with the filter results in the final response. Finally, they evaluated three algorithms: Scale Adaptive Tracker (SA), Multiple Features Tracker (MF), and SAMF. The latter, which combines the concept of SA and MF, was able to benefit from both approaches, leading to a significant improvement in performance. However, exhaustive scale search strategies as implemented in SAMF

are computationally expensive. Discriminative Scale Space Tracker (DSST) [19] copes with this problem and investigates another approach on scale estimation by learning separate DCFs for translation and scale estimation. The tracker is based on DualCF and improves it by providing a better scale estimation scheme, dispensing with the need for exhaustive search strategies. Similar to other trackers, the maximum score in the response map determines the location of the tracked object in new frames. In addition to this two-dimensional translation filter, Daneljan et al. introduce a one-dimensional scale filter. After obtaining the new position, several samples of different sizes are extracted at the object's position and mapped to d-dimensional feature vectors. The sample with the highest correlation score determines the new target size. An aggregation of these samples is used as a training sample to update the filter, assuming a one-dimensional gaussian as the desired correlation output. Figure 2.13 exemplifies the training sample generation for the translation filter and the scale filter, respectively. They further reduce the computational cost of DSST by sub-grid interpolation of the correlation scores and performing Principal Component Analysis (PCA) for dimensionality reduction. The increased speed of the calculations due to lower complexity allows to enlarge the target search space, thus improving robustness. The extended version of DSST is referred to as fast Discriminative Scale Space Tracker (fDSST).



(a) Translation filter sample. (b) Scale filter sample.

Figure 2.13.: Illustration of training samples used to update DSST. The sample for the translation filter (a) is extracted from a rectangular patch centered at the object's position. The sample for the scale filter (b) is an aggregation of d-dimensional feature vectors of rectangular patches of different scales, sampled at the estimated object's position by applying the translation filter [19].

Ma et al. tracker. The sophisticated results of CNNs in other visual recognition tasks and the performance increase of DCFs that engage multiple feature representations motivated the work of Ma et al. [64]. They have developed a discriminative tracker that uses a pretrained VGGNet as feature extractor. Furthermore, they assumed that the last convolutional layers represent semantics, which are rather invariant to significant appearance variations, while lower layers encode spatial details that are more useful for precise localization. Therefore they proposed layer-wise correlation filters. More precisely, three correlation filters are used, processing the extracted features from the third, fourth and fifth convolutional layer of the VGGNet. The correlation filters then

locate the target object in a coarse-to-fine approach. The reason why this works is the reduction of the spatial resolution when using deeper features in the network due to pooling operations. Bilinear interpolation leads to same-sized features from the feature extractor. Therefore in deeper layers, an object's location can only be determined by some accuracy, encouraging the coarse-to-fine approach. Figure 2.14 demonstrates the algorithm. Ma et al. concluded that using hierarchical convolutional features learned from a large-scale dataset is more effective than conventional hand-crafted features such as HOG or CN. However, the performance increase comes at the cost of a drastic speed reduction as the calculation of CNNs is much more computationally expensive than other features. Another downside is the sensitivity to occlusion, which can deteriorate the performance.

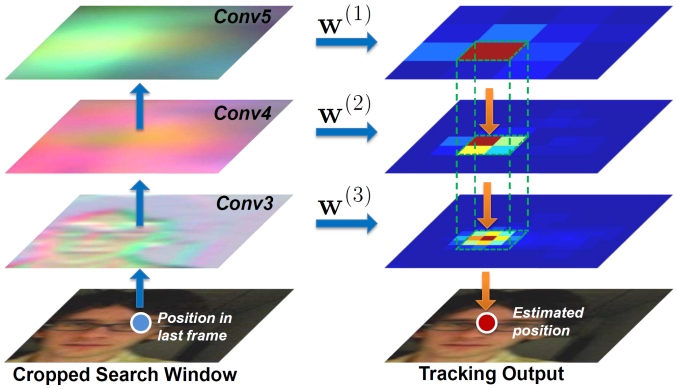


Figure 2.14.: Main steps of the algorithm proposed by Ma et al. A cropped search window is extracted from an image frame centered at the last object's position. Conv3, Conv4, and Conv5 contain the target representations at different hierarchical levels, obtained from the respective convolutional layers in the feature extractor. Each representation is correlated with a filter $\mathbf{w}$ to generate a response map whose location of the maximum value corresponds to a location estimate. The exact position of the target object is obtained by refining the estimates in a coarse-to-fine approach, starting with the deepest layer [64].

CSR-DCF. A quite sophisticated DCF-based tracker has been presented by Lukežič et al. [62] in 2018. Their tracker, Discriminative Correlation Filter with Channel and Spatial Reliability (CSR-DCF), introduces spatial and channel reliability. It uses a spatial prior and color segmentation to construct a spatial reliability map. That adjusts the filter support to the target object. More specifically, the spatial reliability map encourages the filter not to learn background which is essential for irregularly shaped objects. Weights control the impact of different feature channels. Feature channels with high discriminative power are assigned a higher weight. CSR-DCF uses HOG, CN and grayscale channels. The algorithm splits into a localization and an update step. At the localization step, the new position estimate of the target object is obtained by a weighted sum over the response maps of all correlation filter channels. The weights of all channels are referred to as reliability weights. They sum up to one and are affected

by the maximum response value of the appropriate filter and by how uniquely each channel votes for a single target position. The new target location is obtained by the combination of all channel responses. It is then used in the update step to extract the training region, upon which the estimate of the spatial reliability map is updated. The spatial reliability map determines which pixels are labeled as containing the object, thus alleviating the problem of letting the filter learn the background. Finally, the correlation filters are updated channel-wise and new reliability weights are calculated for use in the next frame. Figure 2.15 portrays the procedure.

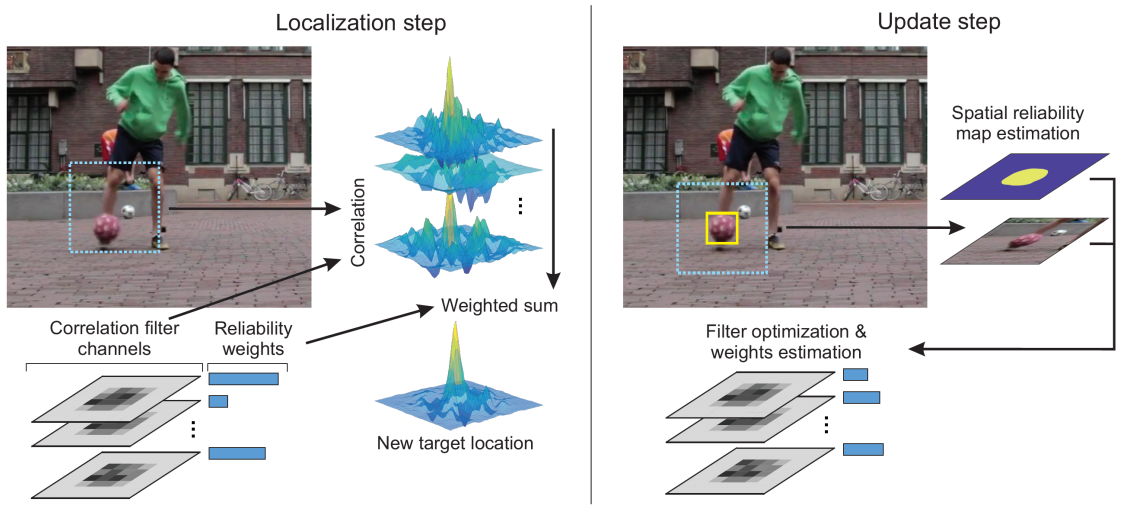


Figure 2.15.: CSR-DCF tracking algorithm, consisting of a localization step (left) and an update step (right). In the localization step, the new position of the target object is estimated based on the weighted sum of the channel-wise correlation filters. In the update step, the spatial reliability map estimate is adjusted, followed by correlation filter and reliability weights updates [62].

CSTEM, RPT. Channels weighted and spatial-related Tracker with Effective response-map Measurement (CSTEM) [95] builds on CSR-DCF. It proposes a tracker that obtains good results even when occlusions occur. Therefore, a new occlusion measure effectively detects such situations. As an extra correlation filter is employed to locate the object while occlusions occur, the position estimate switches between these trackers depending on the occlusion measure. Besides, two scaling approaches are combined to estimate the scale of the target better. Other concepts introduce part-based strategies, such as Liu et al. [59] or Li et al. [56]. These strategies implement several DCFs, which track specific parts of the object, such that detection is still possible if some parts of the target are occluded. The difficulty of part-based trackers is to define a proper mechanism to handle multiple results from different DCFs. In [59], five independent trackers are used to locate different parts of the object. A weighted sum of the individual response maps forms the joint confidence map. Weighting reduces the impact of single trackers if occlusion is probable. Reliable Patch Trackers (RPT) [56] develop a framework that automatically selects parts to track by sampling. It introduces reliable patches, which can be used for tracking. The reliabilities are updated online, such that the tracker only

uses parts that are meaningful for tracking.

Struck. Other than DCF-based methods, some trackers still fall in the category of tracking-by-detection. Structured Output Tracking with Kernels (Struck) [26] is based on structured output prediction. The method uses a kernelized structured output SVM, which updates online to provide adaptive tracking. SVMs have been the superior method in object detection at that time. Hare et al. developed Struck as a tracking framework that benefits from SVMs, obtaining benefits in terms of generalization and robustness. Unlike the algorithms mentioned beforehand, this approach does not contain a classification step. Thus, unlike other algorithms, no samples around the last estimated target position are labeled and then used to update the classifier and therefore, this integrates learning and tracking. Interestingly, Struck allowed the use of different image feature representations by appropriately modifying the kernel function. As stated in other trackers [55, 62], Hare et al. postulated that different feature representations are often complementary and can improve performance when adequately combined.

Re^3. Real-time Recurrent Regression-based tracker (Re^3) [22] is a real-time deep object tracker that is capable of incorporating temporal information into its model. It prioritizes offline training on a large-scale dataset and limits online adaption to recurrent state updates. Using a Recurrent Neural Network (RNN), Gordon et al. are utilizing previous frames to predict the object's position in a new video frame. More detailed, at each timestep, two image crops centered at the last position estimate of the target are fed into a pretrained CNN, which serves as a feature extractor. The outputs from the CNN are stacked and fed into a fully connected layer and an RNN, specifically a Long Short-Term Memory (LSTM) [33] network. The outputs of the LSTM lead into a fully connected layer that regresses the upper left and lower right corner of the output bounding box. Figure 2.16 demonstrates the network architecture. The network parameters of the CNN are not updated at test-time in order to obtain real-time performance. Thus, no retraining on the network is performed. However, relevant object information is collected during tracking and gets stored in the recurrent parameters, allowing online adaption at low computational cost.

The performance on the OTB100 [88] dataset is evaluated to provide some measure of comparison between the presented tracking-by-detection frameworks. The choice of this specific dataset results from the fact that it is the one where most of the trackers published their performance. Evaluation on the OTB100 depends on the area under curve (AUC) of the precision and success plot. The precision plot maps the percentage of correct object locations to given pixel thresholds. Similarly, the success plot correlates the IoU of predicted frames with the ground truth bounding boxes over all IoU thresholds. The AUC is then simply the area under the respective curve. Thus, a good tracker should have a high precision AUC and a high success AUC at the same time. Figure 2.17 shows the results of the presented trackers on the OTB100 dataset. It shows a trend towards better performance with more sophisticated architectures. However, the best performing ones again follow the template-matching approach.

As a summary of all trackers in subsection 2.1.1 and 2.1.2, Figure 2.18 claims to visualize the dependencies among them. Most of the tracking-by-detection frameworks discussed in the scope of this work built on handcrafted features rather than using CNNs. This is mostly due to the fact that online updating of deep networks is not

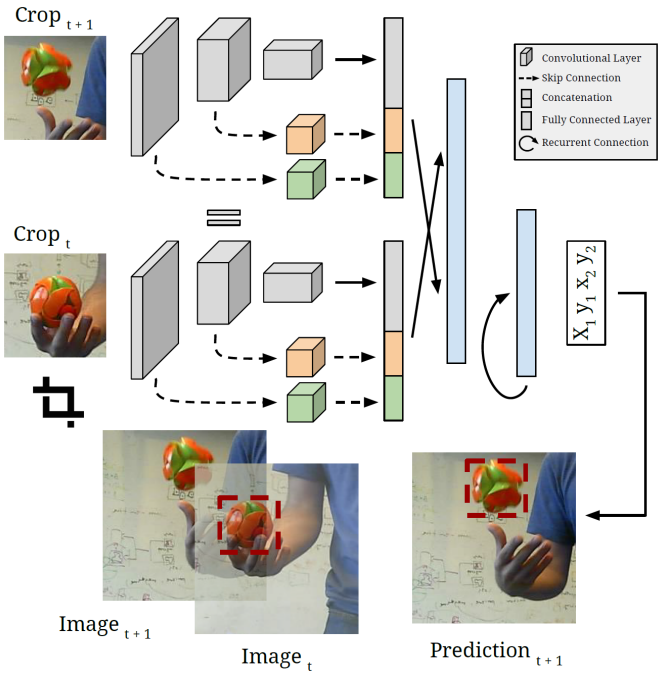


Figure 2.16.: Architecture of Re^3. Two image crops are fed into the feature extractor CNN. Before every pooling operation, skip connections preserve high-resolution spatial information. The output of the network leads into a fully connected layer and an LSTM. The outputs of the network are the new coordinates of the upper left and lower right corner of the bounding box estimate [22].

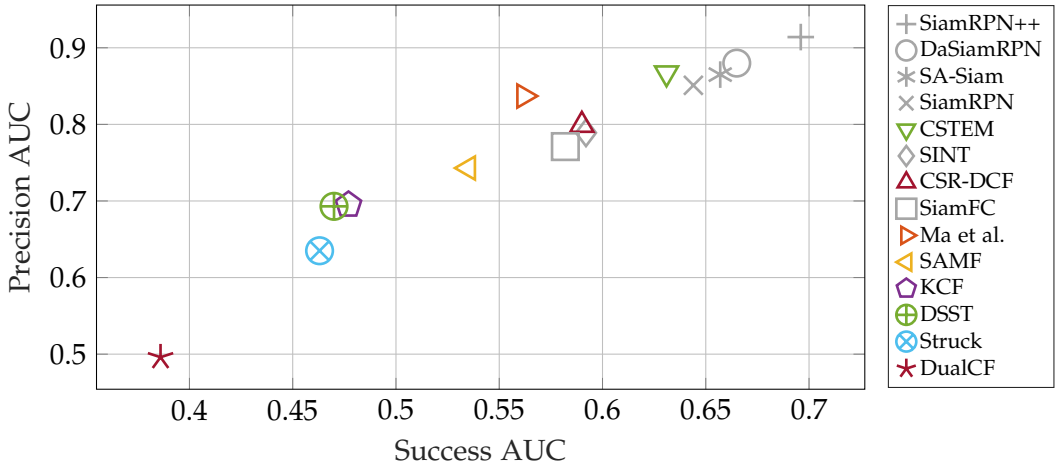


Figure 2.17.: Performance of presented trackers on the OTB100 [88] dataset. Each tracker is plotted with its specific precision and success AUC values. In addition to the architectures that belong to tracking-by-detection, template-matching based trackers are also shown in gray wherever results have been available. The figure shows that the best performances are obtained by template-matching trackers. Best viewed in color.

possible due to performance issues. On the contrary, template-matching methods rely on a rich feature representations. Figure 2.18 shows a clear trend from relatively small backbones as AlexNet to deeper networks like ResNet or MobileNet. This reflects the development in other computer vision tasks. The top-performing long-term trackers in the VOT2019 [48] challenge seem to extend state-of-the-art short-term trackers for enabling re-detection whenever they detect target loss.

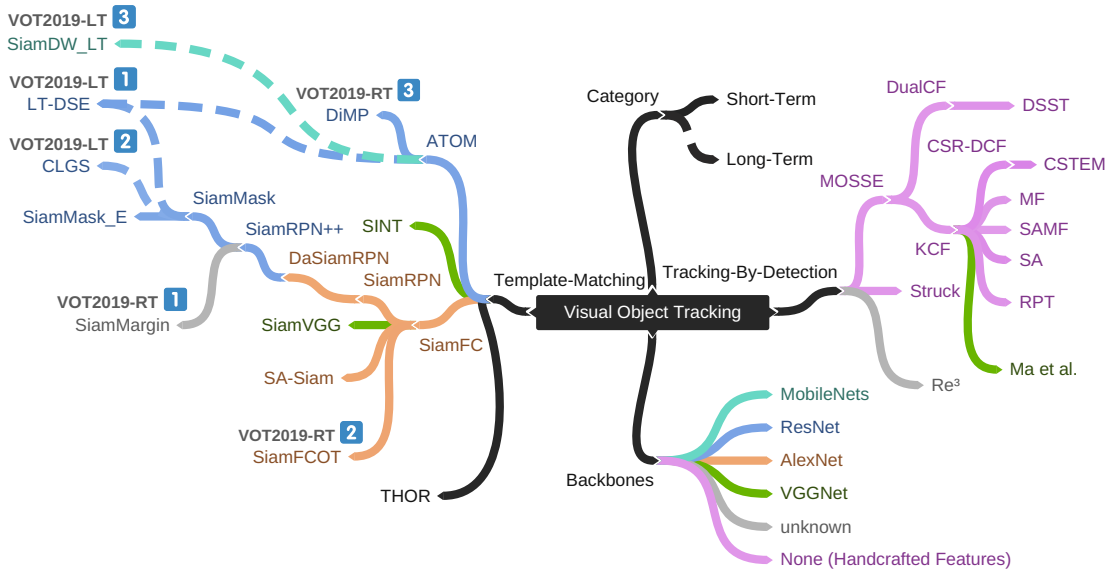


Figure 2.18.: Overview of all discussed visual object trackers and their dependencies. The approaches divide into template-matching methods and tracking-by-detection. The colors represent the backbone network that is used for feature extraction in each framework. Because modern long-term trackers build on short-term trackers with global re-detection strategies, most of the presented trackers are short-term trackers. However, the top three long-term trackers of the VOT2019-LT [48] are also portrayed with dashed lines. For long-term trackers, the color represents the feature extractor backbone of the underlying short-term tracker. Although this illustration only includes a selection of approaches, it coarsely represents the evolution of state-of-the-art trackers. The best-performing architectures rely on CNNs, with a clear trend towards deeper backbones like ResNet [29] or MobileNet [34]. On the other hand, tracking-by-detection approaches mostly rely on handcrafted features and often have inferior performance compared to architectures that profit from deep features.The top three performers on the VOT2019-LT as well as the VOT2019-RT challenge are annotated. Best viewed in color.

2.2. Using Depth Information in Computer Vision Applications

The rise of depth cameras, which can capture distances from targets to the camera lens, triggered investigations on how to profit from this spatial information when performing

different computer vision tasks. The property that depth data contains information about the object's shape that is invariant to lighting or color variation seems appealing for integration in existing frameworks.

Eitel et al. [20] proposed a framework to perform RGBD object detection by making use of CNNs. Specifically, they use two separate networks that are combined with a late fusion network consisting of a fully connected and a softmax layer. For feature extraction, a ZF-Net [93] architecture is used, which is an improvement over AlexNet by hyperparameter tuning. The two streams process color and depth information, respectively. Figure 2.19 shows the architecture. Both network streams are pretrained on ImageNet due to the lack of large-scale RGBD datasets that could have been used for training the network from scratch. The absence of large datasets as present in the field of object recognition might as well be a challenge in the development of an RGBD tracker in the scope of this work. The concept for processing depth information with CNNs applied by Eitel et al. summarizes as follows: A depth image is encoded as RGB image, spreading the data over all three channels.

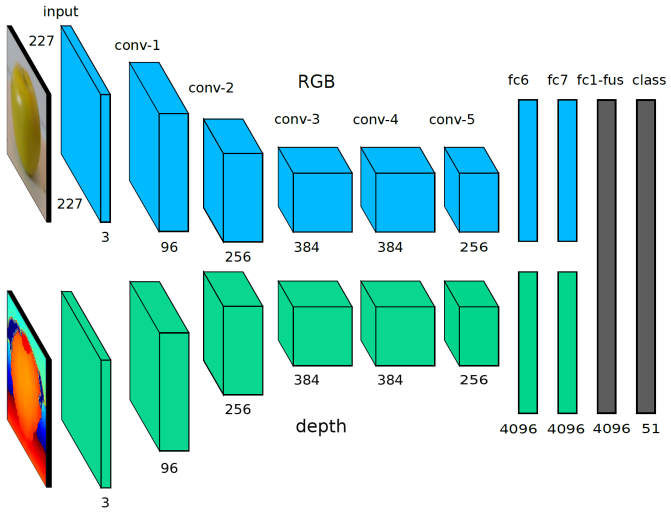


Figure 2.19.: Two-stream CNN architecture that Gupta et al. use for RGBD object detection. The two inputs are the RGB color image for the first and a depth encoded color image for the second stream. The first stream resembles an RGB approach, while the second stream solely relies on the depth information. Combining the outputs with a late fusion network allows weighting the importance of the two modalities [20].

There are different possibilities for encoding depth information, such as depth-gray encoding [20], surface normals [6], HHA encoding [24] or depth-jet encoding [20]. HHA encodes the depth information with three channels at each pixel. These are horizontal disparity, height above ground and the angle the pixel's local surface normal spans with the gravity direction [24]. It was published in 2014 and comes into effect in several approaches in handling RGBD data [2, 24, 27, 60, 80]. Eitel et al. observed superior behavior of their depth-jet encoding technique over HHA [20]. All of these encoding strategies utilize different methods for rendering color images in order to be able to

use CNNs with depth information. Another method was introduced by Zia et al. [98], who present an architecture that combines three different models: the standard 2D VGGNet, VGG3D, and 3DCNN. VGG3D is a modified version of the standard 2D VGGNet, whose first layer is replaced in order to allow 3D input images. 3DCNN is a smaller architecture compared to VGGNet. The VGGNet and VGG3D are pretrained on ImageNet, while 3DCNN trains from scratch. It also takes 3D input images, but just contains two convolutional layers. The 3D input images are obtained by preprocessing the depth information to produce a 3D voxel representation, combining RGB and depth information. At first, a third dimension is introduced based on the depth values present in the image. This generates a 3D image-sized voxel grid. Subsequently, the RGB pixels are placed at the same position in the voxel grid as in the original image, but at their corresponding depth. Figure 2.20 shows an example. In that way, features that are related to depth and color can jointly be learned rather than treating both types of features separately. The general problem with CNNs that take 3D representations as input is the high memory and computational overhead compared to standard 2D CNNs [86].

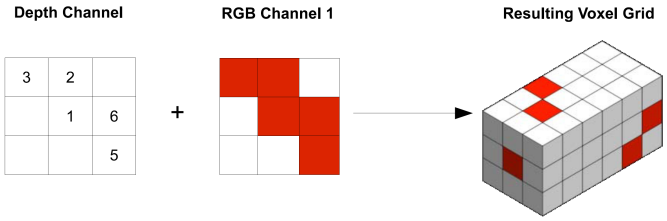


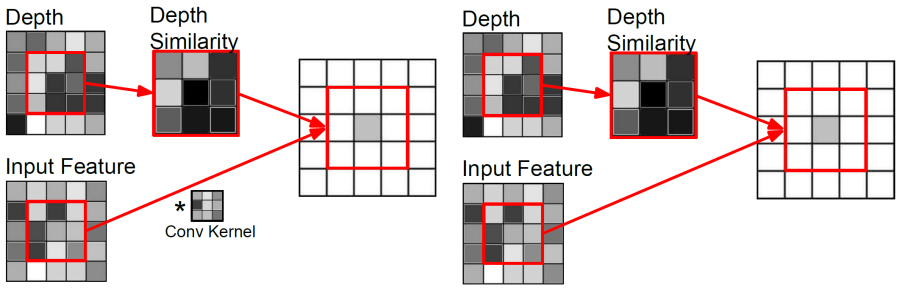
Figure 2.20.: Generation of 3D representations from depth and RGB information that are used as inputs in VGG3D and 3DCNN, exemplary for a 3×3 input image. Accordingly, a $3 \times 3 \times 6$ voxel is produced as the depth values are quantized into 6 intervals [98].

In 2013, Couprie et al. [15] were the first to address multi-class segmentation with RGBD inputs with a feature learning approach. They did not rely on hand-crafted features as it was usual by that time. Couprie et al. consider depth as an additional input channel on top of color channels and expect the convolutional layers to extract appropriate features. They state that using depth information can help to improve recognition of objects that have mostly similar depth appearance and location. However, according to the authors, their approach performed better when only using RGB information for classes that contain high depth-variability. Referring to recent trends, this supports the observations that using more sophisticated methods of including depth information in existing frameworks delivers better results.

FuseNet. FuseNet [27] attracted attention by incorporating depth information in CNNs to solve the semantic segmentation problem on RGBD images. It consists of an encoder-decoder network where the encoder is composed of two branches. One branch extracts features from the RGB image, while the other one extracts depth features. The innovative part is the fusion mechanism. In FuseNet, performing element-wise summation with feature maps of the depth-branch enhances the RGB feature maps. A straightforward approach would be to stack the depth information and the RGB data to

a four-channel input. In contrast, Hazirbas et al. [27] claim that applying their fusion technique obtains feature maps that are more discriminant and therefore better-suited for segmentation tasks. Compared to Eitel et al. [20], who also obtain a two-stream CNN architecture to extract depth features, FuseNet only takes a 1D grayscale depth image as input for the network. Both branches use parameters of a pretrained VGGNet. The weights of the first layer are averaged along the channels for the depth branch, thus allowing to use the pretrained weights even with a 1D input.

In 2018, Wang et al. [86] addressed the problem of doubled network parameters and computational cost with architectures that construct two CNNs to process RGB and depth data separately. Therefore an efficient way of bringing spatial information into CNNs is introduced by two operations: depth-aware convolution and depth-aware average pooling. The concept of both operations is to emphasize pixels that have similar depth while reducing the impact of other pixels at the same time. Both layers consist of their 2D counterparts. However, the result is weighted by some depth similarity measure, thus constructing a depth-aware receptive field. Figure 2.21 illustrates the calculus applied in both layers. Replacement of standard convolutional and average pooling layers with their depth-aware variations in existing networks allows exploiting depth information while obtaining the efficiency of CNNs. Furthermore, no additional parameters are introduced in the network. Wang et al. observed the performance of the CNN layers in a segmentation background. Nevertheless, the authors claim that this framework might be useful for any computer vision task when considering RGBD inputs.



(a) Depth-aware Convolution (b) Depth-aware Average Pooling

Figure 2.21.: Illustration of the depth-aware convolution layer and the depth-aware average pooling layer that have been introduced by Wang et al. The concept is demonstrated with a 3×3 convolutional filter for a single channel. For the depth similarity maps, darker colors represent higher similarity with the pixel in the center of the filter's receptive field. (a) performs convolution between the input feature map and the convolutional kernel, element-wise multiplying the result with the depth similarity map. Likewise, (b) performs average pooling on the input feature map, weighted with the depth similarity [86].

Porzi et al. [71] introduced a depth-aware CNN for pixel-level classification in the context of pose estimation. Specifically, they introduce a CNN layer that adapts the size of the convolutional filters depending on the depth. In that way, areas of the

image corresponding to objects that are far away are convolved with small filters, while areas corresponding to close objects convolve with bigger filters. Implementation-wise, convolutions are performed at different scales. The result is a weighted sum over these feature maps, with the features being dependant on the depth value. The general idea is that each scale is best-suited for one distance from the camera. Their architecture improved compared to the same architecture using a standard convolutional layer when applying this concept.

Depth information is also vulnerable for scene recognition, which combines the tasks of semantic segmentation and detection as well as pose estimation. The overall goal is to identify a complete scene and classify it accordingly. The common practice in this field when including depth information was to apply transfer learning from RGB CNNs due to the lack of RGBD datasets. Rethinking this habit, Song et al. [80] observed that learning from scratch may obtain better results. Their studies use HHA [24] encoding for the depth channel as input for the CNN. The main finding is that RGB and depth images have significantly different low-level visual patterns. This fact turned out to be a significant drawback when only finetuning a network that was pretrained on RGB data. Finetuning does generally not affect the early layers in the network. According to their findings, this results in a large number of low-level features not being activated when using HHA encoded features on RGB pretrained networks. To cope with the small datasets, Song et al. designed a network with fewer parameters that can learn depth-specific features while training from scratch.

2.3. Advances in RGBD Tracking

In the meanwhile, there has already been some progress on tracking with RGBD images. The following tries to present existing approaches. The selection of the RGBD trackers is inspired by results on the Princeton Tracking Benchmark [79] and the newer VOT2019-RGBD [48] challenge. The Princeton Benchmark was published in 2014, the participating trackers in the years 2015 to 2017. The presentation of these trackers shall function as overview of what approaches have been taken in considering depth information in trackers in the past. Though, these trackers have not competed in the VOT2019-RGBD challenge. In total, six trackers that incorporate depth information have been published in the scope of the VOT2019-RGBD challenge, with only four being able to outperform baseline RGB-only trackers. As the results of the VOT2019 challenges have just been published at the time of writing, only a few papers are already available. Thus, not all trackers are described in detail. However, the official VOT2019 [48] challenge results include short descriptions of all trackers.

OAPF. Occlusion-Aware Particle Filter (OAPF) [68] was the first RGBD tracker that directly included an occlusion state in the formulation of a tracker. As many RGB trackers have difficulties in dealing with occlusions, this approach concentrates explicitly on such situations and presents an occlusion recovery mechanism. Hand-crafted features are used to process the depth channel, which then get fused with the features from the color image. OAPF is based on a particle filter. Particle filters are composed of a prediction and an update step. In the prediction step, the new target position in the

current frame is estimated based on samples (particles) from a posterior distribution. The update step updates the posterior according to the new observations. Long-term occlusions have to deal with drastic changes in size, motion direction, distance from the camera, and orientation. Meshgi et al. intend to profit from the fact that the motion model used in the particle filter can help to re-detect the target object. For each particle, the algorithm checks whether it is occluded. An occlusion flag is set if the number of occluded particles exceeds the number of non-occluded ones. Setting the flag stops model updating, and the motion model of the particles changes to a random walk pattern. Hence, the particles scatter from the last known position of the target, and the search area increases with the number of frames in which occlusion is present according to that measure. If a particle is similar to the target object, that particle replicates such that fast recovering from the occlusion state is possible and allows continuing the regular tracking strategy. Combining multiple feature representations in their experiments, namely HOG, Histogram of Texture [70], 3D Shape Parameters [40], Template of Edges [36], Histogram of Depth and Adaptive Histogram of Colors [67], obtained the best performance. Introducing a particular observation confidence mask further improved the tracker. Many trackers utilize a kernel to emphasize pixels in the middle of the search region. Meshgi et al. propose to assign weights to each cell in a regular grid based on the occlusion probability distribution of their approach. They expect further improvement when weighting the features by their discriminative power.

3D-T. Bibi et al. [5] published another particle filter based RGBD tracker. Their 3D Part-Based Sparse Tracker with Automatic Synchronization and Registration (3D-T) is novel in the sense that the motion and appearance model are both formulated in 3D. Hence, 3D-T utilizes depth information directly in the particle representation. The particle filter samples 3D cuboid candidates for representing the target object. Each particle is represented with CN and 3D shape features. 3D-T is a part-based tracker consisting of one centered cuboid to represent holistic object representations and eight cuboids that only concentrate on specific parts of the object. Their experiments showed that assigning multiple parts to an object is beneficial and provides a more robust representation of the target object which increases the overall tracking performance. Similar to OAPF [68], the motion model is dependant on occlusion but with a different occlusion detection paradigm. The object is assumed to be occluded whenever the average depth-normalized number of 3D points per cuboid drops below some threshold. Occlusion leads to an enlarged standard deviation in the particle sampling of the motion model, which allows re-detection after occlusion states. The motion of the particles consider 3D rotations and translations. Unlike in the 2D case, scale estimation is not necessary as the scale of the target in 3D is hardly changing. 3D-T is currently the best RGBD tracker on the Princeton [79] challenge, according to their website[1]. Bibi et al. introduced methods to correct synchronization and registration noise in the dataset because of inferior performance due to some videos suffering from these problems. Camplani et al. [9] also worked on correcting the dataset. However, they did not publish an automatic correction scheme, as implemented in the scope of 3D-T. These problems could be a general issue with this dataset. Thus the comparison to other trackers might

[1] `http://tracking.cs.princeton.edu/eval.php`

not be fair. Eventually, other trackers would perform better when using the preprocessed data as well. According to the authors, however, the problem is mostly relevant for trackers that perform representation sampling and tracking in 3D.

DS-KCF. Camplani et al. [9] published a KCF-based (see subsection 2.1.2) RGBD algorithm called Depth Scaling Kernelized Correlation Filter (DS-KCF). The tracker improves its RGB counterpart with better scale and occlusion handling by using depth information. It still obtains real-time performance. Figure 2.22 shows the block diagram. HOG features are extracted from the depth and RGB channel and serve as input for the KCF. A depth segmentation map is constructed by performing K-means clustering followed by Connected Component Analysis to refine the obtained Regions of Interest (ROIs). It serves as a basis for scale analysis and occlusion handling. The scale of the object's template adapts according to the object's depth compared with its initial depth. By updating the template size in the Fourier domain, convolutions in the spatial domain turn into element-wise operations. Camplani et al. assume a gaussian depth distribution of the tracked target. Regions that do not belong to this model are considered as occluders. During occlusion, the occluder is tracked and a local search strategy recovers from occlusion. The strategy is composed of the new and old occluding's object position and the best target candidate. This local search approach results in similar computational complexity in regular operation and occlusion. The follow-up work of Hannuna et al. [25] further extends the DS-KCF framework by a Kalman filter motion model to improve the tracking during occlusions.

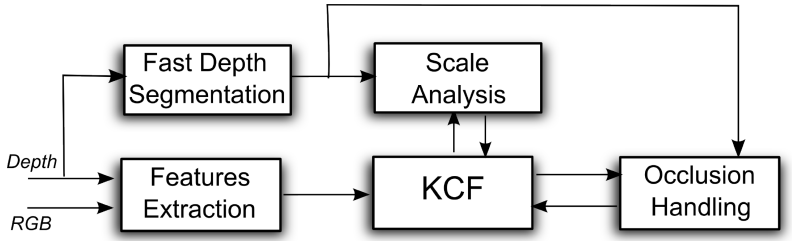


Figure 2.22.: Block diagram of DS-KCF tracker. It expands the baseline KCF tracker by depth features. Furthermore, a clustering-based segmentation method is applied. It is used for scale analysis and occlusion handling of the tracked object [9].

Gu et al. [23] published a different KCF-based algorithm in 2017. They treat the RGB picture and the grayscale depth image as separate inputs with different feature extraction methods. The color channel uses CN, HOG, and deep features, while the depth channel only uses HOG and deep features. A VGG-19 network pretrained on ImageNet serves as the extractor network. All features form a set of RGBD features that serves as input for a KCF. The response map then determines the center of the new bounding box. The major contribution of this paper is the attempt to tackle the KCF sensitivity to scale variation of the target object by exploiting depth information. Therefore, Gu et al. leverage the fact that the size of the object is proportional to the distance to the camera to estimate a scale factor for the predicted bounding box. The depth distribution in the previous frame is the reference for the next frame. Using the underlying correlation between distance to

the camera and size of the object in the image plane is efficient. Figure 2.23 outlines the strategy. While short-time occlusions can be handled quite well, the main problem of the presented method is its deficiency against long-time occlusion which leads to tracking failure. Also, Gu et al. observed that a single tracker which fuses RGB and depth information outperforms two separate trackers that treat both domains separately.

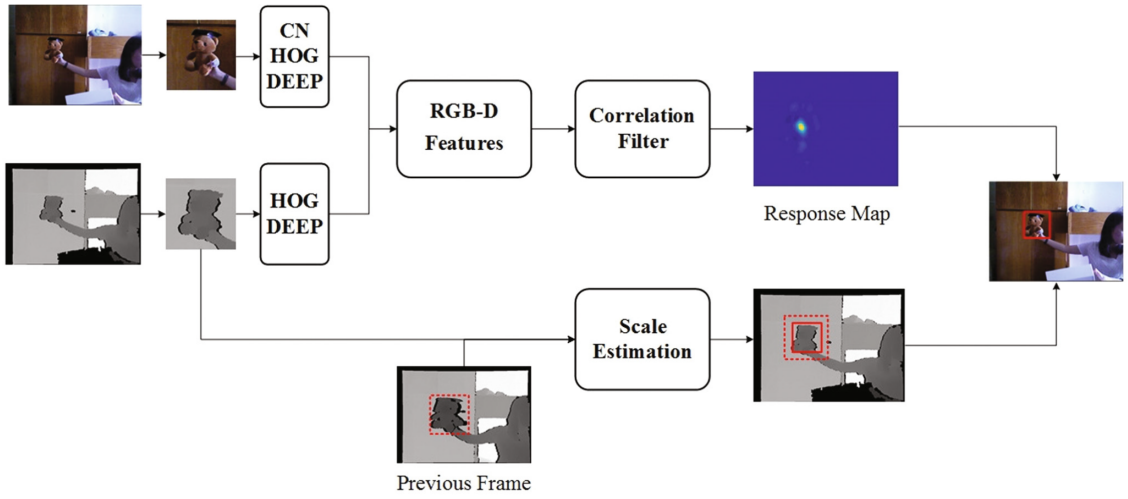


Figure 2.23.: RGBD tracker by Gu et al. RGB and depth information are two separate input channels with different feature representations. The RGB features are CN, HOG and deep features, while the depth channel uses HOG and deep features. These features are combined and a Kernelized Correlation Filter produces a response map with the maximum value corresponding to the new bounding box position. The depth input is compared to the previous frame to estimate the scale of the predicted bounding box [23].

As mentioned before, the former RGBD trackers have been evaluated on the older Princeton [79] dataset. All of them are based on hand-crafted features. That does not comply with the recent development of generic RGB object trackers, as the best-performing architectures are using features extracted from deep CNNs (see subsection 2.1.1, Table 2.1 Table 2.2, Figure 2.9 and Figure 2.17). So far there has not been much work on RGBD trackers due to the lack of appropriate datasets. Most recently, the VOT2019-RGBD [48] challenge initiiated researchers to come up with novel RGBD tracking architectures. It evaluates RGBD trackers on the Color and Depth Visual Object Tracking Dataset and Benchmark (CDTB) [61]. It has been the first time that a separate RBGD challenge was included in the series of VOT. It addresses long-term tracking with color images and additional depth information. Figure 2.24 presents the newly published challenge results. The top four trackers in the scope of the challenge leverage the power of deep features, reflecting and profiting from the development in conventional RGB tracking. Object Tracking by Reconstruction with View-Specific Discriminative Correlation Filters (OTR) and CSRDCF-D have been added as baseline trackers by the authors of the challenge for comparison with the pariticipating architectures. Therefore, they have not just been published and thus their papers are available. Both are based on DCFs and performed well on the Princeton dataset. Their considerably lower score compared to the others

yields the trend towards extensively using deep feature extraction CNNs. Figure 2.24 illustrates the gap between the two baselines (rank five and six) compared to the winning architectures (rank one to three). The performance of the top three trackers is very similar, while SiamM_Ds cannot keep up with them. However, it still performs significantly better than OTR and CSRDCF-D. The papers of SiamDW_D, Accurate Tracking by Category-Agnostic Instance Segmentation for RGBD Images (ATCAIS), LTDSEd and SiamM_Ds are not published yet. Hence, only a short description can be given based on the official VOT2019 challenge document [48] as well as the poster session of the CDTB [61] dataset. Noteworthy, it is not apparent how extensively the four best trackers exploit depth information. However, it shall be assumed that the information of [48, 61] are correct. The following paragraphs will discuss the different approaches.

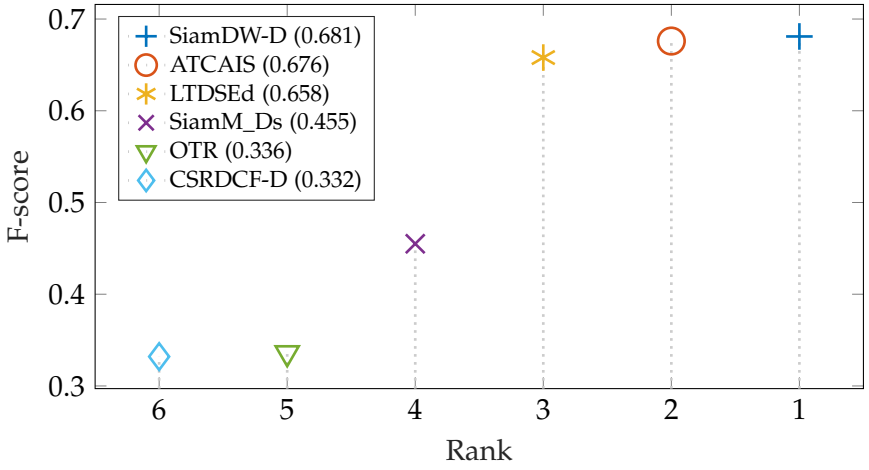


Figure 2.24.: Winners of the VOT2019-RGBD [48] challenge. Only rank one to four have been published in the scope of the challenge. OTR and CSR-DCF have been added as baseline architectures for comparison. The top three performers are very close to each other. All presented trackers (rank one to four) outperform the two baselines.

SiamDW-D. SiamDW-D (Yu et al.) implements a short-term tracker based on SiamRPN with a modified backbone as in [96] and a component with online updates similar to ATOM [18]. Furthermore, it consists of a global re-detection module, similar to state-of-the-art long-term RGB trackers (see subsection 2.1.1). Re-detection is performed using a refinement module based on IoU-Net [39]. According to the available information, the tracker seems to be very similar to its long-term RGB counterpart SiamDW-LT (see subsection 2.1.1). However, SiamDW-D is equipped with a multi-template-matching module, comparing unconfident tracking results with history templates to improve the prediction. Thus, the global re-detection strategy is only activated if the tracking results with the current and history templates do not exceed a certain confidence level. Depth information is specifically utilized to estimate target disappearance. At the time of writing, it is, however, not obvious how exactly the framework incorporates the additional information.

ATCAIS. ATCAIS (Wang et al.) combines the task of tracking with depth information

and instance segmentation. Its short-term tracker is based on Hybrid Task Cascade (HTC) [12] and ATOM [18]. HTC has been the winner of the 2018 evaluation of the MSCoco [58] dataset for image segmentation and thus is a state-of-the-art image segmentation algorithm. Being a cascade architecture, it consists of several stages with further stages refining the bounding box and mask predictions. As bounding box estimation and mask prediction are interlaced, the framework profits from incorporating complementary features of both tasks in each stage during training and inference. The predicted bounding boxes serve as an input for ATOM which refines them. The depth information is used for the detection of occlusion and target disappearance. Again, no details of the exact procedure are published so far. According to the paper presentation of the CDTB dataset, ATCAIS also uses the depth information for target re-detection.

LTDSEd. In addition to ATCAIS, Wang et al. published long-term tracking using depth information by diving videos into successive short episodes (LTDSEd) in the scope of the VOT2019-RGBD challenge. According to the poster presentation of the dataset, LTDSEd only uses depth for target loss detection. Refer to subsection 2.1.1 for the description of the long-term tracker.

SiamM_Ds. The SiamM_Ds (Chen et al.) tracker is a modification of SiamMask [85] such that it can process depth information. It enhances SiamMask by assigning the depth of the target to the average depth of the predicted segmentation mask. In that way, the target estimation can be constrained to only allowing small displacements in consecutive frames. Thus, depth is incorporates as a constraint to detect tracking failure.

OTR. The baseline tracker OTR [43] uses a 3D object representation and several view-specific DCFs to improve robustness. The DCFs are variants of CSR-DCF (see subsection 2.1.2) with the difference that the segmentation mask is generated from RGBD input and the estimated 3D model. This shall lead to a richer representation that e.g. allows better modelling of out-of-plane rotation. The 3D model is initialized at the start of tracking. During inference, rotation and translation of the model into the current object position are calculated by cloud-matching techniques, specifically an Iterative Closest Point method. This is used to for 3D model updating. The framework stores a new filter whenever the view has changed. View-changes are detected as a change of bounding-box width to height ratio of the current estimate and the 2D projection of the model. The filter memory with filters for different views is used for re-detection after occlusion or disappearance, thus making the whole architecture long-term tracking approved. Therefore, all DCFs are applied to re-detect the object. The filter with the highest response is then checked again with a scaled search region to cope with size changes. Apart from target loss or occlusion, all DCFs are applied once every few iterations to always choose the one that delivers the best response and therefore best represents the current view.

CSRDCF-D. Recently, Kart et al. [42] tried to leverage the progress made in short-term tracking to build a reasonable RGBD tracker. Therefore, their proposed generic framework deals with the problem of short-term trackers that they tend to include the background in their model updates in case of occlusion. This leads to drifting. The idea is similar to OAPF in the way that the occlusion state initiates a separate occlusion strategy. The implementation of this framework assumes the following: First, depth information does not improve visual object tracking as it does not represent

texture. Second, state of the art (SOTA) short-term trackers have decent performance in scenarios without occlusion. Combining these, they propose to use depth information for occlusion detection. The framework performs occlusion-dependant switching between the deployed short-term tracker and a specific occlusion recovery concept. Occlusion is detected based on a foreground segmentation mask. The mask is generated by a graph cut algorithm that minimizes a cost function. The cost function is adapted to the problem and incorporates depth and color information as well as a spatial prior, which favors pixels in the middle of the estimated object center. The ratio of foreground to background in the bounding box functions as an occlusion detector. If occlusion is present, a recovery mode ensures to increase the search region with an increasing number of frames that the object has been occluded. Step-wise enhancing the search region provides advantages to a simple full-frame grid search concerning computational efficiency. The motion history of the target defines the search region in recovery mode. Particularly, Kart et al. assume that the object will continue moving with the last-known velocity. Re-detection in recovery mode is performed by recognizing a response that is similar to previous tracked frames. This assumes that the target's appearance does not change significantly while being occluded. As the short-term tracker itself is not modified, progress on RGB tracking can conveniently be integrated to the RGBD domain. Figure 2.25 shows the general structure of the framework. It has been tested with several trackers. The best performance was obtained with CSR-DCF [62] as short-term tracker (see subsection 2.1.2) by replacing its mask with the foreground segmentation mask of this framework. This variant of the tracker is thus called CSRDCF-D.

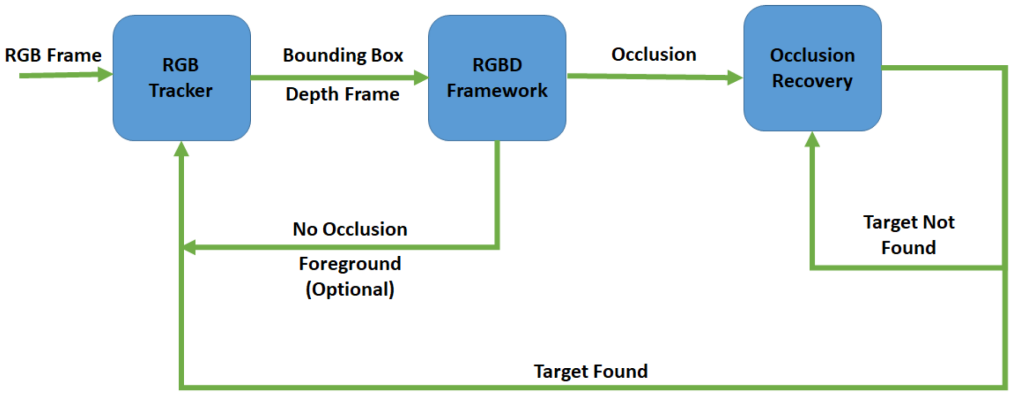


Figure 2.25.: The framework of Kart et al. convert short-term RGB trackers to be RGBD compliant. In the case of no occlusion, the RGB tracker estimates the target position. Whenever the target object is occluded, a special occlusion recovery mode activates until the target object is re-detected. The RGB tracker is not modified, thus not taking depth information into account [42].

Table 2.3 lists all participants of the VOT2019-RGBD challenge and splits them up into their components. Most of the architectures mainly use the depth information for target loss detection. Therefore, it is not surprising that the trackers have a similar architecture as conventional long-term trackers, consisting of a short-term tracker and a re-detection component. In addition, Table 2.3 shows how the depth information is

incorporated in the frameworks. The information is mostly taken from the CDTB [61] poster presentation.

Table 2.3.: Participating trackers in the VOT2019-RGBD challenge, split up into their components. The frameworks are divided into short-term tracker and global re-detection module. Furthermore, the use of the depth information is annotated. The trackers are sorted according to their rank in the challenge, starting with the best entry. [48, 61].

Tracker	Short-Term	Re-Detection	Depth
SiamDW-D	ATOM,	RPN, IoUNet	Failure Detection
ATCAIS	ATOM, HTC	RPN	Failure Detection, Re-Detection
LTDSEd	ATOM, SiamMask, RT-MDNet	RPN, RT-MDNet	Failure Detection
SiamM_Ds	SiamMask	SiamMask, enlarged search	Failure Detection
OTR [43]	DCF, Segmentation	DCF (image-wide)	3D Model
CSRDCF-D [42]	DCF, Segmentation	DCF (image-wide)	Segmentation

2.4. Discussion

There has been much activity in the field of visual object tracking in the last years. Bearing in mind the numerous challenges that go along with tracking, many sophisticated algorithms have been published and obtained impressive results on object tracking benchmarks. Template-matching methods (see subsection 2.1.1) dominate the leaderboard of today's trackers, delivering high speed and performance. Starting with SINT [82], learning a similarity function to differentiate between objects by training a deep convolutional network on large-scale datasets turned out to be a neat way of performing generic object tracking. Given the similarity function, template-matching methods often take the first frame in a video as a template and then compute a similarity score for possible positions of the object in new frames. The highest similarity score then leads to the new prediction, as the similarity function should learn to output high scores for the template and the search image containing the same object, and low scores otherwise. Despite being published in 2016, SiamFC [3] still achieves reasonable performance on current benchmarks and serves as a starting point for many other algorithms. Specifically, SiamRPN [54], DaSiamRPN [97] and SiamMask [85] are important milestones that build on SiamFC (see Figure 2.18). All of these approaches focus on finding better feature embeddings and similarity measures. THOR [75], on the other hand, focuses on finding better template representations over time and can operate on any template-matching method. Newer architectures like SiamMargin and SiamFCOT cope with both problems by proposing a template updating strategy to obtain a better similarity measure. Other algorithms implement a completely different approach named tracking-by-detection. Bolme et al. [7]

published the first tracker (MOSSE) in 2010, long before SINT [82] in 2016, which has been the first tracker that belongs to template-matching. Other than template-matching, these trackers learn to discriminate the object against background online. MOSSE [7] and KCF [32] have been important milestones. CSR-DCF introduces a weighting of different feature channels, CSTEM [95] further modifies it by improving tracking in occlusion scenarios. RPT [56] utilizes the concept of part-based tracking, leveraging several DCFs and combining their predictions. Generally, most trackers operate on hand-crafted features such as HOG or CN. Template-matching approaches often outperform tracking-by-detection approaches as hand-crafted features seem to obtain inferior performance compared to deep CNNs. Some approaches (Ma et al. [64]) compensate this by applying correlation filters on top of deep features. However, this often comes at the cost of speed reduction, especially when adapted online. Struck [26] exploits SVMs for tracking, Re^3 [22] tries to solve tracking by applying RNNs. Recently, some algorithms try to leverage the advantages of template-matching and tracking-by-detection. DiMP [4], CLGS (Zhao et al.), LT-DSE (Dai et al.) and SiamDW_LT (Du et al.) combine an offline trained siamese architecture with an online learning component. Thus, they try to profit from prior knowledge extracted from large-scale datasets and then improve the result by adapting a smaller component during inference. These architectures are somewhat complex and always face the tradeoff between speed and performance. In conclusion, in RGB tracking, there is a clear trend towards using deeper architectures to extract characteristic features of images, similar to other computer vision tasks. This applies to both approaches, template-matching as well as tracking-by-detection. This thesis will concentrate on template-matching methods as the best-performing trackers belong to this category, especially when it comes to real-time requirements (see Figure 2.9).

When working on tracking with depth information, research on how to leverage this additional information in other disciplines could be useful for the tracking domain. In object detection, Eitel et al. [20] propose to use two separate streams with the same architecture for feature extraction on depth and color information, and then combine them by a fully-connected fusion layer. The depth information is encoded as a 3D input by applying a jet colormap. Zia et al. [98] applies the concept of model ensembles by combining three different models. One model uses the depth information together with the color input to create a 3D voxel as input to a CNN that is learned from scratch. In semantic segmentation, Couprie et al. [15] considered depth as an additional input channel without any specific treatment. Hazirbas et al. [27] introduced FuseNet, which uses a pretrained network on color images for the depth channel by averaging the first layer weights along the channel dimension. Wang et al. [86] try to directly make use of the depth information in the model implementation by proposing two new layers: depth-aware convolution and depth-aware average pooling. Incorporating depth in the model layers reduces complexity and memory requirements compared to having two separate architectures. In pose estimation, Porzi et al. [71] change the convolution filter size based on depth, whereas Song et al. [80] inspect the potential of depth in pose estimation.

In tracking with depth information, only a few algorithms exist due to the lack of large-scale datasets and challenges. The focus of the tracking community is, therefore, on RGB tracking. OAPF [68] and 3D-T [5] use a particle filter approach that incorporates

depth information. The former uses the depth information to detect occlusions and based on that changes the motion model, while the latter formulates the motion and appearance model in 3D. OTR [43] is based on DCFs, but also saves a 3D model of the object. DS-KCF [9] extends the identically named RGB tracker by utilizing depth information for better scale and occlusion handling. CSDCF-D [42] presents a general framework that could extend any RGB tracker to include depth information to switch to an occlusion recovery mode and use the RGB tracker otherwise. All of these trackers operate on hand-crafted features. The recent results of the VOT2019-RGBD challenge present four new architectures: SiamDW-D (Yu et al.), ATCAIS (Wang et al.), LTDSEd (Wang et al.) and SiamM_Ds (Chen et al.). As the papers have not been published by the time of writing, no reference can be given. Other to the mentioned methods, these are all complex architectures that build on their RGB tracker counterparts and extend them by using the depth information for target loss detection. In that case, they switch from a local search to a global re-detection strategy. All of them, except for SiamM_D, are hybrid architectures, trying to combine tracking-by-detection and template-matching. None of the algorithms explicitly uses the depth for training, thus motivating the investigation in this thesis.

3. Approach

This chapter explains the approach in this thesis. At first, the challenges that go along with tracking with depth information are outlined. Secondly, the baseline RGB tracker deployed for all subsequent experiments is introduced in detail, such that the reader is confident about the architecture before moving to the next part.

3.1. Challenges

In general, the number of RGBD trackers is rare compared to the RGB case. Besides, none of the state-of-the-art RGBD trackers incorporates the additional depth information in the training process (see chapter 2). What is the reason for this situation, and what challenges have to be faced when doing so? After performing experiments on different strategies on how to improve tracking performance with RGBD, the three main problems are as follows:

- Lack of large-scale datasets
- Lack of benchmarks
- Uncertainty about preprocessing and architecture

Lack of large-scale datasets. The biggest challenge that complicates the training process of generic object tracking architectures on depth information relates to the problem that only a few datasets are available in this domain. Many are object-specific for performing tasks like pedestrian tracking, object pose estimation for robotics, and others [48]. Therefore, they are not well-suited for training generic object trackers. Additionally, the datasets are rather small compared to RGB datasets like ImageNet [74] or the recently released RGB tracking dataset LaSOT [21]. The most popular dataset for RGBD tracking before VOT2019-RGBD has been the Princeton Tracking Benchmark [79] published in 2013, indicating that not much effort has been put into RGBD tracking in the last years.

Lack of benchmarks. Another point which is in a way related to the lack of data issue is the limited number of benchmarks for RGBD tracking. By the knowledge of the author, only three benchmarks are available for comparing the performance of generic RGBD trackers, namely Princeton Tracking Benchmark [79], Spatio-Temporal Consistency (STC) dataset [90] and the recent VOT2019-RGBD challenge [48]. For comparison, there are several yearly challenges for RGB tracking (short-term, real-time, and long-term) only by the VOT initiative, neglecting other benchmarks such as GOT-10k [35] or OTB [88, 89]. Thus, the tracking community did not pay much attention to this research area. This now changed with the publication of the VOT2019 and VOT2020 challenges, which motivate modern long-term tracking approaches that utilize the additional depth information.

Uncertainty about preprocessing and architecture. In RGB tracking, the input is an image of shape $(w, h, 3)$, with w and h being the width and height of the image, respectively. The depth of the input is 3, representing the color channels (red, green, and blue) of each pixel. Note that in literature, depth in neural networks often corresponds to the number of layers that a specific network is composed of, whereas depth in the context of RGBD trackers describes the distance of pixels from the camera additional to the color information. Keeping that in mind, we have a 4D input, being red (R), green (G), blue (B), and depth (D). Different to RGB, where the pixel intensities are scaled in a fixed range $[0, 255]$, the range of the depth values is dependant on the sensor. Therefore, preprocessing here splits into two points: First, depth information should be preprocessed such that the network input is independent of the depth recording hardware, and therefore the network can be trained and evaluated on sequences recorded with different sensors, as necessary in VOT2019-RGBD. Second, one can think of different preprocessing techniques on an architecture level. So far, it is not obvious what architecture works well for depth information. RGBD can be used as a $(w, h, 4)$ input, which would increase the filter size of first the convolutional layer in the neural network. However, this means that the first layer can no longer use pretrained weights on large-scale RGB datasets (e.g. ImageNet) as the filter size increases. Generally, it is then no longer clear whether RGB pretraining helps, as the network learned how to extract important features from three channel color-only inputs. Using deep neural networks pretrained on a large-scale dataset like ImageNet (image classification dataset) and then performing finetuning on other computer vision tasks or different datasets is a prevalent practice in modern AI development, as training deep networks from scratch requires much data, computing resources, and time. Unfortunately, pretrained RGBD networks are not available yet. It is a similar situation to the tracking field, where the most effort is currently spent on image classification with color input. Hence, another approach would be to take a pretrained network for RGB and another network for depth, profiting from the availability of well-performing RGB architectures and still incorporating depth information by training a separate network and combining the features from both domains. Training the depth network from scratch on the one channel input could be one option, encoding the depth as color image of shape $(w, h, 3)$ would be another. The latter again offers different implementations, ranging from easy approaches like replicating the value to 3 channels to more sophisticated techniques as colormap encoding. Encoding to 3D might allow leveraging pretrained networks on RGB images in the depth domain, reducing the amount of data needed for training. Having access to pretrained networks would be a tremendous advantage compared to training from scratch. A third idea would be spatially encoding the image to a 4D volume of shape $(w, h, z, 3)$ with z corresponding to the distance from the camera, while the fourth channel is the color information. Though, this is problematic as it requires 3D convolutions, which are computationally expensive compared to 2D convolutions [87]. Again, pretraining can be an option here by replicating the filter weights of a 2D pretrained RGB along the depth dimension, as applied in [98].

Summarizing the challenges above, the lack of large-scale datasets is the main problem when training a tracker on RGBD information. The need for big datasets follows from the fact that deep convolutional networks, as present in all state-of-the-art approaches,

contain millions of parameters. Extensive training with lots of data is necessary to allow learning these. This challenge will hopefully become obsolete with the publication of large datasets in the future that specifically address RGBD tracking. The situation concerning the second challenge already changed, and the interest of the community is now also on RGBD tracking. This thesis has to arrange with the constraints given by these points and can only try to exploit the data with sophisticated data augmentation techniques. However, this only improves the situation to some extent. For instance, the number of object classes cannot be compensated by data augmentation, demanding the need for large-scale RGBD datasets in the future. The experiments in chapter 4 tackle the challenges of uncertainty about preprocessing as well as uncertainty about the architecture to exploit depth, establishing novel insights in RGBD tracking. Future trackers can further build on these findings to improve RGBD tracking.

3.2. DaSiamRPN as baseline RGB tracker

A well-performing RGB tracker will serve as the baseline for the subsequent investigations. Here, this architecture is DaSiamRPN [97], as it won the VOT2018 real-time challenge (short-term tracking) and ranked second in the VOT2018 long-term challenge [45]. Therefore, it meets the requirements of this work, as it is state-of-the-art, allows real-time performance, and implements a long-term strategy. Equally important, the code is publicly available and therefore allows modifications while guaranteeing a correct baseline implementation. This section gives a quick overview of the adapted DaSiamRPN architecture. Understanding the structure is crucial, as the inclusion of depth information accompanies adjustments to the baseline tracker. The interested reader should also have a look at the original publication [97] as well as [54], which covers everything in more detail.

3.2.1. Architecture

Figure 2.3 visualizes the architecture of DaSiamRPN. It is identical to SiamRPN, however with some differences in the training process. Recapturing the basics from section 2.1, template-matching based trackers compare a search frame to a template frame. For DaSiamRPN, the template z is an $(127 \times 127 \times 3)$ image centered at the object location in the first frame, while the search image x is of shape $(255 \times 255 \times 3)$ centered at the last predicted object position. For each frame in a video sequence, the architecture takes the template and the extracted search image as input and eventually predicts the most probable object location. DaSiamRPN consists of a deep siamese subnetwork φ for feature extraction and a RPN for bounding box proposal generation. The siamese subnetwork transforms the information from pixel level to the feature space. It will further be referred to as the backbone, as the RPN builds upon the generated features to make predictions. The backbone architecture could be any CNN that allows extracting characteristic features in images. However, the original implementation utilizes a backbone adapted from AlexNet [50] that has already been used in SiamFC [3]. For computational reasons, this work also utilizes an AlexNet-like architecture. However, deeper architectures might enable better object representations, but come at the cost of

more parameters and thus longer training times. Note that there is only one backbone with one set of parameters for encoding deep features from the two input images – thus being siamese. The intuition here is to find the most similar region to the template frame in the search frame in terms of feature similarity. Hence, the feature extracting network should be identical for both inputs.

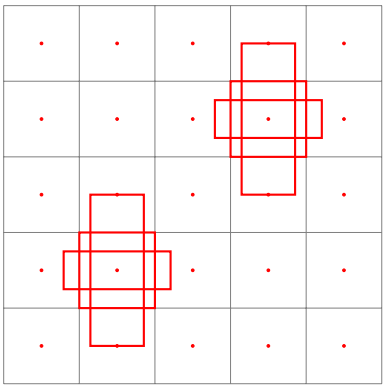


Figure 3.1.: Visualization of anchor boxes. The red circle in each grid cell denotes the center position of anchors located in that cell. Each grid cell contains k anchors of different shapes. The image shows the anchors of two grid cells for $k = 3$ with different aspect ratios. For visibility reasons, only the anchor positions are displayed for the remaining grid cells.

Given these feature representations, the RPN performs the bounding box proposal generation. Specifically, it utilizes the concept of anchors, dividing the search image into a regular $n \times n$ grid. Each grid cell contains k different anchors of various shapes and sizes. Figure 3.1 shows an exemplary 5×5 grid with three anchors at each grid position. The RPN consists of a classification branch and a bounding box regression branch and therefore splits the tracking problem into two distinct tasks on anchor level: Classifying each anchor whether it does contain the object on the one hand and predicting a bounding box correction for each anchor in terms of $(\mathrm{d}x, \mathrm{d}y, \mathrm{d}w, \mathrm{d}h)$ on the other hand. $\mathrm{d}x$ and $\mathrm{d}y$ is an offset in the anchor position, and $\mathrm{d}w$ and $\mathrm{d}h$ a change of the width and height of the bounding box of that anchor, respectively. The first CNN in the classification branch of the RPN (see Figure 2.3) consists of a 3×3 convolutional layer, which increases the channel dimension of the template feature representation to $(2k \times 256)$ to encode foreground and background scores on anchor level. Similarly, another layer in the bounding box regression branch increases the channel dimension to $(4k \times 256)$ to capture offsets in x, y, w, and h. The channel dimension of the feature representation of the search image stays constant for both branches. The classification branch then performs a depth-wise cross-correlation of the search image features with the feature representation of the template image being the correlation kernel. This leads to a $(17 \times 17 \times 2k)$ score map, giving a score for each anchor for containing the object or background. For the regression, the same procedure leads to a score map of shape $(17 \times 17 \times 4k)$, giving a correction to each anchor of each grid cell. The correlations for

classification and regression calculate to:

$$A^{cls}_{w\times h\times 2k} = [\varphi(x)]_{cls} \star [\varphi(z)]_{cls} \tag{3.1}$$
$$A^{reg}_{w\times h\times 4k} = [\varphi(x)]_{reg} \star [\varphi(z)]_{reg} \tag{3.2}$$

$\star$ denotes the convolution operation with $[\varphi(z)]_{cls}$ and $[\varphi(z)]_{reg}$ being the kernels, while A denotes the correlation output for each branch, respectively. As described earlier, the procedure only differs in the channel upsampling convolutional layer. Note that the convolutional layers that split the feature representations of the search image for classification and regression do not share parameters. Although the architecture of the RPN is similar for both branches, weight sharing only applies in the siamese backbone φ.

As mentioned in chapter 2, many long-term trackers follow a short-term tracking long-term detection approach. This technique also applies to DaSiamRPN, whose structure divides into these two components.

Short-Term Tracker

Given the discussed architecture, the short-term tracking approach is pretty straightforward. In the first frame of a video, the tracker precomputes the two kernels in Equation 3.1 and Equation 3.2 as they stay fixed for an entire sequence. Then, for each frame, the network extracts a fixed-size search region at the last predicted object position and computes the output scores by utilizing the precomputed template features. The search and template image sizes are defined by

$$(w + p) \cdot (h + p) = A^2 \tag{3.3}$$

where w, h denote the width and the height of the target's bounding box and $p = \frac{w+h}{2}$ controls the amount of context. Resizing leads to the final patch sizes of 127×127 for the template and 255×255 for the search image. Penalties modify the output scores such that scale and aspect ratio changes only prevail when leading to significantly better scores. For every proposal, the penalty calculates to

$$\text{penalty} = \exp\left(1 - k \cdot \max(\frac{r'}{r}, \frac{r}{r'}) \cdot \max(\frac{s'}{s}, \frac{s}{s'})\right) \tag{3.4}$$

with r being the proposal's ratio of height and width, and r' that of the last frame. s and s' denote the scale of the proposal in the current and the last frame and k is a hyperparameter that controls the amount of penalty for scale and ratio changes. Furthermore, another penalty strategy applies to large displacements from the last position by applying a hanning window to cope with outliers. Finally, the anchor with the highest probability of containing the object predicts the estimated regressed bounding box. The tracker outputs the position of this bounding box with the width and height being smoothened by applying a moving average filter:

$$w_{pred} = lr_{track} \cdot w + (1 - lr_{track}) \cdot w' \tag{3.5}$$
$$h_{pred} = lr_{track} \cdot h + (1 - lr_{track}) \cdot h' \tag{3.6}$$

where again w is the width of the predicted bounding box after penalizations and w' that of the last frame. h and h' denote the height of the predicted bounding box and the one in the last frame. lr_{track} calculates as the product of a learning rate hyperparameter and the highest classification score after applying all penalizations.

Unlike the original DaSiamRPN, the distractor-aware incremental learning during tracker execution (section 3.3 in [97]) is left out to simplify the overall architecture, as the goal is mainly to assess the benefit of depth information in the training process, rather than the absolute performance during evaluation. However, this might be inspected in subsequent works.

Long-Term strategy

In long-term tracking, the object can leave and re-enter the field of view. After such out-of-view situations, the tracker might not be able to redetect the object. These failures happen when the search region extracted at the last detected position does not contain the object when it re-enters the scene. The inclusion of distractor-aware training (see subsection 3.2.2) enables reliable target loss detection by evaluating the classification score. More precisely, a drop in the output score reflects the absence of the target in the video, which triggers a local-to-global search strategy. In such cases, the search image enlarges to the full image size intending to redetect the target. Once the classification score exceeds a threshold and thus indicates target presence, the search image strategy switches back to a local search region of smaller size centered at the object position. The threshold for detecting target loss and re-detection are hyperparameters of the tracker that need tuning for the specific application.

3.2.2. Training

The architecture is trained end-to-end by sampling from large-scale datasets. For tracking datasets, the template and the search patches are extracted from the same video with a maximum configurable sampling difference in frames. In this thesis, two frames in the training process are a maximum of 100 frames apart. Utilization of object detection datasets is also possible by taking the same object as template and search image; however, with enhanced shift and scale data augmentations to simulate a tracking scenario. The following illustrates the training procedure in more detail.

In every training iteration, the network gets a batch of input image sets containing a template and a search image each. The training procedure equals the approach described in [54]. Anchors only use one scale with anchor ratios of $[0.33, 0.5, 1, 2, 3]$, thus $k = 5$. The network is trained with positive and negative samples. Anchors with IoU $> \tau_{high}$ are labeled positive and anchors with IoU $< \tau_{low}$ negative. From these, num_{pos} positive and num_{neg} negative anchors are selected for an image set. Two separate loss functions define the classification and the bounding box regression error that a Stochastic Gradient Descent (SGD) optimizer will minimize. The loss functions come from Faster-RCNN [73]. Therefore, the loss function for the classification branch is the cross-entropy loss. For

each anchor at each grid position (compare Figure 3.1) it calculates as

$$L_{cls}(x_i) = \log\left(\frac{\exp(x_i)}{\sum_j \exp(x_j)}\right) \tag{3.7}$$

where x_i is the classification score for an anchor x belonging to class i, with the two possible classes being containing the object or not containing the object. j iterates over both classes. The total classification loss L_{cls} accumulates the respective loss values at all anchor positions that have either been labeled positive (IoU $> \tau_{high}$) or negative (IoU $< \tau_{low}$). Different from [73], the bounding box regression loss is a weighted L_1 loss that takes the normalized distance of each anchor into account. With T_x, T_y, T_w and T_h being the position and the shape of the ground truth bounding box and A_x, A_y, A_w and A_h the center position and shape of the anchor box, the normalized distances calculate as

$$\begin{aligned} \delta[0] &= \frac{T_x - A_x}{A_w} & \delta[1] &= \frac{T_y - A_y}{A_h} \\ \delta[2] &= \log\left(\frac{T_w}{A_w}\right) & \delta[3] &= \log\left(\frac{T_h}{A_h}\right) \end{aligned} \tag{3.8}$$

The weighted L_1 loss is defined as

$$L_1(x, y) = w \cdot |x - y| \tag{3.9}$$

with x being the output and y the target score. w is a weight factor. The total regression loss L_{reg} is the sum of the regression losses for all anchors that are labeled positive (IoU $> \tau_{high}$), stating that they contain the object. The regression loss for each anchor is

$$L_{reg}(x) = \sum_{i=0}^{3} L_1(x[i], \delta[i]) \tag{3.10}$$

where x is the anchor that the loss is calculated for, $\delta[i]$ the normalized difference as described above, and $x[i]$ the corresponding normalized distance for the regressed bounding box of that anchor. Thus more loss is accumulated with bigger normalized distances. The overall loss function is a weighted sum of the classification loss L_{cls} and the regression branch loss L_{reg}.

$$L = L_{cls} + \lambda \cdot L_{reg} \tag{3.11}$$

The weighting factor λ is another hyperparameter that influences the training performance. Table 3.1 summarizes the general training parameters discussed in this section.

A key concept in DaSiamRPN is the inclusion of negative pairs in the training process, which shall increase the discriminative power of the architecture. Precisely, some image sets utilized during training do not display the same target object. In these cases, the network learns to classify the object as not present in the search frame. This technique is referred to as negative sampling. The regressed bounding boxes do not contribute to the loss in the case of negative sampling, as none of the anchors is labeled as containing the target object.

Table 3.1.: Training settings for DaSiamRPN.

Hyperparameter	Value
k	5
anchor ratios	[0.33 0.5 1 2 3]
τ_{high}	0.6
τ_{low}	0.3
num_{pos}	16
num_{neg}	48
λ	1.2

3.3. System Overview

Figure 3.2 demonstrates the proposed framework to train a deep visual object tracker with depth information. Utilizing two encoder networks in the supervised training part allows to profit from pretrained RGB networks. This goes along with the need for a fusion layer, which allows to combine and learn cross-domain features. The drawback of this part is the need for big labeled RGBD datasets, such that the encoder networks as well as the fusion layer can learn generic object features which are applicable for various objects. The self-supervised pretraining part copes with this problem by enabling the use of unlabeled RGBD data. This strategy is often referred to as pretext training in the literature. Depending on the pretext task, the network is forced to learn features that help to solve it. These features should allow to distinguish between different objects. Therefore, the network parameters of the encoder network which has been trained in a self-supervised fashion can be used for supervised training in the second step.

3.3.1. Pretraining with different RGBD datasets

Some of the existing RGBD datasets only focus on specific object categories such as cars or pedestrians. However, as the domain of this thesis is generic object tracking, the tracker needs to be class-agnostic and should therefore leverage datasets that contain a variety of different objects to avoid overfitting to one category and maintain generalization capabilities to unseen objects. Table 3.2 lists publicly available labeled RGBD datasets that are generic in the sense that they don't focus on one object category. To the best knowledge of the author, no other labeled RGBD datasets of greater size are available. All RGBD datasets except for SceneNetRGBD are captured in reality and are therefore hand-labeled. Taking only these into account, the table shows that all of them are rather small compared to e.g. ImageNet with several million images. The VOT2019-RGBD dataset is the biggest dataset concerning the number of frames, while still being two magnitudes smaller than ImageNet and representing significantly less object categories. Only three tracking datasets are available, with the ground truth of the Princeton dataset not being available for training. Besides, some datasets do not contain bounding box annotations and can thus not directly be used to train a tracker. Also, others provide the captured depth data as point clouds, and in general the acquired depth data is

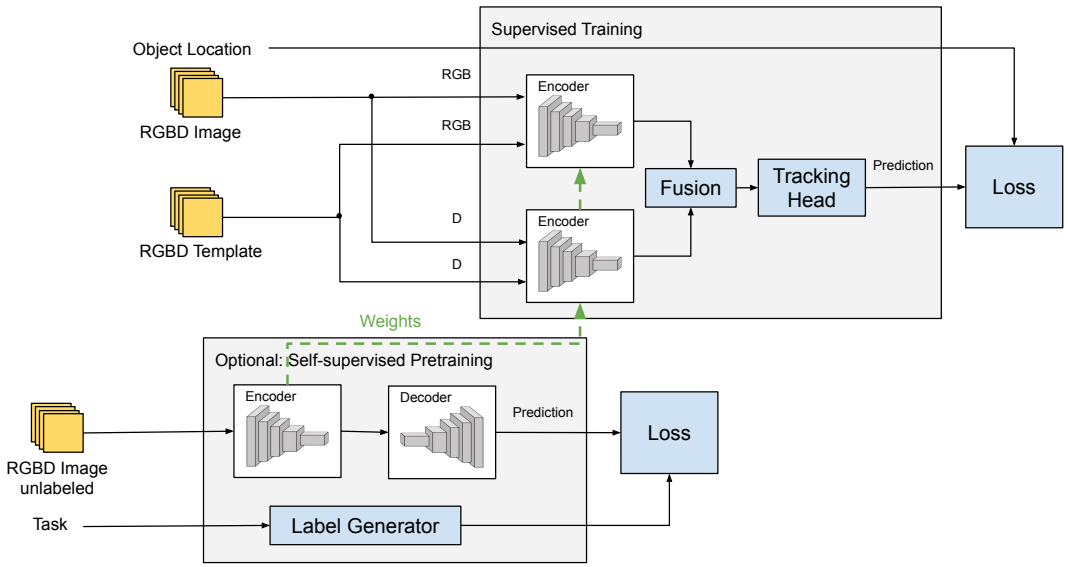


Figure 3.2.: System overview. The approach can be divided into two parts: Supervised training and self-supervised pretraining. The former is essential, while the latter is optional. Supervised training takes an RGBD search and template image as input. RGB and depth information are passed through separate encoder networks. For this work, the encoder network is a siamese network, transforming from pixel to feature space. The fusion component learns a cross-domain feature representation. It leads into the tracking head network which finally outputs a prediction of the object location. Given the ground truth position, this allows the learning algorithm to perform end-to-end optimization by minimizing the loss function. The self-supervised pretraining part takes unlabeled RGBD images as input. An encoder network transforms to feature space followed by a decoder network which is dependant on the pretext task. The label generator component is calculating labels, such that the encoder-decoder architecture can be trained end-to-end by minimizing a loss function. When utilizing self-supervised pretraining in the first step, the pretrained weights of the encoder network are used as initialization for the encoder network in the supervised training in the second step.

often different to the data present in the VOT2019-RGBD dataset which is the focus in this work, e.g. in terms of noise or fragments in the depth maps. Apart from that, videos in the VOT2019-RGBD dataset are captured with static cameras, while almost all RGBD video datasets have moving cameras, often in static scenes, which makes the data distribution quite different. Therefore, it is not clear whether the tracker would profit from utilizing these datasets. Apart from that, SceneNetRGBD is significantly bigger in terms of videos as well as images in the dataset. As McCormac et al. [66] have rendered the videos, the dataset provides pixel-perfect ground truth annotations which can be used for semantic segmentation, instance segmentation, object detection, and others. In a nutshell, constructing the videos follows several steps: First, indoor scenes are created by randomly placing objects in a room. Second, their framework calculates a random trajectory of the camera through the room. Third, the video is captured by calculating the view from the camera's perspective following the pre-defined path. The size of the dataset makes it appealing to use for training an RGBD tracker. However, some drawbacks might complicate the situation. The dataset does only contain indoor scenes, which might only capture a small portion of tracking scenarios. Also, depth sensors do not provide pixel-perfect annotations, which could harm the generalization capabilities of a network that trains on SceneNetRGBD. Apart from that, the variety of objects is limited to static objects such as chairs, tables, etc. but do not include dynamic objects like humans. Also, all objects in the scene are static while the camera is moving through the room, in contrast to the VOT2019-RGBD dataset with the camera being static and people or objects are moving while the background of the scene does not change much. Apart from that, the diversity of object categories is significantly smaller when compared to a "real" captured dataset. In that sense, the SceneNetRGBD has some potential for training RGBD architectures due to its size. However, compared to RGB tracking, it still suffers from a lack of diversity. Note that in contrast to RGBD tracking, large scale datasets of comparable or even greater size than ImageNet exist for the RGB tracking domain (e.g. YouTube-BoundingBoxes [72], Microsoft COCO [58], ImagenetDetection [74], ImageNetVideo [74], LaSOT [21]).

3.3.2. Fusion Layer

Making use of networks that have been trained on ImageNet for image classification is a common strategy, and the weights can easily be obtained by different sources. As pretrained networks are not available for depth inputs, one strategy is to transfer weights from the RGB to the depth domain. When considering the straight RGBD input of shape $(w, h, 4)$, the first layer filters of the encoder network have to be changed. Therefore, the RGB pretrained weights can not be used for the first encoder layer and could only be applied for the remaining ones. However, as the first layer filters directly influence the following layers, taking a random initialization for the first layer and pretrained weights for the following ones does not seem to be a good option. This also contradicts with the idea of transfer learning, which builds upon the fact that lower layer filters are general and thus task independent, while higher layer filters are task specific [92]. Therefore, taking a pretrained network on image classification and finetuning the last

[1] `http://www.image-net.org/`

Table 3.2.: List of labeled RGBD datasets. For each datasets, the table shows whether it contains videos and bounding box annotations. "#Vids" denotes the number of videos, domain the computer vision task that the dataset addresses, "#Images" the number of RGBD images. ImageNet is not an RGBD dataset and serves as a reference. It does not contain depth information, thus "#Images" refers to the number of RGB images in this case. All datasets except SceneNetRGBD have been captured with cameras and been labeled afterwards. In contrast, SceneNetRGBD has completely been rendered on computers with the goal of producing realistic-looking images, thus the labels go along "for free".

Name	Vids	BBoxes	#Vids	#Images	Domain
Active Vision [1]	✓	✓	20	25,896	Detection
Berkely 3D Object [38]	✗	✓	-	849	Detection
RGBD Scenes [52]	✓	✓	8	2,982	Detection
RGBD Scenes v2 [51]	✓	✗	14	11,427	Detection
NYU Depth v1 [76]	✗	✗	-	2,283	Segmentation
NYU Depth v2 [69]	✗	✗	-	1,449	Segmentation
Object Disappearance [65]	✓	✗	3	995	Segmentation
SUN RGBD [78]	✗	✓	-	10,335	Segmentation
Princeton [79]	✓	✓	100	20,332	Tracking
STC [90]	✓	✓	36	9,195	Tracking
VOT2019-RGBD [48]	✓	✓	80	101,956	Tracking
SceneNetRGBD [66]	✓	✗	17,865	5,359,500	Detection, Segmentation, and others
ImageNet [74]	✗	✗	-	14,197,122[1]	Classification

layers for object tracking is reasonable. However, changing the lower layer filters due to an additional input dimension would require to retrain the whole network. Furthermore, the pretrained filters did not learn how to use the depth information as fourth input, which somehow questions the sense of using RGB pretrained weights at all. Therefore, Figure 3.2 utilizes two separate encoder networks for RGB and depth information. The RGB network can profit from RGB pretraining. For the encoder network that extracts features from depth information two options will be investigated:

- Averaging the ImageNet-pretrained first layer filters and keep the depth input as 1D channel
- Encoding the depth information to 3D; therefore the original ImageNet-pretrained weights can be applied for all layers

For the second option, one strategy is to interpret the depth data as a grayscale image and copy the depth values along three dimensions. The second strategy that is analyzed is to use a Jet colormap encoding, as this led to better results compared to a simple replication in an object recognition setup [20]. The motivation behind encoding depth information and using RGB pretrained weights states that prevalent features in RGB images such as edges, corners, etc. are also visible when looking at a grayscale encoded depth image. Therefore, the assumption is that having learned from RGB images, this knowledge should also improve the situation for depth. The presence of one encoder network for each domain demands for a fusion strategy to combine the extracted features from the color and depth domain, such that during inference, the tracker can perform object localization based on RGBD feature embeddings.

3.3.3. Self-supervised Pretraining

Self-supervised pretraining tries to cope with the data shortage differently. In contrast to supervised learning, where large labeled datasets are necessary, self-supervised learning exploits labels that come with the data. The intuition behind this paradigm is that producing big labeled datasets is expensive, and in general labeled datasets are rare compared to unlabeled data. On the other hand, unlabeled data can easily be acquired and is often available at much bigger scales. By convention, the task that is used for pretraining is referred to as "pretext task". In tracking, self-supervised pretraining can be applied by taking the backbone architecture and adding a pretext task head network (decoder). The architecture then trains on unlabeled datasets. The decoder network is removed after self-supervised training and exchanged by the tracking head network (fusion if any and tracking head network). In Figure 3.2, this is equivalent to taking the pretrained encoder weights in the supervised training in the second step. This architecture has pretrained backbone weights from self-supervised pretraining. The whole architecture then needs to be finetuned end-to-end for tracking, as the tracking head and fusion components only have default intialization weights. Therefore, self-supervised pretraining copes with the dataset problem by training an encoder on RGBD images that do not have any labels. For the self-supervised pretraining experiments in chapter 4, training always consists of two steps:

1. Self-supervised pretraining
2. Finetuning for tracking (supervised training)

4. Results

The beginning of this chapter describes the evaluation dataset and metrics. Furthermore, it documents the results obtained by numerous experiments concerning the integration of depth information into the training process of a tracker, exemplary with DaSiamRPN as baseline RGB variant. The backbone architecture for feature extraction is a modified version of AlexNet [50], as proposed in [3]. It has a smaller total network stride to keep the spatial resolution and implements batch normalization [37] after every convolutional layer to speed up learning.

4.1. Evaluation

Experiments are evaluated on the VOT2019-RGBD challenge [48]. It is one of the challenges published in the scope of the seventh annual tracking benchmark by the VOT initiative. The dataset addresses long-term tracking with RGB and depth information. Contrary to short-term tracking, where the target always has to stay in the camera's field of view, long-term tracking is even more challenging as this requirement no longer holds true. Besides, the videos are often longer, and full occlusions might occur. During inference, a long-term tracker has to report a confidence score of target presence on top of the predicted target position for evaluation reasons. VOT2019-RGBD is the first dataset that incorporates depth information in the scope of the VOT benchmarks. Performance is measured by the long-term tracking measures introduced in [63]: tracking precision (Pr), tracking recall (Re) and tracking F-score. Specifically, tracking precision and recall are dependant on the tracker prediction certainty threshold τ_Θ. Tracking predictions with a certainty below this threshold are ignored, thus assuming the tracker did not predict the target in this case. The tracking F-score combines the two measures into a single scalar, thus allowing for easy comparison among trackers

$$F(\tau_\Theta) = \frac{2Pr(\tau_\Theta)Re(\tau_\Theta)}{Pr(\tau_\Theta) + Re(\tau_\Theta)} \tag{4.1}$$

The threshold τ_Θ is defined such that it maximizes the tracking F-score. Participating trackers in the VOT2019-RGBD challenge are ranked based on their tracking F-score, with the highest value indicating the winner. The dataset consists of 80 sequences acquired with three different sensors for depth recording: a Kinect v2 RGBD sensor, a pair of Time-of-Flight and RGB camera, and a setup with a stereo RGB pair [61]. Sequences are recorded indoor and outdoor, with objects containing several household and office objects, persons, etc. The videos have various resolutions and an average length of 1274 frames, which corresponds to approximately 42s. The VOT2020 challenge reuses the same dataset for using depth to improve long-term RGB tracking with target

disappearances. The obtained findings in this chapter can be used to participate in this challenge in the future.

4.2. Proof of concept

This experiment shall ensure that depth information is beneficial in the training process and thus serves as a proof of concept. It utilizes the VOT2019-RGBD dataset for training and evaluation, even though in the long run, it must only be used for validation purposes when participating at the challenge to meet their learning restrictions[1]. However, this ensures that the images in both splits belong to a similar data distribution. While this often is the case when training machine learning models on large image datasets, the situation is different when considering depth information with significantly smaller datasets and a variety of very different hardware for depth capturing. Training with depth images obtained by one sensor and deploying the model in a scenario with a different depth sensor might not lead to satisfying results. For a proof of concept, it should be a good practice to ensure the training and validation sets are similar, thus utilizing the VOT2019-RGBD dataset for both sets should be fine. Specifically, the dataset is randomly split into a training and a validation set, containing 64 and 16 videos. Generally, training the DaSiamRPN architecture on the VOT2019-RGBD training set and then evaluating the performance on the validation set utilizing the tracking F-score (see section 4.1) might be a reasonable approach. However, this is problematic as the results are highly dependant on the tracker hyperparameters. Furthermore, the measure is sensitive to drifting, as losing the target once might lead to a total loss of the target for the rest of the sequence. Suppose there are two trackers A and B. Tracker A uses depth information, while B does not. Both trackers have been trained on the same dataset, with B neglecting the depth information. At frame x, A predicts the wrong location of the target object, leading to a drift of the local search region, which ends up not containing the object. On the contrary, B predicts the location at frame x better. However, the mean accuracy of the predicted bounding boxes obtained by B is low, and thus the predictions have a low overlap with the target. For all subsequent frames, A would give perfect predictions with a high overlap, given a adequate search region. As it is, however, stuck in a local search region which does not contain the object, A might end up with a lower tracking F-score than B due to a single misprediction at frame x; although leading to better predictions in the rest of the frames, given a search region that does contain the object.

For the following experiments, the goal is to investigate the benefit of depth information in the training process of deep visual trackers utilizing DaSiamRPN as baseline architecture. The analysis should be robust to the mentioned misprediction issue, as this would fit the insights to individual frames in the validation dataset, which would then decide over the performance of an experiment. Thus, evaluating the tracking F-measure somewhat overfits to the performance on a few frames, possibly at the cost of lower performance on the lion's share of the overall frames. This is especially important in the VOT2019-RGBD dataset, as the number of sequences is limited, some are rather

[1] `https://www.votchallenge.net/vot2020/participation.html`

long, and generally drifting might happen due to the complexity of the sequences [61]. A simple yet effective strategy alleviates the mentioned problems and copes with the misprediction issue. The evaluation technique is identical to the training process in terms of training sample generation. For a given video, two frames centered at the object position, which are a maximum of 100 frames apart, are given to the network after applying shift augmentations. Shifting the images during training is necessary to ensure that the network is not biased towards predicting the center of the image. During validation, shifting allows to test whether that has actually been learned. Further augmentation techniques during training might improve generalization. However, as a proof of concept, other augmentations are not leveraged to allow faster training. Note that for validation, no negative sampling is applied as the quality of the predicted bounding box shall be evaluated. The network outputs the classification scores and regressed bounding boxes for all anchors. A simple validation tracker takes the anchor with the maximum probability of containing the object as prediction and outputs the corresponding bounding box. Compared to DaSiamRPN, it does not implement any penalties (scale change, aspect ratio change, displacement), as well as no moving average to update the predicted bounding box. Hence, the output bounding box corresponds to the unprocessed network output. The approach calculates the IoU of the predicted bounding box with the ground truth and is, therefore, a more practical measure on how well the present network performs compared to other approaches. Also, it allows faster evaluation compared to tracking F-measure, which is not feasible for fast experiments. The general setup is motivated by the common practice in image classification, where after each epoch, the validation accuracy is calculated. From a conceptional point of view, the mean IoU somewhat corresponds to the validation accuracy in this case, which should allow analyzing performance in the tracking scenario. Each of the 16 validation videos is used 300 times to extract image pairs that are fed into the network. Seeding random number generators guarantees that the same images are presented to the network after each epoch as well as during different training runs, thus providing a fair base for comparison. The central hypothesis of this work states that depth information should be beneficial in the training process, as it is an additional channel of information, which should at least allow the same performance. This happens if the network learns that it performs best when not using the additional information. However, it might also enable better performance when leveraging depth in situations where it would have problems to track an object with color information.

In the following, two architectures are trained on the generated VOT2019-RGBD training split with SGD for 50 epochs. One neglects the depth information, while the other one includes it as an additional channel. Training for each experiment is performed seven times with different Learning Rates (LRs) and a batch size of 16 per Graphics Processing Unit (GPU). Weight Decay (WD) is set to 0.0. Performing training on different sets of hyperparameters minimizes the risk that these are fit to one specific architecture and thus would not allow generalization on what is useful for depth integration. The mean IoU of the validation tracker on the validation set is used for evaluation. Higher LRs than 0.1 turned out to not allow learning at all, while lower learning rates than 0.0001 led to no convergence within the 50 epochs. A LR warmup increases the LR from $\frac{1}{5}$LR to the target value within the first 5 epochs. After that, it is constant for the

remaining 45 epochs. The whole network consisting of the feature extractor and the RPN is trained from scratch utilizing Kaiming [30] initialization for the convolutional layers. Negative sampling is set to 0.05, thus utilizing this technique on 5% of the training pairs. Shift augmentations are performed on the image pairs presented to the network. Table 4.1 summarizes the training hyperparameters. Training is performed on a DGX-1 station with eight Nvidia Tesla P100 GPUs.

Table 4.1.: Training settings for proof of concept.

Hyperparameter		Value
Optimizer		SGD
Epochs		50
Batch size		16 per GPU
LR		[0.1 0.05 0.01 0.005 0.001 0.0005 0.0001]
WD		0.0
LR warmup	Epochs	5
	Start LR	1/5 LR
	End LR	LR
LR schedule	Type	Constant
Negative Sampling		0.05
Training Pairs per epoch		19200
Validation Pairs per epoch		4800
Augmentations		shift

4.2.1. Training baseline DaSiamRPN on RGB

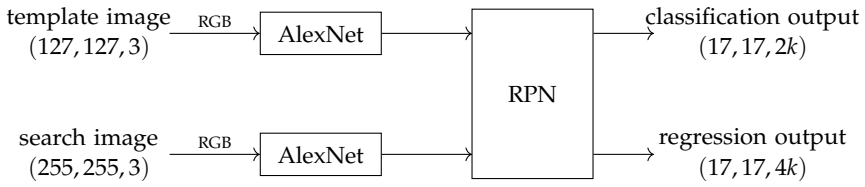


Figure 4.1.: Setup for RGB training of the DaSiamRPN architecture. The AlexNet backbone extracts features from template image and search image, respectively. Based on these, the RPN predicts foreground-background classification and bounding box regression outputs on anchor level. k reflects the number of anchors.

The first experiment trains the DaSiamRPN architecture without modifications on the RGB data of the training set, neglecting the additional depth channel. The number of anchors k, as well as other parameters, are kept from the original implementation. For details on the architecture, refer to section 3.2 or to the original paper [97]. The result serves as a baseline for comparison with the following architectures. Figure 4.1 outlines the setup.

4.2.2. Training on RGBD

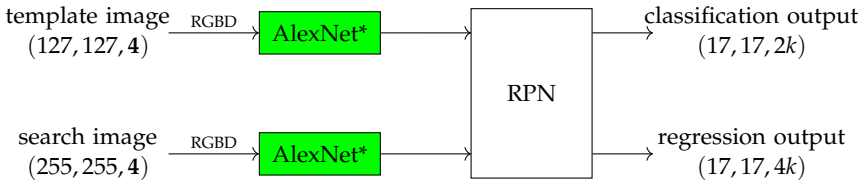


Figure 4.2.: Setup for RGBD training. Depth information is preprocessed and stacked on top of the color information. Input vectors therefore consist of 4 channels. $*$ denotes the modifications to the first layer filters of the backbone architecture to accept the additional channel channel.

Training is performed on all available data, including the depth channel. As mentioned in section 4.1, the VOT2019-RGBD challenge applies depth recording with three different sensors. Section 3.1 mentioned the problem of different value ranges for the depth channel dependant on the acquiring hardware. Importantly, when testing the tracker on the dataset, it does not know which sensor captured the depth data, demanding for a suitable preprocessing method. While the minimum value of all sensors employed in the dataset is zero, the maximum differs among chosen hardware. Figure 4.3 visualizes the distribution of the maximum depth value in all sequences in the first frame. Note that in tracking, we only now the object position in the first frame, therefore the information from the first frame is all there is. One sensor in the VOT dataset can be identified by looking at the maximum value. These are probably the outdoor sequences, as objects might be far away and the depth data can be noisy when e.g. capturing depth for the sky. The other two sensors output similar maximum values, which does not allow for a simple separation based on the maximum depth value in the first frame. The same applies when evaluating the mean depth values in the first frame. Besides, detecting the sensor and performing different preprocessing is not desirable, as this would fit too much to the dataset and the acquiring hardware. Thus, it would not solve the problem for the general case. Therefore, all conducted experiments in this thesis that train with depth information perform a simple yet effective technique that allows preprocessing during training and evaluation of the tracker. At the start of tracking, only the first frame is available. Therefore, rescaling fits the depth channel in the range of $[0, 255]$. Advantageously, all inputs to the network (pixel intensities and depth information) then have the same range, thus choosing 255 as the upper limit for the depth channel seems reasonable. In all subsequent frames, the scaling factor of the first frame has to be applied. Otherwise, this could lead to a change of the object's depth, even in a fixed camera setup with a static object. During training, the same technique can be applied. Every image pair that the architecture trains on will be rescaled with the information of the template image. The same applies to the IoU testing of the validation tracker. Figure 4.2 shows the setup. Note that the input vectors have four channels, three for color representation, and one for depth. The first layer of the backbone is modified by increasing the channel dimensions to accept 4-channel inputs.

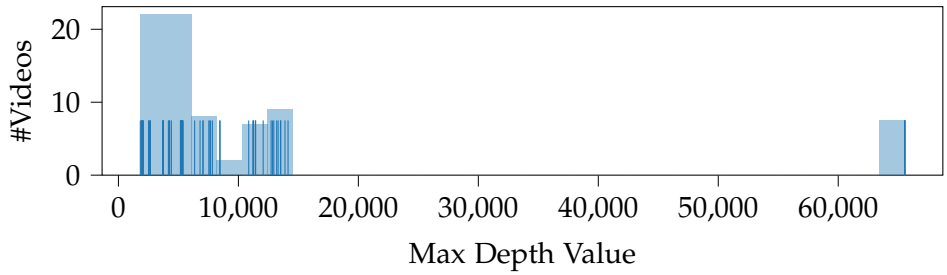


Figure 4.3.: Histogram and distribution of the maximum depth value in all sequences of the VOT2019-RGBD dataset in the first frame. The dataset utilizes three different sensors for depth recording. One sensor can potentially be identified, as some sequences have a maximum value of 65535. The first two sensors have similar sensor value ranges, which makes separation based on the maximum depth value in the first frame inappropriate. The depth values are encoded as 16 bit integers, therefore values range from 0 to 65535.

4.2.3. Results

Figure 4.4 presents the results of training both architectures; baseline DaSiamRPN as well as the modified version that considers depth in the training process. During training, the mean IoU of the final predicted bounding box with the ground truth on train and validation split are logged after each epoch. For every setup, the figure shows the snapshot yielding the highest score on the validation set, as well as the associated training IoU. The validation scores show that the overlap is higher for the network that considers depth in the training process. This applies independently of different LRs. The best validation IoU for the RGB-only architecture is 66.8% with a LR of 0.01. The best score for RGBD is 69.8% with a LR of 0.005. Thus, the mean overlap on the validation set is about 3% IoU better when using the additional depth information, which is an absolute improvement of 4.6%. Obtaining better validation results for RGBD strengthens the hypothesis that depth information might help to generalize to unseen data in tracking when considered in the training process. At this point, it can only be stated that the depth preprocessing works well for the training case and might also work well in a real tracking scenario. However, this still has to be confirmed. The IoU on the training split is similar for the RGB and RGBD case. LRs bigger than 0.01 are too high and thus the model does not achieve good overlaps on the training split. These cases will not be covered in the following investigations. Very small learning rates, on the other hand, also lead to a lower IoU, indicating that the training either did not fully converge or has been stuck in a local minimum. For the training set, the IoUs are very similar when considering a proper lr of e.g. 0.005. Still, there is a large gap between the training and validation split. This can basically have two reasons: Data mismatch or overfitting. Data mismatch happens when the splits come from different data distributions. In this case, there will always be a large gap between the performances, as the training data is different from the validation data. Note that this does not mean that training on data from a different distribution does not help. It can still improve the overall model performance, however only up to a gap between training and validation which depends

on the similarity of the distributions. Applying data augmentation techniques to increase the overall amount of data might improve the problem. However, this is unlikely in the present case, as it is not possible to increase the number of object categories by augmentation, which probably is the main problem. Another solution would be to either acquire more or different data. However, this is expensive due to additional effort for capturing and labeling videos with depth information, and is not in the scope of this thesis. Overfitting, on the other hand, indicates that the model captured the noise along with the underlying pattern of the dataset and therefore the model has high variance. Synthetically increasing the dataset size by data augmentation or applying other regularization strategies is a common practice to cope with overfitting.

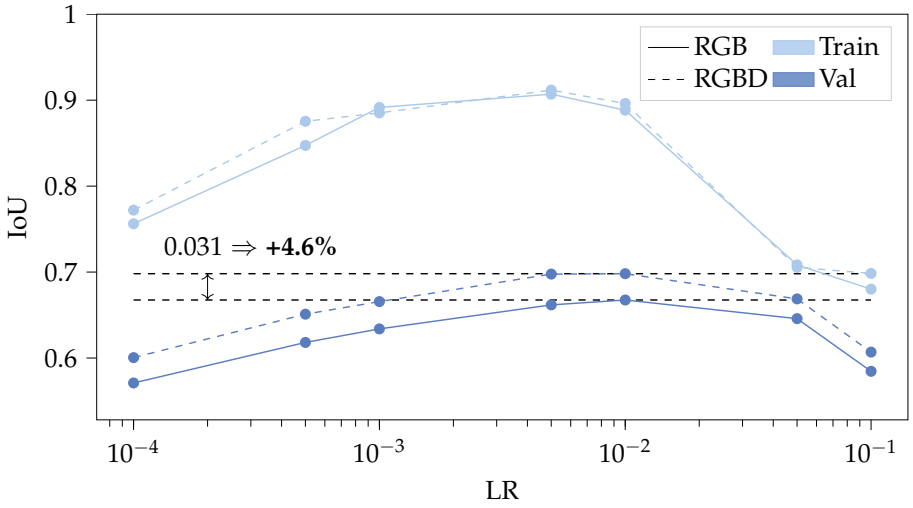
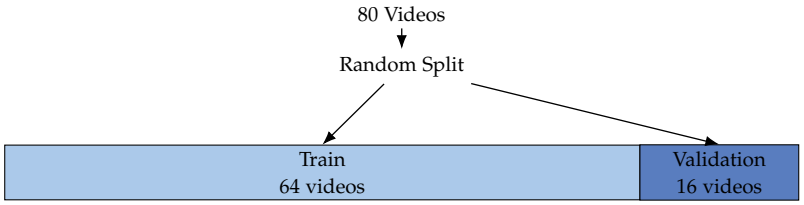


Figure 4.4.: Best IoU on the validation (Val) set and corresponding score on the training set obtained for different runs. RGB stands for the original DaSiamRPN architecture which takes color images as input. The RGBD architecture preprocesses the depth information and then stacks it to the color information as a 4-channel input. The graphic only shows the best model during a training run for each setup.

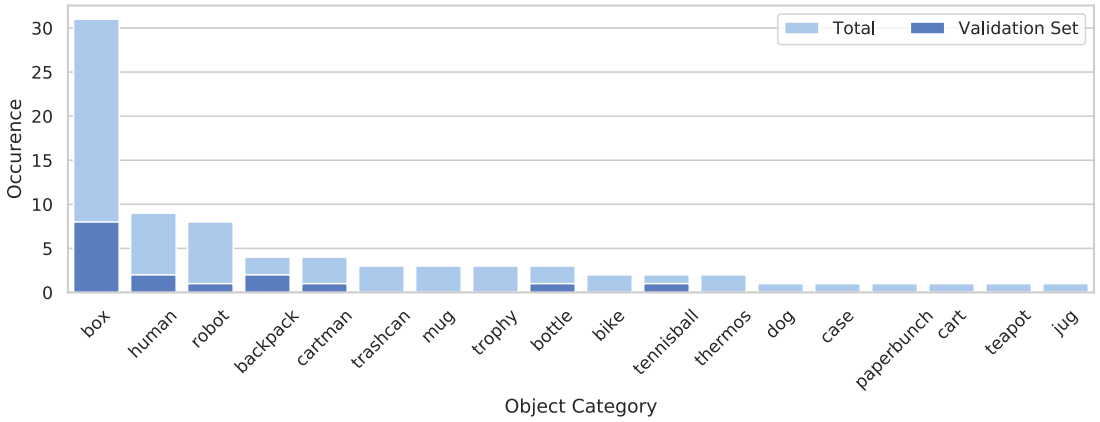
4.2.4. Data Distribution

Figure 4.4 showed a big gap between the validation and training IoU of more than 20%. Overfitting or data mismatch between the training and validation split have been identified as possible reasons. This subsection will give a detailed investigation of the observed issue. When utilizing the VOT2019-RGBD dataset for training and validation, it turns out that the training process and thus the IoU highly depends on the split of the data. Figure 4.5 shows the distribution of the VOT2019-RGBD dataset in terms of object categories of target objects. When utilizing it for training, the tracker will be biased towards correctly predicting objects that dominate the training set. By looking at the total object category distribution, this might even be concerned as the dataset overfitting to specific object categories, which is a severe issue for training. However, as the VOT

datasets should generally be used for evaluation when participating in the challenge, it is not so much of a concern. However, this also applies to other datasets with depth information, as these are rather small and therefore have similar properties. Nevertheless, even when not utilized for training, the dataset as it is theoretically promotes trackers that overfit to object categories like box, human, robot etc and might rank the same tracker that has been trained on a large variety of objects worse. It should be said that the represented boxes in the dataset are of different shapes and sizes, which somewhat reduces this fact, but it is still an issue. As usual in deep learning, the training, validation, and test data should belong to the same distribution as the data in the environment that the tracker gets deployed in. This guarantees a well-defined training setup and allows good generalization to unseen data.



(a) Dataset split by videos.



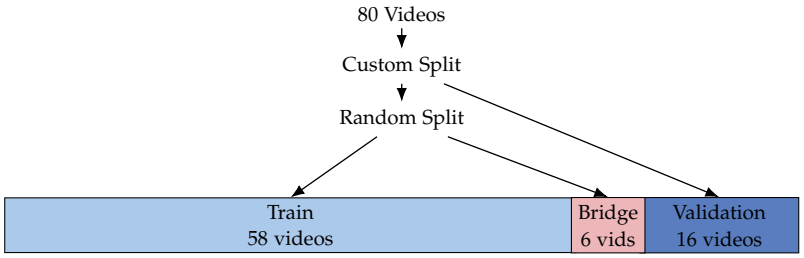
(b) Dataset split by object category.

Figure 4.5.: Data distribution of a random split of the VOT2019-RGBD dataset into training and validation set. (a) shows the split of the 80 videos. (b) analyzes the validation set in terms of target object categories in the videos. The dataset is biased towards categories like box or human, as they are represented in multiple videos, compared to e.g. cart, teapot or jug.

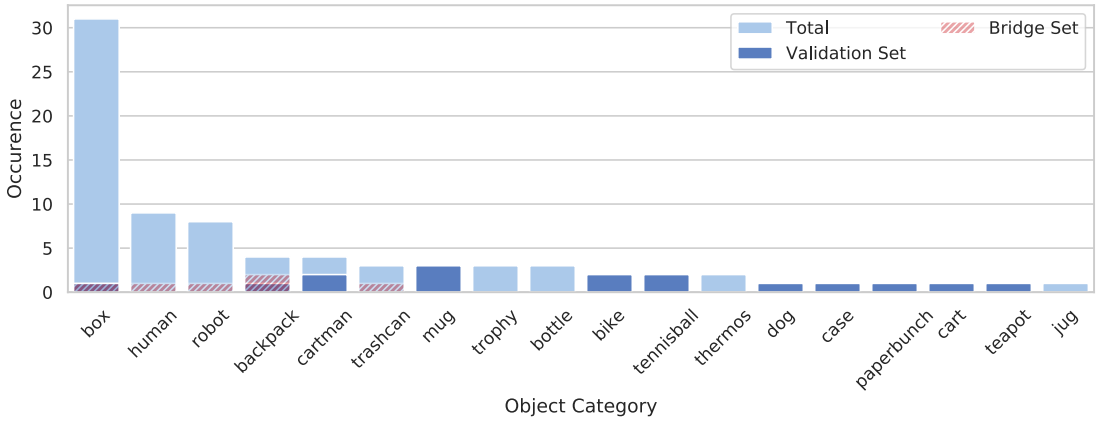
The first plot in Figure 4.5 visualizes the random split of all 80 videos in the dataset into a training set of 64 and a validation set of 16 videos. The second plot furthermore shows the object categories of the validation videos compared to the overall distribution of object categories in the dataset. This split has been utilized for generating the results in subsection 4.2.3. It becomes obvious that every category in the validation set is also present in the training set at least once. Hence, the validation data is somewhat similar to the training data.

To further investigate whether the performance gap between training and validation is due to the data distribution, the same experiment is conducted again with a custom split of the data. A third split referred to as bridge set splits the training set by randomly taking 10% of the videos. The first plot in Figure 4.6 shows the overall partitioning of the data. The bridge set is not used for training but validation in terms of IoU as well. Comparison of the IoUs on train, bridge, and validation set generally allows identification of data mismatch as well as overfitting. Specifically, if the performance on bridge and validation is comparable, and the performance on the training set is significantly better, the difference is mainly due to overfitting. On the other hand, if the performance on train and bridge set is similar and rather high, while it is considerably lower for the validation set, this indicates that the gap mostly results from a data mismatch problem. In this case, the model performs well on unseen data if it comes from the same distribution (like the bridge set). However, it performs worse on the unseen validation set, and thus, this indicates data mismatch. The second plot in Figure 4.6 shows the distribution of object categories in the custom validation set, the bridge set, and the training set. Compared to (b) in Figure 4.5, the represented object categories in the validation set have a much smaller overlap with the object categories present in the training set. The randomly sampled bridge set, however, only contains object categories that are also present in the training set.

Figure 4.7 shows the result of training the DaSiamRPN architecture on RGB and RGBD data for the custom split of the dataset. The setup is identical to the prior experiment with a random split of the data. See Table 3.1 for the training settings. However, due to the additional split of the training set into bridge and training, this time 17400 training pairs are utilized per epoch, while 1800 pairs of the bridge set allow calculating an IoU. Again, 4800 pairs are used for the estimation of a corresponding mean IoU on the validation split. The numbers result from utilizing each video 300 times per epoch for all splits. The graph shows a similar trend for both architectures, RGB and RGBD. Again, $LRs > 0.01$ lead to worse performance on the train split, indicating that the LR is too high. The architecture that considers depth information leads to the best validation IoU of 60.1% with a LR of 0.01. The best score for overlap for the RGB architecture is 57.0%, thus the mean overlap is again roughly 3% IoU higher when considering depth information, giving an absolute improvement of 5.5%. This is consistent with the prior observations with a random split of the data. Given the split of the dataset, the IoUs are also better on the training and bridge set. The best training IoU is 89.5% with RGB and 92.3% with RGBD, the best bridge IoU 88.2% and 90.5%, respectively. There is a large gap between the IoU of the final predicted bounding box on the training / bridge and the validation split. As mentioned above, this indicates data mismatch. The data distribution of the validation set is different to the bridge set, as the model did not see any data of these splits during training but it performs significantly better on the bridge set. Compared to the first experiment with a random split of the data (see Figure 4.4), the highest validation IoU of the model with depth is now 60.1%, compared to 69.8% before, while the score on the training set is almost the same (91.2% and 92.3%). The gap between training and validation set increased by 8.6% IoU, indicating that the data mismatch problem has become even more prevalent with the custom dataset split. To conclude, the data mismatch problem has been present when testing on the random



(a) Dataset split by videos.



(b) Dataset split by object category.

Figure 4.6.: Data distribution of a custom split of the VOT2019-RGBD dataset into training, bridge, and validation set. (a) shows the split of the 80 videos. The training set obtained by the custom split divides further into the final training and bridge sets. (b) analyzes the validation and bridge set from (a) in terms of target object categories in the videos. The custom validation set has a small overlap with objects in the training set. This is not true for the bridge set, which again contains objects in the training set due to random sampling.

split (see Figure 4.5). Doing a custom split further increases the presence of the problem, leading to a lower maximum IoU on the validation set. Recapturing the data distribution of the splits (see Figure 4.6), all categories in the bridge set also exist in the training set, which is not true for the validation set. The model learns to locate these categories while it has poor prediction capabilities for unseen object categories – as present in the validation set. Note that in generic object tracking, the amount of object classes is unknown beforehand. Therefore, it is a crucial requirement of a tracker to adapt to objects that have not been seen during training. More generally, this experiment indicates that training a tracker from scratch might not be reasonable with small datasets like the VOT2019-RGBD due to a prevalent data mismatch problem. The root cause of this problem is a lack of sufficient object categories, which makes the training highly dependant on the split of the dataset into training and validation. This applies to both, RGB and RGBD trackers. However, in practice mostly RGBD trackers are affected, as large-scale datasets are widely available for the color domain, while not a single dataset of comparable size is available for depth and color information. Therefore, this problem can be alleviated for RGB tracking by choosing big datasets, which are not yet available for RGBD.

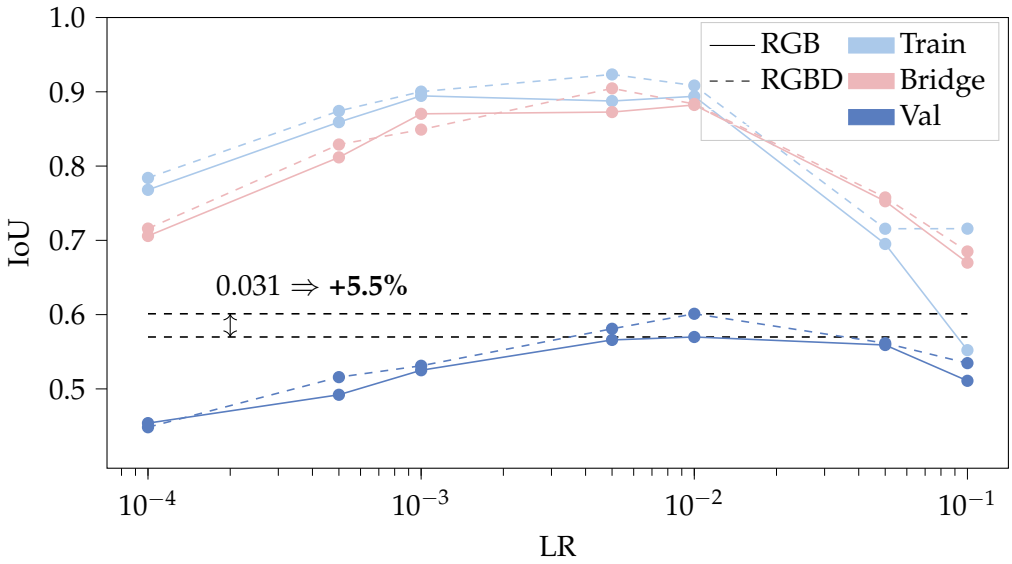


Figure 4.7.: Best IoU on the validation (Val) set and corresponding score on the training and bridge set obtained for different runs given a custom split of the VOT2019-RGBD dataset into training, bridge, and validation set. RGB stands for the original DaSiamRPN architecture which takes color images as input. The RGBD architecture preprocesses the depth information and then stacks it to the color information as a 4-channel input. The graphic only shows the best model for each setup.

This section presented an effective preprocessing strategy for depth information, such that data from different depth capturing devices can effectively be used for training visual object tracking architectures. The observations by the conducted experiments show that training with depth information can enable trackers to achieve better performance.

Besides, the second experiment with custom split of the data enabled insights into the problems that occur when training an architecture from scratch on small datasets with only a few object categories. This applies to VOT2019-RGBD, which is rather small, with only 80 videos and a total of 18 object categories. The evaluation metric that has been introduced in this section is the mean IoU of the predicted bounding box with the highest probability of containing the object, averaged over all validation pairs. It turns out that this measure allows fast comparison of different experiments. A second experiment with a custom dataset split shows that in the case of small datasets, the performance in terms of mean IoU is highly dependant on the partitioning of the data into training and validation set. Specifically, when training from scratch, the main concern is a data mismatch problem, which does not allow the model to generalize well to unseen object categories. The next section investigates how to leverage pretraining to cope with the problem of data mismatch and small datasets, profiting from having seen a diversity of object categories in advance.

4.3. Pretraining on RGB(D) Input

This section investigates pretraining methods for architectures with and without depth information. Pretraining is a common practice in deep learning that often significantly improves the performance, especially in domains where data is either rare or hard to acquire. Training a deep architecture from scratch on a large dataset like ImageNet can take up to several weeks on multi GPU setups and might require careful hyperparameter tuning. Generally, transfer learning from one task A to another task B makes sense whenever a lot more data exists for task A compared to task B and low-level features from one task could be helpful for learning another task. This most of the time applies to computer vision, with pretraining on large image classification datasets. Also, detecting low-level features such as edges in an image might be useful for many different target tasks. Thus, using pretrained networks allows profiting from tedious work of researchers in the computer vision field, accelerating progess on related computer vision tasks, such as e.g. object tracking. Most people use pretrained networks that have been trained on ImageNet [74]. This has several reasons. First, the networks are publicly available and often integrated in deep learning frameworks. Secondly, the networks are assumed to have learned to extract general features that are meaningful to most types of images, as the ImageNet dataset contains over 14 million labeled images from 200 object categories. However, ImageNet only contains RGB data, and it is therefore not clear what to use with RGBD inputs. The most promising approach might be to have two backbones, where one extracts features from the RGB, and the other one from the depth domain. With this approach, the RGB extractor profits from the progress on RGB image processing as well as pretrained RGB networks, while the second network can concentrate on depth. This also has the advantage that both can be trained separately, such that the RGB branch profits from the availability of large-scale datasets. This section concentrates on feature extraction from RGB information at first, while then trying to also profit from transfer learning with the two backbone approach. Nevertheless, it is a challenging task to come up with a suitable pretraining strategy for tracking architectures that use depth

information, as no pretrained networks for depth information are available. Furthermore, it is not clear to what extent an architecture with depth information can profit from RGB pretraining. For the following experiments, one half of the VOT2019-RGBD dataset is utilized for training, the other one for validation. Therefore, the tracker should profit from being finetuned on the target dataset, while not overfitting to only the few present object categories in the training set by validating on a rather big validation set (50 percent compared to usual validation splits of 10% to 20%). This section is split into two parts. The first part investigates pretraining effects when utilizing ImageNet pretrained weights for color-only inputs. The second part then tries to leverage RGB pretraining for an architecture with depth information, where pretrained models are not commonly available.

4.3.1. RGB Input

This subsection only uses the color images as input. Therefore the feature extracting network can use pretrained weights on any RGB dataset as initialization. Note that the feature extracting network is also referred to as backbone. The experiment does not use any depth information and serves as a baseline for the upcoming observations. The network is trained with SGD for 50 epochs with a batch size of 16 on 8 GPUs. A LR warmup is performed during the first 5 epochs, increasing the LR from 1/5 to the base LR. For the remaining 45 epochs, an exponential LR decay with an end LR of 1/20 of the base LR is applied. For the following experiments, some pretrained backbone layers are finetuned. Finetuning the backbone weights starts at epoch 10 with 1/10 of the LR that is used for the remaining network. The VOT2019-RGBD dataset is randomly split into 40 training and 40 validation videos. Each sequence is sampled 300 times per epoch for training and 200 times for validation. Shift and scale augmentations are applied for template and search images. Table 4.2 summarizes the training configuration.

The upcoming experiment investigates how pretraining affects the maximum obtained validation IoU when only considering color information. The simple validation tracker strategy (see section 4.2) is used for validation, thus predicting the most probable bounding box without any scale or position change penalties. It is a much simpler version of DaSiamRPN with probably worse performance. However, it reduces the dependance on tracker hyperparameters and is thus more suitable for the pretraining investigations in this section. The experiment consits of two parts. The first part investigates ImageNet pretraining. This is a common practice due to the size of ImageNet and availability of pretrained network parameters. The second part takes a pretrained DaSiamRPN model with an AlexNet backbone from the PySOT Model Zoo[2]. This network has also used ImageNet pretrained weights as initialization. However, the last two layers have then been finetuned on YouTube-BoundingBoxes [72], Microsoft COCO [58], ImagenetDetection [74] and ImageNetVideo [74] on the tracking task. Therefore, it has been trained on even more images on the one hand, and it also might have learned specific features that are primarily useful for the tracking task on the other hand. The PySOT network also contains pretrained weights for the RPN, thus the network has also learned how to make use of extracted features to locate an object. Seven

[2] `https://github.com/STVIR/pysot/blob/master/MODEL_ZOO.md`

Table 4.2.: Training settings for pretraining experiments.

Hyperparameter		Value
Optimizer		SGD
Epochs		50
Batch size		16 per GPU
LR		0.01
WD		0.0
LR warmup	Epochs	5
	Start LR	1/5 LR
	End LR	LR
LR schedule	Type	Exponential
	Epochs	45
	Start LR	LR
	End LR	1/20 LR
Backbone Finetuning	Start Epoch	10
	Finetune factor	0.1
Negative Sampling		0.05
Training Pairs per epoch		12000
Validation Pairs per epoch		8000
Augmentations		shift, scale

architectures have been trained for each experiment. The following description applies to both pretrained networks. As a baseline, one architecture does not use pretrained weights and trains from scratch. The remaining architectures utilize the pretrained weights as initialization, which can either be from ImageNet or from the PySOT model zoo. One of the architectures does not perform finetuning and thus keeps the backbone fixed. The other architectures keep parts of the backbone fixed, while finetuning the last layers. Note that the AlexNet backbone that is used as feature extractor consists of 5 layers. Finetuning the last two layers, while keeping the first three layers fixed, would be denoted as "L45". Figure 4.8 shows the maximum training and validation IoUs obtained by the configurations for both pretraining strategies, namely "ImageNet" and "PySOT".

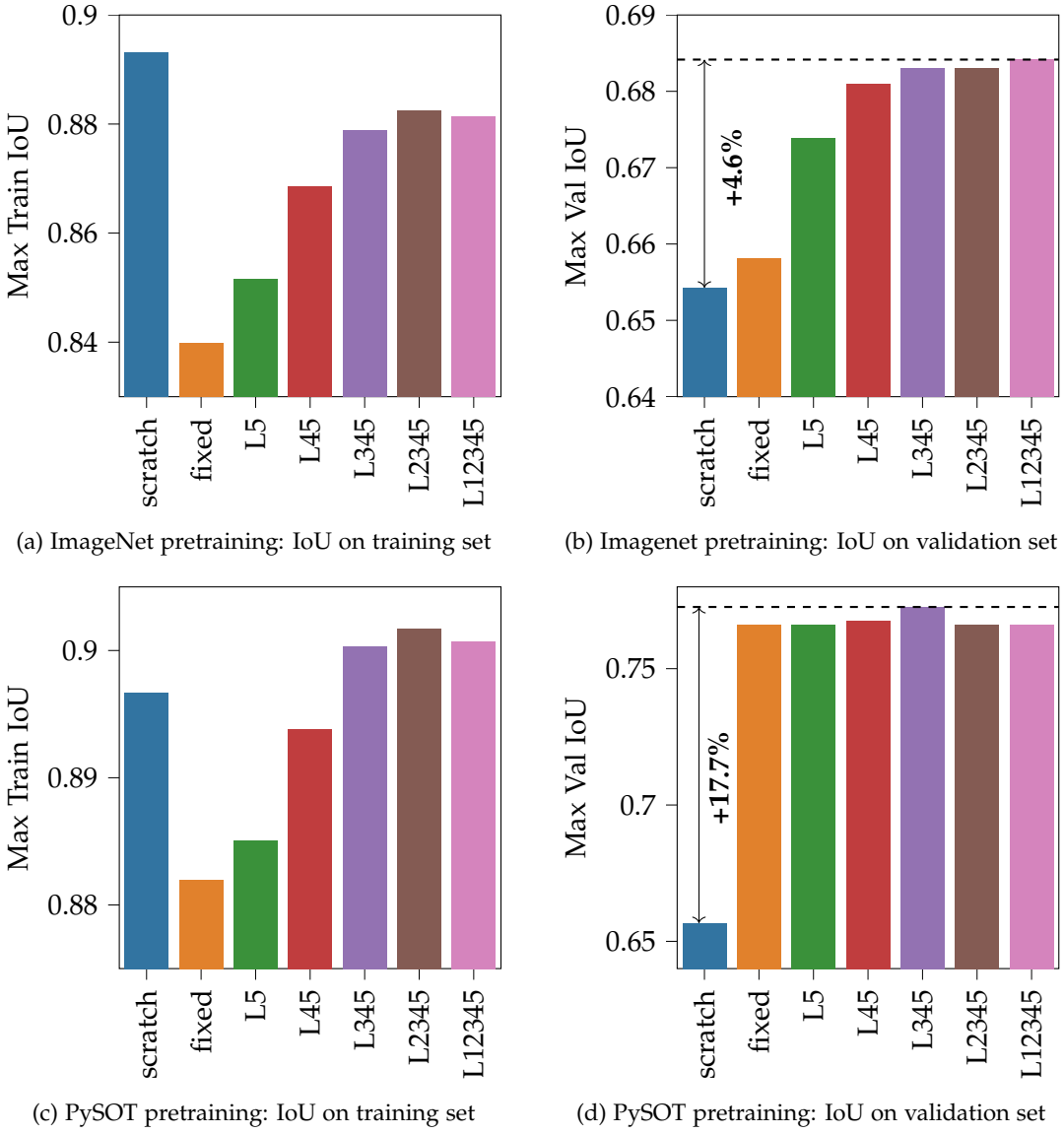
For the ImageNet pretraining experiment, training from scratch leads to the highest training IoU and at the same time to the lowest validation IoU. This seems reasonable as generally taking pretrained weights that have been trained on another dataset might improve generalization to unseen data as well as prevent overfitting to some extent. Therefore, training from scratch allows to better fit the training data, at the cost of validation performance, indicating overfitting. When using the pretrained weights, the performance on the training set is generally lower. The IoU on the training set is the lowest when the backbone is fixed and increases with finetuning more layers. However, the final goal is to get a high score on the validation set. In terms of validation IoU, the obtained score on unseen data is the lowest when training from scratch. Taking the fixed feature extractor leads to an increase of 0.4% IoU. The performance also increases with finetuning more layers. The best score is obtained for configuration L12345 when

finetuning all layers of the backbone, leading to an 3.0% IoU increase compared to training from scratch, resulting in an absolute improvement of 4.6%. Generally, keeping more pretrained layers fixed should be preferred if the performance does not increase with more layers being finetuned to prevent overfitting. Again, the size of the target dataset as well as the similarity to the pretraining dataset (e.g. ImageNet) also impacts the decision on how many layers to finetune. In the present case, finetuning the last three (L345) or all (L12345) layers might be the best option. Looking at the first layer filters of both architectures (see Figure 4.9) shows that there are no significant changes between both, leading to the assumptions that the extracted lower-layer features are very similar. Therefore L12345 might be the best setting as it leads to the highest validation IoU while not severely changing the first layer filters and thus keeping the capability of extracting low level features.

The utilization of the PySOT pretraining shows interesting results in terms of training as well as validation IoU. On the training set, taking the pretrained backbone also lead to worse performance. However, this changes for L345, L2345 and L12345. These configurations lead to a better overlap on the training score compared to training from scratch. In terms of training IoU, even taking the pretrained weights and not finetuning them leads to a remarkable increase of 11.0% IoU (16.7% absolute). Finetuning does not really improve the performance except for configuration L345, which increases the IoU another 0.7%, leading to an absolute improvement of 17.7%. The large gap between ImageNet pretraining (4.6% increase) and PySOT pretraining (17.7% increase) indicates that the PySOT model has learned features that are significantly better suited for tracking, and thus using these weights as initialization leads to a huge performance increase, even when not finetuning the backbone. Intuitively speaking, the network already knows how to extract features that really help for tracking. In addition, the PySOT model has already pretrained the RPN for tracking, and therefore also the RPN profits from transfer learning. This is not true for the ImageNet pretraining, as these networks have been trained on image classification and therefore only the weights for the backbone might be used. In contrast, the ImageNet pretrained backbone knows how to extract general features, but they still need to be finetuned to the tracking domain to really profit from pretraining. However, the performance of the ImageNet initialization is still worse compared to PySOT even with finetuning, given VOT as data.

4.3.2. Simple Approach for RGBD

Figure 4.8 showed that pretraining helps to obtain better performance on the VOT2019-RGBD dataset with RGB inputs, neglecting the depth information. This is especially true when the network has been pretrained for tracking. Generally, this should also be true when considering depth and might also lead to a performance boost. However, as mentioned in section 3.1, pretrained networks are not available for RGBD data. This results from the lack of large-scale generic RGBD datasets compared to the RGB case as well as a lack of challenges and general interest of the computer vision community compared to RGB tasks. Due to the limited size of RGBD datasets and the mentioned drawbacks of SceneNetRGBD (see Table 3.2, subsection 3.3.1), this section investigates the use of RGB pretraining for a tracker that considers both, RGB and depth data.



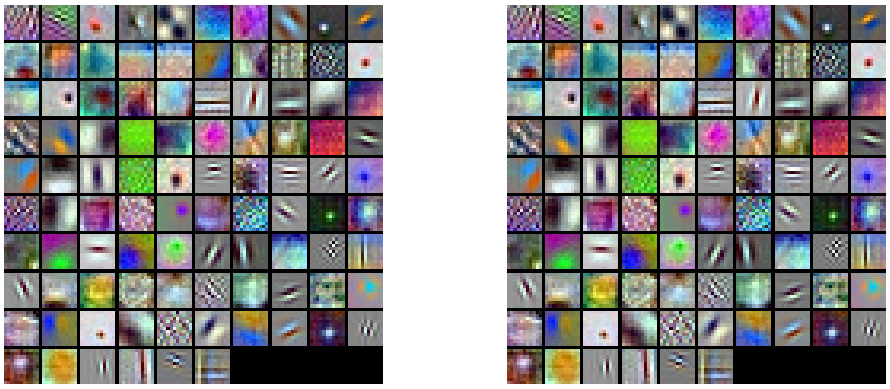
(a) ImageNet pretraining: IoU on training set

(b) Imagenet pretraining: IoU on validation set

(c) PySOT pretraining: IoU on training set

(d) PySOT pretraining: IoU on validation set

Figure 4.8.: Effect of pretraining the DaSiamRPN architecture with RGB-only inputs on training and validation performance, obtained with a simple validation tracker. The network neglects the depth information and thus only operates on RGB input. "Scratch" indicates no pretraining, "fixed" that no finetuning is applied and thus the pretrained weights are kept, and "L[12345]$^{+}$" denotes which layers are finetuned, while the remaining ones are the fixed pretrained weights. (a) and (b) have used ImageNet pretraining, while (c) and (d) have used a pretrained network from the PySOT model zoo, which has been finetuned on tracking with large-scale datasets.



(a) pretrained filters (b) finetuned filters (L12345)

Figure 4.9.: First layer filter visualization of the RGB ImageNet pretrained AlexNet and the configuration L12345 from the RGB ImageNet pretraining experiment. No significant changes can be seen in the first layer filters of architecture L12345.

Specifically, the same ImageNet pretrained weights as in subsection 4.3.1 are used. PySOT weights might again lead to better performance. However, it should not matter too much in this case, as this experiments investigates how to profit the most from RGB pretraining, rather than how to get the highest possible score. Figure 4.10 shows the implemented DaSiamRPN architecture with a 3-channel encoded depth input. When averaging the first layer filters, the depth inputs would consequently be $(127, 127, 1)$ and $(255, 255, 1)$ for template and search images.

Fusion Strategy C1BR

For this experiment, the fusion network in Table 4.3 is implemented. The input to the fusion layer are the stacked features from the RGB and depth domain extracted by the respective backbones. The size of the activation maps for the depth branch is identical to the RGB case. The RPN is not changed. The strategy is called C1BR as it consists of a 1×1 convolution followed by a batchnorm and a rectified linear unit (ReLU) layer. Note that this experiment is a rather simple approach. It is assumed that the same backbone architecture is a good choice for the depth information. Also, the C1BR fusion scheme is rather simple. It is not clear if this is sufficient to combine the information from both domains in a meaningful and profitable way in terms of final validation IoU. However, it allows a first estimate of whether RGB pretraining might also help for depth information. The upcoming sections will further investigate which backbone architecture and which fusion layers to use in order to maximize performance.

Figure 4.11 shows the results of utilizing ImageNet pretraining with AlexNet backbones and the C1BR fusion strategy. Both backbones make use of ImageNet pretraining. When training from scratch, using the depth information by stacking it to RGB leads to an 1.6% increase in IoU (2.4% absolute), thus leading to a better performance with depth. Note that this time, the validation data is half of the VOT2019-RGBD dataset. No

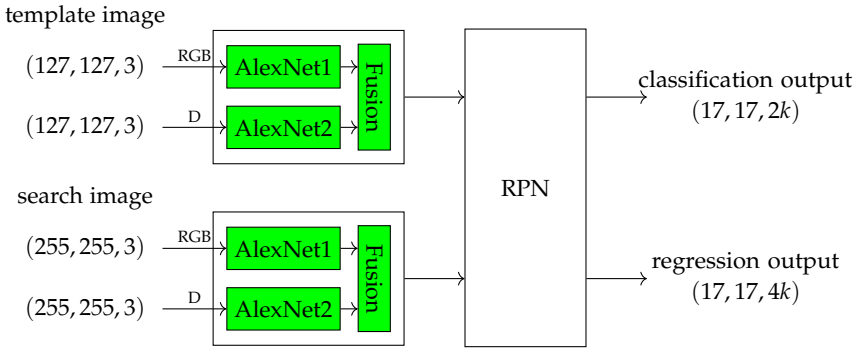


Figure 4.10.: Setup for RGBD training of the DaSiamRPN architecture utilizing ImageNet pretrained weights for AlexNet backbones. One AlexNet is implemented for RGB and depth data, respectively. Both use pretrained weights as initialization. The depth data is encoded to 3 dimensions. The extracted features from both domains are combined with a fusion layer to form an RGBD feature embedding. "D" denotes the depth information. The RPN performs object presence classification as well as bounding box regression based on the RGBD features.

Table 4.3.: C1BR fusion layer. The channel map property describes the number of output and input channels of the convolutional layer. ReLU nonlinearities follow the convolutional layer. Batch normalization is used after every linear layer. The extracted features (255 channels) of both backbones are stacked along the channel dimension to form the 512-channel input.

				Activation size		
Layer	Support	Channel Map	Stride	For Exemplar	For Candidate	Chan.
				6×6	22×22	$\times 512$
conv1	1×1	255×512	1	6×6	22×22	$\times 256$

pretrained weights are available for a 4D RGBD input, thus no pretraining can be used for rgbd_stack. The rgb experiment shows the results from Figure 4.8 for comparison. The maximum IoUs of the configurations rgbd_repl, rgbd_avg and rgbd_jet exceed the ones of rgb and rgbd_stack. This indicates that the implemented architecture (see Figure 4.10) somewhat profits from the ImageNet pretrained weights for the depth domain. Also, increasing the number of finetuned layers in the backbone improves the IoU for all architectures. The best performance is achieved when replicating the depth information to 3D with the configuration L2345, which gives an increase of 2.5% IoU (3.5% absolute) compared to training from scratch with simply stacking the depth information to the RGB data, and an increase of 1.1% IoU (1.4% absolute) compared to the pretrained RGB architecture. Note that the performance gap between rgb and rgbd decreases from 1.6% IoU when training from scratch to 1.1% IoU when applying RGB pretraining to both backbones, indicating that with the current setup, the profit of transfer learning from ImageNet in terms of IoU increase is less than for the RGB case. However, it is not exactly clear how much of the improvement follows from finetuning the RGB architecture, how much from finetuning the depth backbone, and how much from the integration of features from both domains. Taking the maximum validation IoU as examination criterion, encoding the depth information to 3D by replicating the depth channel along the channel dimension outperforms Jet encoding as well as averaging the first layer filters in this experiment.

4.4. Backbones for Depth Information

The first approach of transfer learning from ImageNet on RGBD input data in subsection 4.3.2 already led to an improvement of the validation IoU. Some assumptions have been made, which might limit the potential performance gain that might result from RGB pretrained architectures. Specifically, the fusion strategy was rather easy, and the same backbone architecture was taken for extracting features from depth. However, it is not clear if the same network architecture is a good choice for depth, or if e.g. a simpler shallower network might be sufficient to process the depth information. Therefore, this section investigates the effect of different backbones, only concentrating on depth. The networks again utilize half of the VOT2019-RGBD dataset for training, and the other half for validation. No encodings are applied, the depth information is taken as an 1D input. The objective in this section is to find an architecture that achieves low bias, thus best fitting to the training data. Therefore, the training IoU will be used for examination. Generally, the experiment can be summarized as examining deeper vs shallower networks. The setup is shown in Figure 4.12, only consisting of the backbone for depth processing and the RPN. For the training settings, refer to Table 4.2. In addition to the augmentations in the table, half of the depth images are flipped horizontally.

Figure 4.13 shows the results for different backbone architectures. When taking an AlexNet-like architecture as an example for a deep backbone, taking more layers and thus implementing deeper backbones leads to better performance. When only taking an one layer backbone "L1", the performance drops by a large margin, indicating that this small network is not capable of learning representative features, leading to a maximum

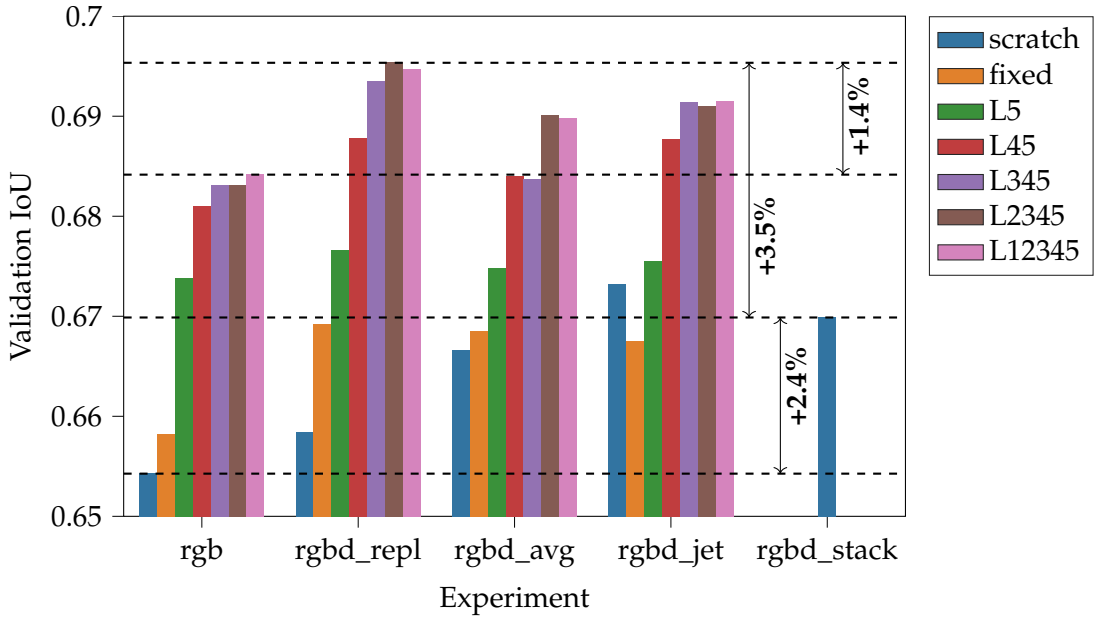


Figure 4.11.: Results of ImageNet-pretraining when considering RGBD inputs. "rgbd_repl" represents the encoding of the depth information to 3D by replicating the depth, "rgbd_jet" uses a Jet colormap as 3D encoding, "rgbd_avg" averages the first layer weights and treats the depth information as 1D. Therefore it modifies the first layer of the backbone. "rgbd_stack" only uses one backbone architecture that processes the stacked RGBD data as a 4D input, while "rgb" reflects the architecture without depth. The latter serves as a reference. "rgbd_stack" does not use pretraining, as no pretrained networks are available for RGBD inputs. Furthermore, "scratch" indicates no pretraining, "fixed" that no finetuning is applied and thus the pretrained weights are kept, and "L[12345]$^+$" denotes which layers of the backbone are finetuned, while the remaining ones are fixed.

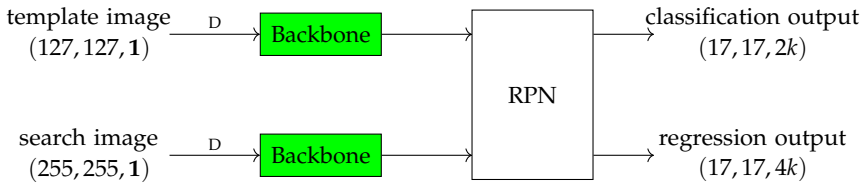


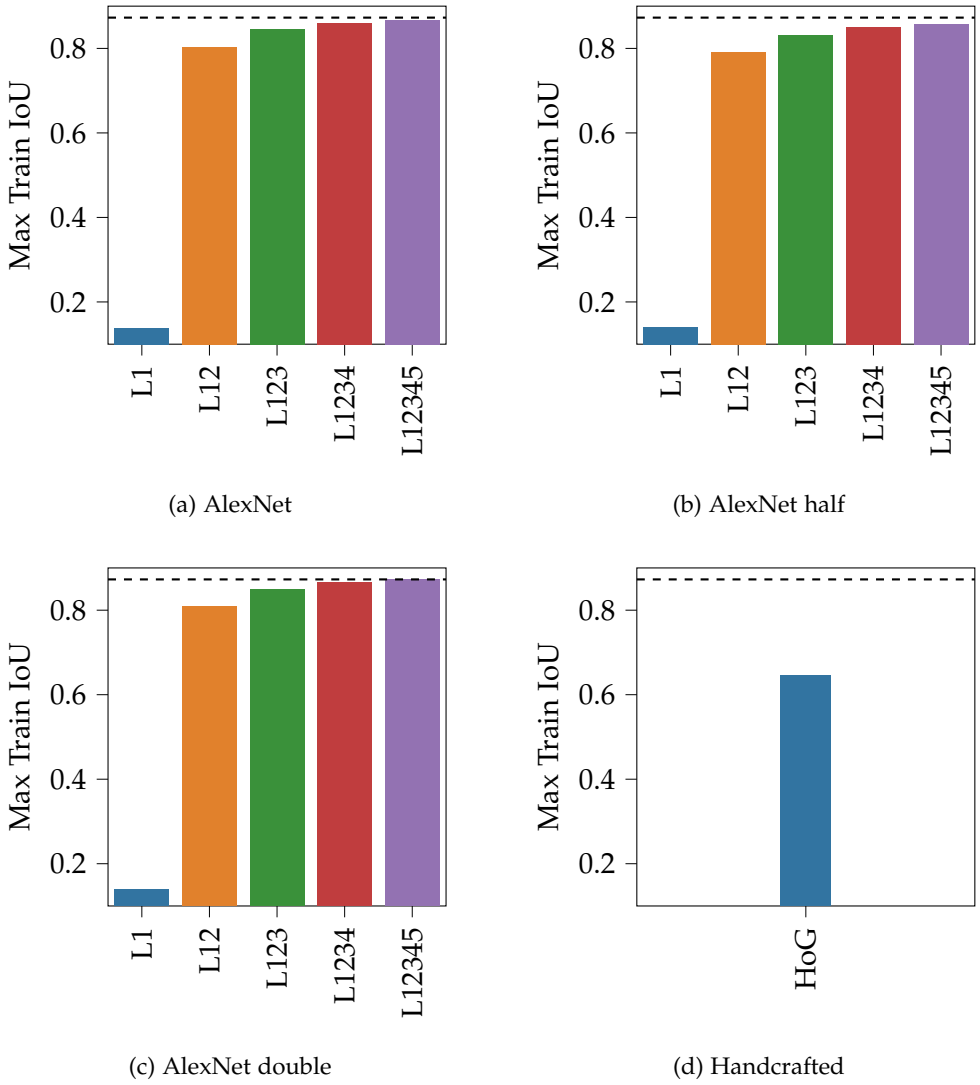
Figure 4.12.: Setup for searching suitable backbone architectures for feature extraction from depth input. The backbone extracts features from template and search image, respectively. Based on these, the RPN predicts foreground-background classification and bounding box regression outputs on anchor level. k reflects the number of anchors. Different architectures can be used as backbone, indicated by "Backbone" as a placeholder

training IoU of less than 10% for (a), (b), and (c). Using two layers already leads to a decent performance of roughly 80% IoU, with even more layers further increasing the maximum training score. The AlexNet half architecture with half the filters in each layer has a similar but slightly worse training IoU compared to the original AlexNet, assuming the same number of implemented layers. This is also true for AlexNet double, however, with a slightly increased performance compared to baseline AlexNet. The best performance is achieved by the models "L12345". More precisely, AlexNet double gives the highest score for this configuration, followed by AlexNet and AlexNet half. Consequently, deeper as well as broader (in terms of filters per layer) networks help to learn better features for tracking from depth information. As a comparison, (d) shows the performance when not taking a deep convolutional network architecture, but rather a handcrafted HOG [16] extractor which has been a common feature extractor in early DCF-based tracking architectures, leading to 64.5% IoU. The performance is significantly lower compared to 87.3% overlap obtained by AlexNet double.

To sum up, the obtained results show a similar trend for depth information as it is the case for RGB data in a variety of computer vision tasks, with deeper convolutional networks being able to extract more meaningful features, which then allow for an overall increased performance. Also, the superiority of a convolutional feature extractor over the handcrafted HOG has been shown. Although the performance has been the highest with the AlexNet double architecture, the standard AlexNet architecture will be considered for further investigations. This seems reasonable as AlexNet double contains double as much parameters and leads to an training IoU increase of 0.6%, which is rather small. However, the significant advantage of utilizing standard AlexNet is the availability of pretrained networks on e.g. ImageNet or PySOT, as already used in section 4.3. This should allow better performance in the long run, when effectively applying transfer learning.

4.5. RGB Pretraining for Depth Information

The last section showed that, indeed, taking an equivalent backbone architecture for depth information works well. This has been assumed in section 4.3 without further investigation, and has been proven to be a reasonable approach. This investigation probably refers to the fact that recognizing edges and low-level patterns helps for both, RGB and depth, to identify and locate an object. Therefore, the underlying structure of both domains has some similarities, which intuitively points at taking a similar function space to search for a good feature extractor. To recap, depth information led to better IoUs when training from scratch compared to RGB-only. Transfer learning helps significantly for RGB input data, especially when taking pretrained networks which have been trained for tracking. Furthermore, a similar or equivalent backbone seems to be a good architecture to extract features from depth data. However, pretraining results for a simple RGBD architecture has not exploited the full potential of the depth data. Therefore, this section will deal with how to apply transfer learning on the AlexNet architecture for depth information. Thus, color information will not be used in this section. It investigates RGB pretraining for the depth network, similar to subsection 4.3.1,



(a) AlexNet

(b) AlexNet half

(c) AlexNet double

(d) Handcrafted

Figure 4.13.: Maximum training IoU for different backbone architectures for feature extraction from depth. (a) uses AlexNet as baseline architecture, (b) a modified version of AlexNet with every layer containing half the amount of filters, and (c) doubles the number of filters in each layer. For each subplot, L[12345]$^+$ denotes how many layers of the baseline architecture are implemented. (d) does not utilize a deep backbone, but rather uses a handcrafted HOG feature extractor for comparison with the convolutional network approaches, as e.g. often used in earlier DCF approaches (see chapter 2). The dotted line shows the maximum achieved training IoU of "AlexNet double: L12345".

with the difference of using depth as input, rather than RGB. Referring to Figure 4.11, replication of the depth information to 3 channels led to the best results when utilizing RGB pretraining for depth data. Thus, this section utilizes replication, copying the depth data to be a 3-dimensional input, and then using pretrained RGB weights. This is possible as now the input dimensions are the same, thus there is no need to adapt the network. Again, the training hyperparameters in Table 4.2 are used. Figure 4.14 shows the general setup.

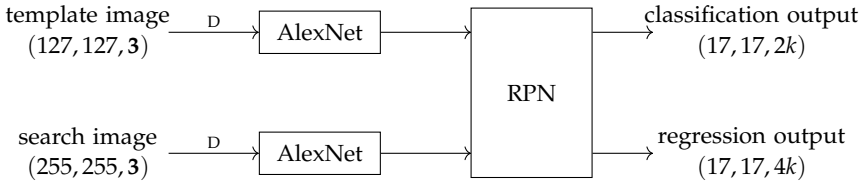


Figure 4.14.: Setup for investigating pretraining methods for depth with the DaSiamRPN architecture. The backbone extracts features from template and search image, respectively. Based on these, the RPN predicts foreground-background classification and bounding box regression outputs on anchor level. k reflects the number of anchors. The depth information is copied along the channel dimension to be 3-dimensional. Therefore no adaption to the backbone is necessary.

Figure 4.15 shows the results of pretraining an AlexNet backbone which takes depth information as input, utilizing RGB pretrained network weights from either training on ImageNet or on large-scale object detection datasets (model from the PySOT model zoo, see subsection 4.3.1). The same notation as for the pretraining experiment with RGB input data is used. Note that training from scratch takes the depth data as 1D input, while the rest of the architectures copy the data to three dimensions. As this is only done to allow using pretrained weights, training from scratch should not follow the same procedure. Thus it modifies the first layer filters to accept 1D input data. When using transfer learning from ImageNet, the training IoU decreases and thus the model is not overfitting the training data as much. At the same time, the validation performance increases, with the best model (L12345) leading to a maximum validation IoU of 61.5%, exceeding the score when training from scratch by 1.3% IoU (2.2% absolute). The performance gain of 2.2% is rather small compared to 4.6% in the RGB case in subsection 4.3.1. However, it becomes clear that transfer learning from the RGB to the depth domain is able to increase performance, thus extracted features from both domains seem to be correlated, and one can profit from the other. Again, remarkably better performance is achieved when applying transfer learning from a model that has been trained for tracking on large-scale object detection datasets. This has also been the case when examining pretraining for RGB inputs (see subsection 4.3.1). Again, the performance increase is significantly also for depth input data, even when not finetuning the backbone at all. The best validation IoU is 67.8%, which means an increase of 7% IoU (11.5% absolute) compared to training from scratch. Recall that the increase with RGB input data has been 17.7% absolute. This indicates that the features in the depth domain are somewhat different. Thus, the performance gain is less with depth data,

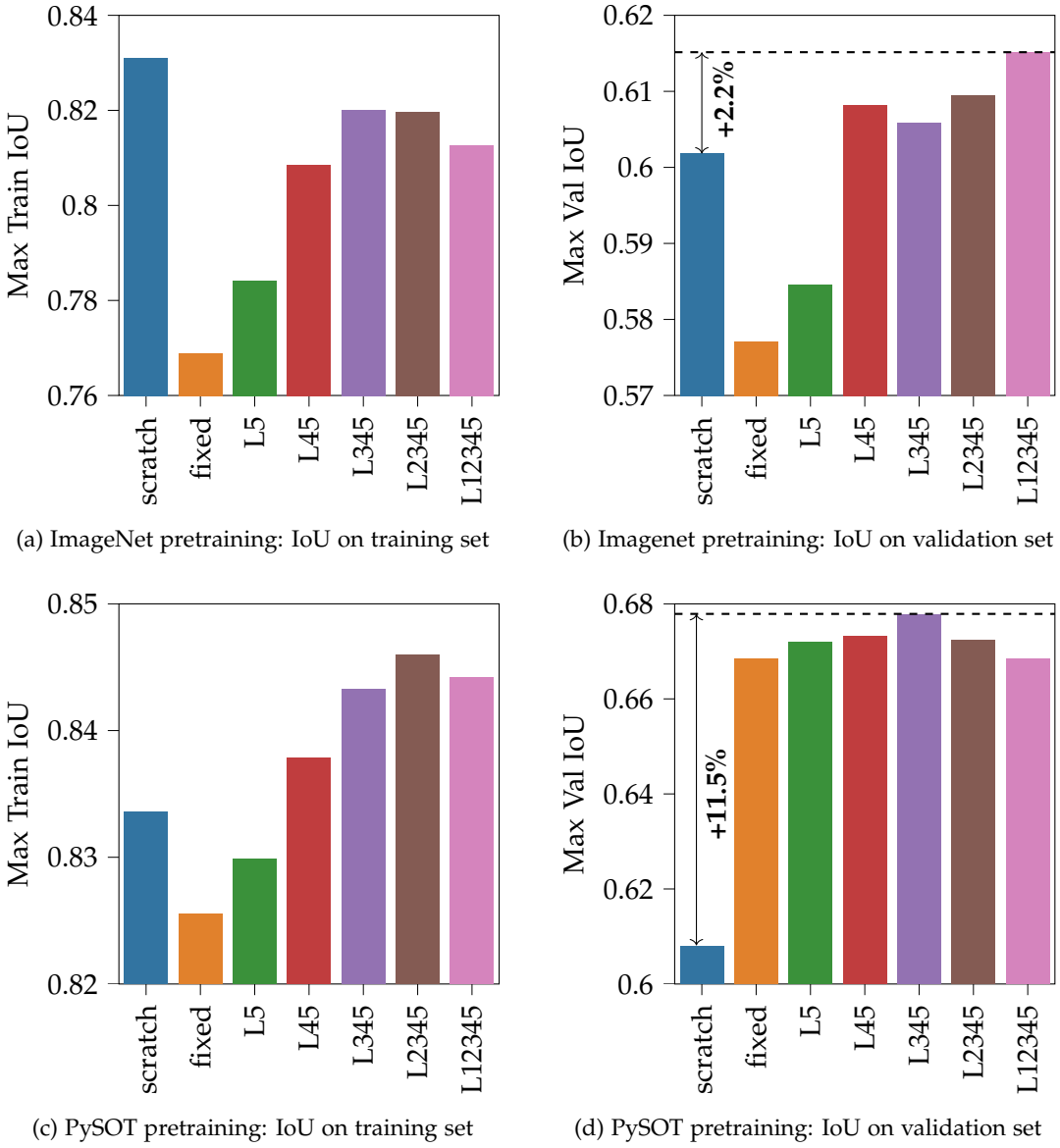
however, still significant. The network is also able to better fit the training data when applying transfer learning from RGB tracking, which emphasizes the impact of having been trained on a large variety of different object categories from several datasets. Similar to the pretraining experiment on RGB inputs, finetuning the last three layers (config L345) achieves the best performance.

To summarize, features learned on RGB data transfer well to the depth domain, however with less performance gain compared to an architecture that processes RGB inputs. This indicates similarities of features in both domains, however, only to some extent. According to these findings, applying RGB pretraining should definitely be considered when training an RGBD tracker, as the backbone that extracts features might profit significantly. This is especially important in the current situation, where large-scale datasets with depth information are rare. It might be the case that in the future, RGBD datasets will be large enough to train deep backbones from scratch. However, as this is not true at the moment, transfer learning from RGB is a good practice, especially when utilizing networks that have been trained for RGB tracking on rather large datasets.

It has been proven that transfer learning helps for backbones with RGB input data as well as depth. However, the realtive performance gain by applying transfer learning is 56.7% less compared to the RGB case for ImageNet pretraining, and 35.8% less compared to the RGB case for PySOT pretraining. Note, however, that it clearly improves performance in both cases. The next section investigates fusion strategies to profit from color and depth information.

4.6. Fusion Strategies for Leveraging RGB and Depth Encodings

Different backbones as well as pretraining strategies have been thoroughly analysed in the last sections. As transfer learning from RGB to the depth domain is crucial to achieve good performance especially with small RGBD datasets, desinging an architecture with two backbones and a fusion layer is a reasonable strategy to utilize pretrained networks. The first approach of implementing this architecture in subsection 4.3.2 used a C1BR fusion strategy, which was rather simple and might not unleash the full potential of combining features extracted from both domains. Therefore this section investigates different fusion architectures which allow the tracker to learn embedded features from RGB and depth. The training uses the same data as in the last sections, splitting the VOT2019-RGBD dataset by half for training and validation, respectively. The backbones are initialized with pretrained weights from the PySOT model zoo. As encoding the depth information to form a 3D input by replication lead to the highest performance when utlizing a pretrained network (see Figure 4.11), this technique is also implemented in this experiment. Being pretrained on RGB tracking has been proven to achieve significantly better performance for the case of RGB (see Figure 4.8) as well as depth (see Figure 4.15) inputs compared to ImageNet pretraining. The backbones are intitialized with the pretrained weights and kept fixed during training in this experiment. The examined fusion layers in this section as well as the RPN are trained from scratch. The experiment generally follows the setting in Table 4.2. However, a WD of 0.0001 is applied.



(a) ImageNet pretraining: IoU on training set

(b) Imagenet pretraining: IoU on validation set

(c) PySOT pretraining: IoU on training set

(d) PySOT pretraining: IoU on validation set

Figure 4.15.: Effect of pretraining the DaSiamRPN architecture with depth-only inputs on training and validation performance, obtained with a simple validation tracker. The network neglects the color information and thus only operates on depth input. "Scratch" indicates no pretraining and 1D input data, "fixed" that no finetuning is applied and thus the pretrained weights are kept, and "L[12345]$^{+}$" denotes which layers are finetuned, while the remaining ones are the fixed pretrained weights. (a) and (b) have used ImageNet pretraining, while (c) and (d) have used a pretrained network from the PySOT model zoo, which has been finetuned on tracking with large-scale datasets. All networks except for "scratch" copy the depth data along the channel dimension to form a 3D input vector, that allows to take the same backbone as for RGB.

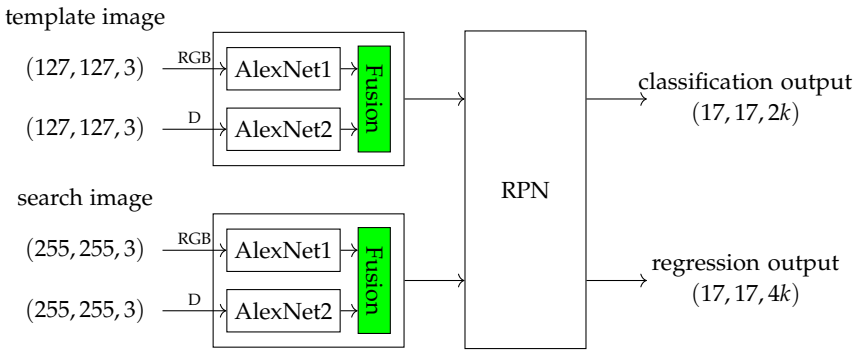


Figure 4.16.: Setup for finding good fusion architectures using the DaSiamRPN architecture with AlexNet backbones. One AlexNet processes RGB and depth data, respectively. Both use pretrained PySOT weights as initialization. The depth data is encoded to 3 dimensions by replication. The extracted features from both domains are combined with a fusion layer to form an RGBD feature embedding. "D" denotes the depth information. The RPN performs object presence classification as well as bounding box regression based on the RGBD features.

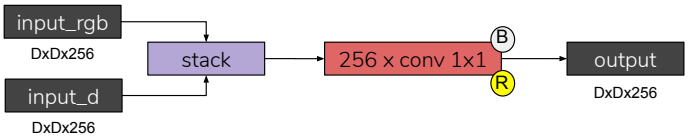
Apart from that, backbones are fixed. Instead of having an exponential LR decay, the LR in these experiments is kept constant apart from a warmup schedule in the first 5 epochs. The optimizer uses a base LR of 0.005.

4.6.1. Fusion Layers

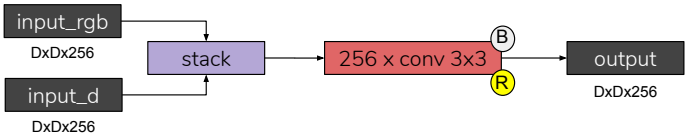
This subsection examines the use of different fusion layers which differ in the number of layers, the number of trainable parameters as well as the complexity of the fused output RGBD features. Table 4.3 introduced the C1BR fusion scheme. Figure 4.17 shows all implemented fusion layers. Despite having been introduced before, C1BR is also shown for completeness. The extracted features from both, RGB and depth backbone, feed into the fusion layer (see Figure 3.2). All schemes concatenate the feature maps to form a 512 dimensional input volume. The fusion schemes are referred to by their name as defined in Figure 4.17. The efficient layers are a modified version of their baseline counterpart. 1×1 convolutions reduce the channel dimension. This significantly reduces the number of parameters and therefore computational complexity. Table 4.4 lists the number of trainable parameters for each fusion network.

4.6.2. Results

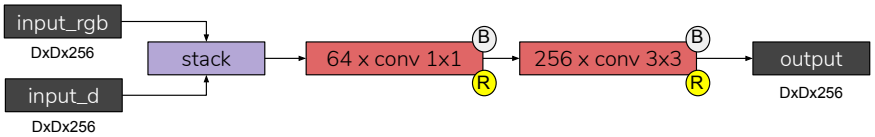
Figure 4.18 shows the results of the fusion experiment. "C3BR efficient" leads to the best validation IoU of 70.7%, followed by C3BR, $[\text{C3BR}]^2$ efficient, and C1BR. Note that when taking the validation loss into account, which aggregates loss over 16 positive and 48 negative bounding boxes (see section 3.2), the best three architectures to use are the efficient variants of the fusion layers $[\text{C3BR}]^2$ efficient, followed by $[\text{C3BR}]^3$ efficient and C3BR efficient. Altogether, C3BR efficient as well as $[\text{C3BR}]^2$ efficient seem to be the best



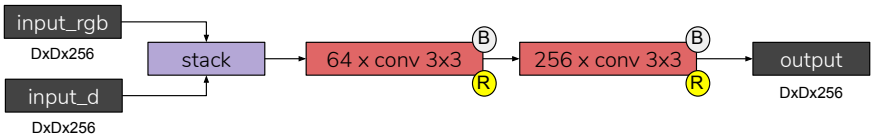
(a) C1BR.



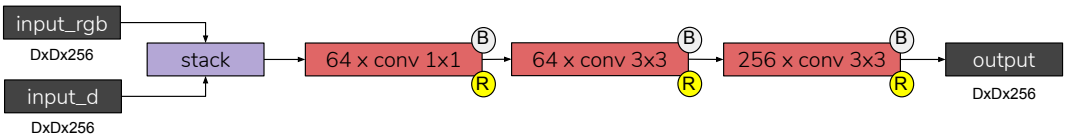
(b) C3BR.



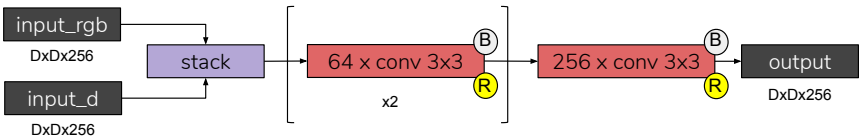
(c) C3BR efficient.



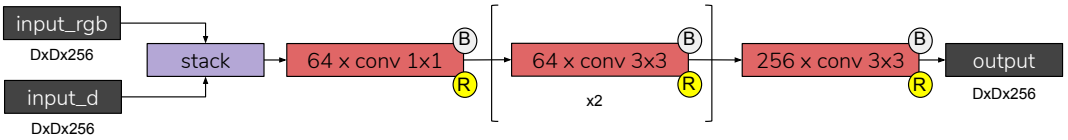
(d) $[C3BR]^2$.



(e) $[C3BR]^2$ efficient.



(f) $[C3BR]^3$.



(g) $[C3BR]^3$ efficient.

Figure 4.17.: Visualization of fusion layers. "D" denotes the input feature size, which is 22 for the search and 6 for the template image. Red boxes represent conv layers, defining the number of filters as well as the filter size. Circled "B" and "R" denote Batchnorm and ReLU layers that follow the convolutions. The output layer feeds into the RPN.

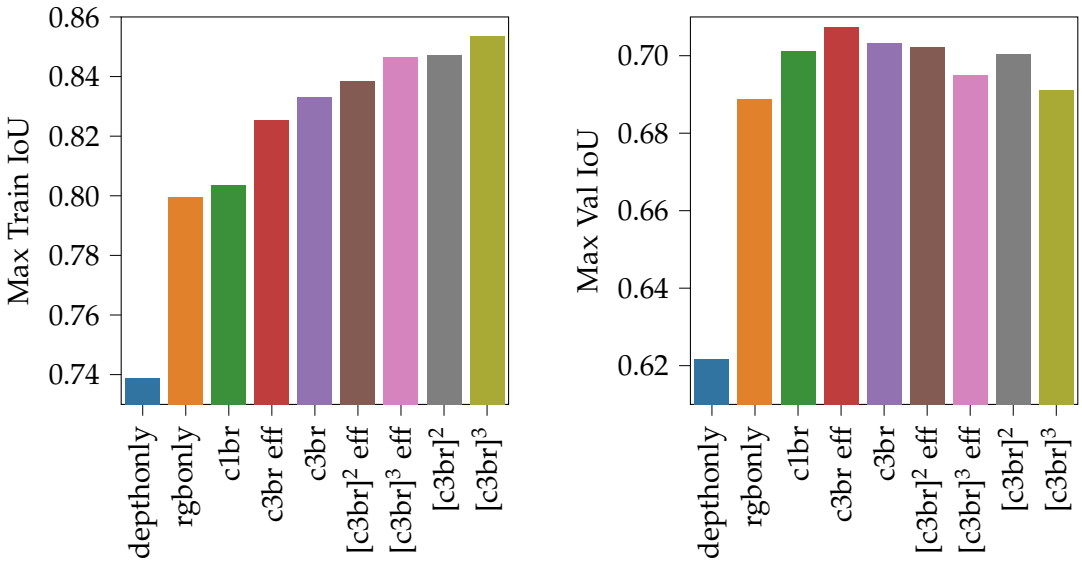
Table 4.4.: Number of trainable parameters for different fusion networks.

Network	Trainable Parameters
C1BR	131,584
C3BR	1,180,160
C3BR efficient	180,864
$[C3BR]^2$	443,008
$[C3BR]^2$ efficient	217,856
$[C3BR]^3$	480,000
$[C3BR]^3$ efficient	254,848

architectures. They should both be tried when training the architecture end-to-end for deployment. Also note that the choice of fusion is also influenced by the availability of RGBD data. With more data being available, bigger fusion architectures with more parameters can be used without the risk of overfitting too much to the training data. However, as RGBD data is still rather rare, this supports the observations of the efficient layers with fewer parameters leading to better results. As expected, the depthonly network achieves the worst performance on training and validation set, followed by the rgbonly network. This shows that combining the information from both domains allows to fit the training data better. Furthermore, it not only helps to overfit, but also increases the IoU on the validation set for all fusion layers, approving that considering depth information in a fusion architecture is important for tracking. Specifically, C3BR efficient leads to an increase of 8.6% IoU (13.8% absolute) compared to depthonly, and an increase of 1.8% IoU (2.7% absolute) compared to rgbonly in terms of validation IoU.

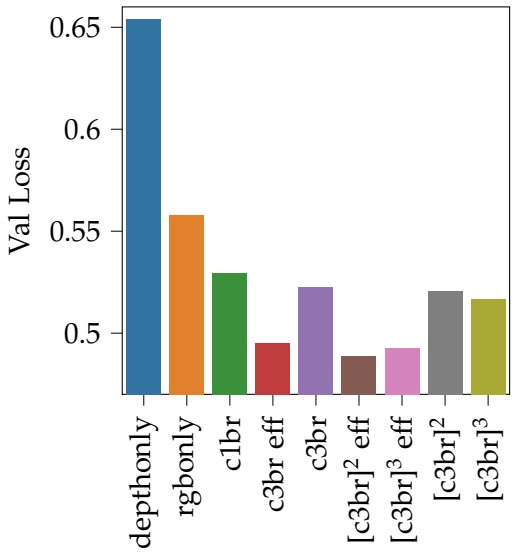
4.7. Benchmark Evaluation of DaSiamRPN vs. DaSiamRPN_D

Finally, this section makes use of the preceding results to implement the RGBD version of DaSiamRPN, further referred to as DaSiamRPN_D. In contrast to all sections before, VOT2019-RGBD is no longer used for training to comply with the VOT participation guidelines. Therefore, this section will train on some other RGBD dataset and evaluate on the full VOT2019-RGBD consisting of 80 sequences. Referring to Table 3.2, SceneNetRGBD might be the best candidate mainly due to its size, which should allow to train a tracker without severe overfitting to specific videos. The architecture incorporates findings from prior sections. Evaluation is performed twice. Once per validation IoU on the VOT dataset, and second according to the tracking F1 score (see section 4.1), which is the official evaluation measure for the VOT-RGBD subchallenge. The RGB-only architecture from the PySOT model zoo serves as a reference. As in the previous sections, an AlexNet backbone serves as feature extractor.



(a) IoU on training set

(b) IoU on validation set



(c) Loss on validation set

Figure 4.18.: Training different fusion architectures with backbones being initialized with weights from the PySOT model zoo. "Depthonly" as well as "rgbonly" are added as baselines. They only consist of the depth/rgb network for feature extraction and therefore do not need a fusion network.

4.7.1. Model Zoo DaSiamRPN

The model zoo tracker with AlexNet backbone called "siamrpn_alex_dwxcorr" is provided through the PySOT model zoo website[3]. It uses RGB data and has been trained end-to-end on YouTube-BoundingBoxes [72], Microsoft COCO [58], ImagenetDetection [74] and ImageNetVideo [74] while making use of ImageNet pretraining. The tracker configuration for the long-term strategy (see section 3.2) has been taken from the long-term model configuration which utilizes a ResNet backbone (siamrpn_r50_l234_dwxcorr_lt). The configuration is also published in the model zoo. Specifically, no hyperparameter search has been performed on the tracker parameters and the same parameters are used for different models. In the future, tuning these hyperparameters would allow to improve the performances of the trackers. As the goal is to highlight tendencies rather than tweaking the performance of every single architecture to the optimum, utilizing the same parameters for multiple architectures is sufficient. Table 4.5 shows the performance of the model zoo baseline. Analogous to previous sections, the validation IoU results from randomly sampling search and candidate images and evaluating the IoU of the most probable bounding box prediction with the ground truth box multiple times for all videos. However, compared to earlier sections, all videos in the VOT2019-RGBD dataset belong to the validation set.

Table 4.5.: Performance measures and validation loss of the DaSiamRPN model zoo tracker on VOT2019-RGBD.

Evaluation Metric	Score
F1	0.451
Validation IoU	0.677

4.7.2. Training DaSiamRPN_D

The architecture consists of two AlexNet backbones with a C3BR efficient fusion layer to combine RGB and depth features (see Figure 4.17). The backbones are initialized with the weights from the model zoo and fixed during training. Therefore, only the fusion layer and the RPN are trained. SceneNetRGBD is used for training. The dataset provides a training and a validation set. The latter serves as bridge set (terminology see subsection 4.2.4). VOT2019-RGBD is utilized as validation data, as achievinging a high tracking F1 score on this dataset is the overall goal. Similar to the recent introduction of a bridge set in the data distribution subsection, this will allow to identify data mismatch apart from overfitting. Several experiments have been evaluated with the outlined architecture. In addition, each experiment is also performed on the baseline model from the model zoo. Again, this architecture is RGB-only and therefore has no fusion layer. It uses pretrained backbone weights and trains the RPN. The first experiment (RPN scratch) trains the RPN from scratch without data augmentation techniques. The next experiment (+augmentation) adds data augmentations to both, RGB and depth data.

[3] https://github.com/STVIR/pysot/blob/master/MODEL_ZOO.md

The augmentations are only performed on the training set, not on the bridge, nor on the validation set. The third experiment (RPN pretrained) uses pretrained weights for the RPN. The last experiment (+more objects) uses more of the objects that are present in the SceneNet dataset, as the prior experiments have only used approximately half of the overall objects. Each experiment is performed once for the DaSiamRPN_D architecture (named C3BR eff according to the fusion scheme) and once for the RGB-only model (named PySOT Zoo). The last experiment also evaluates a C3BR efficient fusion layer with a smaller hidden size, specifically taking 16 1×1 conv filters rather than 64. This reduces the number of trainable parameters in the fusion layer from $180,864$ to $45,600$. All experiments are performed on a system with 2 GPUs. Table 4.6 lists the training settings.

Table 4.6.: Training settings for DaSiamRPN vs. DaSiamRPN_D experiments.

Hyperparameter	Value
Optimizer	SGD
Epochs	50
Batch size	32 per GPU
LR	0.01
WD	0.0001
LR schedule	Constant
Negative Sampling	0.05
Training Dataset	SceneNetRGBD
Bridge Dataset	SceneNetRGBD
Validation Dataset	VOT2019-RGBD
Training Pairs per epoch	84328
Bridge Pairs per epoch	2000
Validation Pairs per epoch	8000
Augmentations	Gaussian Blur, Gaussian Noise, Brightness, Contrast

4.7.3. Results

Table 4.7 shows the results of the conducted experiments. The best overall F1 score is obtained by the baseline that has not been trained on SceneNet. Even finetuning the RPN of the RGB architecture leads to significantly lower F1 scores, indicating that the training data distribution is different to the validation data. Therefore, the C3BR architecture does not achieve superior performance on the VOT-RGBD dataset, but on data that is similar to the training distribution (bridge set). The results reveal interesting relationships. For each experiment, the RGBD model has better performance in terms of IoU on training and bridge set compared to PySOT Zoo. On the validation set, the contrary is true, with the RGB architecture outperforming RGBD. This clearly identifies a data distribution problem. The RGBD model fits the training data better, and it also does generalize better to unseen data that comes from the same distribution, e.g. the bridge set, by a large margin. As the training data is not diverse enough, the RGB

Table 4.7.: Results for DaSiamRPN vs. DaSiamRPN_D experiments. The baseline configuration has not been trained and shows the performance of the model zoo network. Experiments are separated by horizontal lines. For each experiment, bold numbers indicate the best score on each measure for a specific experiment. C3BR eff stands for the model with depth information, while PySOT Zoo means finetuning the RPN of the RGB-only architecture. Performances are measured on training set (Train IoU), bridge set (Bridge IoU), and validation set (Val IoU, F1). The table shows the lowest F1 score obtained by the three model snapshots that lead to the lowest validation losses during training.

Configuration	Model	Train IoU	Bridge IoU	Val IoU	F1
baseline	PySOT Zoo	–	–	0.677	0.451
RPN scratch	C3BR eff	**0.808**	**0.843**	0.593	0.251
	PySOT Zoo	0.717	0.781	**0.680**	**0.379**
+augmentation	C3BR eff	**0.771**	**0.812**	0.626	0.249
	PySOT Zoo	0.715	0.773	**0.690**	**0.386**
+RPN pretrained	C3BR eff	**0.783**	**0.822**	0.648	0.299
	PySOT Zoo	0.708	0.777	**0.704**	**0.400**
+more objects	C3BR eff	**0.698**	**0.746**	0.651	0.274
	C3BR eff_16	0.677	0.728	0.634	0.279
	PySOT Zoo	0.602	0.681	**0.703**	**0.405**

model performs better on VOT (validation set) in terms of mean IoU as well as F1 score. One hypothesis to explain this is that depth adds another data channel, which again adds another source of dissimilarity between training and validation data distributions. Training the RPN from scratch leads to the worst validation IoU for both, C3BR eff and PySOT Zoo. Adding augmentation, using pretrained weights for the RPN as well as adding more objects to the training set consistently improves the validation IoU for both architectures. This is not always true for the F1 score. Achieving a higher mean IoU reflects better average localization of objects. However, the F1 score can still be worse if in a few of the validation videos, a model leads to target loss for the rest of the sequence. Reducing the hidden size of the fusion layer, as in C3BR eff_16, is worse on train, bridge and validation set, indicating that this fusion scheme does not add enough complexity to the model.

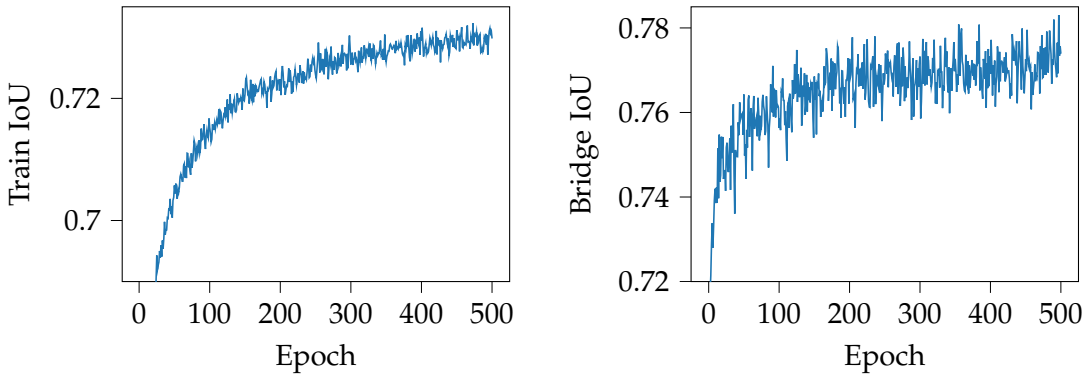
One more experiment will take the configuration of the last experiment (+more objects) and perform the training for 500 epochs to ensure that the model has converged and will not become better with further training given the available data. The training took 8 days on a 2 GPU setup. Figure 4.19 visualizes the training process. The training IoU as well as the bridge IoU consistently improve, indicating that performance on data similar to the training data still improves. This hints that further training might be necessary to make the training converge. This even holds for the validation IoU, meaning that the overlap of the most probable bounding box with the ground truth increases also on the VOT2019-RGBD validation data. However, for the validation set, optimization of the model on the training data does not lead to a minimization of the validation loss. The noise again hints at data distribution problem, as small updates with respect to the training data lead to big improvements as well as degradations in terms of validation loss. Still achieving a higher validation IoU means that the best bounding box gets better, while other bounding boxes might either be classified wrong or get worse at predicting the object location. The highest achieved validation IoU is 68.2%, which is 0.5% IoU (0.7 % absolute) better than the baseline model zoo model (see Table 4.7). The models with the highest validation IoU (epoch 472) as well as the lowest validation loss (epoch 402) are picked for F1 evaluation (see Table 4.8). Although the model with depth

Table 4.8.: Long training results of DaSiamRPN_D on VOT2019-RGBD data in a supervised learning setup.

Configuration	Model	Val IoU	F1
baseline	PySOT Zoo	0.677	**0.451**
+more objects (epoch 402)	C3BR eff	0.670	0.358
+more objects (epoch 472)	C3BR eff	**0.682**	0.319

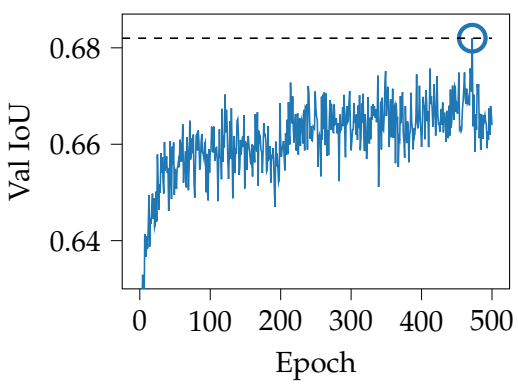
information is able to achieve a higher average overlap when considering all frames in the VOT2019-RGBD dataset, the F1 score is higher for the baseline. This is due to target loss in videos.

These results sharply reveal the importance of having very large and diverse datasets for training RGBD architectures. This is even more important than in the RGB case

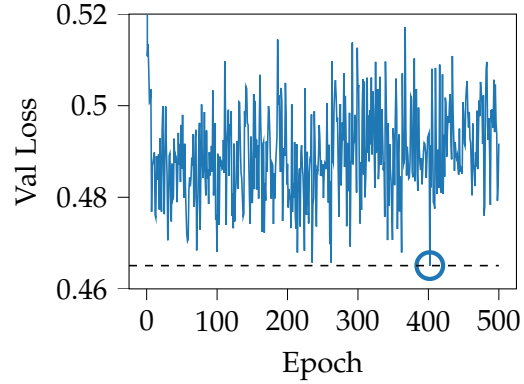


(a) Train IoU.

(b) Bridge IoU.



(c) Val IoU.



(d) Val loss.

Figure 4.19.: Training DaSiamRPN_D for 500 epochs on SceneNetRGBD. Train and bridge set are taken from the SceneNetRGBD dataset, while VOT2019-RGBD is the validation set. The circles mark the epoch with the highest validation IoU as well as lowest validation loss, which are evaluated for F1 tracking score.

due to depth information as an additional source of dissimilarity. As such datasets are not available so far, self-supervised pretraining is a promising option to cope with this problem until labeled datasets are available at large scale. As soon as large-scale datasets become available, this experiment shows that the RGBD architecture will outperform its RGB counterpart.

4.8. Alternate Approach: Self-Supervised Pretraining

Prior experiments focused on training an RGBD tracking architecture in a supervised fashion. Many insights have been found, e.g. how to preprocess depth data, what architecture to use when training with depth, how to use RGB pretraining for depth, as well as what approach works well when considering RGB and depth data as joint inputs to one architecture. However, the results in section 4.7 clearly identified data mismatch between training and validation data to be the main concern with training the DaSiamRPN_D tracker on SceneNet with the goal to achieve good performance on the VOT2019-RGBD subchallenge. To recap, the RGBD architecture showed superior performance over the RGB version on the training and bridge set. Therefore, the derived architecture outperforms the RGB tracker on data that comes from the same data distribution as the data it has been trained on. However, in terms of performance on the VOT2019-RGBD dataset, the situation is reversed, with the RGB version outperforming its RGBD counterpart. As SceneNetRGBD is by far the biggest labeled RGBD dataset that can be used for training a generic RGBD tracker, taking more data and even bigger datasets into account is not possible by the time of writing. However, with bigger datasets becoming available, the architecture from section 4.7 might have the potential to deliver state-of-the-art results when training in a supervised setup.

4.8.1. Datasets

In addition to the labeled dataset, NYU Depth v2 (see Table 3.2) also provides a bigger raw dataset with image and depth videos. The biggest labeled RGBD dataset from Table 3.2 is SceneNetRGBD. However, as mentioned earlier, the data is rendered and therefore it is not clear how well the data represents data from real camera setups as e.g. in the VOT2019-RGBD challenge. Apart from that, Princeton dataset is the biggest in terms of videos (100 vids), while VOT2019-RGBD is the biggest in terms of number of images (101,956 imgs). The preprocessed unlabeled NYU dataset has 407,024 images from 582 videos and therefore exceeds the size of labeled captured RGBD datasets. For details on how to preprocess the raw dataset, refer to section A.2. Note that for Princeton, the ground truth data is not available. While it can therefore not be used for supervised training, it can be leveraged in an unsupervised training setup. The self-supervised setups in this section will mainly use NYU Depth v2 for training due to its size. However, incorporating even more datasets is likely to further improve the results. As more data is always a way of regularization, it most often helps the network to generalize better to unseen data.

4.8.2. Reconstruction Learning

Figure 4.20 shows the setup. The RGBD data is treated as a 4D input vector. An AlexNet backbone with modified first layer filters to allow 4D inputs extracts features. The pretext head network is an AlexNetDecoder network that outputs a $(255, 255, 4)$ matrix. The input as well as the output from the pretext head feed into a Mean Squared Error (MSE) loss function. Thus, this is a reconstruction task, where the overall architecture learns to reproduce the input. This is a non-trivial task, as the AlexNet architecture projects from pixel-space to feature-space, and the AlexNetDecoder tries to map features back to the pixel-space. It is referred to as reconstruction learning.

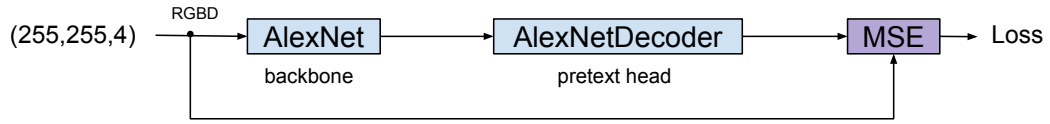


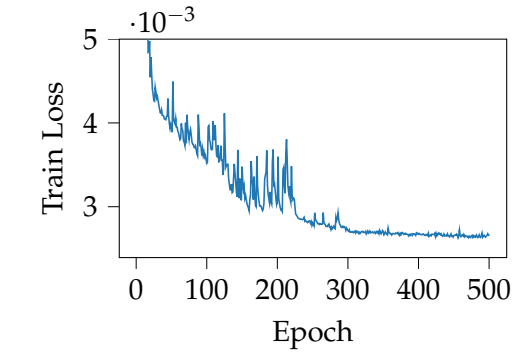
Figure 4.20.: Self-supervised reconstruction learning.

Training Settings

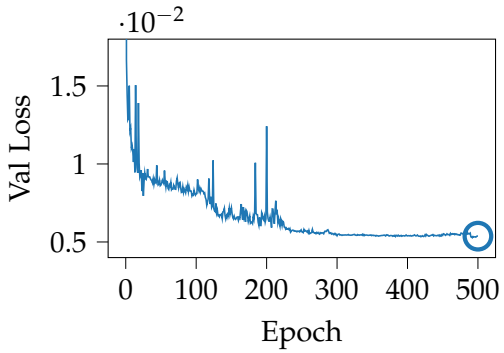
Pretext training uses SGD as optimizer with a multi-step LR decay. The base learning rate is 0.05, and it is divided by 5 at epochs 25, 225, and 300. WD is set to 0.0001. Training takes place for 500 epochs on the NYU-Depth v2 dataset with 29500 videos per epoch. Batch size is 16 per GPU. Finetuning for tracking uses adam optimizer with a constant LR of 0.0003. Training is performed for 150 epochs on SceneNetRGBD. The remaining settings for the second step are as in Table 4.6. When initializing the backbone with weights from self-supervised pretraining, finetuning the backbone starts at epoch 30, always with 1/10 of the LR of the rest of the architecture. When training from scratch the same learning rate is applied for the whole architecture. Finetuning for tracking uses the SceneNetRGBD dataset (train and bridge set). For pretext training and finetuning, VOT2019-RGBD serves as the validation set. The network does not need a fusion layer as it directly gets the 4D data as input.

Results

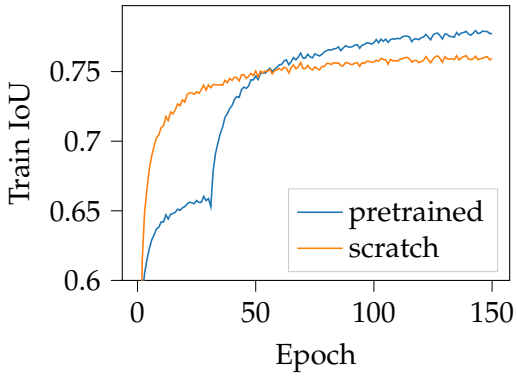
Figure 4.21 shows the pretext training as well as finetuning for tracking. The loss decreases for both, training and validation, indicating that the network learns a good mapping from pixel-space to features back to pixels. The model snapshot of epoch 500 is taken as initialization for the backbone when finetuning for tracking. Another model trains from scratch for comparison. When only training the RPN, training from scratch seems to have better performance. However, when starting to finetune the backbone at epoch 30, the performance of the pretrained model clearly exceeds the baseline in terms of IoU for training, bridge and validation set, by a significant margin. The validation IoUs have been smoothed with a 1D gaussian to cope with outliers. The validation set is checked once per epoch. Comparing smoothed versions for both should be fair, as otherwise it could happen that one model has been tested once when



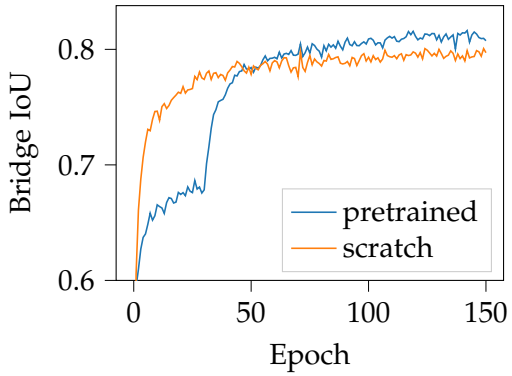
(a) Pretext training loss.



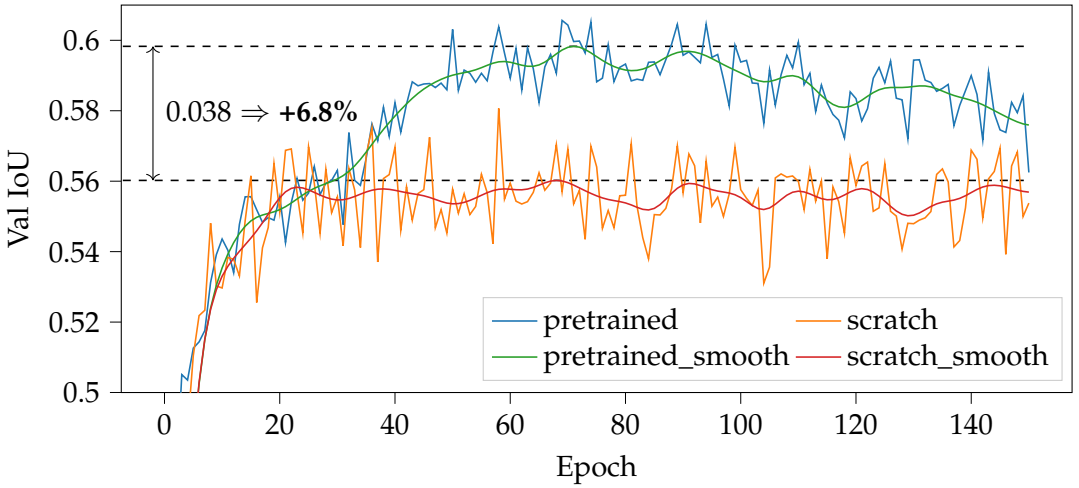
(b) Pretext validation loss.



(c) Tracking training IoU.



(d) Tracking Bridge IoU.



(e) Tracking Validation IoU.

Figure 4.21.: Results for reconstruction learning. (a) and (b) show pretext learning, while (c), (d), and (e) display finetuning the pretrained architecture for tracking vs training from scratch. The validation IoUs are smoothed with a 1D gaussian filter with $\sigma = 3$. The circle marks the epoch that is taken for finetuning.

the weights perform well on the validation set, while the other one has had a similar situation, however just not exactly when the validation IoU has been tested. Therefore, comparison of the smoothed values should give more insight into how much one model outperforms the other. The maximum smoothed validation IoU of the model that uses self-supervised pretraining exceeds the scratch model by 3.8% IoU, which makes an absolute improvement of 6.8%. This results is remarkable. It means that reconstruction learning with the given data is able to improve RGBD tracking by training on data without any labels by a large margin. As even more unlabeled RGBD data should be available in future applications, this finding can help to significantly improve the performance without being dependant on huge labeled datasets.

4.8.3. Domain Transfer Learning

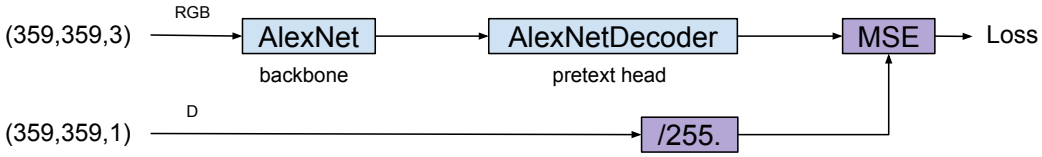
This subsection treats another self-supervised learning approach which is further referred to as domain transfer learning. The overall architecture has one network for RGB and one network for depth data together with a fusion layer, as in previous sections. The self-supervised pretraining itself consists of two separate subtrainings:

- Learn features from RGB that are meaningful for depth
- Learn features from depth that are meaningful for RGB

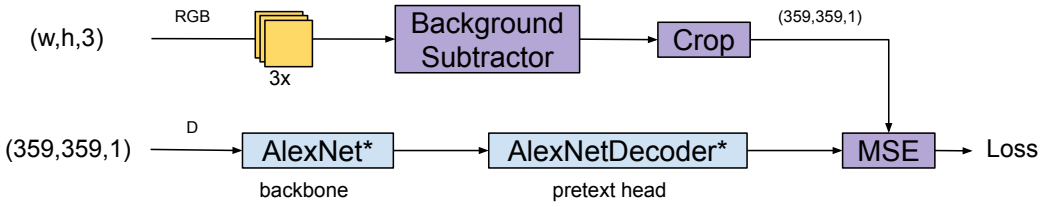
This is somewhat more complex compared to reconstruction learning, as it consists of two trainings rather than one. The intuition behind is to separately learn features from one domain that are useful for the other one. This forces the representation to have a causality between RGB and depth, which might turn out to be useful when used for tracking in the second step. The correlation of learning features from one domain, that transfer to the other one leads to the name domain transfer learning.

Foreground Background Segmentation

Figure 4.22 shows the setup for depth prediction from RGB as well as foreground background prediction from RGB. More specifically, "RGB to depth" tries to predict the depth image given the RGB image. Inputs to the network are images that are cropped to a size of $(359, 359, x)$. In that way, the backbone learns to extract features from the RGB image that are meaningful for depth. On the other hand, "depth to foreground background segmentation" tries to learn a segmentation of foreground and background from the RGB image given the depth information. More specifically, the last three RGB images of a frame from one video are captured and fed into an algorithm which performs background subtraction based on the current and the last two frames. Inputs to the architecture are a cropped $(359, 359, 1)$ depth image and the full size RGB image. After background subtraction, pixel values that are bigger than 64 contribute to the foreground mask. The output is postprocessed by applying an erosion with a 3×3 filter followed by a dilation with a 5×5 filter with 10 iterations to reduce noise. Finally, the RGB foreground background segmentation mask is cropped to be of size $(359, 359, 1)$. Both matrices feed into a MSE loss function, which allows the backbone together with the pretext head to learn the segmentation task. For background subtraction, the OpenCV



(a) RGB to depth.



(b) Depth to foreground background segmentation.

Figure 4.22.: Self-supervised domain transfer learning with foreground background segmentation as pretext subtask.

MOG2[4] algorithm is used. Due to its definition, background subtraction methods only work in static camera setups. This is however not true for the videos in NYUDepthv2. Therefore, this dataset can not be used for training. For STC [90] and Princeton [79], half of the videos are captured in static camera setups. Therefore, these datasets are used for self-supervised learning rather than NYUDepthv2, despite being significantly smaller. STC provides information about which videos are captured in static setups. For Princeton these videos are detected by filtering images that have up to 40% foreground, which effectively removes moving camera scenes. Figure 4.23 shows some RGB and depth images together with the background segmentation. The images show that in static camera setups this technique works quite well. However, it is hard to find parameters that work well for most of the training images. By e.g. looking at the 4th image from the left, the segmentation has the contours of the person, but not the body. Also, some noise in the outputs can not always be suppressed (e.g. first image on the left).

Training Settings

Due to the restriction to static camera setups, STC and PTB (see Table 3.2) are used for training with 11540 videos per epoch. As only half of the videos have static cameras, this makes 68 videos in total. Both pretext trainings use Adam with a LR of 0.0003 and a multi-step LR schedule which multiplies the LR by 1/5 at epochs 150, 200, and 250. Batch size is 32 per GPU. Training is performed for 300 epochs for RGB to depth, as well as depth to foreground background segmentation task. The architecture implements a C3BR efficient fusion layer to combine the two backbone networks from both domains. Adam is used with a constant LR of 0.0003. Training is performed for 150 epochs. The remaining settings for the second step are as in Table 4.6. When initializing the backbone

[4] `https://docs.opencv.org/3.4/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html`

(a) RGB.

(b) Depth.

(c) Background segmentation.

Figure 4.23.: Visualization of foreground background segmentation through background subtraction on STC and PTB. (a) and (b) are images provided by the datasets. (c) is the processed output of the background subtractor.

with weights from self-supervised pretraining, finetuning the backbone starts at epoch 30, always with 1/10 of the LR of the rest of the architecture. When training from scratch the same learning rate is applied for the whole architecture. Finetuning for tracking uses the SceneNetRGBD dataset (train and bridge set). For pretext training and finetuning, VOT2019-RGBD serves as validation set.

Results

The training results for pretext training are presented in Figure 4.24. While the training loss consistently decreases for RGB to depth, the validation loss indicates overfitting to the training data after approximately 120 epochs. The model with the lowest validation loss (epoch 107) is used for pretraining. For depth to segmentation pretraining, the situation is similar, however overfitting starts even earlier at about 20 epochs. Again, the model with the lowest validation loss (epoch 9) is taken for pretraining. Compared to reconstruction learning, the validation losses seem to be more noisy. This is most probably because of the fact that here the training data is significantly smaller, which makes it not generalize well to unseen data and promotes early overfitting. In addition to the models with the lowest validation losses, the model snapshots at epoch 300 will also be taken as initialization for finetuning. Figure 4.25 shows the results for finetuning for tracking. Pretrained1 refers to the models with the lowest validation loss, while pretrained2 refers to the snapshots at epoch 300 (see circles in Figure 4.24). After epoch 30, when finetuning of the backbone starts, the training IoU gets better for the pretrained models compared to training from scratch. This is also the case for the bridge IoU, however not as much. The validation IoUs are again smoothed due to the reason mentioned in the reconstruction learning subsection. Comparing

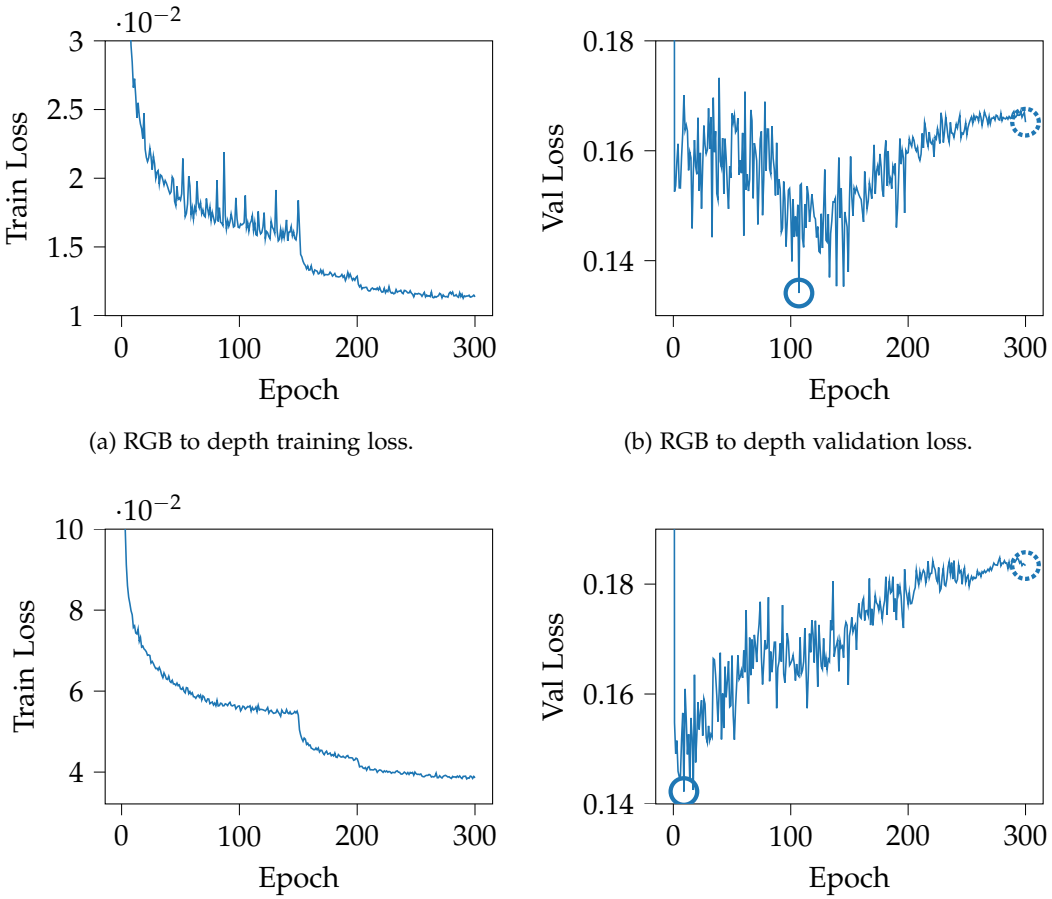
the smoothed maximum validation IoUs gives an improvement of 3.4% IoU which is an absolute improvement of 5.9% when performing domaintransfer learning with foreground background segmentation on unlabeled data. The performance gain is significantly higher for the pretreained2 model. Considering the validation losses in Figure 4.24, this means that having a low validation loss does not necessarily mean that the model is suited better for initializing the tracking architecture in the second step. This finding is in general true for self-supervised pretraining, where one does not care too much about how good a model performs on the pretext task. It is much more important that the model learns good filters / representations rather than having a good performance. The performance gain is less than for reconstruction learning. However, the margin of improvement is impressive, considering the fact that only a fraction of the training data can be used here. For reconstruction learning, pretext training was performed on NYU with 582 videos, while domaintransfer learning with foreground background segmentation only uses 68 videos in total (static sequences of STC and PTB).

Grayscale Prediction

The major drawback of domain transfer learning with foreground background segmentation is the restriction to videos with static camera setups due to background subtraction. This inhibits the use of the NYU Depth v2 datasets which is much bigger compared to PTB and STC. To cope with these cons, another task can be used to learn features from depth which does not restrict the choice of data. Specifically, the RGB to depth part stays the same, while the foreground background segmentation is replaced by grayscale prediction (see Figure 4.26 for the overall setup). Inputs to the architecture are a cropped $(359, 359, 1)$ depth and a $(359, 359, 3)$ RGB image. The RGB image is converted to grayscale. The decoded feature representation of the depth image and the grayscale image lead into a MSE loss. Figure 4.27 shows some examples.

Training Settings

As grayscale prediction does not restrict the choice of data, the NYU-Depth v2 data is used for training with 36300 videos per epoch. Both pretext trainings use Adam with a LR of 0.0003 and a multi-step LR schedule which multiplies the LR by $1/5$ at epochs 150, 200, 250, and 275. Batch size is 32 per GPU. Training is performed for 300 epochs for RGB to depth, as well as depth to grayscale prediction task. Similar to domaintransfer learning with foreground background segmentation, the architecture implements a C3BR efficient fusion layer to combine the two backbone networks of both domains. Adam is used with a constant LR of 0.0003. Training is performed for 150 epochs. The remaining settings for the second step are as in Table 4.6. When initializing the backbone with weights from self-supervised pretraining, finetuning the backbone starts at epoch 30, always with $1/10$ of the LR of the rest of the architecture. When training from scratch the same learning rate is applied for the whole architecture. Finetuning for tracking uses the SceneNetRGBD dataset (train and bridge set). For pretext training and finetuning, VOT2019-RGBD serves as validation set.



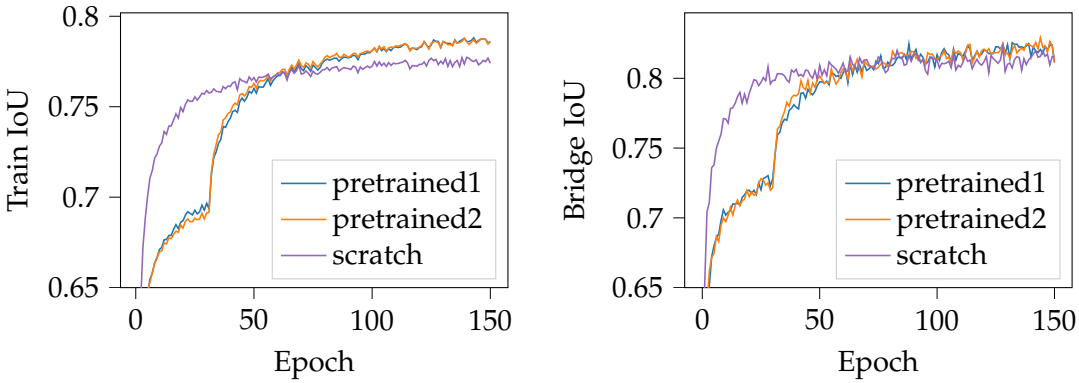
(a) RGB to depth training loss.

(b) RGB to depth validation loss.

(c) Depth to segmentation training loss.

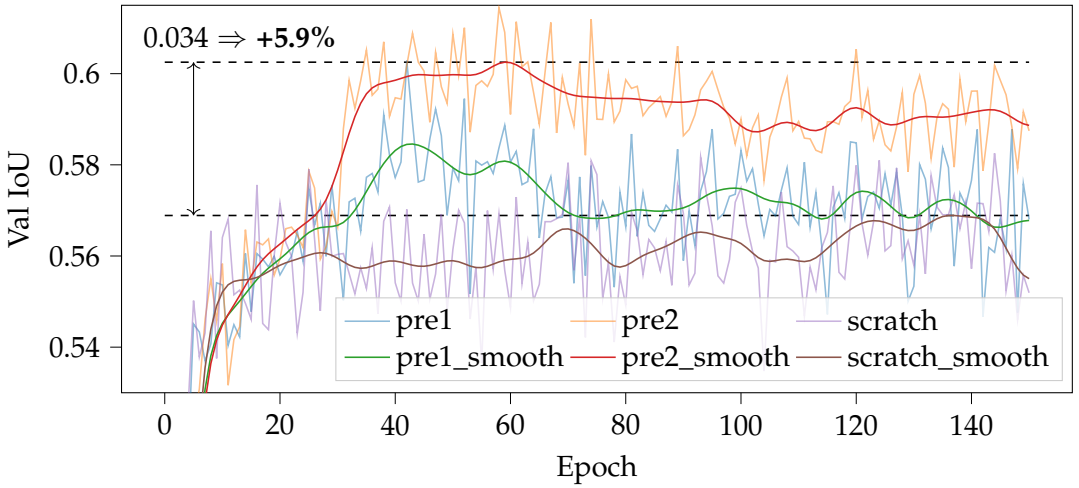
(d) Depth to segmentation validation loss.

Figure 4.24.: Results for domaintransfer pretext training with foreground background segmentation pretext task. (a), (b) show RGB to depth training, (c), (d) depth to foreground background segmentation. The solid circle marks the snapshots with the lowest validation loss, while the dotted circle marks the validation losses at the end of training (epoch 300). Both candidates are evaluated for finetuning for tracking.



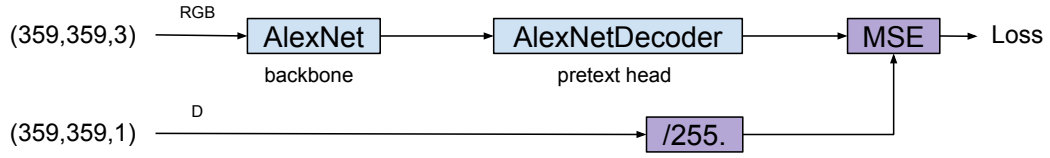
(a) Tracking training IoU.

(b) Tracking bridge IoU.

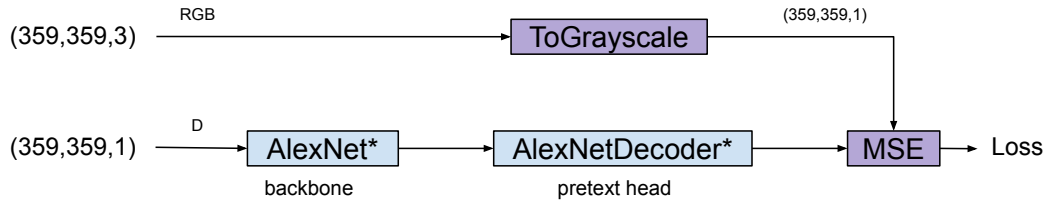


(c) Tracking validation IoU.

Figure 4.25.: Results for domaintransfer finetuning with foreground background segmentation pretext with the respective model snapshots from Figure 4.24. The validation IoUs are smoothed with a 1D gaussian filter with $\sigma = 3$.

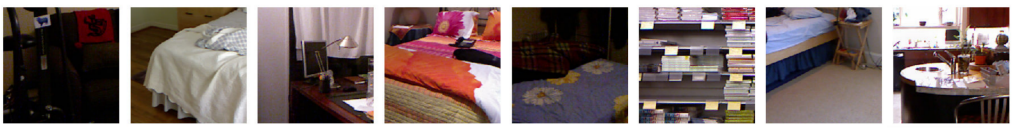


(a) RGB to depth.



(b) Depth to grayscale.

Figure 4.26.: Self-supervised domain transfer learning with grayscale prediction as pretext subtask.



(a) RGB.

(b) Depth.

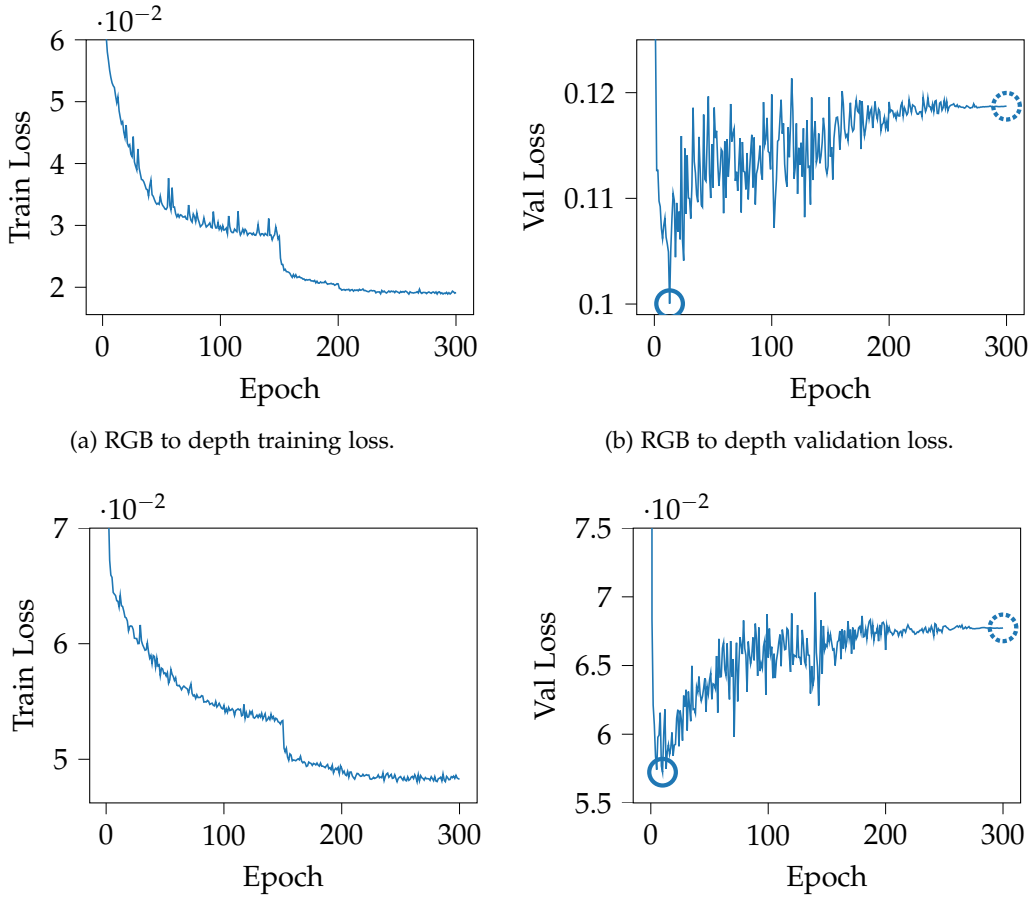
(c) Grayscale version of RGB image.

Figure 4.27.: Visualization of grayscale prediction images on the NYU Depth v2 dataset. (a) and (b) are images provided by the dataset. (c) is the grayscale version of the RGB image that the network shall learn to predict given the depth image.

Results

Figure 4.28 shows the results for pretext training. For RGB to depth learning the training loss consistently decreases, while the validation loss indicates overfitting after approximately 15 epochs. The depth to grayscale prediction subtask follows the same trend, with overfitting starting at about epoch 15. The noise in the validation losses might potentially result from a data mismatch problem between the training data (NYU Depth v2) and the validation set (VOT2019-RGBD). The model snapshots with the lowest validation losses (RGB to depth: epoch 13, depth to grayscale: epoch 10) as well as the snapshots at epoch 300 will be taken as initialization for finetuning. The results for tracking are visualized in Figure 4.29. Pretrained1 refers to the models with the lowest validation loss, while pretrained2 refers to the snapshots at epoch 300 (see circles in Figure 4.28). After epoch 30, when finetuning of the backbone starts, the training and bridge IoU clearly starts to improve. The pretrained1 model gets better than the model that trains from scratch in terms of training and bridge IoU, while the pretrained2 model achieves similar performance. The validation IoUs are smoothed with a 1D gaussian filter to cope with outliers and allow better comparison. When comparing the validation IoUs, pretrained1 and pretrained2 clearly outperform training from scratch without self-supervised pretraining. The maximum improvement is obtained by pretrained2 with an improvement over training from scratch of 0.022, which corresponds an absolute improvement of 3.9%. Again, the model profits from self-supervised pretraining. The margin is slightly smaller compared to foreground background segmentation despite training on much more data, indicating that the latter is a better suited pretext task for self-supervised pretraining with tracking as final application.

The investigations in this section revealed the potential of self-supervised pretraining in an RGBD tracking setup, where large-scale datasets are not yet widely available. All experiments showed superior performance compared to training from scratch by significant margins. Reconstruction learning leads to the best improvement compared to the two domaintranfer methods. Most probably, this is due to the data distribution problem between the training (SceneNetRGBD) and validation data (VOT2019-RGBD) that has already been addressed in the supervised section (see section 4.7). For the domaintransfer approaches, the final tracking architecture needs a fusion layer to combine features from both domains. This layer needs to be trained from scratch during finetuning for tracking. Therefore, the overall architecture has more parameters and can fit the training data better. However, as the training and validation data distribution is somewhat different, the domaintransfer models overfit more to the SceneNetRBGD data distribution and thus have inferior performance on VOT2019-RGBD. For reconstruction learning, no domaintransfer layer is needed as the model directly processes 4D input data. This might explain the improvement of reconstruction learning over domaintransfer learning. Recalling that for domaintransfer learning with foreground background segmentation the training data is significantly less compared to the other self-supervised experiments, it is remarkable that the performance is close to the performance of reconstruction learning, and even better compared to domaintransfer learning with grayscale prediction, indicating that this training method might be the most elegant one considering the amount of data needed.



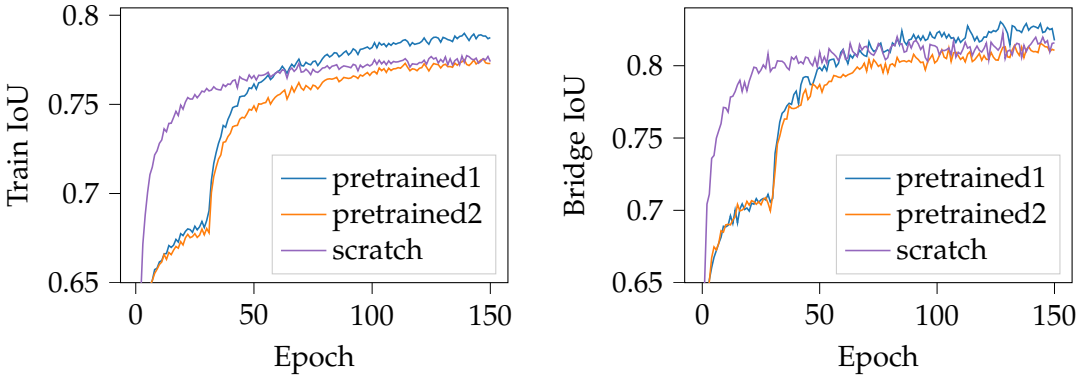
(a) RGB to depth training loss.

(b) RGB to depth validation loss.

(c) Depth to grayscale training loss.

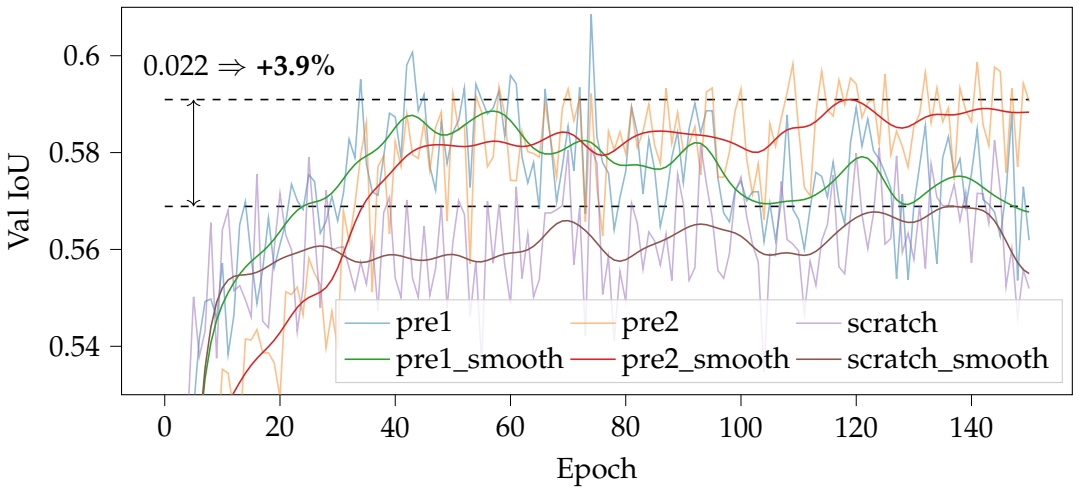
(d) Depth to grayscale validation loss.

Figure 4.28.: Results for domaintransfer pretext training with grayscale prediction pretext task. (a), (b) show RGB to depth training, (c), (d) depth to grayscale prediction. The solid circle marks the snapshots with the lowest validation loss, while the dotted circle marks the validation losses at the end of training (epoch 300). Both candidates are evaluated for finetuning for tracking.



(a) Tracking training IoU.

(b) Tracking bridge IoU.



(c) Tracking validation IoU.

Figure 4.29.: Results for domaintransfer finetuning with grayscale prediction pretext task with the respective model snapshots from Figure 4.28. The validation IoUs are smoothed with a 1D gaussian filter with $\sigma = 3$.

5. Discussion and Future Work

The goal of this thesis is to investigate the use of depth information in deep visual object trackers. Chapter 2 gives a thorough analysis over the progress of the tracking community in the topic of generic visual object tracking. It explained the separation of approaches into template-matching methods and tracking by detection. In addition, the progress in processing depth information in other computer vision applications has been analysed, before finally enlarging on RGBD tracking. Chapter 3 presented the evaluation metrics and outlined the challenges in RGBD tracking. Out of the approaches presented in the related work section, DaSiamRPN has been taken as RGB baseline tracker for further investigations with depth information. The algorithm has been presented, particularly mentioning the training process, as the main concern of the thesis is to incorporate the depth information in the training process of a deep visual object tracker. Chapter 4 explains and analyzes all conducted experiments. For all experiments the final goal has been to achieve good performance on the VOT2019-RGBD tracking dataset, as it currently is the most relevant and as well the biggest benchmark for generic object tracking with depth information. As a proof of concept (see section 4.2), half of the data has been used for training, while the other half served as validation data. It became obvious that adding depth information to the architecture leads to approximately 5% absolute improvement on the validation set over a variety of hyperparameters. As using pretrained networks is a common tool in machine learning that helps to develop models especially in scenarios with little data, the next experiment (see section 4.3) has tested if and to what extent pretrained networks can be used for RGBD tracking. The use of pretrained networks with depth information is completely different to the RGB case, as no RGBD pretrained networks are available. Therefore, the experiment tested how RGB pretrained weights can be used for a tracker with depth. As RGB and depth have to be used as separate channels in order to exploit pretrained networks (no 4D input pretrained networks available), a C1BR fusion scheme is introduced to combine features from both domains. The results reveal that RGB pretraining helps to significantly increase the performance when training and validating on VOT2019-RGBD. More specifically, replication of the depth channel to 3D lead to the best performance. The absolute increase is even more compared to the RGB only case, indicating that for RGBD inputs, pretrained networks seem to have at least the same if not more benefit than for RGB. Another experiment (see section 4.4) investigates the influence of deeper and shallower backbone architectures for feature extraction with depth-only input data. It also used HOG as a popular handcrafted feature extractor in the history of generic object tracking. The major outcome is that the deeper versions of the AlexNet backbones led to better performance. However, taking the unmodified version is the best option for the rest of the thesis, as only for these pretrained weights are available. Also, the deep AlexNet backbone outperformed the HOG extractor by a

large margin, justifying the research question on how to train deep visual object trackers with depth information. As the pretrained weights have such a big influence on the final performance given VOT as training data, section 4.5 compares ImageNet pretraining vs PySOT model zoo pretraining, which is provided by the DaSiamRPN authors, for depth-only inputs. The model zoo model has been initialized with ImageNet pretrained weights and been finetuned on large-scale RGB tracking datasets. The results show that transfer learning improves the performance in both cases, with the model zoo pretraining clearly outperforming ImageNet pretraining. Transfer learning is able to improve the performance with depth-only inputs in both cases, however, by over 35% less compared to the RGB case. Taking the knowledge about what backbone to use for depth information, as well as further investigations about RGB pretraining methods applied to depth information, section 4.6 goes one step further and implements and evaluates different fusion layers and their performance. Considering the data together with the validation IoU as well as the number of parameters, C3BR efficient as well as $[C3BR]^2$ efficient turned out to be best suited for this application. This experiment concludes the investigations on how to set up an RGBD generic tracking architecture that utilizes transfer learning and achieve the best performance. Finally, section 4.7 exploits all previous findings to come up with one architecture that is evaluated on the official VOT2019-RGBD benchmark. As the participation guidelines prohibit the use of the VOT datasets for training, SceneNetRGBD serves as training and bridge data, while validation data is the VOT. The evaluation is performed in terms of IoU as well as tracking F1 score. Every architecture is benchmarked against the model zoo tracker with the same training settings. While validation IoU and tracking score are better without depth data, the opposite is true for training and bridge IoU for all settings. This reveals that the RGBD tracker outperforms its RGB counterpart on data that is from the same distribution as the training data. However, there is a data mismatch problem between SceneNetRGBD and VOT2019-RGBD, resulting in worse validation IoU and F1 score on VOT. Keeping the data mismatch problem in mind, section 4.8 presents three self-supervised training methods to cope with the shortage of large-scale RGBD tracking training data. The central idea is to train on unlabeled RGBD data. All implemented approaches outperform their counterparts which have been trained from scratch, with the best approach (reconstruction learning) leading to a 6.8% improvement. Domaintransfer learning with foreground background segmentation pretext task achieved an improvement of 5.9% while training on significantly less data. Another domaintransfer approach with depth prediction pretext leads to an improvement of 3.9%

This work has taken AlexNet-like backbone architectures into account. In the future, to go along with the general trend in deep learning, deeper architectures like ResNet [29], Inception [81], etc could lead to even more meaningful features, resulting in better tracking performance. However, this requires bigger RGBD datasets, as with the current situation it is not feasible to train these architectures given the available datasets. The supervised experiments in this thesis proved that an RGBD tracker will outperform its RGB counterpart assuming that the data distribution of training and validation data are similar. With more large-scale datasets becoming available, this requirement will be more and more fulfilled. Once this is the case, the approaches and insights that have

been found might be able to eventually improve any tracker over its RGB counterpart. Therefore, despite having found many useful insights that might help the tracking community to come up with suitable RGBD tracking algorithms, re-evalution of the conducted experiments with bigger datasets might already lead to SOTA results. In addition, the findings may enable other researchers to include depth information in their trackers and push overall progress in RGBD tracking. Future RGBD datasets should tackle data dissimilarity, ideally utilizing many different sensors to capture depth data, as this will help the trackers to be invariant to the capturing hardware. So far, the VOT2019-RGBD dataset has been captured using three different sensors, however, many datasets just use one sensor or are rendered (SceneNetRGBD). This adds another degree of dissimilarity, leading to situations like data mismatch problems, as encountered in the experiments.

6. Conclusion

The availability of the VOT2019-RGBD visual object tracking challenge motivated this work. Following the trend in generic object tracking, it has been investigated how depth information can be effectively utilized during training of a deep visual object tracker. The findings in the supervised training setups demonstrated how trackers could profit from the additional information. However, in the current situation, the sparsity of RGBD datasets poses data distribution problems between training and validation data. As soon as the focus of the tracking community goes more into RGBD tracking, big datasets will become available and tackle these problems. Self-supervised pretraining has shown remarkable results and might help to solve this problem in the near future by exploiting unlabeled RGBD data. This is a big advantage, as unlabeled data is usually cheap to acquire. As soon as large-scale datasets are available, the techniques derived in this thesis might enable the adaption of SOTA RGB trackers to the RGBD domain by effectively combining information that is obtained by both, RGB and depth.

A. General Addenda

A.1. Training on SceneNet-RGBD for tracking

RGBD datasets are still rare. As mentioned in subsection 4.3.2, SceneNet-RGBD [66] is a large rendered dataset of indoor scenes and therefore has pixel-perfect annotations. Specifically, each video consists of 300 separately stored rgb and depth images. In addition, an instance map is provided with every frame, which gives a segmentation of the image. The dataset as is does not provide bounding boxes for objects, However, given the instance map it is possible to compute the bounding boxes for every object in the image. The instance mapping is consistent in each sequence such that e.g. ID 1 would correspond to the same object all the time. This allows an identification of the same object in nearby images which can then be used to train a tracker. The dataset also provides a protobuf file which specifies all objects in a sequence together with scales, rotation matrices, camera capturing details as well as the instance type, which can be random objects, background, layout objects, as well as light objects. Random objects are sampled from ShapeNet [10], which is a large-scale dataset of 3D shapes. The dataset also provides a mapping to NYU [69, 76] classes, which are the following: Unknown, bed, books, ceiling, chair, floor, furniture, objects, picture, sofa, table, TV, wall, window. Considering the instance maps and utilizing the NYU classes, "objects" falling into the floor, ceiling, or wall category are removed. The processed dataset has bounding box annotations for all objects in every frame. These can be used to train an RGBD tracking architecture.

A.2. Processing Raw NYU Depth v2 for self-supervised pretraining

The labeled NYU Depth v2 [69] is rather small (see Table 3.2). However, the authors also provide a bigger raw unlabeled dataset with $407,024$ images from 582 videos. It provides the color and depth images as well as accelerometer dumps from the kinect. The timestamps of all files are included in their filenames. The files are not yet synchronized. The authors of NYU provide a toolbox that provides some functions for processing the data. Each RGB image is synchronized with the nearest depth frame in terms of capturing time. The RGB image is cropped such that only the part of the image is used where the depth signal is projected. The depth images often contain missing values, which can be filled by postprocessing. Specifically, the toolkit provides a function which uses a grayscale version of the RGB image as weighting factor for smoothing the depth image. For more information, refer to the toolkit[1]. The postprocessed and synchronized

[1] `https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html`

frames can then be saved and used for self-supervised pretraining, as in section 4.8. The raw dataset has a size of 428GB. Processing has taken about 4 days on a machine with an Intel i9 CPU.

A.3. Evaluating top performing architectures from VOT2019-RGBD

After obtaining the results from sections section 4.2 to section 4.6, one approach was to enhance the implementation of a top performing architecture on the VOT2019RGBD challenge by adding depth information. The three architectures SiamDW_D, ATCAIS, and LTDSEd achieved almost the same performance on the challenge. To recap, all three implement deep backbones as feature extractors. Also, in addition to an offline trained feature extractor, they implement a discriminative localization strategy with a small network which is trained online. According to [48, 61], the depth information is used for target loss detection. None of these seems to incorporate depth information in the training phase. The plan has been to take either one of these architectures and incorporate depth information in the training process. Specifically, two methods should have been tried. First, depth information can be included in the backbone feature extractor. This is promising, however, it might run into similar data similarity problems as the final supervised experiment in section 4.7. Even more promising would be the integration of the depth information into the online trained discriminative localization strategy. This would not have the data shortage drawback as the task of this unit is to do foreground / background differentiation. Also, as the online adapted network is small, overfitting is not a concern. Assuming that data distribution is not a problem, incorporating depth into both modules should fully leverage depth information and lead to even better results. All of the mentioned strategies seem very appealing and might be capable to improve the architectures to become even better.

Unfortunately, these experiments could not be performed due to driver and hardware discompatibilities. The code of the top three trackers is published on GitHub. However, all of them have used CUDA9.0, Cudnn7.0.x drivers and PyTorch0.3.x together with older Nvidia GPUs (e.g. Nvidia GeForce GTX Titan). The setup of the available machines in the scope of this thesis use CUDA10.2, Cudnn7.6.5 drivers and PyTorch1.3.1 with Nvidia GeForce RTX2080 GPUs. As the RTX2080 is not compatible with older drivers it is not trivial to downgrade the system in order to use the existing code for the published trackers. Furthermore, many python modules are no longer compatible with newer PyTorch libraries and also many PyTorch functionalities have changed from $0.3.x$ to $1.3.x$, which does not allow an upgrade of the trackers' code to newer GPU drivers and PyTorch versions. Potentially, this strategy can be applied with the publication of the VOTRGBD2020 trackers.

List of Figures

List of Tables

Acronyms

3D-T 3D Part-Based Sparse Tracker with Automatic Synchronization and Registration. 30, 37

ATCAIS Accurate Tracking by Category-Agnostic Instance Segmentation for RGBD Images. 33, 34, 36, 38

ATOM Accurate Tracking by Overlap Maximization. 10, 12, 14–16, 33, 34, 36, 110

AUC area under curve. 23, 24, 111

CDTB Color and Depth Visual Object Tracking Dataset and Benchmark. 32–34, 36

CLGS Complementary Local-Global Search for Robust Long-term Tracking. 13–16, 37

CN Color Name. 19, 21, 30–32, 37, 113

CNN Convolutional Neural Network. 1, 3–5, 10, 16, 20, 21, 23–29, 32, 33, 37, 41, 42, 111, 112

CSR-DCF Discriminative Correlation Filter with Channel and Spatial Reliability. 21, 22, 33–35, 37, 111, 113

CSTEM Channels weighted and spatial-related Tracker with Effective response-map Measurement. 22, 37

DaSiamRPN Distractor-aware Siamese Region Proposal Network. 6, 7, 36, 41, 43–46, 53–56, 58, 61, 65, 68–70, 75, 77, 78, 80, 82, 114–116, 119

DCF Discriminative Correlation Filter. 4, 17, 19–23, 32, 34, 36–38, 73, 74, 116

DFT Discrete Fourier Transform. 19

DiMP Discriminative Model Prediction for Tracking. 13, 15, 37, 110

DS-KCF Depth Scaling Kernelized Correlation Filter. 31, 38, 112

DSST Discriminative Scale Space Tracker. 19, 20, 111

DualCF Dual Correlation Filter. 17, 19, 20

EAO Expected Average Overlap. 13

FCOS Fully Convolutional One-Stage Object Detector. 15

fDSST fast Discriminative Scale Space Tracker. 19, 20

GPU Graphics Processing Unit. 55, 56, 64–66, 83, 85, 88, 91, 93, 108

HOG Histogram of Oriented Gradients. 19, 21, 30–32, 37, 73, 74, 101, 113, 116

HTC Hybrid Task Cascade. 34, 36

IoU intersection-over-union. 12, 13, 23, 44, 45, 55, 57–59, 61, 63–67, 69, 71, 73–75, 78, 80, 82–86, 88–90, 92, 93, 95, 97, 99, 102, 110, 114, 116, 117, 120

KCF Kernelized Correlation Filter. 17, 19, 31, 32, 37, 112, 113

LR Learning Rate. 55, 56, 58, 61, 65, 66, 78, 83, 88, 91–93

LSTM Long Short-Term Memory. 23, 24, 111

LT-DSE long-term tracking by diving videos into successive short episodes. 13–16, 37

LTDSEd long-term tracking using depth information by diving videos into successive short episodes. 34, 36, 38

LTM Long-term module. 10, 11, 109

MBMD MobileNet based tracking by detection algorithm. 16

MF Multiple Features Tracker. 19

MOSSE Minimum Output Sum of Squared Error. 16, 17, 37

MSE Mean Squared Error. 88, 90, 93

OAPF Occlusion-Aware Particle Filter. 29, 30, 34, 37

OTR Object Tracking by Reconstruction with View-Specific Discriminative Correlation Filters. 32–34, 36, 38, 113

PCA Principal Component Analysis. 20

Re^3 Real-time Recurrent Regression-based tracker. 23, 24, 37, 111

ReLU rectified linear unit. 69, 70, 119

RNN Recurrent Neural Network. 23, 37

ROI Region of Interest. 31

RPN Region Proposal Network. 6–8, 15, 16, 36, 41–43, 56, 65, 67, 69–72, 75, 76, 78, 79, 82–85, 88, 109, 114–116, 120

RPT Reliable Patch Trackers. 22, 37

RT-MDNet Real-time MDNet. 15, 16, 36

SA Scale Adaptive Tracker. 19

SAMF Scale Adaptive with Multiple Features Tracker. 19

SGD Stochastic Gradient Descent. 44, 55, 56, 65, 66, 83, 88

SiamFC Siamese Fully-Convolutional. 5, 6, 9, 36, 41, 109

SiamFCOT Siamese Fully Convolutional Object Tracking. 13–15

SiamRPN Siamese Region Proposal Network. 6–10, 36, 41, 109

SINT Siamese Instance Search Tracker. 4, 5, 36, 37, 109

SOTA state of the art. 35, 103, 105

STC Spatio-Temporal Consistency. 39, 49

STM Short-term module. 10, 11, 109

Struck Structured Output Tracking with Kernels. 23, 37

SVM support vector machine. 4, 15, 23, 37

THOR Tracking Holistic Object Representations. 9–11, 36, 109

VOT Visual Object Tracking Challenge. 3, 32, 59

WD Weight Decay. 55, 56, 66, 76, 83, 88

YouTube-VOS Video Object Segmentation dataset. 8

Bibliography

[1] P. Ammirato, P. Poirson, E. Park, J. Kosecka, and A. C. Berg. "A Dataset for Developing and Benchmarking Active Vision". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[2] A. Ayub and A. Wagner. *Centroid-Based Scene Classification (CBSC): Using Deep Features and Clustering for RGB-D Indoor Scene Classification*. 2019.

[3] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr. *Fully-Convolutional Siamese Networks for Object Tracking*. 2016.

[4] G. Bhat, M. Danelljan, L. van Gool, and R. Timofte. *Learning Discriminative Model Prediction for Tracking*. 2019.

[5] A. Bibi, T. Zhang, and B. Ghanem. "3D Part-Based Sparse Tracker with Automatic Synchronization and Registration". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 1439–1448.

[6] L. Bo, X. Ren, and D. Fox. "Unsupervised feature learning for RGB-D based object recognition". In: *International Symposium on Experimental Robotics (ISER)* (2012), pp. 387–402.

[7] D. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. "Visual object tracking using adaptive correlation filters". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010*. Piscataway, NJ: IEEE, 2010, pp. 2544–2550.

[8] Z. Cai and N. Vasconcelos. *Cascade R-CNN: Delving into High Quality Object Detection*. 2017.

[9] M. Camplani, S. Hannuna, M. Mirmehdi, D. Damen, A. Paiement, L. Tao, and T. Burghardt. "Real-time RGB-D Tracking with Depth Scaling Kernelised Correlation Filters and Occlusion Handling". In: *Procedings of the British Machine Vision Conference 2015*. Ed. by X. Xie, M. W. Jones, and G. K. L. Tam. British Machine Vision Association, 2015, pp. 145.1–145.11.

[10] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012 Titel anhand dieser ArXiv-ID in Citavi-Projekt übernehmen[cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[11] B. X. Chen and J. K. Tsotsos. "Fast Visual Object Tracking with Rotated Bounding Boxes". In: *CoRR* abs/1907.03892 (2019).

[12] K. Chen, J. Pang, J. Wang, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Shi, W. Ouyang, C. C. Loy, and D. Lin. *Hybrid Task Cascade for Instance Segmentation*.

[13] Z. Chen, Z. Hong, and D. Tao. *An Experimental Survey on Correlation Filter-based Tracking*. 2015.

[14] S. Chopra, R. Hadsell, and Y. LeCun. "Learning a Similarity Metric Discriminatively, with Application to Face Verification". In: *CVPR 2005*. Ed. by C. Schmid, S. Soatto, and C. Tomasi. CVPR '05. Los Alamitos, Calif.: IEEE Computer Society, 2005, pp. 539–546.

[15] C. Couprie, C. Farabet, L. Najman, and Y. LeCun. *Indoor Semantic Segmentation using depth information*. 2013.

[16] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection". In: *CVPR*. Ed. by C. Schmid. Los Alamitos, Calif. [u.a.]: IEEE Computer Society, 2005, pp. 886–893.

[17] M. Danelljan, F. S. Khan, M. Felsberg, and J. v. d. Weijer. "Adaptive Color Attributes for Real-Time Visual Tracking". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1090–1097.

[18] M. Danelljan, G. Bhat, F. S. Khan, and M. Felsberg. *ATOM: Accurate Tracking by Overlap Maximization*.

[19] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg. *Discriminative Scale Space Tracking*. 2016.

[20] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard. *Multimodal Deep Learning for Robust RGB-D Object Recognition*. 2015.

[21] H. Fan, L. Lin, F. Yang, P. Chu, G. Deng, S. Yu, H. Bai, Y. Xu, C. Liao, and H. Ling. *LaSOT: A High-quality Benchmark for Large-scale Single Object Tracking*.

[22] D. Gordon, A. Farhadi, and D. Fox. "Re3 : Real-Time Recurrent Regression Networks for Visual Tracking of Generic Objects". In: (2018).

[23] S. Gu, Y. Lu, L. Zhang, and J. Zhang. "RGB-D Tracking Based on Kernelized Correlation Filter with Deep Features". In: *Neural information processing*. Ed. by D. Liu. Vol. 10636. LNCS sublibrary: SL1 - Theoretical computer science and general issues. Cham, Switzerland: Springer, 2017, pp. 105–113.

[24] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. *Learning Rich Features from RGB-D Images for Object Detection and Segmentation*. 2014.

[25] S. L. Hannuna, M. Camplani, J. Hall, M. Mirmehdi, D. Damen, T. Burghardt, A. Paiement, and L. Tao. "DS-KCF: a real-time tracker for RGB-D data". In: *Journal of Real-Time Image Processing* (2016), pp. 1–20.

[26] S. Hare, A. Saffari, and P. H. S. Torr. "Struck: Structured output tracking with kernels". In: *ICCV 2011*. [Piscataway, N.J.]: [IEEE], 2011, pp. 263–270.

[27] C. Hazirbas, L. Ma, C. Domokos, and D. Cremers. "FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-Based CNN Architecture". In: *Computer Vision – ACCV 2016*. Ed. by S.-H. Lai, V. Lepetit, K. Nishino, and Y. Sato. Vol. 10111. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 213–228.

[28] A. He, C. Luo, X. Tian, and W. Zeng. *A Twofold Siamese Network for Real-Time Object Tracking*. 2018.

[29] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. 2015.

[30] K. He, X. Zhang, S. Ren, and J. Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015.

[31] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. "Exploiting the Circulant Structure of Tracking-by-Detection with Kernels". In: *Computer vision– ECCV 2012*. Ed. by A. W. Fitzgibbon. Vol. 7575. LNCS sublibrary. SL 6, Image processing, computer vision, pattern recognition, and graphics. Berlin and New York: Springer, 2012, pp. 702–715.

[32] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. "High-Speed Tracking with Kernelized Correlation Filters". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.3 (2015), pp. 583–596.

[33] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[34] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*.

[35] L. Huang, X. Zhao, and K. Huang. "GOT-10k: A Large High-Diversity Benchmark for Generic Object Tracking in the Wild". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018).

[36] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. "Comparing images using the Hausdorff distance". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.9 (1993), pp. 850–863.

[37] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167.

[38] A. Janoch, S. Karayev, Yangqing Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell. "A category-level 3-D object dataset: Putting the Kinect to work". In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011, pp. 1168–1174.

[39] M.-x. Jiang, C. Deng, M.-m. Zhang, J.-s. Shan, and H. Zhang. "Multimodal Deep Feature Fusion (MMDFF) for RGB-D Tracking". In: *Complexity* 2018.1 (2018), pp. 1–8.

[40] A. E. Johnson and M. Hebert. "Using spin images for efficient object recognition in cluttered 3D scenes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.5 (1999), pp. 433–449.

[41] I. Jung, J. Son, M. Baek, and B. Han. *Real-Time MDNet*.

[42] U. Kart, J.-K. Kämäräinen, and J. Matas. "How to Make an RGBD Tracker?" In: *ECCVW*. 2018.

[43] U. Kart, A. Lukezic, M. Kristan, J.-K. Kamarainen, and J. Matas. *Object Tracking by Reconstruction with View-Specific Discriminative Correlation Filters*.

[44] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernandez, T. Vojir, G. Häger, G. Nebehay, and R. Pflugfelder. "The Visual Object Tracking VOT2015 challenge results". In: *Workshop on the VOT2015 Visual Object Tracking Challenge*. 2015.

[45] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pfugfelder, Luka Čehovin Zajc, Tomas Vojir, Goutam Bhat, Alan Lukezic, Abdelrahman Eldesokey, Gustavo Fernandez, and et al. *The sixth Visual Object Tracking VOT2018 challenge results*. 2018.

[46] M. Kristan, A. Leonardis, J. Matas, et al. "The Visual Object Tracking VOT2017 Challenge Results". In: *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*. IEEE, 2017, pp. 1949–1972.

[47] M. Kristan, A. Leonardis, J. Matas, et al. "The Visual Object Tracking VOT2016 Challenge Results". In: *Computer Vision - ECCV 2016 Workshops*. Ed. by G. Hua and H. Jégou. Vol. 9914. Lecture Notes in Computer Science. Cham and s.l.: Springer International Publishing, 2016, pp. 777–823.

[48] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, R. Pflugfelder, J.-K. Kamarainen, L. Č. Zajc, O. Drbohlav, A. Lukezic, A. Berg, A. Eldesokey, J. Kapyla, and G. Fernandez. *The Seventh Visual Object Tracking VOT2019 Challenge Results*. 2019.

[49] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Čehovin. "A Novel Performance Evaluation Methodology for Single-Target Trackers". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2016).

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[51] K. Lai, L. Bo, and D. Fox. "Unsupervised feature learning for 3D scene labeling". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 3050–3057.

[52] K. Lai, L. Bo, X. Ren, and D. Fox. "A large-scale hierarchical multi-view RGB-D object dataset". In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 1817–1824.

[53] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, and J. Yan. *SiamRPN++: Evolution of Siamese Visual Tracking with Very Deep Networks*. 2018.

[54] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu. "High Performance Visual Tracking with Siamese Region Proposal Network". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Piscataway, NJ: IEEE, 2018, pp. 8971–8980.

[55] Y. Li and J. Zhu. "A Scale Adaptive Kernel Correlation Filter Tracker with Feature Integration". In: *Computer Vision - ECCV 2014 Workshops*. Ed. by L. Agapito, M. M. Bronstein, and C. Rother. Vol. 8926. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 254–265.

[56] Y. Li and J. Zhu. "Reliable Patch Trackers: Robust visual tracking by exploiting reliable patches". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 353–361.

[57] Y. Li and X. Zhang. *SiamVGG: Visual Tracking using Deeper Siamese Networks*. 2019.

[58] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. *Microsoft COCO: Common Objects in Context*. 2014.

[59] T. Liu, G. Wang, and Q. Yang. "Real-time part-based visual tracking via adaptive correlation filters". In: *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. [Piscataway, New Jersey]: [EEE], 2015, pp. 4902–4912.

[60] J. Long, E. Shelhamer, and T. Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2014.

[61] A. Lukežič, U. Kart, J. Käpylä, A. Durmush, J.-K. Kämäräinen, J. Matas, and M. Kristan. *CDTB: A Color and Depth Visual Object Tracking Dataset and Benchmark*.

[62] A. Lukežič, T. Vojíř, L. Čehovin, J. Matas, and M. Kristan. "Discriminative Correlation Filter with Channel and Spatial Reliability". In: *International Journal of Computer Vision* 126.7 (2018), pp. 671–688.

[63] A. Lukežič, L. Č. Zajc, T. Vojíř, J. Matas, and M. Kristan. *Now you see me: evaluating performance in long-term visual tracking*.

[64] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang. "Hierarchical Convolutional Features for Visual Tracking". In: *2015 IEEE International Conference on Computer Vision*. Ed. by I. I. C. o. C. Vision. Piscataway, NJ: IEEE, 2015, pp. 3074–3082.

[65] J. Mason, B. Marthi, and R. Parr. "Object disappearance for object discovery". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 2836–2843.

[66] J. McCormac, A. Handa, S. Leutenegger, and A. J.Davison. "SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation?" In: (2017).

[67] K. Meshgi, S.-i. Maeda, S. Oba, and S. Ishii. "Fusion of multiple cues from color and depth domains using occlusion aware bayesian tracker". In: *IEICE Tech. Rep. Neurocomp.* 113.500 (2014), pp. 127–132.

[68] K. Meshgi, S.-i. Maeda, S. Oba, H. Skibbe, Y.-z. Li, and S. Ishii. "An occlusion-aware particle filter tracker to handle complex and persistent occlusions". In: *Computer Vision and Image Understanding* 150 (2016), pp. 81–94.

[69] P. K. Nathan Silberman Derek Hoiem and R. Fergus. "Indoor Segmentation and Support Inference from RGBD Images". In: *ECCV*. 2012.

[70] T. Ojala, M. Pietikäinen, and T. Mäenpää. "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 7 (2002), pp. 971–987.

[71] L. Porzi, A. Penate-Sanchez, E. Ricci, and F. Moreno-Noguer. "Depth-aware convolutional neural networks for accurate 3D pose estimation in RGB-D images". In: *IROS Vancouver 2017*. [Piscataway, New Jersey]: IEEE, 2017, pp. 5777–5783.

[72] E. Real, J. Shlens, S. Mazzocchi, X. Pan, and V. Vanhoucke. *YouTube-BoundingBoxes: A Large High-Precision Human-Annotated Data Set for Object Detection in Video*. 2017.

[73] S. Ren, K. He, R. Girshick, and J. Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015.

[74] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. *ImageNet Large Scale Visual Recognition Challenge*. 2014.

[75] A. Sauer, E. Aljalbout, and S. Haddadin. *Tracking Holistic Object Representations*. 2019.

[76] N. Silberman and R. Fergus. "Indoor Scene Segmentation using a Structured Light Sensor". In: *Proceedings of the International Conference on Computer Vision - Workshop on 3D Representation and Recognition*. 2011.

[77] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014.

[78] S. Song, S. Lichtenberg, and J. Xiao. "SUN RGB-D: A RGB-D scene understanding benchmark suite". In: June 2015, pp. 567–576.

[79] S. Song and J. Xiao. "Tracking Revisited Using RGBD Camera: Unified Benchmark and Baselines". In: *2013 IEEE International Conference on Computer Vision*. New York: IEEE, 2014, pp. 233–240.

[80] X. Song, L. Herranz, and S. Jiang. "Depth CNNs for RGB-D scene recognition: learning from scratch better than transferring from RGB-CNNs". In: *AAAI Conference on Artificial Intelligence 2017* (2017).

[81] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions". In: *Computer Vision and Pattern Recognition (CVPR)*. 2015.

[82] R. Tao, E. Gavves, and A. W. M. Smeulders. "Siamese Instance Search for Tracking". In: *Proceedings 29th IEEE Conference on Computer Vision and Pattern Recognition : CVPR 2016*. 2016, pp. 1420–1429.

[83] Z. Tian, C. Shen, H. Chen, and T. He. *FCOS: Fully Convolutional One-Stage Object Detection*.

[84] J. van de Weijer, C. Schmid, J. Verbeek, and D. Larlus. "Learning color names for real-world applications". In: *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society* 18.7 (2009), pp. 1512–1523.

[85] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. S. Torr. *Fast Online Object Tracking and Segmentation: A Unifying Approach*. 2018.

[86] W. Wang and U. Neumann. *Depth-aware CNN for RGB-D Segmentation*. 2018.

[87] I. R. Ward, H. Laga, and M. Bennamoun. *RGB-D image-based Object Detection: from Traditional Methods to Deep Learning Techniques*. 2019.

[88] Y. Wu, J. Lim, and M.-H. Yang. "Object Tracking Benchmark". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9 (2015), pp. 1834–1848.

[89] Y. Wu, J. Lim, and M.-H. Yang. "Online object tracking: A benchmark". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 2411–2418.

[90] J. Xiao, R. Stolkin, Y. Gao, and A. Leonardis. "Robust Fusion of Color and Depth Data for RGB-D Target Tracking Using Adaptive Range-Invariant Depth Models and Spatio-Temporal Consistency Constraints." In: *IEEE Transactions on Cybernetics* 99 (2017), pp. 1–15.

[91] N. Xu, L. Yang, Y. Fan, J. Yang, D. Yue, Y. Liang, B. Price, S. Cohen, and T. Huang. *YouTube-VOS: Sequence-to-Sequence Video Object Segmentation*. 2018.

[92] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. "How transferable are features in deep neural networks?" In: *CoRR* abs/1411.1792 (2014). arXiv: 1411.1792.

[93] M. D. Zeiler and R. Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. 2014, pp. 818–833.

[94] Y. Zhang, D. Wang, L. Wang, J. Qi, and H. Lu. *Learning regression and verification networks for long-term visual tracking*.

[95] Z. Zhang, Y. Li, J. Ren, and J. Zhu. *Effective Occlusion Handling for Fast Correlation Filter-based Trackers*. 2018.

[96] Z. Zhang and H. Peng. *Deeper and Wider Siamese Networks for Real-Time Visual Tracking*.

[97] Z. Zhu, Q. Wang, B. Li, W. Wu, J. Yan, and W. Hu. "Distractor-Aware Siamese Networks for Visual Object Tracking". In: *Computer Vision - ECCV 2018*. Ed. by V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss. Vol. 11213. Image Processing, Computer Vision, Pattern Recognition, and Graphics. Cham: Springer International Publishing, 2018, pp. 103–119.

[98] S. Zia, B. Yuksel, D. Yuret, and Y. Yemez. "RGB-D Object Recognition Using Deep Convolutional Neural Networks". In: *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*. IEEE, 2017, pp. 887–894.