University of Münster

Department of Information Systems

Detection and Segmentation of Tree Instances on a Rwandan Satellite Dataset

Bachelor Thesis

submitted by

Steffen Backmann

Chair of Data Science:
Machine Learning and Data Engineering

Principal Supervisor	Prof. Dr. Fabian Gieseke
Supervisor	Christian Lülf, M.Sc.
	Chair for Data Science:
	Machine Learning and
	Data Engineering
Student Candidate	Steffen Backmann
Matriculation Number	503356
Field of Study	Information Systems
Contact Details	sbackmann@uni-muenster.de
Submission Date	29.07.2022

Contents

1 Introduction

Ever since Landsat 1, the first satellite with the purpose of producing satellite data of the earth's surface, was launched into space, the amount of satellites orbiting earth has grown constantly [CW11, p. 15]. As of 2020, more than 1500 active payloads, which are the scientific instruments aboard a satellite, were located in the lower earth orbit alone [McD20, p. 3]. Combined with new remote sensing technologies such as unmanned aerial vehicles (UAVs), the availability of earth surface data with extremely fine spatial resolution has soared [Gha+19, p. 7]. Concurrently, substantial advancements in machine learning in general and deep learning in particular have been achieved, so that starting in 2014, deep learning has found its way into remote sensing [Ma+19, p. 166]. This has led to a variety of research drawing from machine learning models trained on satellite images and covering such different questions as the connection between nighttime lighting and poverty [Jea+16] or the prediction of wetland occurrences [Hir+17].

A particularly auspicious application of deep learning in satellite data analysis is the field of detecting and segmenting objects. This is relevant for fields like ship detection [Wei+20] and building extraction [Wag+20] but also for mapping ecosystems in terms of trees, bushes and plants in general since it promotes our understanding of nutrient cycles such as carbon and water as well as that of general biogeography [HA20, p. 42]. Consequently, respective studies regarding the identification of trees on satellite images have been conducted [Bra+20], [Oce+20], [Sun+22]. These all rely on deep convolutional neural networks (CNNs).

The idea of artificial neural networks (ANNs) was brought forward as early as 1943 when McCulloch and Pitts [MP43] modeled biological neurons with a series of logic gates. The Perceptron, an adjusted model by Frank Rosenblatt [Ros58] and the development of an efficient algorithm to train ANNs called backpropagation [RHW86], helped to shape ANNs into their modern form. Meanwhile, building upon the discovery of Hubel and Wiesel [HW68] that cats perform visual processing by combining the outputs of simple, lower-level neurons within higher-level neurons, LeCun et al. [LeC+89] developed the landmark LeNet architecture, introducing convolutional and pooling layers as essential components of today's CNNs.

Albeit these early advances, CNNs did not play a major role until 2012 [LBG15, pp. 439–440], when Krizhevsky, Sutskever, and Hinton [KSH12] proposed a new CNN architecture called AlexNet and won the ImageNet competition, an important image classification contest, by a large margin. Since then, CNNs power state-of-the-

art solutions in various image recognition tasks [LBG15, p. 436]. In 2014, Hariharan et al. [Har+14] first introduced the task of instance segmentation which combines the detection of individual objects within an image (object detection) and classifying the image on a per-pixel level (semantic segmentation) [Har+14, p. 297]. Following this, a vast number of instance segmentation models have been proposed, as surveyed by e.g., Hafiz and Bhat [HB20].

Respective networks have also been used in the segmentation of trees on satellite data. Brandt et al. [Bra+20] identified trees with crown sizes of more than 3 m^2 in desert areas on a massive scale by combining a U-Net semantic segmentation model [RFB15] with the highlighting of spaces between overlapping trees. Ocer et al. [Oce+20] used Mask R-CNN [He+17], an instance segmentation network, to detect trees on UAV data stemming from a campus and a lake area. Sun et al. [Sun+22] developed a Cascade Mask R-CNN [CV19] model that counted 112 million individual trees in an urban area for trees with a minimum crown size of only 1 m^2.

In this thesis, a solution for detecting and segmenting trees in a dataset of satellite images from Rwanda is implemented. The data originates from all over the country and is annotated with labels for the individual tree crowns, thus allowing a supervised learning approach with an instance segmentation network. A suitable model is determined by evaluating different instance segmentation approaches. The results of the deployed model are evaluated mainly on its mask average precision (AP). To follow a standardized process, the project's general structure approaches the CRISP-DM (**CR**oss **I**ndustry **S**tandard **P**rocess for **D**ata **M**ining) methodology as suggested by Wirth and Hipp [WH00]. Visualized in Figure 1, CRISP-DM defines the phases Project Understanding, Data Understanding, Data Preparation, Modeling, Evaluation and Deployment. While it could be integrated into the Modeling phase, this work highlights the Model Selection as an additional phase. The Deployment phase is omitted.

Being a large-scale instance segmentation approach on tree crowns, this thesis contributes to the current research state. Ocer et al. [Oce+20] and Sun et al. [Sun+22] used instance segmentation networks for similar problems, yet both are restricted to a limited area and while Brandt et al. [Bra+20] covered a much larger area, they did not utilize an instance segmentation model. Looking at the broader research field, this work is one small mosaic in the application of instance segmentation approaches to more restricted but also specialized problems than the segmentation of daily objects in datasets such as COCO or ImageNet. As such, they are likely to show differing properties and challenges.

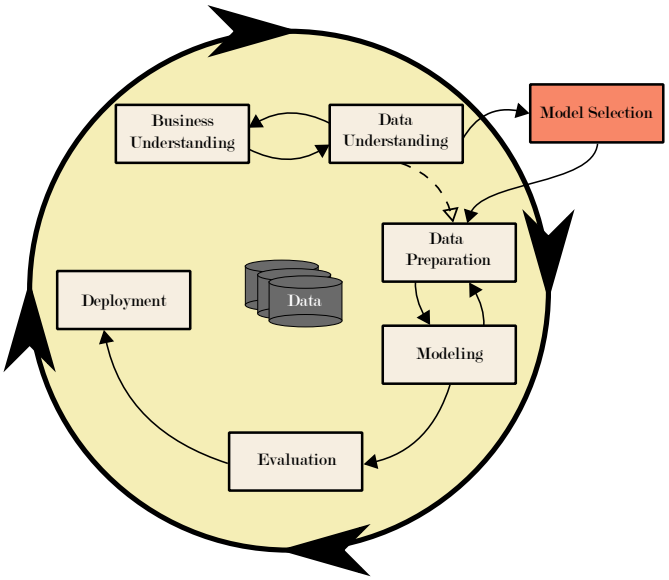
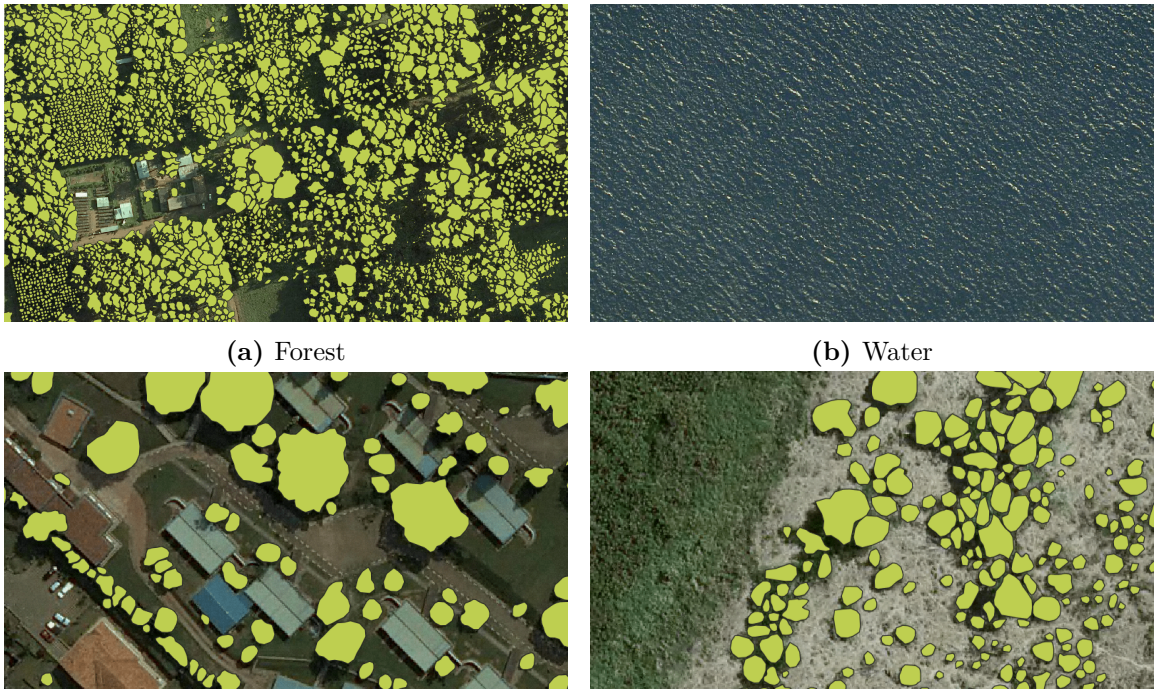


Figure 1 Adapted CRISP-DM model, originally from Wirth and Hipp [WH00].

Business Understanding. Wirth and Hipp [WH00] list four generic tasks to develop a business understanding. These are the determination of business objectives, an assessment of the situation, determining data mining goals, and the production of a project plan. The data has been collected in cooperation with the Rwanda Forest Agency. Due to limited insights in the organization and the local situation, at this point only business objectives and data mining or rather system goals are discussed. A network to count individual trees would simplify surveys on the number of trees within a given region, reducing the need for field work. For this, an object detection system would be sufficient. Apart from that, the tree canopy area should be predicted, one reason being that this allows the estimation of the underlying biomass and thus the amount of CO_2 that is stored within. Combining those tasks requires an instance segmentation network to identify and segment individual trees. In addition to enabling further analyses, a potential use for the timber industry is expected.

Data Understanding. The dataset consists of 112 satellite images with 133870 labeled tree crowns in total. The images have three color channels (RGB) and different sizes. They are sampled from all over Rwanda and thus cover different terrains. A number of sample image patches and their ground truth annotations are shown in Figure 2. When looking at the samples, a number of challenges can be identified. For a start, the number of objects vary greatly between the different samples, even featuring images such as Figure 2b which contains no objects at all. Contrasting this, in Figure 2a the sheer amount of objects, the small size and the proximity of the trees could hamper the identification as well as the delineation between tree instances. Containing a clear demarcation line, Figure 2d shows a wetland terrain. Due to its visual similarity to a forest region, this could generate a number of false positives.



(a) Forest **(b)** Water

(c) Urban area **(d)** Wetlands and desert

Figure 2 A small number of samples from different terrains reveal a set of challenges to cope with.

Thesis Structure. In the following, the structure of the remainder of this thesis is shortly outlined.

In Chapter 2, fundamental concepts of deep learning necessary to understand modern instance segmentation architectures are explained. Additionally, different image recognition tasks are introduced, thus demonstrating how instance segmentation unites aspects from neighboring problems.

Chapter 3 is dedicated to the collection and in-depth explanation of different instance segmentation networks. Finally, a selection is made for the implementation phase.

In Chapter 4 the methodology of the implementation, its setup and the data preparation steps are described.

In Chapter 5 the results of the different configurations are presented and discussed.

The last chapter provides a conclusion and highlights the insights as well as limitations of this thesis.

2 Background

In order to select and implement a model that segments tree instances on the dataset at hand, it is essential to understand the theoretical fundamentals behind the model on the one hand and the properties as well as objectives of the problem, that is instance segmentation, on the other hand. Thus, in this chapter essential deep learning concepts are explained, introducing ANNs, CNNs and respective architectures. This is followed by an elaboration on different image recognition tasks. Besides instance segmentation itself, disciplines which are closely related are also examined.

2.1 Deep Learning

2.1.1 Artificial Neural Networks

While performing complex mathematical operations or simulating climate systems can be done much better by computers than humans, there are other tasks like identifying cars in an image or understanding and processing speech that are easy for humans but have been incredibly hard for computers. This is due to the ways in which the human brain works and thus the idea to translate those concepts into computer programming is not yet far-fetched.

The first corresponding advances were done by McCulloch and Pitts [MP43] in 1943 who described the activities of animals' neural nets as a series of logic gates, thus constituting a computational model. They assume each neuron to be activated in a so-called all-or-none process depending on a fixed number of excited input synapses. Later, Rosenblatt [Ros58] proposed the Perceptron, a slightly different model whose neurons are activated if the weighted sum of their inputs surpass a specific threshold. Rosenblatt proposed a more complex model with an input area that takes in stimuli called the Retina, a subsequent projection area and association area that further process the inputs, and finally a response area. However, today the term Perceptron most often means a structure where the input layer is directly connected to the output layer [Gér19, pp. 284–285]. Flaws of this initial Perceptron version as its inability to handle datasets that were not linearly separable were solved by using a Multilayer Perceptron (MLP) and nonlinear activation functions [Gér19, p. 288]; [GBC16, pp. 171–173].

A good understanding of a neural net's basic components can be gained from taking a look at the MLP in Figure 3. For each input – which might be pixels of an image or one instance's features in a dataset – there is one input neuron. Because of the fact

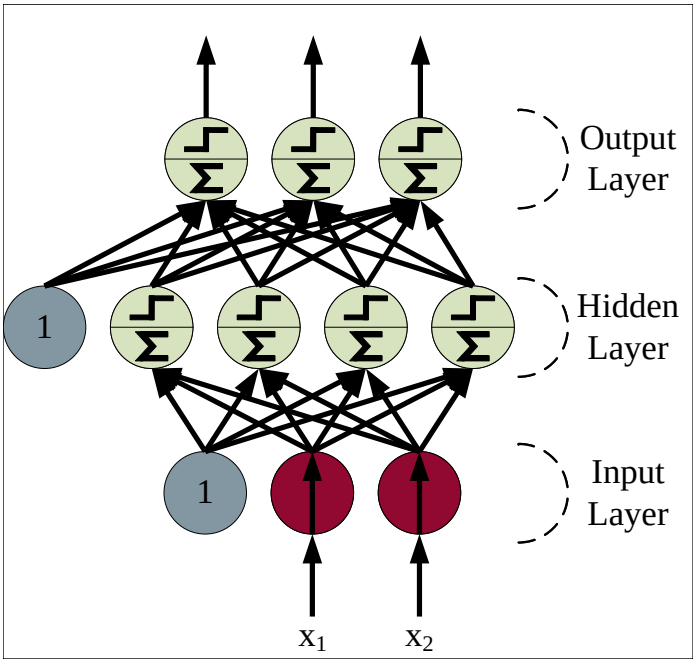


Figure 3 This MLP consists of two inputs, one hidden layer with four neurons, and three outputs. The layers are fully connected and bias neurons are added. Adapted from Géron [Gér19, p. 289].

that the inner part of the network with inputs and outputs is not instantly visible and accessible from the outside, the subsequent layer is called a hidden layer. Although it is the only one in this case, there can be multiple hidden layers in a neural net which is then called a deep neural network [GBC16, pp. 168–169]. Its neurons are connected to every output of the previous layer making it a fully connected layer or dense layer. Each neuron has weights associated with its inputs including the bias and computes the sum of those weighted inputs (thus the Σ symbol). A non-linear activation function is then applied to this sum, its result being the output of the neuron. From a variety of possible activation functions, here the heaviside step function (⎍) is chosen that outputs 0 if its input is smaller than 0 and 1 otherwise. The same logic applies to the output layer. As the above computations, which are also called a forward pass, are unidirectional from the input to the output layer, this is called a feedforward network with layers close to the inputs called lower layers and those close to the outputs called upper layers [Gér19, p.289].

Training an ANN. Like in other machine learning approaches, in order to evaluate and optimize the outputs of a neural network, a loss function quantifying the error of the current solution is needed which has then to be minimized [GBC16, pp. 177–178]. Given a neural network, the number of layers, the number of neurons and the activation functions remain constant. As the inputs are determined by the data, the only parameters left to tweak are the weights that are associated with each input of the neurons (except for those in the input layer). Thus, when speaking of training a neural

network, this means optimizing its weights with regard to the loss function. Due to the nonlinearity of neural networks (caused by the nonlinear activation functions), neither can the loss function's minimum be determined by solving a linear equation system, nor is the loss function convex which would guarantee gradient descent to converge to the global minimum [GBC16, pp. 177–178].

Backpropagation. In the iterative process of gradient descent, the loss function's gradient, which points in the direction of the steepest ascent, is determined [Gér19, p. 118]. The negative of the gradient thus yields how to marginally change the weights to maximize the reduction of the loss function's value. For many years, a major challenge has been to efficiently calculate the gradients for the training process. In 1986, Rumelhart, Hinton, and Williams [RHW86] achieved a breakthrough and presented a method called backpropagation. This algorithm needs only one forward and one backward pass to efficiently compute the gradient. After the forward pass, the authors determine how much each output neuron adds to the total loss and how a marginal change in each output neuron would affect the total loss. In other words, these are the partial derivatives of the loss function with respect to the outputs. As these outputs are merely a function influenced by the weights in the output layer, they apply the chain rule to compute the derivatives with respect to those weights. In this manner, going backwards through the network, they compute the derivatives for the weights of the lower layers, which are dependent on the upper layers' derivatives and thus those results are used for further computations, eventually yielding the partial derivatives with respect to every weight, i.e. the gradient.

Here, activation functions play a key role in that their gradients heavily influence the weight updates, making the heaviside step function with its flat gradient from the example in Figure 3 unsuitable. Consequently, other functions were used instead, one example being the sigmoid function defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

which provides the advantage of a clearly defined derivative [Gér19, pp. 143, 291].

2.1.2 Convolutional Neural Networks

While ANNs are a powerful tool, they have some limitations. The fully connected layers expect a fixed number of inputs, thereby complicating the handling of images which come in different sizes. Additionally, as each pixel in an ANN translates to one input neuron and the subsequent hidden layers are fully connected, complex networks with a huge number of weights arise [Lec+98, p. 2282]. Meanwhile, the

2-D structure of an image and the high correlation between nearby pixels are ignored [Lec+98, p. 2283]. Consequently, the performance of ANNs in tasks such as image classification has been meager [Gér19, p. 445]. In turn, breakthroughs in these fields have been reached by CNN architectures [Gu+18, p. 370].

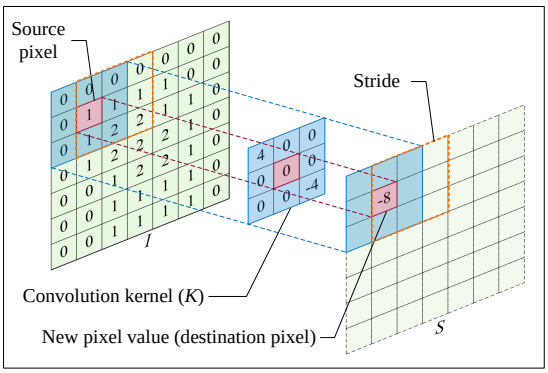
In 1968, Hubel and Wiesel [HW68] discovered that monkeys perform visual processing by combining the outputs of simple lower-level neurons within higher-level neurons. Moreover, their neurons respond only to very specific light stimuli and only within a small receptive field. Building upon that, Fukushima [Fuk80] presented the Neocognitron which incorporated S-cells and C-Cells in its architecture, thereby introducing the ideas of convolution and pooling. In the context of recognizing handwritten digits, LeCun et al. [LeC+89] applied a training algorithm using backpropagation and convolutions directly to images. This can be seen as the first modern convolutional neural network [GBC16, p. 368].

Convolutional layers. Apart from dense layers, the two most important components of CNNs are convolutional layers and pooling layers [ON15, p. 4]. In contrast to a dense layer, a neuron in a convolutional layer is not connected to every neuron of the previous layer but only to a handful of neurons in its receptive field [Gér19, p. 445], thus following the findings of Hubel and Wiesel [HW68]. The receptive field is defined by a width and height called the kernel size as can be seen in Figure 4a. In combination with the receptive field, the kernel, which is also called filter, performs the convolution, that is an element-by-element multiplication and summation of the respective matrices [GBC16, pp. 331–333]. For one location (i, j) in an output feature map S, this is shown in the following formula with an input layer I consisting of l feature maps and a kernel K with size (m, n):

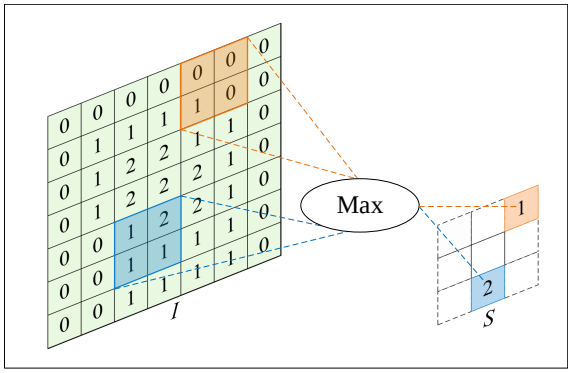
$$S(i,j) = \sum_m \sum_n \sum_l I_l(i+m, j+n) K(m,n) \text{ [GBC16, p. 333]; [Gér19, p. 453].}$$

Note that in Figure 4a, I only consists of one input feature map. There are usually multiple filters with learnable configurations applied to extract a variety of features, thus producing multiple output feature maps. As it is the kernel parameters (weights) that are trained, convolutional layers have the property of sharing weights, that is all input locations are subject to the same set of weights, thereby drastically reducing the parameters in the network and enabling the detection of the same feature irrespective of its position [Lec+98, p. 2283]. The output feature maps' sizes are determined by the stride, which indicates by how many pixels the receptive field in I is shifted for every pixel in S [Gér19, pp. 448–449]. With a stride of 1 as in Figure 4a the sizes remain unchanged.[1] Like the kernel size, the stride can have different values

[1] This also depends on the padding which shall not be explained here.



(a) A convolutional layer with a kernel size of (3, 3) and a stride of (1, 1). Adapted from Albawi et al. [Alb+18, p. 5].



(b) A pooling layer with a 2 x 2 max pooling kernel and a stride of 2. Adapted from Albawi et al. [Alb+18, p. 5].

Figure 4 Convolutional layers and pooling layers are the main components of CNNs.

for width and height. A convolutional layer is typically followed by a detector layer implementing a nonlinear activation function, most prominently rectified linear unit (ReLU) [GBC16, p. 339]. The ReLU function is defined as

$$f(x) = \max(0, x).$$

Introduced by Nair and Hinton [NH10], its main advantages are that it is easy to compute and in contrast to logistic activation functions, it has no asymptotic upper bound. Consequently, as shown by Glorot, Bordes, and Bengio [GBB11], it circumvents the problem that gradients tend to vanish for saturating activation functions.

Pooling layers. The second characteristic layer type of a CNN, pooling layers, help reduce the dimension of the feature maps, thereby also reducing the number of calculations and the computational requirements of the network [ON15, p. 8]. Using a pooling kernel and a pooling function, a pooling layer takes an input feature map's neighborhood and computes a summary for the output feature map [GBC16, pp. 339–340]. The most prominent pooling functions are max pooling and average pooling. For max pooling, as can be seen in Figure 4b, the maximum value out of the neighborhood is forwarded to the output feature map. Similarly, average pooling layers average the pixel values, either equilibrated or weighted by their distance to the central pixel [GBC16, p. 339–340]. Apart from reducing model complexity, pooling helps making the network more invariant to slight shifts of the input. That means, many values of the pooling layer's output feature map do not change if the input is marginally translated because of the pooling layer's sensitivity to the neighborhood instead of the exact pixel location [Gér19, pp. 457–458].

CNN architectures. Over the past decades, a variety of CNN architectures were proposed to cope with all kinds of image recognition tasks. One of the earliest exam-

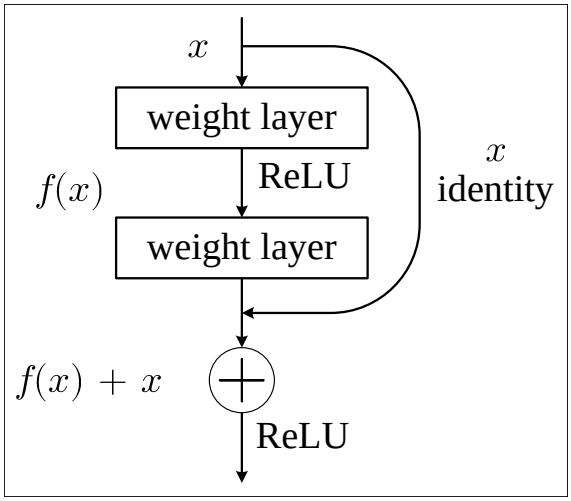


Figure 5 In ResNet, skip connections skip two layers. Adapted from He et al. [He+16].

	Number of convolution and dense layers
LeNet-5	5
AlexNet	8
ResNet-50	50
ResNet-101	101

Table 1 CNN architectures became much deeper in the past years.

ples is LeNet-5 by Lecun et al. [Lec+98], originally developed to recognize handwritten digits. Although LeNet-5 has been around since 1998, due to the lack of large training sets and respectively sized CNN architectures [RFB15, p. 234] and because GPUs were not yet usable for efficient computation [LBG15, p. 440], CNNs did not gain real momentum until 2012. At that time Krizhevsky, Sutskever, and Hinton [KSH12] introduced a deeper CNN architecture, later known as AlexNet, trained with 1.2 million labeled images. This large amount of training data was even more increased by applying data augmentation techniques, such as changing the intensity of the RGB channels next to translating and horizontally reflecting the images. As a consequence, one original image serves as multiple, different training images.

Boosted by AlexNet's success, in the following years new CNN architectures emerged in an astounding speed, among them ResNet. Developed by He et al. [He+16], it came in multiple variants with differing numbers of layers. However, generally it followed the tendency towards deeper models, as shown by the comparison in Table 1. It also made use of shortcut or skip connections. Depicted in Figure 5, these take an input x and circumventing a number of layers (two in ResNet), x is added to the in-between layers' output $f(x)$ at that stage. As the skip connection performs an identity mapping, there are no additional parameters introduced and the shortcut connections do not increase the computational complexity. The authors' intent behind adding skip connections was the insight that shallow models sometimes have a lower training error than much deeper models. They suggest that a stack of nonlinear layers struggles to approximate a desired output $\mathcal{H}(x)$, which is similar to the layers' input, i.e. an identity mapping. By adding the input x via a skip connection, the skipped layers now have to approximate the residual function $\mathcal{H}(x) - x$. If $\mathcal{H}(x)$ is indeed close to x, this can be reached by pushing the in-between layers' parameters close to zero.

Transfer Learning. Very deep and large networks usually require a tremendous amount of training data if trained from scratch. Furthermore, for most tasks in the field of image recognition, the lower layers extract basic features and thus the weights from a fine-tuned model can often be reused when training a new network for a different task [Gér19, pp. 345–346]. This technique is called transfer learning and enables training CNNs with only moderate dataset sizes. Often, for similar tasks the entire network barring the output layers is used for initialization [Gér19, p. 346].

2.2 Tasks in the Field of Image Recognition

The area of image recognition is wide. Earlier advances like the LeNet-5 architecture by Lecun et al. [Lec+98] were used by banks for example, to recognize hand-written digits [Gér19, p. 447]. From the field of image classification, the more demanding tasks of object detection, semantic segmentation and instance segmentation emerged, depicted in Figure 6. These are important to distinguish in terms of their goals, capabilities and limitations and thus their relevance for the problem at hand. It should be stressed how and where those disciplines are connected and interleaved, thereby enabling the transfer of established solutions to neighboring fields.

2.2.1 Image Classification

Classification is one of the basic tasks in machine learning [Gér19, p. 85] and aims at assigning one or multiple classes to an instance, as shown in Figure 6a. In terms of image recognition this means classifying an image in its entirety and attributing it to a specific label [WS19, p. 2]. While traditional machine learning techniques that neither rely on CNNs nor ANNs can produce good results in image classification [XRV17, pp. 3–6], here it is only looked upon from a deep learning perspective as this is predominantly used for more difficult tasks in image segmentation [SSD20, p. 2].

In a neural network for multinomial classification, an output layer with one neuron per class is needed. With the softmax activation function, each output neuron yields the respective class probability and the total of all output values is 1. This is easily verified by the following equation for a class probability $\hat{p}_k$ for one instance $\mathbf{x}$:[2]

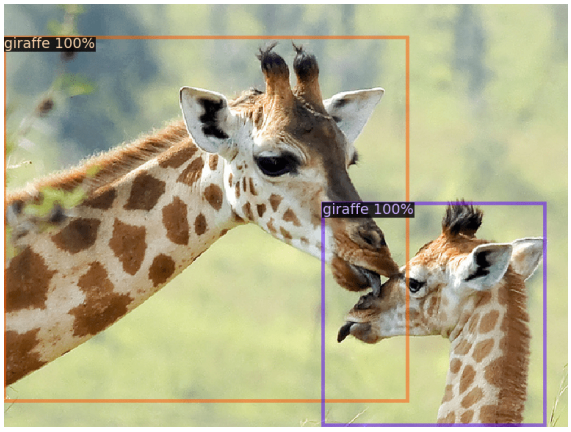
$$\hat{p}_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^{K} \exp(s_j(\mathbf{x}))} \quad \text{where} \quad s_k(\mathbf{x}) = \boldsymbol{\theta}_k \mathbf{x} \text{ [Gér19, pp. 176, 294].}$$

K is the number of classes and $\boldsymbol{\theta}_k$ the parameter vector for class k. However, it is not stated how many objects are in the image or where they are located.

[2] Following, all vectors and matrices are printed in bold.

giraffe

(a) Image Classification

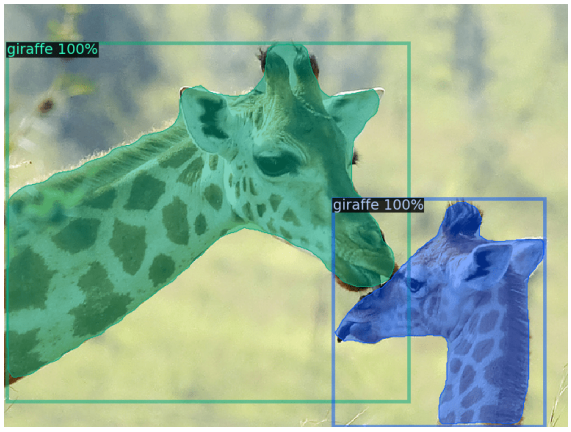


giraffe, giraffe

(b) Object Detection

giraffe, tree

(c) Semantic Segmentation



giraffe, giraffe

(d) Instance Segmentation

Figure 6 There are several different tasks in the field of image recognition. The image is from the COCO dataset (`https://cocodataset.org/#explore?id=191675`).

2.2.2 Object Detection

When performing object detection, the goal is to locate and classify all objects within an image. The location is indicated by a bounding box around the object [WS19, p. 2]. This can be seen in Figure 6b. Thus, a second output layer with 4 neurons that either predict the x and y coordinates of the box center or those of a corner along with its width and height is necessary [Gér19, pp. 483–484]. In order to detect multiple objects, different deep learning techniques are available that either follow a two-stage detection or one-stage detection approach [Zou+19, p. 3].

Two-stage detectors. Two-stage detectors generate a number of region proposals, also called regions of interest (RoI) in the first stage [WS19, pp. 2–3]. One of the earliest representatives – R-CNN (Region-based convolutional neural network) – uses selective search for region proposals [Gir+14, p. 581], a non-learnable algorithm that splits the image into small regions, scans them for potential objects based on similarity

metrics and then iteratively groups them into bigger regions to make proposals for differently sized objects [San+11, pp. 1881–1882]. As every RoI has to be processed by a CNN for feature extraction, this is relatively slow and Spatial Pyramid Pooling Networks (SPPNet), a subsequent model, led to speed enhancements by introducing a special pooling layer to forward the whole image just once to the CNN [He+15]. In the second stage, the RoIs' features are extracted and are then passed on for classification. Additionally, the bounding box location along with its width and height is predicted [WS19, p. 3]. While R-CNN uses linear support vector machine (SVM) classifiers and regressors [Gir+14, p. 581], its successor Fast R-CNN adds pooling, fully connected, softmax and regression layers instead and thus solely relies on neural networks [Gir15, pp. 1441–1442]. However, Fast R-CNN still expects region proposals to be created by selective search (or any other region proposal algorithm). This being a performance bottleneck and a hindrance for an end-to-end model training, Faster R-CNN introduced a region proposal network (RPN) where RoIs are generated as anchor boxes within the CNN [Ren+15, pp. 2–3].

Single-stage detectors. Introduced by the YOLO architecture (You Only Look Once), one-stage models do not use RoIs. Usually, this makes them orders of magnitude faster than two-stage methods [Red+16, pp. 783–784]. In YOLO for example, the whole image is processed by a CNN and divided into parts, each of which gets a fixed number of bounding box predictions and class probabilities [Red+16, p. 780]. In return, one-stage detectors used to lag behind in accuracy, a gap partly bridged by Lin et al. [Lin+20] with RetinaNet and a tweak in the loss function. They identified the imbalance between many regions that do not contain objects and only a few that do as a problem. This leads to many easy, negative training examples which contain an object with a very low probability. In response, they put more weight on hard examples starting from the binary cross entropy loss (also known as log loss) seen in the equation below where $\hat{p}$ is the predicted probability for an object to belong to a given class.[3]

$$\mathrm{CE}(p_t) = -\log(p_t) \quad \text{with } p_t = \begin{cases} \hat{p} & \text{if } y = 1 \\ 1 - \hat{p} & \text{otherwise} \end{cases}$$

The addition of the hyperparameter γ and the modification of the function as shown in the equation below moderates the cross entropy term and the higher the probability associated with a prediction is, the more its term is reduced and the lower is its impact on the loss function.

$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} \log(p_t)$$

Still, in object detection a trade-off between accuracy and speed remains [RF18, p. 1].

[3] For simplicity, the case of binary classification is considered. For multinomial classification this can be extended to categorical cross entropy.

Intersection over Union. While the Mean Squared Error (MSE) is the traditional metric for regression tasks, it is not very well suited for bounding box regression. As an example, take a ground truth box and a slightly smaller predicted bounding box. Clearly, if the prediction for the corner coordinates is off by a certain amount, it is better if the predicted box is lying entirely within the ground truth box than partially outside of it. However, this is not reflected in the MSE. Therefore, the default measure in object detection is Intersection over Union (IoU) [Gér19, p. 485]. As the name implies, the intersection area of the ground truth box and the predicted bounding box is taken and divided by their union area. This can be formulated as

$$\text{IoU} = \frac{\text{true positives}}{\text{true positives} + \text{false positives} + \text{false negatives}}.$$

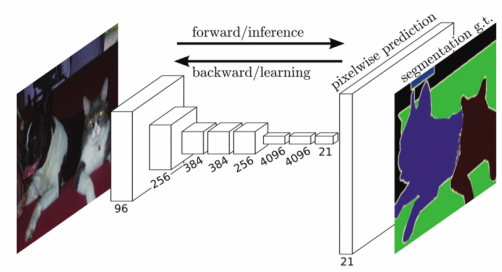
Thus, it is simply a ratio between 0 and 1 and resistant to differently sized objects.

2.2.3 Semantic Segmentation

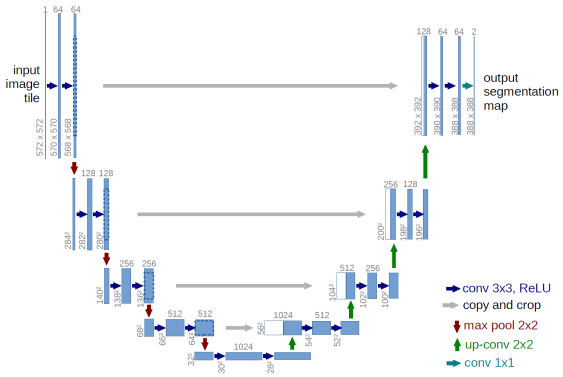
In semantic segmentation, classification takes place on a pixel level, meaning each pixel is assigned to the class of the object it belongs to (see Figure 6c), but without distinguishing between single object instances [Gér19, p. 492]. Semantic segmentation has been tackled for a long time with traditional machine learning techniques before deep learning methods based on CNNs superseded them [Gar+17, p. 1].

Fully Convolutional Network. Long, Shelhamer, and Darrell [LSD15] laid the basis for today's state of the art semantic segmentation models [Gar+17, pp. 8–9]. They built upon existing classification CNNs and by exchanging the dense layers for convolutional layers[4], they constructed a fully convolutional network (FCN), whose architecture can be seen in Figure 7a. This allows for a flexible image input size but due to the applied stride over multiple layers, the output feature map is too small, so that results for a semantic segmentation model would be too coarse. Consequently, they add an upsampling step that restores the original resolution by performing a so-called backwards convolution or deconvolution. While simple interpolation for the additional pixels is the most intuitive solution, the deconvolution filter is learnable, thus retaining the potential for better results. Still, as they have to increase the resolution by a factor of 32 to match the input size, there is a loss of detail in the upsampled image. To mitigate this effect, they only double the resolution and before increasing it further by the factor of 16, they add the output of a lower layer with that respective size via a skip connection. This retrieves some spatial information.

[4] For this conversion the convolutional layer's amount of filters must be identical to the number of neurons in the dense layer and the filter size must match the size of the input feature map [Gér19, pp. 487–488].



(a) FCN: The architecture consists of a set of convolutional and pooling layers and a singular upsampling step. Image from [LSD15, p. 3431]



(b) U-Net: Output feature maps of the convolutional layers are used during upsampling via a skip connection. Image from [RFB15, p. 235]

Figure 7 The FCN and U-Net architectures primarily differ in the decoding phase.

U-Net. The first stage of the FCN pipeline where the image is processed by the convolutional and pooling layers is also known as encoding while the upsampling is called decoding. Especially the latter has been further optimized by subsequent models. Ronneberger, Fischer, and Brox [RFB15] introduced U-Net which modifies the decoding path so that it is symmetric to the encoding path (thus resembling the letter U). For each pooling step (depicted by the red arrows in Figure 7b) in the encoding phase where the resolution is halved and the number of channels doubled, there is a counterpart in the decoding phase (depicted by the green arrows) where the resolution is doubled and the number of channels is halved. Additionally, in the decoding phase the output is combined with the corresponding feature map from the encoding phase via a skip connection to restore spatial details (depicted by the gray arrows). Then again, also U-Net has been extended in several ways and had a particularly large impact in medical image analysis [HZG20, p. 306].

2.2.4 Instance Segmentation

Instance segmentation combines the goals of object detection and semantic segmentation by performing pixelwise classification and distinguishing individual objects [Gér19, p. 495]. For each instance, a bounding box and a per pixel segmentation mask is predicted. Consequently, a combination of methods from object detection as well as semantic segmentation can be found in many instance segmentation models.

Under the term *simultaneous detection and segmentation* (SDS) this problem was initially introduced by Hariharan et al. [Har+14] who started with generating region, i.e. mask proposals and then performing feature extraction and classification on those proposals. Next to this bottom-up mask proposal framework, different approaches evolved which Hafiz and Bhat [HB20] classified into three additional categories. *De-*

tection followed by segmentation models first generate bounding boxes and segment the object masks within those, while models falling into the category *labeling pixels followed by clustering* first classify individual pixels before they are then clustered and separated into object instances. Lastly, they make out *dense sliding window methods* that identify instances by looking at image patches and sliding across the image. A more detailed instance segmentation model overview is conducted in Chapter 3 to determine a suitable model for the problem at hand.

Evaluation metrics. For the evaluation of instance segmentation models, the most prevalent metric to be found is average precision (AP) or rather mean average precision (mAP). It is calculated for the bounding boxes (this metric is also used in object detection) and for the segmentation masks. Following, the process of calculating average precision is outlined for the bounding boxes as described in Elgendy [Elg20, p. 292]. Each bounding box has an associated objectness score, that is a probability for the box to actually contain an object. For varying score thresholds at which a box is predicted to contain an object, the precision (p) and recall (r) are determined for each class (k) separately, thereby yielding the precision/recall curves ($p(r)_k$). Now, for each class the AP corresponds to the area under its curve (AUC). The mAP is simply the average of the class-specific APs over all classes (K).

$$\text{mAP} = \frac{1}{K} \sum_{k} \int_{0}^{1} p(r)_k dr \text{ (adapted from [SYZ13, p. 5])}.$$

Additionally, the mAP is often calculated for different IoU thresholds, meaning the minimum IoU that is still considered a correct prediction. For challenges based on *Common Objects in Context* (COCO) – a dataset containing more than 330,000 images showing objects from 80 different categories – the mask mAP is calculated for IoU values between 0.5 and 0.95 with a step size of 0.05 [COC22]. The average of those mAPs is then reported as the primary challenge metric.

3 Comparison of Instance Segmentation Networks

To approach the present task of segmenting the individual trees in the Rwandan satellite imagery dataset, an important step is to choose a suitable model to be applied. Consequently, in this chapter different instance segmentation models are introduced, examined and finally assessed with respect to the problem.

Selection Procedure. The following comparison and selection consists of three stages. Firstly, in a screening process potential instance segmentation approaches are garnered and a preselection is conducted, leading to a reduced number of models to further investigate. In a second step, each pre-selected model is then explained in more detail. Lastly, tying back to the particular requirements of the task, the models' respective strengths and shortcomings are highlighted and a selection is made for implementation.

3.1 Model Screening

Related Work. As described in Chapter 2.2.4, the general task of instance segmentation was put forward by Hariharan et al. [Har+14]. Since then, a number of instance segmentation models have been developed, often building upon existing solutions for object detection and semantic segmentation. In 2020, these have been surveyed by Hafiz and Bhat [HB20]. Another similar but more recent review has been published by Gu, Bai, and Kong [GBK22] in 2022. Besides scanning those surveys for potential instance segmentation models, papers with a focus on a similar task, that is instance segmentation on tree canopies, are examined and their deployed models are taken into consideration. Notably, this includes the work of Sun et al. [Sun+22] who delineated trees in an urban context in Guangzhou, China and the work of Brandt et al. [Bra+20] who segmented trees in the Sahara and the Sahel region. Likewise, albeit with a different focus on dimensionality reduction techniques, Xi et al. [Xi+21] identified individual ginkgo tree crowns using different instance segmentation networks. Ocer et al. [Oce+20] detected trees on a campus area as well as a lake region.

In Chapter 2.2.4, one taxonomy of instance segmentation methods was already briefly described. However, its focus is on the semantic level, describing what is being done, e.g., *detection followed by segmentation*. Here, to classify the different models, a more structural taxonomy based on Gu, Bai, and Kong [GBK22] is used. On a top level, this translates to the categories single-stage methods, two-stage methods, multi-stage methods and semi and weakly supervised methods. The latter however are not considered hereafter since the training data at hand is sufficiently annotated.

Category	Model	Source
Two-stage methods	DeepMask	[HB20], [GBK22]
	DeepSnake	[GBK22]
	Deep Watershed Transform (DWT)	[HB20], [GBK22]
	Discriminative Loss Function (DLF)	[HB20], [GBK22]
	Dynamically Instantiated Network (DIN)	[HB20], [GBK22]
	Hypercolumns	[HB20], [GBK22]
	MaskLab	[HB20], [GBK22]
	Mask R-CNN	[HB20], [Xi+21], [GBK22]
	Mask Scoring R-CNN	[HB20], [GBK22]
	Pixel-level Encoding and Depth Layering (PEDL)	[HB20], [GBK22]
	SharpMask	[HB20], [GBK22]
	U-Net[6]	[Bra+20]
Multi-stage methods	Cascade Mask R-CNN	[HB20], [Sun+22], [GBK22]
	Hybrid Task Cascade (HTC)	[HB20], [GBK22]
	Mask2Former	[GBK22]
	Multi-task Network Cascades (MNC)	[HB20], [GBK22]
	Queryinst	[GBK22]
	SOLQ	[GBK22]
Single-stage methods	BlendMask	[HB20], [Xi+21], [GBK22]
	CenterMask	[HB20], [GBK22]
	CondInst	[GBK22]
	Fully convolutional instance-aware semantic segmentation (FCIS)	[HB20], [GBK22]
	InstanceFCN	[HB20], [GBK22]
	PolarMask	[HB20], [GBK22]
	SOLO	[HB20], [GBK22]
	SOLOv2	[HB20], [GBK22]
	TensorMask	[HB20], [GBK22]
	YOLACT	[HB20], [GBK22]

Table 2 Since 2014, a palette of instance segmentation networks has been developed.

To limit the amount of models to choose from, only those that are either covered in both instance segmentation surveys by Hafiz and Bhat [HB20] and Gu, Bai, and Kong [GBK22] or those that are used in one of the mentioned tree instance segmentation papers are regarded.[5] The results of the collection process are condensed in Table 2.

After Gu, Bai, and Kong [GBK22] the two-stage methods separately perform bounding box detection and mask segmentation and can be further divided into top-down

[5] Models which were only recently developed and thus not included in the survey of Hafiz and Bhat [HB20] are included as well.

[6] The semantic segmentation model was extended with pixel-wise weight maps with higher weights at canopy gaps to delineate the instances.

and bottom-up approaches. Depending on the exact architecture multi-stage methods are defined by more than two stages, e.g., instead of producing classified bounding boxes in the first and mask segmentations in the second stage, class-agnostic bounding box regression and mask generation mark the first and second stage, while classification marks the third [DHS16, p. 3150]. At last, in single-stage detectors pixel classification and the localization of an object are not performed sequentially but simultaneously in one stage.

Making a preselection. As an encompassing assessment of all models listed in Table 2 would exceed the scope of this thesis, only a small selection – that is Mask R-CNN, Cascade Mask R-CNN and BlendMask – is explored further and presented in more detail. These three are selected because they have been applied to similar tasks. Mask R-CNN is one of the most widely known networks and has become the benchmark for other instance segmentation models over the past years [GBK22, p. 9]. Moreover it was applied by Xi et al. [Xi+21] to segment ginkgo trees on images taken with an unmanned aerial vehicle (UAV) and by Ocer et al. [Oce+20]. Cascade Mask R-CNN was successfully used by Sun et al. [Sun+22] to delineate tree canopies in Guangzhou. As they coped with the most similar task, Cascade Mask R-CNN is also taken into consideration. Lastly, BlendMask is one of the best-performing single-stage instance segmentation models on COCO [GBK22, p. 22] and was also put to use by Xi et al. [Xi+21]. Thus, for each of the three categories single-stage, two-stage and multi-stage networks, one method will be covered in the remainder of this chapter.

3.2 Mask R-CNN

Building atop the Faster R-CNN architecture and adding a branch for mask segmentation, He et al. [He+17] proposed Mask R-CNN in 2017. It follows a top-down approach by using RoIs that are then segmented. Thus, to properly understand Mask R-CNN, apart from the mask segmentation itself, it is crucial to fully comprehend the steps of RoI generation, feature extraction, classification and bounding box regression in Faster R-CNN, the general setup of which is described in Chapter 2.2.2. The complete Mask R-CNN architecture is depicted in Figure 8 and is referenced to provide an understanding of the network's structure as a whole.

Backbone CNN. All upcoming steps such as generating region proposals and locating bounding boxes are dependent on the backbone CNN for feature extraction. While Mask R-CNN can be instantiated with different backbones, He et al. [He+17] proposed a ResNet architecture (with both 50 and 101 layers) enriched by a Feature Pyramid Network (FPN). The latter was presented by Lin et al. [Lin+17] and it

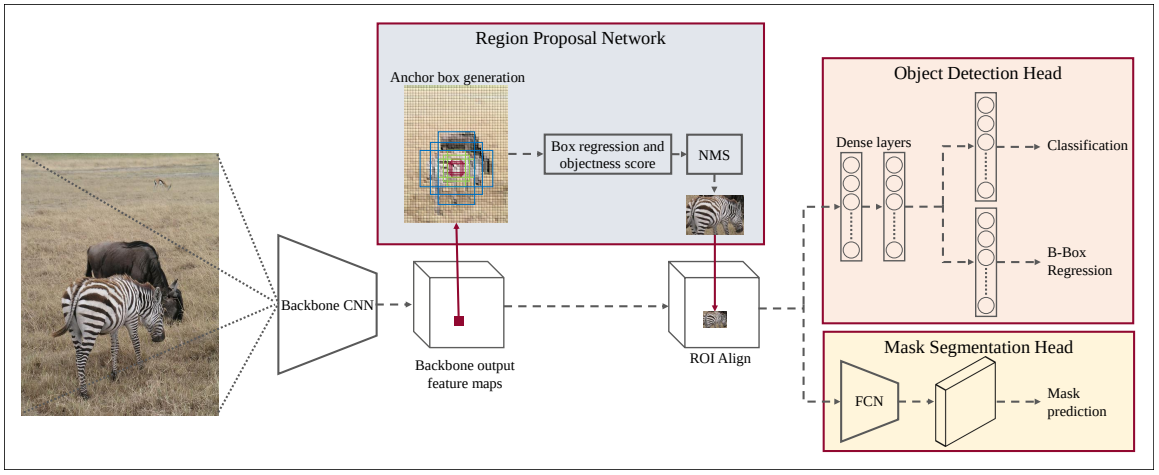


Figure 8 Mask R-CNN extends the Faster R-CNN architecture with a mask segmentation head and exchanges the RoI pooling layer for an RoIAlign layer. The sample image is from the COCO dataset (`http://cocodataset.org/#explore?id=282957`)

exploits the pyramidal structure of the feature maps in a CNN that arises from the applied strides. First, in a bottom-up pathway each layer which is directly located before reducing the feature map size (either through pooling layers or convolutional layers with a stride greater than one) is selected, thereby obtaining one layer of every size. Afterwards, in a top-down pathway each selected layer is upsampled to the size of the next larger layer and a lateral connection that adds the upsampled layer to the equally sized layer from the bottom-up pathway is applied. Finally, the authors feed each result from the lateral connections to a 3x3 convolutional layer. The intent for this procedure can be affiliated to deeper, smaller feature maps being semantically stronger in that a more complex, meaningful set of features is extracted. By combining them with the larger output feature maps, high-resolution layers with stronger semantic features are created.

Region Proposal Network. Given an input of arbitrary size, output feature maps are generated by the backbone CNN, which are then fed to the RPN to generate RoIs. Originally proposed for Faster R-CNN by Ren et al. [Ren+15], a sliding window of size n x n is shifted across the output feature maps. As the feature maps are much smaller than the input image, the receptive field of the sliding window corresponds to a larger area with respect to the input image. For each sliding window position, a set of k anchor boxes is generated. Ren et al. [Ren+15] set k = 9 and use three different anchor sizes with three different aspect ratios to capture differently sized objects. The spatial features within the receptive field of the sliding window serve as input for a subsequent mini-network that classifies each anchor whether it contains an object or not and predicts an anchor box offset. Thus, it outputs $2k$ scores for classification and $4k$ values that are the offset coordinates as well as the width and height of the anchor boxes. As the anchors overlap to a large degree, the authors

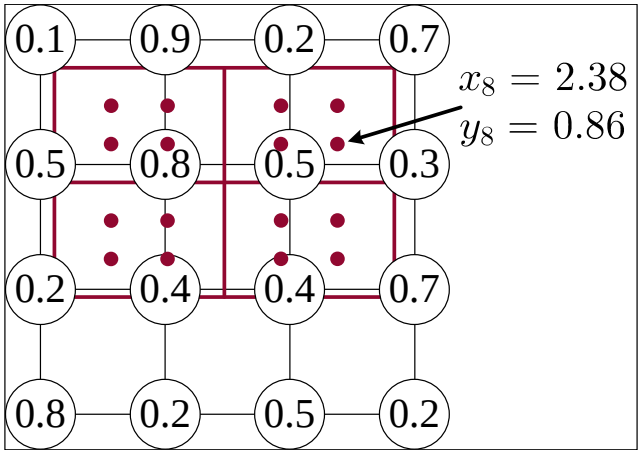


Figure 9 In RoIAlign, each sample point's value is computed via bilinear interpolation.

use non-maximum suppression (NMS) to remove redundant RoIs. From all anchor boxes that overlap with each other with an IoU greater than 0.7, only those with the highest class score remain while the rest is removed. Anchors can also be assigned with a positive label if either the respective anchor has the highest IoU out of all anchors with a ground truth box or if it has an IoU greater than 0.7 with any ground truth box. Alternatively, a negative label is assigned if its highest IoU with a ground truth box is below 0.3. Otherwise no label is assigned.

RoIAlign. The object detection head, which is also used in Fast and Faster R-CNN, involves fully connected layers and thus requires fixed-size inputs [Gir15, p. 1441]. Meanwhile, the input image can be of arbitrary size and to further process the anchor boxes generated by the RPN, a respective layer to adjust the size is needed. He et al. [He+17] use an RoIAlign layer, omitting the RoI pooling layer used in Fast and Faster R-CNN. Both output a feature map of fixed size H x W by mapping the anchors to the output feature maps from the backbone CNN (their size is denoted as H' x W'), dividing the mapped part into a H x W grid and performing max pooling within each grid cell. The authors use RoIAlign because the anchor boxes from the RPN refer to coordinates from the input image but due to the applied stride throughout the network, a conversion of the coordinates onto the feature maps leads to floating point numbers. While RoI pooling quantizes them via rounding to align the locations, thereby deviating from the original coordinates, RoIAlign uses bilinear interpolation. Taking four sample points per grid cell (as indicated by the red points in Figure 9), a weighted average of the neighboring feature map values is computed for each point to obtain the value at that location. This can be formulated as

$$V_i^c = \sum_n^{H'} \sum_m^{W'} I_{nm}^c \max(0, 1 - |x_i - m|) \max(0, 1 - |y_i - n|) \text{ (adapted from [Jad+15, p. 5]).}$$

The output V_i^c, where i is the point and c the feature map index, is thus calculated by weighting the input feature map values I_{nm}^c according to their distance from the point in question. The process is exemplified for an output feature map with a spatial extent of 2 x 2 in Figure 9, where $V_8^d = 0.2 \cdot (1 - |2.38 - 2|) \cdot (1 - |0.86 - 0|) + \ldots + 0.3 \cdot (1 - |2.38 - 3|) \cdot (1 - |0.86 - 1|)$. Per grid cell He et al. [He+17] decide the eventual output value among the four sample points via max pooling. Consequently, RoIAlign layers do not introduce misalignment which is crucial for pixel-accurate mask prediction.

Object Detection Head. The fixed-size output of the RoIAlign layer is put through some fully connected layers and then fed into two sibling layers as described by Girshick [Gir15]. The first one uses a $K + 1$-softmax layer for classification with K being the number of object classes and one additional background class (for details on the softmax function see Chapter 2.2.1). In parallel, a layer for bounding box regression outputs four box position values as specified in Chapter 2.2.2. As this happens simultaneously to the classification, K bounding boxes are predicted for each RoI, one for each class, the final selection pending on the classification result.

Mask Segmentation Head. Alongside the object detection head, He et al. [He+17] introduce a mask prediction branch. Again, due to this being independent from the class, the output are K binary masks. Each mask is predicted as an output of m x m for its spatial extent. Here, no dense layers are used but a fully convolutional network as described in Chapter 2.2.3.

Training. When it comes to training Mask R-CNN, the authors use a multi-task loss for each RoI:

$$L = L_{cls} + L_{box} + L_{mask}.$$

With L_{cls} and L_{box} identical to Girshick's [Gir15] Fast R-CNN, it holds

$$L_{cls}(\hat{p}, u) = -\log(\hat{p}_u),$$

i.e. it is the log loss of the predicted class value $\hat{p}$ for the true class u. The box loss is defined as

$$L_{box}(\hat{\mathbf{t}}^u, \mathbf{t}) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(\hat{t}_i^u - t_i).$$

For each class k a four-tuple of predicted bounding box values $\hat{\mathbf{t}}^k = (\hat{t}_x^k, \hat{t}_y^k, \hat{t}_w^k, \hat{t}_h^k)$ exists. $\hat{\mathbf{t}}^u$ is the tuple for the true class u and $\mathbf{t}$ is the corresponding ground truth box tuple. The $\text{smooth}_{L_1}(x)$ is defined as

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise.} \end{cases}$$

L_{box} is thus only defined for box proposals that were made for the ground truth class u. For the mask loss, the authors state that it is the average binary cross-entropy loss of a per-pixel sigmoid function and only defined on the mask for the ground truth class u. Bearing in mind that each mask is a binary output of size m x m, this can also be written as

$$L_{mask}(\hat{\mathbf{y}}^u, \mathbf{y}^u) = -\frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_{ij}^u \log(\hat{y}_{ij}^u) + (1 - y_{ij}^u)\log(1 - \hat{y}_{ij}^u),$$

where $\hat{y}_{ij}^u$ is the value of the sigmoid function (see Chapter 2.1.1) at the position (i, j) for the mask that is associated with the ground truth class u. However, as Ren et al. [Ren+15] point out for Faster R-CNN, the head branches rely on the RoIs from the RPN as input. Thus the RPN needs training as well. The loss function

$$L(\hat{p}_i, p_i, \hat{\mathbf{t}}_i, \mathbf{t}_i) = \frac{1}{N_{cls}}\sum_i L_{cls}(\hat{p}_i, p_i) + \lambda\frac{1}{N_{reg}}\sum_i p_i L_{reg}(\hat{\mathbf{t}}_i, \mathbf{t}_i)$$

is similar to those used in the object detection head. Here, i is the anchor index, $\hat{p}_i$ the predicted objectness score and p_i the ground truth label which is 1 if the anchor is positive and 0 if it is negative (non-labeled anchors do not contribute to the loss). L_{cls} is again the binary cross entropy loss (see Chapter 2.2.2). L_{reg} is identical to L_{box} and $\hat{\mathbf{t}}_i$ and $\mathbf{t}_i$ are specifically parameterized versions[7] of the predicted anchor box and ground truth box respectively. Both terms are normalized with N_{cls} and N_{reg} and balanced with λ. At first, Ren et al. [Ren+15] train the RPN and the head branches separately on a pre-trained backbone to optimize region proposals and detection. Then the RPN is incorporated in the detection network and the backbone layers are fixed to only fine-tune the RPN layers.

3.3 Cascade Mask R-CNN

Claiming that one difficulty of box regression is the fine-grained boundary of correct and incorrect predictions determined by an IoU threshold, and a problem of two-stage methods are thus many close false positives, that is bounding boxes that are almost but not actually correct, Cai and Vasconcelos [CV19] proposed Cascade R-CNN for object detection and Cascade Mask R-CNN for instance segmentation. They further argue that detectors achieve best results for IoU thresholds close to the threshold that was used for training the detector. However, they note that simply increasing the IoU threshold at training time to get a detector that produces high-quality detections leads to exponentially fewer positive training examples, for only a small number of such RoIs exist and therefore makes the detector prone to overfitting. The cascading

[7] The exact parameterizations can be found in Ren et al. [Ren+15, pp. 3–4].

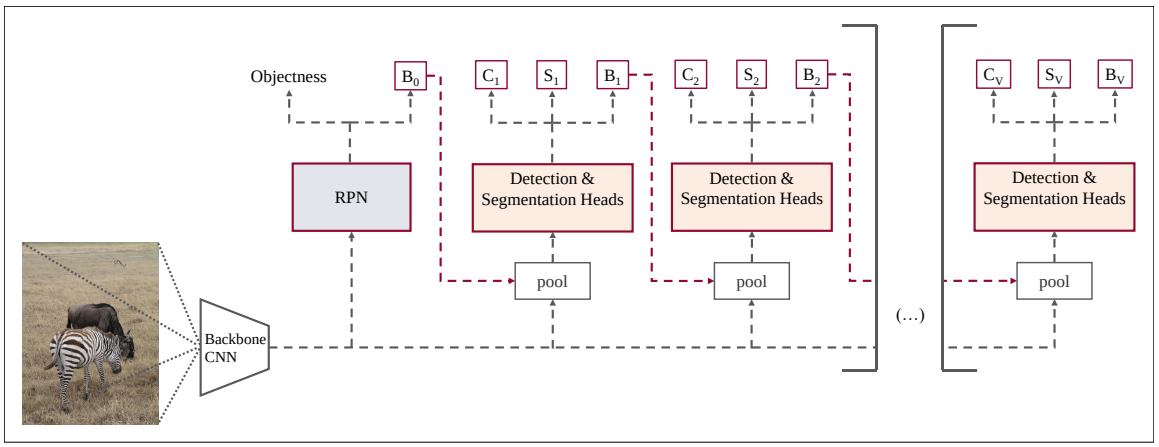


Figure 10 In Cascade Mask R-CNN, a sequence of bounding box regressors is used with different possible design choices when it comes to mask segmentation. Adapted from Cai and Vasconcelos [CV19, pp. 1486, 1489].

structure they propose extends Faster R-CNN and is built of a series of sequenced detectors that use the output of the previous stage and become more selective towards later stages. This rests on the assumption that a bounding box regressor's output IoU of predicted boxes and the ground truth are usually better than the input IoU, i.e. it improves the bounding box regression. Following, the architecture is explained, focusing on how the cascading structure is used for bounding box regression and how this is extended to mask segmentation.

Cascade R-CNN Architecture. The RPN is adopted from Faster R-CNN as described in Chapter 3.2. Then, bounding box regressors are chained, thereby forming a cascade:

$$f(\mathbf{x}, \hat{\mathbf{t}}) = f_V \circ f_{V-1} \circ \ldots \circ f_1(\mathbf{x}, \hat{\mathbf{t}}).$$

With V being the total number of regressors, $\mathbf{x}$ the input image patch and $\hat{\mathbf{t}}$ the anchor box prediction coming from the RPN, each regressor f_v is trained on the output of the previous regressor f_{v-1} instead of on the original input itself. Since the regressors become gradually more selective about what IoU is considered a positive prediction and generally improve the IoU compared to their input's IoU, the number of positive training examples is supposed to remain constant when moving towards the later regressors.

Cascade Mask R-CNN Architecture. Similar to Mask R-CNN, Cascade Mask R-CNN also adds a mask segmentation branch to its object detection network. An overview of the architecture can be seen in Figure 10. In there, B represents the bounding box output, C the classification output and S the mask segmentation output. Cai and Vasconcelos [CV19] note that as there are multiple object detection branches in Cascade R-CNN, there are also multiple possible positions at which a

mask segmentation branch could be added during training. Consequently, they propose three different approaches, one with mask segmentation only added at the first stage (S_1 in Figure 10), one with it only added at the last stage (S_V) and one where every stage is extended with a mask segmentation branch (S_1 up to S_V). The decision of whether mask segmentation is added at the first stage or the last stage has an impact on the number of training examples it receives. While the authors point out that there are more positive training examples in the last stage due to the increasing IoU, these also highly overlap. However, during inference mode mask segmentation always utilizes the bounding boxes from the last cascade stage regardless of what stages might be used in training.

Training. To train the detectors at each stage v, the loss function

$$L(\mathbf{x}^v, \mathbf{t}) = L_{cls}(h_v(\mathbf{x}^v), u^v) + \lambda \left[u^v \geq 1\right] L_{loc}(f_v(\mathbf{x}^v, \hat{\mathbf{t}}^v), \mathbf{t})$$

is employed. Each stage's classifier h_v and regressor f_v take the input image batch $\mathbf{x}^v$ and the regressor additionally $\hat{\mathbf{t}}^v$ which is $f_{v-1}(\mathbf{x}^{v-1}, \hat{\mathbf{t}}^{v-1})$, i.e. the box prediction of the previous regressor. The indicator function $[\cdot]$ turns 0 if u is 0, i.e. the background class, leading to L_{loc} being ignored. λ is again used to balance the two losses. L_{cls} and L_{loc} are adopted from Fast R-CNN and thus alike with L_{cls} and L_{loc} from Mask R-CNN. To combine the losses of all the different stages, the authors propose two alternative strategies: *avg* and *decay*. With *avg*, every stage's loss is weighted equally ($w_v = 1/V$) while the *decay* strategy puts higher weights on earlier stages, reducing them exponentially ($w_v = 1/2^{v-1}$).

3.4 BlendMask

As a single-stage detector, BlendMask does not separate its stages of detection and segmentation and also does not rely on generating RoIs. Presented by Chen et al. [Che+20], they build upon the FCOS object detector and add a bottom module that generates score maps and a single convolution layer that predicts so-called attention masks. Both are then put into a blender that combines score maps and attention masks into the final mask predictions. The different parts, that is the FCOS detector, the bottom module, the top layer and the blender are explained more extensively in the following paragraphs, whereas Figure 11 supports in relating to the overall network structure.

FCOS Object Detector. Tian et al. [Tia+19] presented FCOS, a one-stage and thus proposal-free network that performs anchor-free object detection. The latter also differentiates it from other one-stage detectors such as YOLOv3. Instead, they pursue

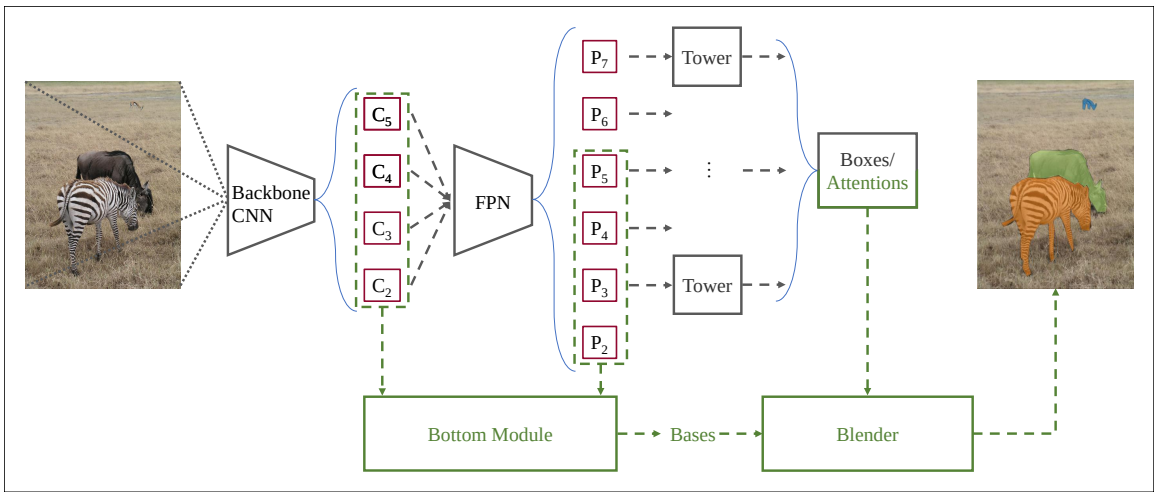


Figure 11 BlendMask extends the FCOS detector (depicted in darkgray) by a bottom module, attention maps and a blender (depicted in green). Adapted from Chen et al. [Che+20, p. 8572].

per-pixel prediction drawing from the idea of the FCN (see Chapter 2.2.3). Following a backbone CNN, a bounding box is regressed for each location in the output feature maps. Thus, the location itself is seen as a training sample, contrasting anchor-based methods where anchor boxes are proposed at each location which are then further regressed. A location is considered a positive sample if it lies within any ground truth box, choosing the box with the smallest area as the regression target in case of overlapping instances. For each location a 4D vector $\hat{\mathbf{t}}$ is predicted, though not with coordinates, width and height but with offsets to the left, top, right and bottom borders of the bounding box. Additionally, a binary classifier per class is used for classification. One arising problem is that due to the large stride of the backbone CNN, small objects might not be captured since no locations in the output feature maps point to those objects when mapped back to the input image. In response, the authors propose multi-level prediction using a FPN backbone (described in Chapter 3.2) and starting from each of the differently sized output feature maps with a distinct target object size range per feature map. To prevent low-quality predictions, NMS is applied and each location's prediction is enriched by a centerness score (a score between 0 and 1, indicating how close the location is to the object's center) that is multiplied with its classification score to get a final score for each prediction.

Bottom Module. For BlendMask, Chen et al. [Che+20] add a bottom module that takes output feature maps from the FCOS detector as inputs which are either those from the backbone of FCOS (C_2 to C_5 in Figure 11) or those outputted by the FPN (P_2 to P_5 in Figure 11). The bottom module generates a set of G score maps per input image which the authors call bases (B). Score maps are prototype masks that span the entire image and thus similar to what is being done in semantic

segmentation, albeit there is no dedicated loss for the score maps but only the final mask loss serves as supervision [Bol+19, p. 9158]. In BlendMask, the G bases have a size of $\frac{H}{s}$ x $\frac{W}{s}$ each, where H and W are the input image's width and height and s is the score map output stride.

Top Layer. Apart from the bottom module which uses feature maps from the FCOS backbone or the FPN, Chen et al. [Che+20] stack a single convolution layer atop each detector. Their final outputs, that is the bounding boxes, are thereby enriched with attention maps. These encode coarse instance information, such as its shape or pose. The convolution layer has $G \cdot M \cdot M$ output channels (M = attention map resolution) and a feature map size of H_l x W_l which is unchanged from that of the previous layer l. As the attention maps should only roughly estimate the instance, M is usually smaller than the mask resolution in e.g. Mask R-CNN. Then the top D box predictions P along with the corresponding attentions A are selected for further processing.

Blender Module. Arguably the most important tweak in Chen et al.'s [Che+20] BlendMask is the blender module. Taking the bases B, the attentions A and the bounding box proposals P as inputs, the bases are cropped with each box proposal and given into a RoIAlign layer (see Chapter 3.2) to generate a fixed size feature map r_d of size R x R. Then, as R is larger than M, the attentions are upsampled to size R x R via interpolation and normalized with the softmax function. With the normalized attentions s_d and the cropped regions r_d the mask is calculated as

$$m_d = \sum_{g=1}^{G} s_d^g \circ r_d^g \quad \forall d \in \{1 \dots D\}$$

where $\circ$ is the element-wise product between s_d and r_d. Thus, the sum over all G bases produces the final mask.

Training. Tian et al. [Tia+19] use the loss function

$$L(\hat{\mathbf{p}}_{x,y}, \hat{\mathbf{t}}_{x,y}) = \frac{1}{N_{\text{pos}}} \sum_{x,y} L_{\text{cls}}(\hat{\mathbf{p}}_{x,y}, u_{x,y}) + \frac{\lambda}{N_{\text{pos}}} \sum_{x,y} [u_{x,y} > 0] L_{\text{reg}}(\hat{\mathbf{t}}_{x,y}, \mathbf{t}_{x,y}).$$

for every location (x, y) in the output feature maps of the FCOS object detector. N_{pos} is the amount of positive samples while L_{cls} is the focal loss as specified in Chapter 2.2.2 and L_{reg} is the IoU loss. This is defined as

$$L_{\text{IoU}}(\hat{\mathbf{t}}, \mathbf{t}) = -\ln(\text{IoU}(\hat{\mathbf{t}}, \mathbf{t}))$$

by Yu et al. [Yu+16]. There is no specification given about the mask loss being used by Chen et al. [Che+20].

3.5 Model Selection

When looking at the three models described above and deciding which one to use for implementation, a set of factors are considered. Those are the general performance of the model, its specific performance and suitability for tree/satellite instance segmentation and its accessibility in terms of implementation. The inference speed is not a major criterion, as no real-time detection and segmentation is required.

General Performance. The general performance should be considered alongside the specific performance on similar problems, for not all models might have been deployed to similar problems, leaving the general performance as the only indicator for its suitability. Moreover, albeit similar projects offer great insights, it should be noted that even similar projects can vary in their requirements and challenges, thus leaving the need for a general performance evaluation. As elaborated in Chapter 2.2.4, mask AP is the most common metric to compare instance segmentation models. Meanwhile, COCO is the de facto benchmark dataset and all three networks covered in this chapter use it for evaluating their results [He+17, p. 2984], [CV19, p. 1496], [Che+20, p. 8576]. These can be seen in Table 3. AP_S is the average precision for small objects which is particularly relevant for the task at hand due to the small size of individual trees in a satellite image. BlendMask is performing slightly better than Mask R-CNN and Cascade Mask R-CNN, overall as well as on small objects. However, these results should not be treated as immutable, as the results depend on many factors such as backbone, training schedule and augmentations to name only a few. In Facebook AI Research's algorithm library Detectron2 for example, the model zoo has Mask R-CNN models ranging from 32.2 to 43.7 mask AP on COCO [Wu+19].

Task-specific Performance. All three networks have been applied to similar tasks. Sun et al. [Sun+22] deployed Cascade Mask R-CNN for tree instance segmentation. They did not report AP but the F_1-Score of 0.83. The F1 score is defined as

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

and thus the harmonic mean between precision and recall [Gér19, pp. 92–93]. Xi et al. [Xi+21] identified ginkgo trees (having to differentiate them from other tree species) and used BlendMask and Mask R-CNN. Here, BlendMask achieves slightly better results, however it should be noted that the BlendMask model is equipped with a ResNet-101 backbone, while their Mask R-CNN only uses ResNet-50. Mask R-CNN was also used by Ocer et al. [Oce+20] to detect trees on a campus and in a lake region. Only performing tree detection, they report F_1-Scores for the individual test images, ranging from 0.82 to 0.91.

Method	Backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask R-CNN	ResNeXt-101-FPN	37.1	60.0	39.4	16.9	39.9	53.5
Cascade Mask R-CNN	ResNeXt-101-FPN	38.6	60.6	41.5	18.5	41.3	57.2
BlendMask	ResNet-101-FPN	41.3	63.1	44.6	22.7	44.1	54.5

Table 3 The mask AP on the COCO test set. Numbers from Gu, Bai, and Kong [GBK22, pp. 22–23].

Ease of Implementation. Bearing in mind time and resource limitations of this thesis, the ease of implementation of the different models should be considered. There is a large variety of libraries implementing Mask R-CNN. Abdulla [Abd17] implemented a very popular version based on Keras and Tensorflow that has been forked over 10,000 times with 2,500 issues that could support with potential difficulties. As mentioned above, Detectron2 also implements instance segmentation solutions which are all variations of Mask R-CNN [Wu+19]. Based on PyTorch, it has been forked more than 5,000 tines, provides over 3,000 issues and has a sophisticated documentation. MMDetection is an open source project and also based on PyTorch [Che+19]. It implements a variety of instance segmentation networks, among them Mask R-CNN and Cascade Mask R-CNN. It has been forked 7,500 times and over 6,000 issues have been posted. It too comes with a proper documentation. Building atop Detectron2 Tian et al. [Tia+19] implement a number of networks, among them BlendMask. The project has been forked more than 500 times and 500 issues have been posted. In contrast to BlendMask, for Mask R-CNN there are numerous additional resources such as blog posts and tutorials, which can further support in developing the model. To some degree, these also refer to Cascade Mask R-CNN as both networks are so similar.

Selection. All three models show decent performances on COCO with BlendMask performing slightly better than the two R-CNN models. They all have been successfully applied to similar challenges, making all of them reasonable approaches for the problem at hand. However, in terms of implementation accessibility Mask R-CNN in particular, but also Cascade Mask R-CNN have the advantage over BlendMask since there are multiple large libraries implementing them providing detailed documentations and a large set of answered issues. Additionally, since Cascade Mask R-CNN extends Mask R-CNN, their architectures are identical to a large degree and many hyperparameters settings could be used for either Mask R-CNN or Cascade Mask R-CNN. Consequently, in the implementation phase the MMDetection library is used, as it offers Mask R-CNN as well as Cascade Mask R-CNN models and thus both networks can be evaluated on the Rwandan tree dataset.

4 Methodology and Implementation

Looking back at the CRISP-DM model, with the model selection finished and as an understanding of the business objective and the data has been established, the steps of data preparation and modeling have to be realized. In this chapter, the corresponding methodology is described. The data preprocessing steps as well as the general implementation setup is presented. Furthermore, the implementation details regarding Mask R-CNN and Cascade Mask R-CNN are outlined. The code is publicly available at `https://github.com/sbackmann/rwanda-segmentation`.

4.1 Data Preparation

COCO Format. To use the full capabilities of the MMDetection library including the evaluation, it is recommended to convert custom datasets into COCO format. This requires a JSON file containing the keys *images*, *annotations* and *categories*. *Images* is a list with one entry per image, each containing information about its file name, height, width and an image ID. *Annotations* contains all ground truth labels, which are defined by a polygon mask, a bounding box and some additional information such as the image that it refers to. *Categories* contains all the object classes of the dataset. One JSON file is created for the train, validation and test set each. The COCO format is adopted for this project.

Train-Test Split. A train, validation and test set is created following the ratio 60 %, 20 %, 20 %. The images for the different sets are picked by hand since the number of images is rather small and a random split could lead to skewed datasets in multiple ways. First, the images are from different terrains. They cover deserts, bodies of water, cities, wetlands, forests and agriculturally used areas. All of them should be represented in both training and evaluation. Moreover, the images vary greatly in size. Consequently, if one set had some very large images, these would distort the desired split ratio and give few images an overly large impact on evaluation metrics.

Reading Input Data. The satellite images are present in the GeoTiff file format and their size varies from 451 x 258 pixels to 5122 x 3145 pixels. Therefore, and because of the original images containing up to 6,000 trees per image, smaller image patches are created. The minimum width and height for the patches are set to 300 pixels (maximum = 600 pixels) and each image is then divided into evenly sized patches. Images which are smaller than the minimum value are kept as they are. The additional information of the GeoTiffs such as the geo-coordinates is retained for the

image patches. The corresponding labels are all present in one *.shp* file and each label is associated with an ID and stored as series of coordinates which together make up its polygon mask. The labels are processed with the python package *Fiona*, an API for reading and writing geospatial vector data [Gil+11]. As their polygon vertices are denoted as geo-coordinates, the satellite images are read using the module *Rasterio*, a package that enables the processing of geospatial raster data [Gil+13]. Both Fiona and Rasterio hold their read data in a GeoJSON structure, thereby working well together. By using Rasterio, the GeoTiffs' geo-information is retained and the labels can be linked to the images. Labels are only matched to an image if the objects lie fully within the image bounds. To prevent an excess loss of ground truth examples at the edges resulting from the patch creation, the image patches overlap by 20 pixels.

Data Augmentations. The train and test pipelines of the Mask R-CNN models contain some data augmentations, which are kept at default values for the base model. Those default augmentations are shortly described. For training, each image patch is resized to one of six different sizes ranging from 1333 x 640 to 1333 x 800 (picked randomly) and for testing it is always scaled to 1333 x 800. However, the aspect ratio is kept unchanged, meaning the shorter side is scaled to 640 or 800 pixels and the larger side scaled respectively. The remaining transformations are identical for training and testing. The image patch is flipped horizontally with a probability of 0.5. The images are also normalized across the three color channels. For this, the mean RGB values of COCO are used, whereas the standard deviation is set to 1.

4.2 Implementation Details

Apart from choosing the model in general, a number of implementation choices have to be made. A library for implementing the framework has to be chosen, the hyper-parameter settings have to be adapted to the dataset and a training routine needs to be defined. Following, the respective implementation details are described.

Model Library. For the implementation, OpenMMLab's MMDetection as presented in Chapter 3.5 is used. It is based on *PyTorch*[8], an open source framework for machine learning and *mmcv*[9], a library also developed by OpenMMLab that provides basic functionalities for computer vision research like image processing and visualization. The Python package *Pycocotools*[10] enables the usage of the COCO evaluation for automatically calculating mAP. As the dataset contains only one class, mAP and AP are identical, which is why AP is used hereafter. AP is computed by calculating and averaging it over IoU thresholds between 0.5 and 0.95 in steps of 0.05.

[8] `https://pytorch.org/`
[9] `https://github.com/open-mmlab/mmcv`
[10] `https://github.com/cocodataset/cocoapi`

Training. The network is not trained from scratch but initialized with a model pre-trained on COCO (see transfer learning in Chapter 2.1.2). MMDetection features a variety of pre-trained COCO models for both Mask R-CNN and Cascade Mask R-CNN. For this project, the initial model is trained with a Mask R-CNN model based on a ResNet-50 backbone, extended with a FPN. Following Sun et al.'s [Sun+22] approach, the generalized IoU (GIoU) is used as bounding box loss function instead of the smooth_{L_1} loss. Since the IoU value for two non-overlapping objects A and B is zero, regardless of how far they are apart, the gradient of the IoU loss becomes zero as well, thus preventing further optimization [Rez+19, p. 659]. Proposed by Rezatofighi et al. [Rez+19], GIoU extends IoU by determining the smallest possible shape C that encloses A and B. The GIoU is then computed as

$$GIoU = IoU - \frac{|C \setminus A \cup B|}{|C|}$$

and thus takes the IoU minus the ratio of C's area, excluding A's and B's area, and the area of C [Rez+19, p. 660]. For objects with increasing distances, this ratio converges to 1 and the GIoU converges to -1. The model is trained on one Nvidia GeForce RTX 3090. The number of images per GPU is varied between one and two depending on the backbone and the resulting memory usage. The learning rate is set to 0.0025 and reduced by a factor of 10 after 12 and 24 epochs with 36 being the maximum number of epochs[11]. The training losses are logged during training and the model is evaluated every three epochs on the validation set. Checkpoints are created in the same interval, whereas for each configuration, only the checkpoint with the best validation set performance is saved permanently and training is stopped if no improvement on the validation set is achieved over the course of three consecutive epochs. Additionally, the model configuration is saved for future inference.

Hyperparameter settings. As the models were pre-trained on COCO and also the configuration settings are fine-tuned for this and similar datasets, some hyperparameters should be adjusted. First and foremost, since the tree satellite images can contain significantly more objects per image, the maximum number of detections per image is raised from 100 to 1000. Also the number of region proposals is increased from 2,000 to 10,000 (before NMS) and from 1,000 to 5,000 (after NMS). These adjustments are also applied to the evaluation. By default, COCO evaluation computes AP for an amount of 100 maximum detections per image, which is again increased to 1,000. Moreover, AP is computed separately for small, medium and large objects (AP_S, AP_M and AP_L). If the default size thresholds were applied, almost 95 % of all labels would fall into the smallest size category. Thus, the thresholds are adjusted

[11] For Cascade Mask R-CNN and Mask R-CNN with a ResNet-101 backbone the number of iterations per epoch are tripled. Thus, all epoch related numbers are divided by three.

Size class	Default configuration		Adjusted configuration	
	Size thresholds (px)	Trees within class	Size thresholds (px)	Trees within class
Small	$0 \leq x < 32^2$	94.2 %	$0 \leq x < 12^2$	53.5 %
Medium	$32^2 \leq x < 96^2$	5.7 %	$12^2 \leq x < 24^2$	33.9 %
Large	$96^2 \leq x < 10,000^2$	0 %	$24^2 \leq x < 10,000^2$	12.6 %

Table 4 By changing the pixel (px) size thresholds, the training set labels are spread more evenly among the size categories.

as shown in Table 4, allowing a meaningful evaluation for small, medium and large objects. All the aforementioned hyperparameter settings are valid for Mask R-CNN and Cascade Mask R-CNN.

Ablation Experiments. The initial model is trained following the preprocessing steps and hyperparameter settings, described above. In order to find a good base model, in a first iteration several experiments with deviating configurations are conducted. Besides ResNet-50, a ResNet-101 backbone is used to investigate whether the model's results can benefit from a deeper model. Several data augmentations are applied for the training pipeline. These configuration changes are listed in Table 5 and are hereafter referenced by their change key. If one configuration change has a positive effect on the segmentation AP, it is kept for any further experiments. Eventually, also a Cascade Mask R-CNN model in its base configuration with three cascade stages and using the adjustments of the best-performing Mask R-CNN model is trained. Then to address specific issues of the first iteration models, additional experiments are run based on the evaluation results and visual inspections in inference mode. In the end, the model with the best performance is evaluated on the test set.

Change Key	Configuration Changes
[1]	Custom Normalization: Application of training set RGB means and standard deviations.
[2]	Custom Normalization: Application of training set RGB means.
[3]	Resizing: Resizing scale 1000 x 1000 is added during training and solely used during inference.
[4]	Resizing: Resizing scale 1333 x 1333 is added during training and solely used during inference.
[5]	Random flip: Addition of vertical and diagonal flip to the training pipeline. Probability for each direction is 0.25.
[6]	Random flip: No flipping of the input image.
[7]	Photometric distortions: Brightness, contrast, saturation, hue and color scheme are set randomly in train pipeline.

Table 5 Apart from the experiments with different backbones and Cascade Mask R-CNN, these configuration changes are tested.

5 Results and Discussion

This chapter puts its focus on the evaluation phase of the CRISP-DM framework. Covering the first of the two iterations, the base model including the ablation experiments described in Chapter 4 are discussed with respect to their AP scores, while also taking a look at the training time. Validation set prediction results from inference mode are presented to highlight issues of the models and countermeasures are proposed. In the second iteration, these are then evaluated in the same procedure.

5.1 Base Model Results

Table 6 shows the evaluation results of the different models on the validation set. Besides the overall mask AP, detailed AP values for different IoU thresholds (AP_{50} and AP_{75}) and for objects of different sizes (AP_S, AP_M and AP_L) are presented. For each configuration, the overall bounding box AP, the number of epochs to reach the best performing model and the respective training time are indicated as well.

The base model with a ResNet-50-FPN backbone and no changes of the default configuration produces the best overall mask AP and also shares the top spot in bounding box AP and AP_S. A deeper ResNet-101-FPN backbone does not lead to better segmentation results, although needing slightly less training time. However, it should be noted that training time can vary, as the IoU calculation and the evaluation are performed on CPU. A custom normalization does not bring a noticeable advantage. While a normalization based on the training set RGB channel means and standard deviations ([1]) significantly worsens the results, a normalization only with the RGB means ([2]) slightly improves AP_{50}, AP_{75}, AP_M and AP_L values but due to poorer AP_S values, it, too does not bring an overall advantage. Different resizing scales ([3], [4]) do not indicate an overall improvement, although performing marginally better in terms of AP_{50} and AP_{75}. Randomly flipping the image horizontally, vertically or diagonally during training ([5]) instead of only in horizontal direction proves just as little beneficial as omitting the random flip altogether ([6]). Applying photometric distortions ([7]) also worsens the segmentation performance. Across all configurations, a very low AP_S score can be found as well as a substantial gap between AP_{50} and AP_{75} values that are much higher than those on COCO in Table 3. This hints at two difficulties which are the identification of small objects and a precise segmentation of the object mask. As none of the additional augmentations and hyperparameter adaptations indicate a better performance, the ResNet-50-FPN model in its default configuration is taken as a blueprint for the Cascade Mask R-CNN model. With the same hyperparameter settings, its results lack behind those

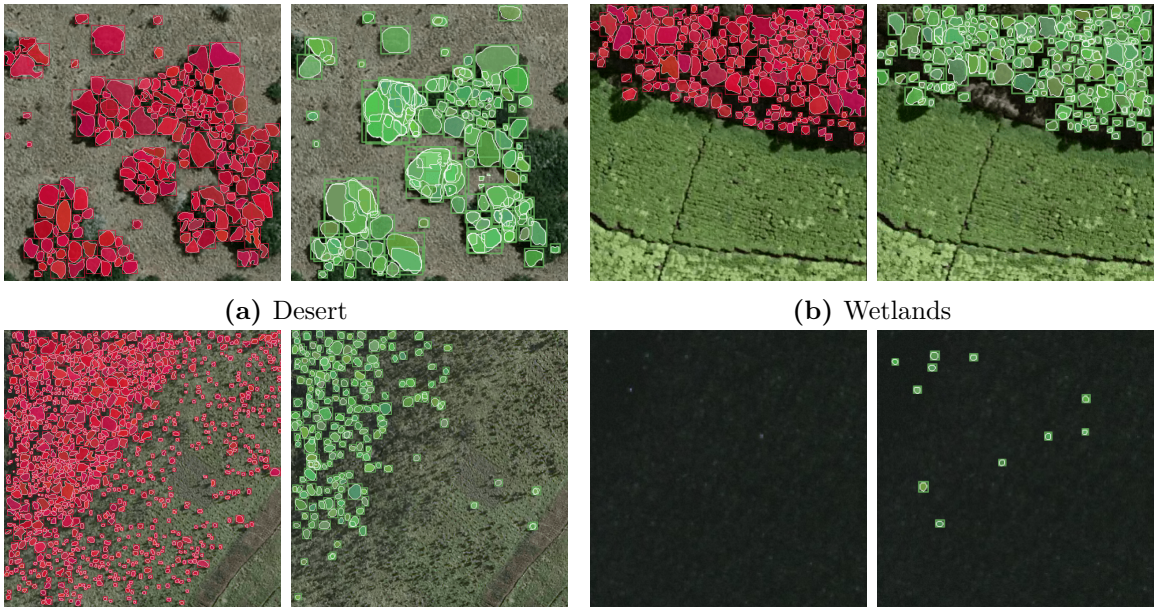
Backbone & Configuration Changes	Trained Epochs	Training time	BBox	Segmentation					
			AP	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask R-CNN									
R-50 & -	15	5402 s	**14.6**	**14.2**	37.0	7.5	**6.8**	19.8	33.5
R-101 & -	5	**5083** s	14.5	14.1	37.0	7.4	6.6	19.9	32.8
R-50 & [1]	33	12214 s	12.7	12.4	33.8	5.8	5.8	17.9	28.9
R-50 & [2]	18	7495 s	**14.6**	14.1	37.1	**7.7**	6.3	**20.3**	**33.6**
R-50 & [3]	15	5337 s	**14.6**	14.1	37.2	**7.7**	6.7	20.0	32.9
R-50 & [4]	15	6733 s	**14.6**	14.1	37.2	**7.7**	**6.8**	19.7	32.7
R-50 & [5]	21	8934 s	13.6	13.1	34.5	7.0	5.9	18.6	31.5
R-50 & [6]	15	5655 s	14.5	13.9	**37.3**	7.5	6.6	19.7	33.2
R-50 & [7]	18	7108 s	13.6	13.4	35.2	7.0	5.8	19.6	32.9
Cascade Mask R-CNN									
R-50 & -	7	11491 s	13.8	13.4	34.9	7.3	5.7	19.6	32.9
R-101 & -	6	10640 s	13.4	12.9	33.9	6.9	5.3	18.7	32.7

Table 6 The validation set evaluation results of the base models from the first iteration. As the number of iterations for Mask R-CNN with a ResNet-101-FPN backbone and all Cascade Mask R-CNN networks is three times as high as for the Mask R-CNN R-50 models, the former were evaluated after every epoch.

of the Mask R-CNN models in every respect while needing a longer training time. Again, the ResNet-101-FPN version has no beneficial effect.

In addition to the evaluation metrics, the model performance is visually assessed by comparing the inference mode predictions to the ground truth images. To identify patterns in the detection issues, for every terrain multiple image patch predictions are inspected. Figure 12 shows a sample of the inference results. These all stem from the Mask R-CNN ResNet-50-FPN base model.

Without addressing any specific sample image, it can be seen that the network principally works in its basic function, that is to differentiate trees from other structures. Providing a clear contrast, trees located on sandy areas as in Figure 12a are successfully detected. Within the same sample, though, it can also be seen that the delineation of different instances is not always accurate and sometimes, e.g., at the right tail of the bottom left cluster of trees, multiple trees are predicted to be just one. Contrasting this, at the right border where the tree crowns are very dense and hard to delineate, a bunch of trees are missed completely. Figure 12b shows that the model effectively separates between wetlands and tree crowns and across other similar image patches containing wetlands and grasses, the model produces only few false positives. However, as can be seen by the gap at the bottom center of the tree line, some false negatives are generated. Even more false negatives are visible in Figure



(a) Desert **(b)** Wetlands

(c) Forest **(d)** Water

Figure 12 A comparison of the ground truth annotations (red) and the model predictions (green) show where the model already produces decent results and which sceneries lead to poor segmentations.

12c that shows a forest region with many, very small trees. While the bigger trees in the top left are still recognized, albeit also streaked with a number of false negatives, almost all of the small trees towards the bottom right are missed. This phenomenon also occurs in inference results with comparably small tree sizes and is evidence of the significantly lower AP_S values in contrast to the other size classes. Figure 12d on the other hand shows an issue of the model with one particular image that contains dark water. In this patch and all other associated image patches varying but substantial amounts of false positives are generated. As other images of water do not trigger these detections, it could be due to the low brightness and because of the training set that does not contain images showing water of similar darkness. Interestingly, the addition of photometric distortion ([7]) leads to even more false positives here.

Second Iteration Adjustments. The main issue found in the models of the first iteration is a poor segmentation performance for small objects (AP_S), presenting itself in a number of false negatives as in Figure 12c. To mitigate this effect a number of adjustments are proposed. First, as elaborated in Chapter 3.2, anchors are generated in different sizes and aspect ratios and smaller anchors should be able to capture small objects. In MMDetection the anchor size is determined by two parameters, which are the anchor scales (default = 8) and the strides of the feature maps from the FPN that are used for anchor generation (default = [4, 8, 16, 32, 64]). The actual anchor scale is then the product of these parameters. Adding a scale value of 4 allows additional, halved anchor scales. Moreover, when the image patches are made smaller, due to the

Change Key	Configuration Changes
$[a]$	Smaller anchors scales: Additional RPN anchor scales, reduced by a factor of two.
$[b]$	Smaller image patches: Reduction of the train and validation image patch scale by a factor of two.
$[c]$	Oversampling: Addition of larger image patches to the training set.
$[d]$	More maximum detections: Increase of the maximum amount of RoIs and detections during training and testing by a factor of two.
$[e]$	Additional dark training image: A duplicated and darkened training image showing water is added to the training set.

Table 7 To counter the apparent problems of the first iteration models, these adjustments are experimented with in the second iteration.

resizing in the data pipeline, the objects become larger. Thus, another experiment is to reduce the image patch minimum width and height to 150 pixels. Furthermore, Kisantal et al. [Kis+19] propose to oversample small objects and copy and paste small objects into other parts of an image. These measures were proposed for better detection of small objects on COCO. They are based on the arguments that small objects are less frequent than medium and large objects and since small objects cover a smaller area and thus occur only in a limited context, Mask R-CNN has difficulty in generalizing from the training examples. Although the first argument does not apply to this dataset, these measures are tested nonetheless. As a custom transformation implementing the copying and pasting of small objects and their integration into other parts of an image would extend the scope of this thesis, the oversampling effect is mimicked by generating image patches with a minimum scale of 600 pixels, thereby creating the opposite effect as described for the small image patches. At a scale of 600 pixels, the objects are smaller compared to the image patches with a minimum scale of 300 pixels. These larger image patches are added to the normal training set to oversample small objects during training. Lastly, the hyperparameters presented in Chapter 4.2 affecting the maximum number of detections are further increased to inspect whether these were a limiting factor so far. To make the network more resistant to images showing dark water such as in Figure 12d, a very large training image showing water is reduced in its brightness and added to the training set a second time. An overview of these adjustments is given in Table 7.

5.2 Adjusted Model Results

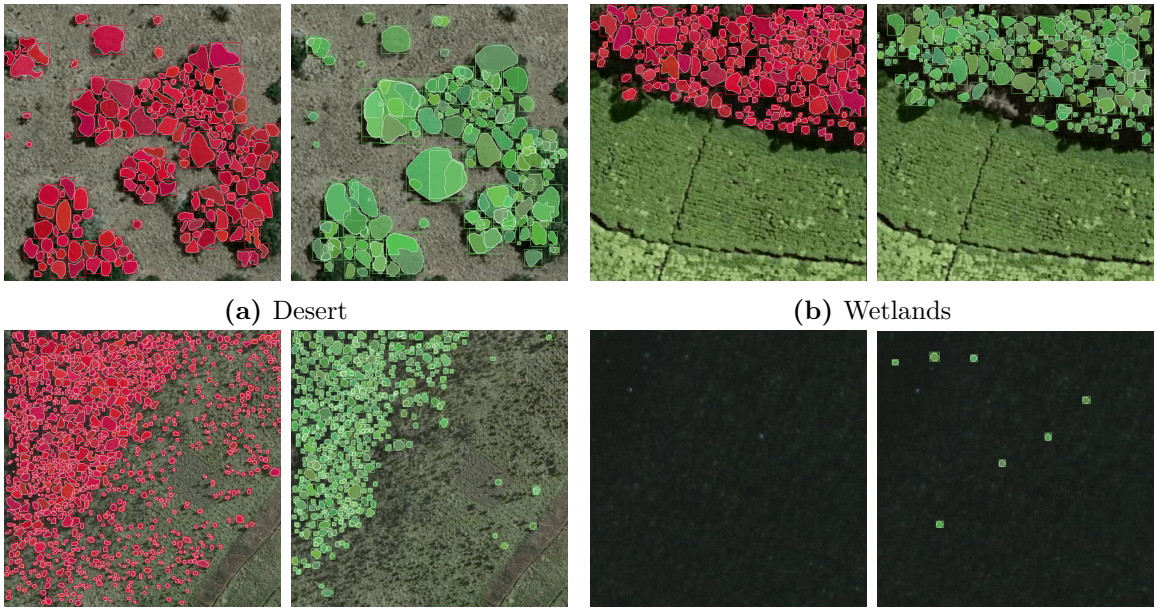
Again, the evaluation metrics but also the model results in inference mode are inspected. The former are condensed in Table 8. Allowing smaller anchor scales ($[a]$), albeit prolonging the training time and leading to a lower AP_L value, positively af-

Backbone & Configuration Changes	Trained Epochs	Training time	BBox	Segmentation					
			AP	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask R-CNN									
R-50 & -	15	**5402** s	14.6	14.2	37.0	7.5	6.8	19.8	33.5
R-50 & [a]	15	7043 s	15.3	14.8	39.4	7.7	7.5	20.3	33.0
R-50 & [a],[b]	15	12970 s	13.9	13.1	35.5	6.5	6.6	19.1	31.2
R-50 & [a],[c]	24	14415 s	15.6	15.0	40.0	7.8	7.8	20.9	32.5
R-50 & [a],[c],[d]	24	16571 s	**15.9**	**15.1**	**41.0**	7.6	**8.1**	**21.2**	32.4
R-50 & [a],[c],[d],[e]	21	14311 s	13.6	13.1	34.5	7.0	5.9	18.6	31.5
Cascade Mask R-CNN									
R-50 & -	7	11491 s	13.8	13.4	34.9	7.3	5.7	19.6	32.9
R-50 & [a],[c],[d]	5	16119 s	15.1	14.9	38.6	**8.4**	7.0	20.9	**35.4**

Table 8 The validation set evaluation results of the configurations from the second iteration. For comparison, the Mask R-CNN and Cascade Mask R-CNN models in their default configuration are displayed once again. Configuration changes with an overall positive effect are kept for further experiments.

fects the segmentation results, leading to a relative performance improvement of 10 % for small objects. The smaller image patches ([b]) lead to worse results while significantly prolonging the training time. Using smaller image patches, besides reducing the global context information, inevitably leads to an increased number of cases where trees are cut in half at the border of an image patch. These are thereby not labeled as an object which could potentially irritate the model. The oversampling ([c]) has a positive impact on the model's performance, although coming at the cost of a lower AP_L score. Again, the training time gets longer, which is in part due to the doubled number of training objects and IoU calculation running on CPU for image patches with a large number of objects. Further increasing the maximum number of detections ([d]) also marginally improves the network and in combination with [a] and [c], this shows the best overall performance with a 19.1 % improvement in AP_S. The addition of a dark water training image ([e]), although reducing the number of false positives in Figure 12d, is not beneficial. This might be due to an increased number of false negatives in visually similar terrains, e.g., dark forests. Cascade Mask R-CNN also benefits from the adjustments, almost closing the gap to Mask R-CNN. Its AP_L and AP_{75} surpass those of Mask R-CNN, which makes sense since the cascading structure promises higher quality detections and segmentations [CV19, p. 1495].

By again taking a look at some sample ground truth as well as predicted image patches depicted in Figure 13, the differences between the base model and the adjusted version become visible. In Figure 13a, the missed trees at the border are now detected. The delineation of individual trees is still subpar, evidenced by many cases



(a) Desert **(b)** Wetlands

(c) Forest **(d)** Water

Figure 13 Using the best performing model with adjustments [a], [c], [d] and comparing it to the ground truth, shows how some issues are partly alleviated while others remain.

where the predicted mask does not align with the ground truth, thereby reflecting the low AP_{75} score. In Figure 13c, a lot more trees are detected than with the base version. However, the smallest trees in the bottom right are still not recognized. Similarly, although Figure 13d shows less false positives, these still occur on dark water. Thus, although the adjustments improved the model, the problems of detecting small objects, performing precise segmentations and preventing false positives remain.

Test Set Evaluation. The model with the best validation set performance (Mask R-CNN R-50 & [a], [c], [d]) performs substantially worse on the test set than on the validation set, as shown in Table 9. As the mask AP is worse than those of all models on the validation set, this does not stem from the model being fine-tuned on the validation set. When taking a look at the inference results, it can be noticed that three rather large images which in total make up 25 % of the image patches are partly responsible for the worse performance. Two of them generate a large number of false positives and one a large number of false negatives. If these images are removed, the performance on the reduced test set is similar to that on the validation set.

Test Set	BBox	Segmentation					
	AP	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask R-CNN: R-50 & [a],[c],[d]							
Full Test Set	12.1	11.8	31.7	5.8	4.9	16.1	28.0
Reduced Test Set	15.1	15.0	38.9	8.1	6.0	17.6	29.4

Table 9 Test set evaluation results.

6 Conclusion

In this thesis, a particular case of application for instance segmentation networks was investigated. Based on a dataset of Rwandan satellite images and following the CRISP-DM methodology, an overview of the problem as well as the data was established. Then, different instance segmentation models were presented, compared and eventually Mask R-CNN and Cascade Mask R-CNN were chosen for implementation. Using the open source library MMDetection, these were applied to the mentioned dataset and evaluated with respect to their mask average precision and training time.

It could be shown that the annotated satellite images can be converted in a concise pipeline into a format that allows the application of a range of instance segmentation networks from MMDetection's model library. Mask R-CNN and Cascade Mask R-CNN were both successfully configured, trained and tested. In its default configuration Mask R-CNN exceeds Cascade Mask R-CNN in terms of performance and training time. The application of different data transformations such as adjusted resizing scales, a custom normalization or the random setting of image properties like brightness and contrast could not improve the model's overall segmentation results. Similarly, a deeper backbone network did not lead to benefits in the model's performance.

Albeit far away from AP results achieved on datasets like COCO, the model could successfully segment the individual trees. On large objects, which are still small by COCO size standards, a decent AP was recorded. Small objects in turn posed a problem for the network. By adjusting the anchor size and oversampling small objects during training, it was possible to achieve a relative improvement of 19.1 % in the model's AP_S on the validation set. Furthermore, precise segmentations were made out as an issue, evidenced by much lower AP_{75} values in comparison with the AP_{50} scores. Here, Cascade Mask R-CNN, although still showing a lower overall mask AP score than Mask R-CNN, reached higher quality segmentations. Lastly, the network is prone to producing false positives in sceneries similar to those featuring trees, e.g., dark water. Changing image parameters such as its brightness or contrast could not mitigate this and although deliberately adding training images of dark water seemed to alleviate the problem, this produced an overall negative effect. While these difficulties were already apparent during the evaluation on the validation set, they were especially noticeable on the test set where three images featuring those problems significantly impaired the evaluation metrics.

The limitations of this thesis encompass the generalization of the results, the exploration as well as selection of alternative models and the preprocessing and augmentation of the data. As became visible in the diverging results between validation and test set, it is uncertain how the model performs on additional satellite data. Since the dataset contained only 112 images, it is questionable whether this is enough to capture the diversity of Rwanda's landscape, let alone that of other countries. Additionally, since validation and test set were made up of only 22 images, the small sample size gives each image, especially the larger ones a great impact on the evaluation, further raising the question of generalization. Apart from that, the amount and the rate at which new instance segmentation networks are developed leaves a great number of alternative models uncovered. But also within the bounds of Mask R-CNN and Cascade Mask R-CNN only a limited amount of adjustments were exploited. Many hyperparameters, e.g., the number and IoU thresholds of the cascade stages were not altered so far and other architectural choices such as the used backbone offer further options. On the data side, next to a number of MMDetection's inbuilt augmentations (all of them providing their own hyperparameters to twist), a range of custom augmentations that potentially enhance the model were not applied. For example, Kisantal et al. [Kis+19] proposed copying and pasting small objects into other parts of the image, thus setting them into different contexts.

Consequently, further research should be conducted, focusing on the segmentation of small, overlapping objects in remote sensing data. This could be done by investigating models that show a particularly good AP_S on COCO, e.g., QueryInst [Fan+21] or by exploring augmentations designed to enhance the detection of small objects. Linked to this field of research is that of generating high quality segmentations. Intuitively, it should be harder to precisely segment small objects, for a small difference of say one pixel between the ground truth and predicted mask affects the IoU much stronger than the same offset does for larger objects. Thus, investigating AP scores for different IoU thresholds for each size class separately could show whether the large gap observed between AP_{50} and AP_{75} values is limited to small tree canopies only. Beyond that, it might be feasible to train an object detection and a semantic segmentation framework separately, the former to count trees and the latter to estimate the biomass. However this depends on the exact organization goals, which should be thoroughly discussed for a deployment in practice.

Bibliography

[Abd17] Waleed Abdulla. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. 2017. URL: https://github.com/matterport/Mask_RCNN (visited on 07/05/2022).

[Alb+18] Saad Albawi et al. "Social Touch Gesture Recognition Using Convolutional Neural Network". In: *Computational intelligence and neuroscience* (2018). DOI: 10.1155/2018/6973103.

[Bol+19] Daniel Bolya et al. "YOLACT: Real-Time Instance Segmentation". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9156–9165. DOI: 10.1109/ICCV.2019.00925.

[Bra+20] Martin Brandt et al. "An unexpectedly large count of trees in the West African Sahara and Sahel". In: *Nature* 587 (2020), pp. 78–82. ISSN: 1476-4687. DOI: 10.1038/s41586-020-2824-5.

[Che+19] Kai Chen et al. *MMDetection: Open MMLab Detection Toolbox and Benchmark*. 2019. URL: https://github.com/open-mmlab/mmdetection (visited on 07/05/2022).

[Che+20] Hao Chen et al. "BlendMask: Top-Down Meets Bottom-Up for Instance Segmentation". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 8570–8578. DOI: 10.1109/CVPR42600.2020.00860.

[COC22] COCO Consortium. *Detection Evaluation*. 2022. URL: https://cocodataset.org/#detection-eval (visited on 06/21/2022).

[CV19] Zhaowei Cai and Nuno Vasconcelos. "Cascade R-CNN: High Quality Object Detection and Instance Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.5 (2019), pp. 1483–1498. DOI: 10.1109/TPAMI.2019.2956516.

[CW11] J.B. Campbell and R.H. Wynne. *Introduction to Remote Sensing, Fifth Edition*. 2nd ed. Guilford Publications, 2011. ISBN: 9781609181772.

[DHS16] Jifeng Dai, Kaiming He, and Jian Sun. "Instance-Aware Semantic Segmentation via Multi-task Network Cascades". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 3150–3158. DOI: 10.1109/CVPR.2016.343.

[Elg20] Mohamed Elgendy. *Deep Learning for Vision Systems*. Manning Publications Co., 2020.

[Fan+21] Yuxin Fang et al. “Instances as Queries”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 6890–6899. DOI: 10.1109/ICCV48922.2021.00683.

[Fuk80] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological Cybernetics* 36 (1980), pp. 193–202. DOI: 10.1007/BF00344251.

[Gar+17] Alberto Garcia-Garcia et al. *A Review on Deep Learning Techniques Applied to Semantic Segmentation*. 2017. DOI: 10.48550/ARXIV.1704.06857.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pp. 315–323.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN: 0262035618.

[GBK22] Wenchao Gu, Shuang Bai, and Lingxing Kong. “A review on 2D instance segmentation based on deep neural networks”. In: *Image and Vision Computing* 120 (2022). ISSN: 0262-8856. DOI: 10.1016/j.imavis.2022.104401.

[Gér19] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly Media, Incorporated, 2019.

[Gha+19] Pedram Ghamisi et al. “Multisource and Multitemporal Data Fusion in Remote Sensing: A Comprehensive Review of the State of the Art”. In: *IEEE Geoscience and Remote Sensing Magazine* 7.1 (2019), pp. 6–39. DOI: 10.1109/MGRS.2018.2890023.

[Gil+11] Sean Gillies et al. *Fiona is OGR’s neat, nimble, no-nonsense API*. Toblerity, 2011. URL: https://github.com/Toblerity/Fiona (visited on 07/08/2022).

[Gil+13] Sean Gillies et al. *Rasterio: geospatial raster I/O for Python programmers*. Mapbox, 2013. URL: https://github.com/rasterio/rasterio (visited on 07/08/2022).

[Gir+14] Ross Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[Gir15] Ross Girshick. “Fast R-CNN”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[Gu+18] Jiuxiang Gu et al. “Recent advances in convolutional neural networks”. In: *Pattern Recognition* 77 (2018), pp. 354–377. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2017.10.013.

[HA20] Niall P Hanan and Julius Y Anchang. “Satellites could soon map every tree on Earth”. In: *Nature* 587 (2020), pp. 42–43. ISSN: 1476-4687. DOI: 10.1038/d41586-020-02830-3.

[Har+14] Bharath Hariharan et al. “Simultaneous Detection and Segmentation”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 297–312. ISBN: 978-3-319-10584-0. DOI: 10.1007/978-3-319-10584-0_20.

[HB20] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. “A survey on instance segmentation: state of the art”. In: *International Journal of Multimedia Information Retrieval* 9 (2020), pp. 171–189. DOI: 10.1007/s13735-020-00195-x.

[He+15] Kaiming He et al. “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9 (2015), pp. 1904–1916. DOI: 10.1109/TPAMI.2015.2389824.

[He+16] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[He+17] Kaiming He et al. “Mask R-CNN”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[Hir+17] Jennifer N. Hird et al. “Google Earth Engine, Open-Access Satellite Data, and Machine Learning in Support of Large-Area Probabilistic Wetland Mapping”. In: *Remote Sensing* 9.12 (2017). ISSN: 2072-4292. DOI: 10.3390/rs9121315.

[HW68] D. H. Hubel and T. N. Wiesel. “Receptive fields and functional architecture of monkey striate cortex”. In: *The Journal of Physiology* 195.1 (1968), pp. 215–243. DOI: 10.1113/jphysiol.1968.sp008455.

[HZG20] Shijie Hao, Yuan Zhou, and Yanrong Guo. “A Brief Survey on Semantic Segmentation with Deep Learning”. In: *Neurocomputing* 406 (2020), pp. 302–321. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2019.11.118.

[Jad+15] Max Jaderberg et al. “Spatial Transformer Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015.

[Jea+16] Neal Jean et al. “Combining satellite imagery and machine learning to predict poverty”. In: *Science* 353.6301 (2016), pp. 790–794. DOI: 10.1126/science.aaf7894.

[Kis+19] Mate Kisantal et al. “Augmentation for small object detection”. In: *CoRR* (2019). DOI: 10.48550/ARXIV.1902.07296.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012.

[LBG15] Yann Lecun, Yoshua Bengio, and Hinton Geoffrey. “Deep Learning”. In: *Nature* 521 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[LeC+89] Yann LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[Lec+98] Yann Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[Lin+17] Tsung-Yi Lin et al. “Feature Pyramid Networks for Object Detection”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106.

[Lin+20] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.2 (2020), pp. 318–327. DOI: 10.1109/TPAMI.2018.2858826.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965.

[Ma+19] Lei Ma et al. “Deep learning in remote sensing applications: A meta-analysis and review”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 152 (2019), pp. 166–177. ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2019.04.015.

[McD20] Jonathan C. McDowell. “The Low Earth Orbit Satellite Population and Impacts of the SpaceX Starlink Constellation”. In: *The Astrophysical Journal* 892.2 (Apr. 2020), p. L36. DOI: 10.3847/2041-8213/ab8016.

[MP43] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5 (4 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259.

[NH10] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *ICML*. 2010, pp. 807–814.

[Oce+20] Nuri Erkin Ocer et al. “Tree extraction from multi-scale UAV images using Mask R-CNN with FPN”. In: *Remote Sensing Letters* 11.9 (2020), pp. 847–856. DOI: 10.1080/2150704X.2020.1784491.

[ON15] Keiron O’Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. DOI: 10.48550/ARXIV.1511.08458.

[Red+16] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[Ren+15] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015.

[Rez+19] Hamid Rezatofighi et al. “Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 658–666. DOI: 10.1109/CVPR.2019.00075.

[RF18] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. Tech. rep. 2018. DOI: 10.48550/ARXIV.1804.02767.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (6088 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

[Ros58] Frank Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65 (6 1958), pp. 386–408. DOI: 10.1037/h0042519.

[San+11] Koen E. A. van de Sande et al. "Segmentation as selective search for object recognition". In: *2011 International Conference on Computer Vision*. 2011, pp. 1879–1886. DOI: 10.1109/ICCV.2011.6126456.

[SSD20] Farhana Sultana, Abu Sufian, and Paramartha Dutta. "Evolution of Image Segmentation using Deep Convolutional Neural Network: A Survey". In: *Knowledge-Based Systems* 201-202 (2020). ISSN: 0950-7051. DOI: 10.1016/j.knosys.2020.106062.

[Sun+22] Ying Sun et al. "Counting trees in a subtropical mega city using the instance segmentation method". In: *International Journal of Applied Earth Observation and Geoinformation* 106 (2022), p. 102662. ISSN: 0303-2434. DOI: 10.1016/j.jag.2021.102662.

[SYZ13] Wanhua Su, Yan Yuan, and Mu Zhu. *Threshold-free Evaluation of Medical Tests for Classification and Prediction: Average Precision versus Area Under the ROC Curve*. 2013. DOI: 10.48550/ARXIV.1310.5103.

[Tia+19] Zhi Tian et al. "FCOS: Fully Convolutional One-Stage Object Detection". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9626–9635. DOI: 10.1109/ICCV.2019.00972.

[Wag+20] Fabien H. Wagner et al. "U-Net-Id, an Instance Segmentation Model for Building Extraction from Satellite Images—Case Study in the Joanópolis City, Brazil". In: *Remote Sensing* 12.10 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12101544.

[Wei+20] Shunjun Wei et al. "HRSID: A High-Resolution SAR Images Dataset for Ship Detection and Instance Segmentation". In: *IEEE Access* 8 (2020), pp. 120234–120254. DOI: 10.1109/ACCESS.2020.3005861.

[WH00] Rüdiger Wirth and Jochen Hipp. "CRISP-DM: Towards a standard process model for data mining". In: *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*. 2000, pp. 29–39.

[WS19] Shuai Wang and Zhendong Su. *Metamorphic Testing for Object Detection Systems*. 2019. DOI: 10.48550/ARXIV.1912.12162.

[Wu+19] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019. (Visited on 07/05/2022).

[Xi+21] Xiangshu Xi et al. "Evaluation of dimensionality reduction methods for individual tree crown delineation using instance segmentation network and UAV multispectral imagery in urban forest". In: *Computers and Electronics in Agriculture* 191 (2021). ISSN: 0168-1699. DOI: 10.1016/j.compag.2021.106506.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. DOI: 10.48550/ARXIV.1708.07747.

[Yu+16] Jiahui Yu et al. "UnitBox: An Advanced Object Detection Network". In: *Proceedings of the 24th ACM International Conference on Multimedia*. MM '16. Amsterdam, The Netherlands: Association for Computing Machinery, 2016, pp. 516–520. ISBN: 9781450336031. DOI: 10.1145/2964284.2967274.

[Zou+19] Zhengxia Zou et al. *Object Detection in 20 Years: A Survey*. 2019. DOI: 10.48550/ARXIV.1905.05055.

Declaration of Authorship

I hereby declare that, to the best of my knowledge and belief, this thesis titled *Detection and Segmentation of Tree Instances on a Rwandan Satellite Dataset* is my own, independent work. I confirm that each significant contribution to and quotation in this thesis that originates from the work or works of others is indicated by proper use of citation and references; this also holds for tables and graphical works.

Münster, 29.07.2022

Steffen Backmann

Consent Form

Name: Steffen Backmann
Title of Thesis: Detection and Segmentation of Tree Instances on a Rwandan Satellite Dataset

What is plagiarism? Plagiarism is defined as submitting someone else's work or ideas as your own without a complete indication of the source. It is hereby irrelevant whether the work of others is copied word by word without acknowledgment of the source, text structures (e.g. line of argumentation or outline) are borrowed or texts are translated from a foreign language.

Use of plagiarism detection software. The examination office uses plagiarism software to check each submitted bachelor and master thesis for plagiarism. For that purpose the thesis is electronically forwarded to a software service provider where the software checks for potential matches between the submitted work and work from other sources. For future comparisons with other theses, your thesis will be permanently stored in a database. Only the School of Business and Economics of the University of Münster is allowed to access your stored thesis. The student agrees that his or her thesis may be stored and reproduced only for the purpose of plagiarism assessment. The first examiner of the thesis will be advised on the outcome of the plagiarism assessment.

Sanctions Each case of plagiarism constitutes an attempt to deceive in terms of the examination regulations and will lead to the thesis being graded as "failed". This will be communicated to the examination office where your case will be documented. In the event of a serious case of deception the examinee can be generally excluded from any further examination. This can lead to the exmatriculation of the student. Even after completion of the examination procedure and graduation from university, plagiarism can result in a withdrawal of the awarded academic degree.

I confirm that I have read and understood the information in this document. I agree to the outlined procedure for plagiarism assessment and potential sanctioning.

Münster, 29.07.2022

Steffen Backmann