Technische Universität München

Detection of Bicycle Racks from Geodata Using Deep Learning

Karin Erbe

Master's Thesis

Study Course:	Geodesy and Geoinformation (Master)
Supervisors:	Univ.-Prof. Dr. rer. nat. Thomas H. Kolbe (Chair of Geoinformatics)
	Dr. Andreas Donaubauer (Chair of Geoinformatics)
In Cooperation with:	Prof. Dr. Melanie Brandmeier (Esri, FHWS)
	Jan-Andreas Liebscher (Landeshauptstadt München)
	Prof. Dr. Michael Schmitt (Hochschule München)
Submission:	12th October 2021

2021

Declaration

With this statement I declare that I have independently completed this Master's thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

Munich, 12th October 2021

Karin Erbe

Abstract

The City of Munich provides more than 34,000 public bicycle racks at approximately 350 sites. For inventorying them in a geoinformation system, automatic detection is desirable. This thesis considers the approach of using deep learning to detect the unroofed bike parking facilities in aerial imagery. For this purpose, a Mask R-CNN deep learning model with ResNet-50 backbone is trained using the ArcGIS API for Python. RGB, CIR (color-infrared), nDSM (normalized Digital Surface Model) and NDVI (Normalized Difference Vegetation Index) rasters from an image flight in winter 2019 are available, and ground truth data is derived from these. To evaluate, which of the band combinations gives the best result, the model is trained with RGB, CIR, RGB+nDSM, RGB+NDVI, and RGB+nDSM+NDVI, respectively. The trained models are then applied in ArcGIS Pro. It is shown that the best results are achieved when using the RGB imagery in combination with the nDSM raster. The detection reaches a precision of 83.2% and a recall of 77.3%. Limitations are in particular the shadows cast by buildings and trees. Since 74% of the bicycle racks in the training data are in the shadow, bicycle racks that are in the sun are more difficult to detect. In addition, dark shadows also prevent detection. Additionally, the analysis of the detected objects shows that parking facilities are recognized better when many bikes are parked there. The reason is the training data as it only contains a few empty bike parking facilities. Furthermore, this thesis investigates to what extent a classification of the bicycle rack types is possible. A detailed type differentiation is not attainable but a rough classification is achieved by the trained model.

Zusammenfassung

Die Landeshauptstadt München stellt an etwa 350 Orten mehr als 34.000 öffentliche Fahrradständer bereit. Für die Verarbeitung dieser in einem Geoinformationssystem ist eine automatische Erfassung wünschenswert. Diese Arbeit betrachtet den Ansatz, mittels Deep Learning Fahrradständer in München, die nicht überdacht sind, in Luftbildern zu detektieren. Dazu wird ein Mask R-CNN Deep-Learning-Modell mit ResNet-50 Backbone mittels der ArcGIS API für Python trainiert. Zur Verfügung stehen die Raster RGB, CIR (color-infrared), nDOM (normalisiertes digitales Oberflächenmodell) und NDVI (Normalized Difference Vegetation Index) aus einer Winterbefliegung im Jahr 2019. Aus diesen werden Ground Truth Daten abgeleitet. Um zu testen, welche der Bandkombinationen das beste Ergebnis liefert, wird das Modell jeweils mit RGB, CIR, RGB+nDOM, RGB+NDVI bzw. RGB+nDOM+NDVI trainiert. Die Anwendung der trainierten Modelle erfolgt anschließend in ArcGIS Pro. Dabei zeigt sich, dass bei der Verwendung der RGB Bilddaten in Verbindung mit dem nDOM Raster die besten Ergebnisse erzielt werden. Die Detektion erreicht eine Precision von 83.2% und einen Recall von 77.3%. Einschränkungen stellen hierbei insbesondere der Schattenwurf durch Gebäude und Bäume dar. Da 74% der Fahrradständer in den Trainingsdaten im Schatten liegen, werden zum einen Fahrradständer, die in der Sonne stehen, schlechter detektiert. Zum anderen verhindern sehr dunkle Schatten ebenfalls eine Detektion. Die Analyse der detektierten Objekte zeigt zudem, dass Fahrradständer besser erkannt werden, wenn dort viele Fahrräder abgestellt sind. Der Grund ist auch hier in den Trainingsdaten zu finden, da diese nur wenige leere Fahrradständer enthalten. Des Weiteren untersucht diese Arbeit, inwiefern eine Klassifizierung der Fahrradständertypen möglich ist. Es wird deutlich, dass eine detaillierte Typunterscheidung nicht möglich ist, jedoch eine grobe Einordnung durch das trainierte Modell erreicht wird.

Table of Contents

List of Abbreviations

ADFC	Allgemeiner Deutscher Fahrrad-Club e.V., German Cyclists Association
ANN	Artificial Neural Network
BN	Batch normalization
CIR	Color-Infrared
CNN	Convolutional Neural Network
conv	Convolutional layer
DGM	German abbreviation for: Digitales Geländemodell (see DTM)
DOM	German abbreviation for: Digitales Oberflächenmodell (see DSM)
DSM	Digital Surface Model
DSS	Data Storage Service
DTM	Digital Terrain Model
FC	Fully-connected
FCN	Fully Convolutional Network
FDR	False Detection Rate
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPN	Feature Pyramid Network
GIS	Geoinformation System
GPU	Graphics Processing Unit
IoU	Intersection over Union
LHM	Landeshauptstadt München, City of Munich
LRZ	Leibniz-Rechenzentrum, Leibniz Supercomputing Centre
MVG	Münchner Verkehrsgesellschaft, Munich Transport Corporation
MWN	Münchner Wissenschaftsnetz, Munich Scientific Network
nDOM	German abbreviation for: normalisiertes digitales Oberflächenmodell (see nDSM)
nDSM	Normalized Digital Surface Model
NDVI	Normalized Difference Vegetation Index
NIR	Near Infrared
px	Pixel
R-CNN	Region based CNN
ReLU	Rectified Linear Unit
RoI	Region of Interest
RPN	Region Proposal Network
SLURM	Simple Linux Utility for Resource Management
SSD	Single Shot Detector
SSH	Secure Shell
TN	True Negative
TP	True Positive

List of Figures

List of Tables

1 Introduction & Motivation

Bicycles are a very important means of transport in Munich – every day hundreds of thousands residents choose their bike for an emission-free, fast, athletic and timetable-independent arrival at their desired destination (Schmidt 2019). In addition to safe bike paths, parking facilities at the destination are an important part of the bike infrastructure. The City of Munich (Landeshauptstadt München, LHM) provides at approximately 350 sites more than 34,000 bike parking racks. These are distributed over 2000 bike parking facilities. This way, many cyclists can safely and orderly park and lock their bikes at subway stations, bus stops, public buildings, universities, schools, etc.

According to Allgemeiner Deutscher Fahrrad-Club e.V. (2011) (ADFC, German Cyclists Association) the supply of bicycle racks depends on several factors. One is the choice of the right type of rack. This should be easily visible to passers-by and not pose a risk of injury to users, as well as to children playing. For theft-proof parking, it must be possible to lock the frame and one of the wheels. In addition, the rack must prevent the bike from being damaged, for example by tipping over. This also includes a suitable distance to the neighboring rack to prevent the handlebars, cables, etc. from getting entangled. In addition, the choice of location is decisive. If the bicycle parking facilities are used for short periods, e.g., for shopping, or for several hours, e.g., during working hours, the racks must be easily and quickly accessible. For racks designed for long-term parking, e.g., at the train station, weather protection is desirable. In addition, the question of location requires an analysis of demand.

With so many bike racks spread throughout the City of Munich, a good system for inventory and maintenance of the facilities is needed. For these purposes, LHM would like to achieve a geo-referenced data storage for the bike racks. The accounting asset management keeps information about the bike racks without the possibility for georeferenced evaluations. A manual recording and transfer into a geoinformation system (GIS), as well as keeping the parking facility footprints up to date is time-consuming and costly. Therefore, an automized process for detection and, if possible, classification of bike racks is desired.

Within the administration, there is now the question of how changes in the bicycle rack distribution can be tracked directly in a GIS without detours via the used SAP system. Manual maintenance takes a lot of effort and an amount of 70-80% correct data through automated processes would be a great step forward.

The idea is now to use deep learning on aerial images to find the footprints of bike parking facilities in Munich. In recent years, deep learning has become an important tool for the

automatic classification and detection of objects in images. It also enriches the processing of satellite and aerial imagery and has become an important research topic in the field of remote sensing (Ma et al. 2019). As a result, this project brings together methods from different disciplines: geoinformatics, computer vision and remote sensing.

This results in the task and the research questions of this thesis:

The task is to train a deep learning model on aerial images provided from 2019. It shall be examined whether the deployment of a deep learning model for object detection in aerial images trained only on RGB is sufficient. Or whether a model using additional data such as infrared data (CIR), a Normalized Difference Vegetation Index (NDVI) raster and a normalized digital surface model (nDSM/nDOM, difference between DSM and DTM) can support the detection. ArcGIS Pro supports the deep learning workflow by providing a “Deep Learning“ toolset including functions for preprocessing the data and a *Detect Objects using Deep Learning* tool (ESRI 2021e). Subsequently, postprocessing steps should be applied to store the position and size of the racks in the geodatabase in a meaningful way. A further task is to assess if it is possible to also identify the type of rack with the help of the trained deep learning model. Finally, a manual in German is to be written in order to make the steps for utilizing the model applicable for the administration. To summarize the individual steps of postprocessing, it is beneficial to aggregate them with an ArcGIS toolbox. Such an application can help the City of Munich to inventory their bike racks on a georeferenced basis whenever new true orthophotos are available.

Ground truth data of the existing bike racks is necessary as the basis of the training. For this purpose, a linear geodataset with bicycle parking facilities of the LHM is available at the time of 2019. Before the training can be executed, this must be converted into polygons that exactly represent the footprints of the facilities. In addition, aerial imagery from 2017 is available, on which the found model can be tested (with the restriction that the trees are in leaf and will therefore cover some of the racks).

This results in the following research questions for this thesis. On the one hand, it is a question of determining which bands of the raster data represent the best combination to achieve the highest precision in bike rack detection. For this, it is important to find optimal model parameters. On the other hand, it is about figuring out to what extent a classification of the different rack types is possible and which types can be distinguished. In addition, there is the question how shadows or empty bicycle racks and many bicycles in the facilities, respectively, influence the result.

2 Theoretical Background and Related Work

2.1 Deep Learning and its Concepts

2.1.1 Deep Learning Background

Deep learning belongs to the field of machine learning and even more general to artificial intelligence. With its breakthrough in the last decade, deep learning has gained importance in the field of remote sensing. It has become a valuable tool for processing remote sensing data due to its strength in big data image analysis (Zhu et al. 2017). Once a deep learning model has been trained for a specific application, one can apply it to similar but new images.

The concept of deep learning is based on artificial neural networks (ANNs). These relate to the functioning of the neural networks in the human brain. A single component of both the human and the artificial neural network is called a neuron. Figure 1 shows its simplified mathematical model. Each neuron receives its input signals x_i either from the input data or the neurons of the previous layer. Each of the input signals is multiplied by a weight w_i. Inside the neuron, the weighted input signals are summarized together with a bias b. The weights and the biases are the elements of a neural network which are adjusted during the training. The result of the neuron is processed with an activation function (see 2.1.2 Activation Layer) and then passed on to the next neuron. Several of these neurons form together a layer of the neural network. The entire ANN consists of layers linked by the connections between the neurons. The final layer is called output layer and the outputs of its neurons present the predictions made on the input data. (Li et al. 2021)

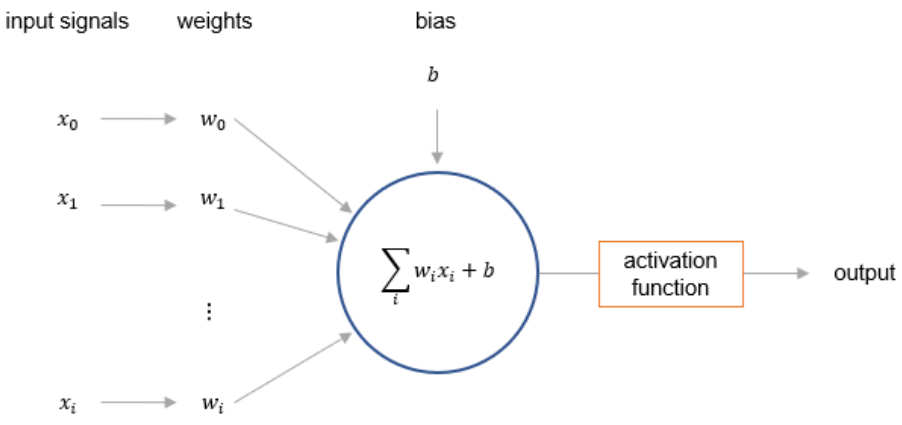


Figure 1: Mathematical model of a neuron (adapted from: Li et al. 2021)

The mathematical model of tensors is used to represent the input data and also the data that is passed on from neuron to neuron. For example, a 3D tensor is required to save an image with height, width, and multiple color channels. Each of the entries of the tensor represents one input signal x_i. (Chollet 2018)

In addition to the parameters that are adapted during the training of a neural network, there are hyperparameters. These are parameters that the user must define before starting the training. In the next chapter 2.1.2, some of these hyperparameters are described.

If a network consists of many layers (dozens to hundreds), it is referred to as deep learning. The number of layers represents the depth of the network. Between the first layer, the input layer and the output layer are the so-called hidden layers. Convolutional neural networks (CNNs), a certain type of ANNs, are the typical method for processing images in the field of deep learning. Based on CNNs, the functionality of deep learning is explained in the following. (Chollet 2018)

2.1.2 Convolutional Neural Networks (CNNs)

The layers of a CNN perform different arithmetic operations and give each network its individual architecture. Figure 2 shows the working principle of a neural network. The data of an image (input X) is transformed by tensor operations in several layers and thus lead to predictions Y'. In the model training phase, prelabeled images (true targets Y) are available for each input image. With the help of a **loss function**, the predictions Y' and Y are compared and feedback on the current performance of the model is obtained. Based on this feedback,

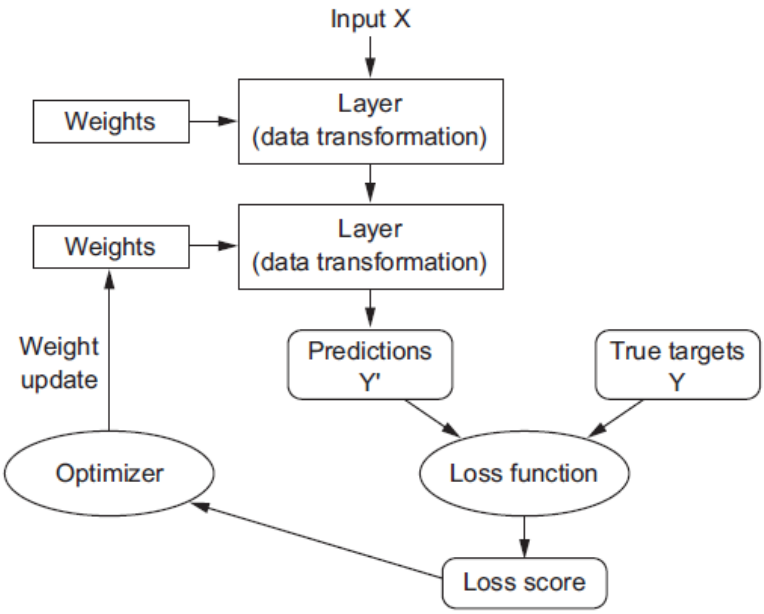


Figure 2: Outlined functionality of a CNN (Source: Chollet 2018)

the weights of the layers can now be adjusted, so that in further runs, the predictions Y' match the true targets Y even better, and the loss function improves iteratively. The training of a model is finished as soon as the loss function is minimal. Thus, the training can be described as an optimization problem. For the optimization, the chained derivatives of the tensor operations are needed. These derivatives are called gradients. With the iterative process of gradient descent the optimum is calculated. (Chollet 2018)

The forward steps, when the input passes through the layers, is referred to as feedforward. The backward steps, which is the adjustment of the weights and biases, are carried out by means of **backpropagation**. The optimizer describes the backpropagation algorithm, which calculates the gradients going backwards from the last layer to the first, uses the chain rule for derivatives and applies the updates accordingly. An important hyperparameter is the **learning rate** of the optimization problem. The optimizer is responsible for improving the network step by step. The learning rate defines the size of the steps for the gradient descent. If it is too small, reaching the minimum takes a long time and training can get stuck in local minima. On the other hand, if it is too large, it can happen that the function skips the optimum. (Chollet 2018)

The training images fed into the network are split into two sets: a **training set** and a **validation set**. Only the training set passes through feedforward and backpropagation. The validation set is not involved in the backpropagation process. By this, two losses, the training loss and the validation loss, emerge. These are used to evaluate the model performance. The percentage distribution of the images between the two groups is a hyperparameter and therefore used to improve the model configuration. A common ratio is 80:20. (Chollet 2018)

Supplementary, the training images are divided into batches. The **batch size**, another hyperparameter, describes the number of images that are processed before the weights of the model are updated. As soon as all images of the training and the validation set have been processed once, an **epoch** is complete. The number of epochs, like the other hyperparameters, must be set beforehand by the user. The number of epochs required varies depending on the architecture of the network, the amount of training data, and other hyperparameter settings. Too few epochs result in a model that is not yet optimally trained. The loss function has not yet reached its optimal value. Too many epochs can lead to overfitting. Therefore, it is essential to find an optimal number of epochs (see chapter 2.1.3). (Chollet 2018)

As the architecture of a model is described by the sequence of different layer types, the most important and most frequently used layers of CNNs are explained briefly in the following:

Convolutional Layer

Convolutional layers (often abbreviated to *conv*) make up the largest part of a CNN and give this type of network its name. The principle of such a layer is that a filter slides over the input tensor. Each value is replaced by the weighted average of the values surrounding it. A possible correlation between the filter and the input would be reflected in the result. Thus, features are recognized in the input data. The results of the convolution are written into so-called **feature maps**. (Dumoulin and Visin 2016)

Important parameters are the size of the filter and the stride, i.e., how far the filter is moved after each calculation. A typical filter size is 3 x 3. The third dimension of the filter always corresponds to the depth of the input. For example, if it is an RGB-image with three bands, the filter is 3 x 3 x 3 in size. With a stride of 1, the filter is shifted to the neighboring value in the next step. With a stride of 2, one value is skipped. This also results in a feature map that is halved in width and height. Another parameter of a convolutional layer is padding. It specifies whether further numbers, often zeros, are inserted at the edges of the input. This means that the filter can also be applied at the edge and there is no reduction in the feature map size. (Dumoulin and Visin 2016)

Figure 3 shows an example for a 3 x 3 filter in 2-dimensional space. The filter (gray) slides over the matrix (blue) with a stride of 1. The result of each filter operation is then saved in the feature map (green). Since no padding is used here, the size of the feature map is reduced to 3 x 3 compared to the input (5 x 5).

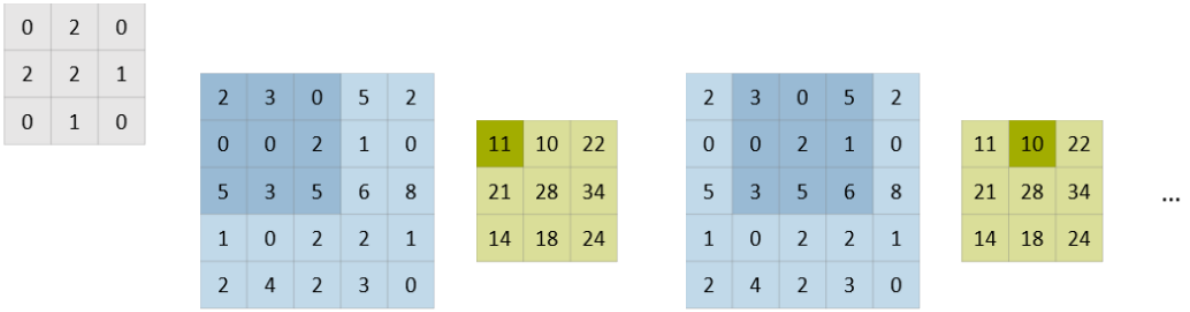


Figure 3: Convolution example: filter (gray) slides over matrix (blue), dot product is calculated at each position and written into the feature map (green) at the corresponding position (adapted from: Dumoulin and Visin 2016)

Activation Layer

The activation of a layer can either be interpreted as part of the previous layer or as a separate layer. The **activation function** decides which results, often from the convolution, are passed on to the feature map. The activation function ensures that the linearity of the layers is supplemented by a non-linear function. One also speaks of a non-linearity. CNNs often implement **ReLU** (rectified linear unit) (formula (1)) as an activation function. Adding a bias can be used to shift the function. (Chollet 2018; Li et al. 2021)

$$f(x) = \max(0, x) \tag{1}$$

Softmax Layer

A network that implements classification needs a last layer that outputs the probability distribution for the classes. A softmax layer is used for normalizing a vector and for transforming it so that it contains values between 0 and 1 that sum up to 1 with formula (2). (Goodfellow et al. 2016)

$$softmax(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2}$$

$\vec{z}$: input vector $\begin{pmatrix} z_0 \\ \cdots \\ z_K \end{pmatrix}$, K number of classes

z_i: value at position i in $\vec{z}$

Fully-Connected Layer

When all input signals of one layer are connected to all neurons of the following layer, it is called a fully-connected layer (FC). This type of layer is often used at the end of a CNN for classification. (Li et al. 2021)

If a network is referred to as a **Fully-Convolutional Network** (FCN), it implies that it does not contain any fully-connected layer. Instead, it contains 1 x 1 convolutional layers that perform the classifier tasks towards the end of the network.

Batch Normalization Layer

Batch normalization (BN) layers are used to make learning easier and to normalize the values processed by the neural network, e.g., after convolution. The inputs are linearly transformed to obtain zero means and unit variances. The BN layers make the network more robust with regard to poor initialization and allow higher learning rates. They are usually used after convolutional and fully-connected layers. (Ioffe and Szegedy 2015)

Pooling Layer

The purpose of a pooling layer is to reduce the dimension of the feature map. By this downsampling, the number of parameters decreases, and the feature information is accentuated. The layer is often used after convolutional layers.

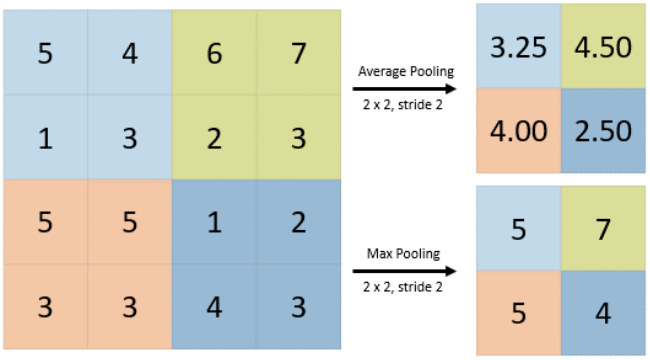


Figure 4: Examples of Average and Max Pooling with a 2 x 2 filter and a stride of 2 (adapted from: Li et al. 2021)

It performs a special kind of convolution. Usually, a filter of the size 2 x 2 is slided over the raster with a stride of 2 to decrease the dimensions by a factor of 2. The operation applied by the filter is in most cases a max function that returns the largest of the four values (Max Pooling). Alternatively, the average can also be calculated (Average Pooling). Figure 4 gives examples of Average and Max Pooling with 2 x 2 filters and a stride of 2. The four red pixel values (5, 5, 3 and 3) are for example mapped to 4 when calculating the average and to 5 when using the Max Pooling. (Li et al. 2021)

Residual Connections

When stacking many layers to get a deep network, the danger of gradient vanishing arises. To make stacking possible nevertheless, He et al. (2016) introduced residual connections. In addition to the block of layers that the input passes through, there is also an identity connection that concatenates the original input with the feature map at the end of the block (Figure 5). Thus, this "shortcut", the residual connection, skips layers. This identity connection allows for a simplified and more stable update of the weights during backpropagation.

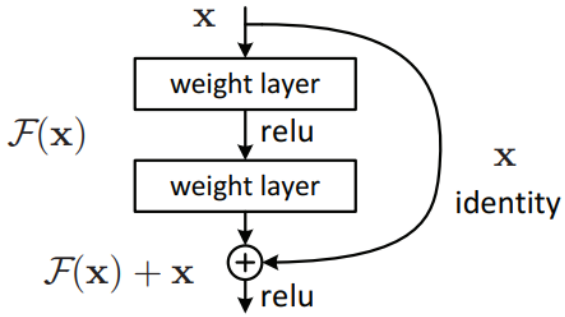


Figure 5: Residual Connection Block (He et al. 2015)

Backbone & Head

When describing a network architecture, layers are often summarized with the terms backbone and head. In the field of deep learning, the **backbone** of a neural network performs the feature extraction. It receives the input images and is the first part of the model. Commonly used backbones include AlexNet (Krizhevsky et al. 2012), VGGNet (Simonyan and Zisserman 2015), or ResNet (He et al. 2016), which are pretrained on large image datasets such as ImageNet (Deng et al. 2009). The final part of the network is called **head**. It is possible for a network to have more than one head. The head or heads of a network include the layers that are used for finetuning and for special tasks such as class prediction.

2.1.3 Optimal Model

In the previous paragraphs, several hyperparameters were mentioned that the user has to set to initialize the network before training. Therefore, finding an optimal model requires trying different parameter combinations and iteratively improving the model by adjusting them.

For this process, two values in particular are of great importance: the training loss and the validation loss. The first goal is to minimize the training loss. When training a model, however,

it is important that neither underfitting nor overfitting occurs. For this purpose, the validation loss function is also considered.

Underfitting occurs when the model has not yet been sufficiently trained. It is also possible that the model itself is not suitable for recognizing the features. As a result, the loss cannot be reduced sufficiently. Solutions for this are to increase the number of epochs or to increase the number of trainable parameters by improving the model's complexity. Figure 6 shows that up to the best fit (dashed line), both the validation and training loss decrease. Up to this point, therefore, underfitting is present.

Overfitting occurs when the model has memorized the training data too well and can no longer be applied to unknown data. Figure 6 shows the point at which underfitting turns into overfitting. When overfitting, the difference between training and validation loss is large and the validation loss starts to increase. To avoid overfitting, it is necessary to adjust the hyperparameters accordingly. The training must be stopped as soon as the validation loss increases. In addition, the model can be regularized by dropout. This means a random exclusion of neurons in computation steps. Another possibility to extend the generality of the model is to increase the training data. This also includes data augmentation with which allows for the training data to be extended and modified to prevent overfitting. Data augmentation applies random transformations to copies of the input data and enlarges by these the amount of training data. Examples of transformations are rotation, brightness changes, color modifications, cropping, etc. (Li et al. 2021; Chollet 2018; Goodfellow et al. 2016)

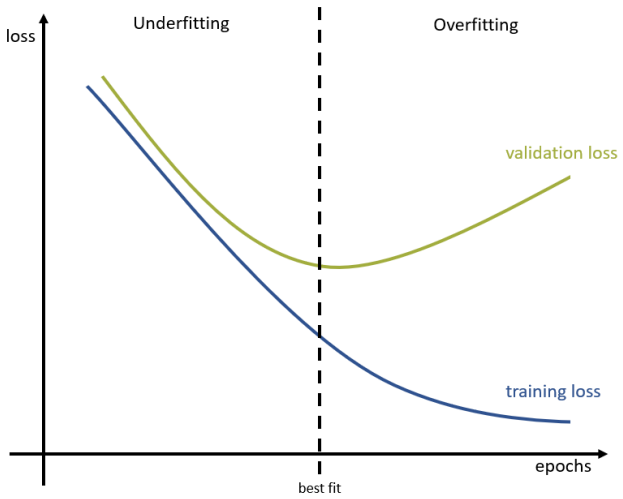


Figure 6: Overfitting and underfitting indicated by training loss and validation loss
(adapted from: Baeldung 2020)

2.1.4 Performance Evaluation

The performance of the trained model after object detection can be evaluated using the following methods:

Confusion Matrix

Detected objects belong to either the group of correctly detected (true positive, TP) or the group of wrongly detected objects (false positive, FP). When comparing the outcome with the ground truth, two more groups are a possible result: objects that are present in the ground truth but are not detected (false negative, FN) and areas where both in ground truth and detection are no objects, which is a correct result (true negative, TN). The latter case is in terms of object detection not countable and is therefore not of interest. The numbers of the groups can be summarized in a so-called confusion matrix (Figure 7). (Tharwat 2021)

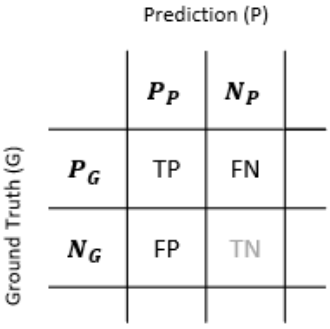


Figure 7: Confusion Matrix
(P: positive; N: negative)

With the values shown in the confusion matrix, further parameters can be calculated (Tharwat 2021):

Precision & Recall

$$precision = \frac{TP}{TP + FP} \tag{3}$$

$$recall = \frac{TP}{TP + FN} \tag{4}$$

Their counterparts are:

False Discovery Rate (FDR) & False Negative Rate (FNR)

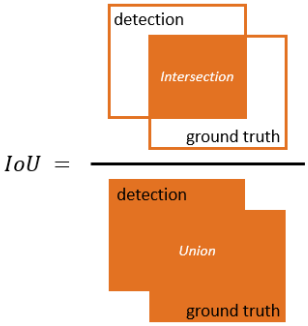
$$FDR = 1 - precision = \frac{FP}{TP + FP} \tag{5}$$

$$FNR = 1 - recall = \frac{FN}{TP + FN} \tag{6}$$

The goal is to reach high values for precision and recall while FDR and FNR are to be minimized.

A value for comparing the detected object area with the actual object area is the **Intersection over Union (IoU)**. It represents the ratio of the area of overlap to the area of union (formula (7), Figure 8). The IoU is also an important indicator for the Non Maximum Suppression, for

example. **Non Maximum Suppression** chooses the best fitting bounding box proposal by means of IoU and a confidence value that represents the probability of an object being correctly detected. (ESRI 2021d)



$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \quad (7)$$

Figure 8: Intersection over Union (IoU)
(adapted from: ESRI 2021d)

2.1.5 Classification, Object Detection, Segmentation & Instance Segmentation

In the area of image processing, deep learning is used for four main tasks (see Figure 9) (Garcia-Garcia et al.; ESRI 2021c):

- **Image Classification**: the entire image is assigned to one class
- **Object Detection**: bounding boxes highlight the location of object(s) on the image
- **Semantic Segmentation**: every pixel in the image is assigned to a certain class
- **Instance Segmentation**: combination of object detection and segmentation, in which the location of objects is determined; the positions of the objects are not only roughly indicated by a rectangular mask, but the pixel accurate shape is extracted

Figure 9: Image classification, object detection, segmentation, and instance segmentation examples
(Source: Wu et al. 2020)

Since the detection of bicycle racks in aerial imagery involves determining the exact position of the facilities, this is an object detection or an instance segmentation task. A segmentation of the imagery is not practical as the area of the bike racks is small compared to the rest of the city area and the objects are to be classified as a whole and not pixel by pixel.

2.2 Object Detection in ArcGIS Pro

ArcGIS Pro and the ArcGIS API for Python (version 1.8.5), respectively, provide five different deep learning frameworks for object detection and instance segmentation. In the following, these are briefly described, and it is discussed which architecture is used for this work.

For the descriptions of the network architectures the term **bounding box regression** needs to be described: To define the location of an object in an image, the bounding box of the object is defined by four coordinates: x and y (image coordinates) for the upper left corner and x and y for the lower right corner or x and y together with w and h for width and height. In a regression process, this bounding box is adapted to the object. The bounding box regression returns with the help of fully-connected layers and sigmoid activation, the bounding box, also called mask, of an object.

SSD

SSD, which is spelled out as Single Shot Detector, describes a one-stage Deep Learning model that detects objects in a single pass of the image. It consists of a backbone model (ResNet or VGGNet) and of convolutional layers added to the backbone, called SSD head (Figure 10). The images are divided using a grid. In each of the grid cells the class and location of the objects in that region are predicted. It uses the concept of anchor boxes with different aspect ratios and zoom levels for this. (Liu et al. 2016; ESRI 2021k)

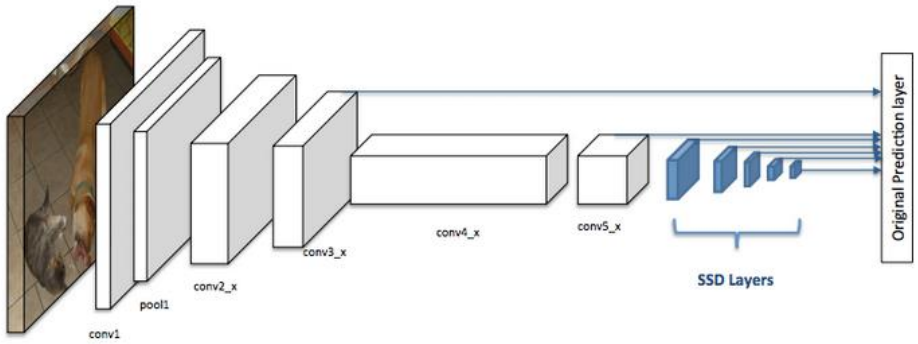


Figure 10: Simplified representation of the Single Shot Detector architecture (Source: ESRI 2021f; adapted from Liu et al. 2016)

YOLOv3

YOLOv3 (You Only Look Once) was presented by Redmon and Farhadi in 2018 as an improvement of the first two versions of YOLO from 2015 and 2016. It is also a one-shot object detector. The ArcGIS implementation uses Darknet-53, a CNN with 53 convolutional layers including residual connections, as a backbone. It adds upsampling and detection layers for detection at different zoom levels (Figure 11), which leads to a total number of layers of 106.

YOLO particularly stands out due to its real-time detection capability, e.g., for videos. (Redmon and Farhadi 2018; ESRI 2021l)

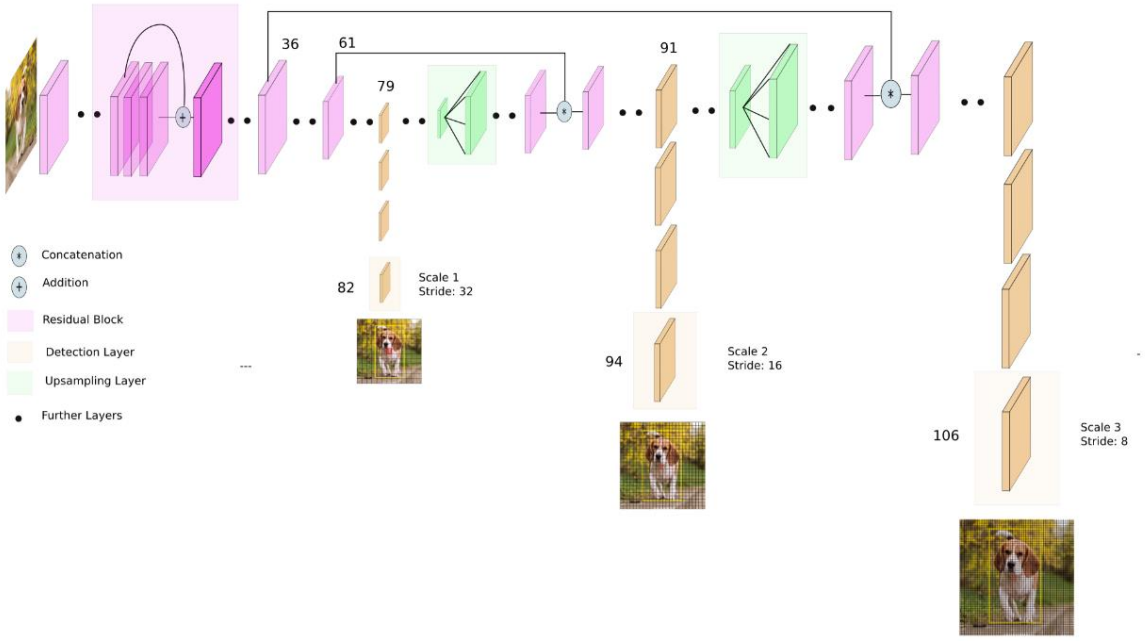


Figure 11: YOLOv3 architecture (Source: Kathuria 2018)

RetinaNet

Lin et al. introduced RetinaNet in 2017b – likewise a one-stage architecture for object detection. It consists of a Feature Pyramid Network (see chapter 2.3.1) based on a ResNet backbone that produces a convolutional feature map over the entire input image. Two subnetworks follow the backbone layers (Figure 12): the first subnetwork performs the object classification being an FCN, while the second subnetwork, also an FCN, determines the location of the object with reference to the anchor box. In order to handle class imbalances, they introduced the new Focal Loss function. (Lin et al. 2017b; ESRI 2021j)

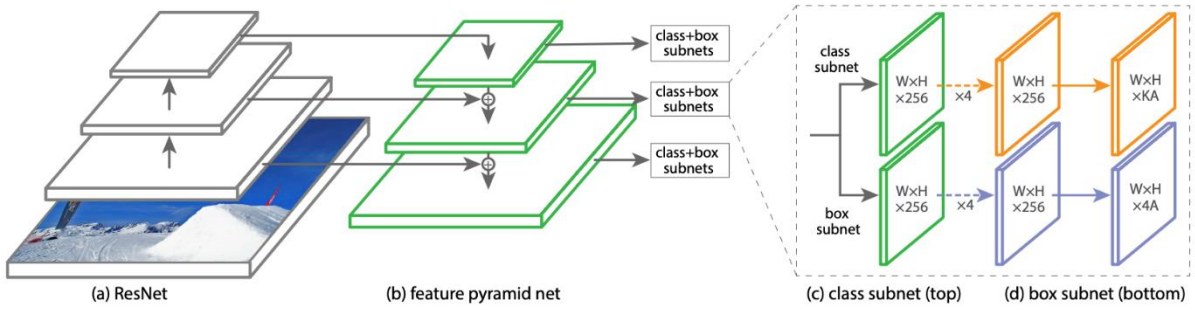


Figure 12: RetinaNet architecture (Source: Lin et al. 2017)

Faster R-CNN

Faster R-CNN, presented by Ren et al. (2017), is a successor of R-CNN (Girshick et al. 2014) and Fast R-CNN (Girshick 2015). It belongs to the group of two-step models as it has an extra region proposal network. This is implemented after the convolutional backbone (ResNet) to determine regions of interest (RoIs). After the step of RoI Pooling, which matches the feature maps of the CNN with the RoIs, the classifier follows. It is a fully connected layer with two heads, a softmax classifier, and a bounding box regressor (Figure 13). (Ren et al. 2017; ESRI 2021h)

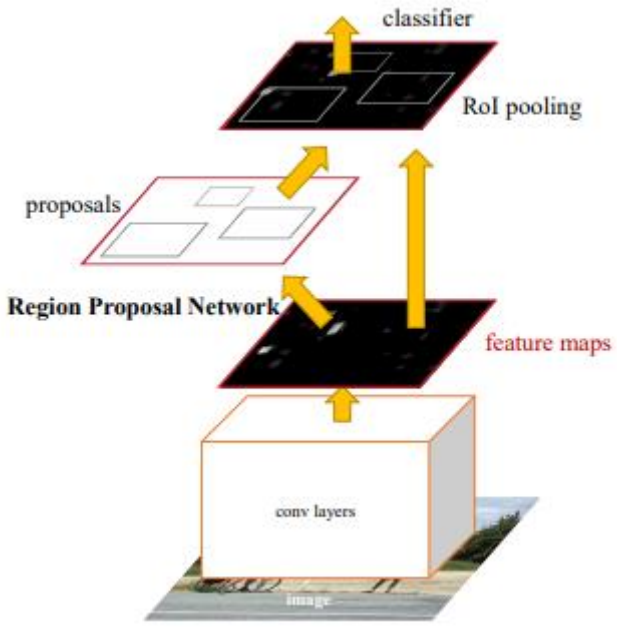


Figure 13: Simplified representation of the Faster R-CNN architecture including Region Proposal Network (Source: Ren et al. 2015)

Mask R-CNN

He et al. presented the region proposal network Mask R-CNN in 2017. It stands for Mask region-based convolutional neural network, and it extends Faster R-CNN (Ren et al. 2017) and is therefore also successor of Fast R-CNN (Girshick 2015). The network adopts the basic structure of the two successive stages. It is an architecture for object instance segmentation as it predicts objects and its class at the same time as predicting a pixel wise segmentation mask. It implements two parallel head networks: one for classification and bounding box regression and the other one for the exact calculation of the segmentation mask (Figure 14). More details on this architecture in chapter 2.3. (He et al. 2017; ESRI 2021i)

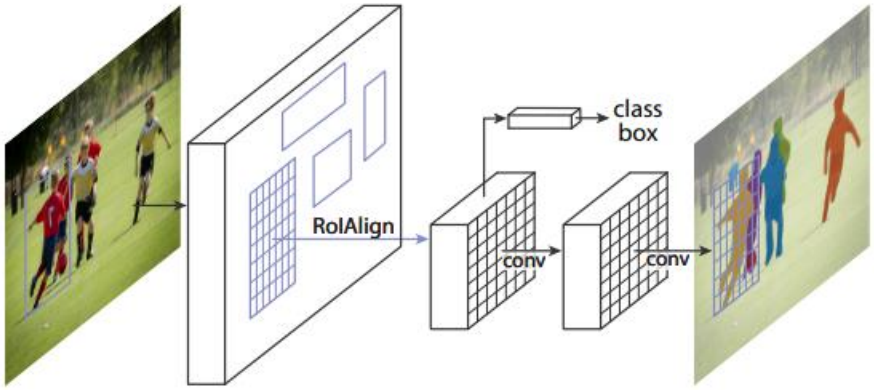


Figure 14: Simplified representation of the Mask R-CNN architecture (Source: He et al. 2017)

Comparing these five models, the one-stage detectors are faster than Faster R-CNN and Mask R-CNN as they do not have a region proposal network. However, in this work, speed does not play an important role, as the detection of bike parking facilities in the aerial imagery is not a real-time application. Liu et al. (2020) found that the two-stage detectors were superior to the one-stage models in terms of accuracy. Furthermore, Mask R-CNN achieves better results than Faster R-CNN. In addition, Mask R-CNN has the decisive advantage that it is an instance segmentation network. Thereby, the objects are segmented with pixel accuracy and the position is not only shown approximately with a rectangular mask. Therefore, Mask R-CNN is the chosen model for the purpose of this thesis.

2.3 Implementation Details of Mask R-CNN

In order to get a precise idea of the functionality of the model used, the components of Mask R-CNN are described in detail below. Accompanying this, Figure 17 shows the network structure with focus on the most important layers and relationships between the individual components.

In order for the input images to be optimally processed by the network, they must be adjusted beforehand. First of all, the mean value of all pixel values of the entire training dataset is calculated for each band and then subtracted from each pixel, in order to achieve a distribution of the values around zero. Then the image matrix is converted into a tensor and its dimensions are adjusted with rescaling and padding so that the image height and width are multiples of 32. This is necessary when using Feature Pyramid Networks (FPNs). The resulting tensors are then ready to be put in the network. (Fractal AI@Scale Research Group 2019)

2.3.1 Backbone ResNet and Feature Pyramid Network (FPN)

The backbone of Mask R-CNN is a Residual Network / ResNet introduced in 2016 by He et al. It exists in different versions and is named after the respective number of layers: ResNet-18, ResNet-34, ResNet-50, ResNet-101, Resnet-152, etc. ResNet-50 is the most widely used, as it has a good average in terms of weights to train and therefore run time and performance compared to the other ResNets. Residual connections including convolutional and batch normalization layers characterize this network. The residual connections facilitate a higher number of layers, as they prevent the gradient from vanishing as mentioned before. ResNet-50 (Figure 15) can be described as a network consisting of four stages (C2, C3, C4, C5), which are made up of residual blocks. Layers at the beginning and at the end frame this. Within each stage, the number of feature maps doubles, while the height and width of each raster halve.

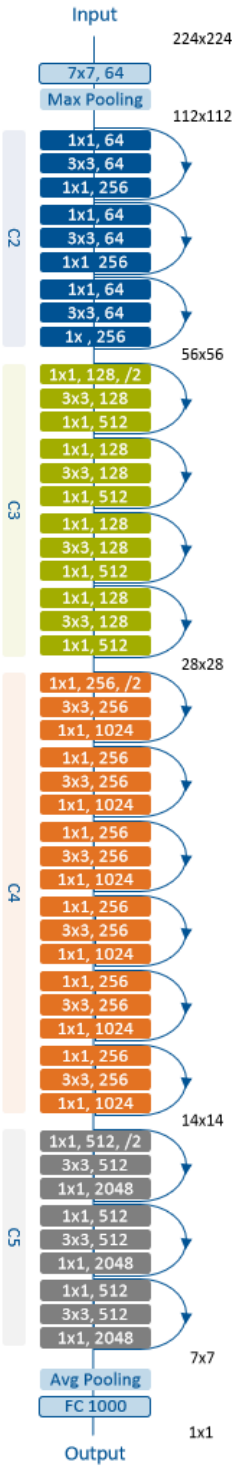


Figure 15: ResNet-50 convolutional layers (with filter size) grouped in blocks by residual connections and arranged into four stages (C2 - C5), numbers in black give the size of the output after each stage (adapted from He et al. 2016; Ankit 2019)

The ResNet backbone is connected with layers that form a FPN (Lin et al. 2017a) depicted in Figure 16 (simplified in Figure 17a)). An FPN has a bottom-up network, here ResNet-50. The ResNet is connected with horizontal connections to a top-down network. The connections perform 1 x 1 convolutions to adapt the third dimension of the feature maps to 256, so that all have the same depth. The top-down pathway adds the upsampled feature map to the output feature map of the previous stage. Each of the feature maps in the top-down part of the model is processed with a 3 x 3 convolution. This results in four feature maps of different sizes. A fifth feature map is created by Max Pooling. (Fractal AI@Scale Research Group 2019)

The resulting feature maps can then be passed on to the next layer of the network. In this case, the Region Proposal Network.

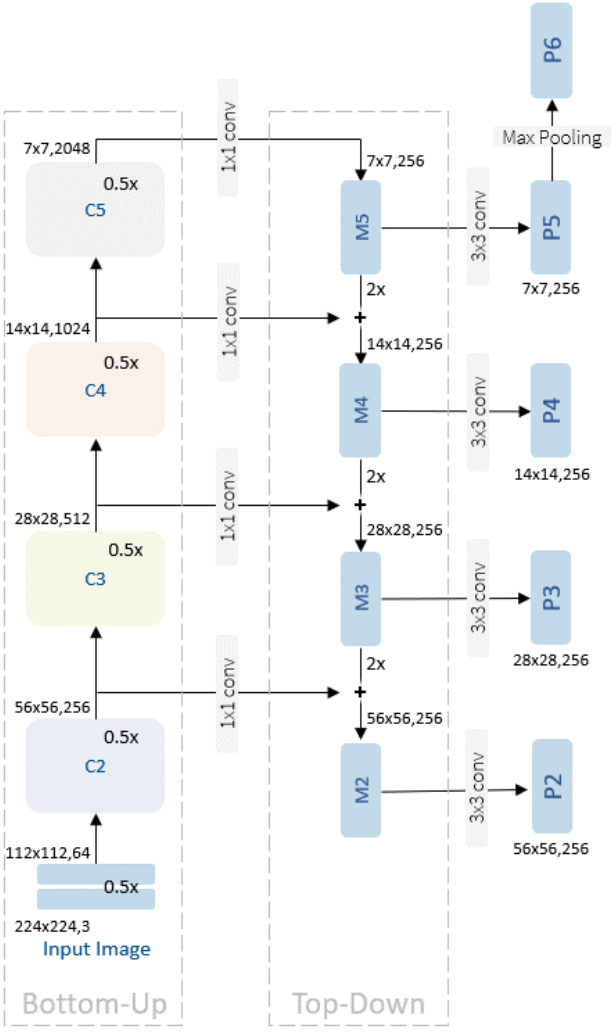


Figure 16: Feature Pyramid Network (FPN) based on ResNet-50 as Bottom-Up Backbone and Top-Down Upsamling for generating feature maps of different size (adapted from: Lin et al. 2017a; Hui 2018; Fractal AI@Scale Research Group 2019)

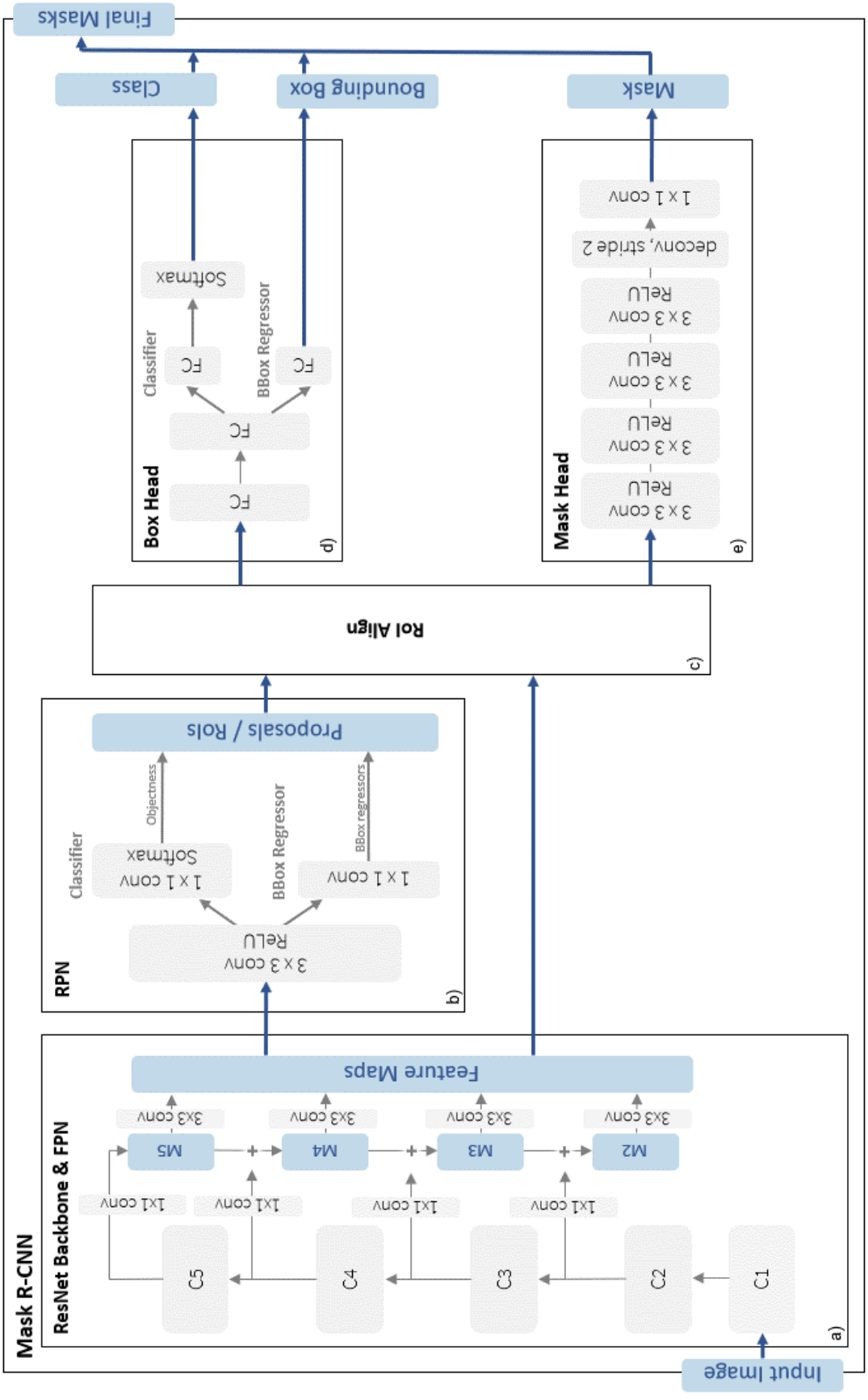


Figure 17: Architecture of Mask R-CNN; a) Backbone and FPN; b) RPN; c) RoI Align; d) Box Head; e) Mask Head (adapted from: He et al. 2017; Hui 2018; Fractal AI@Scale Research Group 2019)

2.3.2 Region Proposal Network (RPN)

The Region Proposal Network (RPN) was introduced by Ren et al. (2017) for Faster R-CNN. It returns rectangular object proposals for each input image. Each of the proposals receives an objectness score that measures the probability of object presence.

The RPN starts with a 3 x 3 convolutional layer (see Figure 17b)). At each filter location, $k = 9$ anchors are generated with three different scales and three different aspect ratios. Each anchor is centered at the filter's center point (Figure 18). This results in $W * H * k$ anchors (W x H: width x height of feature map). This anchor method makes the proposal generation translation-invariant.

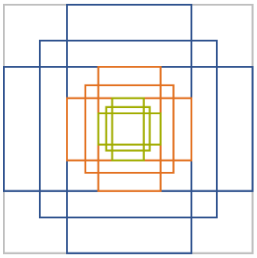


Figure 18: Anchor Generation (adapted from: Mohan 2021)

After this convolutional layer and ReLU, two branches follow: the classifier and the regressor. The classifier consists of a 1 x 1 convolutional layer, followed by a softmax layer, and gives for each anchor the probability of being foreground or background. This results in $2 * k$ objectness scores. The regressor computes $4 * k$ coordinates giving the offset between anchor box and the object's position in the ground truth also with a 1 x 1 convolutional layer.

The loss of the regressor is dependent on the IoU overlap of the anchor boxes and each respective ground truth. An anchor is assigned positive when the IoU ratio is greater than 0.7. When no anchor reaches this ratio, the one with the highest value is chosen. It gets a negative label when the IoU ratio is lower than 0.3. Anchors with a IoU overlap in between are dropped. Simplified, the loss can now be described as in formula (8)[1]. Whenever an anchor is assigned negative, the regression loss is deactivated by p_i^* being 0.

$$L = \frac{1}{N_{cls}} \sum_i L_{cls} + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg} \qquad (8)$$

L_{cls}: classification loss (log loss over two classes object / no object)

L_{reg}: regression loss (smooth L_1)

i: index of an anchor in a mini-batch

$p_i^* = 1$: when anchor is positive; $p_i^* = 0$: when anchor is negative

N_{cls}, N_{reg}: normalization

λ: weighting (default: 10)

[1] for detailed loss function see Ren et al. (2015)

2.3.3 RoI Align

The next step (Figure 17c)) is to match the proposals, also called RoIs, with the computed feature maps.

The precursors of Mask R-CNN use RoI Pool for obtaining small feature maps from RoIs. As Mask R-CNN is a model for instance segmentation, it requires good alignments between each RoI and the extracted features. RoI Pool uses quantizations that are sufficient for object detection but introduce inaccuracies in terms of pixel-precise detection of segmentation masks. For this reason, He et al. (2017) introduce RoI Align to properly align the RoI and the extracted features.

Figure 19 describes the functionality of RoI align. The RoI's boundaries do not fit the pixel arrangement of the feature map. The first step is to divide the RoI into m x n bins, where m and n are width and height representing the desired size of the aligned output feature maps. Each bin is assigned four regularly sampled locations. The value for each of the points is calculated with bilinear interpolation of the four closest feature map pixel values. The final value of each bin is obtained by average pooling of the four values in each bin. This method is used for all RoIs and the corresponding feature maps. This results in aligned output feature maps, all of which are m x n in size.

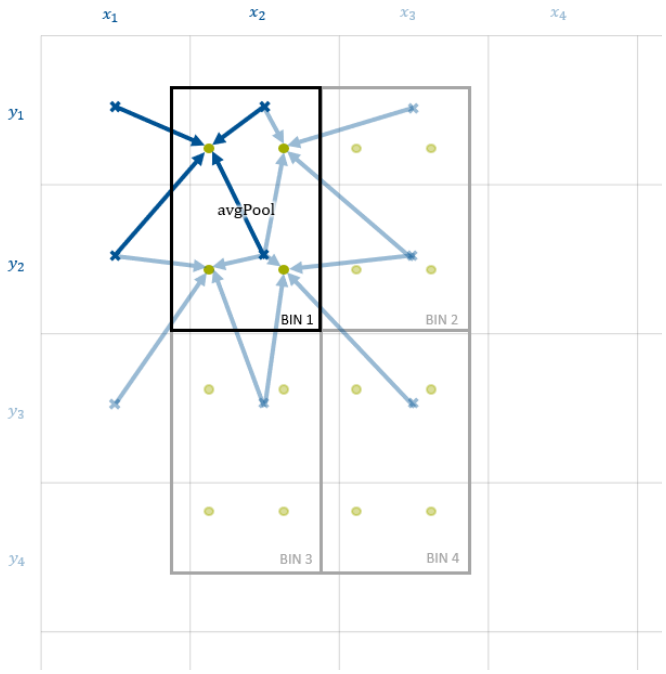


Figure 19: RoI Align
each bin has four regularly sampled points (green), the value of each point is calculated with bilinear interpolation of the closest pixels' values (blue), the value of each bin is obtained by average pooling (adapted from: Hui 2018; Fractal AI@Scale Research Group 2019)

2.3.4 Heads

The aligned feature maps are now forwarded to the network's heads. The bounding box and class prediction head works in parallel with the mask head that produces the segmentation masks.

Box Head

In the box head, ROI-Align is followed by two FC layers. Next, one FC layer is provided for bounding box prediction. It outputs bounding box regression values. Likewise, an FC layer is supplied for the class prediction (see Figure 17d)).

Mask Head

Finally, the mask head is responsible for the binary segmentation masks on each RoI. It is a small FCN consisting of four 3 x 3 convolutional layers followed by ReLU, one 2 x 2 deconvolutional layer with stride 2 and one 1 x 1 convolutional layer (see Figure 17e)).

2.3.5 Loss Function

The total loss of Mask R-CNN is the sum of three values: the loss of the class prediction L_{cls}, the loss of the bounding box regression L_{box} and the loss of the mask head L_{mask}:

$$L = L_{cls} + L_{box} + L_{mask} \tag{9}$$

L_{cls} and L_{box} are the same as of Fast R-CNN (Girshick 2015):

$$L_{cls}(p, u) = -\log p_u \tag{10}$$

$p = (p_0, \dots, p_K)$: discrete probability distribution (per RoI) over $K + 1$ classes

u: true class

$$L_{box}(t^u, v) = \sum_{i \in \{x,y,w,h\}} smooth_{L_1}(t_i^u - v_i) \tag{11}$$

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & if\ |x| < 1 \\ |x| - 0.5 & otherwise \end{cases} \tag{12}$$

$v = (v_x, v_y, v_w, v_h)$: true bounding box regression

$t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$: predicted bounding box regression

L_{mask} is the average binary cross-entropy loss (formula (13)). To avoid competition between classes and to separate mask and class prediction, this loss is defined only for an RoI that is linked to the ground truth class k. The branch generates a mask with size $m\ x\ m$ for each RoI

and each class K. This results in $K * m^2$ outputs. y_{ij} describes the ground truth label of a cell at the position i,j. $\hat{y}_{ij}$ is the predicted label at the same position. (He et al. 2017; Zhou 2021)

$$L_{mask} = -\frac{1}{m^2} \sum_{1 \le i,j \le m} \left[y_{ij} * \log \hat{y}_{ij}^{k} + \left(1 - y_{ij}\right) * \log(1 - \hat{y}_{ij}^{k}) \right] \quad (13)$$

2.4 Localization and Inventory of Bike Racks

The approach of using areal images and deep learning for detecting bike racks is not present in the literature, although many cities are pushing the expansion of the cycling infrastructure. The cities have progressed to different degrees. While Amsterdam (Gemeente Amsterdam 2017) and Copenhagen (City of Copenhagen 2012), for example, have made considerable progress with their cycling strategies, other cities such as Rome (Roma Capitale 2020) or Vancouver (City of Vancouver 2021) are not that advanced yet. Apart from the expansion of cycle paths, planners view the availability of public cycle parking facilities as a crucial point. Individual cities rely on different strategies – for example bicycle parking garages, the promotion of private but publicly accessible bicycle racks, long-time parking bike lockers or after analysis of the needs, the setting up of city-owned racks. The inventory is carried out, as far as documents on this can be viewed, manually (e.g. Berlin (GB infraVelo GmbH 2021), Gothenburg (Kajsa Rosén 2021), Stockholm (Stockholms stad 2015)) or occasionally via crowd-sourcing projects (e.g. London (CycleStreets Ltd. 2021), Yakima, WA (City of Yakima Bicycle Committee 2021)). Some cities also offer cyclists the opportunity to view the location and number of racks in a web-based map application, sometimes even with a photo and additional layers with information on bicycle pumps, bicycle paths, etc. (e.g. Oslo (Bymiljøetaten Oslo Kommune 2020), Paris (Ville de Paris 2021), Wien (Stadt Wien 2011)). Figure 20 shows an example of a webmap that provides information including location, capacity, and picture on parking facilities in London.

The aerial photography and deep learning approach to localize the bicycle racks is therefore new. Also, other street furniture of a similar type, such as benches and rubbish bins, have not yet been associated with deep learning and aerial imaging in the literature. In contrast, publications on vehicle or airplane detection are more common. What is certain, however, is that many cities are thinking about how good and safe bicycle racks in appropriate numbers and in suitable places can promote bicycle traffic.

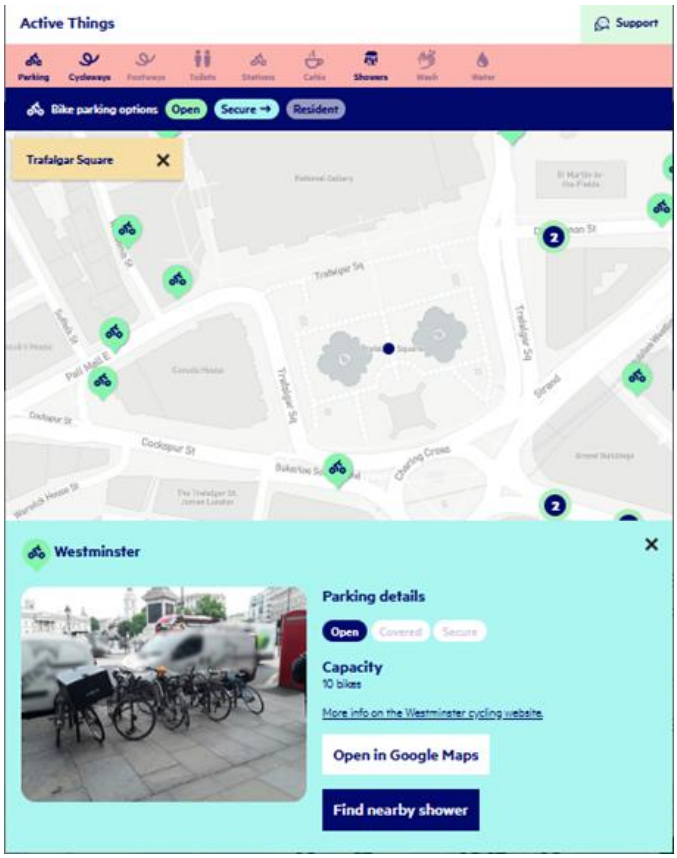


Figure 20: Webmap for, inter alia, bike parking facilities in London (Source: Active Things 2021)

Remote sensing poses special challenges to object detection and instance segmentation. This is due to the special nature of the data, which is acquired from a bird's eye view and not in profile like most imagery datasets. This results in 360° object rotations, complex and cluttered backgrounds (Li et al. 2018), different imaging conditions such as weather, season, and resolution (Li et al. 2020) but also occlusion, illumination changes, and shadowing (Su et al. 2019). In addition, remote sensing data can consist of more than the three common bands of red, green, and blue.

Remote sensing uses the same deep learning architectures that are being developed in computer vision. To meet the special requirements of remote sensing imagery, these are extended (e.g. Su et al. (2019)). Another important approach, to improve deep learning results in remote sensing, is to provide large aerial image datasets with labels to pretrain models. An example of such a dataset is DIOR (Li et al. 2020). It contains 23463 optical remote sensing images with 192472 objects in 20 object classes. Thus, good results can be obtained even with small datasets. ESRI provides pre-trained models for certain applications and continues to expand this (Viswambharan and Singh 2021). This already includes for example models for road extraction, the recognition of pools or building footprints, landcover classification and also for the detection of shipwrecks in bathymetry data (ESRI 2021m).

3 Data

3.1 Terminology

When speaking of a *rack* or *stand* in the following, it always means one parking spot for one bike (German term: *Stellplatz*). The term *bicycle/bike parking facility* describes a row of bike racks structurally recognizable as connected (German term: *(Fahr-)Radständer*). When the *location* is mentioned, the position of a bike parking facility is meant (German term: *Standort*). The grouping of several bike parking facilities is described with the term *macro-location* (German term: *Makrostandort*).

3.2 Bike Racks Data Situation

The Baureferat (building department) of the City of Munich primarily maintains data on the bicycle racks in an SAP-based platform, which does not allow geo-spatial analysis. Therefore, in 2014 a spreadsheet (PlanR_Datentabelle_stammdaten_2016_Auswertung.ods) including coordinates was created manually from paper documentations, the SAP-platform, etc.

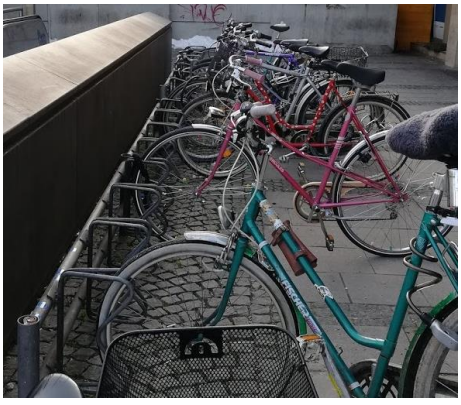
The content of the spreadsheet was transformed into a shape-file (Fahrradstaender_bis_2019.shp) with a line object for every parking facility. The data was continuously updated, so that the latest version contains bike parking facilities until 2020/03. When super-imposed with aerial photos, a non-systematic displacement and scaling is visible in some cases (see Figure 21). Some racks even do not exist anymore due to, for example, construction sites (further details see chapter 4.4.1).

Figure 21: Detail from LUFTBILD2019_RGB: Orleansplatz (Ostbahnhof: Nordkopf, NO-Aufgang): line-geometries (blue) are shorter than bike parking facilities or not on exact position
(Source: Landeshauptstadt München 2020)

It contains information on location (including street name and number), the number of racks, their type, the year and month of construction, some specifications like roofing, lighting and pavement, administrative data (funding, frequency of bike removal, …) and a column for annotations. Since some of the information is stored using coded values, the explaining tables in PlanR_Datentabelle_stammdaten_2016_Auswertung.ods are useful. Another shape-file (Makrostandort_bis_2019.shp) contains the macro-locations, grouping bike parking facilities close to each other and representing them by point geometry.

The Münchner Verkehrsgesellschaft (MVG; Munich Transport Corporation) offers a bike rental service called MVG Rad that has parking stations exclusively for the rental bikes in and around Munich. As these stations belong also to city owned bike parking facilities, they are added to the dataset. The locations of the stations are stored in 20210113_MVG_Radstationen.shp with point geometries. The file contains also stations outside the city, so it is clipped by the municipal boarder.

All the existing types of bicycle stands can be roughly divided into the following categories (for a better readability both English and German terms are included):

English term / German Term	**Commonness (bike racks / locations)**	**Picture**
Inverted-U / Anlehnbügel - old (Trabant) - new - alt (Trabant) - neu	ca. 7400 / 532	 Figure 22: Anlehnbügel neu, Tengstr. / Agnesstr. (new 2020)
Wheelwell / Klemmbügel	ca. 6000 / 274 no new installations (can lead to wheel damage, frame cannot be locked)	 Figure 23: Klemmbügel, Hauptbahnhof: Nordkopf, SO-Aufgang

English term / German Term	Commonness (bike racks / locations)	Picture
L15 - one-sided, low - one-sided, high/low - two-sided, low - two-sided, high/low - at an angle - einseitig, tief - einseitig, hoch/tief - doppelseitig, tief - doppelseitig, hoch/tief - schräg	ca. 15700 / 1073 - ca. 8900 / 718 - ca. 700 / 38 - ca. 3200 / 175 - ca. 1300 / 35 - ca. 1600 / 107	 Figure 24: L15 schräg, Elisabethplatz
Arreta	ca. 300 / 12	 Figure 25: Arreta, U-Bahnhof Maillingerstr.: Westkopf, S-Aufgang
Kappa - one-sided, low - one-sided, high/low - two-sided, low - two-sided, high/low - at an angle - einseitig, tief - einseitig, hoch/tief - doppelseitig, tief - doppelseitig, hoch/tief - schräg	ca. 1100 / 33	 Figure 26: Kappa schräg and Kappa hoch/tief, Laimer Unterführung

English term / German Term	Commonness (bike racks / locations)	Picture
MVG	not in data / 137	 Figure 27: MVG Rad Station, Oberwiesenfeld
Others - BikeCare / BCS - RGT	ca. 700 / 44	 Figure 28: BikeCare, Waldperlach
Double Decker / Doppelstockparker (roofed)	ca. 2800 / 31	 Figure 29: Doppelstockparker, Marienhof (new 2020)

Table 1: Bike rack types in Munich
(Source: Landeshauptstadt München 2020, own photographs)

As described, the locations of the bike parking facilities are only stored as lines. However, a polygon representation of each parking facility is necessary for training the model as exact

spatial extent is needed. Moreover, as mentioned above, not all lines are in correct position. The details on creating the polygon file are described in chapter 4.4.1.

3.3 Imagery

Table 2 and Table 3 list the data of aerial flights in 2017 and 2019 that is available as File Geodatabase Rasters. The extent of the rasters can be seen in Figure 30. The images are true orthophotos. Therefore, tilts of tall objects are eliminated and for example bike racks close to building walls are not covered. Nevertheless, artifacts can occur on the edges of buildings and trees that make some small regions impossible to interpret.

The bands of the rasters are composed as follows:

- **RGB**: spectral reflectance signature for Red, Green and Blue spectral bands
- **CIR**: spectral reflectance signature for NIR (near infrared), Red and Green spectral bands – this band combination is vegetation sensitive
- **DSM** (DOM): Digital Surface Model: height value representing earth surface including natural and built features like trees, buildings, etc.
- **DTM** (DGM): Digital Terrain Model: height value representing earth surface excluding natural and built features
- **nDSM** (nDOM): normalized Digital Surface Model, difference between DOM and DTM
- **NDVI**: Normalized Difference Vegetation Index, indicator for green vegetation, calculated from NIR and Red bands: $NDVI = \frac{NIR-Red}{NIR+Red}$

Composite layers can be formed from the individual raster files by stacking the bands. ArcGIS Pro provides the *Composite Bands* tool in *Raster Functions* for this purpose. With this tool the following composites are generated:

- RGB + nDSM
- RGB + NDVI
- RGB + nDSM + NDVI

2017 in leaf	**Resolution**	**Uncompressed Size (each)**	**Pixel Depth and Type**	**Number of Bands (each)**
Luftbilder_2017 (RGB)	0.1 m	192.78 GB	8 Bit unsigned char	3
Luftbilder_2017_CIR				

2017 in leaf	**Resolution**	**Uncompressed Size (each)**	**Pixel Depth and Type**	**Number of Bands (each)**
DOM_2017		257.05 GB	32 Bit floating point	1
DGM_2017				
nDOM_2017				
NDVI_2017_2				

Table 2: File Geodatabase rasters 2017 (Source: Landeshauptstadt München 2020)

2019 leafless	**Resolution**	**Uncompressed Size (each)**	**Pixel Depth and Type**	**Number of Bands (each)**
LUFTBILD2019_RGB	0.08 m	278.52 GB	8 Bit unsigned char	3
LUFTBILD2019_CIR				
LUFTBILD2019_DOM		371.36 GB	32 Bit floating point	1
LUFTBILD2019_DGM				
LUFTBILD2019_nDOM				
LUFTBILD2019_NDVI				

Table 3: File Geodatabase rasters 2019 (Source: Landeshauptstadt München 2020)

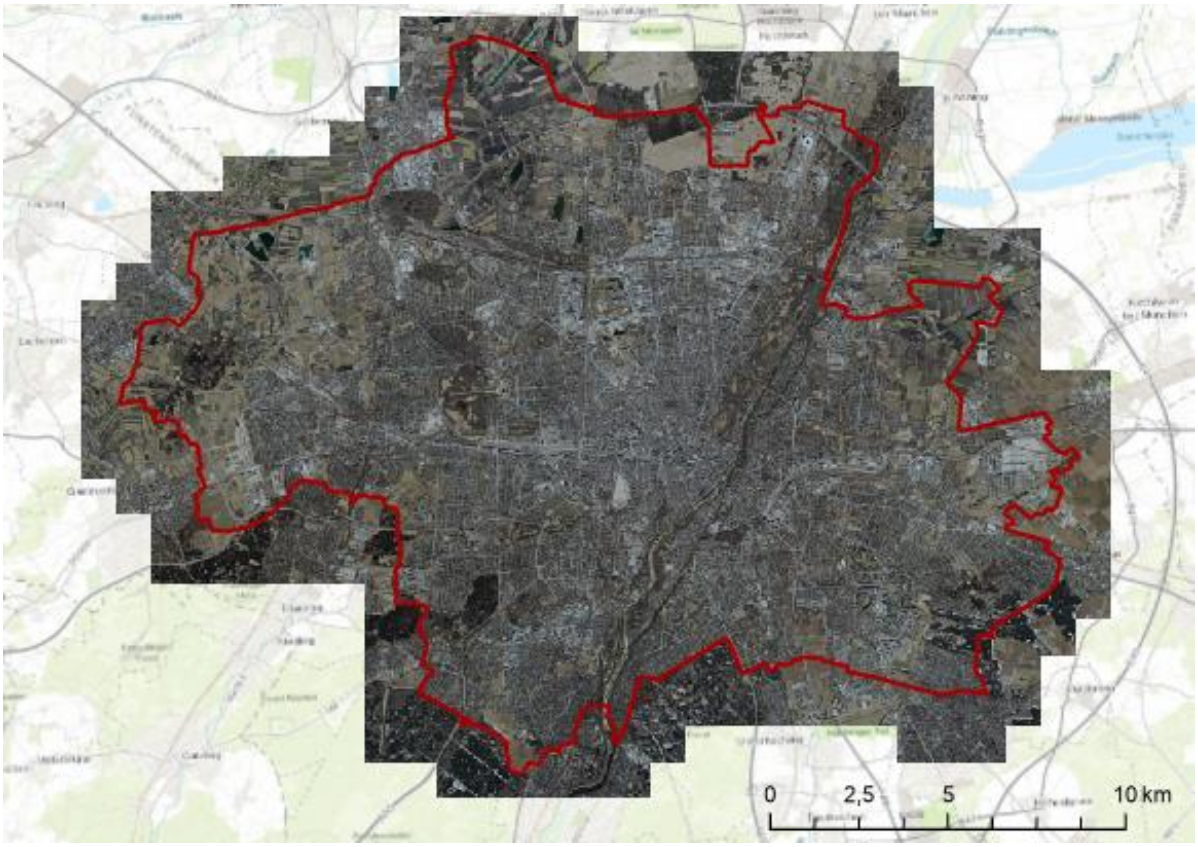


Figure 30: Area of LUFTBILD2019_RGB including municipal border of Munich in red
(Source: Bayerische Vermessungsverwaltung 2020, Landeshauptstadt München 2020, ArcGIS Map Service 2021)

3.4 Additional Municipal GIS Data

Additionally, a polygon dataset is available covering the public spaces in Munich to support the object detection process. Figure 31 shows an excerpt of the lhm_oeffentliche_Flaechen_v2.shp. It can be seen that it includes urban space like streets and parks together with buildings owned by the LHM like the school complex in Berg am Laim (*Sonderpädagogisches Förderzentrum, Städt. Ludwig-Thoma-Realschule, Mittelschule Am Echardinger Grünstreifen)* marked in blue. This dataset is intended to represent the areas on which the City of Munich places bicycle racks. However, this dataset is not perfect, even though it covers all the land areas where the labeled bicycle racks are located. But it does not cover, for example, all the property of Deutsche Bahn (German railroad company) which are also available for the installation of new public bicycle parking facilities. However, in order to clarify the functionality of such a layer in the step of object detection (described in chapters 4.8 and 4.9), this dataset is sufficient in any case.

Figure 31: Detail of lhm_oeffentliche_Flaechen_v2.shp (orange): Area between Ostbahnhof and Ostpark, with bike parking facilities (red) and school complex (*Sonderpädagogisches Förderzentrum, Städt. Ludwig-Thoma-Realschule, Mittelschule Am Echardinger Grünstreifen)* (blue); background: LUFTBILD2019_RGB
(Source: Landeshauptstadt München 2020)

4 Methodology

4.1 Overview

One of the goals of this work is a tool that can be used in ArcGIS Pro to detect bicycle parking facilities in Munich in aerial images with a trained model. The methodology to reach this goal is described in the following chapters. First, hardware and software are introduced before the individual steps are explained in more detail. Figure 32 gives an overview of the individual processing steps and shows in the colored boxes which software and hardware are used in each step. The initial step is the data preprocessing. It is divided into two steps: the first step, the labeling of the objects in ArcGIS Pro, is performed prior the second step, which is exporting the training data. The intermediate results of labeling and export are described and evaluated. A small part of the data preparation is performed right at the beginning of the training, which follows next: the training of the Mask R-CNN model takes place. The exact procedure is described and the parameters for finding the optimal model are presented. The results from the training step are now used to detect bicycle parking facilities. For this purpose, the raster data from 2019 as well as from 2017 are used to evaluate the results. These two steps can be summarized as the deep learning approach. They are executed iteratively to improve the model also on the basis of detected results. Finally, the postprocessing steps are described, in which the detected objects are further processed and summarized in an ArcGIS tool.

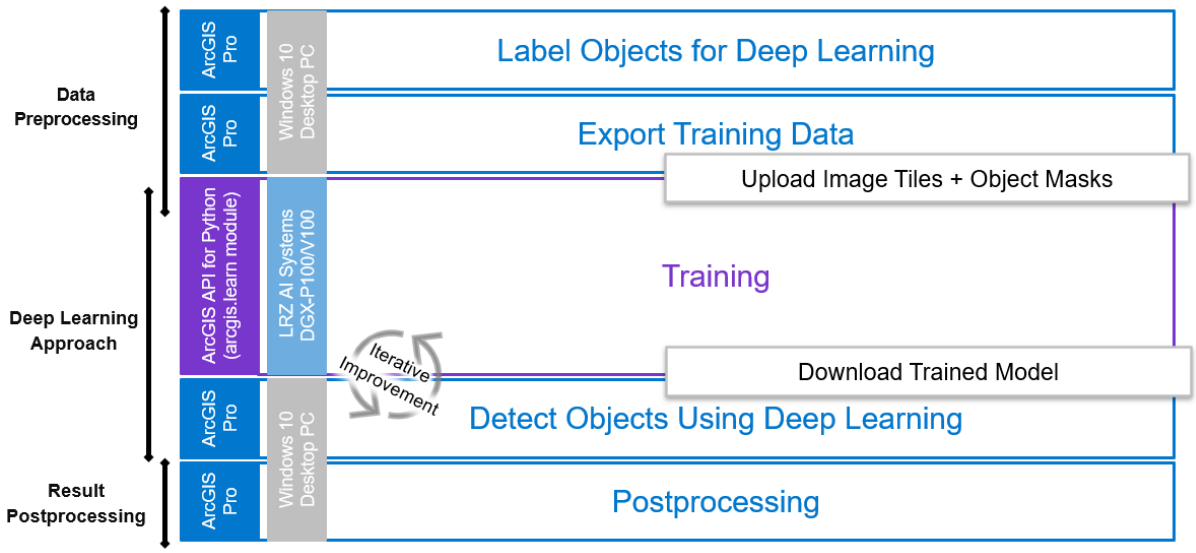


Figure 32: Overview processing steps

4.2 Hardware

Choosing the right hardware is crucial for training CNNs and its duration. GPUs (graphics processing unit) are the most important hardware for deep learning, which allow parallel processing on a high level. The supercomputing centre operated by the Bavarian Academy of Sciences and Humanities (Leibniz-Rechenzentrum (LRZ)) offers students to rent an NVIDIA DGX-1 GPU system with 8 Tesla P100 GPUs or 8 Tesla V100 GPUs (16 GB memory per GPU and 512 GB DDR4 of memory per node, 80 CPU Cores per node) (Leibniz-Rechenzentrum 2021c). The LRZ AI Systems are allocated to the LRZ Linux Cluster and can be remotely accessed inside the Münchner Wissenschaftsnetz (Munich Scientific Network, MWN) via Secure Shell (SSH). For this, a Linux Cluster account in combination with a separate application for the LRZ AI Systems is necessary (Leibniz-Rechenzentrum 2021a). The GPUs can be booked via a scheduler on the system (for more details see 4.6.1).

The software ArcGIS Pro (see 4.3.1) runs on common Windows computers. The amount of image data leads to slow processing in terms of exporting image tiles and detecting objects.

4.3 Software

The software required for the different process steps is presented below. The detailed application description of the software components follows in the next chapters.

4.3.1 ArcGIS Pro

ArcGIS Pro (ESRI Inc.) is a geographic information system software provided by ESRI and the current version is 2.8. As it is the standard GIS-software at the City of Munich, the final toolset shall be executable with this software. For working with aerial images and deep learning, the *Image Analyst* extension is necessary. It provides tools to prepare the training data by supporting labeling and export, the possibility to train the model in- or outside of ArcGIS Pro, and tools to use the trained model (ESRI 2021f). It is therefore a valuable tool to realize this project: it will be used to preprocess the data (labeling and tiling) and steps after training (applying the model / detecting the objects, some of the testing and creating the final toolbox).

4.3.2 Software and Libraries for the Training

As the GPUs run in a Linux environment, the desktop application ArcGIS Pro cannot be used for training the deep learning models. ESRI offers the **ArcGIS API for Python** together with the **arcgis.learn module** that has the same deep learning tools as the desktop software, based on Python (Python Software Foundation) coding. The arcgis.learn module provides functions for data augmentation and for training deep learning architectures. It is based on the deep learning libraries PyTorch and fast.ai and, if desired, also TensorFlow elements. **Jupyter**

Notebook (Project Jupyter) is chosen as the Python programming environment. Advantages of using Jupyter Notebook are the ability to run the code step by step and visualizing graphs and images directly when the code is executed.

PyTorch and **TensorFlow** are both open-source machine learning frameworks. PyTorch is based on the machine learning library Torch. The Facebook's AI Research team released its first version in 2016. **fast.ai**, also open-source, is built on top of PyTorch and provides high-level component libraries for deep learning. TensorFlow was released by the Google Brain team in 2015. **TensorBoard** is the visualization toolkit of TensorFlow (TensorFlow 2021) and helps with visualizing the loss functions of the different models.

4.4 Preparation of the Training Data

4.4.1 Labeling

The foundation for the training of a neural network is the dataset with ground truth data – the training dataset. In this case, it means that the exact location and the extent of each bike parking facility need to be determined in the reference aerial image and documented accordingly together with information on the rack type (=class). ArcGIS Pro provides the Classification Tool *Label Objects for Deep Learning* in the Image Analyst extension. With this tool the labeling of the bike parking facilities can be performed.

The underlying raster is the LUFTBILD2019_RGB image. Step by step every line feature of the vector dataset Fahrradstaender_bis_2019.shp is examined and, when recognizable, a polygon of a certain class is manually drawn around the object according to the rack type.

4.4.2 Image Characteristics and Labeling Limitations

During the labeling process, imagery characteristics and limitations for the labeling and possible challenges for the training and detection processes came to the fore.

First of all, there are bike racks that are not existing at the time the image was taken. This can be due to dismantlement, temporary removal due to construction sites or because the facility was built after the remote sensing flight took place. Others might be displaced due to transformations in the streetscape. Figure 33a) shows an excerpt of LUFTBILD2019_RGB where bike parking facilities are removed due to a construction site. Figure 33b) shows the same area with the redesigned station forecourt in a Google Maps (2021) image.

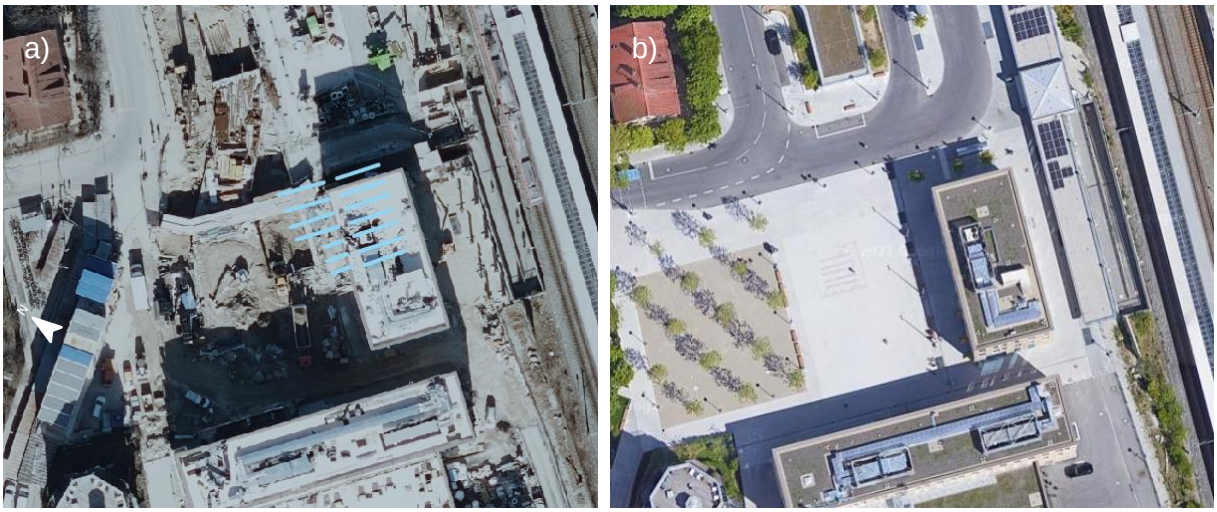


Figure 33: Construction site: S-Bahnhof Allach
a) Detail from LUFTBILD2019_RGB: Oertelplatz with bike parking facilities (blue) not existing anymore due to building and construction site (Source: Landeshauptstadt München 2020)
b) Situation after construction was finished, displaced: now Double Decker racks in building with solar panels on the right (image March 2020) (Source: Google Maps 2021)

Secondly, some are not visible as they are roofed parking facilities or as they are located in underpasses (Figure 34). In these two cases, the facilities can obviously not be labeled and are therefore not content of the training dataset. The covered bike racks are anyhow outside of the scope of this work because they are not detectable as bike racks. In addition, they often have a more fixed structural location anyway and are not that often subject to changes.

Figure 34: Detail from LUFTBILD2019_RGB: U-/S-Bahnhof Neuperlach Süd: on the left: bike parking facility under the bridge and in the upper right roofed bike parking facilities and therefore not labeled (Image source: Landeshauptstadt München 2020)

In addition, it sticks out that some of the line features in the dataset are slightly dislocated and sometimes also shorter or longer than the imagery shows (Figure 21, Figure 34, Figure 36). This requires a close look and prudent labeling.

Furthermore, shadows make it also hard to recognize bike racks in certain areas. Shadows (in images from 2019: casted from the southeast to southwest) are clearly visible on the images: on the one hand, there are shadows that are clearly cast by a bicycle and thus indicate a parked bike (see Figure 35); on the other hand, the system may be overfitted concerning paying too much attention to the shadows. In addition, in areas of shadowing, for example through buildings, there is a risk of racks being not recognizable by the human eye (see Figure 36a). Shadows are therefore important and to be considered when evaluating the results and when training the network. They are a well-known challenge in remote sensing (e.g., Dare (2005)).

Figure 35: Detail from LUFTBILD2019_RGB: U-Bahnhof Richard-Strauss-Str.: Nordkopf, S-Aufgang: Shadows clearly cast by bikes locked to inverted-Us (Source: Landeshauptstadt München 2020)

The aerial photographs show that some bicycle racks are placed under trees. However, they are often recognizable in leafless imagery, but not always due to for example dense branches (see Figure 36b). Detection in leafy images is unprofitable as the treetops cover them often entirely.

Figure 36: Detail from LUFTBILD2019_RGB: Rotkreuzplatz: shadows and trees

a) orange: bike racks in shadow, line-geometry not on exact position; red: bikes cast shadow overlayed by shadow of a tree; green: tree is covering part of the facility

b) LUFTBILD2019_RGB overlayed with LUFTBILD2019_nDOM: shows where branches of trees cover bike racks on the ground

(Image source: Landeshauptstadt München 2020)

4.4.3 Result

The Fahrradstaender_bis_2019.shp dataset contains 1,999 objects. Of these, 536 objects are not labeled due to the reasons mentioned above. Figure 37 gives the reasons and shows that 42% (=228) of the 536 objects not labeled are roofed facilities.

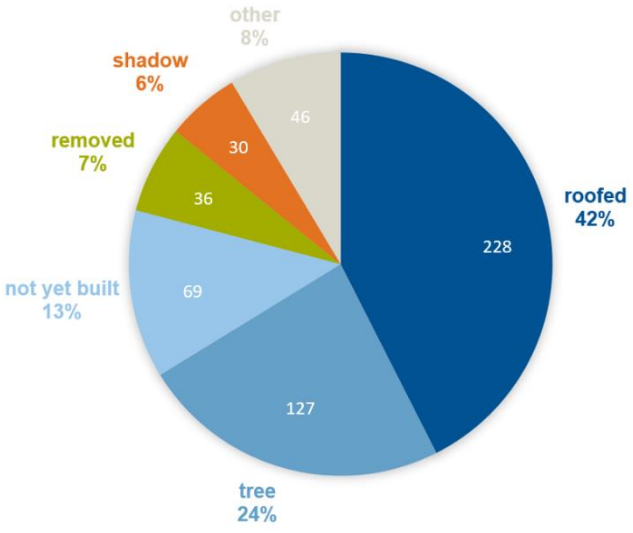


Figure 37: Reasons for bike parking facilities not being recognizable & therefore not being labeled and the corresponding proportions with reference to LUFTBILD2019_RGB (total: 536 facilities not labeled)

Although LUFTBILD2019_RGB was taken at a time with no foliage, in 127 cases trees still cover the bike racks. 105 facilities did not exist at that time

and 30 objects are not recognizable on the imagery due to black shadows and no reflections or similar irregularities. In addition, 135 MVG Rad stations in the city area could be labeled.

In total, 1,463 objects (=71.8%) are recognizable and therefore labeled. As some consist of more than one line, this results in 1,839 polygons and labels, respectively, for the training dataset. Table 4 shows the number of objects of each class together with the number of labels on the left. As ten rack types have less than 50 labels representing their class, rack types that are similar according to their appearance from above are grouped so that each class has at least 80 representatives to train on. Table 4 shows the new rack type groups on the right.

Rack Type	Classvalue	# objects	# Polygons/ Labels
Anlehne neu	**10**	**422**	**442**
Anlehne alt	**11**	**109**	**129**
A. geschwungen	12	1	3
L15G E T	**20**	**718**	**614**
L15G E H/T	21	38	10
L15G D T	**22**	**175**	**149**
L15G D H/T	23	35	7
L15 E T (schräg)	**24**	**107**	**82**
Arreta E	30	12	9
Kappa E H/T	41	13	2
Kappa D H/T	43	11	4
Kappa (schräg)	44	1	1
Kappa E H/T DB	46	4	0
Kappa D H/T DB	48	2	0
Kappa (schräg) DB	49	2	0
Klemmbügel	**50**	**274**	**225**
BCS	60	12	15
RGT	70	7	8
Doppelstockparker E	80	12	0
Doppelstockparker D	81	19	0
MVG	**99**	**137**	**135**
Unknown	100	25	4

Rack Type	Classvalues	# Polygons/ Labels	Sum
Anlehne neu	**10**	442	**442**
Anlehne alt	**11**	129	**132**
A. geschwungen	12	3	
L15G E T	**20**	614	**635**
L15G E H/T	21	10	
Arreta E	30	9	
Kappa E H/T	41	2	
L15G D T	**22**	149	**160**
L15G D H/T	23	7	
Kappa D H/T	43	4	
L15 E T (schräg)	**24**	82	**83**
Kappa (schräg)	44	1	
Klemmbügel	**50**	225	**252**
BCS	60	15	
RGT	70	8	
Unknown	100	4	
MVG	**99**	135	**135**

Table 4: Number of labels for the training dataset according to their rack type / class and grouping of classes; left: number of objects in Fahrradstaender_bis_2019.shp and the number of labels (=polygons) for each rack type (=class); right: groups of classes; (E: one-side/einseitig; D: double-sided/doppelseitig; T: low/tief; H/T: high-low/hoch-tief; DB: Deutsche Bahn)

4.5 Export of the Training Data

Having the objects labeled, it is now time to export the image tiles and the corresponding labels. The export takes place in the second tab of the *Label Objects for Deep Learning* tool. In this thesis, it makes sense to use image tiles of 256 px x 256 px or 512 px x 512 px (px: pixel): with a resolution of 8 cm a ground section of 20.48 m x 20.48 m and 40.96 m x 40.96 m, respectively, is displayed in one tile and thus bike parking facilities are not too fragmented, but still take up a reasonable area of the cutout. By adding an overlap with a stride of 50% of the tile size, both horizontally and vertically, the training dataset is enlarged artificially to extent the generality of the model.[2]

The image tiles are saved in the TIFF format as this format supports more than three bands contrary to PNG and JPEG format. The masks for Mask R-CNN are binary images of the same size as the image tiles with one band (256 px x 265 px x 1 and 512 px x 512 px x 1, respectively). A mask corresponds to one class and sets the pixel values of areas with an object – here bike parking facilities – to 1 while the value of the background pixels stays 0. Figure 38 shows an image tile with its corresponding mask.

Figure 38: left: image tile 256 px x 256 px (Feldmoching), right: corresponding mask with three objects (Image source: Landeshauptstadt München 2020)

Tiles that have at least one bike parking facility included are exported. This results for the whole area of Munich in about 7500 image tiles for 256 px x 256 px export and in about 4900 image tiles for 512 px x 512 px export without additional rotation.

[2] A further way to enlarge the dataset artificially would be to add rotated image tiles. By setting the *Rotation Angle* parameter in the tool's settings to 315°, the number of image tiles is doubled as each tile is rotated by -45° and added to the dataset. The reason for rotating by 315° is the shadow cast that needs to fit the positions of the sun in Munich: the sun never shines from the north in Munich. Rotating by 315° results in shadows that fall into the range of west to north. One single rotation by +45° is not possible with this tool as it adds all rotations that are multiples of 45° up 360° to the dataset. Unfortunately, ArcGIS Pro could not execute this export and aborted repeatedly with undefined error. Therefore, the rotation was implemented as part of the data augmentation (see chapter 4.6.2).

The output of the tool, when exporting for Mask R-CNN, includes six elements:

- images folder: includes all exported image files (.tif), together with the corresponding world files (.tfw) for georeferencing
- labels folder: includes folders named after the different classes and each of them contains the corresponding binary masks (.tif) together with their world files (.tfw) and their small auxiliary xml-files (.aux.xml)
- map.txt: text file that maps all image file-paths to the corresponding mask(s) by listing the file-paths of the mask TIFFs
- stats.txt: text file that contains statistics on the number of images, their size, and how many images belong to each class together with information on min, mean and max size of the features on the image
- esri_accumulated_stats.json: file that contains statistics on the image tiles and classes (e.g. number of features per class, number of features per image, feature area per class, band statistics) in JSON format
- esri_model_definition.emd: incomplete Esri model definition file (JSON format) that provides the user with a framework to describe the trained model; it already contains information about the exported images and the classes settled by the labeling and this export step

The different rasters and composites of 2019 are exported as training data to train the models and to compare their performance and characteristics. As a result, we now have the following exported image tiles for 256 px x 256 px and 512 px x 512 px, respectively. In addition, every raster is exported with labels of all classes, of the grouped classes (see Table 4) and with all labels assigned to the same class.

- LUFTBILD2019_RGB
- LUFTBILD2019_CIR
- Composite: LUFTBILD2019_RGB + LUFTBILD2019_nDSM
- Composite: LUFTBILD2019_RGB + LUFTBILD2019_NDVI
- Composite: LUFTBILD2019_RGB + LUFTBILD2019_nDSM + LUFTBILD2019_NDVI

The size of an entire output folder is for 256 px x 256 px with three bands about 1.1 GB, with four bands about 3.5 GB and with five bands about 5.3 GB. The output with grouped classes has around 40,000 files. The size of one folder of the 512 px x 512 px image tiles is with three bands 2.9 GB and with four bands 9.1 GB. With this image size, the output with grouped

classes has around 27,500 files. It takes between 0.5 and 48 hours to export the image tiles for the whole city, depending on the image size and the computer performance. It can be accelerated enormously by using a mask for reducing the area where the tool is checking for image tiles to export.

4.6 Training

As already described, the training phase is not carried out in ArcGIS Pro but on the LRZ AI Systems in a Python environment. This chapter deals with both the soft- and hardware-specific processes and the procedure for finding the best-performing model.

4.6.1 Usage of LRZ AI Systems

The LRZ provides NVIDIA GPUs for deep learning purposes. Currently, the LRZ AI Systems include a *DGX-1 P100 Architecture* with 8 NVIDIA Tesla P100 GPUs and a *DGX-1 V100 Architecture* with 8 NVIDIA Tesla V100 GPUs (Leibniz-Rechenzentrum 2021c). As mentioned above (chapter 4.2), a Linux Cluster account and approval for the LRZ AI Systems are required to access the GPUs.

The remote access is realized with an SSH-Connection to datalab2.srv.lrz.de (xxyyzz: alias for Linux Cluster account ID) onto the Ubuntu system (connection to MWN, e.g., via VPN needed):

```
ssh -L 8888:localhost:8888 -Y datalab2.srv.lrz.de -l xxyyzzz
```

Files can be transferred via the Globus Research data management portal (University of Chicago, Argonne National Laboratory) into a personalized $HOME folder on a DSS (Data Storage Service) container accessible on the LRZ AI Systems. The default disk quota is 150 GB and 200,000 inodes. It can be extended upon request (Leibniz-Rechenzentrum 2021b). The global path on datalab2.srv.lrz.de to the uploaded data is `/home/xxyyzzz/` and navigation is done by the standard Linux commands.

The installation of custom software stacks is realized within Enroot containers. The Enroot container framework by NVIDIA works analogous to Docker software. A container is defined by and created out of a container image. Enroot supports images provided by the Nvidia NGC Cloud as well as from the Docker Hub. To use Jupyter Notebooks, the foundation is a miniconda image – a minimal installer for conda with Debian as operating system – from the Docker Hub. This container is modified by installing the ArcGIS API for Python and saved as a new enroot image (`my_arcgis_container.sqsh`). The exact versions of the installations are important to avoid errors in the installation process and in the execution later on. (Anaconda Inc. and Docker Inc. 2021; ESRI 2021g))

```
enroot import docker://dockeruser@continuumio/miniconda3
enroot create continuumio+miniconda3.sqsh
enroot start continuumio+miniconda3

conda install python=3.7.7
conda install -c esri arcgis
conda install -c esri -c fastai -c pytorch arcgis=1.8.2 scikit-
  image=0.15.0 pillow=6.2.2 libtiff=4.0.10 fastai=1.0.60 pytorch=1.4.0
  torchvision=0.5.0 tensorflow-gpu=2.1.0 --no-pin
conda install -c esri gdal=2.3.3
```

The container also needs the driver for the NVIDIA-GPU. Therefore, the installation of CUDA (as of today: version 11.4) is necessary (NVIDIA Developer 2021)[3]:

```
apt-get install gnupg
apt-get --allow-releaseinfo-change update
apt-get install software-properties-common
apt-key adv --fetch-keys
  https://developer.download.nvidia.com/compute/cuda/repos/debian10/x86
  _64/7fa2af80.pub
add-apt-repository "deb
  https://developer.download.nvidia.com/compute/cuda/repos/debian10/x86
  _64/ /"
add-apt-repository contrib
apt-get update
apt-get -y install cuda
```

Finally, the container can be exported to a new Enroot image (ending: .sqsh) to create new containers with the desired software components ready for usage:

```
enroot export --output my_arcgis_container.sqsh continuumio+miniconda3
enroot create --name my_arcgis_container_1 my_arcgis_container.sqsh
```

SLURM (Simple Linux Utility for Resource Management, SchedMD LLC) is the system for scheduling the jobs. The command `sinfo` informs on the available hardware resources and some specifications, `squeue` reports the jobs' statuses. The allocation of resources is performed with `salloc` and options on desired partition, number of parallel tasks, number of gpus and the maximum of needed time:

```
salloc -p dgx-1-v100 --ntasks=1 --gres=gpu:1 --time=0-12:30:00
```

or

```
salloc -p dgx-1-p100 --ntasks=1 --gres=gpu:1 --time=0-12:30:00
```

Subsequently `srun` is used to submit the job. The following command executes a terminal within the allocated machine:

```
srun --pty bash
```

[3] be sure to have started the container in root mode: `enroot start –root container_name`

Now, the container, if not already built, can be created, and then started:

```
enroot create --name my_arcgis_container_1 my_arcgis_container.sqsh
enroot start my_arcgis_container_1
```

The Jupyter Notebook is started with

```
jupyter notebook --ip=0.0.0.0 --allow-root --port=8889
```

and a second terminal is used to set up further port-forwarding to be able to access the notebook structure via the browser (`localhost:8889`):

V100: `ssh -L 8889:dgx-002:8889 xxyyzzz@datalab2.srv.lrz.de`
or:
P100: `ssh -L 8889:dgx-001:8889 xxyyzzz@datalab2.srv.lrz.de`

The Jupyter Notebook usage is cancelled with Ctrl+C, the container, the allocation, and the ssh-connection with exit. A job can be deleted with `scancel` *`jobid`*.

In order to take advantage of TensorBoard visualizations that show graphs of the training loss and the validation loss already during the training but also afterwards for comparing different runs, the extension has to be installed:

```
conda install tensorboard=2.2.1
conda install -c conda-forge tensorboardx=2.1
pip install tensorboardX
```

Due to security concerns the usage of Jupyter Notebooks was deprecated by the LRZ in August 2021. Therefore, the less convenient way of starting Python-scripts was used to run the code.

4.6.2 Preparation

Writing the actual python code for executing the training is the next step. It starts with importing the required libraries:

```
% import of pathlib for getting home-directory path
from pathlib import Path

% import of required arcgis_learn functionalities
from arcgis.learn import prepare_data, MaskRCNN

% import of fastai and torch libraries
import fastai
import torch
```

Before starting the actual training, the data has to be split into training and validation data. The arcgis.learn module provides the `prepare_data`-function for data preparation. It not only splits the data but also offers data augmentation and the possibility to set parameters to customize it. The only required argument is the path to the directory, where the exported data

is saved. The arguments of this function are described in Table 5, which gives a brief explanation of the arguments and describes what and how parameters are set for the preparation of the bike-parking-facilities-dataset.

Argument	Brief description	Remark relating this work
path	path to data directory	required path: home/xxyyzzz/UploadedFolder
val_split_pct	percentage of validation data, default: 10%	subject of testing: 10%, 20%, 30%
batch_size	batch size, default: 64	subject of testing: 4,8,32, (64: OutOfMemory-Error)
seed	random seed for reproducible train-validation split	optional integer, set to make different trainings easier to compare
collate_fn	function to collate data at PyTorch	default
transforms	fast.ai transforms for data augmentation	subject of testing: default (ESRI has set good defaults that work well for aerial imagery) & other transformations
chip_size	training images cropping size, default: 224	default
working_dir	path where folder including trained model is saved	optional, when None: new folder 'model' in data directory
class_mapping	dictionary for mapping id label to string label	not needed
dataset_type	type of images and labels, when map.txt is missing	not applicable, as map.txt is existing
resize_to	size for resizing images	not applicable for RCNN_Masks
imagery_type	'sentinel', …, 'ms' (any other type of image)	optional
bands	bands used to export data, e.g. ['r','g','b','nir']	optional
rgb_bands	indices of red, green, blue bands	optional
extract_bands	indices of bands for training	not needed, as all exported bands used for training
norm_pct	percentage of training data for statistics for normalizing, default: 30%	default
downsample_factor	downsampling factor, default: 4	default

Argument	Brief description	Remark relating this work
encoding	encoding to read csv/json-file	not applicable for RCNN_Masks
min_points classes_of_interest extra_features remap_classes background_classes	*arguments for PointCloud preparation*	not applicable

Table 5: Arguments of the prepare_data-function (black: required parameter, blue: parameter for which various values are tested to improve the training, green: optional parameters, sometimes set, or parameters where default option is optimal) (adapted from (ESRI 2021a))

The `transforms` parameter is responsible for the data augmentation of the input image data, which can help to achieve better training results and prevent overfitting. It is implemented with the help of fast.ai. ESRI offers default values that are already well adapted to satellite data. The data augmentation is a tool to counteract overfitting, so it is worth taking a closer look at the default values. In addition, further transformations can be considered.

ESRI implements the transformations for Mask R-CNN as follows (ESRI 2021b):

```
    chip_size = 224
[...]
    ranges = (0, 1)
[...]
        train_tfms = [
            crop(size=chip_size, p=1., row_pct=ranges,
              col_pct=ranges),
            dihedral_affine(),
            brightness(change=(0.4, 0.6)),
            contrast(scale=(1.0, 1.5)),
            rand_zoom(scale=(1.0, 1.2))
        ]
[...]
    val_tfms = [crop(size=chip_size, p=1., row_pct=0.5,
                col_pct=0.5)]
    transforms = (train_tfms, val_tfms)
```

First, all images (`p=1.0`) are cropped to the size of 224 to have the correct size for the FPN. For this, a random part of the image is selected (`range: (0,1)`). In addition, the images are rotated by multiples of 90° (`dihedral_affine()`). The brightness is changed in the range of 0.4 to 0.6, so it is slightly increased or slightly decreased, while the contrast is increased by random values between 1.0 and 1.5. Figure 39 shows how brightness and contrast changes affect an image. Finally, a random zoom (`rand_zoom`) is applied to the image. The image will have a zoom level between 100% and 120%.

Figure 39: Brightness (top) and contrast (bottom) changes
(Source: fast.ai 2021)

Other possible transformations that are not yet applied by the default settings, but provided by fast.ai (2021), are `flip_lr` and `rotate`. `flip_lr` flips an image horizontally. Specifying `p=0.5` causes 50% of the images to be flipped. `rotate` rotates an image by the specified value or a value within a specified range of values. A percentage can be specified here as well. Typical properties of aerial images such as shadow directions must be considered when rotating.

An example of calling the `prepare_data`-function with using the default transformations is:

```
home = Path.home()
data_path = Path(home,'myOutputFolderName')
print(data_path)

data = prepare_data(data_path, batch_size=8, imagery_type='ms',
  rgb_bands = [0,1,2], val_split_pct=0.2, seed=174413)
```

The next step is initializing the desired model by calling the class constructor. This includes defining the backbone model together with arguments for detailed model modification (Table 6). For example:

```
model = MaskRCNN(data, backbone='resnet101')
```

Argument	**Brief description**	**Remark relating this work**
`data`	fast.ai Databunch	data-object returned from prepare_data function
`backbone`	compatible backbones: resnet18, resnet34, resnet50, resnet101, resnet152 default: resnet50	subject of testing: resnet34, resnet50, resnet101
`pretrained_path`	path to pretrained model	pretrained model not available
`pointrend`	boolean for indicating if PointRend used, default: False	default, not needed as shape of bike parking facilities is rectangular
	further arguments for RPN and classification-head	default

Table 6: Arguments of the MaskRCNN class (black: required parameter, blue: parameter for which various values are tested to improve the training, green: optional parameters, sometimes set, or parameters where default option is optimal); adapted from (ESRI 2021a)

In addition, the arcgis.learn module provides a learning rate finder (`lr_find()`) to determine the initial learning rate:

```
lr = model.lr_find()
```

4.6.3 Training of the Models

The next step is the actual training with the `fit`-function. Here, the number of epochs must be set. When it is unknown, how many epochs the training will need to perform best, the early stopping argument can be used by setting a high number of epochs but at the same time let the function stop as soon either the loss or the training function do not improve anymore. Table 7 includes all arguments for the function.

When training the model, only the weights of the Mask R-CNN specific layers and not of the backbone are updated. This is called freezed model. The backbone was pretrained before and runs now with the weights found for the previously trained model. In the case of using a ResNet backbone in ArcGIS, the backbone was pretrained on large image sets like ImageNet. To also update the weights in the backbone layers, the `unfreeze`-function can be called previously. It has no arguments.

```
model.unfreeze()
model.fit(epochs=30, lr=lr, early_stopping=True, checkpoint='all',
   tensorboard=True)
```

Argument	Brief description	Remark relating this work
`epochs`	number of epochs	required, subject of testing, depending on training performance
`lr`	learning rate used for training	float returned from learning rate finder
`one_cycle`	boolean for usage of 1cycle learning rate schedule, default: True	default, as small corrections of the learning rate support optimization
`early_stopping`	boolean for stopping when monitor-value does not increase for 5 epochs, default: False	optional
`checkpoint`	boolean/String for saving checkpoints, False: no checkpoints, True: best model based on monitor-value saved, all: all checkpoints saved	optional / all
`tensorboard`	boolean for writing training log for visualization with TensorBoard, default: False	set to True
`monitor`	parameter for early stopping & checkpoints, default: valid_loss (else: train_loss)	optional

Table 7: Arguments of the fit-function (blue: parameter for which various values are tested to improve the training, green: optional parameters, sometimes set, or parameters where default option is optimal); adapted from (ESRI 2021a)

During the training, the python output provides information about the current epoch with the number and percentage of batches processed, as well as the training and validation loss at

the end of each epoch. In addition, an estimate of the duration of the current epoch is displayed, as well as the time actually required afterwards.

The trained model is then saved in the desired storage location that was set as `working_dir` when preparing the data. The function `save` itself only needs the parameter for naming the model and the folder, respectively.

```
model.save('ModelFolderName')
```

This function offers more parameters to set, but apart from the model folder name all are not applicable in this case.

4.6.4 Improving the Model

The output of training and validation loss for each epoch during training give a first impression on how the model performs. Calling the following functions give additional graphical feedback:

```
model.plot_losses()
model.show_results(mode='bbox_mask', rows=7)
```

The `plot_losses`-function plots the two graphs of training and validation loss in one figure. The `show_results`-function prints ground truth image tiles next to the corresponding image tiles with the predicted object(s). Here, `rows=7` indicates that seven image pairs are output. These two functions only work in the Jupyter Notebook environment. But the graphics are also saved in the model metrics file in the model folder and can be opened there.

Further, the average precision score can be printed with:

```
model.average_precision_score()
```

It returns the average precision on the validation set for each class. The numbers can also be found in the model metrics file.

4.6.5 Output – Model Definition

In addition to the model metrics file, there is also the following data stored in the model folder (ESRI 2021f):

- *name_model*.dlpk: deep learning model package that contains an emd-file and a trained model file, so all files and data required to do object detection
- *name_model*.emd: Esri model definition file: JSON file including information on the framework used (e.g., PyTorch), on the trained model and on the training images (optional) as well as paths to related files for describing the trained deep learning model
- *name_model*.pth: PyTorch file with saved model

- model_metrics.html: document including fundamental information on the trained model: backbone, learning rate, training and validation loss plot, average precision score for all classes, and sample results (four image pairs of ground truth and predictions)
- ArcGISInstanceDetector.py: instance segmentation python raster function to inference an arcgis.learn deep learning model
- ModelCharacteristics: folder including the images loss_graphs.png (plot of training and validation loss functions) and show_results.png (four image pairs of ground truth and predictions)

Globus helps with downloading the files. Now the trained model is ready for object detection back in ArcGIS Pro.

4.7 Experiments for Validation of the Model Parameters

With this knowledge presented of losses and average precision scores, adjustments and improvements can be applied: different parameter setting combinations are tested and evaluated to get closer to the best model. The parameters to be changed are image size, backbone, batch size, percentage of validation data, data augmentation and the number of epochs.

Different values are tried for one value, while the others remain the same. This gives a feeling for the best parameter configuration. It starts with default settings for backbone, percentage of validation data and data augmentation. This gives the batch size. Then different backbones and validation percentages can be tried out. If a good value for a parameter is found, it is useful to alter and check the previously found parameters again to see if they are still a good choice. In this process, the number of epochs can at first be chosen to be large to see when the losses approach their minima. Afterwards, this can be adjusted in such a way that no overfitting occurs.

In the meantime, the models can be tested in ArcGIS Pro with the *Detect Objects Using Deep Learning* tool (see chapter 4.8) to assess the performance based on the results there. This results in an interative process.

The following values were determined as the best model parameter combination:

- image size: 256 px x 256 px
- batch size: 8
- backbone: ResNet-50
- validation: 20%
- data augmentation: rotation: in the range between -30° and 30° for 50% of image tiles
  left-right flip: 50% of images
  no 90° rotations
  crop, brightness, contrast and zoom: standard settings
- unfreeze
- epochs: 100

From the settings for the data augmentation, the following code for the transformations results:

```
train_tfms = [rotate(degrees=(-30,30), p=0.5), flip_lr(p=0.5),
  crop(size=chip_size, p=1., row_pct=ranges, col_pct=ranges),
                  brightness(change=(0.4, 0.6)),
                  contrast(scale=(1.0, 1.5)),
                  rand_zoom(scale=(1.0, 1.2))]
val_tfms = [crop(size=chip_size, p=1., row_pct=0.5, col_pct=0.5)]
tfms = (train_tfms, val_tfms)
```

The entire Python code for building and training the model with the arcgis.learn module is given in Appendix B.

The justification of the choice is now given by the following training and validation loss plots based on the RGB image tiles. For this purpose, the selected model is compared with models where one parameter is changed at a time and where the other parameters remain those of the best model. The losses of the model with the described setting is shown in blue in the following plots. The loss values were exported from the TensorBoard log-files as .csv and plotted using Matlab (The MathWorks, Inc.). The results of the parameter comparisons can be transferred to the other band combinations, since they react very similarly to the parameter values.

Image Size

A comparison of the two image crop sizes of 256 px x 256 px (blue) and 512 px x 512 px (orange) is shown in Figure 40. The 256 px x 256 px image size results in lower loss values. The difference between the two losses is also lower.

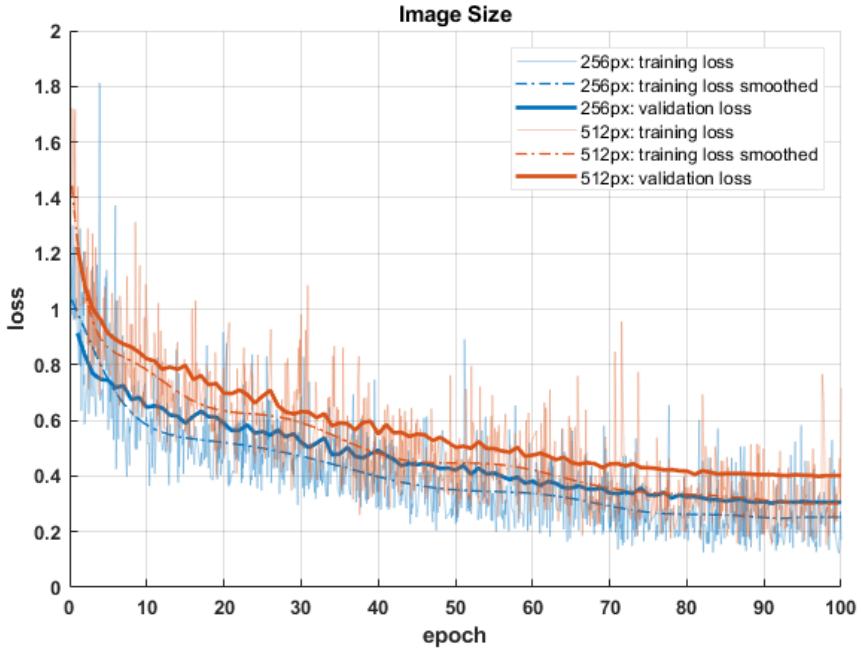


Figure 40: Training and validation loss for different image sizes: blue: 256 px x 256 px, orange: 512 px x 512 px; training losses smoothed by 12th degree polynomial for better comparison

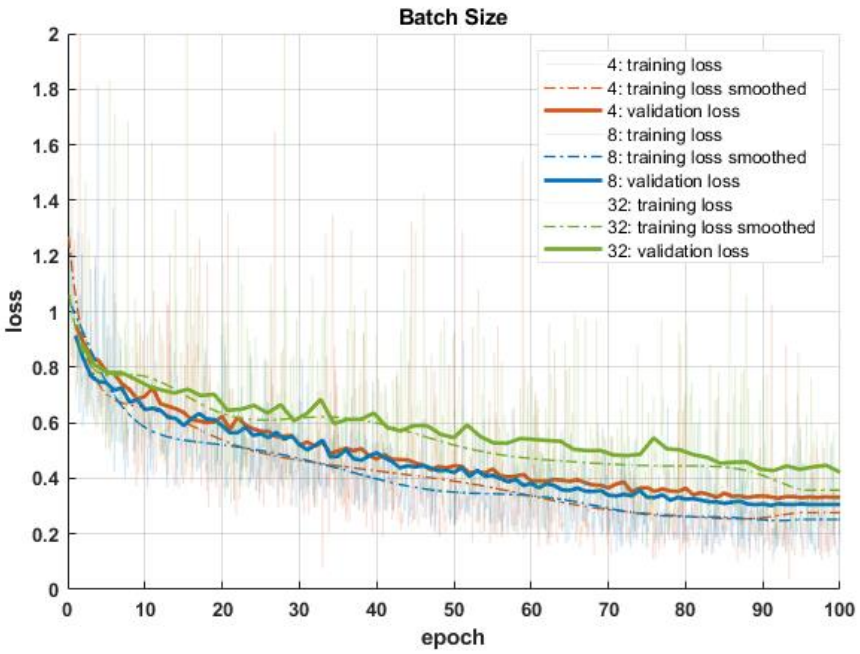


Figure 41: Training and validation loss for different batch sizes: orange: 4, blue: 8, green: 32; training losses smoothed by 12th degree polynomial for better comparison

Batch Size

The batch size depends on the performance of the GPU. Figure 41 shows the losses of the models with 4, 8 and 32 images per batch. The training of a model with batch size 64 fails at the available hardware. A batch size of 8 turns out to be the best choice.

Backbone

ResNet-50 is the default backbone in ArcGIS Pro. Using ResNets with fewer or more layers is represented in Figure 42 by ResNet-34 (orange) and ResNet-101 (green), respectively. It can be seen that ResNet-34 does not lead to such a high level of fitting as the other two backbones. Using ResNet-101 does not lead to any improvement. The validation loss is even a little worse than ResNet-50 (blue). In addition, it takes more time for each epoch as more parameters need to be updated. Therefore, the default backbone setting is kept here.

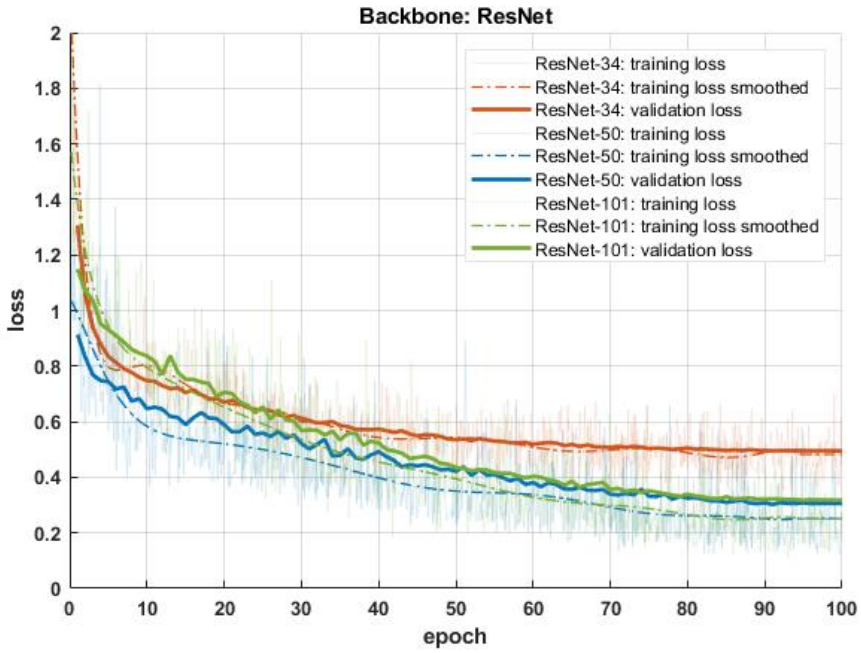


Figure 42: Training and validation loss for different backbones: orange: ResNet-34, blue: ResNet-50, green: ResNet-101; training losses smoothed by 12th degree polynomial for better comparison

Validation

Changing the validation set percentage results in the losses shown in Figure 43. The default setting of 10% (orange) yields similarly good results as 20% (blue). Using 30% (green) of the data as validation set results in a higher validation loss and a higher difference between validation and trainining loss. Here, as less data is available for training, the generalization is not as good. For a stable validation and since a ratio of 80:20 is a common value, 20% is chosen as parameter.

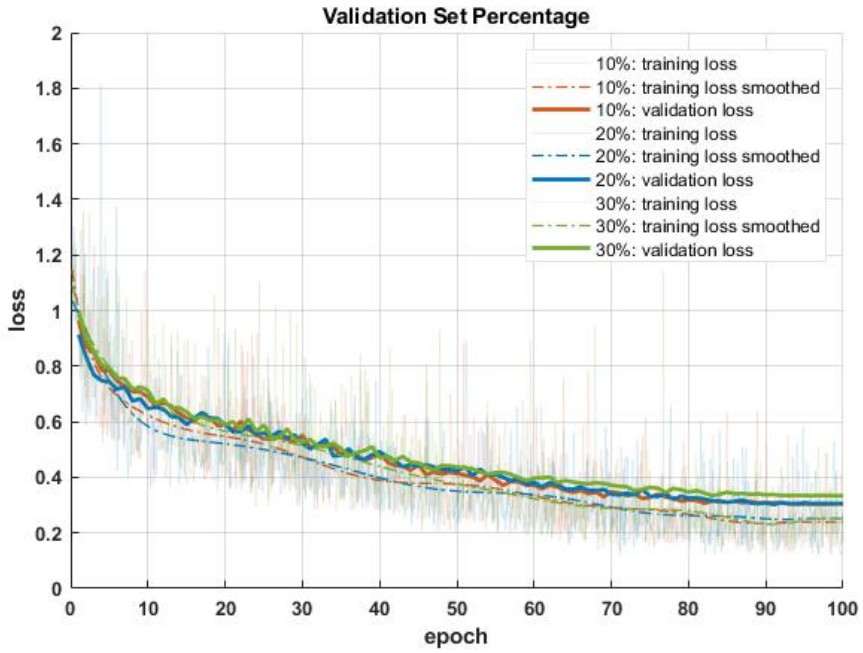


Figure 43: Training and validation loss for different validation set percentages: orange: 10%, blue: 20%, green: 30%; training losses smoothed by 12th degree polynomial for better comparison

Data Augmentation

Figure 44 compares four different data augmentations. While the standard setting (orange) turns into an overfitting at 23 epochs at the latest, the other models improve further. The additional rotation of 30% of the images in a range of -10° to 10° (green) prevents overfitting, but still includes rotations by multiples of 90°. Replacing the 90° rotations with a wider contrast change (0.8 to 1.5) (yellow) reduces the loss. However, rotating 50% of the images in a range from -30° to 30° and an additional left-right flipping of 50% of the images without additional 90° rotations (blue) performs best. Here the contrast change remains at the default setting in the range between 1.0 and 1.5. Here, the validation loss is identical to the third alternative, but the difference between training and validation loss is smaller.

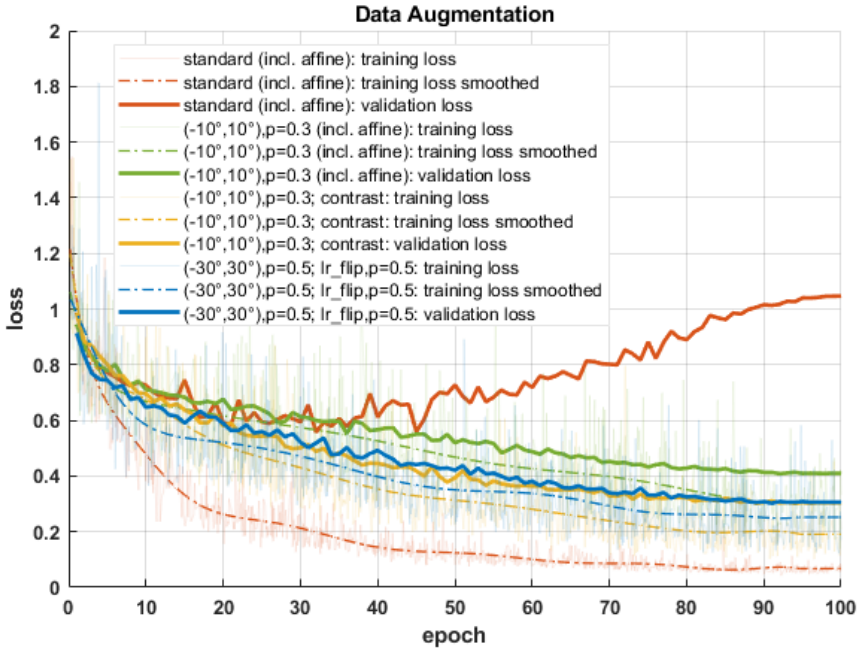


Figure 44: Training and validation loss for different data augmentations: orange: default, green: rotations +-10°, yellow: ±10°, contrast, blue: ±30°, left-right flips; training losses smoothed by 12th degree polynomial for better comparison

Freeze/Unfreeze

For the sake of completeness, a comparison of the freezed (orange) and unfreezed (blue) backbone is shown in Figure 45. The unfreezed model starts with the same pretrained weights in the backbone, but adjusts them further during training, while freezing only improves the layers of the heads. This confirms the assumption that an unfreezed model leads to slightly better results.

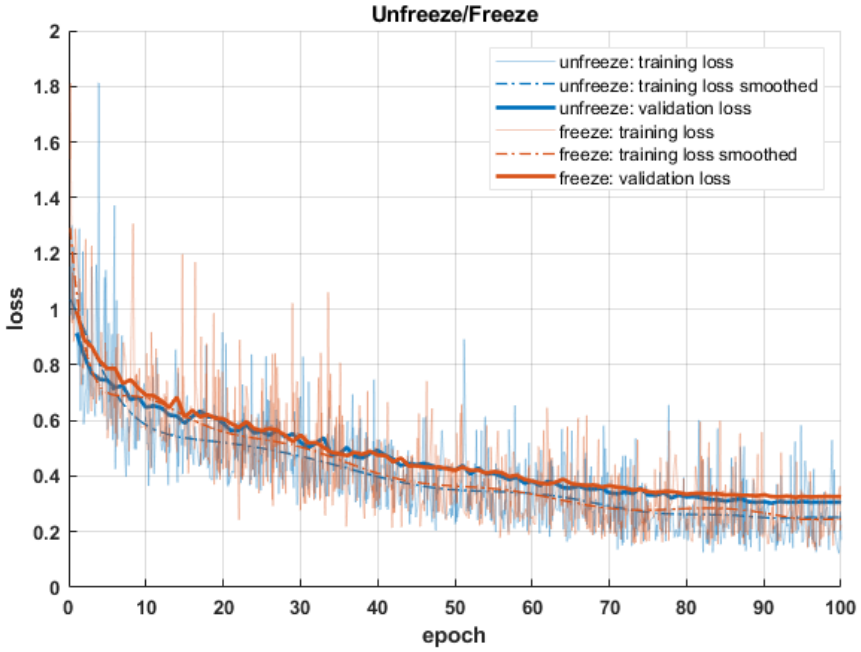


Figure 45: Training and validation loss for freezing/unfreezing the backbone: blue: unfreezed backbone, orange: freezed backbone; training losses smoothed by 12th degree polynomial for better comparison

Epochs

When the number of epochs is about 100, the values for training and validation loss remain the same and will no longer improve. Therefore, the training can stop here, and the number of epochs can be set to 100.

Finally, it has to be decided whether training is done over all classes, the grouped classes or without classes. A comparison of the loss curves (Figure 46) shows that they do not differ considerably. In addition, average precision scores can be used for comparison. The following Table 8 (as an extension of Table 4) shows the average precision scores for the training on 256 px x 256 px RGB images. Since small or overfitting values result for the classes that are only represented by very few labels, grouping is preferable. The grouped classes give good values; therefore this option is used. After all, it is also about the detection of the bicycle rack type. An exact evaluation of the classification is done when looking at the object detections.

Classvalue	# objects	# Polygons/ Labels	Average Precision Score
10	**422**	**442**	0.8063
11	**109**	**129**	0.7618
12	1	3	0.5
20	**718**	**614**	0.8642
21	38	10	1
22	**175**	**149**	0.8017
23	35	7	0.8333
24	**107**	**82**	0.7772
30	12	9	0.7308
41	13	2	1
43	11	4	0.75
44	1	1	1
46	4	0	
48	2	0	
49	2	0	
50	**274**	**225**	0.7862
60	12	15	0.7749
70	7	8	1
80	12	0	
81	19	0	
99	**137**	**135**	0.8222
100	25	4	0.5833

Classvalues	# Polygons/ Labels	Sum	Average Precision Score
10	442	**442**	0.7963
11	129	**132**	0.7318
12	3		
20	614	**635**	0.8278
21	10		
30	9		
41	2		
22	149	**160**	0.8322
23	7		
43	4		
24	82	**83**	0.7872
44	1		
50	225	**252**	0.7310
60	15		
70	8		
100	4		
99	135	**135**	0.7926

Classvalues	# Polygons/ Labels	Average Precision Score
all	1839	0.8346

Table 8: Average Precision Scores for training on RGB images; left: average precision score together with number of objects and labels for each rack type (=class); middle: grouped classes with number of labels and average precision score; right: average precision score when all 1839 labels belong to the same class

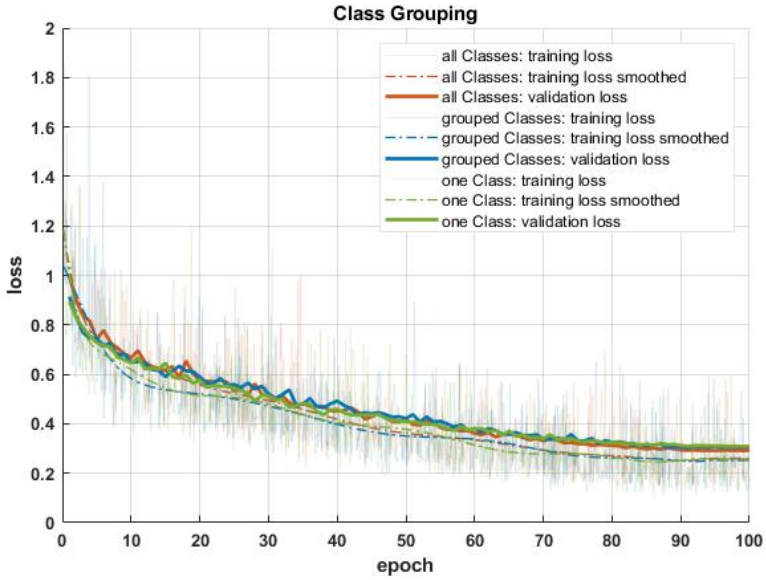


Figure 46: Training and validation loss for different class groupings: orange: all classes, no grouping, blue: grouped as in Table 4, green: one class; training losses smoothed by 12th degree polynomial for better comparison

4.8 Detect Objects Using Deep Learning

The object detection in ArcGIS Pro is performed using the *Detect Objects Using Deep Learning* tool. The tool needs the .emd or the .dlpk path of the trained model for the *Model Definition*. The *Input Raster* must be set to the same type of band combination that was used for training. In addition, a name for the output-shp has to be defined. The arguments will be filled in by the tool, when the .emd-file or .dlpk-file is loaded. On the *Environments* tab of the tool, the *Processing Extent* can be defined, to delimit the area in which objects are to be detected with the trained model.

At the position where an object was detected, it is framed with the segmentation mask and by this its extension and position are displayed (Figure 47).

Figure 47: Example of detection masks (Orleansplatz) (Image source: Landeshauptstadt München 2020)

For the evaluation of the detection, a dataset is required, on which no training has been performed, but for which ground truth data is available. Therefore, four areas were taken out of the training dataset. This concerns the area between Rosenheimer Platz and Ostbahnhof (Figure 48a)), Rotkreuzplatz (Figure 48b)), the area around the U- and S-Bahn station Harras (Figure 48c)), and the U-Bahn station Messestadt West (Figure 48d)). These areas were chosen to represent different environments: while Harras and Rotkreuplatz are busy squares in the city, the Messestadt is a bit outside with comparatively newly established bicycle parking facilities and the area near the Ostbahnhof is characterized by multi-story buildings and thus shadows and backyards. In terms of shadows, three of the areas can be categorized as follows: while the area around the Ostbahnhof is characterized by shadows from buildings, the

Rotkreuzplatz area has many trees that cast shadows, and the Harras area has most bicycle racks in the sun. The four areas contain a total of 128 labeled bike parking facilities (7% of all labeled facilities). The size of the study areas is limited because, on the one hand, the training dataset should not be reduced too much, since it is already small, and as, on the other hand, it takes a long time[4] to detect the bike racks.

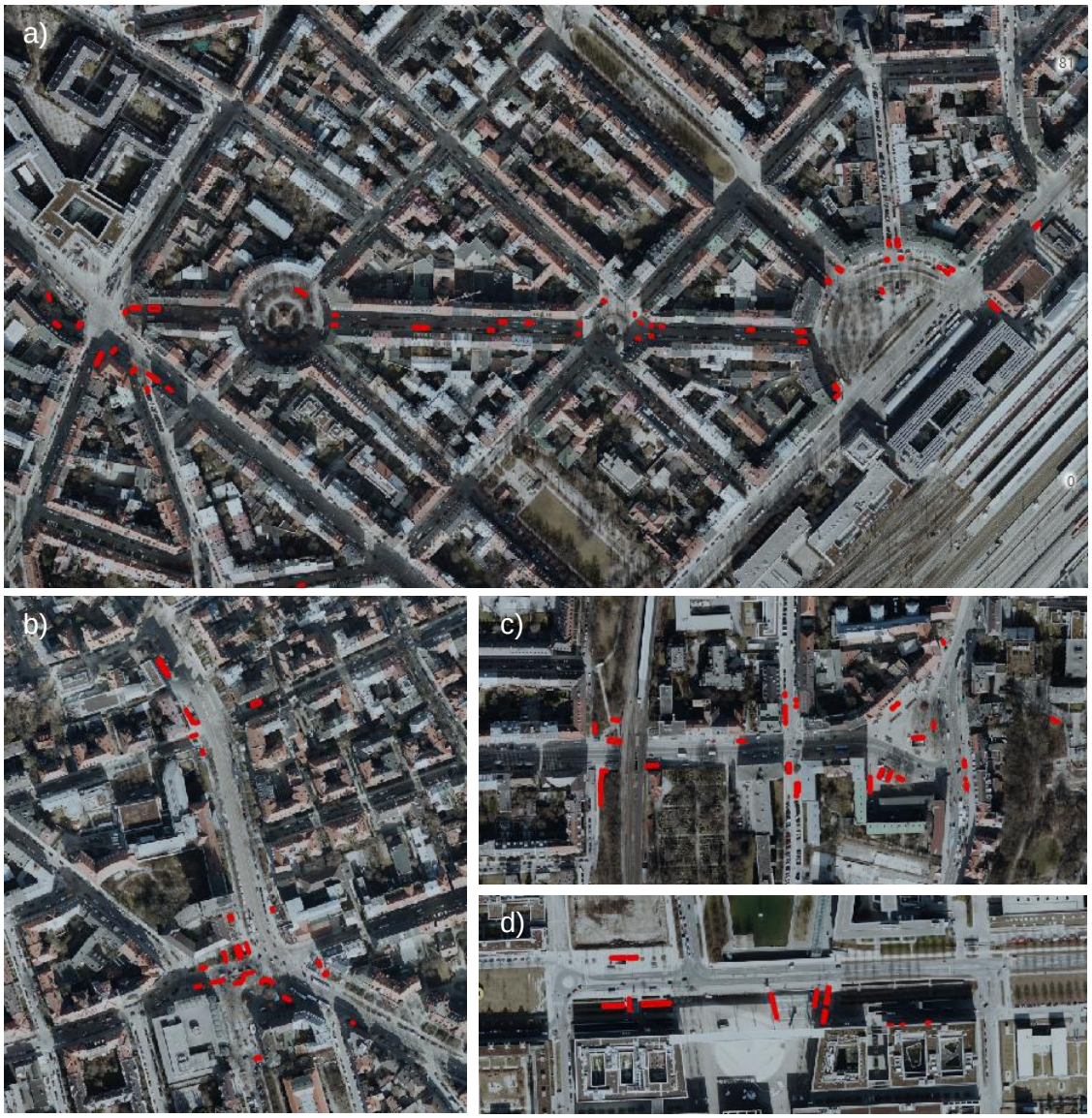


Figure 48: Areas taken out of the training dataset for evaluation; labeled bike parking facilities in red; a) Rosenheimer Platz to Orleansplatz/Ostbahnhof (50 labels); b) Rotkreuzplatz (31 labels); c) Harras (30 labels); d) Messestadt West (17 labels) (Image source: Landeshauptstadt München 2020)

Each of the five reduced image and label datasets is trained with the best performing parameters (see 4.7). The result of *Object Detection Using Deep Learning* tool is then analyzed for each band combination and for each of these four areas. Table 9 shows an example of such an analysis. All detected objects are looked at individually and accordingly recorded in

[4] ca. 12 hours for all four areas for one band combination

the table. If a bicycle parking facility can be seen at the location of the object mask in the aerial image, it is classified as a TP. If this facility is labeled, a distinction is made whether the classification is correct. A distinction is also made between bicycle racks on private property, single bikes, for example locked to lamp posts, and bicycle racks in public spaces. If an object is detected wrongly, it is described and assigned to the FP accordingly. In addition, in order to detect FN, all labeled bicycle racks in the corresponding area are examined.

	RGB	**Ostbhf.**	**Rotkreuz-pl.**	**Harras**	**Messe**	**sum**	**% of detected**
TP + TP_other	**correctly detected**	**87**	**46**	**22**	**15**	**170**	**86.3%**
	private	47	19	4	1	71	36.0%
	correctly classified	25	17	14	11	67	34.0%
	wrongly classified	5	8	4	3	20	10.2%
	single bikes detected	9	2			11	5.6%
	unlabeled & public	1				1	0.5%
FP	**wrongly detected**	**13**	**4**	**8**	**2**	**27**	**13.7%**
	roof	3		2	1	6	3.0%
	groundclutter	5	2		1	8	4.1%
	cemetery			4		4	2.0%
	car		1	1		2	1.0%
	bush, tree, grass	4	1	1		6	3.0%
	shadow					0	0.0%
	rectangular object					0	0.0%
	motorbike, etc.	1				1	0.5%
sum	**detected**	**100**	**50**	**30**	**17**	**197**	**100.0%**

							% of to detect
	to detect	50	31	30	17	128	
TP	**detected & labeled**	**30**	**25**	**18**	**14**	**87**	**68.0%**
	correctly classified	25	17	14	11	67	52.3%
	wrongly classified	5	8	4	3	20	15.6%
FN	**labeled & not detected**	**20**	**6**	**12**	**3**	**41**	**32.0%**

Table 9: Detailed breakdown of TP, FP, and FN for detection in RGB for all areas of study

From this recall, precision, FDR and FNR are calculated. Afterwards, the true positives are tested to what extent the detected masks correspond to the labeled rectangles. Here IoU supports. The false positives are examined on the basis of examples. The confidence value returned by the *Detect Objects Using Deep Learning* tool helps here. Subsequently, an analysis of the false negative bicycle racks takes place. Reasons, why these were not detected, are discussed.

4.9 Postprocessing

When a suitable deep learning model has been found for the detection of the bicycle racks, it can be used to detect the objects in the aerial imagery. Afterwards, the postprocessing can be performed on the output polygon features. The steps are summarized in an ArcGIS model to perform the entire workflow at once. The individual processing steps are concatenated with the Model Builder of ArcGIS Pro. Since recognition is a computationally intensive and long task, it is performed beforehand, and the result is then used as input for the model.

Figure 50 shows the individual steps in the Model Builder. The detected objects from the *Detect Objects Using Deep Learning* tool and a polygon feature class serve as input. This second input covers the areas that are appropriate for the installation of bicycle racks by the LHM. Here, the lhm_oeffentliche_flaechen_v2.shp file described in chapter 3.4 serves as an example. With these two input feature classes and the *Pairwise Intersect* tool, all detected bicycle racks that are located on private property can be filtered out. If no *Non Maximum Suppression* was applied during the detection, the next step is to use the same to ensure that only one object is retained if more than 60% of two or more objects overlap. Then, a minimum bounding rectangle is placed around each of the objects. This new geometry represents more accurately the intended effective area of bicycle parking facilities. From this, the *Polygon to Centerline* tool calculates lines. Line objects are used, so far, at the LHM for the representation of bike parking facilities in GIS (Fahrradstaender_bis_2019.shp). Since fields/attributes are lost in this processing step, the values for class and confidence are added to the line features with *Join Field*. Since the workflow contains fields that are not required in the output feature classes, additional *Delete Field* steps are added. To be able to name the output of the rectangular objects by the user, the intermediate step *Rename* is inserted. This results in two output feature classes: rectangles that represent the area of the detected bicycle racks and the associated line features.

Figure 49 shows the input mask for the user. The user hands over the two required feature classes in the first two fields and then assigns names to the two output feature classes.

The created workflow can be used for new aerial images in the future to postprocess detected bike racks. Chapter 5.3 shows the resulting rectangles and lines.

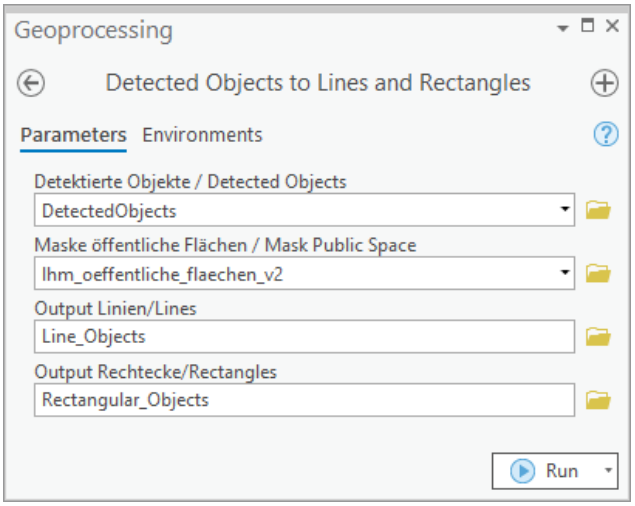


Figure 49: Input mask for Detected Objects to Lines and Rectangles tool

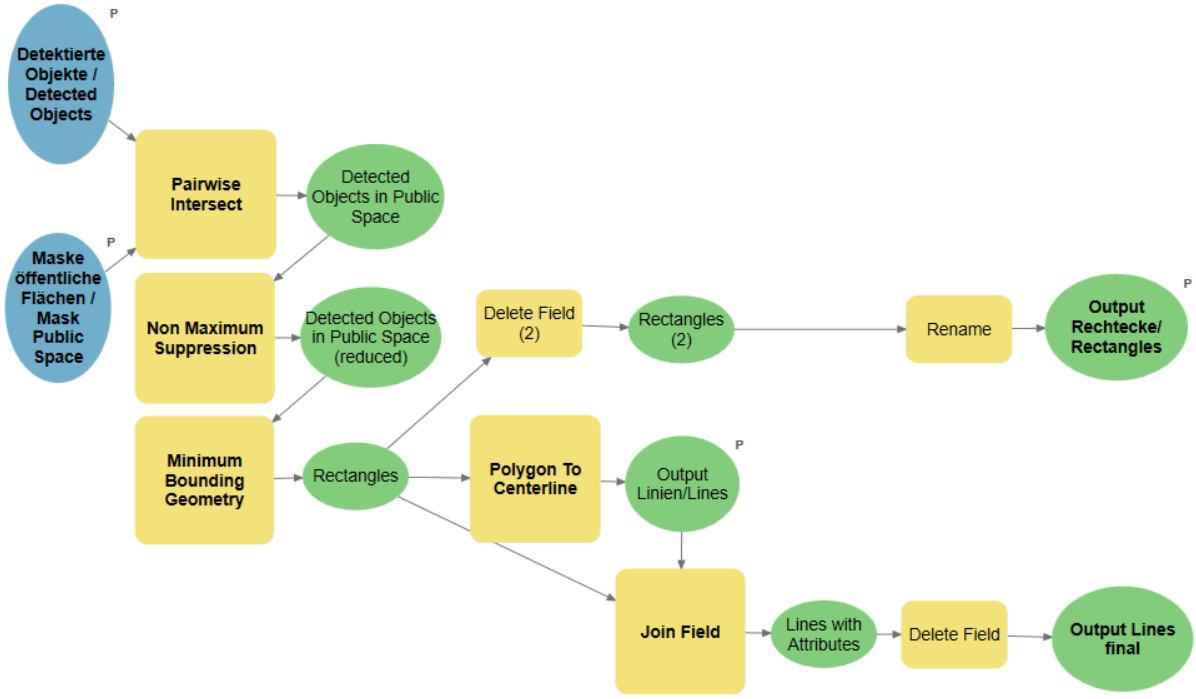


Figure 50: Postprocessing steps in Model Builder: blue: input feature classes; yellow: calculations; green: intermediate results/final results

5 Results

One model is trained for each band composition. The advantages and disadvantages of using additional datasets (CIR, nDSM, NDVI) need to be analyzed and statistically evaluated. The trained models are applied to different areas of the available imagery.

5.1 Performance Evaluation

The analysis of the detected objects, which was performed as shown in Table 9, results for all band combinations in the following Table 10:

		RGB	% of detected	CIR	% of detected	+nDSM	% of detected	+NDVI	% of detected	+nDSM +NDVI	% of detected
$TP + TP_{other}$	**correctly detected**	**170**	**86.3%**	**187**	**82.4%**	**188**	**90.4%**	**174**	**89.7%**	**166**	**85.1%**
	private	71	36.0%	78	34.4%	69	33.2%	70	36.1%	71	36.4%
	correctly classified	67	34.0%	79	34.8%	80	38.5%	77	39.7%	75	38.5%
	wrongly classified	20	10.2%	17	7.5%	19	9.1%	16	8.2%	17	8.7%
	single bikes	11	5.6%	10	4.4%	17	8.2%	6	3.1%	1	0.5%
	unlabeled & public	1	0.5%	3	1.3%	3	1.4%	5	2.6%	2	1.0%
FP	**wrongly detected**	**27**	**13.7%**	**40**	**17.6%**	**20**	**9.6%**	**20**	**10.3%**	**29**	**14.9%**
	roof	6	3.0%	5	2.2%	0	0.0%	1	0.5%	2	1.0%
	groundclutter	8	4.1%	7	3.1%	4	2.4%	3	1.5%	8	4.1%
	cemetery	4	2.0%	3	1.3%	0	0.0%	2	1.0%	3	1.5%
	car	2	1.0%	2	0.9%	0	0.0%	0	0.0%	4	2.1%
	bush, tree, grass	6	3.0%	9	4.0%	8	3.8%	4	2.1%	8	4.1%
	shadow	0	0.0%	4	1.8%	0	0.0%	2	1.0%	1	0.5%
	rectangular object	0	0.0%	3	1.3%	2	1.0%	4	2.1%	1	0.5%
	motorbike, etc.	1	0.5%	7	3.1%	5	2.4%	4	2.1%	2	1.0%
Σ	**detected**	**197**	**100.0%**	**227**	**100.0%**	**208**	**100.0%**	**194**	**100.0%**	**195**	**100.0%**

			% of to detect		% of to detect		% of to detect		% of to detect		% of to detect
	to detect	128		128		128		128		128	
TP	detected & labeled	**87**	**68.0%**	**96**	**75.0%**	**99**	**77.3%**	**93**	**72.7%**	**92**	**71.9%**
	correctly classified	67	52.3%	79	61.7%	80	62.5%	77	60.2%	75	58.6%
	wrongly classified	20	15.6%	17	13.3%	19	14.8%	16	12.5%	17	13.3%
FN	labeled & not detected	**41**	**32.0%**	**32**	**25.0%**	**29**	**22.7%**	**35**	**27.3%**	**36**	**28.1%**

Table 10: Detailed breakdown of TP, FP, and FN for detection in all band combinations summarized over all areas of study

The bold percentages in Table 10 give precision, FDR, recall and FNR. For a better overview and comparison, Table 11 presents these values once again. Here, a distinction is made between two TP values. On the one hand TP, which contains the true positives that are also contained in the ground truth labels (correctly classified and wrongly classified), and, on the other hand, $TP_2 = TP + TP_{other}$. Here TP_{other} are all additional correctly identified bike racks and bikes (private, single bikes, unlabeled & public). In this case, no recall and FDR percentages can be calculated as no FN values are available for additional parking facilities.

		RGB	CIR	+nDSM	+NDVI	+nDSM +NDVI
Precision$_2$	$\frac{TP_2}{TP_2 + FP}$	86.3%	82.4%	90.4%	89.7%	85.1%
FDR$_2$	$\frac{FP}{TP_2 + FP}$	13.7%	17.6%	9.6%	10.3%	14.9%
Recall	$\frac{TP}{TP + FN}$	68.0%	75.0%	77.3%	72.7%	71.9%
FNR	$\frac{FN}{FN + TP}$	32.0%	25.0%	22.7%	27.3%	28.1%
Precision	$\frac{TP}{TP + FP}$	76.3%	70.6%	83.2%	82.3%	76.0%
FDR	$\frac{FP}{TP + FP}$	23.7%	29.4%	16.8%	17.7%	24.0%

Table 11: Precision, FDR, Recall and FNR for all band combinations

If TP_{other} is included, RGB+nDSM has the best results with a precision of 90.4%. CIR performs worst with 82.4%. If the TP_{other} is left out, the recall values range between 68.0% (RGB) and 77.3% (RGB+nDSM). Precision reaches values between 70.6% (CIR) and 83.2% (RGB+nDSM). In all cases, RGB+nDSM achieves the best values for the detection of bike parking facilities (recall: 77.3%, precision: 83.2%). When it comes to the proportion of TP compared to all detected objects (precision), CIR (70.6%) still performs worst. When considering the recall, it is RGB (68.0%) being worst. RGB+NDVI is closest to RGB+nDSM in terms of precision. For recall, it is CIR. This shows that RGB+nDSM yields the best results.

5.1.1 IoU for True Positives

Looking at the correctly detected masks, it is noticeable that they cover the bicycle parking facilities very well. In most cases, they are slightly smaller than the labeled rectangle, because the masks have round corners. Now and then, they are a little wider or longer. Figure 51 shows five examples with the red ground truth label in the background. Above it, the detected objects are in color. The area of the detections is usually very similar to that of the label. On the right,

two object masks differ: the model trained with RGB+NDVI (green) detects only about half of the facility while the one trained with RGB+nDSM+NDVI (sand) includes extra space.

Figure 51: Masks of detected bicycle parking facilities at Franziskaner Str./Rosenheimer Str. to visualize IoU: red: ground truth label; pink: RGB; purple: CIR; blue:RGB+nDSM; green: RGB+NDVI; sand: RGB+nDSM+NDVI (Image source: Landeshauptstadt München 2020)

For all band combinations, IoU values around 80% are obtained for the detected LHM bike parking facilities together with the corresponding labels. In terms of IoU, there are no significant differences recognizable to determine the best band combination.

5.1.2 Analysis of False Positives

When detecting objects, only those are output whose confidence value lies above a certain threshold. This is set to 0.9, which means that only objects that are detected with a probability of correctness of over 90% are stored in the output feature class. Nevertheless, false positives are detected, as already shown in Table 9 and Table 10. Table 10 shows that, depending on the band combination, between 9.6% (RGB+nDSM) and 17.6% (CIR) of the detections are wrongly detected. The reasons for the errors differ. Figure 52 shows examples of wrongly detected objects. Reasons for false detection are unsteady roof structures (Figure 52a)). This type of false detection is filtered out with adding nDSM to the RGB bands. Furthermore, cluttered background is a common source of error. This also includes the imagery of a cemetery, as seen in Figure 52c). For RGB, CIR and RGB+nDSM+NDVI band combinations, dark cars on dark backgrounds are also a problem (Figure 52d)). Shadow patterns (Figure 52e)) and bushes (Figure 52f)) as well as dense branches can also be misdetected.

Concerning vegetation, all band combinations return erroneous object masks. Even those that are stronger correlated with the vegetation as CIR and NDVI. Figure 52g) does not originate from the observed areas but is still worth mentioning. The structure of the track bed is very similar to the arrangement of bikes in a parking facility and is thus misinterpreted just like the platform roof here. Overall, the combination of RGB + nDSM performs best in terms of false positive detections since no roofs are detected and cars and the cemetery in Sendling do not cause any misdetections.

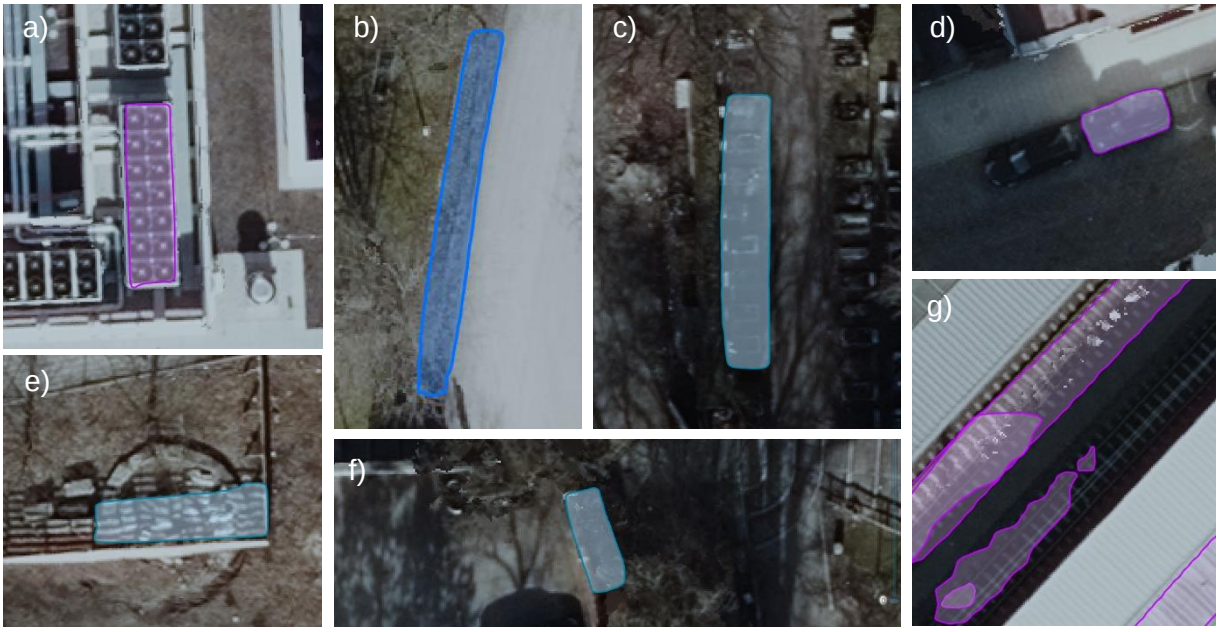


Figure 52: Examples for false positive detections: a) roof (Willy-Brandt-Platz, with RGB); b) groundclutter (Preysingstr., with RGB+nDSM); c) cemetery (Friedhof Sendling, with CIR); d) dark car (Volkartstr., with RGB); e) shadow pattern (Kidlerplatz, with CIR); f) bush (Kidlerplatz, with CIR); e) tracks and roof (Ostbahnhof, with RGB) (Image source: Landeshauptstadt München 2020)

To see how well the *Detect Objects Using Deep Learning* tool itself evaluates the detection, the confidence values can be viewed for each object. As described above, these are between 90% and 100%. Figure 53 shows the frequency of these in histograms for TPs that are correctly classified (a)), for TPs to which the wrong class is assigned (b)), TPs that are on private property and not in the ground truth data (c)), and the confidences for the FPs (d)). For this, objects of all band combinations and of all observation areas are summarized. The histogram of the correctly classified objects shows that these often have a high level of confidence and that the tool classifies them as reliable. This trend can also be observed in the case of correctly detected objects with incorrect classification, but not to the same extent. In the case of private bicycle racks, the trend is also noticeable, although to an even lower extent. In contrast, incorrectly detected objects have an even distribution over all confidence values. It concludes that objects with higher confidence are more likely to belong to the group of true positives. In this case, the value keeps its promise. Nevertheless, false positive detections can still be found in all value ranges. However, the threshold value of 0.9 is already a good filter for false detections.

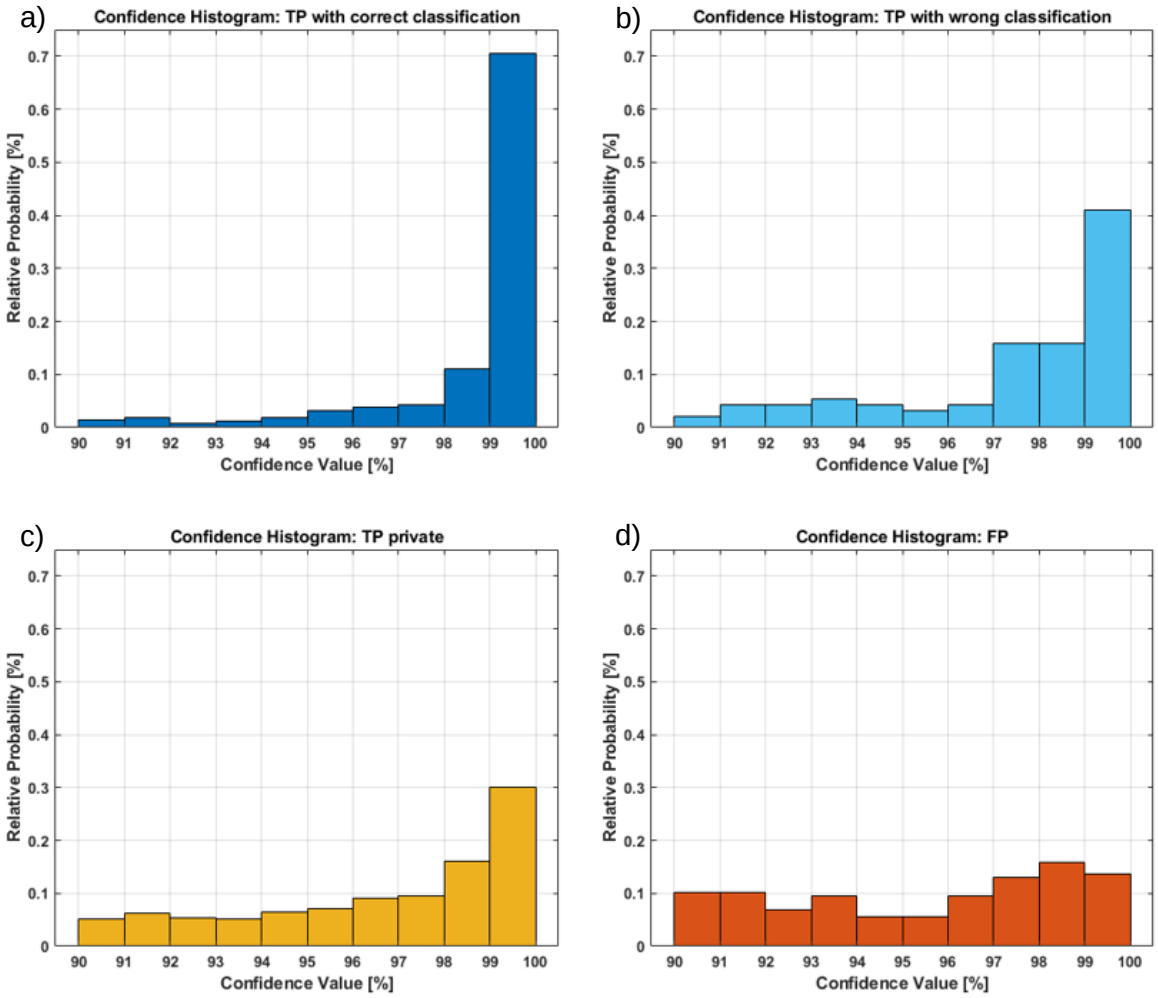


Figure 53: Histograms for confidence value distributions; a) TPs that are correctly classified; b) TPs that are wrongly classified; c) TPs on private areas and not in ground truth data; d) false positive detections

5.1.3 Analysis of Reasons for False Negatives

RGB+nDSM has an FN to FN+TP ratio of 22.7% and thus performs best. RGB has the worst result with 32% non-detected bike racks. The reasons for a lack of detection are diverse. False negatives occur when there are not enough training examples for a particular pixel pattern. In the following some examples are given. Occlusion of the parking facility as in Figure 54a) is one reason. In this case, the branches of the tree are too dense, and the structure of the parked bicycles cannot be seen. No band combination was able to detect this facility. In example Figure 54b) the background is very dark due to shadows cast by the building and the bike racks cannot stand out from it. So, also here, none of the band combinations helps to detect this facility. The problem in Figure 54c) might not be apparent to the human observer. It is probably due to the number of bicycles that were not properly parked. Therefore, the trained network (RGB) cannot recognize this as bike racks. In Figure 54d) the bike parking facility is difficult to recognize. There are no bikes parked and the shadows cast by the trees make detection more difficult. Only RGB+nDSM was able to detect parts of the racks. Figure 54e)

shows a facility in which only one bicycle is parked. Again, it was not possible for all band combinations (only RGB+nDSM) to recognize them as a facility.

Rotkreuzplatz, characterized by many tree shadows, and Ostbahnhof area, characterized by building shadows, have the lowest FNRs. This can be explained by the fact that about 74% of the labeled bike racks are mostly shadowed (37% cast by trees, 37% cast by buildings). Only 18% of the ground truth data are not affected by shadows from the surrounding area. In 8% of the cases only small shadows by lamp posts, street signs or similar objects are visible.

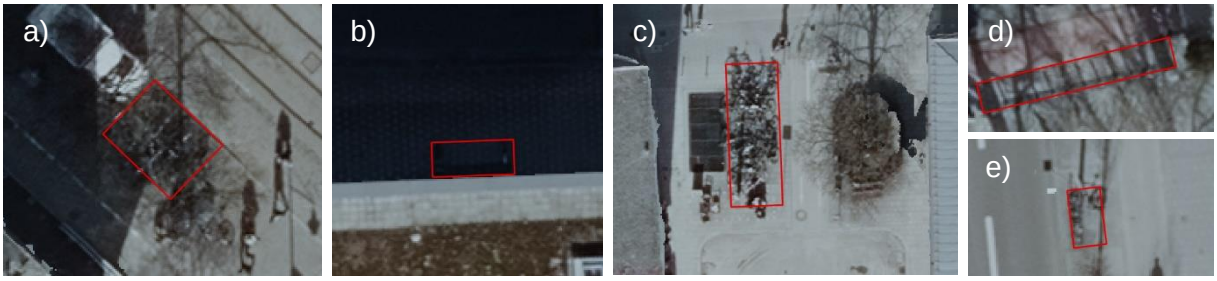


Figure 54: Examples for false negative detections (label in red); a) dense branches (Orleansplatz); b) dark shadow (Willy-Brandt-Allee); c) high number of bikes (Albert-Roßhaupter-Str.); d) tree shadow and no parked bikes (Rotkreuzplatz); e) (almost) empty bike parking facility (Plinganserstr.) (Image source: Landeshauptstadt München 2020)

5.1.4 Private Parking Facilities and Single Bikes

Due to the choice of the four study areas, many of the detected and at the same time private bike racks are in completely dark, shaded backyards and only identifiable as such at second glance (Figure 55a)). Again, this type of shade is not included sufficiently in the training data as bike racks that were not identifiable due to dark shadows are not labeled. Figure 55b) shows a clearly visible private parking facility without shadows. Overall, many private facilities are detected, but the individual is not detected by all band combinations. In addition, some are only detected by one band combination. If detection of private bike racks were desired, examples would have to be added to the training data to achieve even better results.

The reason, why single bicycles, e.g., locked to street signs (Figure 55c)) or other street furniture (Figure 55d)), are detected, is that the training data also contains single inverted-Us, which offer space for up to two bicycles. The similarity is therefore very high. In specific cases, bicycle collections in public spaces are also detected. These are listed as unlabeled & public. Either a bicycle parking facility of the LHM was detected here, which is not included in the database (Figure 55e)) and the labeled objects, respectively, or it is a facility of a business. It may also be a loose collection of bikes at the time of the image flight.

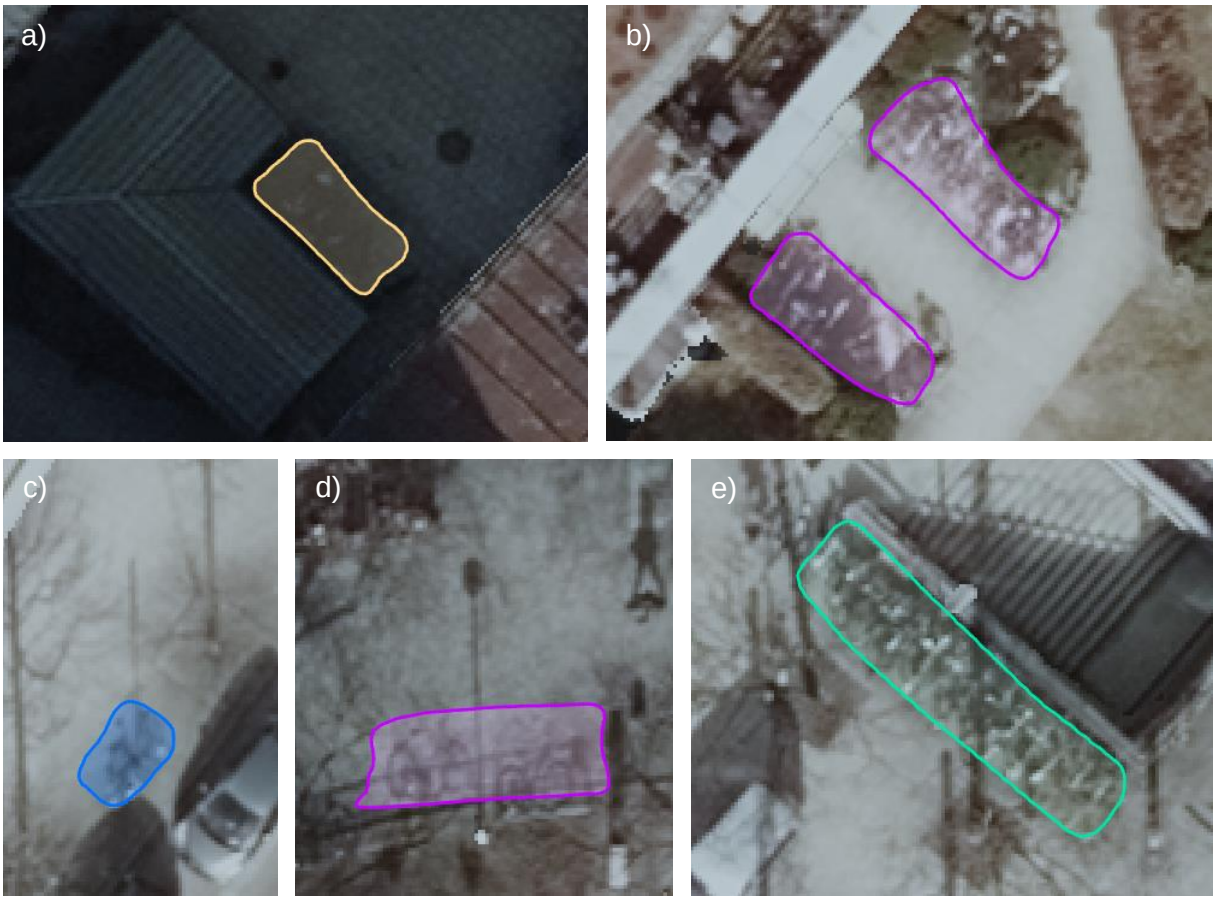


Figure 55: Examples for true positive detections that are not in the ground truth data: a) private parking facility in backyard with dark shadow (Pariser Str., with RGB+nDSM+NDVI); b) private parking facility without shadow (Spicherenstr., with RGB); c) single bike locked to street sign (Steinstr., with RGB+nDSM); d) single bikes locked to handrails (Weißenburger Pl., with RGB); e) bike parking facility at S-Bahn station not in database (Rosenheimer Pl., with RGB+NDVI) (Image source: Landeshauptstadt München 2020)

5.1.5 Classification

Table 10 shows that of all the labeled bicycle parking facilities, RGB can only classify 52.3% correctly. RGB+nDSM has the best result with 62.5%. CIR (61.7%), RGB+NDVI (60.2%) and RGB+nDSM+NDVI (58.6%) perform comparably well. The ratio of false detections to correct detections for CIR, RGB+nDSM, RGB+NDVI and RGB+nDSM+NDVI is for all close to 0.22. RGB has a value of 0.30.

When looking at the rack types, it is noticeable that in particular facilities with classvalue 50 are often classified incorrectly. Here the ratio of wrongly to correctly classified objects is $\frac{35}{7} = 5.0$, while for classvalue 10 it is $\frac{17}{156} \approx 0.11$, for classvalue 20 $\frac{27}{171} \approx 0.16$ and classvalue 22 it is $\frac{7}{65} \approx 0.11$. The wheelwells (classvalue 50) are similar in structure to the facilities with classvalue 20 and classvalue 22. However, the formation of the bikes is in practice more chaotic as in those of classvalue 20 and 22 due to the poor locking possibilities of this rack type. In terms of bike formations, it resembles racks of classvalue 10. Therefore, incorrect classifications occur for the racks of type 50. It should be noticed that the chosen areas of

study are not representative for all classes. Classvalue 99, classvalue 11 and classvalue 24 are only included in small proportions (<5%) in the test areas.

5.1.6 Application to Bicycle Parking Facilities Trained On

The results so far show the application of the trained models to areas that have been taken out of the training data. Applying the models to bicycle parking facilities that are part of the training data has a recall of over 98%. Only very few facilities are not detected. This is mostly because they are difficult to detect due to branches or dark shadows. The confidence of the detected and previously labeled objects is primarily over 99.8%. Figure 56 shows that the detected objects match the labeled objects very good. It is remarkable that this image section includes a parking facility at the U-Bahn exit that was not in the labeled data and has now been detected (Figure 56, highlighted by the star). The IoU is, with more than 90%, also higher than for facilities that were not used for training. In addition, the classification is correct for almost all of the detected and previously labeled facilities.

Figure 56: Detected bicycle parking facilities with RGB+nDSM in Laim (Gotthardstr./Friedenheimer Str.): red: labeled parking facilities; blue: detected parking facilities; star: facility not labeled but detected and property of the LHM

.

5.2 Application of the Model to Additional Aerial Imagery

To evaluate the applicability of the model to other unknown areas, the aerial imagery from 2017 is used. Furthermore, an area outside of the City of Munich is used. Here the area around the S-Bahn station of Ottobrunn (Landkreis München) is applicative as there are unroofed bicycle racks near the station and in the vicinity there is a school with many bicycle parking facilities. Since in the previous analyses, the model trained with RGB+nDSM performs best, this model is also used for the following detections.

5.2.1 Aerial Imagery of 2017

To test the transferability of the model to other aerial images, the same types of raster from 2017 are used. The model trained over all labeled bicycle parking facilities with RGB+nDSM is applied to the four known areas. It becomes apparent that the application to images with leafy trees is not beneficial with respect to bicycle racks. Figure 57 shows a section of the Rotkreuzplatz. Only one of the bicycle racks is almost completely free of occlusion. This is the only one that was detected in that image section. Two others could have been partially detected but are not. Eleven others, however, are completely hidden by trees.

As the images of 2017 were taken in summer, the shadows and lighting conditions are different than in the 2019 images. The shadows of the buildings, trees, and other objects are not as long, and more bike racks are in the sun unless they are covered by foliage.

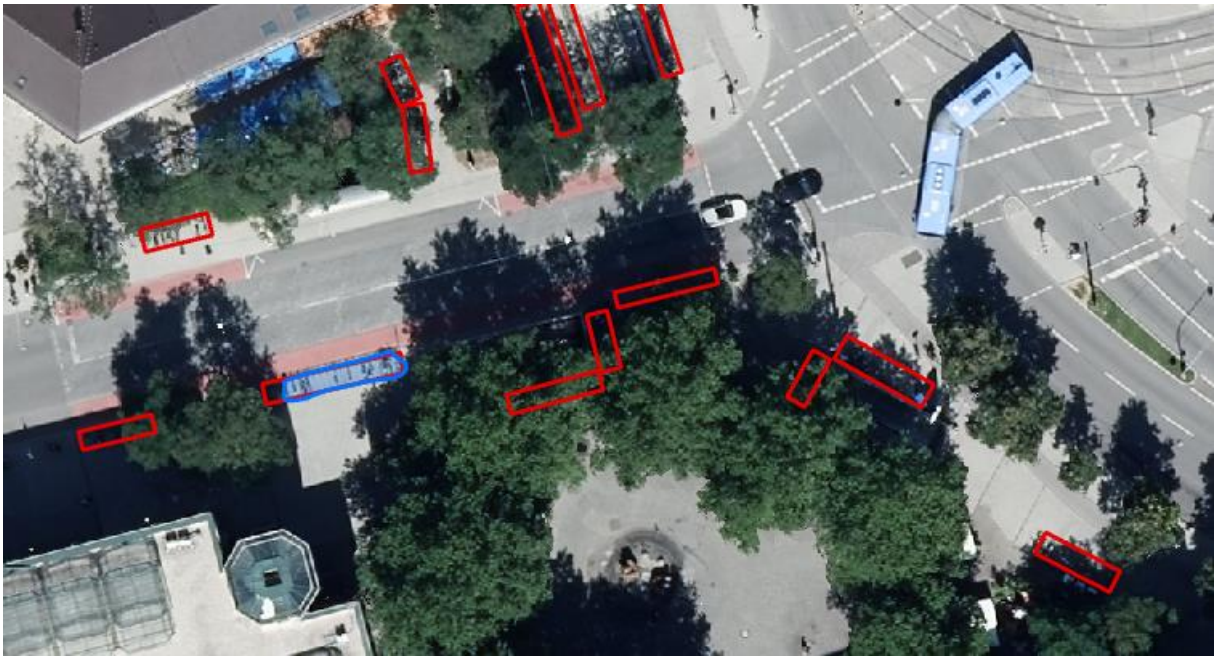


Figure 57: Detected bike parking facility at Rotkreuzplatz in 2017 imagery with RGB+nDSM; blue: detected object; red: labeled parking facilities (Image source: Landeshauptstadt München 2020)

Figure 58 shows a section of the less leafy area at Harras. Here, eight of the labeled bike racks are at least partially detected. Three are completely covered by foliage.

Figure 58: Detected bike parking facilities at Harras in 2017 imagery with RGB+nDSM; blue: detected objects; red: labeled parking facilities (Image source: Landeshauptstadt München 2020)

On the left of the image, six labeled parking facilities are not detected. The uppermost one is not present at the time of the image flight because there is a construction site here. To find reasons for the missing detection of the others, a comparison with the aerial image of 2019 is shown in Figure 59. For a good comparison, a bicycle parking facility is used, which is in the sun on the image of 2017 as well as on the image of 2019. On the left image, the facility is not detected, while it is detected on the right. It is obvious that there is a difference in the brightness of the images. In addition, the difference in resolution (2017: 10 cm; 2019: 8 cm) becomes apparent. While on the left the bicycles are not recognizable as such, on the right, better contours of the wheels and the shadows cast by the bikes are recognizable.

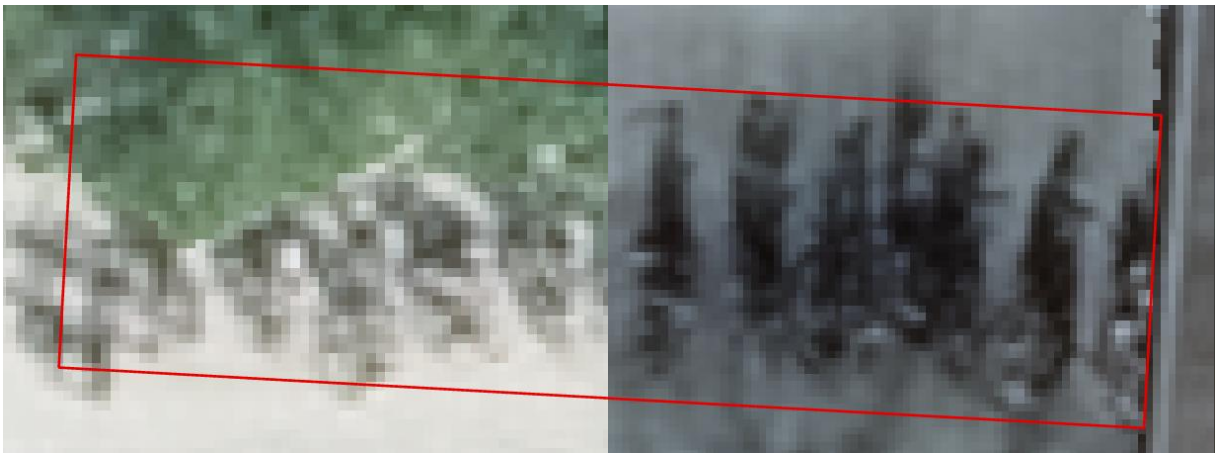


Figure 59: Comparison of RGB Image of 2017 (left) and 2019 (right) (S-Bahn Harras) (Image source: Landeshauptstadt München 2020)

5.2.2 Ottobrunn

For Ottobrunn, the data from 2019 is applied since no training was performed over this area. 15 out of 19 bike parking facilties were detected (Figure 60). This corresponds to 78.9%. One reason for the lack of detection of the four facilities might be their location in the sun. As described above, there is more training data in the shadow. Additionaly, since Ottobrunn is not part of the City of Munich, other types of bicycle racks are used here. This could also be a plausible reason. However, with a recall of more than 78% and no false positives, this image section shows that bicycle racks can also be detected satisfactorily outside of Munich in the same aerial image.

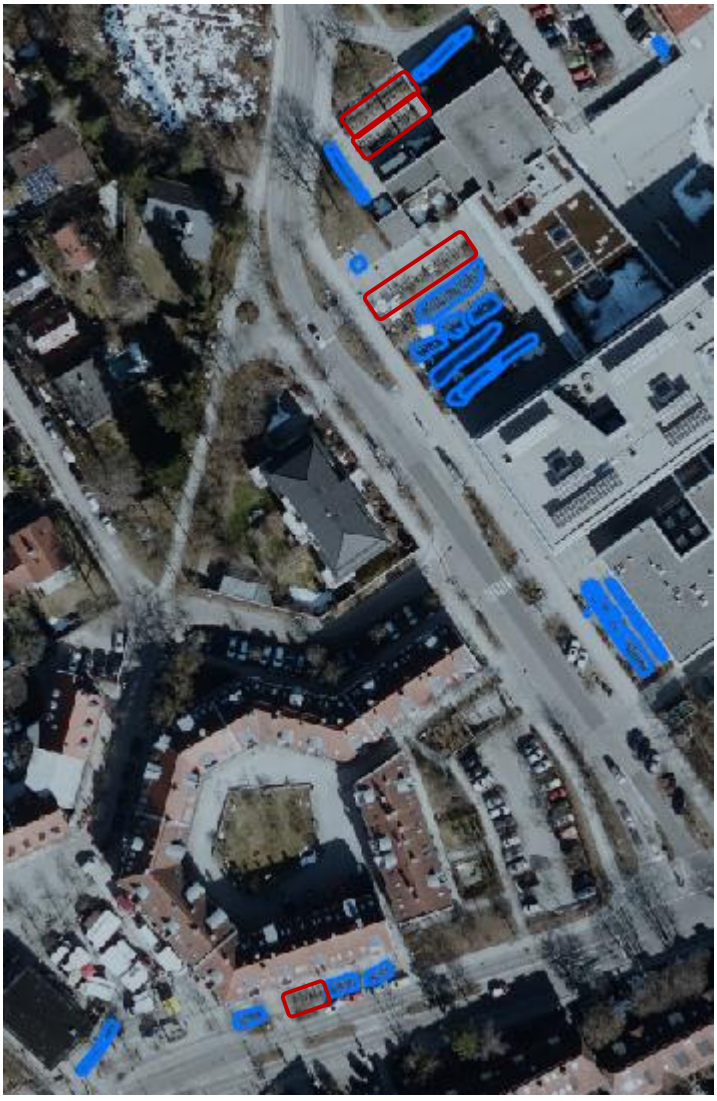


Figure 60: Detected bike parking facilities at S-Bahn Ottobrunn & Gymnasium Ottobrunn (with RGB+nDSM); blue: detected objects; red: false negative objects (Image source: Landeshauptstadt München 2020)

5.3 “Detected Objets to Lines and Rectangles” tool

The processing steps described in chapter 4.9 and the created *Detected Objets to Lines and Rectangles* tool that transforms the detected object masks into rectangles and lines can be executed by only giving the two intput feature classes and the two output names. Figure 61 shows an excerpt of the result. Here, all five labeled bike parking facilities (red) could be detected (pink masks). These masks are transformed into the orange rectangles and reduced to the green lines. These line objects are now of the same spatial structure as the line objects in Fahrradstaender_bis_2019.shp. Of course, they do not contain the same attributes. Fahrradstaender_bis_2019.shp contains attributes not attainable by deep learning like StandortNr, address, year of construction, etc. The new lines include next to the object id information on the length, the detected class, and the confidence value. The rectangular objects have the same attributes as the new line objects supplemented by information on the area, the width, the length, and the orientation.

Figure 61: Application of the new *Detected Objets to Lines and Rectangles* tool (Weißenburger Straße) incl. zoom detail; light blue: bike racks in Fahrradstaender_bis_2019.shp; red: ground truth labels; pink: detected object masks (with RGB); orange: new bounding rectangle; green: new line (Image source: Landeshauptstadt München 2020)

6 Discussion

The first major task of this thesis, to find a deep learning model for the detection of bicycle racks, is answered with a Mask R-CNN framework with a ResNet-50 backbone. The ArcGIS default settings showed to be not sufficient. A best possible trained model resulted from iteratively adjusting the parameters and an additional data augmentation, a best possible trained model resulted.

The application of this shows that with RGB data alone, good recall values of 68.0% and good precision values of 76.3% can be achieved. However, the additional implementation of a fourth band improved the detection. By adding the NDVI raster to the RGB bands a recall of 72.7% and a precision of 82.3% are achieved. Using a combination of RGB together with nDSM improved recall and precision to 77.3% and 83.2%, respectively. Both reduced false positive detections by the same number while RGB+nDSM increased the amount of true positive detections best. The precision and recall values mentioned also satisfy the requirements of the LHM (automation rate of 70 to 80%). The training with CIR, i.e., NIR, Red and Green, caused more false detections than RGB and even twice as many as RGB+nDSM respecitvely RGB+NDVI. Though, it performed comparatively well in true positive detection. This shows that the blue band is more important than the NIR for detecting racks. Adding two bands (nDSM and NDVI) to RGB does not show any improvement. On the contrary, it results in more misdetections than using the RGB alone. In all cases, it can be said that the area of detected racks reflects the area of an actual facility well. The superior performance of nDSM can be explained by the additional height difference that is included in the training set. Bicycles have a maximum height of approximately 1 m. The height of bikes also reflects the maximum height of detectable racks. Larger height differences than those of labeled parking facilities are excluded in the training process. Therefore, for example roof or cars are not included in the detected objects. The use of nDSM data is therefore not only helpful for large differences in height, e.g. with buildings (e.g., Roschlaub et al. (2020)), but also for small heights of the objects to be detected.

Adding bands based on NIR (NDVI) results in fewer falsely detected vegetation objects like hedges, trees, bushes, or grass areas. However, it does not eliminate these false detections completely. This is due to the fact that the training data and also the imagery tested on are images taken in winter with much less vegetation than in summer. Bushes and trees are mostly without foliage. However, NIR is only sensitive to green vegetation and not to naked branches. Moreover, it does not help much with detecting bike racks itself as they appear to be mostly in gray and black but rather with avoiding vegetative misdetections. To what extent training with

data in which green vegetation is present would further improve the results is difficult to say since conversely these would show significantly fewer bike racks due to the obscuring of the bike racks by their foliage.

Private bicycle racks are always included in the results. These makes the evaluation more difficult, but also showed that the model recognizes bicycle racks beyond the known ones. Since these are not relevant in the final results concerning public bicycle racks, they can be filtered out with a mask. The provided mask lhm_oeffentliche_flaechen_v2.shp reflects the public areas correctly in most locations but in some places, it does not match the areas where the LHM actually places bike racks. For example, not all areas near S-Bahn stations are included as here the Deutsche Bahn is the property owner. On the other hand, areas of schools are included, which often use their own bike racks. Such a mask is important to filter out false detections on railroad tracks. The results show that the structure of railroad tracks is too similar to the structure of bike parking facilities and are therefore prone to false detections. It should be mentioned for the actual application, a better mask than the one used would be helpful. However, the mask is not relevant for the interpretation of the results in this work.

Overall, shadow is a big issue. Due to the angle of the sun in winter, there is a lot of shadow in the city from trees and buildings. Accordingly, the training dataset also includes many bike parking facilities in shadow areas. While buildings cast continuous shadows, naked deciduous trees cast shadow patterns on their surroundings. The former reduces the contrast between the paving and the parking facility. The latter creates patterns that can make detection more difficult. In some cases, the shadows cast by individual bicycles leaning against a street sign, lamp post, etc. are detected since the dataset also contains single inverted-Us. Their appearance is therefore remarkably similar to that of a single bicycle. However, the number of single bike detections is small. The presence of bicycles in the parking facilities is of significant importance. More than 90% of the facilities in the training data have at least two parked bikes. Without bicycles, it is difficult to detect the thin and small racks with a resolution of 8 cm. The more densely bicycles are parked, the more a parking facility stands out from its surroundings.

Li et al. (2018) address the problem of 360° rotation of objects in remote sensing data when using deep learning. This is not a problem for bicycle racks, since they exhibit point symmetry due to their approximately rectangular shape.

A limitation of this study is the comparatively small test area and the small amount of training data. It appears to be sufficient. Though, for testing not that many labeled parking facilities can be taken out of the training dataset. Accordingly, the results of this thesis are based on small amounts of test data. In addition, more examples for certain rack types and more cases with

empty or almost empty bike racks as well as bike parking facilities not obscured by shadows would benefit the detection.

When applying the trained model to bike racks that are included in the training data, the detection results, as expected, are in a very good fit. In addition, it has already been possible to locate individual LHM racks that were, at least at the time of the image flight, at a location that was not recorded in the database. And this is, in the end, the objective of the deep learning approach on bike parking facilities in Munich.

One of the research questions of this work is, to what extent a classification of bike racks is possible. It turns out that the classification works partly, and with additional bands better than with RGB alone. RGB+nDSM again performs best here. For certain classes, however, it is not possible to evaluate the classification properly due to the small test dataset. However, the differentiation between inverted-Us and others, in which the front wheel is parked, such as those with classvalues 20 and 22 is possible. The classification can be used as a first indication. For classvalues 11, 24 and 99 the test areas did not include enough examples. For classvalues 99 and 24 it is expected that these are well distinguishable from the other rack types, provided that the number of training data is sufficient. The classvalue 99 represents MVG Rad facilities, which are very distinctive, and racks of classvalue 24 have a smaller width due to the inclination of the bikes. Classvalues 10 and 11 are very similar and mix-ups between these rack type classes are to be expected. Classvalue 50 has already proven to be problematic. However, the two rack types 11 and 50 are disappearing increasingly from the urban landscape and are no longer being installed by the LHM or even replaced by new ones.

The application of the model to the aerial imagery of 2017 has shown that it is ineffective to use images taken in spring or summer for bike rack detection. Nevertheless, it shows that bicycle racks are also detected in other images than in the one trained on. For a better result, however, the training dataset would have to be supplemented with brighter images with fewer shadows.

Finally, the possibilities and limitations of using ArcGIS Pro for such a deep learning task are to be discussed. ArcGIS Pro as a modern GIS has proven to be a valuable tool to view, process, and analyze the data. An advantage of ArcGIS / the ArcGIS Python API is that it provides many deep learning models to choose from. These are easy to apply, and the user does not need much deep learning experience or programming skills to get started. Drawbacks here are that it is not intended to modify the models extensively and not all background calculations are documented. However, it is possible to build and train models with other deep learning frameworks such as TensorFlow or PyTorch and to apply the trained model to the

data in ArcGIS with the Detect Objects Using Deep Learning tool. When working with large amounts of image data, it results naturally in long processing times for exporting and detecting. Unfortunately, the progress bars in ArcGIS Pro are not meaningful as they are jumping back and forth or have non-linear progress. This means that the duration of long calculation steps can only be estimated on the basis of empirical values. When trying to export image tiles with additional rotation, errors occur. As the tool feedbacks “Unknown Error” the reason cannot be determined but with the fast.ai transformations this is compensable. Applying the ArcGIS API for Python on Linux was initially challenging but once everything is installed and set up, the training can be performed with little coding. ArcGIS Pro also provides helpful tools for the analysis following the detection.

7 Conclusion & Outlook

It has been successfully shown that a detection of bicycle parking facilities in the City of Munich using deep learning leads to satisfactory results. For this purpose, a Mask R-CNN model with a ResNet-50 backbone was trained for instance segmentation. The precision of the result is improved by the additional inclusion of an nDSM raster. It helps to prevent some false detections due to the additional height information and supports the detection of bicycle parking facilities that have a maximum height of approximately 1 m. The classification and thus the distinction of the individual bicycle rack types is sufficient for a first categorization but cannot distinguish all subtypes.

It can be seen that, based on the available training data from the winter months in the beginning of the year 2019, the model was trained with many parking facilities that were completely or partially shaded at the time of the image flight. There are two major types of shadows to distinguish from: those cast by buildings and those cast by trees. The former casts continuous shadows while the latter casts light-dark patterns on their surroundings. Thus, bicycle racks, which are not affected by shadows, are underrepresented in the sample set. In addition, shadows cast by bicycles are an indication for the model that there is a parked bicycle at this location.

For detection, it is important that there are bicycles parked in the facilities. Empty bike parking facilities are difficult to detect as they are, on the one hand, underrepresented in the training data and, on the other hand, their structure is not adequately mapped at an image resolution of 8 cm.

So far, the model has been verified on four selected areas in the City of Munich. In addition, it has been applied to a small section in Ottobrunn outside of Munich, to an area from which the training data originates, and to aerial images from 2017. For an even better evaluation of the performance, an expansion of the area is worthwhile.

For the postprocessing of the detected objects, the steps used are combined in an ArcGIS Pro tool. With this tool rectangles and lines are derived from the object masks. One input parameter of this tool is a feature class, which contains all areas on which the LHM maintains or places bicycle racks. With the help of such a mask, private bicycle racks and some erroneous detections can be filtered out. For the application of the tool, it might be useful to supplement it in the future with information about the dimensions and distances of the individual racks within a bike parking facility. This would enable the length and area of the detected objects to be multiplied by a factor to estimate the capacity of a bike rack. Thus, the trained model

together with the toolbox can support the inventory of bike racks in the City of Munich. Nonetheless, it must be kept in mind that roofed parking facilities and bike racks in underpasses, etc. will not be part of the detection results.

In addition to the digital appendix containing the model and the toolbox (see Appendix C), there is a summary of the steps for an application of the trained model in German (Appendix A) for an easy implementation by the users at LHM.

The results of the model application can not only help to inventory the current state but also provide information for the future planning of parking facilities. The detections of single bikes, for example locked to lamp posts or street signs, as well as loose groupings of bicycles, indicate locations where no parking facility is at a reasonable distance from the cyclist's destination.

It will be exciting to find out which bicycle racks the model will recognize in future aerial images. For example, many new bike racks were installed in Maxvorstadt at the beginning of 2021. When choosing aerial images, however, it should be noted that in images taken in spring and summer months the foliage will obscure many of the facilities.

The transfer of the knowledge gained to other street furniture is also a possible extension beyond this work.

Bibliography

Active Things (2021): Active Things. Available online at https://activethings.app/home, updated on 4/26/2021, checked on 9/21/2021.

Allgemeiner Deutscher Fahrrad-Club e.V. (Ed.) (2011): Empfehlenswerte Fahrrad-Abstellanlangen. Anforderungen an Sicherheit und Gebrauchstauglichkeit. ADFC (TR 6102). Available online at https://www.adfc.de/fileadmin/user_upload/Expertenbereich/Politik_und_Verwaltung/Download/TR6102_0911_Empfehlenswerte_Fahrrad-Abstellanlagen.pdf, checked on 10/5/2021.

Anaconda Inc.; Docker Inc. (Eds.) (2021): continuumio/miniconda. Docker Image | Docker Hub. Available online at https://hub.docker.com/r/continuumio/miniconda, updated on 9/27/2021, checked on 9/27/2021.

Ankit, Sachan (2019): Detailed Guide to Understand and Implement ResNets. Available online at https://cv-tricks.com/keras/understand-implement-resnets/, updated on 9/20/2021, checked on 9/20/2021.

ArcGIS Map Service (2021): World_Topo_Map. Available online at https://services.arcgisonline.com/ArcGIS/rest/services/World_Topo_Map/MapServer.

Baeldung (Ed.) (2020): What is a Learning Curve in Machine Learning? Available online at https://www.baeldung.com/cs/learning-curve-ml, updated on 7/30/2020, checked on 9/25/2021.

Bayerische Vermessungsverwaltung (2020): Verwaltungsgebiete. Landkreise. Bayern: Bayerische Vermessungsverwaltung – www.geodaten.bayern.de. shp-file. Available online at https://opendata.bayern.de/detailansicht/datensatz/verwaltungsgebiete;jsessionid=D790317C6B4268890517A6CDBB07943A?0, checked on 9/3/2021.

Bymiljøetaten Oslo Kommune (Ed.) (2020): Sykkel i Oslo. Available online at https://www.arcgis.com/apps/webappviewer/index.html?id=6fe5df71a00f4d02bcd0a9622cd0524a, updated on 12/9/2020, checked on 10/7/2021.

Chollet, François (2018): Deep learning with Python. Shelter Island, NY: Manning (Safari Tech Books Online). Available online at https://learning.oreilly.com/library/view/deep-learning-with/9781617294433/OEBPS/Text/title.xhtml, checked on 10/7/2021.

City of Copenhagen (Ed.) (2012): Good, Better, Best. The City of Copenhagen's Bicycle Strategy 2011-2025. Technical and Environmental Administration Traffic Department. Available online at https://kk.sites.itera.dk/apps/kk_pub2/index.asp?mode=detalje&id=823, checked on 10/7/2021.

City of Vancouver (Ed.) (2021): Cycling in Vancouver. Available online at https://vancouver.ca/streets-transportation/biking.aspx, updated on 10/7/2021, checked on 10/7/2021.

City of Yakima Bicycle Committee (Ed.) (2021): Bike Rack Inventory. City of Yakima, WA. Available online at https://www.arcgis.com/apps/webappviewer/index.html?id=6beb8da8eea74e3794b3e5bfcbf2c50c, updated on 7/3/2021, checked on 10/7/2021.

CycleStreets Ltd. (Ed.) (2021): Urban Cycle Parking website goes live across London. London Cycling Campaign. Available online at https://www.urbancycleparking.org.uk/current, checked on 4/30/2021, not available on 10/7/2021,

https://www.cyclestreets.org/news/2016/03/21/urban-cycle-parking/, updated on 3/21/2016, checked on 10/7/2021.

Deng, Jia; Dong, Wei; Socher, Richard; Li, Li-Jia; Li, Kai; Fei-Fei, Li (2009): ImageNet: A Large-Scale Hierarchical Image Database. In : CVPR09. Available online at https://www.image-net.org, checked on 9/27/2021.

Dumoulin, Vincent; Visin, Francesco (2016): A guide to convolution arithmetic for deep learning. Available online at https://arxiv.org/pdf/1603.07285v2.pdf, checked on 9/24/2021.

ESRI (Ed.) (2021a): arcgis.learn module. arcgis 1.8.5 documentation. Available online at https://developers.arcgis.com/python/api-reference/1.8.5/arcgis.learn.toc.html, updated on 10/6/2021, checked on 10/7/2021.

ESRI (2021b): ArcGIS\Pro\bin\Python\envs\arcgispro-py3\Lib\site-packages\arcgis\learn_data.py. Source code of ArcGIS Pro Deep Learning.

ESRI (Ed.) (2021c): Introduction to deep learning. ArcGIS Pro | Documentation. Available online at https://pro.arcgis.com/en/pro-app/latest/help/analysis/deep-learning/what-is-deep-learning-.htm, updated on 4/18/2021, checked on 10/5/2021.

ESRI (Ed.) (2021d): Compute Accuracy For Object Detection (Image Analyst). ArcGIS Pro | Documentation. Available online at https://pro.arcgis.com/en/pro-app/latest/tool-reference/image-analyst/compute-accuracy-for-object-detection.htm, updated on 4/20/2021, checked on 9/25/2021.

ESRI (Ed.) (2021e): Detect Objects Using Deep Learning (Image Analyst). ArcGIS Pro | Documentation. Available online at https://pro.arcgis.com/en/pro-app/latest/tool-reference/image-analyst/detect-objects-using-deep-learning.htm, updated on 4/20/2021, checked on 10/7/2021.

ESRI (Ed.) (2021f): Deep Learning in ArcGIS Pro. ArcGIS Pro | Documentation. Available online at https://pro.arcgis.com/en/pro-app/latest/help/analysis/image-analyst/deep-learning-in-arcgis-pro.htm, updated on 4/21/2021, checked on 10/7/2021.

ESRI (Ed.) (2021g): ArcGIS API for Python. [Python Library]. ArcGIS Developer. Available online at https://developers.arcgis.com/python/, updated on 9/22/2021, checked on 9/27/2021.

ESRI (Ed.) (2021h): Faster R-CNN Object Detector. | ArcGIS Developer. Available online at https://developers.arcgis.com/python/guide/faster-rcnn-object-detector/, updated on 10/6/2021, checked on 10/7/2021.

ESRI (Ed.) (2021i): How Mask R-CNN works. | ArcGIS Developer. Available online at https://developers.arcgis.com/python/guide/how-maskrcnn-works/, updated on 10/6/2021, checked on 10/7/2021.

ESRI (Ed.) (2021j): How RetinaNet works? | ArcGIS Developer. Available online at https://developers.arcgis.com/python/guide/how-retinanet-works/, updated on 10/6/2021, checked on 10/7/2021.

ESRI (Ed.) (2021k): How single-shot detector (SSD) works? | ArcGIS Developer. Available online at https://developers.arcgis.com/python/guide/how-ssd-works/, updated on 10/6/2021, checked on 10/7/2021.

ESRI (Ed.) (2021l): YOLOv3 Object Detector. | ArcGIS Developer. Available online at https://developers.arcgis.com/python/guide/yolov3-object-detector/, updated on 10/6/2021, checked on 10/7/2021.

ESRI (Ed.) (2021m): Living Atlas of the World | ArcGIS. Deep Learning Packages. Available online at https://livingatlas.arcgis.com/en/browse/#d=3&q=type%3A%20deep%20learning%20package&type=tool, updated on 10/7/2021, checked on 10/7/2021.

ESRI Inc.: ArcGIS Pro. [Software]. Version 2.8.1.

Facebook's AI Research team: PyTorch. [Deep Learning Library]. Version 1.9: Facebook Inc. Available online at https://pytorch.org/, checked on 9/27/2021.

fast.ai: fastai. [Deep Learning Library]. Available online at https://docs.fast.ai/, checked on 9/27/2021.

fast.ai (Ed.) (2021): vision.transform | fastai. List of transforms for data augmentation in CV. Available online at https://fastai1.fast.ai/vision.transform.html, updated on 1/5/2021, checked on 9/25/2021.

Fractal AI@Scale Research Group (2019): Mask R-CNN Unmasked - Fractal AI@Scale Research Group - Medium, 4/26/2019. Available online at https://fractaldle.medium.com/mask-r-cnn-unmasked-c029aa2f1296, checked on 9/20/2021.

Garcia-Garcia, Alberto; Orts-Escolano, Sergio; Oprea, Sergiu; Villena-Martinez, Victor; Garcia-Rodriguez, Jose: A Review on Deep Learning Techniques Applied to Semantic Segmentation. Available online at https://arxiv.org/pdf/1704.06857.pdf, checked on 10/5/2021.

GB infraVelo GmbH (2021): Fahrradstellplätze an Berliner S- und U-Bahnhöfen. Available online at https://www.infravelo.de/projektarten/fahrradparken/standort-und-potenzialanalysen/, updated on 10/7/2021, checked on 10/7/2021.

Gemeente Amsterdam (Ed.) (2017): Voor fietsers en een gezonde en bereikbare stad. Meerjarenplan Fiets 2017-2022. Verkeer en Openbare Ruimte. Available online at https://www.amsterdam.nl/bestuur-organisatie/volg-beleid/verkeer-vervoer/meerjarenplan-fiets/, updated on 5/3/2019, checked on 10/7/2021.

Girshick, Ross (2015): Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

Girshick, Ross; Donahue, Jeff; Darrell, Trevor; Malik, Jitendra (2014): Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016): Deep Learning: MIT Press. Available online at http://www.deeplearningbook.org, checked on 10/7/2021.

Google Brain: TensorFlow. [Deep Learning Library]. Version 2.6.0: Google LLC. Available online at https://www.tensorflow.org/, checked on 9/27/2021.

Google Maps (2021): München-Allach - Google Maps. Available online at https://www.google.com/maps/place/M%C3%BCnchen-Allach/@48.1896964,11.46722,207m/data=!3m1!1e3!4m5!3m4!1s0x479e777e61f7d25b:0xc7aa71f83f07468d!8m2!3d48.1896955!4d11.4681306, updated on 9/28/2021, checked on 9/28/2021.

He, Kaiming; Gkioxari, Georgia; Dollár, Piotr; Girshick, Ross (2017): Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2016): Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

Hui, Jonathan (2018): Understanding Feature Pyramid Networks for object detection (FPN). In *Medium*, 3/27/2018. Available online at https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c, checked on 9/20/2021.

Ioffe, Sergey; Szegedy, Christian (2015): Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International*

Conference on International Conference on Machine Learning - Volume 37, pp. 448–456. DOI: 10.5555/3045118.3045167.

Kajsa Rosén (2021): Inventering och behovsanalys för cykelparkeringar. Edited by Afry. Available online at https://afry.com/sv/projekt/inventering-och-behovsanalys-cykelparkeringar, updated on 1/20/2021, checked on 10/7/2021.

Kathuria, Ayoosh (2018): What's new in YOLO v3? - Towards Data Science. In *Towards Data Science*, 4/23/2018. Available online at https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b, checked on 9/11/2021.

Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. (2012): ImageNet Classification with Deep Convolutional Neural Networks. In *Neural Information Processing Systems*. DOI: 10.1145/3065386.

Landeshauptstadt München (2020): File Geodatabases. 2017 + 2019 (RGB, CIR, nDOM, DOM, DGM, NDVI).

Leibniz-Rechenzentrum (Ed.) (2021a): 3. Access and Getting Started. Available online at https://doku.lrz.de/display/PUBLIC/3.+Access+and+Getting+Started, updated on 10/7/2021, checked on 10/7/2021.

Leibniz-Rechenzentrum (2021b): DSS Understanding DSS on the LRZ AI Systems. Available online at https://doku.lrz.de/display/PUBLIC/DSS+Understanding+DSS+on+the+LRZ+AI+Systems, updated on 10/7/2021, checked on 10/7/2021.

Leibniz-Rechenzentrum (Ed.) (2021c): LRZ AI Systems. Available online at https://doku.lrz.de/display/PUBLIC/LRZ+AI+Systems, updated on 10/7/2021, checked on 10/7/2021.

Li, Fei-Fei; Ranjay, Krishna; Danfei, Xu (2021): Convolutional Neural Networks for Visual Recognition. Course Notes CS231n - Stanford University. Available online at https://cs231n.github.io, updated on 9/3/2021, checked on 9/22/2021.

Li, Ke; Cheng, Gong; Bu, Shuhui; You, Xiong (2018): Rotation-Insensitive and Context-Augmented Object Detection in Remote Sensing Images. In *IEEE Transactions on Geoscience and Remote Sensing* 56 (4), pp. 2337–2348. DOI: 10.1109/TGRS.2017.2778300.

Li, Ke; Wan, Gang; Cheng, Gong; Meng, Liqiu; Han, Junwei (2020): Object detection in optical remote sensing images: A survey and a new benchmark. In *ISPRS Journal of Photogrammetry and Remote Sensing* 159, pp. 296–307. DOI: 10.1016/j.isprsjprs.2019.11.023.

Lin, Tsung-Yi; Dollár, Piotr; Girshick, Ross; He, Kaiming; Hariharan, Bharath; Belongie, Serge (2017a): Feature Pyramid Networks for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 936–944. DOI: 10.1109/CVPR.2017.106.

Lin, Tsung-Yi; Goyal, Priya; Girshick, Ross; He, Kaiming; Dollár, Piotr (2017b): Focal Loss for Dense Object Detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.

Liu, Li; Ouyang, Wanli; Wang, Xiaogang; Fieguth, Paul; Chen, Jie; Liu, Xinwang; Pietikäinen, Matti (2020): Deep Learning for Generic Object Detection: A Survey. In *International Journal of Computer Vision* 128 (2), pp. 261–318. DOI: 10.1007/s11263-019-01247-4.

Liu, Wei; Anguelov, Dragomir; Erhan, Dumitru; Szegedy, Christian; Reed, Scott; Fu, Cheng-Yang; Berg, Alexander C. (2016): SSD: Single Shot MultiBox Detector. In *Computer Vision -- ECCV 2016*, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2.

Ma, Lei; Liu, Yu; Zhang, Xueliang; Ye, Yuanxin; Yin, Gaofei; Johnson, Brian Alan (2019): Deep learning in remote sensing applications: A meta-analysis and review (152). Available online at https://reader.elsevier.com/reader/sd/pii/S0924271619301108?token=1AB9565714A8FCC9D65F8CE5ABF2A7BA108944F6DB31C0E9BDE7AB5A96846C76DA08B941BFA7000BEF02EEAF9374F251.

Mohan, Ankur (2018): Object Detection and Classification using R-CNNs. Available online at https://www.telesens.co/2018/03/11/object-detection-and-classification-using-r-cnns/, updated on 9/27/2021, checked on 9/27/2021.

NVIDIA Developer (2021): CUDA Toolkit 11.4 Update 1 Downloads. Download Installer for Linux Debian 10 x86_64. Available online at https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=Debian&target_version=10&target_type=deb_network, updated on 8/24/2021, checked on 8/27/2021.

Project Jupyter: Jupyter Notebook. [Programming Interface]. Available online at www.jupyter.org, checked on 10/8/2021.

Python Software Foundation: Python. [Programming Language]. Version 3.7.7. Available online at www.python.org, checked on 10/8/2021.

Redmon, Joseph; Farhadi, Ali (2018): YOLOv3: An Incremental Improvement. Available online at https://arxiv.org/pdf/1804.02767.pdf, checked on 9/10/2021.

Ren, Shaoqing; He, Kaiming; Girshick, Ross; Sun, Jian (2017): Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (6), pp. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031.

Roma Capitale (Ed.) (2020): Fase 2, al via piano straordinario per realizzare 150 chilometri di nuove ciclabili. Available online at https://www.comune.roma.it/web/it/notizia/fase-2-al-via-piano-straordinario-per-realizzare-150-chilometri-di-nuove-ciclabili.page, updated on 5/2/2020, checked on 10/7/2021.

Roschlaub, Robert; Li, Qingyu; Auer, Stefan; Möst, Karin; Glock, Clemens; Schmitt, Michael et al. (2020): KI-basierte Detektion von Gebäuden mittels Deep Learning und amtlichen Geodaten zur Baufallerkundung. In *zfv* (03/2020), pp. 180–189. DOI: 10.12902/zfv-0299-2020.

SchedMD LLC: SLURM. [Software]. Version 21.08. Available online at https://slurm.schedmd.com/, checked on 10/7/2021.

Schmidt, Paulina (2019): Mehr Radfahrer, weniger Unglücke. In *Süddeutsche Zeitung*, 8/10/2019. Available online at www.sz.de/1.4558847, checked on 10/7/2021.

Simonyan, Karen; Zisserman, Andrew (2015): Very Deep Convolutional Networks for Large-Scale Image Recognition. In *The 3rd International Conference on Learning Representations (ICLR2015)*. Available online at https://arxiv.org/pdf/1409.1556.pdf, checked on 9/23/2021.

Stadt Wien (Ed.) (2011): Radverkehrsanlagen und Radabstellanlagen im wien.at-Stadtplan. Available online at https://www.wien.gv.at/verkehr/radfahren/stadtplan.html, updated on 10/7/2021, checked on 10/7/2021.

Stockholms stad (Ed.) (2015): Cykelplan. En del av Framkomlighetsstrategin. Available online at https://start.stockholm/globalassets/start/om-stockholms-stad/sa-arbetar-staden/trafik/cykelplan.pdf, checked on 10/7/2021.

Su, Hao; Wei, Shunjun; Yan, Min; Wang, Chen; Shi, Jun; Zhang, Xiaoling (2019): Object Detection and Instance Segmentation in Remote Sensing Imagery Based on Precise Mask R-CNN. In *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, pp. 1454–1457. DOI: 10.1109/IGARSS.2019.8898573.

TensorFlow (2021): TensorBoard | TensorFlow. Available online at https://www.tensorflow.org/tensorboard, updated on 6/4/2021, checked on 9/27/2021.

Tharwat, Alaa (2021): Classification assessment methods. In *ACI* 17 (1), pp. 168–192. DOI: 10.1016/j.aci.2018.08.003.

The MathWorks, Inc.: MATLAB. [Software]. Version 2018a.

University of Chicago, Argonne National Laboratory (Ed.): Globus. [Software]. Available online at https://www.globus.org/, checked on 10/7/2021.

Ville de Paris (Ed.) (2021): Paris à vélo. Available online at https://www.paris.fr/pages/paris-a-velo-225, updated on 1/6/2021, checked on 10/7/2021.

Viswambharan, Vinay; Singh, Rohit (2021): Pre-trained deep learning models update. (February 2021). ArcGIS Blog. Edited by ESRI. Available online at https://www.esri.com/arcgis-blog/products/arcgis/announcements/announcing-new-pretrained-models-at-fedgis/, updated on 3/14/2021, checked on 10/7/2021.

Wu, Xiongwei; Sahoo, Doyen; Hoi, Steven C.H. (2020): Recent advances in deep learning for object detection. In *Neurocomputing* 396, pp. 39–64. DOI: 10.1016/j.neucom.2020.01.085.

Zhou, Ting (2021): Machine Learning Questions - GitBook. Mask RCNN. Available online at https://ztlevi.github.io/Gitbook_Machine_Learning_Questions/cv/two-stage-detector/mask-rcnn.html, updated on 6/9/2021, checked on 9/28/2021.

Zhu, Xiao Xiang; Tuia, Devis; Mou, Lichao; Xia, Gui-Song; Zhang, Liangpei; Xu, Feng; Fraundorfer, Friedrich (2017): Deep Learning in Remote Sensing: A Comprehensive Review and List of Resources. In *IEEE Geoscience and Remote Sensing Magazine* 5 (4), pp. 8–36. DOI: 10.1109/MGRS.2017.2762307.

Appendix A: Instructions on Detection & Tool for LHM (in German)

Anleitung für Detektion von Fahrradständern in ArcGIS Pro

Diese Anleitung beschreibt die Vorgehensweise für die Detektion von Fahrradständern mittels des in dieser Masterarbeit trainierten Deep Learning Models in ArcGIS Pro.
(verwendete Tools in kursiv; sowohl der englische als auch der deutsche Name des Tools ist angegeben)

Benötigte Daten für die Detektion von Fahrradständern

! Für die Nutzung der Deep Learning Tools ist die **Image Analyst** Erweiterung in ArcGIS Pro erforderlich.

Außerdem müssen die **Deep-Learning-Framework-Bibliotheken** installiert sein.
(Infos dazu unter https://pro.arcgis.com/de/pro-app/latest/help/analysis/deep-learning/install-deep-learning-frameworks.htm)

Benötigte **Rasterdaten** (für eine sinnvolle Detektion: Winterbefliegung, da durch das Laub im Sommer viele Fahrradständer verdeckt sind):

- RGB
- nDOM

Zusätzliche Dateien (in Ordner: Masterarbeit Karin Erbe – Appendix C):

- **trainiertes Deep Learning Model** (Mask R-CNN)
  der Ordner enthält:
  - model_metrics.html: Hintergrund-Informationen zu trainiertem Model
  - ModelCharacteristics (Ordner): Bilder, die Informationen zur Performance des Trainings enthalten
  - ArcGISInstanceDetector.py: Python Rasterfunktion
  - .emd-Datei (Esri model definition file): JSON-file, das Informationen zum Deep Learning Model und die Pfade zu den anderen Teilen des Models enthält
  - .dlpk-Datei (deep learning model package): Paket, das alle wichtigen Daten des trainierten Models enthält
  - .pth-Datei: PyTorch File mit dem gespeicherten Model
- **Toolbox** mit der Funktion für die Nachbearbeitung der detektierten Objekte

Zum Verständnis: Was wurde bis hierhin bereits ausgeführt?

Die Schritte in diesem Absatz müssen nicht ausgeführt werden. Sie dienen nur dazu, nachzuvollziehen, welche Schritte in der Masterarbeit durchgeführt wurden und worauf das trainierte Model basiert.

1. Labeln der Fahrradständer mit *Beschriften von Objekten für Deep Learning / Label objects for deep learning*: auf Basis von Fahrradstaender_bis_2019.shp wurden alle erkennbaren Fahrradständer der LHM entsprechend ihres Typs mit einem Rechteck umrundet. Dies dient als Ground Truth für das Training.

2. Export von Bildkacheln (.tif) mit Fahrradständern (Größe: 256 px x 256 px, Überlappung x und y: 50%) und entsprechend dazugehörenden Masken (R-CNN Masks), die zeigen an welcher Stelle im Bild ein Fahrradständer ist, mit *Trainingsdaten für Deep Learning exportieren / Export Training Data For Deep Learning.*

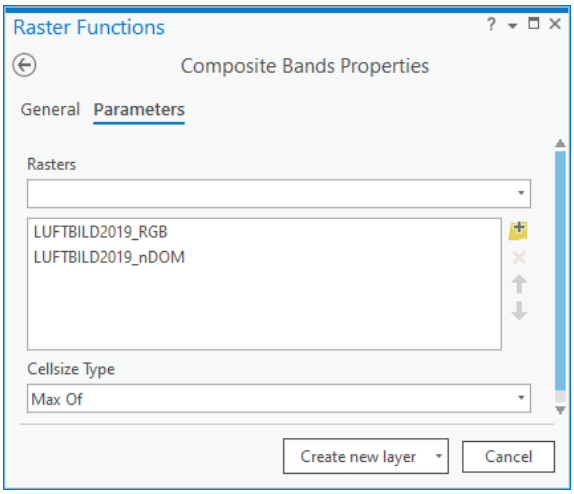
3. Für das anschließende Training ist eine GPU erforderlich. Daher wurde dieser Schritt mithilfe der ArcGIS API für Python auf einer Linux-Umgebung des Leibniz-Rechenzentrums durchgeführt. Trainiert wurde ein Mask R-CNN Model, um die genauen Umrisse der Objekte zu erhalten. Die einzelnen Einstellungen finden sich in der Masterarbeit. Das Ergebnis dieses Schrittes ist das Model, welches in dem oben beschriebenen Ordner zu finden ist.

Vorbereitende Schritte

Um das Model nun auf die gewünschten Bilddaten anzuwenden ist ein vorbereitender Schritt notwendig:

Mit *Bänder zusammensetzen (Raster-Funktionen) / Composite Bands (Raster Functions)* das RGB-Bild und nDOM-Raster zusammensetzen:

Tab Allgemein/General: Name vergeben



Tab Parameter/Parameters: beide Raster einfügen

Neuen Layer erstellen/Create new layer

Detektion (Anwenden des trainierten Models)

Nun kann das Model, welches auf RGB+nDOM trainiert wurde, für die Detektion der Fahrradständer eingesetzt werden:

Objekte mit Deep Learning erkennen / Detect Objects Using Deep Learning:

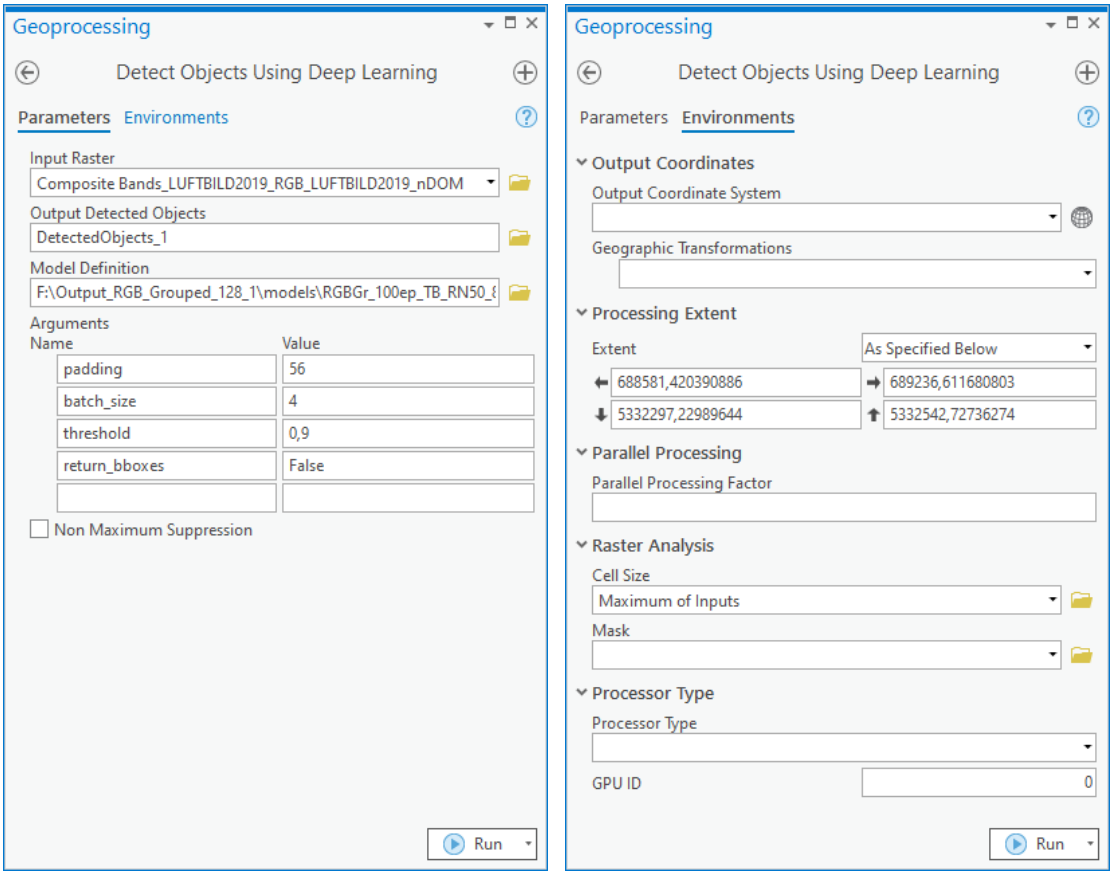
Tab Parameter/Parameters:

- Eingabe-Raster/Input Raster: das im vorherigen Schritt erzeugte Raster
- Ausgabename/Output Detected Objects: Name für den Output
- Eingabemodell/Model Definition: .emd-File (trainiertes Model)
- (weitere Argumente werden automatisch ausgefüllt)
- Non Maximum Suppression auswählen, wenn gewünscht ist, dass bei sich überlagernden Objekten das bessere ausgewählt wird

Tab Umgebungen/Environments:

- Verarbeitungsausdehnung/Processing Extent definieren – in welchem Bereich soll detektiert werden?
- ist eine GPU vorhanden, unten entsprechend auswählen

Ausführen/Run



! Die Ausführung des Tools dauert lange (abhängig von Leistung des Rechners), daher ist es sinnvoll eine Verarbeitungsausdehnung zu definieren oder eine leistungsfähige GPU zu verwenden

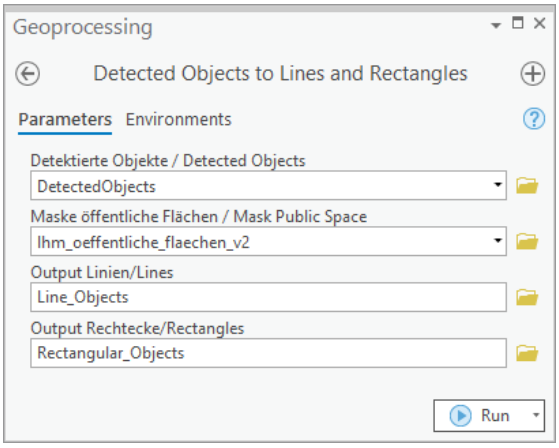
Ein Ausschnitt des Ergebnisses könnte dann so aussehen:

Class gibt die Klassifizierung / ID des Fahrradständertyps an (10: Anlehnbügel), die Confidence zeigt an, mit welcher Wahrscheinlichkeit das Objekt korrekt detektiert ist (durch den Grenzwert/Threshold bei Angabe der Argumente ist dieser mindestens bei 0.9).

Postprocessing

Das Model erkennt auch Fahrradständer auf Privatgrund. Daher müssen diese noch herausgefiltert werden. Zusätzlich ist es sinnvoll, die unförmigen Masken in Rechtecke und Linien umzuwandeln. Dies kann mit der beigefügten Toolbox durchgeführt werden:

Detected Objects to Lines and Rectangles (Detektierte Objekte zu Linien und Rechtecken):

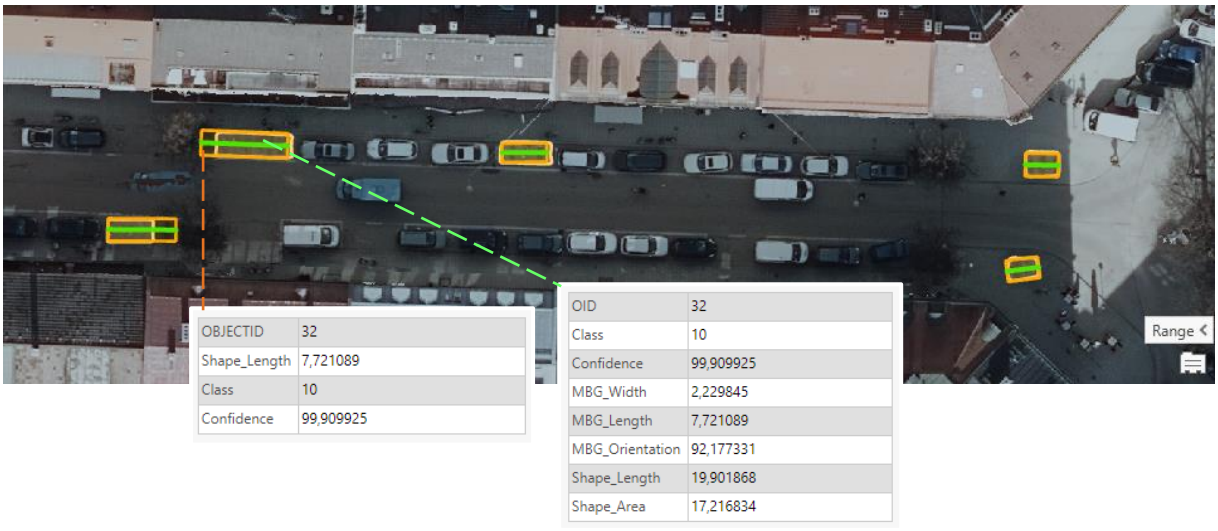


Tab Parameter/Parameters:

- Detektierte Objekte / Detected Object: Übergeben der Ausgabe aus dem vorherigen Schritt
- Maske öffentliche Flächen / Mask Public Space: Polygon Feature Klasse, die alle Flächen enthält auf denen die LHM Fahrradständer aufstellt/betreut
- Output Linien/Lines: Name für Output
- Output Rechtecke/Rectangles: Name für Output

Ausführen/Run

Das Ergebnis sieht dann so aus:



Was dieses Model nicht kann? Oder besser: Was ist zu erwarten?

Dieses Modell kann nur Fahrradständer detektieren, die nicht verdeckt sind (durch beispielsweise Überdachung, Bäume, etc.).

Bei Tests wurden bei Fahrradständern, die im Trainingsdatensatz enthalten sind, sehr gute Werte sowohl bei der Erkennung als auch bei der Klassifizierung erreicht. Allerdings kann es trotzdem vorkommen, dass einzelne Fahrradständer nicht detektiert werden oder eine falsche Klassifizierung erhalten.

Bei Fahrradständer, die nicht im Trainingsdatensatz enthalten sind, liegt die Präzision bei ca. 80%. Hier kommt es häufiger zu falschen Klassifikationen, insbesondere bei seltenen Typen. Das Model erkennt Fahrradständer weniger gut, wenn keine bzw. wenig Fahrräder dort abgestellt sind, diese in der Sonne liegen oder im dunklen Schatten stehen.

Neben der Detektion von privaten Fahrradständern, kommt es auch zu falschen Detektionen von Objekten wie Hecken, abgestellten Motorrädern, Bahngleisen, Baustellencontainern,… – durch eine geeignete Wahl der Maske bei der Nachverarbeitung können solche falschen Detektionen reduziert werden.

Außerdem werden teilweise auch einzelne Fahrräder erfasst, welche beispielsweise an Laternen oder Straßenschildern festgeschlossen sind, sowie sonstige Ansammlungen von Fahrrädern.

Bei der Klassifizierung ist zu beachten, dass einzelne Fahrradständertypen aufgrund ihres Erscheinungsbildes aus der Vogelperspektive zusammengefasst wurden:

Bezeichnung Fahrradständertyp	ID
Anlehne neu	**10**
Anlehne alt	**11**
A. geschwungen	12
L15G E T	**20**
L15G E H/T	21
L15G D T	**22**
L15G D H/T	23
L15 E T (schräg)	**24**
Arreta E	30
Kappa E H/T	41
Kappa D H/T	43
Kappa (schräg)	44
Kappa E H/T DB	46
Kappa D H/T DB	48
Kappa (schräg) DB	49
Klemmbügel	**50**
BCS	60
RGT	70
Doppelstockparker E	80
Doppelstockparker D	81
MVG	**99**
Unknown	100

Bezeichnung Fahrradständertyp	ID
Anlehne neu	**10**
Anlehne alt	**11**
A. geschwungen	12
L15G E T	**20**
L15G E H/T	21
Arreta E	30
Kappa E H/T	41
L15G D T	**22**
L15G D H/T	23
Kappa D H/T	43
L15 E T (schräg)	**24**
Kappa (schräg)	44
Klemmbügel	**50**
BCS	60
RGT	70
Unknown	100
MVG	**99**

Aus den Informationen über die Fläche und Länge der Rechtecke ließe sich die Kapazität einer Fahrradabstellanalge abschätzen. Dafür wären Informationen über die genauen Maße und Abstände der einzelnen Fahrradständer nötig. Auf Grundlage der Klassifizierung könnte dann die Zahl der abstellbaren Fahrräder für eine grobe Abschätzung berechnet werden.

Appendix B: Python Code - Training

```
# -*- coding: utf-8 -*-
"""
Python Code for Training a Mask R-
CNN Deep Learning Fragmwork based on the ArcGIS API for Python and argis.learn
module in order to detect bike parking facilities in aerial imagery
Masterthesis: Detection of Bicycle Racks from Geodata using Deep Learning
@author: Karin Erbe
"""

#imports
from pathlib import Path
from arcgis.learn import prepare_data, MaskRCNN

import fastai
import torch
from fastai.vision.transform import rotate, flip_lr, crop, brightness, contrast,
rand_zoom, dihedral_affine

#pre-checking: if CUDA is available & Torch version
if torch.cuda.is_available():
    print("CUDA is available.")
else:
    print("CUDA not available - no GPU connection! ")

print("Torch version: "+torch.__version__)

#initializing  path that includes images and labels
home = Path.home()
data_path = Path(home,'Output_ExportTrainingDataFolder')
print("Data path: "+data_path)

#data augmentation transformation settings for fastai
chip_size=224
ranges = (0, 1)

train_tfms = [rotate(degrees=(-30,30), p=0.5),
                    flip_lr(p=0.5),
                    crop(size=chip_size, p=1., row_pct=ranges, col_pct=ranges),
                    brightness(change=(0.4, 0.6)),
                    contrast(scale=(1.0, 1.5)),
                    rand_zoom(scale=(1.0, 1.2))]
val_tfms = [crop(size=chip_size, p=1., row_pct=0.5, col_pct=0.5)]
tfms = (train_tfms, val_tfms)

#data preparation with BatchSize = 8 and ValidationPercentage: 20%
data = prepare_data(data_path, transforms=tfms, batch_size=8, imagery_type='ms',
val_split_pct=0.2, seed=174413)

#model: Mask R-CNN with ResNet-50 backbone
model = MaskRCNN(data, backbone='resnet50')

#finding optimal Learning Rate lr
lr = model.lr_find()
lr
print('lr:', lr)

#training of the model and visualization of training&validation loss in TensorBoard
model.unfreeze()
model.fit(epochs=100,lr=lr, tensorboard=True, checkpoint='all')

#save results
name = 'TrainedModel_FolderName'
model.save(name)
print('Name: ',name)
```

Appendix C: List of Attached Files

Folder Masterarbeit Karin Erbe – Appendix C

- Trained Model: RGBnDOM_MaksRCNN_Model.zip
- Toolbox: DeepLearningFahrradstaender.tbx