Evaluation of Parking Space Detection from Aerial Imagery Using Convolutional Neural Networks

Max Lindblom, Carl Nilsson

Master's thesis
2019:E6

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Evaluation of Parking Space Detection From Aerial Imagery Using Convolutional Neural Networks

Carl Nilsson & Max Lindblom

Department of Mathematics
Lund University

Supervisor: Niels Christian Overgaard & Erik Bylow

Examiner: Kalle Åström

February 13, 2019

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

In this thesis, the viability of using Convolutional Neural Networks to detect parking spaces using aerial imagery has been evaluated. Three state of the art networks have been tested - YOLOv3, RetinaNet, and Mask R-CNN. A dataset of urban parking lots and corresponding annotations was generated from scratch using a custom built GUI to annotate automatically generated images of parking lots from Open Street Map, from varying aerial imagery providers. This dataset was used to test and evaluate the different networks, and Mask R-CNN was used for a lengthy parameter tuning process, as it seemed to perform optimally of the three networks. The resulting model did not perform the best, believed to be because of the low amount of features represented in parking spaces. While results indicated that a somewhat complete solution is possible, it might not be feasible using a pure single CNN approach.

An example of the pipeline with its final results can be seen in Figures 0.1– 0.3.

Figure 0.1: Input image from validation dataset.

Figure 0.2: Manually annotated ground truth labels.

Figure 0.3: Detection results from Mask R-CNN.

Acknowledgements

This report was written in its entirety by us, Carl Nilsson and Max Lindblom, who also performed the process described in it. The workload between us has been equal, and we have both contributed to every part of the process. We would like to thank Jayway for hosting us during work on this master thesis and for all the generous help and resources provided by them. We would also like to thank Zenuity for the problem proposal, as well as for access to their GPU server farm, which helped speed up network training time extraordinarily. Furthermore we would like to thank Lantmäteriet, through which we gained access to necessary data thanks to their cooperation with Lund University.

Finally we would like to thank our mentors for guidance and help during the process - Niels C. Overgaard and Erik Bylow at the Department of Mathematics at Lund Faculty of Engineering (LTH), Martin Gunnarsson and Hans Löfgren at Jayway, and our contacts at Zenuity, Edgar Nunez and Vilhelm Frändberg.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Problem Background and Relevance

Autonomous cars is one of the main points of interest in machine learning. Within this area however, several problems are being explored in parallel. One field which could be a stepping stone for fully autonomous cars, is automated parking. Starting with experiments in a controlled and limited environment, with reduced risk of accidents and damages, could pave way for future development in the area. Thus, in the problem space of automated parking, a crucial first step is to generate a model of the layout for the parking lot or parking area. The naive approach would be for a car to simply drive around, using sensors to scan the surrounding area, in order to deduce what constitutes a parking space, and what does not. Another solution would be to acquire an overhead view of the parking lot, and segment the area into possible individual parking spaces. This would allow the vehicle to make informed decisions about where to look for open spots. This is the problem space that this study will occupy. Autonomous cars, as with all of Artificial Intelligence, is not realized in one step, but rather several. For Artificial Intelligence to succeed in self driving cars, the problem ought to be split up into sub-problems, where each sub-problem requires a solution producing a specific output. With this in mind, this study could improve the development of car autonomy as a whole.

1.2 Problem Description

The problem description for this thesis is as follows. Is it possible to use satellite or aerial imagery in order to train a machine learning algorithm to reliably detect parking spaces. Our goal is to use a variety of current state of the art Convolutional Neural Networks to solve this problem, and compare them to each other.

Class identification in images is a rather well studied field in machine learning. Since the advent of Artificial Neural Networks in the 1940 – 1950s, see for example the perceptron, first proposed by F. Rosenblatt [8], it has been one of the signature tasks that machine learning is prophesized to solve. Nonetheless, neural networks did not come into its own at that time, but has made a resurgence as of late. This is mostly due to advancement in computing power, enabling one to train more and more complex networks in shorter time. Currently, machine learning, and neural networks in particular, is a very active research field.

1.3 Glossary

For the sake of clarity and simplicity, a short list of words and specific terms will be defined below.

- **Machine Learning**. Machine Learning is the concept of a system or an algorithm, or a collection of them, that intuitively and systematically learn specified concepts or tasks. A more extensive definition and explanation can be found in Section 2.1.
- **Model**. In machine learning, the model is the term for the trained system, or more specifically the weights associated with it. Thus, a model is a supposedly complete version of a particular algorithm, ready to be used for the task it was designed for.
- **Autonomous Car**. An autonomous car is a car, or in some cases the software controlling the car, that makes decisions and performs actions without any human interference. Sometimes autonomous cars will be described in a specific context, in which case autonomy will be assumed only for the given context.
- **ANN - Artificial Neural Network**. A digital network made of nodes and connections, intended to simulate a biological neural network. Similar to that of biological brains. See Section 2.2.
- **CNN - Convolutional Neural Network**. An Artificial Neural Network modeled after the visual cortex of animals, named for its inclusion of convolutional layers, which perform convolutional operations on its input. See Section 2.2.1.
- **Bounding Box**. A bounding box is defined as a 2D rectangular area which completely encapsulates an area of interest. The lines of the bounding box are parallel to the x- and y-axis, which means the orientation of the bounding box is not necessarily optimal, but it does encapsulate the object completely.
- **ROI - Region Of Interest**. A Region Of Interest is simply an area of data that has been identified to explore some kind of relevance. In the case of this report, ROIs are 2D and will usually be evaluated to determine if they contain an instance of a class, in this case a parking space.
- **Parking space**. A boxed in area meant to hold one car, represented by a marking on the ground.
- **Parking lot**. An area specifically intended for parking cars. A parking lot is made up of 1 or more parking spaces.
- **Spatial Resolution**. The actual length in meters per pixel in an image.
- **mAP - mean Average Precision**. An evaluation measure using the average of the average precision, or accuracy, for classifications over all classes. While the problem described in this paper only uses one class, thus resulting in a regular average precision, the term is ubiquitous for Machine Learning, so it will be referred to as mAP regardless.

- **Dataset**. A collection of relevant data. The content of the dataset and the data type in question is different depending on content but the term will be used as a catch-all for a complete gathering of data of some sort.

1.4 Previous Works

To the best of our knowledge, no work has been done within this specific task. However, quite some previous relevant research has been done. Machine learning is a rather well established field of research, and image classification has had a resurgence in popularity due to the increase of computing power, as mentioned.

G. Amato et al. have done some work in the field of parking space occupancy detection [9] [10], in which they focus on real time determining if parking spaces are occupied or not. They use smart cameras, that is, cameras with some amount of computing power to extract features of the captured images, mounted on the side of buildings to monitor parking lots. This is certainly attractive to business owners who might wish to monitor the status of their parking areas, in order to increase revenue and energy efficiency, as S. Valipour et al. [11] brought up in their paper on the same subject. While their results are for the most case impressive, they do not occupy the same problem space as us. Namely, they focus on real-time monitoring from an angle, whereas our solution focuses top-down aerial imagery.

Something that all of these solutions have in common, however, is the use of a deep Convolutional Neural Network, henceforth referred to as CNN. A CNN is a specific type of Artificial Neural Network, a network of nodes constructed to emulate the neurons of the biological brain [12]. Convolutional Neural Networks, sometimes also called deep networks or deep Convolutional Networks due to the complex operations they perform, are specifically commonly used in image analysis, mainly because they require little to no pre-processing. See for example C. Dan et al. [13], where a CNN was used to classify handwritten digits in the MNIST dataset, very accurately. This is at the cost of more complex image processing in the net, which is feasible given the increased processing power of modern computers. The deep keyword is in reference to the multitude of hidden layers in the network, meant to simulate the receptive field of the visual cortex in mammal brains. The development of deep networks is essentially based on the article by D H Hubel & T N Wiesel [14] which showed that certain specific neurons respond differently to certain specific patterns and stimuli. According to this, neurons and layers in a CNN will perform in a similar way when presented with similar visual input. Image classification can thus be derived from this technology - by exposing the network to a large amount of images of the same class, it can recognize its common attributes and decide whether or not new images presented to it belong to said class or not. This is done implicitly by extracting or defining certain features belonging to a class, which all members of said class do have. For any new input, the network can then derive whether or not those features are present. This forms the basis of object detection, and CNNs are among state of the art in the field of end-to-end object detection, as they avoid the need for manual feature definitions. This also yields better results for most complex classes, as defining features beforehand quickly becomes too difficult, and is essentially only even feasible for

very simple classes, such as basic geometric shapes.

A similar area of research that has been done is solely vehicle detection from satellite imagery. One of which by H. Zheng [15] who made use of a morphological shared-weight neural network (MSNN) for classification. It combines the feature extraction capability of mathematical morphology with the function-mapping capability of neural networks in a single trainable architecture. The morphological operations performed by Zheng was top-hat and bottom-hat transformations, which segments out elements from the image, and serves as potential candidates for vehicles. To increase the segmentation, a thresholding algorithm, more specifically Otsu Thresholding, was applied. Afterwards, the segmented images were fed into a feed-forward neural network which, based on the extracted features, determined whether an element was a vehicle or not. This achieved relatively high accuracy.

Another approach by L. Abraham [16] also used, similarly to Zheng, top-hat and bottom-hat transformations together with Otsu Thresholding. The difference was in the choice of classifier where Abraham, instead of a neural network, used Connected Component Labeling of 4-connected neighborhoods, which is an algorithmic application of graph theory. This also achieved a high accuracy score, which in turn makes the problem of parking lots seem a whole lot more feasible.

1.5 Existing Networks

The design architecture of convolutional neural networks (CNNs) vary and can take many forms. Each have their own unique approach to solve a particular problem. One of the most famous of which started back in 2012 with **AlexNet** [17] as the the first "modern" deep CNN that outperformed all other competitors in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). In the following year 2013, yet another CNN called **ZFNet** [18] won the challenge. It had the same structure as AlexNet but with a tweaking on some of its hyper-parameters. Next up in 2014, **GoogLeNet** [19] (also called Inception V1 by Google) was declared the winner. It optimized the AlexNet architecture with its so called inception module. This module was based on much smaller convolutions which drastically reduced the number of trainable parameters from 60 million in AlexNet down to only 4 million. The runner up in 2014 is called **VGGNet** [20] and is also worth mentioning. Similarly, it is also using small convolutions, but with a usage of lots of filters as well. It has a very simple and uniform architecture, which makes it appealing. It is also widely used for its capability of extracting features from images very well. However, VGGNet has 138 million parameters, which does not make it as time efficient. In 2015, **ResNet** [21] (Residual Neural Network) took the stage and was the first neural network to actually beat human-level performance on the ImageNet dataset. It builds on the concept of pyramid cells found in the cerebral cortex of our brain, and so utilizes so called skip connections, residual connections, or short-cuts, to skip some layers. These skip connections work similarly to the gated recurrent units found in most RNNs (Recurrent Neural Networks). ResNet only features 1 layer skip at most, and likewise to VGGNet, ResNet also has this low complexity structure, which is a sought after trait. Many other well performing

networks uses ResNet as a backbone and adds extra functionality and techniques on top of it to make it unique. Usually it comes configured with pre-trained weights from the ImageNet or COCO dataset as a starting point. Such examples are **RetinaNet** [22] and **Mask R-CNN** [23].

For this report, we will use and compare three state-of-the-art region based object detection convolutional neural network architectures. You Only Look Once (YOLO), RetinaNet and Mask R-CNN. Read further about them in Section 2.6.

1.6 Images as Data

For most, if not all of image classification tasks in machine learning, images are used as the data for the problem. Convolutional Neural Networks are no exception, which will be the approach of this thesis. Images will be fed into the network, and will then be manipulated and evaluated in some way. This process is further described in Section 2.2.1. Nonetheless, the success of this thesis and really any image classification task will no doubt be based on the quality of the data, so generating a good dataset will be the first and perhaps most important step in the process. A detailed description of the data required and how it will be produced are described in Section 3.1, but for now let us be content in defining what data the problem requires. As specified in the problem description, satellite and/or aerial imagery is required. In particular, urban parking lots are most relevant, so the desired dataset will consist of images of parking lots in rather large cities. This task will be accomplished by gathering large amounts of aerial imagery, and using Open Street Map [24], an open source application with labeled parking lots, to crop these large images into more manageable and relevant images. This process is described in detail in Section 3.1.3. The images will then receive labels of where the parking spaces are in the images, which together with their accompanying images will form the basis of our solution, and be used to train a network that can detect parking spaces.

Chapter 2
Theory

2.1 Machine Learning

While Machine Learning can certainly be used as a broad, including definition, for this report it will be used in a quite more narrow way. Machine Learning, for what this report is concerned, is the way certain software gains inherent understanding of a problem, without being provided any explanation or explicit features. The system rather gains this understanding from positive and negative examples of the problem it is designed to solve. Machine learning is typically based on operation heavy problems, where the operation part can easily be extracted and automated. Machine Learning is usually split into two different categories - supervised learning and unsupervised learning. The main difference is that supervised learning has some kind of label coupled with its data, i.e. a correct answer to associate with it, while unsupervised does not. The process is somewhat different depending on the type, and the problem. Typically however, classification is done using supervised learning as its labels allows for the model to be evaluated on how well it did in hindsight. The classification equivalent for unsupervised learning is called clustering, which is the process of implicitly grouping data objects based on its features. After all the system cannot verify what the objects actually are, so it rather extracts features based on the input. A visualization of the two different cases can be seen in Figure 2.1. This report will only consider supervised learning, as it is best suited for classification, which is the problem described in this report.

In general, and certainly for the machine learning algorithms described in this report, machine learning is built on the basis of defining a dataset containing examples of data relevant to the problem. This data is then labeled according to what the algorithm needs, which vary considerably depending on problem and system. Most commonly, the dataset is divided into a training set and a testing set. The algorithm is then designed and specified. Then the training phase starts, where the dataset is fed into the system. The purpose of the training phase is to learn not explicitly but rather implicitly the necessary features defined by the problem. The system receives the training set as input data, along with its labels. The training is certainly algorithm specific, but the purpose of it is to give the system the concept of the data being presented to it. Commonly, the test set is presented as a way to validate the performance of the algorithm, and as such is frequently tested during training to ascertain the capability of the system. The

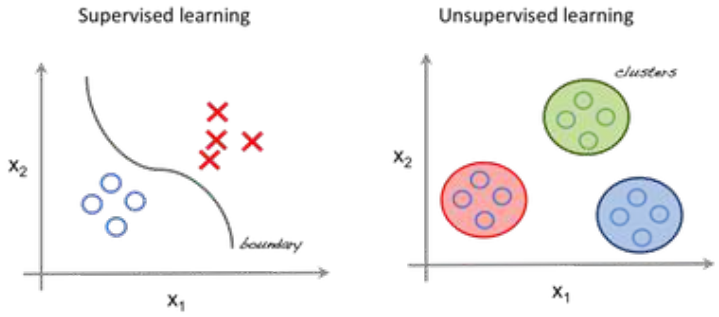

Figure 2.1: Examples of supervised and unsupervised learning, respectively. The supervised learning example is for classification, and the unsupervised example is for clustering.

procedure is usually the same as for the training phase, but the key difference is that the system has not trained on this data, and as such can reliably be used as a metric for evaluation. Training is usually conducted until the validation performance no longer improves.

Machine learning, and specifically neural networks, have many practical use cases, object detection and image recognition being most popular. An example of such is the famous MNIST classification challenge, where the task is to train a function or model to accurately classify images of handwritten digits, as illustrated in Figure 2.2. The type of machine learning algorithm that by far outperforms all other techniques, are namely neural networks. Lots of images of handwritten digits along with corresponding ground truth labels are fed into a neural network, which processes the input and outputs a prediction of what number or class it thinks it is. At first the model is guessing completely random, but with the more samples it is shown, the more accurate predictions it will make. After the training phase, the model is evaluated on unseen test data, which also has corresponding ground truth labels. This is to properly measure the accuracy of the model. With the problem of classifying handwritten digits, neural networks does in fact perform generally as good as, or if not better, than most humans.

2.2 Neural Networks

An Artificial Neural Network, commonly referred to as a Neural Network, is a collection of nodes, or neurons, and the connections between them. It is essentially a mapping $y = f(x; w)$ from an input $x = (x_1, x_2, ..., x_m) \in R^m$ to an output $y = (y_1, y_2, ..., y_n)$ given the weights $w = (w_1, w_2, ..., w_p)$. The network is built up by several layers — an input layer, an output layer, and several hidden layers. Traditionally, each node in one layer is connected to every other node in the next layer. The connections are represented by weights w, one for each connection. See Figure 2.3. During training, these weights are updated, and it is in this respect that the network learns, whatever its task might be. All inputs are multiplied by their corresponding weights into the result node, and summed as input for

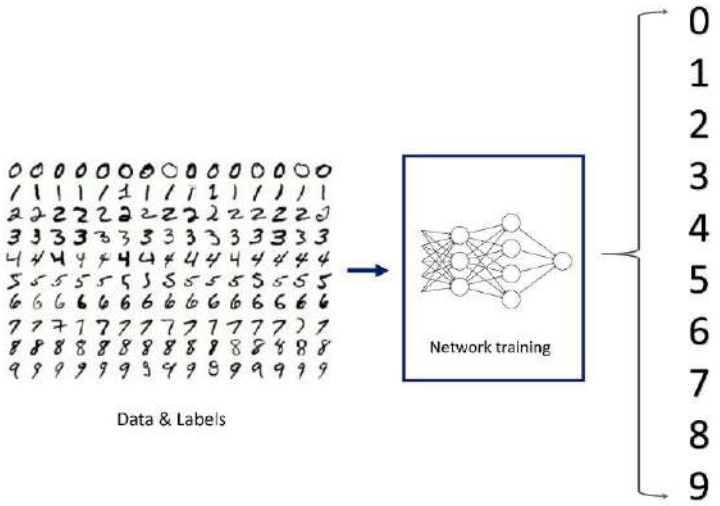

Figure 2.2: MNIST image data of handwritten digits along with corresponding ground truth labels are fed into a neural network, which outputs predictions of what numbers or classes it thinks they are, based on earlier examples from training. The model's accuracy can be measured by comparing the predictions with the ground truth labels. Image by Basia Fusińska [1].

that particular node. There is also commonly an activation function to keep the resulting values within some interval. This procedure continues throughout the entire network, and then the weights are updated according to some function. Thus, the inherent knowledge of the problem is represented simply by the complete collection of weights for the trained network.

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks, or CNN, is a subclass of Neural Networks. Similarly to Neural Networks, it consists of input and output layers. What distinguishes CNNs however, is the multitude of hidden layers in between. The layers commonly used, in some combination, are:

- Convolutional layers - In these layers, which give CNNs its name a filter is applied to the input, using a sliding window. This window is in fact a matrix consisting of the weights for this particular layer. The window slides over the image and performs element-wise multiplication on the input and the weights, which are sent to the next layer as input. The reason why image classification is usually connected to CNNs is because the weights, or filters, for each layer is rather small. Using a traditional network structure, the respective amount of weights would be massive. One for each connection between all the layers. Thus, defining a filter which gets applied to all pixels of the image means memory usage and training time is reduced. Each

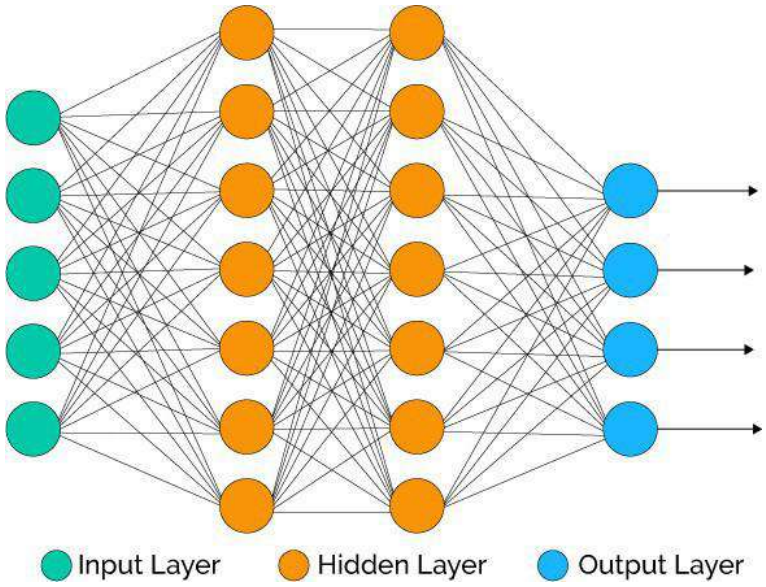

Figure 2.3: A sample neural network, by Conor McDonald [2]. Circles are nodes of the network, and the edges are represented by individual weights.

convolutional layer can be seen as a feature extractor, producing a feature map which is the summary of all pixels of the input, after going through the convolution process.

- Pooling layers - A pooling layer simply combines its multi-neuron input into one neuron output for use in the next layer. The pooling effects vary, and a common example is max pooling, which simply outputs the max value of the input neurons.
- Fully Connected layers - These layers connect each neuron from the previous layer to each neuron in the next. This is quite similar to how layers in traditional neural networks act. The purpose of these layers is to act as high level deciders, consolidating the information provided by previous more specific layers.
- ReLU layers - ReLU stands for Rectified Linear Unit, which applies the activation function

  $$\phi(t) = \max(0, t)$$

  i.e. sets negative values in the input to 0, while preserving positive values. This increases the non-linearity of images in the network, in order to be able to treat them more reliably. ReLU is usually done directly after convolutional layers.
- Loss layer - Loss layers apply the specified loss function, in order to penalize when the network performs poorly during training. See Section 2.3.

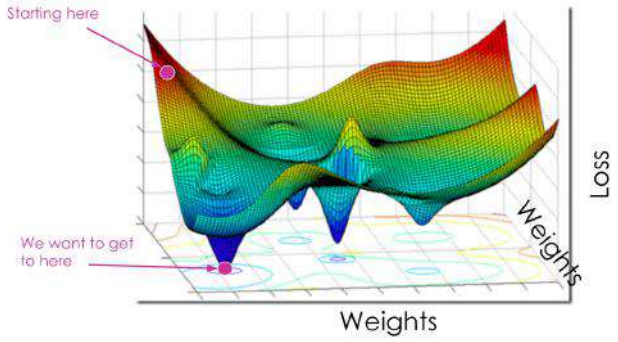

Figure 2.4: Numerous value combinations of weights with corresponding loss value plotted as a landscape, by Rosie Campbell [3].

2.3 Loss functions

A loss function is a measure of how well a model performs for a given problem. Rather than solely looking on accuracy, a loss function generally takes more variables into consideration, such as penalizing false-positive detections, outliers, overlapping detections, etc. It acts as a lens that the model looks through, steering it on where to focus in order to learn. Thus the loss function should generally be tailored to its specific problem, since there is no loss function that works perfectly for all types of data.

A commonly used loss function is Cross Entropy, where a separate loss for each class label per observation is calculated, and then summed. The cross entropy loss is described as:

$$-\frac{1}{N}\sum_{c=1}^{N} y_{o,c} \cdot \log(p_{o,c})$$

Where N is the number of classes (dog, cat, fish), y is a binary indicator (0 or 1) if a class label c is a correct classification for observation o, and p is the predicted probability that observation o is of class c.

The goal is to minimize the loss function. This can be visually represented by plotting the weights between the neurons together with their loss value. This forms a sort of high dimensional landscape that the model is sliding or traversing down from using gradient descent to reach the bottom most point. Unfortunately, the landscape is usually full with local minima. Getting trapped in one and not reaching a global minima, which of course is desired, is what makes this task hard. Many optimizers tries to tackle this using various techniques.

2.4 Regularization

In the sense of over-fitting, the model tries too hard to learn every specific detail in the data it is exposed to, as seen in Figure 2.5. This is not good since such details are usually not generic. The model may perform exceptionally well on training data, but terribly bad on yet unseen data, as seen in Figure 2.6. To prevent this, a regularization term such as $\|w\|_{\ell_2}^2 = \sum_{i=1}^{n} w_i^2$ is often added to the loss function. See Section 2.4.1.

The regularization term typically penalizes large weights and forces the model to strive for small weights, as large values are typically responsible for over-fitting, being dominant in the decision making. Although having too small weights is not desirable either due to the increasing risk of under-fitting, meaning the model gets too basic and cannot generalize properly. The task, thus, is to balance the regularization amount so that the weights get just right.

2.4.1 Weight decay / ℓ_2

The most popular regularizer is called weight decay, and as described makes sure that no weight gets too large. It does this by measuring the weights with either ℓ_1 or ℓ_2 norm, where ℓ_2 is most commonly used. The loss function with the added weight decay regularization term is as following:

$$Costfunction = Loss + \lambda \cdot \sum_{i=1}^{n} w_i^2$$

Where λ is the regularization hyper-parameter, or weight decay parameter, multiplied by the squared sum of all weights. This extra term forces the weights to decay step by step closer to zero, but not exactly zero.

2.4.2 Batch normalization

Batch normalization is another commonly used regularizer. It takes the outputs for each layer and normalizes the values between 0 and 1. Note that this is not the same as weight decay regularization, as weight decay only penalizes large weights before this step. Effectively, this makes the layers in the model more independent of each other, and also speeds up the training process.

2.5 K-fold cross validation

K-fold cross validation is a method used to utilize as much of the available dataset as possible. It works by splitting up the dataset into K number of different training/test sets. The test sets needs to be unique for each split, meaning no samples should be re-used between them. These are then trained on separately, resulting in K number of different models, each with their own test error/loss. Lastly, the mean error of all models is calculated by summing them and dividing by K. This mean error is more general over the dataset than the original error from simply one sole model, and all data is used as validation at some point. Finally, a model

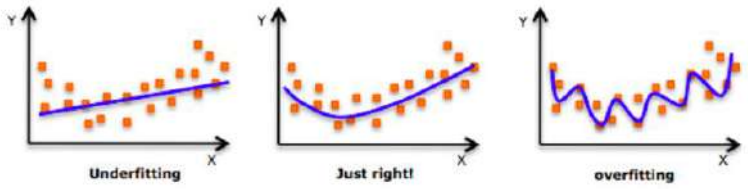

Figure 2.5: A comparison between underfitting and overfitting models. The blue lines are the outputs from the model, and the red dots are ground truth samples, by Shubham Jain [4]

.

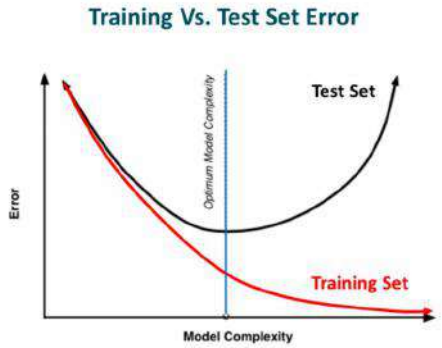

Figure 2.6: Overfitting train loss vs test loss. If a model is trained for too long, it starts to overfit on the training data, resulting in poor performance on other test data. The best model to use is where the test loss reaches its minima, by Shubham Jain [4]

.

is trained using the entire dataset as training data. The loss for this model is then assumed to be the generated estimate.

Figure 2.7: Visualization of K-fold cross validation, by Karl Rosaen [5]. The validation loss for each subset of the model is summed together to achieve an average total validation loss.

2.6 Networks to Evaluate

2.6.1 YOLO v3

You Only Look Once, or YOLO [25] is a bounding box object detector that outputs a bounding box for each detection with corresponding pixel coordinates, together with a confidence percentage score. Unfortunately, the bounding boxes do not provide any rotation information of the detection, only the boundaries as of where the object is within. YOLO is famous for its speed optimization as it is entirely built in C with the Darknet implementation. The network gets it name from only passing the image through the network only once, unlike most R-CNNs which will perform multiple detections on different regions. It achieves a high frame rate without sacrificing too much accuracy compared to other networks. By using YOLO, which is generally worse than RetinaNet and Mask R-CNN, it would provide an idea whether our problem is easy or hard. If it is easy, then YOLO would most likely be the most preferable solution as it is the fastest to perform computations.

2.6.2 RetinaNet

RetinaNet [22] essentially builds upon the same principles of YOLO. The main difference being the usage of a different loss function. RetinaNet uses a newer Focal Loss compared to the cross-entropy loss of YOLO. The idea behind Focal Loss is to change the weights in the loss function relative to the accuracy of the

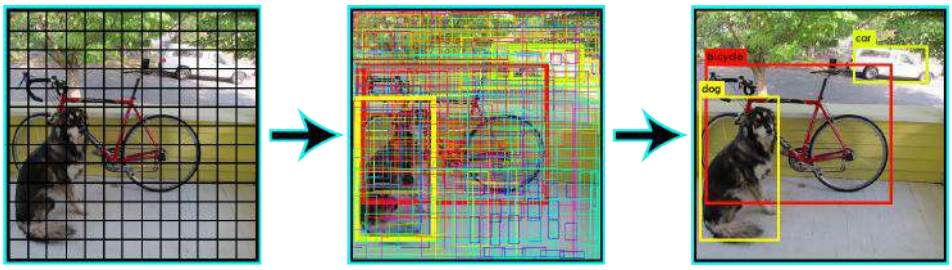

Figure 2.8: YOLO example detection, by Joseph Redmon [6]. The image is divided into a grid, where each cell tries to predict one object. Bounding boxes are then estimated, and the bounding boxes with the highest scores are kept as final predictions.

detection. Easily classified cases gets an assigned reduced loss while hard cases gets an increased loss. This effectively forces the network to put more emphasis on hard, misclassified examples. Unlike YOLO, RetinaNet uses a ResNet as a backbone structure to generate feature maps. Since ResNet is much larger and more complex, RetinaNet does take longer to train, but does generally achieve higher accuracy scores compared to YOLO. This is why RetinaNet's performance could be an interesting candidate for a solution to the problem.

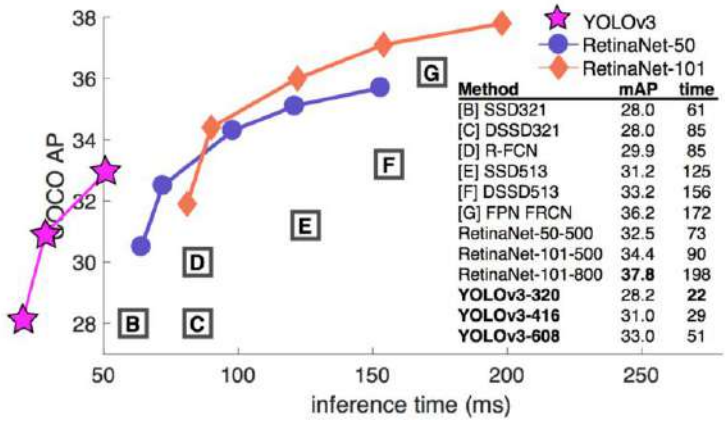

Figure 2.9: YOLO vs RetinaNet in speed and average precision for the COCO dataset, by Joseph Redmon [6].

2.6.3 Mask R-CNN

Similarly to RetinaNet, Mask R-CNN [23] also uses ResNet as a backbone structure. The main difference being that Mask R-CNN is a two-stage classifier whereas RetinaNet and YOLO is a one-stage classifier. The first stage is Regional Proposal

Network (RPN), and the other is the actual mask network. Two-stage classifiers are slower to compute, but generally provides better accuracy. Unlike YOLO and RetinaNet, Mask R-CNN not only provide bounding boxes for the detections, it also provides the corresponding segmentations. This means it outputs is a pixel- to pixel-wise mask of exactly where the objects are in the image, instead of highlighting them with a rectangular bounding box. This is great because it automatically solves a substantial problem compared to YOLO and RetinaNet - the orientation of the parking space. As Mask R-CNN is a two-stage classifier, it would most likely be the slowest of the networks. However, if accuracy seems to be an issue rather than speed, then Mask R-CNN could be the model of choice.

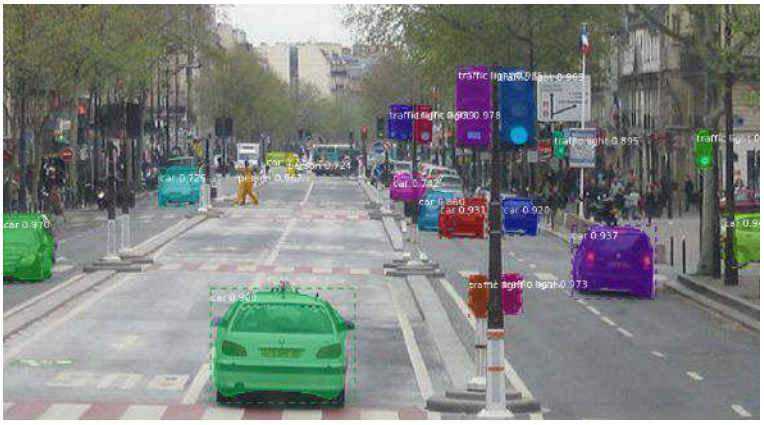

Figure 2.10: Mask R-CNN example detection, by Github user Matterport [7]. Each bounding box is accompanied by a mask and a confidence score.

Chapter 3
Method

This chapter covers the methodology to achieve the goal of the thesis. Firstly the dataset will be specified, followed by the process of manually collecting it and then how it was annotated from scratch. Finally, the three different neural networks will be presented, as well as some finalizing adjustments.

As no other datasets were readily available, a development of an annotation GUI was necessary in order to speed up the laborious process. This was done through clever techniques where a user could annotate multiple parking spaces at once. As seen in Table 4.1, the amount of annotated parking spaces got up to roughly 135.000, which would never have been feasible if the GUI had not been developed. Nonetheless, the data collection process was incredibly extensive and consumed a major amount of time of the project. The different phases of the method will now be presented in chronological order, following the flow of our development.

3.1 Dataset

3.1.1 Data Specification

For the problem to be clearly defined and feasible, the data needs to be clearly defined and feasible. This is particularly true for machine learning applications, where the result will never be better than the data the model is trained on. With this in mind, some simple heuristics had to be defined regarding the data for the problem. According to the problem description, the dataset should consist of satellite or aerial images. Originally only satellite images were considered. However, these rarely have the spatial resolution needed for proper evaluation, so aerial imagery was also included. This, then, provides another necessary requirement. The images need to be of a high enough spatial resolution in order to distinguish independent features, in this instance particularly parking spaces and cars. Furthermore, while not a specific requirement, the images were chosen in predominantly urban areas, where intuitively a higher density of parking lots and indeed parking spaces in general are present.

The individual annotations necessitate some specifications as well. In particular, what should be considered a parking space and what should not? Thus, it was defined that parking spaces are considered as such only if they have some

kind of clear marking on the ground encapsulating the space. This means that parking lots without visible parking spaces ought to be ignored, see Figure 3.1. The solution to the problem, that is, identifying individual parking spaces, for that instance would be ambiguous, even for a human agent. Furthermore, unseen parking spaces completely occluded by e.g. overhanging trees should not be considered positive examples, even if it is reasonable to assume there should be a parking space, given it is in the middle or at the end of a row of parking spaces for example, see Figure 3.2 and Figure 3.3.

Figure 3.1: An example of a parking lot without any visible parking spaces.

Figure 3.2: An example of a parking lot with some spaces occluded by trees.

Figure 3.3: The same image, with a classification overlay. Green spaces are correct (positive) and red are incorrect (negative).

As the problem is to identify parking spaces, both empty and occupied spaces are to be considered, and furthermore no discrimination is to made to those instances in regards to each other, i.e. both empty and occupied parking spaces are considered positive examples. Parked cars, which are illegally or incorrectly parked, are not considered positive examples however. If an incorrectly parked car is situated partially but not entirely in a parking space, the parking space itself should be considered as a positive example, and not the car. Lastly, all spaces should be perfectly rectangular, symbolizing the space wherein the car can be situated and be considered a legal parking.

3.1.2 Data Acquisition

The next step was to find appropriate data providers given the data specifications. As it turns out, public license data which is readily available for free and also complies with the need for high detail spatial resolution, is quite rare. A large amount of research was conducted to find appropriate providers and determine whether the data in question was free and legal to use. A natural choice would be Google Maps, which obviously have a large amount of data including high resolution aerial imagery. However, Google are quite particular in their data not being stored or altered in any form. Especially not training any neural networks on it besides Google themselves. Thus, Google could not act as a data provider. A service of interest was certainly the European Space Agency (ESA), in particular the Sentinel Missions, which in fact specifically produce data for scientific research and development [26]. The problem is that while the data is extensive, the satellite imagery provided is of rather poor spatial resolution. This data is thus better suited for environmental applications, which it has been used for previously, such as by P. Abhisek et al. where several machine learning techniques were explored to predict different types of land cover [27]. A similar service, the United States Geological Survey (USGS) for the U.S. however, does provide adequate data which is free to use, and using their Earth Explorer web service data can be easily defined and downloaded [28]. The data gathered from the USGS was high resolution orthoimagery. This means that the image has been geometrically altered in order to preserve scale and relative distances, which is helpful given the large size of the images. Another provider came in the form of Lantmäteriet, a Swedish government agency which among other things keeps orthoimagery of Sweden geography. Lantmäteriet provide this information for use in education purposes.

3.1.3 Parking Lot Extraction

The dataset necessitated images of parking lots, rather than images of urban environments in general. With this in mind, a solution was developed to automatically extract parking lots from the images in the dataset. Open Street Map (OSM) is a free and collaborative map of the world under an open license, which anyone can contribute to [24]. In OSM, users can manually annotate regions and locations on the map which might be of interest to other users, among other things parking lots. Conveniently, these parking lots are constructed with polygons, with each point in the polygon representing a GPS coordinate. Furthermore, OSM offer an extensively documented API called Overpass [29]. Using this API, all parking lots within a certain area was extracted, namely the area for each individual image in the dataset, as this information was also available. Overlaying the extracted parking lots over the actual image yielded impressively accurate results in most cases, as can be seen in Figure 3.4. The original images were then cropped according to these polygons, which resulted in the images for the final dataset. A minor padding was added to all resulting images to allow for errors in OSM and to introduce some local context for the parking lots. Note that completeness was not necessary in this case, as only some and not all of the parking lots are needed. Some examples of the resulting images can be seen in Figures 3.5 and 3.6. Images from Lantmäteriet did not have properly scaled GPS coordinates, so rather

Figure 3.4: OSM data overlayed on a test image. The drawn polygons on the image represent individual parking lots.

Figure 3.5: Example image of an extracted Washington D.C. parking lot.

Figure 3.6: Example image of an extracted Santa Fe parking lot.

than using OSM to extract parking lots, they were randomly cropped and then validated manually. Only images with parking spaces were kept in the dataset. Such an example cropped image can be seen in Figure 3.7

Figure 3.7: Example image of a randomly cropped section of Malmö.

3.1.4 Annotating Data

A very large amount of time was spent on annotating all of the available data. Given that no existing datasets were openly available, the annotating had to be done manually. The desired form of annotation was a rectangle overlying the space a car can occupy, as described in Section 3.1.1. This uniformity of annotations was utilized to assist the annotation process, as will be described in the following sections.

Annotator GUI

A Graphical User Interface (GUI) was implemented in order to easily annotate multiple parking spaces at once. The front end of the application is a bare bones HTML and jQuery solution, where the image to be annotated takes up the majority of the screen. The image can then be drawn on using an HTML canvas, using simply the cursor.

Optimizing the number of clicks to annotate the dataset is essential in order speed up the laborious annotation process. More clicks will vastly increase the amount of time needed. So instead of naively clicking down every corner of every

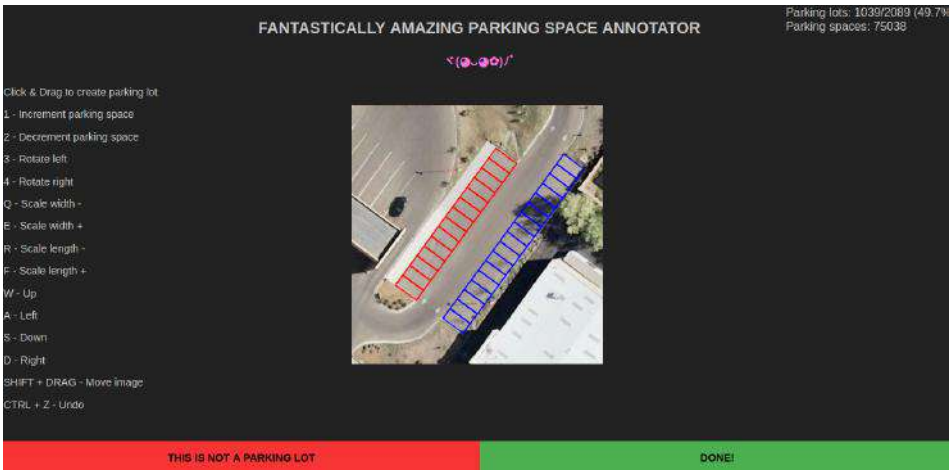

Figure 3.8: Screen shot of the annotator GUI. The images can be drawn on and then saved.

parking space, it is better to make use of the fact that parking spaces tend to be clustered together in rows. With the usage of vector calculations and trigonometry, and a rough estimate of the width of parking spaces the number of clicks for any arbitrary parking cluster was first cut down to only three clicks, as seen in Figure 3.9. The first click determined the starting point, the second to determined the length and orientation, and the third determined the width. This was later optimized down to only two clicks, as seen in Figure 3.10. Instead of marking the length, width and orientation of the row as two separate clicks, it is possible to combine these into just one click by simply diagonally marking the parking cluster. Drag and release. Although this produces infinitely many orientation solutions since there are no constants to lock down into. So by making use of the fact that the images have roughly the same spatial resolution of 0.1m/pixel, any arbitrary parking space will therefore always be around 50x20 pixels (5x2m) in size. By using these constants, it is then possible to calculate the rest of the points and orientation of the parking cluster. Simply with a diagonal two-click drag and release. The GUI then fills in the blanks and distributes parking spaces evenly as where they are most logical to be, and draws them on the HTML canvas. The user is then able to later fine-tune it to add more/less spaces in the row, adjust the width, length, orientation and so on down to pixel level.

The drawn annotations are then saved as a collection of four-point 2D polygons each representing a parking space, in a `.txt` file corresponding to the image. All the images, and the generated files, are stored in the cloud using Amazon S3 Web Storage [30]. The application was hosted with a simple node.js back end, hosted on Heroku, a cloud application platform [31]. This way, the application could be accessed and used from anywhere with a browser and an internet connection. The application has two buttons - one to signal that the image is completely annotated, which marks the file as annotated in S3, and saves the accompanying `.txt` file, and another which marks the images as incorrect in some way. Perhaps this is from

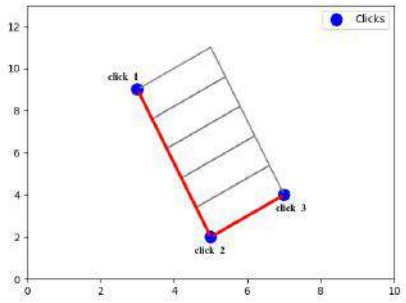

Figure 3.9: 3-click parking cluster generator.

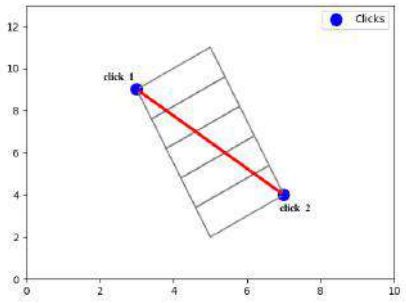

Figure 3.10: 2-click parking cluster generator.

an error in the original OSM extraction, or the parking lot does not adhere to the data specifications, as described in Section 3.1.1. After either button is pressed, a new image is loaded from S3, which can then be annotated, speeding up the work flow. Using the GUI was essential to effective annotation, and without it nowhere near the same amount of images could have been annotated. Thus, the time it took to develop it was absolutely worth it.

Network Specific Annotations

After annotating all the images, the annotations needed some alterations in order to bring them them to the input standard for each specific network.

- YOLO - The YOLO network requires a `.txt` for each image, with the same name as the image, with each annotation on one line of the file. The lines in the file have the following structure: `class x y width height` where x and y is the middle pixel coordinate of the bounding box.
- RetinaNet - For RetinaNet, a CSV file is defined for the entire dataset, where each line represents an annotation. Thus, images with several annotations should have one line per annotation in the CSV file. The lines in this file are in the following format: `path/to/image.jpg,x1,y1,x2,y2,class_name`, where $(x1, y1)$ represents the top left pixel of the bounding box, and $(x2, y2)$ represents the bottom right pixel. The classes should also be defined in another CSV file with an ID mapping. In our case, this was simply a one line file with the following line: `parking, 0`.
- Mask R-CNN - This is certainly, and unsurprisingly, the most complex annotation format. The annotations are defined in a list of regions for each file, written in a `.json` file, with one line per image. An example can be seen below.

```
{"1418_17_00.png423006":
    {"fileref": "",
    "size": 423006,
```

```
        "filename": "1418_17_00.png",
        "base64_img_data": "",
        "file_attributes": {},
        "regions": [{"shape_attributes":
                {"name": "polygon",
                "all_points_x": [55, 60, 72, 65, 55],
                "all_points_y": [85, 90, 72, 83, 85]},
                "region_attributes": {}}]
    }
}
```

Each (`"all_points_x"`, `"all_points_y"`) pair represents a polygon which is an annotation. The example above thus only has one annotation. Additional annotations are appended by adding additional entries to the `regions` list. In this way, each image is represented by one dictionary in the json file.

All the annotation conversions were done using scripts to automatically generate files and annotations using the original GUI annotations.

3.1.5 Image Manipulation

After being annotated, the images went through a manipulation stage. There were a couple of reasons for this:

- The images needed to be roughly the same size, as R-CNNs will commonly resize input data to be the same size. In this case information would be lost for very large images.
- Copying and manipulating the data meant the size of the dataset could be expanded, using the copied and modified image as different data points.
- Using random manipulations meant the variance of the dataset would increase, in theory increasing the generality of the trained model.

With these points in mind, the following manipulations were made. First, large images were segmented into smaller images. These images then replaced the original image in the dataset, along with their new annotations, according to the notations described in Section 3.1.4. Each image was then copied, and the following manipulations were made to each copy:

- With a 50% probability, the image was blurred using a Gaussian blur with a kernel size between 2 and 10, with a discrete uniform distribution.
- Again with a 50% probability, a random Gaussian noise was added.
- The image was rotated, either 90, 180, or 270 degrees.
- With a 50% probability, the image was flipped.

The annotations for each copy were then adjusted to align with the changes made. Thus, the size of the dataset was effectively doubled. All the steps described above were done automatically using scripts. An example manipulation and its original image can be seen in Figures 3.11 and 3.12, respectively.

Figure 3.11: Example image of a modified parking lot.

Figure 3.12: Image of the original parking lot.

3.2 Networks

The network evaluation phase started as soon as a preliminary rudimentary dataset was completed, and then continued parallel alongside development of the dataset, given the extended time it takes to train a CNN.

3.2.1 Comparing Different Networks

The various networks that were compared use different evaluations to ascertain their performance. This meant that for comparing the different models, these numbers (commonly the loss generated by the loss function) cannot be directly compared. Rather, the evaluation for the networks was done by running them as classifiers on some un-annotated validation images. The performance of these models on the images could not be numerically determined in regards to each other, and as such were compared intuitively, by analyzing the different networks' classifications, using metrics such as false negatives and false positives. While more rigorous methods could have been applied here, this was not desirable using the constantly changing not yet complete dataset, and when certain flaws in the model were apparent, moving forward with more successful models and networks were much preferred.

3.2.2 YOLO v3

The YOLO network was trained using this [32] Github repository by user AlexeyAB. This repository was forked from the Darknet repository [33], by user pjreddie, who also wrote the seminal paper on YOLO v3 [25]. The Network was trained using its own network structure, called Darknet53. While installation was somewhat problematic, the repository holds plenty of information and trouble-shooting. YOLO was in all instances run on Windows, on a computer with either a GTX 1080Ti or GTX 1070 graphics card. The implementation required a project built using Microsoft Visual Studio, with quite specific libraries and requirements. Some

hyper-parameters were changed and tested compared to the original settings, and new training runs was also done when the dataset was updated. Evaluation was based on the cross-entropy loss function generated during training, and calculating mean Average Precision (mAP) after training, simply:

$$mAP = \frac{\Sigma_{q=1}^{Q} \frac{TP_q}{TP_q + FP_q}}{Q}$$

Where Q is the amount of classes, TP_q the amount of true positives in class q, and FP_q the amount of false positives in class q. Note that for this problem only one class is present, so $Q = 1$.

3.2.3 RetinaNet

RetinaNet was built using the following Github repository [34] by user fizyr, which is a Python3 and Keras implementation of the network. Using Keras was quite convenient and efficient, in contrast to YOLO which demanded a project built in Microsoft Visual Studio. The Network was trained using the ResNet152 backbone. RetinaNet was trained solely on a desktop computer running Windows, using a GTX 1080Ti graphics card. A virtual environment for Keras with GPU support was used to run training. The model was evaluated by choosing the best performing training weights based on the best mAP for for all epochs, and converting them to an inference model. This inference model was then used as the final model.

3.2.4 Mask R-CNN

Mask R-CNN was built using the following Github repository [7] by user matterport, which is a Python3, Keras and Tensorflow implementation of the network. The network was trained using the ResNet101 backbone and with pre-trained weights from the COCO dataset. Mask R-CNN was initially trained on a desktop computer running Windows, using a GTX 1080Ti graphics card. This was later moved to Zenuity's GPU farm architecture, running on Tesla V100-SXM2 GPUs, with a video memory of 32GB. These were accessible via SSH and run specifically using pre-built Docker images.

3.3 Evaluating and Improving the Model

This section will essentially be entirely focused on the Mask R-CNN network. The reason as to why is that it was found quite early on that this network outperformed YOLO and RetinaNet. As such, the decision was made to continue development on Mask R-CNN, and to abandon the other two, as it would be much too time consuming to continue work on all of them. This is elaborated further in Section 4.2.1.

3.3.1 Dataset Iterations

The dataset went through some different iterations, which were used as milestones in testing different versions of the networks. While the dataset was expanded and

configured, the configurations for the different networks were largely unchanged, as the majority of effort was put into creating an appropriate final dataset. Given how long it takes to train a neural network, work was done on the dataset while the past iteration of the dataset was tested. The networks trained on each iteration of the dataset was also partly used to evaluate the completeness of the dataset and determine what further work needed to be done.

3.3.2 Parameter Tuning

Uncovering the optimal hyper-parameters to find the global minima of a loss function of a model is not an easy task. To some extent it is even impossible due its high dimensional space and multivariate nature. Unfortunately, randomly guessing in hope to find the optimal parameters were not sufficient as it would be too unreliable and time-consuming. A different approach was needed. An example of such was to train multiple values of the same parameter, and go with the one that provided the best loss trend before moving on to the next parameter. Following a sort of decision tree-like pattern. However, this strategy assumed all parameters were independent of each other, which of course they were not. But as there does not seem to be any other technique readily available, this informed brute force was determined to be the best way to proceed. The results from the different runs were automatically saved in event files and logged in Tensorboard, a visualization tool used for machine learning purposes to follow the process of training runs.

With the usage of Zenuity's DGX-1 GPU servers, this technique got a whole lot more obtainable as it made it possible to extensively parallelize the parameter checking. Running $4 - 8$ training instances simultaneously was of tremendous value, and sped up the parameter tuning process considerably.

Chapter 4
Results

In this chapter, the results from the project will be presented. The nature of our methodology necessitated that this be presented in a chronological fashion. Thus, a considerable part of this chapter will be dedicated to explaining intermediate results, that informed and explained the steps and actions taken after them. The alternative would be to explain decisions based on results not yet presented, which would damage the readability and cohesiveness of the report. Therefore, putting the information in this chapter seemed preferable, rather than to include them in the previous chapter, in order to streamline the relevant sections and to put focus a chronological development of the eventual final results. First, the different iterations of the dataset that were used and evaluated will be presented. Then, the process of comparing and deciding on a network will be explained, followed by the improvement phase of Mask R-CNN. Finally, the end results of the final network will be presented.

4.1 Dataset

In this section, the different dataset iterations that were actually used for the project will be presented. While the actual resulting datasets might not be completely relevant to the end product, we found it necessary to include them in order to illustrate our reasoning and changes to be made moving forward.

The complete dataset was compiled using images from USGS and Lantmäteriet. The images were from Washington D.C. and Santa Fe from USGS, and from Malmö from Lantmäteriet. The dataset went through several iterations during the process. At all the different completed stages of the dataset, one or more networks were trained in order to ascertain the completeness of the dataset. The dataset versions to be presented in this section were only the ones that were tested with network training runs, and versions to be considered complete. Numeric values for the dataset versions can be seen in Table 4.1. Specific explanations and motivations for each dataset version can be found in the following sections. To briefly summarize the findings, high image variance in the dataset seemed to imply the greatest results for robustness and reducing the amount of false positives, which should be desirable for this problem. Furthermore, as expected the bigger the dataset, the better the results. Thus focus was put on generating as many images as possible, in addition to introducing as many different looking images as

Version	# of images	# of spaces	# of empty images	P/N ratio
1	1541	43805	206	6,4806
2	3470	88148	544	5,3787
3	6322	88152	3394	0,8627
4	10781	134673	5605	0,9235

Table 4.1: Numeric values for the different dataset versions.

possible.

4.1.1 Version 1

The first version of the dataset only included images from Washington D.C. which were cropped. Images in this version had a max size of both height and width of 500 pixels, and thus were segmented to fit these values. The resulting segmented images from one original were thus of the same size. This also meant a few resulting images were empty, that is, images without any parking spaces in them, which resulted in the number of empty images seen in Table 4.1. The dataset version was considered complete when a total of 40,000 parking spaces had been annotated. No exceptional results were expected from this dataset, as only images from one source were included, and indeed the dataset was quite small. Nonetheless an initial dataset was necessary to start the network evaluation process. Only the YOLO network were trained on this dataset initially, and the dataset was intended to be used as a starting point to get the different networks initialized. RetinaNet was also eventually trained on this dataset. Early results were quite promising, and performed rather well on the test set, but virtually only on the test set. A low variance on the images was quite evident here.

4.1.2 Version 2

This version was quite similar to the previous one, with the exception that the data was augmented. In effect, each image was copied and had modifications done on it, thus doubling the size of the dataset. The augmentations done are described in Section 3.1.5. The same cropping technique from the previous version was still used here. In fact, aside from the augmentations, the dataset is virtually the same. The augmentation methods introduced in this version was kept through all following iterations of the dataset. All the different networks were run on this version, and thus this was the first version of the dataset where all networks were compared to each other. It was also the first version to be used for the Mask R-CNN network. Adding the augmentation seemed to improve the model performance, making the dataset and thus the resulting model more robust, in addition to of course increasing the size of the dataset.

4.1.3 Version 3

Two major changes were introduced in this version. First, some more negative images were introduced to the dataset, in order to bring the ratio of positive/negative images to around 1. This is a commonly used maxim in machine learning to increase context to the algorithm, along with learning from negative examples rather than only positive, thus not only understanding what is a positive example of a class, but also what is not. The negative images were taken from the past versions of the dataset, along with images which were mislabeled or unusable from the annotation process. This included e.g. parking lots without specified spaces, and buildings with parkings inside such as garages. Finally, some images were randomly generated from the original USGS data, verified to be empty, and then saved to the dataset. Second, a random re-size was introduced to attempt to increase scale variance in the dataset. This was only done on the modified data, and were defined as a re-size with a multiplier in the interval $[0.5, 3.0]$. Adding the negative images seemed to quite improve the results in regards to false positives, as fewer objects were incorrectly predicted.

4.1.4 Version 4

The changes in version 4 were numerous and quite considerable. The dataset needed to be consolidated into a complete version which could be used as a standard for the network parameter adjustment phase. First, more original data was introduced to the dataset. This came from two sources, the first in the form of USGS images from Santa Fe, gathered in the same manner as the Washington D.C. data. Santa Fe was chosen because of its distance to Washington D.C. in the hopes that the different environment and colour palette, and certainly different types of parking spaces, would introduce more variance to the dataset. The second type of data came from Lantmäteriet's data gathered from the Swedish city Malmö. Using also Swedish data was originally the intent, but it turned out to be quite a cumbersome process. This data did not have the same spatial resolution as the USGS data, and therefore hopefully could also make the model performance more general. In addition to this, Swedish parking spaces look quite different compared to US ones.

The negative images in this version were also re-examined. The previous completely random images were removed, and new purposeful negative data was introduced. From previous runs of the model it had become apparent that the algorithm had issues with false positives, in particular large intersections with lots of lines and moving cars, as well as buildings with box-like patterns on the roof. Thus, a lot of negative images were manually found, cropped and added to the dataset with high concentration of these particular locations. This included images from all datasets. Another class of negative images were also added, which was images of sport courts such as tennis courts, basketball courts and football courts, which it seemed the model had problems with, in particular tennis courts. The blurring augmentation was also lowered, as it was deemed that blur was too intense and didn't contribute with enough information. The final thing that changed in this version was the introduction of padding. The decision was made to, rather than re-sizing the cropped images, keep the original size and spatial resolution and add

padding up to 850 pixels in width and height. This way the feature that parking spaces tend to be roughly the same size is kept, and the images would keep their size in the network, and thus their spatial resolution, which would hopefully increase performance.

4.2 Networks

4.2.1 Initial Results

The networks were tested alongside developing the dataset, in order to assess early the strengths and weaknesses of each network early on, as well as to highlight inherent flaws of the emerging dataset. Below are some summarizing reflections for each network.

YOLO v3

YOLO was the first tested network, and the first network predicted to be an acceptable solution for the problem. Initial results seemed promising, although were somewhat deceptive. The detections drawn with YOLO's visualization tool were quite thick, alluding to good detections where in reality a lot of positive examples were skipped. An early problem noticed about YOLO was also that detected bounding boxes often overlapped, even when they were parallel, as it tends to generate large bounding boxes which encapsulates the entire object, but not very precisely. It often padded the actual parking space by a bit, surrounding the object but not very firmly. Of course, another problem was that YOLO only generates bounding boxes in the first place, thus necessitating another solution in the detection pipeline, to determine the orientation of the parking spaces themselves. Nonetheless, the results of early YOLO models were used as a baseline for the next-coming solutions.

RetinaNet

The first trained RetinaNet models were introduced for version 2 of the dataset, and its results were at the very least comparable to YOLO. After some iterations it became clear however that RetinaNet was less forgiving with overlaps as compared to YOLO. In particular, adjacent detections tended to not overlap as much. Furthermore, RetinaNet seemed to be more inclusive in its detections, perhaps not perfectly encapsulating all results, but it at least made more detections. However, a consequence of this was that RetinaNet was not particularly confident in its predictions, which was likely also because of its punishing focal loss function. As an added note however, RetinaNet's visualizing function actually does show the confidence of its predictions, compared to YOLO's which does not. As with YOLO, the detections are only bounding boxes.

Mask R-CNN

Mask R-CNN was the third and last network to be tested. It instantly proved to be more useful as compared to YOLO and RetinaNet, given the added benefit of its generated masks in addition to the bounding boxes. Early iterations of the network were quite positive, and the confidence of detections were quite higher than those of RetinaNet. Mask R-CNN also proved to be by far the best at reducing overlap, with virtually no overlapping adjacent detections. It was not without flaws however. It seemed to be the worst at ignoring false positives, so quite a few detections on intersections, roofs, and roads were present here, some with fairly high confidence. Solving those however, would make Mask R-CNN seem to be the prime contender for a good solution. Mask R-CNN seemed to be the slowest of the three networks, but for the presented problem this was considered to not really be an issue, so long as the performance is competent.

Deciding on a Network Structure

Quite soon after testing all of the networks for a few iterations, Mask R-CNN was chosen to be the best network going forward. To continue the analysis and development of a solution, one network had to be determined to investigate further, as doing so for all three would require a considerable amount of time that was simply not available. The inherent masking property of Mask R-CNN was simply too attractive not to explore further. Normally for Mask R-CNN data, objects have to be painstakingly annotated with drawn polygons, but this is generated by default given the nature of the problem - parking spaces are rectangular, so the mask drawing step is already completed from the beginning, as this also was an inferred property from the annotation stage. Thus a massive amount of information is gained instantly, proving a massive theoretical boon for Mask R-CNN. This, alongside pretty much non-existing overlap for detections, made Mask R-CNN the best solution considered. This decision was made during version 3 of the dataset, and as such following results were all from the Mask R-CNN network.

4.2.2 Mask R-CNN Improvements

After deciding on the Mask R-CNN network as the network of choice, work begun on improving the results. The Github repository that was used contains several training hyper-parameters to potentially adjust and improve the performance of the resulting model. For this project, around 15 different parameters were tested and sometimes changed, not counting different data configurations that also were part of the process. While a lot of these hyper-parameters certainly are related to one another, there was not enough time to test all possible configurations. This process is further explained in Section 3.3.2. The different tested parameters are listed below:

- `STEPS_PER_EPOCH` - How many steps are taken in one epoch, before validation and Tensorboard updates.
- `VALIDATION_STEPS` - How many validation steps are taken at the end of each epoch.

- `IMAGES_PER_GPU` - How many images per GPU is loaded for each step. Thus, `IMAGES_PER_GPU` * `STEPS_PER_EPOCH` is how many images are included in each epoch.
- `LEARNING_RATE` - This determines how much to change the weights depending on results from the optimizer. Therefore, this decides how fast the network learns.
- `LEARNING_MOMENTUM` - How much of the previous optimizer result's gradient to be taken into consideration for the next step.
- `WEIGHT_DECAY` - The factor used for weight decay regularization, see Section 2.4.
- `RPN_ANCHOR_SCALES` - Length of square anchor side in pixels for the RPN part of the network.
- `RPN_TRAIN_ANCHORS_PER_IMAGE` - The number of anchors to use in training the RPN.
- `DETECTION_MIN_CONFIDENCE` - The threshold to what is considered a positive classification for each ROI. Classifications below this probability are skipped.
- `RPN_NMS_THRESHOLD` - The threshold used for RPN proposals.
- `TRAIN_ROIS_PER_IMAGE` - Amount of ROIs to consider for classification for each image.
- `ROIS_POSITIVE_RATIO` - Percent of positive to negative ROIs used in training.

As for choosing the order of parameters, the parameters that intuitively lowers training time and impacts the results most were chosen to be tested first. With this in mind, the first parameters to be tested were the epoch parameters - `STEPS_PER_EPOCH`, `VALIDATION_STEPS`, and `IMAGES_PER_GPU`. These ought not to impact the resulting model, but rather together decide how each step and epoch is configured - how many steps and images per epoch, how much time should be spent on validation after each step, thus determining how often the loss results should be updated. After that, the learning parameters were tuned - `LEARNING_RATE`, `LEARNING_MOMENTUM`, and `WEIGHT_DECAY`. Together, these parameters decide how fast and efficiently the network learns during training, and it is a common maxim to start with such parameters in order to not waste unnecessary time on training going forward. After that, the next parameters were chosen in no particular order, except for when parameters reasonably seemed to be very dependant on each other. In those cases, one was tested directly after the other one. The method going forward for this stage was to generally look at the best trend of all values tested for each parameter, and use that value for all subsequent rounds of training. As none of these training runs were ever supposed to represent a finished model, the actual best values for loss were largely ignored, and focus was rather put on what parameter value represented the best improvements as compared to the other ones. The process has been summarized in Figure 4.1, where the parameter tested

is printed in the node, along with the chosen best value. The figure is structured chronologically from start to bottom, where horizontal nodes are tested at the same time. The first row of nodes were the initial parameters from the experimental phase. For each node, some amount of different values were tested, from 3 to 9. Tensorboard graphs for each parameter can be found in Appendix A.

4.3 Finalizing the System

4.3.1 K-fold cross validation

After the perceived optimal hyper-parameters were found, which resulted in a model that produced reasonably good results, k-fold cross validation was performed in order to get a glimpse of the model's actual true performance, rather than being biased on the point where the test set was split. In the case of this report, 5-fold cross validation was performed on the model and dataset.

4.3.2 Detection Script

Since Mask R-CNN could only process images up to a certain size before scaling them down, a detection script was implemented in order to bypass this. Scaling down images means information loss, and the intrinsic property of parking spaces always being roughly the same size in the same spatial resolution domain, would be lost. To still be able to process large images, the script cropped the image into smaller, yet overlapping portions, max 850x850 pixels, and ran separate independent detections for each sub-image. These were later combined into a full mosaic, which visually resulted in far better performance compared to detecting solely on the large image as its own.

4.3.3 Post-processing

Since the sub-images overlapped each other with a few pixels, it also resulted in some duplicate detections. These needed to be removed somehow, and so some post-processing was introduced. For each detection, the centroid was calculated and compared to every other centroid. If the distance was shorter than a few pixels, that meant that detection had a duplicate, and so was removed. Although this was not all that was implemented. The model also gave some false-positive detections such as on pavements, in bushes, in crossings, cars driving on the road, etc. These could also be removed with some post-processing, as they rarely had any neighbouring detections close to them. So during the same distance comparison step, a check for if a detection had any neighbouring detections within a given radius was added. If not, the detection was discarded. Arguably, this meant that some true positive detections would also be thrown away. However, parking spaces that are completely by themselves are much more rare than clustered ones, which could justify the usage of this post-processing trick. Of course, this is a trade-off that is completely up to the user to be used or not, depending on the use case. One last post-processing feature was also added. Namely, if a detection had any neighbouring detection with a low confidence score below a given threshold, then

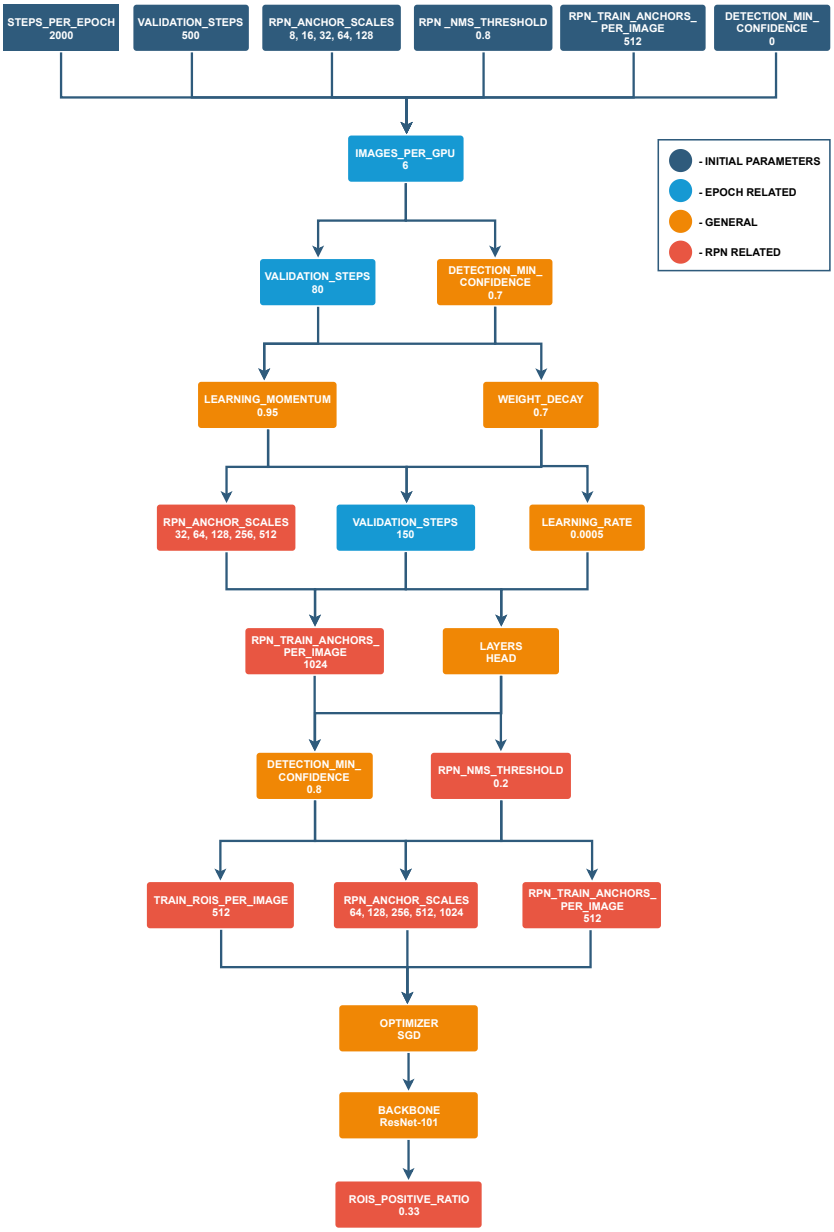

Figure 4.1: Flowchart of the hyper-parameter decision tree. Each box represents a tested parameter and it's decided best value.

both were discarded. This eliminated more false-positives in complex areas such as in crossings, as the model was quite unsure whether that really was a parking lot or not. Detections in real parking lots rarely had any neighbouring detections with low confidence, so this technique made use of that information. A before and after comparison of where these post-processing proposals has been applied can be seen in Figures 4.2 and 4.3.

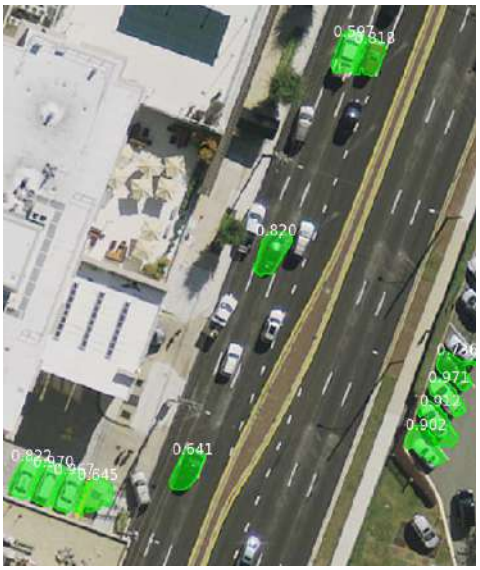

Figure 4.2: Example image before applying post-processing.

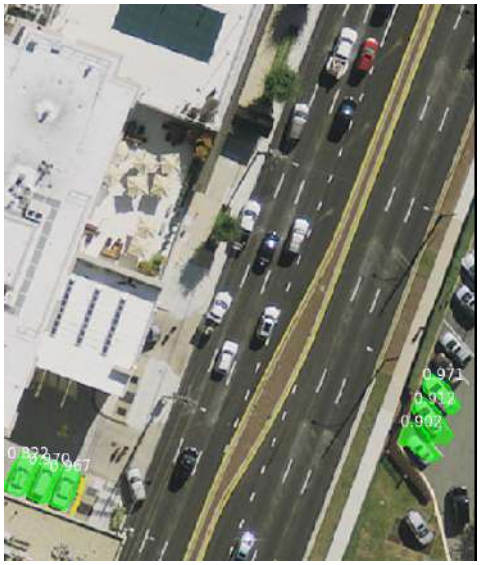

Figure 4.3: The same image after applying post-processing.

The post-processing algorithms were merely experimental and should not be considered a part of the resulting model, as they were not tested and validated in any rigorous manner. They acted more as ideas for any potential future use.

4.4 Detection Results

The resulting detection performance of the finished Mask R-CNN model was varied. It seems some images were easier to handle, while some remained particularly difficult. A couple examples of fairly competent detections are shown in Figures 4.4 through 4.7. The numbers associated with each mask is the confidence of the detection. While not perfect, these are certainly some of the better performances achieved.

Crucially however, the model performed mostly well in avoiding false positives, at least in most cases. In Figures 4.8 and 4.9, the model did actually correctly not label any parking spaces. This was a problem during earlier iterations of the model, where it would incorrectly find parking spaces in intersections and tennis courts, among other areas.

Figure 4.4: Good detection results of a parking lot.

Figure 4.5: Fairly good detection results of a parking lot.

Figure 4.6: Good detection results of a large parking lot.

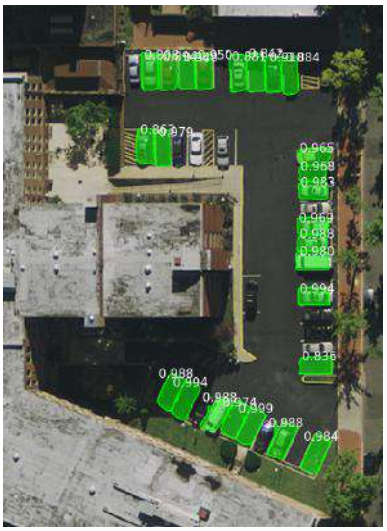

Figure 4.7: Good detection results of a small parking lot.

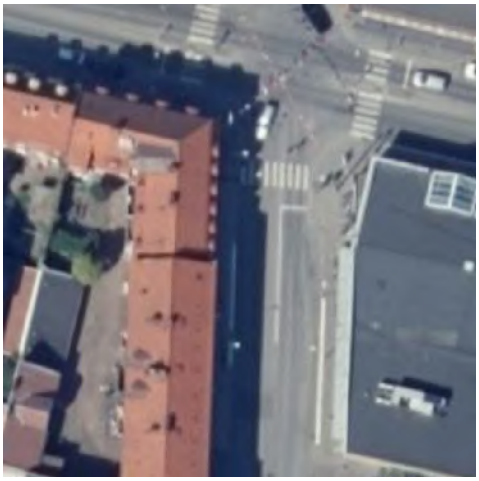

Figure 4.8: Detection results of a fairly large intersection.

Figure 4.9: Detection results of a tennis court.

Again, even here the performance was not perfect, see for example Figure 4.10. Sometimes, it was even quite bad, in particular for large intersections in Malmö, which also unfortunately had quite bad spatial resolution, resulting in fairly hard to distinguish lines, see Figure 4.11.

Indeed, images from the Malmö dataset were quite hard to detect properly, as seen in Figures 4.12 and 4.13. This might be because of the less defined parking spaces in Sweden, with the crosses indicating corners rather than fully drawn lines.

Overall, the detection results were not incredibly good, but certainly hint at a basic understanding of the concept of a parking space, so there is some success to draw from the project.

Figure 4.10: Detection results of an intersection in Washington D.C.

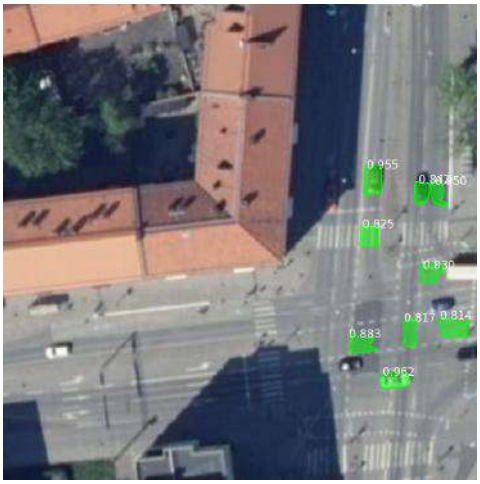

Figure 4.11: Detection results of a large intersection in Malmö.

Figure 4.12: Poor detection results from the Malmö dataset.

Figure 4.13: Poor detection results from the Malmö dataset.

Chapter 5
Discussion

5.1 Difficulties

While exploring the project, several difficulties arose which required solutions. Some were intrinsic to the stated problem, and some were merely a logical or technical challenge.

Right from the start we were faced with the trouble of finding the actual satellite/aerial images. Despite being easily accessible through many providers such as Google Maps, Apple Maps and Bing Maps, they were unfortunately not available under any public license. A user was not allowed to download, manipulate, distribute or specifically use Google Map imagery for machine learning applications in relation to autonomous cars. Though there did exist many other private providers that did in fact sell their own gathered satellite/aerial images for various data research fields. Examples are Geocento, DigitalGlobe, Apollo Mapping and Lantmäteriet. However, these were all very expensive solutions, and thus not really realistic for the scope of this project. Fortunately, one provider had none these problems, namely USGS. Covering large parts of USA, USGS provided many different kinds of geodata free of charge for anyone to use anywhere in the world. In particular very high resolution aerial imagery. This was vital for the project. It was also later found that Lantmäteriet offered their data free of charge as well, but solely for educational purposes at Swedish schools and universities. This data however was not allowed to be shared or distributed further. Converting the Malmö data from Lantmäteriet from the SWEREF coordinate system to longitude/latitude was also quite a difficulty. SWEREF is a plane projected coordinate system, so when trying to naively convert the coordinates to latitude/longitude, which Open Street Map uses, some projection falsities were thereby introduced. This could possibly have been solved if looked into further, but the choice was made to go ahead and include the Lantmäteriet data anyway. These images were simply manually verified and cropped to be relevant to the data specifications.

Another problem was to discern parking spaces from sport courts, such as tennis courts, basketball courts and football courts. Similar features of lines and edges in hard contrast existed in all of these elements. The initial suspicion was that the model had not yet been fully exposed to samples of sport courts. Additionally, cars and lines on roads and crossings caused similar issues. After the attempt of providing the dataset with more of these images, little improvement

was seen unfortunately. It may simply be that this is a hard problem to solve in its very nature, in that the relevant features are so similar.

Finding the optimal hyper-parameters for the model was also quite an obstacle, as it generally is for machine learning tasks. It would simply be too time-consuming and not realistic to, in a brute force fashion, test every possible relevant combination of parameters, though it would of course yield the best results. Blindly and randomly guessing different combinations would not be sufficient either. And so the approach of following a sort of decision tree like structure was performed by training and testing multiple values of the same parameter before locking it, and moving on to the next parameter. The decisions that were made can be seen in the appendix. The issue with this however, was of course that this method assumed all hyper-parameters are independent of each other, which they of course are not. This was thereby not the most optimal of strategies, but it seemed to be the only way to do it somewhat intelligently.

5.2 Limitations

The first and major limitation for the presented problem is that it is not a complete solution for finding parking spaces. Indoor parking is completely ignored, as is parking lots without specified spaces. Furthermore, even if a parking space is correctly detected, it might not be a legal parking spot — e.g. it may be reserved or time restricted. The task presented in this report however, does not concern itself with such limitations, and as such only the limitations explicitly defined by the problem will be described here.

5.2.1 Aerial Imagery

There are certainly quite a few current providers for satellite imagery, which are readily available. The problem with most of them, however, is that they seldom offer adequate spatial resolution to discern parking spaces, much less actually classify them. What seems to be generally preferred for this problem is rather high resolution aerial orthoimagery, which have better spatial resolution, and are furthermore orthorectified, similar to maps. These also have several available providers, but most of them are quite expensive.

Another problem with aerial imagery, or rather machine learning task with image analysis on aerial imagery, is that intrinsic camera properties are specific to the data providers. Properties such as grain, lighting, and intensity can differ greatly from camera to camera, provider to provider, whereas image recognition tasks tend to usually have a wide variety of angles, cameras, and instances. Thus, in order to gain high variance and generality on the dataset, having multiple providers would be preferable.

Finally, some consideration has to be put into how updated the aerial imagery is. Certainly, the images do not have to be live, or close to live, but they need to represent the appearance of the current parking spaces for the area you wish to classify, at least during the training phase. Furthermore, and perhaps more pressingly, the new input to classify must represent the e.g. parking lot as it exists

today in the world. After all, these results should be used with autonomous cars to inform them of current parking possibilities.

5.2.2 Parking Spaces

An inherent property of parking spaces, at least in the context of machine learning, is that they have rather few features. After all, a parking space is merely a drawn box on pavement, so there is not very much for the machine learning algorithm to learn. Furthermore, considering this very simple layman definition of a parking space, several instances of it can be found, which in fact are not parking spaces at all. This severely exacerbates the problem of false positives, as the leap from e.g. parking space to odd pattern on rooftop or intersection is not too large. This is also a problem for false negatives, as the dataset contains several negative images of similar instances which feature-wise seem very similar. This confuses the model considerably, making it quite difficult to make confident predictions. Granted, spaces can also be occupied, at which point it is still defined as a parking space.

Another issue with the generality of parking spaces is that geographical location is quite integral. In other words, depending on where the parking space is, it will look considerably different. A space in Canada may look different from a space in Iran, being constructed or defined differently, for example. Thus, to make a truly variant dataset and vicariously a general model, examples of parking spaces from all over the world need to be included, particularly considering the small amount of features connected to parking spaces, as discussed earlier. Not doing so would restrict the model's detection to really only perform well on a specific region, which is certainly viable as well. Indeed, this is the case for the solution presented in this report.

5.3 Conclusion

5.3.1 Evaluating Model Performance

The results provided by the finished model were not terrible, but certainly not quite satisfactory either. This was noticed early on, in particular when the final dataset version was introduced, see Section 4.1.4. This was when new, more challenging negative images were introduced, and a sharp rise in the validation loss was detected as a result. This indicated that the network had severe problem differentiating a parking space, in particular an empty one from for example lines in the road. Furthermore, the Malmö data was included in this version as well. This represented the largest diversity of types of locations and features ever introduced in the dataset, and as such the validation loss suffered. This did mean however, that the model was forced to learn more about generality, which might be considered a good thing. While using an earlier version of the model, or more specifically the dataset, would yield a better loss and as such a theoretically better performance, it was decided to keep the latest solution. This is because it is less restricted by specificity in the data and more in line with the described problem formulation. This also yields the greatest amount of information, be it positive or negative.

The final validation loss was reduced considerably however, with the tuning of parameters. During the lengthy improvement period the loss steadily dropped as more parameters were optimized. Thus this process had substantial effect on the end model and should be considered a successful procedure.

The end model certainly, although perhaps not performing as well as hoped, did show some remarkable traits. Except for in very problematic images, it had very few false positives. This was quite a concern in earlier iterations, where the model seemed to be rather forgiving for such errors. This was particularly noticeable in blank spaces of concrete between or around parking spaces, as well as on rooftops and parking lot entrances and exits. For the most part, it also produced quite uniform and complete rectangles of the parking spaces, something that had not been observed since the dataset was very tiny.

5.3.2 Resolution To Problem Description

The problem description posed the question if locating parking spaces from satellite or aerial imagery is feasible using machine learning techniques. Our solution does certainly indicate that it is possible, albeit the solution itself is not quite the full answer to the problem. The final model of our solution surely does classify parking spaces, the main issue is simply with its completeness. In that respect, while the result might not be considered a full solution to the presented problem, it is indeed a successful solution. Anyone wishing to expand and improve upon our solution would do well in reading our suggestions in Section 5.5.

5.3.3 Significance Of Solution

While not performing as we had initially hoped, this does not mean that the model can not be used all together. The model can be used as a heuristic for various purposes. Whether it can work as a helping guidance for further annotation of new parking data, or to be used in autonomous cars but not as true fact. It is only as a way to steer it into the right direction where parking spaces are most probable to be.

It was clear that CNNs do perform well on the problem at hand, given that results certainly indicate a basic understanding of the shape and outlook of parking spaces. However, the problem most likely lies in the problem definition and restriction. The more general the problem is, which is indeed the approach for this report, the worse results are obtained it seems. Thus, our solution shows that restricting problem space is key to success in the field of image classification of parking spaces.

5.4 Impact

A general image analysis solution for finding parking spaces does not yet exist, and granted the one presented in this report is not a final nor a complete one. The potential of such a solution however is substantial. The solution would not in any case be an unambiguous way of finding parking, as e.g. parking garages and unmarked parking lots are ignored. Fortunately, this is not how autonomous

cars or AI in general perform complex tasks. They perform them by aggregating different predictions and gathered data to make informed decisions. Thus, adding to the existing knowledge base is always beneficial. An autonomous car can with the predictions produced by the solution in this report somewhat reliably construct an abstract model of the parking lot it is standing in. Given a reliable data provider, it can furthermore potentially do so given only its GPS coordinates. In other words, it can do such autonomously, which is of course one of the main requirements of competent AI. Using this model of the parking lot, it can then make an informed decision on which route to take in searching for available spots. While en route, the car can also take the generated detections, confidences, and masks to aid in its assessment if the space exists or not, and if it is occupied or not. Coupled with other algorithms, predictions, and sensor input, e.g. adjacency of cars, detection of curbs, lines on the ground, signs etc. it should be able to reach an acceptable conclusion.

Academically, this report can be seen as a case study in state of the art R-CNNs, in particular Mask R-CNN. It moreover works as a reference point to anyone wishing to conduct similar studies in parking space detection or using CNNs in general.

5.5 Suggestions and Further Research

A lot of issues came to the forefront during the project, which were either too large in scope for this particular problem, or were realized too late in the process. This section will describe problems and possible solutions to be considered if one wishes to expand upon or redo the project.

- **Two Classes** - The problem described in Section 5.2.2 of parking spaces being generally feature sparse might be solved by introducing two different classes - one for occupied spaces and one for empty ones. This would mean the model can learn features specific for the two separate classes rather than for the two combined. An observed problematic result was that the model had trouble classifying empty spaces, particularly if they were by themselves. Occupied spaces are naturally more feature heavy given the presence of cars, while training for the empty spaces can safely ignore the cars and focus entirely on differentiating between empty spaces and similar looking patterns, as also described in Section 5.2.2.

- **Pairwise Detections** - As also discussed in Section 5.2.2, parking spaces tend to have few features, which can be problematic for the model to detect with proper confidence. One way of increasing the amount of features, as well as making the detections more unique from noise, such as driving cars on regular roads or parking-looking lines in crossings, is to make further use of the fact that parking spaces tend to be clustered together. Instead of a detection being solely one individual parking space, one could group them together in pairs. This way the detections will surely be more unique and feature rich, as the probability for two cars being that close to each other, or two unoccupied spaces for that matter, should be fairly low if it is not a

parking space. Another problem is of course that parking spaces that are completely by themselves, not having any neighbouring parkings, would be left out.

- **More Data Providers** - An inherent problem, as discussed in Section 5.2.1, is the specificity of each provider's images. A way to improve this would certainly be to increase the amount of different data providers, in order to increase generality of the dataset, and thus of the trained model. This way, the model performance would be theoretically less constrained by the type of input data, making the problem less constrained. This might also increase the list of features for parking spaces, further improving upon the suggestions discussed in the previous section. Possibly, one might also consider using several data providers for the same area, reducing the possibility for over-fitting on the mere look of the dataset and not its actual content. Exploring how this affects the model as compared to just using different annotations could also be of interest.

- **Exploring Generality-Accuracy Trade-off** - The generality-accuracy trade-off is defined for this discussion as the exchange of how general the model is as compared to how accurate it is. That is, defining the scope of how large a context the model should take into consideration. Obviously, using the same geographical area, the same data provider, the same size of all images etc, will increase the accuracy of that certain defined area, but will probably reduce the performance on other images/areas from other providers that the model has not seen. Thus the result is a more accurate model, which might give a false impression of performance, as it only performs so well on that area. Then again, this might be desirable even. If a constrained problem space can be determined from the start, generality is not a problem anymore. For example, if the input is a hedged in parking lot, i.e. a polygon, then global context is not needed and thus negative data can be ignored altogether.

5.5.1 Post-processing

Post-processing, as in applying various algorithms to improve the end result of the model, could certainly be implemented. This was indeed experimented with during the project, but results were not conclusive. A fully polished solution would probably utilize a couple of post-processing algorithms, to further increase accuracy. The problem with this approach however, is that it forces human behaviours into the model, on occasion forcing out its machine learning behaviour in favour of the developer's own conceptions of the problem, without directly influencing the model itself. Nonetheless, using such post-processing could increase performance, albeit making performance slightly more difficult to determine. The exact heuristics and implementations are purposefully left vague, but here are some potential examples of such post-processing algorithms:

- Measuring closest distances for each detection to other detections. This might determine if the detection is in or close to a parking space cluster, as

most parking spaces are. Correct spaces are seldom all by themselves. If a parking is by itself, remove it as seen in Figure 5.1.

- If a detection has a neighbouring detection with a low confidence score, remove both detections as seen in Figure 5.2. It is unlikely for a true parking space to have a neighbouring detection with low confidence.
- Check the size and/or shape of the generated masks. Strange shapes and sizes could be indications of incorrectly labeled detections.
- Determining orientation/direction of parking spaces. Comparing these to each other could determine mislabeled outliers in the detection. The direction vectors can be calculated from the parking spaces using either singular value decomposition (SVD), or least squares regression.
- Automatically fill in missing spaces in parking space clusters. Having gaps in a cluster is uncommon and these could simply be filled if they are undetected, as seen in Figure 5.3. An example of such an algorithm is candidate voting. For each detection, consider four possible candidates surrounding the parking space and add them to a list. If any given candidate has at least one or more duplicate overlapping candidates, add it as a real parking space. Do this recursively until no more parking spaces can be added. This would in theory fill in the gaps of a parking cluster.

Figure 5.1: Post-processing idea of removing parking spaces that have no neighbours.

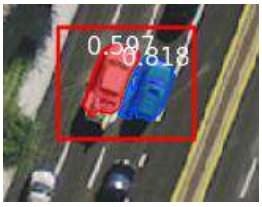

Figure 5.2: Post-processing idea of removing parking spaces that have a neighbouring detection with low confidence score.

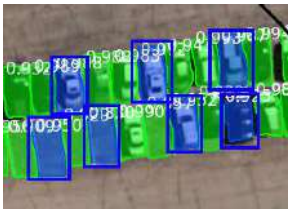

Figure 5.3: Post-processing idea of automatically filling in missing parking spaces in clusters shown in blue.

References

[1] B. Fusińska. (2016-2019) Tensorflow: Mnist for beginners. [Accessed Jan. 22, 2018]. [Online]. Available: https://www.katacoda.com/basiafusinska/courses/tensorflow-getting-started/tensorflow-mnist-beginner

[2] C. McDonald. Machine learning fundamentals (ii): Neural networks. [Accessed Dec. 17, 2018]. [Online]. Available: https://towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef

[3] R. Campbell. (2017) Demystifying deep neural nets. [Accessed Jan. 9, 2019]. [Online]. Available: https://medium.com/@RosieCampbell/demystifying-deep-neural-nets-efb726eae941

[4] S. Jain. (2018) An overview of regularization techniques in deep learning (with python code). [Accessed Jan. 9, 2019]. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

[5] K. Rosaen. (2016) Scikit-learn pipeline gotchas, k-fold cross-validation, hyperparameter tuning and improving my score on kaggle's forest cover type competition. [Accessed Jan. 9, 2019]. [Online]. Available: http://karlrosaen.com/ml/learning-log/2016-06-20/

[6] J. Hui. Real-time object detection with yolo, yolov2 and now yolov3. [Accessed Jan. 22, 2019]. [Online]. Available: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[7] Mask r-cnn for object detection and segmentation. [Online]. Available: https://github.com/matterport/Mask_RCNN

[8] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[9] G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Meghini, and C. Vairo, "Deep learning for decentralized parking lot occupancy detection." *Expert Systems With Applications*, vol. 72, pp. 327 – 334, 2017. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edselp&AN=S095741741630598X&site=eds-live&scope=site

[10] "Car parking occupancy detection using smart camera networks and deep learning." *2016 IEEE Symposium on Computers and Communication (ISCC), Computers and Communication (ISCC), 2016 IEEE Symposium on*, p. 1212, 2016. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edseee&AN=edseee.7543901&site=eds-live&scope=site

[11] "Parking-stall vacancy indicator system, based on deep convolutional neural networks." *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*, p. 655, 2016. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edseee&AN=edseee.7845408&site=eds-live&scope=site

[12] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982. [Online]. Available: http://www.pnas.org/content/79/8/2554

[13] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, ser. IJCAI'11. AAAI Press, 2011, pp. 1237–1242. [Online]. Available: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-210

[14] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex." *The Journal Of Physiology*, vol. 195, no. 1, pp. 215 – 243, 1968. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=cmedm&AN=4966457&site=eds-live&scope=site

[15] I. King, W. Jun, C. Laiwan, W. DeLiang, Z. Hong, P. Li, and L. Li, *A Morphological Neural Network Approach for Vehicle Detection from High Resolution Satellite Imagery.*, 2006, p. 99. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edb&AN=32944210&site=eds-live&scope=site

[16] A. L. and S. M., "Vehicle detection and classification from high resolution satellite images." *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol II-1, Pp 1-8 (2014)*, p. 1, 2014. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edsdoj&AN=edsdoj.1e191c514d94b3ca60db81bc8b4d7a6&site=eds-live&scope=site

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks." *Communications of the ACM*, vol. 60, no. 6, pp. 84 – 90, 2017. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=123446102&site=eds-live&scope=site

[18] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks." 2013. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.1311.2901&site=eds-live&scope=site

[19] C. . . . Szegedy, Y. . . . Jia, P. . . . Sermanet, D. . . . Anguelov, D. . . . Erhan, V. . . . Vanhoucke, W. . . . Liu, S. . . . Reed, and A. . . . Rabinovich, "Going deeper with convolutions." in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June-2015, no. IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, (1)Google Inc., 2015, pp. 1–9. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-84937522268&site=eds-live&scope=site

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition." 2014. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.1409.1556&site=eds-live&scope=site

[21] "Deep residual learning for image recognition." *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, p. 770, 2016. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.7780459&site=eds-live&scope=site

[22] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *arXiv*, 2018.

[23] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," *arXiv*, 2018.

[24] Open street map. [Online]. Available: https://www.openstreetmap.org/

[25] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[26] Sentinel missions. [Online]. Available: https://sentinel.esa.int/web/sentinel/missions

[27] "Land cover prediction from satellite imagery using machine learning techniques." *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Inventive Communication and Computational Technologies (ICICCT), 2018 Second International Conference on*, p. 1403, 2018. [Online]. Available: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.8473241&site=eds-live&scope=site

[28] Earh explorer. [Online]. Available: https://earthexplorer.usgs.gov/

[29] Overpass api. [Online]. Available: https://wiki.openstreetmap.org/wiki/Overpass_API

[30] Amazon s3. [Online]. Available: https://aws.amazon.com/s3/

[31] Heroku cloud application platform. [Online]. Available: https://www.heroku.com/

[32] Yolo-v3 and yolo-v2 for windows and linux. [Online]. Available: https://github.com/AlexeyAB/darknet

[33] Darknet. [Online]. Available: https://github.com/pjreddie/darknet

[34] Keras retinanet. [Online]. Available: https://github.com/fizyr/keras-retinanet

Appendix A
Tensorboard Graphs

Figure A.1: Tensorboard graph - IMAGES_PER_GPU

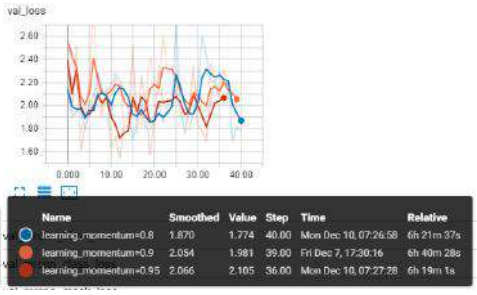

Figure A.2: Tensorboard graph - LEARNING_MOMENTUM

Figure A.3: Tensorboard graph - WEIGHT_DECAY

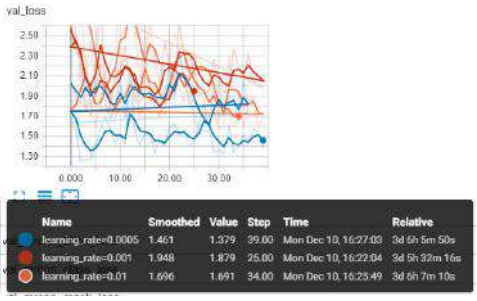

Figure A.4: Tensorboard graph - LEARNING_RATE

Figure A.5: Tensorboard graph - RPN_ANCHOR_SCALES

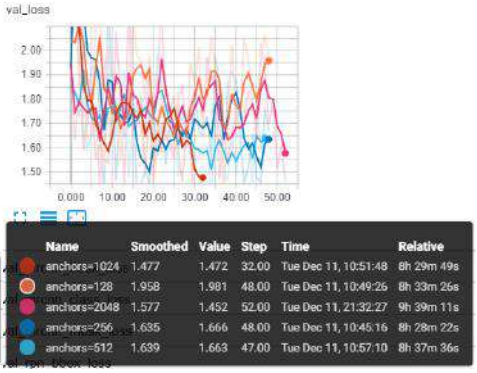

Figure A.6: Tensorboard graph - RPN_TRAIN_ANCHORS_PER_IMAGE

Figure A.7: Tensorboard graph - LAYERS

Figure A.8: Tensorboard graph - DETECTION_MIN_CONFIDENCE

Figure A.9: Tensorboard graph - RPN_NSM_THRESHOLD

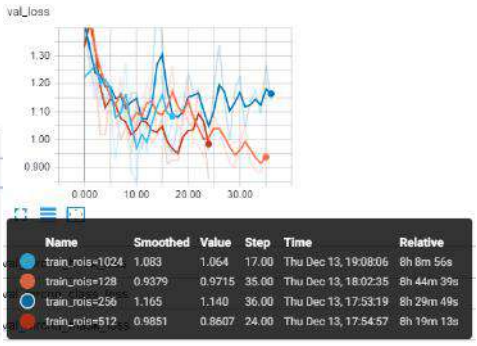

Figure A.10: Tensorboard graph - TRAIN_ROIS_PER_IMAGE

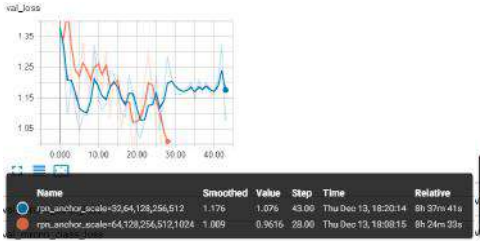

Figure A.11: Tensorboard graph - RPN_ANCHOR_SCALES_2

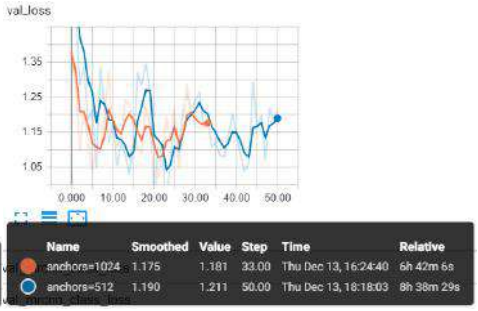

Figure A.12: Tensorboard graph - RPN_TRAIN_ANCHORS_PER_IMAGE_2

Figure A.13: Tensorboard graph - TRAIN_ROIS_PER_IMAGE_2

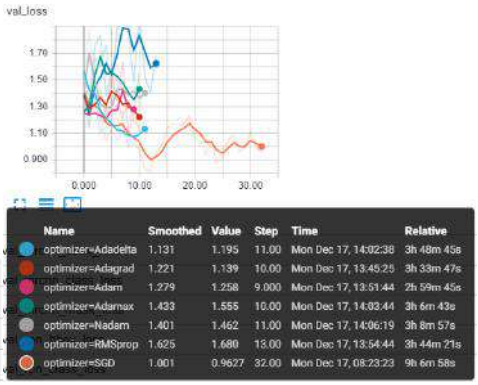

Figure A.14: Tensorboard graph - OPTIMIZER

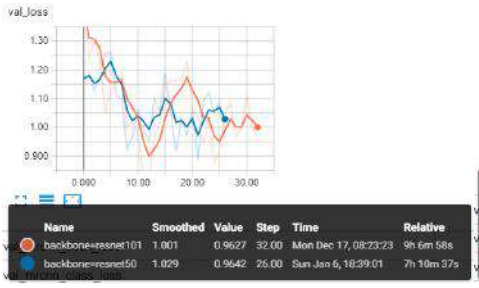

Figure A.15: Tensorboard graph - BACKBONE

Figure A.16: Tensorboard graph - ROIS_POSITIVE_RATIO

Figure A.17: Tensorboard graph - DATASETS

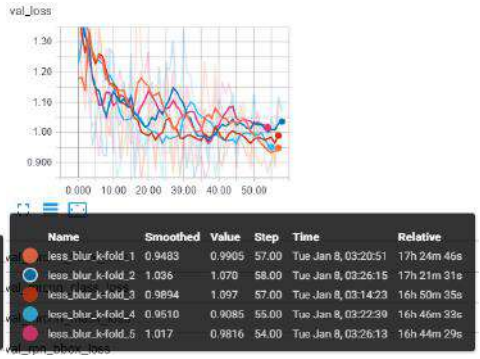

Figure A.18: Tensorboard graph - LESS_BLUR_K-FOLD

Figure A.19: Tensorboard graph - NO_PADDING_K-FOLD

Master's Theses in Mathematical Sciences 2019:E6
ISSN 1404-6342

LUTFMA-3376-2019

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/