UPPSALA
UNIVERSITET

UPTEC STS 22009
Examensarbete 30 hp
Juni 2022

Benchmarking Object Detection Algorithms for Optical Character Recognition of Odometer Mileage

Eric Andersson, Mandus Hjelm

Civilingenjörsprogrammet System i Teknik och Samhälle

UPPSALA
UNIVERSITET

Benchmarking Object Detection Algorithms for Optical Character Recognition of Odometer Mileage

Eric Andersson, Mandus Hjelm

Abstract

Machine learning algorithms have had breakthroughs in many areas in the last decades. The hardest task, to solve with machine learning, was solving tasks that humans solve intuitively, e.g. understanding natural language or recognizing specific objects in images. To overcome these problems is to allow the computer to learn from experience, instead of implementing a pre-written program to solve the problem at hand - that is how Neural Networks came to be. Neural Network is widely used in image analysis, and object detection algorithms have evolved considerably in the last years. Two of these algorithms are Faster Region-based Convolutional Neural Networks(Faster R-CNN) and You Only Look Once(YOLO). The purpose of this thesis is to evaluate and benchmark state-of-the-art object detection methods and then analyze their performance based on reading information from images. The information that we aim to extract is digital and analog digits from the odometer of a car, this will be done through object recognition and region-based image analysis. Our models will be compared to the open-source Optical Character Recognition(OCR) model Tesseract, which is in production by the Stockholm-based company Greater Than. In this project we will take a more modern approach and focus on two object detection models, Faster R-CNN and YOLO. When training these models, we will use transfer learning. This means that we will use models that are pre-trained, in our case on a dataset called ImageNet, specifically for object detection. We will then use the TRODO dataset to train these models further, this dataset consists of 2 389 images of car odometers. The models are then evaluated through the measures of mean average precision(mAP), prediction accuracy, and Levenshtein Distance. Our findings are that the object detection models are out-performing Tesseract for all measurements. The highest mAP and accuracy is attained by Faster R-CNN while the best results, regarding Levenshtein distance, are achieved by a YOLO model. The final result is clear, both of our approaches have more diversity and are far better than Tesseract, for solving this specific problem.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Nicklas Lindqvist Ämnesgranskare: Ewert Bengtsson

Examinator: Elísabet Andrésdóttir

Populärvetenskaplig sammanfattning

Dataanalys är något som människor sysselsatt sig med sedan matematiken var uppfunnen, allt från att räkna på fjolårets skörd till att Henry Ford effektiviserade sitt så kallade rullande band. På senare tid har dock mängden data som analyseras vuxit avsevärt. Det har i sin tur lett till svårigheter att manuellt utföra dessa analyser. Detta leder till att man i större del vill automatisera dessa processer, vilket är det fundamentala syftet för maskininlärning. Då låter man en algoritm lära sig från inmatningsdatan och generera ett resultat som är överskådligt och lätt tolkat av människor [18]. Ett stort område inom maskininlärning är bildanalys, då detta har visat sig vara en framgångsrik uppgift för maskininlärningsmodeller. Dessa modeller kan då lära sig att klassificera olika objekt i en bild, vilket vi har använt oss av i denna uppsats.

Syftet med denna uppsats är att designa en maskininlärningsmodell som ska kunna läsa ut hur långt en bil har kört baserat på bilder på instrumentbrädan inuti en bil. Modellen ska alltså lära sig detta utifrån bilder tagna med mobilkamera på instrumentpanelen av en bil. Detta är en funktion som AI-företaget, Greater Than implementerat i en av sina applikationer men i dagsläget så är avläsnings-algoritmen inte specialiserad mot ändamålet och på grund av detta så presterar den för dåligt och för opålitligt vilket leder till att användaren behöver skriva in miltalet själv. Att denna processen blir automatiserad har två stora fördelar, det tar bort faktorn för mänskliga fel, och att användaren inte kan manipulera miltalet, vilket gör att processen blir snabbare med färre fel.

Det praktiska arbetet har gått ut på att jämföra Greater Than´s nuvarande *optical character recognition*-algoritm med två state-of-the-art object detection-modeller för att se vilken metod som är bäst passande för detta problem.

Resultatet visar att object detection-modeller är mycket bättre på identifiera odometern och dess mätarställning och får en högre precision. De är även bättre på att filtrera bort mycket av de överflödiga siffrorna som förekommer i bilderna, till exempel hastighetsmätarens siffror. Denna studiens tillkortakommande ligger i bildunderlaget, att modellerna som tränas skulle behöva fler bilder och att annoteringarna var sämre än förväntat. Detta gör att resultatet inte är tillräckligt för att föra in i produktion just nu, men med fler bilder och bättre annoteringar tror vi att object detection modeller är en möjlig väg för att kunna implementera den funktionen i produktion.

Acknowledgments

This master thesis project has been carried out at Uppsala University in collaboration with Greater Than, a Stockholm-based machine learning company that is specialized in visualizing and analyzing data from cars. We would especially like to thank our subject reviewer, Ewert Bengtsson, for his guidance and the fruitful discussions we have had in our meetings. We would also like to thank Niklas Lindqvist, our supervisor, and Liselott Johansson, CEO of Greater Than, for giving us the opportunity to write our master thesis in association with the company. And lastly, we would like to give a big thanks to all our friends and teachers for the years at Uppsala University, without them, we would not have reached this far.

Uppsala June 2022
Eric Andersson, Mandus Hjelm

Contents

Acronyms

AI Artificial Intelligence.

BN Batch Normalization.

DL Deep Learning.

DNN Deep Neural Network.

FN False Negative.

FP False Positive.

IoU Intersection over Union.

mAP Mean Average Precision.

NN Neural Network.

TN True Negative.

TP True Positive.

VM Virtual Machine.

Chapter 1

Introduction

1.1 Background

Neural Network (NN) might seem like a new area but it dates back to the 1940s [10]. One definition of NN, is a network that consists of connected logic units performing calculations by mimicking the way the biological neuron work[3]. The reasons for developing NNs are many, for example, automating repetitive tasks, image analysis, making diagnoses in medicine, or natural language processing. One of the greatest challenges when developing Artificial Intelligence (AI) turned out to be solving tasks that humans solve intuitively, e.g. recognizing spoken words or faces in images. One solution to these problems is to let the computer learn from experience, instead of having a preprogrammed computer solve the problem at hand. Neural networks have the possibility to learn very complex concepts built on many simple ones. This hierarchy of simple concepts which are built on each other is very deep. And that's what we call Deep Learning (DL) today[9].

A field where NN and DL have had wide success is in computer vision, analyzing images and videos based on the pixels. Examples of applications where computer vision is used are autonomous driving or face recognition [18]. Computer vision is used by the company *Greater Than*, where their objective is through a customer-taken image over a car dashboard, recognize how far the car has traveled by reading the odometer automatically. Implementing and making this process automated, has mainly two perks. It removes the human factor to making mistakes entering the mileage manually, and eliminates the possibility to enter fraudulent inputs. Greater Than's current implementation, which we will benchmark towards, is an OCR engine called Tesseract.

Tesseract OCR engine is not specialized for reading these types of images, where digits appear on a tiny part of the image. It is built to recognize characters in scanned documents, making the document searchable. In this project, our approach will consist of the object detection algorithms Faster Region-based Convolutional Neural Networks(Faster R-CNN) and You Only Look Once(YOLO). These are the state-of-art algorithms within the computer vision area, object detection.

1.2 Purpose

The purpose of this thesis is to design and evaluate two state-of-the-art deep neural network models, compare them to an optical character recognition engine and analyze their performance based on extracting information through image analysis. The information that we aim to derive is the digits of the odometer in a car. The models in question are the open-source optical character recognition model Tesseract and two object-based models called YOLO and Faster R-CNN. The evaluation metrics used will be Levenshtein distance and mean average precision.

1.3 Structure of Thesis

The outline of this thesis will consist of six chapters. The chapters are *Introduction, Theory, Method, Result, Discussion and Conclusion*. The first chapter will give the reader an introduction to the subject and a good sense of what we aim to examine, and also the demarcation of what is beyond the scope of this thesis. The theory section will present the relevant groundwork which is needed to understand the execution and result of this project. The next chapter, *Method*, will provide a complete presentation of the data, how the practical part is conducted, and the technologies used. In the upcoming chapter, we will present the final result we have obtained after performing this study. In chapter *Discussion*, we will examine the results and problems we encountered. Finally, in the last chapter, conclusions from the experiments and what future work could be done are presented.

1.4 Organization of the Thesis

This master thesis has been conducted in collaboration with Greater Than, an AI data analytics company based in Stockholm. Greater Than is developing AI applications for the insurance industry with the purpose to predict driver-related car accidents and minimizing CO2 emissions. The purpose of this thesis came about when discussing ongoing internal projects with the group from Greater Than, which became the supervisors, they proposed that we should try state-of-the-art approaches for optical character recognition and compare it to their current algorithm, Tesseract.

1.5 Related work

Related to this problem is automatic license plate recognition which has a lot of research and implementations that have been done which is similar to reading an odometer but our problem has an increased complexity with the odometer can have several different appearances in contrast to license plates that have a very similar appearance. Very similar to our problem is the work done by Acharya and Fung[1] that makes a comparison between a single-shot multi-box detector (SSD) and Faster R-CNN when classifying odometer readings and found that Faster R-CNN outperformed the SSD [1]. Because of the promising result of Faster R-CNN, that model was chosen for our experiment.

Faster R-CNN is an algorithm that has many parameters, see 3.3.2. Hence, a model with

another approach and fewer parameters would be interesting to compare to the Faster R-CNN algorithm. Therefore, YOLO is a model developed to be fast and have fewer parameters was chosen. YOLO has as Faster R-CNN great scientific support of being a precise and able to use for real time detection [5].

Chapter 2

Theory

In this chapter, the fundamental theoretical key concepts of machine learning are explained. Furthermore, the models and network architectures, which are used in this study, are described. The aim of this chapter is to provide a fundamental understanding when present the result and evaluating the outcome of this thesis.

2.1 Neural Networks

Neural networks approach a problem with a large number of data points, known as training data, and then infer rules for recognizing patterns based on the data. This data can be labeled where the problem is called supervised learning, or if the data is unlabeled this is called unsupervised learning and the network need to find information that is hidden or unknown.

To understand how the network learns these patterns we first need to go deeper and get familiar with the building blocks of NNs, the artificial neuron. As mentioned earlier, the NN is inspired by the biological neuron, hence the name. There are different kinds of neurons for different networks, but they are basically algorithms with several inputs a and one output. One simple example of an artificial neuron is the perceptron. The perceptron takes all the inputs, multiply them with weights w and then sees if the sum is larger than the perceptron's bias b, if it is, then the output is 1, otherwise, 0, see equation 2.1. The weights come from the previous layer of the network. The bias decides how inclined the neuron is to fire a signal, if it is a very low number it is less likely for the neuron to signal. One limitation with the perceptron is that the output is either 1 or 0[24].

$$output = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b \text{ is} \leq 0, \\ 1 & \text{if } \sum_j w_j x_j + b \text{ is} > 0. \end{cases} \tag{2.1}$$

It is these two parameters that make learning possible. The output is depending on the weight and the bias, and that means that we can adjust these parameters to give us an output closer to what we want. The parameters are evaluated based on the error of the output, if the objective is to classify images of cars, the parameters will be adjusted to make as many correct classifications as possible. This is repeated over and over to produce better and better output, i.e. the network is learning how a car looks like[18].

2.1.1 Convolutional Neural Networks

Convolutional Neural Network(CNN) is a neural network that applies a mathematical operation called convolution. Convolution is performed on each pixel of the image using its surrounding pixels. How this works will be described in the coming section.

Traditional neural networks use matrix multiplication, which ultimately means that every output unit interacts with every single input unit, resulting in a fast increasing complexity. This makes it very computational demanding for big data input sizes. However, CNN´s uses something called sparse interactions. It is achieved by using a kernel that is smaller than the input and applying it all over the image. Even though images can have thousands of pixels in total, the kernel goes through all of them but in stages. This way, the kernel can detect small, meaningful features of an object, e.g. edges or shapes. At the same time the computer needs to save a lot fewer parameters in comparison to fully connected networks, this results both in less memory requirements and improves the efficiency [18]. Another important feature of CNN is the idea of Parameter sharing. It means the network is using the same parameters for more than one input. The parameters that are shared are the weights in the kernel and it is shared since it is applied on every data point, in other words, all the neurons share the same weights which are used on every position of the input. While the fully connected network uses each weight parameter exactly once. The parameter sharing means that the model does not need to learn to detect similar features many times, the network maintains the same feature detector for all input sections [18].

Convolution

The calculations are done by taking a matrix of weights and multiplying them by each pixel and its neighbors. The values are then summed up and placed in the original pixel location. The matrix of weights is called a kernel, these are used for blurring, enhancing, sharpening, or smoothing the image. Once the convolution operation has been executed on every pixel, it has produced a new image with new features based on the chosen kernel [18].

Generally, the convolutional operation is done on two functions with real-valued arguments. Suppose we have a sensor that provides a single output of a position x(t) at the time t. Two things need to be taken into account, potential noise and at what time the measurement was read, since more recent measurements are more relevant. To handle potential noise, coming from x(t), it is sufficient to average many measurements. It is done by adding a weight function w(a) where a is the age of the measurement. This results in the equation 2.2 where s(t) is the position [18].

$$s(t) = \int x(a)w(t-a)da. \tag{2.2}$$

In equation 2.2, x(a) is the input, while w is the kernel and the output of this is sometimes called feature map, which in this case would be the estimated position. When a computer is doing these calculations, the input cannot be continuous since that would result in infinite data points, therefore the data must be discretized. With a discretized equation 2.2 can be defined as follows:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{2.3}$$

However, usually, in machine learning applications the input is a multidimensional array of data. For example, images require extra dimensions to describe the colors or the greyscale of each pixel. These multidimensional arrays of data are called Tensors, a 1D Tensor is a single-dimensional vector while a 2D Tensor is a matrix, and a 3D Tensor is an array of matrixes, it can be described as a cube. Furthermore, a 4D Tensor is a vector of multiple 3D Tensor cubes and a 5D is a matrix of multiple 3D cubes. When working with images the input and output are 3D tensors [18].

Pooling

The anatomy of a CNN consists of multiple layers; including a convolutional layer, non-linearity layer, pooling layer, and fully-connected layer. The pooling and non-linearity do not have any parameters. The fundamental idea with pooling layers is to down-sample the data to decrease the complexity for the following layers by decreasing the number of outputs. To understand the pooling layer, we need to understand the following terms *stride, padding, and pooling function.* The pooling function determines which value it takes from each area the filter is applied. There are different pooling functions that is applicable in different situations. One common function is max pooling where it takes the maximum value. There is also min pooling and average pooling. The stride is how many pixels the kernel is moving for each node. This has a high influence on the number of output parameters. If we have an input size with the dimension 4 x 4 and apply a 2 x 2 filter with stride 2 and use max pooling, that will result in a 2 x 2 output with the highest values from each area, see figure 2.1. The output size O can be described with equation 2.4, where N is the input size and F is the size of the filter and S is the stride [2].

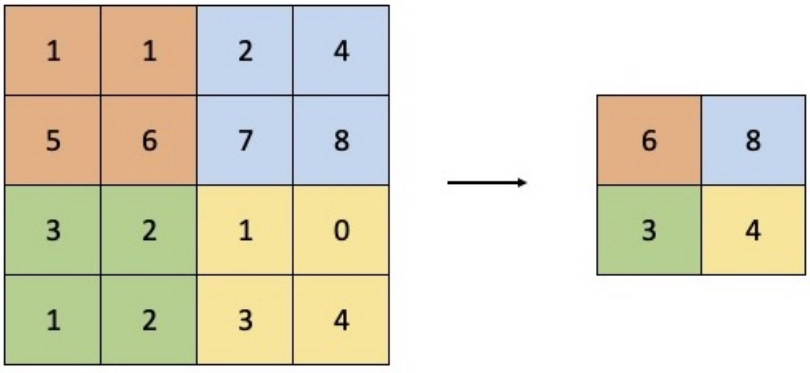


Figure 2.1: Visualization of the pooling operation [2].

$$O = 1 + \frac{N - F}{S} \tag{2.4}$$

A disadvantage of using CNN is the loss of information on the border of the image. One efficient measure to solve this is to add zero padding, which means that we add a border

of zeros around the image. Without any padding, the output shrinks by one pixel less than the kernel width at each layer, which limits the number layers that can be added to the network. Hence, padding makes it possible to build deep networks. When applying padding, the size of output O can then be described with equation 2.5, where P is the number of layers with zero padding [2].

$$O = 1 + \frac{N + 2P - F}{S} \tag{2.5}$$

2.1.2 Optical character recognition (OCR)

Optical character recognition (OCR) is a technology that enables characters in an image to be classified and extracted into data. With OCR it is possible to recognize both handwritten and printed texts but the results are very fragile to poor input data [20]. OCRs are designed to process images that consist almost completely of text. If the input data contains any inconsistencies like dimmed light or any blurry numbers, it is very hard for most OCR applications to recognize the correct text or digits. OCR enables the conversion of different types of documents like scanned paper documents, PDF files or images to editable and searchable data [20].

2.2 Learning, error and fitting

The goal with a NN is for the model to learn from available classified data and make predictions about unseen real-life data. For example, pictures of a cat being classified as a cat. The model learns by trying to minimize its error rate or maximize the model accuracy, for more about model measures see section 2.4.2. To be able to evaluate how the model performs on unseen data the dataset is divided into a train part and a test part. The train dataset is the data that is used to learn the different parameters of the model. During the training of the model, the parameters are tuned. Then the model is tested by trying to predict/classify the features in the unbiased test dataset [9]. The error from how the model performs during training is called training error, similar is the error on the test called test error.

Overfitting and underfitting

Model training of NN is a balancing act between underfitting and overfitting. Underfitting is when the model is not complex enough and having trouble to describe the patterns in the training data. Overfitting is when the model is too complex and too well adopted to the training data. The indicators for how these are behaving are the train error and the test error. The train error will decrease when the complexity increases but the test error is more of a curve where the global minimum is not where the complexity is the highest. And the optimal solution is in that global minimum, see figure A.1 in appendix for visual representation of figure with local and global minimum [9].

2.2.1 Loss function

The loss is calculated by the loss function and the loss can be defined as the distance from the model-prediction of the observed data to the real world data[18, p 40].The loss is used

to calculate the gradients that is used to update the weights which implies that the loss function is important for the networks learning. Mostly used for multi-class classification is cross entropy but Janocha and Czarnecki argues that depending on application the loss function shall be adjusted to fit the area of use [15].

Mean Absolute Error (MAE)

Mean absolute error or **L1 score** is a simple loss function where the distance between target value and predicted value are added together and then divided by sample size [18].

Cross entropy

Cross entropy loss is a loss function used for classification models where the distance between the predicted output possibilities from the output layer, for example, a softmax layer, are compared to the truth values. The cross entropy loss is defined in function 2.6 and where P is the known probability of each class and Q is the model predicted probabilities for each class. As can be seen in the cross entropy loss function is that the negative log likelihood for each label is taken, which means that the loss will be high when predictions differs a lot and small when the predictions are similar [7].

$$-\sum^{x\in X} P(x)\log(Q(x)) \tag{2.6}$$

2.2.2 Gradient Descent

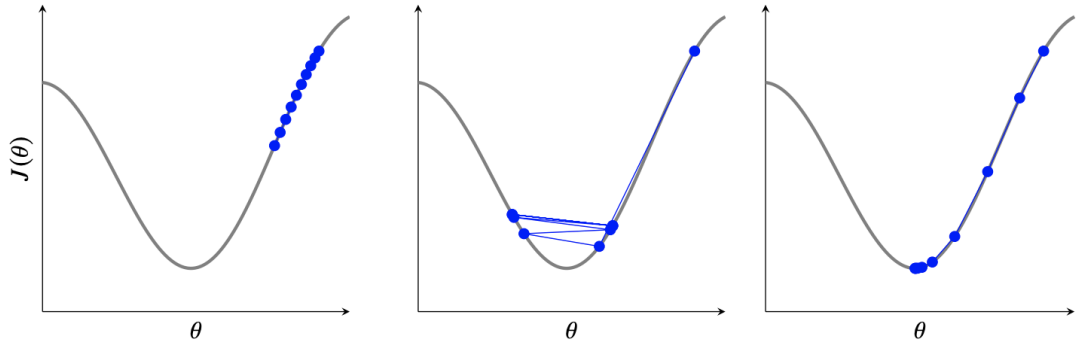
Gradient descent is the processes of finding the optimal solution i.e. global minimum (θ) as can be seen in figure 2.2. Learning rate (γ) is the step size the model is approaching the minimum. As can be seen in the figure 2.2 it is crucial to have a learning rate that is not to high or to low for the model to converge towards (θ) otherwise the whole model can break [18]. When the problem is what can be seen in figure 2.2 it is easy to see the global minimum but in higher dimensions and with a problem with higher complexity there could be several local minimum points where the model could get stuck in, for example see figure A.1 in appendix.

Stochastic Gradient descent

In images there are a lot of data points and therefore a lot of calculations being done when calculating the gradient descent. One can assume that two pixels that are close have similar data and therefore could limit the data points that make up the calculations a thereby limit the calculation time. By introducing a stochastic element by randomly subsampling the dataset.

2.2.3 Backpropagation

Backpropagation is one of the algorithms needed for a neural network to be able to learn and improve from the training data. This is done by adjusting and calculating the gradient to the cost function i.e. average loss function, that later is used by, for example, stochastic gradient descent. Backpropagation is recursively using the chain rule to obtain the gradient



(a) Low learning rate $\gamma = 0.05$ **(b)** High learning rate $\gamma = 1.2$ **(c)** Good learning rate $\gamma = 0.3$

Figure 2.2: Example over learning rate for gradient descent

to the cost function with respect to all parameters, in other words the partial derivative of the cost function C with respect to the weights w and bias b [9]. Since C is dependable on the output from previous layers in the network, see section 2.1, to be able to tell how changes in the last last layers affect the other parameters are dependable on the equation 2.7.

$$\frac{\partial C}{\partial w} = \frac{\partial z}{\partial w} * \frac{\partial a}{\partial z} * \frac{\partial C}{\partial a} \tag{2.7}$$

Activation functions

As previously mentioned, a neural network can be described as a linear regression model see equation 2.8. But when we want to represent a non-linear relationship, for example, the curve in figure 2.2, a non linear function needs to be applied. This non-linear function is called an activation function, see figure 2.9.

$$\hat{y} = (w_1 * x_1) + (w_2 * x_2) + (w_3 * x_3) + \ldots (w_p * x_p) + b \tag{2.8}$$

$$\hat{y} = a * ((w_1 * x_1) + (w_2 * x_2) + (w_3 * x_3) + \ldots (w_p * x_p) + b) \tag{2.9}$$

w = weights, x = input variables, b= bias a = activation function

There are several different activation functions that are designed for different problems and applications. One of these is the Sigmoid function, which is defined by equation 2.10, where the output value is adjusted to be between 0 and 1, see figure 2.3 [24]. Two other activation functions that are of importance are the Softmax and Rectified Linear Unit(ReLU). The output of ReLU is zero when the input is negative, and linear if the value is bigger than zero, see figure 2.3. ReLU is usually used between layers in neural networks, because of its simplicity and speed. Unlike ReLu, the activation function Softmax takes into account all the outputs which are useful in multi-class classification problems, see equation 2.11. The output of Softmax is normalized between zero and one which means that the output can be seen as probabilities whereas higher output means a higher probability that the input shall be classified as that output [18]. For multi-class classification, many argue for the logarithmic Softmax function for its improved numerical performance and perplexity [6].

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.10}$$

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{2.11}$$

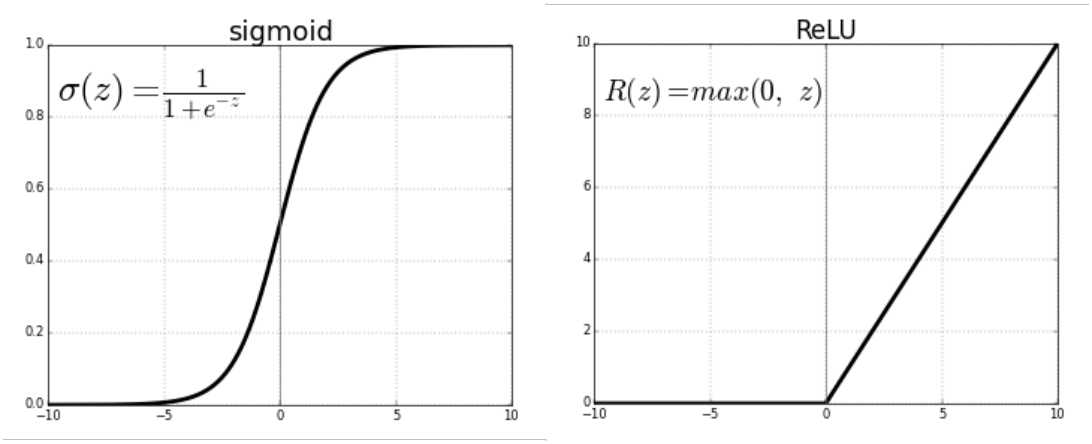


Figure 2.3: Equation and visual representation over Sigmoid and ReLU activation function.

2.2.4 Transfer learning and fine-tuning pretrained models

As mentioned, the amount of data for a deep neural network is crucial for the precision of the model. If there is a lack of data to train a new model, a model trained on a similar dataset can be used. For example, an object detection model trained on pictures of cats can possibly be used to detect dogs. This approach is called transfer learning. The pretrained model can also be adjusted to fit the new purpose, where some layers are "frozen" and the output layers are trained on the problem-specific data. This application is part of transfer learning and is called fine-tuning. The idea is that the earlier layers in the network are extracting features that are universal, for example, in an image, lines, and corners that in the later layers with the problem-specific information are put together to make a classification. This makes the model converge faster and can speed up the training and makes the model less prone to get stuck in sub-optimal solutions [18].

2.3 Batch Normalization

During the training of a Deep Neural Network (DNN), the distribution of each layer's input changes because of the parameters in the previous layers change for each iteration. This shift in distribution is called *internal covariate shift*, and to deal with this problem it is possible to normalize the inputs between the layers, and this is called Batch Normalization (BN). Internal covariate shift will result in a slow training for the model, that is why we need to normalize between layers. With BN, the distribution remains stationary, which makes it possible for the model to have a high learning rate during training. BN makes the model less sensitive when initialize weights while it also acts as a regularizer, and can in some cases decrease the need for *Dropout* [18]. What this means in application is to use big batch size when training since it is beneficial for BN.

2.4 Model evaluation

2.4.1 Confusion matrix

When performing model evaluation a confusion matrix is a good tool to get an objective of how the model predicts the different classes. If the problem is a binary classification problem, for example, a picture of a one or a zero, assume that positive is equal to one and negative means zero. As can be seen in table 2.1 there are four different alternatives. True Positive (TP) means that the network predicts the image of a one as a one. True Negative (TN) means the same but with the image of a zero gets predicted as a zero. False Negative (FN) means that an image of a zero gets classified as a one. FN are the opposite of False Positive (FP) but an image of a one gets classified as a zero.

Table 2.1: Confusion matrix over binary outcome.

		Prediction outcome	
		p	**n**
Actual value	**p′**	True Positive	False Negative
	n′	False Positive	True Negative

2.4.2 Model measures

To evaluate multi-class classification models there are many different approaches with different pros and cons, and depending of the goal with the model, the confusion matrix differs from the table 2.1 by having more options/fields since there are more classes. The approach is similar but an average of the classes are used, see equation 2.13 for example. An easy to understand and compute is accuracy which shows the percentage of correct classified predictions, see 2.12. Two other measurements are precision and recall. A high number, close to one indicates on recall that the positive data points are correctly predicted as positive, and on precision that a lot of the positive data points were classified as positive. Similar but a slight difference as can be seen in equation 2.14 and 2.15 A low score, close to zero indicates for recall that the model classify many FN, and for precision that there are too many FP [18, p. 75]. The advantages of the measurement's accuracy, precision, and recall are being simple, easy to understand, and widely used. The disadvantages are that they do not give much information and are less favorable towards the less dominant classes [11].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.12}$$

$$Averaged\ Accuracy = \frac{1}{n}\sum_{i=1}^{n} \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i} \tag{2.13}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.14}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.15}$$

$$\tag{2.16}$$

Note: For explanation of abbreviations see section 2.4.1

2.4.3 Intersection over Union, IoU

For object detection, the goal is to get the highest similarity between the prediction and ground truth/target. For example, detect in what pixels the odometer is in an image, see 3.1. The most basic and commonly used measurement is the Intersection over Union (IoU) which is built on the Jaccard index which is a coefficient for likeness between data sets. As can be seen in the picture 2.4, IoU = $TP/(TP + FP + FN)$ which means that recall and precision can be used in combined with the IoU. To be able to say if a boundary box shall be classified as TP or FN, usually, different thresholds of IoU are being used [26].

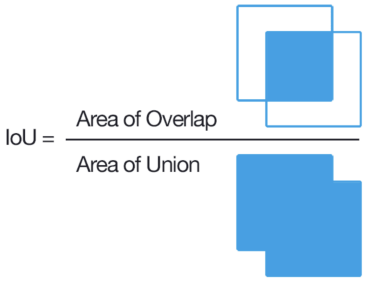


Figure 2.4: Illustration of Intersection over Union [30]

2.4.4 PR-curve and Mean average precision

With multi-class classifications on object detection, a useful measurement is the Mean Average Precision (mAP). The average precision, AP, is the area under the precision recall-curve, PR-curve. The PR-curve is the precision and recall plotted for different thresholds with recall on the horizontal axis and precision on the vertical axis. This means that mAP is the mean over the different classes values for their area under the PR-curve. Since both precision and recall are used, mAP works well with class imbalance [7].

Since mAP is dependable on IoU, different thresholds can be applied, for example, 0.5 and 0.5:0.95. When the IoU threshold is 0.5, every correct prediction with a IoU greater than 0.5 counts as TP. A 0.5:0.95 threshold means that the mAP is calculated for several different thresholds, with a 0.05 step between them, the thresholds and is between 0.5 and 0.95, then taking the mean of that result [26].

2.4.5 Levenshtein distance

Levenshtein distance is a measurement used to compare two text strings and their distance to each other, with a lower distance implicating similarities between the text strings. This distance is built on the three actions to alter a word, substitutions, insertions, or deletions. For example shark and mark get a distance of two because *s* gets deleted and *h* gets substituted to an *m*. Therefore, the highest Levenshtein distance is the length of the longest word that is being compared. [17].

2.5 Models

2.5.1 Tesseract

Tesseract is an open-source OCR engine, it was originally developed by Hewlett Packard between 1984 and 1994 [33]. From 2006 to 2018, it was developed and supported by Google[22]. The Tesseract is well used in many different applications, mainly reading documents in order to make the paper written data editable and searchable data. Tesseract is specialized in high-resolution images or printed text but can also be used in conjunction with an external text detector to recognize text from images [20].

2.5.2 Faster R-CNN

Recent advances when performing object detection are imbursed by *region proposal methods* and *region-based convolutional neural networks* (R-CNNs), more specifically Faster R-CNN [29]. The general idea of Faster R-CNN consists of two parts, one part which proposes regions of interest where there is likely to appear objects, and one part which evaluates and classifies those regions. The first part is called Region Proposal Networks (RPNs) [8], and the second part is the classifier. The full network can be seen in figure 2.5.

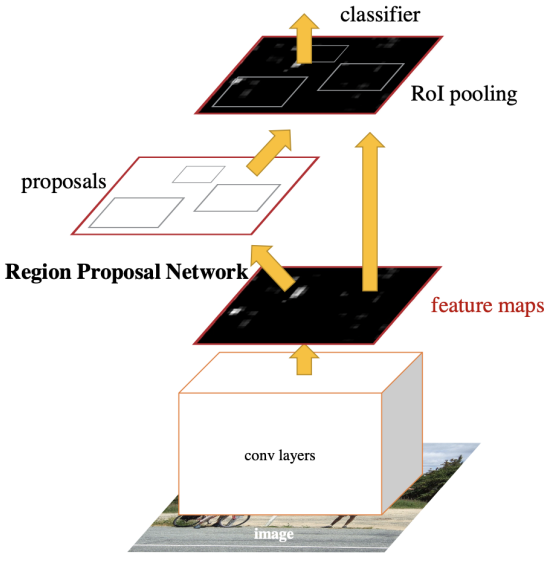


Figure 2.5: Faster R-CNN network.

The RPN takes the feature map output as its input, process that and finally outputs a set of object proposals. The RPN process its input by using a *nxn* sliding window technique, which results in a feature vector. For each sliding window, it predicts several region proposals. This feature vector with proposals are fed into two fully-connected layers, one regression and one classification layer which encode and estimate the probability of an object occurring in the window or not. The first one is a binary classifier which classify the proposal as either background or an object, the second one is generating the bounding box of the object, see figure 2.6 for visualization [29].

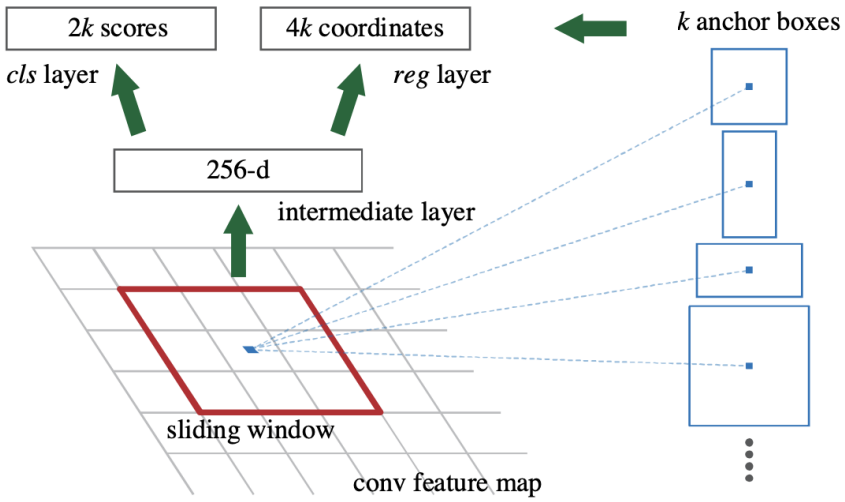


Figure 2.6: Region Proposal Network (RPN) [29]

The output proposals are called anchors, each anchor gets an 'objectness score', which is based on IoU. Each anchor which has an IoU overlap greater than 0.7 is given a positive objectness score, of there are none with an overlap over 0.7, then the highest IoU gets assigned a positive label. For each anchor with an IoU less than 0.3 gets a negative score and is classified as background. Anchors which is assigned neither a positive or a negative score will not affect the training loss [8]. The remaining anchors are then used for training of the next module of the network. [29].

2.5.3 You only look once, YOLO

A popular and extensively used algorithm for object detection is the *YOLO* algorithm (You only look once) proposed in the article *You Only Look Once: Unified, Real-Time Object Detection* [28]. Since the publication of the first algorithm in 2015, 4 major version updates have been developed with a few minor specialized versions. In the paper *YOLOv4: Optimal Speed and Accuracy of Object Detection* the authors show that YOLOv4 has comparable accuracy with other state of the art object detectors but with higher speed [5]. The 5th version is very similar to the 4th version but in the paper *Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs* the authors argue that the fifth version has small improvements and therefore is chosen in this thesis [23].

The YOLO algorithm is developed to be fast, accurate, and light in comparison with the slower more complex Faster R-CNN. The algorithm does that by splitting an image into a grid of cells that is $S * S$ and then B amount of bounding boxes are predicted on each cell

and the bounding boxes confidence score. The bounding boxes can be larger than their cell which has the benefit of not being constrained by the chosen cell size. The confidence score shows how certain the model is that the box contains an object combined with the classification probability, $precision(object) * IOU$. As can be seen in fig 2.7 in the upper picture.

This means that the prediction tensor gets the size $S \times S \times (B * 5 + C)$, B is multiplied by 5 because every bounding box has the x-coordinate and y-coordinate of the center, width, and height of the box and its confidence score, *(x, y, w, h, confidence)* and C is the number of classes. The YOLO algorithm proposes the detection as a regression problem with the benefits of being light and fast without a complex pipeline as in a R-CNN model. Since the network is predicting several bounding boxes simultaneously, the whole picture is seen and used at the same time, which has perks of not missing the whole picture which can be a problem in other algorithms where *sliding window-technique* is used.

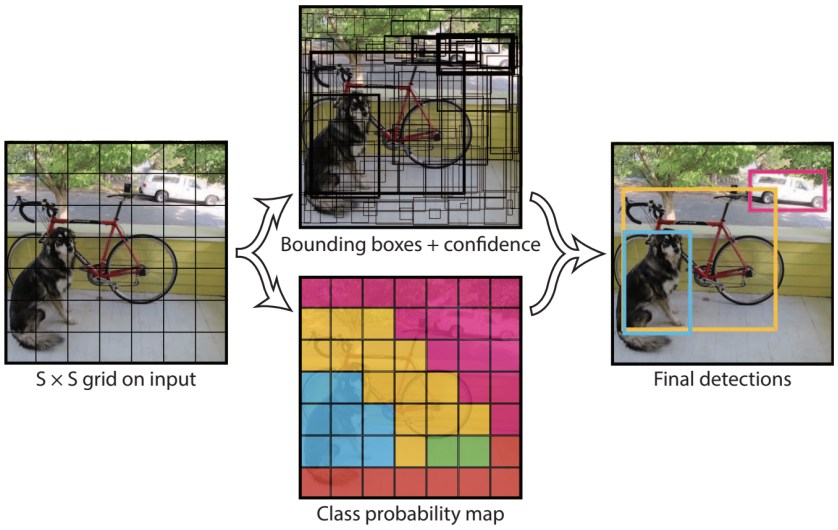


Figure 2.7: The YOLO-model architectural overview.

The YOLOv5 is made up of three parts: Neck, head, and backbone see figure 2.8. The backbone is 29 convolutional layers called CSPDarknet53 which were presented in the previous version of YOLO: YOLOv4. The CSPDarknet53 is several convolutional layers(Conv) combined with different bottlenecks which are layers with fewer neurons after than before which are added to reduce the channels hence the parameters. Focus is a layer made up of two convolutional layers and one bottleneck layer to reduce parameters and increase speed. The SPS-layer or spatial pyramid pooling is a pooling layer that pools the features and generates fixed-length output hence removing the fixed size constraint. The neck is a Path Aggregation Network, or PANet, and is used because of its ability to preserve spatial information when doing the segmentation while being fast. The head is three different convolutional kernels that are performing pixel-wise dense prediction which is predicting a label for each pixel in the image[4].

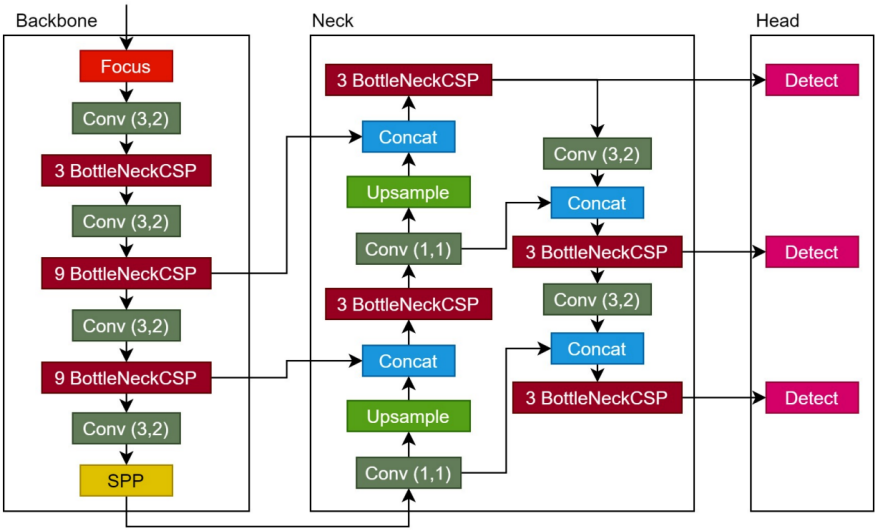


Figure 2.8: The YOLO-model layer architecture.

Chapter 3

Method

This chapter presents the tools, data and models used for this project. We will also describe how we have handled and pre-processed the dataset. Furthermore, we present the implementation of the training process, and the evaluation methods which will later be used when comparing the used models.

3.1 Frameworks and tools

PyTorch is a Python library that is extensively used within machine learning. We chose to use PyTorch since it is one of the fastest performance deep learning libraries. Furthermore, it is also very popular throughout the research community because of its speed and usability. [27]. To make the results replicable the seed was manually set to 3407 both in Pytorch and in Scikit-learn methods. We also used the following open-source Python libraries with packages:

- NumPy
- SciPy
- Pandas
- Matplotlib
- Scikit-learn

Together, these libraries fulfilled the computational and implementation requirements needed for this project [12] [16] [19] [25].

3.2 Data

The dataset that has been used for this thesis is the TRODO dataset [36]. TRODO is a public vehicle odometers dataset for computer vision. The dataset contains 2389 smartphone images taken by users to the company Marketyo [21]. The dataset are images of the dashboard of a car, in front of the driver behind the steering wheel, with purpose to show the odometer of the car.

3.2.1 Data Description

The images consists of both digital and mechanical/analog odometers. In the dataset there are 858 analog odometers and 1531 digital images. When we refer to digital odometers, it means that the digits are presented electronically on a digital screen which can be seen in figure 3.1 b. Analog odometer means that the odometer present the digits on a mechanical roll which can be seen in figure 3.1 a. The images have a high variance in the parameters illumination, resolution and rotation and the dashboards are from different models and car manufacturers. The labeling of the data contains odometer types, position of the odometers and the mileage. The mileage meaning how many kilometers the car has driven.

(a) Analog odometers.

(b) Digital odometers.

Figure 3.1: Samples of the different odometers in both digital and analog in the TRUDO-data set.

Table 3.1: Table over the different digits and their occurrence in the TRODO dataset.

Digit	Occurrence
0	2010
1	2244
2	1552
3	1132
4	1040
5	968
6	1089
7	991
8	1136
9	1024

Images

The images is in JPG format, and the data set contains a total of 2389 images. In figure 3.1 we present some samples of how the different odometers looks within the different odometer types. As shown in 3.1 the odometers may contain additional information to the mileage, such as temperature, date and time. The average size of the images are 886×1173 pixels. Since the odometers belong to different vehicles, it contains a large inter-class variation to the dataset [36].

Annotation

The annotations of the images are provided in a XML file. This file contains information about which objects are in the image, and the corresponding bounding box coordinates. The dataset labels consists of 12 categories, the region of the odometer display, 10 digits and an extra class 'X' which represent all non-digit characters. There is a slight class imbalance for certain digits, which can be seen in tabular 3.1. The region of the odometer display is annotated with 4 coordinates, $x_{min}, y_{min}, x_{max}, y_{max}$, which describes the minimum and maximum value of each axis of which the odometer occurs. Each recognizable digit also has the its own set of coordinates to describe its position, see figure 3.2. On the digital odometer some other digits, a part from the mileage, are annotated. This leads to the implication that our trained models can classify a digit outside the odometer and be correct and are trained on these "faulty annotated digits"[36]. Example of this is that an image contains on average, 6.68 annotated digits, while the maximum number of digits in a odometer is 6. The average number of digits for the mileage are 5.02 digits in this dataset. To annotate the images manually was not done due to time limitation, however, we managed to remove a part of these.

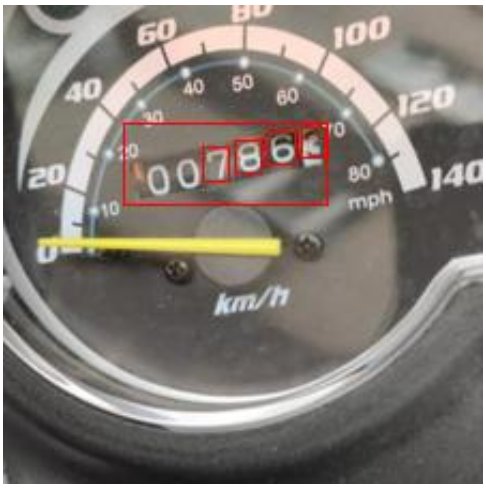


Figure 3.2: Odometer with rectangles drawn on both digits and the entire odometer, using using its $x_{min}, y_{min}, x_{max}, y_{max}$ coordinates.

3.2.2 First and Second Dataset

We separated the data into two different datasets in order to benchmark models that are trained on different datasets, we will elaborate on this in section 3.3. The majority of the images were in different sizes and proportions, so we choose to crop and resize the images, this was done because fewer pixels means fewer parameters which means reduced model

training time. This process was done in two ways, resulting in the two datasets. In the *first dataset* we used as much as possible of the image while making it smaller, for the *second dataset* we cropped only the odometer of the image. Exactly how this was executed will be further explained in the coming section.

The two dataset was randomly splitted with the same seed in to a training part and a validation part with 90% as training and 10% for validation. With the same random seed means that the function that split the datasets are the same. For example, if a specific image was in the first dataset validation subset that image second dataset version would be in the validation subset as well.

3.2.3 Data pre-processing

First of all, we initially removed all characters of the category '*X*', for both datasets, since they do not have any relevance for reading the mileage. For our first dataset, we did standardize the sizes and make all sides the same length, 224 pixels. In this process, we first cropped the images into squares, while using as much as possible of the image. When an image is a rectangle with the horizontal side longer, a vertical square crop is performed using the height of the image and vice versa when the vertical side is longer, see figure 3.3.

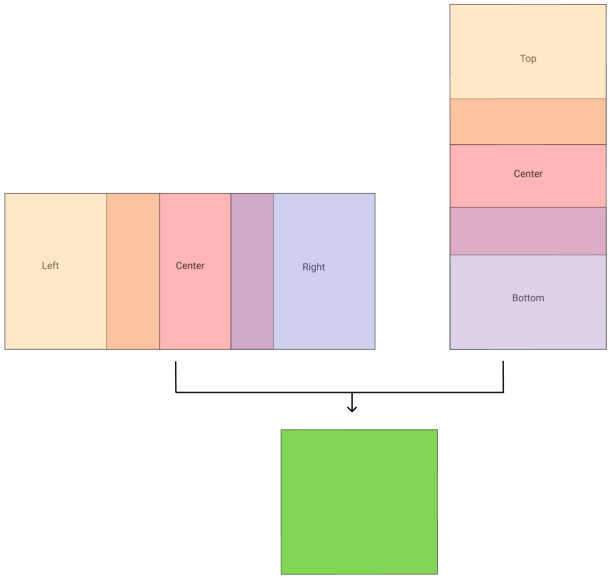


Figure 3.3: Examples of vertical and horizontal cropping depending on where the odometer is in the image. Finally resulting in the green square image.

The first crop results the biggest possible square for each image. Thereafter we choose to crop it further because the images still were very large. We then cropped the images until the odometer area were a minimum of 7% of the entire image. Trough experiments we found that the threshold of 7% were a good match for this dataset. For this process we wrote an algorithm which always keep track of where the odometer is so it does not crop inside the odometer region. In order to avoid cropping the odometer, the crop-process resulted in 9 overlapping crop areas, see figure 3.4. Because of the crop areas are overlapping and there is a threshold for the size of the odometer, the odometer itself will never be cropped

out of the image. Finally when the odometer area is a minimum of 7% of the image, we resize every image to *224 x 224* pixels. The final result is shown in 3.5.

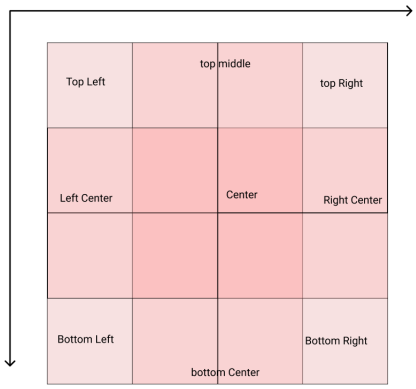


Figure 3.4: The 9 overlapping crop areas which is performed on each image until the odometer area is a minimum of 7% of the entire image.

Figure 3.5: Before and after pre-processing of both analog(top) and digital(bottom) odometer. The final result, for the first dataset, can be seen to the right.

For our second dataset, where we only kept the odometer, we cropped a square around the odometer. After the cropping we performed the same resize procedure as for the first dataset, resulting in images with the same size, *224 x 224* pixels, see 3.6.

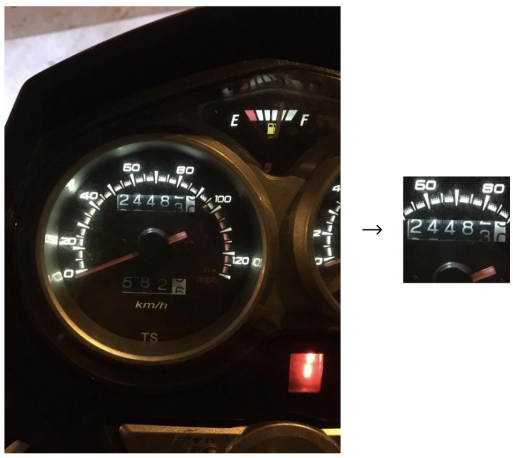


Figure 3.6: Before and after pre-processing, for the second dataset. The final result can be seen to the right.

3.3 Implementation

We have used Virtual Machines(VMs) on different cloud platforms when training our models. The cloud platforms used for this thesis were Microsoft Azure and Google Cloud Platform. These specific platforms were chosen because of their easy availability and free VM options. Since YOLO is a much smaller and faster model, it was trained using a CPU VM which was hosted on Google Cloud, while Faster R-CNN was trained on a GPU VM on Microsoft Azure. On Google Cloud, the VM we used was a CPU with 8 Cores, 32GB RAM, and 50 GB disk. On the Azure platform, we used an NVIDIA Tesla K80-GPU with 6 cores, 56 GB RAM, and 380 GB disk. Our aim was to train both the Faster R-CNN and YOLO on both datasets and evaluate the results.

Our aim was to try two different approaches with both implementations, the purpose of this was to see if the object detection algorithms performed better or worse when the digits were larger and images zoomed in on the odometer. The first approach was to train to detect and recognize both the odometer region together with all digits, we called these models *Multibox-predictor*. For this approach, we only used the first dataset 3.2.2. In the second approach, we divided the goal into two sub-goals, using two different models. One only training to detect and classify the odometer region, using the first dataset, and the second one trained to identify all existing digits, using the second dataset. The model detecting only the odometer will be henceforth called *Singlebox-predictor* and the digit detector will be referenced to as *Digitbox-predictor*, all models are presented in table 3.2. To make the datasets less complex the images in the dataset were transformed to greyscale, but experiments with the YOLO algorithm showed that since the complexity gets reduced the model is more prone to over-fit, see figure A.4. Therefore random grayscaling with a probability of 50% was used when training to avoid this problem. For an overview of the training process, see table 3.3.

Table 3.2: Table over models, and which dataset it is trained on and what it aims to classify

Name	Dataset	Classifying
Multibox	First dataset	odometers, digits
Singlebox	First dataset	odometers
Digitbox	Second dataset	digits

Table 3.3: Table over model metrics from the training process.

Model	Time training	Epochs	Average epoch time
YOLO Digitbox	5h 34m 26s	300	1m 6s
YOLO Singlebox	3h 40m 54s	200	1m 6s
YOLO Multibox	5h 36m 7s	300	1 min 7s
Faster R-CNN Digitbox	9h 4m 51s	15	36m 27s
Faster R-CNN Singlebox	7h 34m 21s	15	30m 21s
Faster R-CNN Multibox	9h 12m 53s	15	36m 58s

3.3.1 YOLO

YOLO was trained with a batch size of 16 with 8 dataload workers with the SGD optimizer. As an activation function in the final detection layer, Sigmoid was used, and in the hidden layers, LeakyReLU was used, which is ReLU modified to have a small slope on the negative side instead of making every negative value zero. The negative slope was set to 0.1 in every use of LeakyReLU. The 5th version of the YOLO algorithm has 5 different sizes and from experiments, the small version was chosen because the bigger more complex versions had slower convergence and lower average precision after 100 epochs. The smaller have the advantage of fewer parameters(approximate 7.2 million) and thereby smaller size [35]. The hyper-parameters were kept on the default setting, see [13]. From this section, when writing YOLO, the small, fifth version is intended if nothing else is specified.

The YOLO models were trained for a maximum of 300 epochs because of the limitations in time and computing power of this thesis. The models were trained in batches of 100 epochs, because of the possibility to stop training when the loss and mAP are not improving, and as can be seen in figure A.3, the loss is improving very little the last 100 epochs. The YOLO models are also approaching over-trained between 200-400 epochs on this dataset which lead to the model's precision dramatically decreasing. An example of a model that is over-trained is when transforming the first dataset to grayscale and training for 300 epochs as can be seen in figure A.4. As can be seen, the mAP dropped to zero at epoch 225 to stay there for a long time and later bump up to the previous result which is an effect of overtraining the model.

When comparing the different YOLO models, as can be seen in figure 3.7, the different YOLO models are reaching similar mAP with small differences, see table 4.2, with the exception for the model that searching only for the odometer boundary box, i.e. the Singlebox-predictor. This can be explained by the second dataset being less complex, with only two classes and only one boundary box to predict in each image.

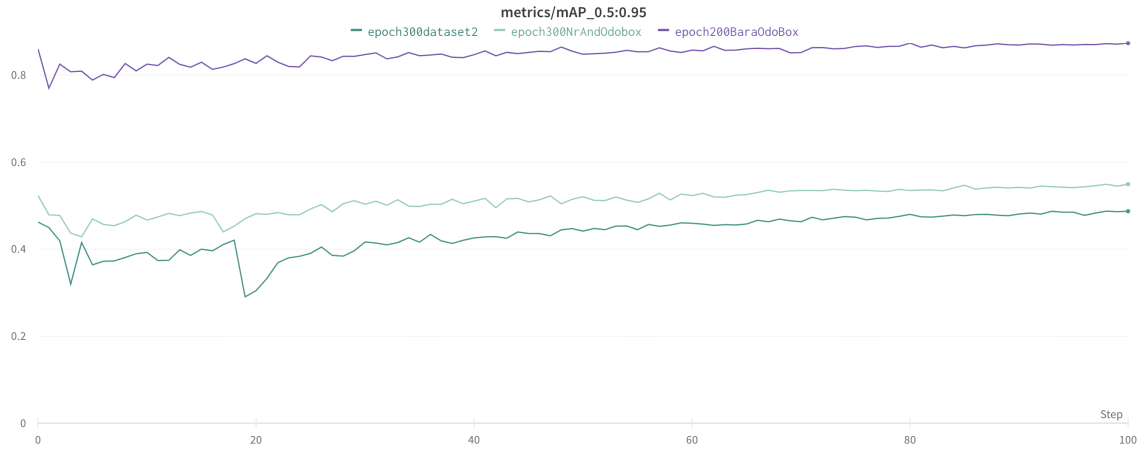


Figure 3.7: YOLO models mAP with threshold 0.5-0.95. The purple line is the Singlebox, the light green is the Digitbox, the green line is the Multibox. Horizontal axis: the epoch-steps, vertical axis: mAP.

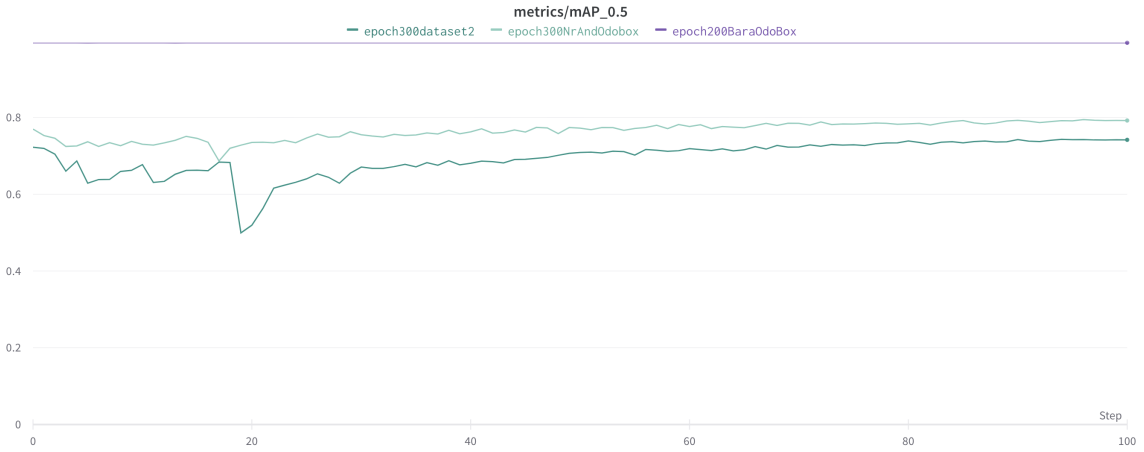


Figure 3.8: YOLO models mAP with threshold 0.5. The purple line is the Singlebox, the light green is the Digitbox, the green line is the Multibox. Horizontal axis: the epoch-steps, vertical axis: mAP.

3.3.2 Faster R-CNN

The Faster R-CNN models have pre-trained weights from the ImageNet dataset [31]. Furthermore, we trained our Faster R-CNN models for an additional 15 epochs, with a batch size of 4. The model uses the SGD optimization function. The learning rate was set to 0.005 and momentum at 0.9. We were limited to 15 epochs per model because of the lack of resources on our VM, but the training result indicated that the models would not perform much better with more training. In figure 3.9, we can see that the mAP has a positive trend curve with an IoU threshold of 0.5:0.95, which may indicate that it has room for improvement. But as mentioned earlier, with a lack of resources we ended the training at 15 epochs.

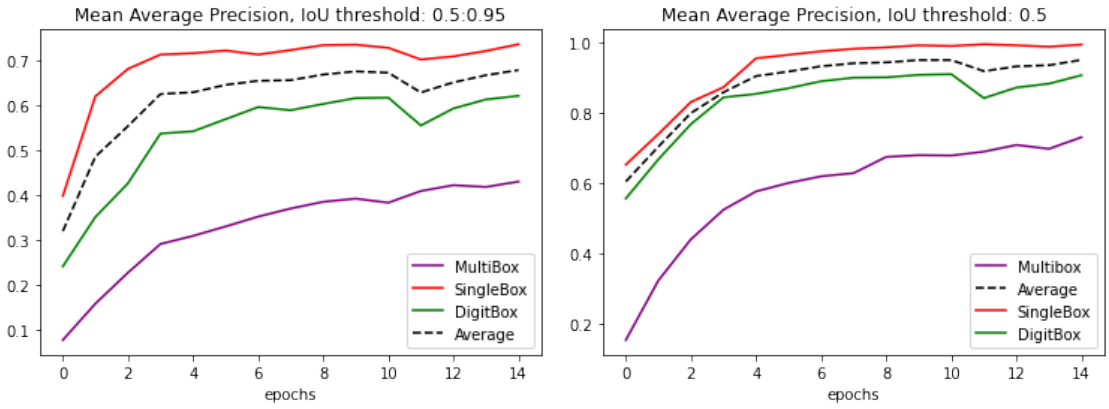


Figure 3.9: Mean Average Precision of each Faster R-CNN model, and the average of Multibox and Singlebox as a dotted line. To the left is the IoU threshold 0.5:95 and to the right 0.5.

VGG-16

The architecture used for the Faster R-CNN models is called *Very Deep Convolutional Networks* (VGG-16) and can be seen in detail in figure 3.10. VGG-16 is using $3x3$ convolutions filters with a depth of 16 layers, which adds up to 138000000 parameters [32]. It has shown that the VGG-16 architecture suits well for image classification. The depth of the model is beneficial for classification accuracy and that state-of-art performance on the ImageNet dataset, and it has shown the importance of depth for a network, in visual representation. The model generalizes well to a wide range of tasks and datasets [29]. The depth of the network makes it more complex, it has approximately 138 million parameters [32].

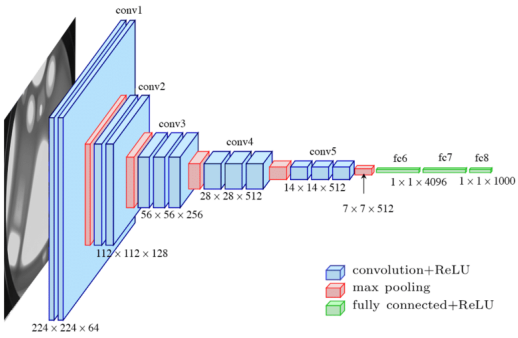


Figure 3.10: VGG-16 standard network [32].

3.3.3 Tesseract

Tesseract is, as previously mentioned, an optical character recognition model(OCR), which means it can not be compared on mAP, in the same way as YOLOv5 and Faster R-CNN, since it does not classify the digits individually but reads the whole image. This implies that all text and digits are included, to increase the accuracy, a custom configuration where page segmentation mode was set to "Assume a single uniform block of text" [14]. The custom configuration was chosen based on the total shortest Levenshtein distance and

easiest to continue to fine-tune. From observations from the data, selective features were added to the Tesseract output, for example, to choose the longest continuous text string and filter away unwanted characters. This was done to improve the Tesseracts accuracy and lower its Levenshtein distance. Since with the custom configuration, the output text string contains *next line* characters(\n) and spaces between the text strings of digits, which made it possible to find the longest continuous number string. And from observing the Tesseract output the longest continuous text string was mainly the one related to the odometer since other readings often were from the clock and other numbers which naturally, is shorter.

3.4 Model Evaluation

Since Faster R-CNN and YOLO are object detection models and Tesseract is an optical character recognition model, evaluation will be done in two steps, where the models' performance first will be evaluated on their Levenshtein distance on 50 randomly selected images from the test dataset. Only 50 images were chosen because of the limitation of time in the thesis. These randomly selected images were 36 digital and 14 analog with an average of 5.36 digits for the analog images and 4.94 digits for the digital images, hence an average of 5.06 digits per image. Furthermore, an additional comparison between the object detection models: Faster R-CNN and YOLO will be evaluated based on their mAP score.

Chapter 4

Results

This chapter presents the final result from the models which is presented in detail in section 2.5. The metrics which is used is mean Average Precision and the Levenshtein distance. Furthermore there are also visualizations included of predictions from each model. The final metrics for the models are summarized in tables 4.1 and 4.2, while some selected predictions are presented in figures 4.1 and 4.2.

In table 4.1, the Levenshtein distance is presented, where we can see how each model is performing when predicting the mileage. Since the Singlebox-model is not classifying digits, it is not included in this table. To compare all the remaining models, we have divided the result into the mean score of analog and digital odometers, and the total mean average of all predictions. Furthermore, the number of exact correct mileage classifications are also presented. As can be seen in table 4.1, Tesseract is on average wrongly classifying 4.02 digits per image on both datasets. In other words, Tesseract are on average classifying 1.04 correct digits per image. The performance is very similar for the analog and digital odometers subsets. It differs an average score of 0.07, which is approximately one digit difference every 14th image. It did however not classify a single correct mileage label.

Table 4.1: Model performance metrics based on Levenshtein distance and mileage predictions. The subset of which the models were tested on, consisted of 50 random images.

Model	Mean total	Mean analog	Mean digital	Mileage prediction
Tesseract, first dataset	4.02	4.07	4.0	0
Tesseract, second dataset	4.02	4.07	4.0	0
YOLO Multibox	2.24	0.29	3.0	26%(13)
YOLO Digitbox	2.52	0.93	3.14	10%(5)
Faster R-CNN Multibox	2.82	0.78	3.61	18% (9)
Faster R-CNN Digitbox	2.82	0.21	3.83	28%(14)

Both the YOLO-models and the Faster RCNN-models are out-preforming Tesseract regarding the Levenshtein distance. They are predicting the mileage with a higher accuracy with more absolute correct results. The YOLO models Multibox and Digitbox has an average of 2.24 and 2.52 wrongly classified digits. Both the Multibox and Digitbox gets a lower Levenshtein distance on the analog odometers, where the Multibox is performing better with an average distance of 0.29. Furthermore, Multibox is better than the Digitbox

on the digital odometers as well as correct predicted mileage. For the Faster R-CNN models, unlike the YOLO models, the Digitbox generally performed better than the Multibox. Digitbox reached a lower Levenshtein distance for the analog odometers, but worse on the digital. The number of correct predictions for the mileage was ultimately achieved by the Faster R-CNN Digitbox. Every model performance is significantly worse for the digital compared to analog odometers. The best performances for the analog odometers is the YOLO Multibox and Faster R-CNN Digitbox, where they achieve a Levenshtein distance of 0.29 and 0.21, which means an average of 1 digit wrong for every 3.5, and 4.8 image.

The final mAP with two IoU thresholds for each model can be seen in table 4.2. Both with YOLO and Faster-R-CNN, the Singlebox-models achieve the highest mAP. For Faster R-CNN the Digitbox-model are outperforming the Faster R-CNN Multibox while the YOLO Multibox attain a higher mAP than YOLO Digitbox. The highest overall mAP is reached by Faster R-CNN Digitbox.

Table 4.2: Table over the models performance.

Model	mAP 0.5-0.95	mAP 0.5
YOLO Digitbox	0.487	0.742
YOLO Singlebox	0.873	0.995
YOLO Multibox	0.549	0.792
Faster R-CNN Digitbox	0.620	0.906
Faster R-CNN Singlebox	0.735	0.993
Faster R-CNN Multibox	0.429	0.730

The following figures shows the predicted boundary boxes, and classification of odometers. Figure 4.1 shows the YOLO-models, and figure 4.2 shows the Faster R-CNN-models. The figures contain the same 5 images, randomly selected among the evaluation-set and the predictions visualized have an confidence score of > 0.5.

Figure 4.1: Visualization of Boundary box predictions, and digit classification from YOLO-models. (a) YOLO Singlebox-model. (b) YOLO Digitbox-model. (c) YOLO Multibox-model.

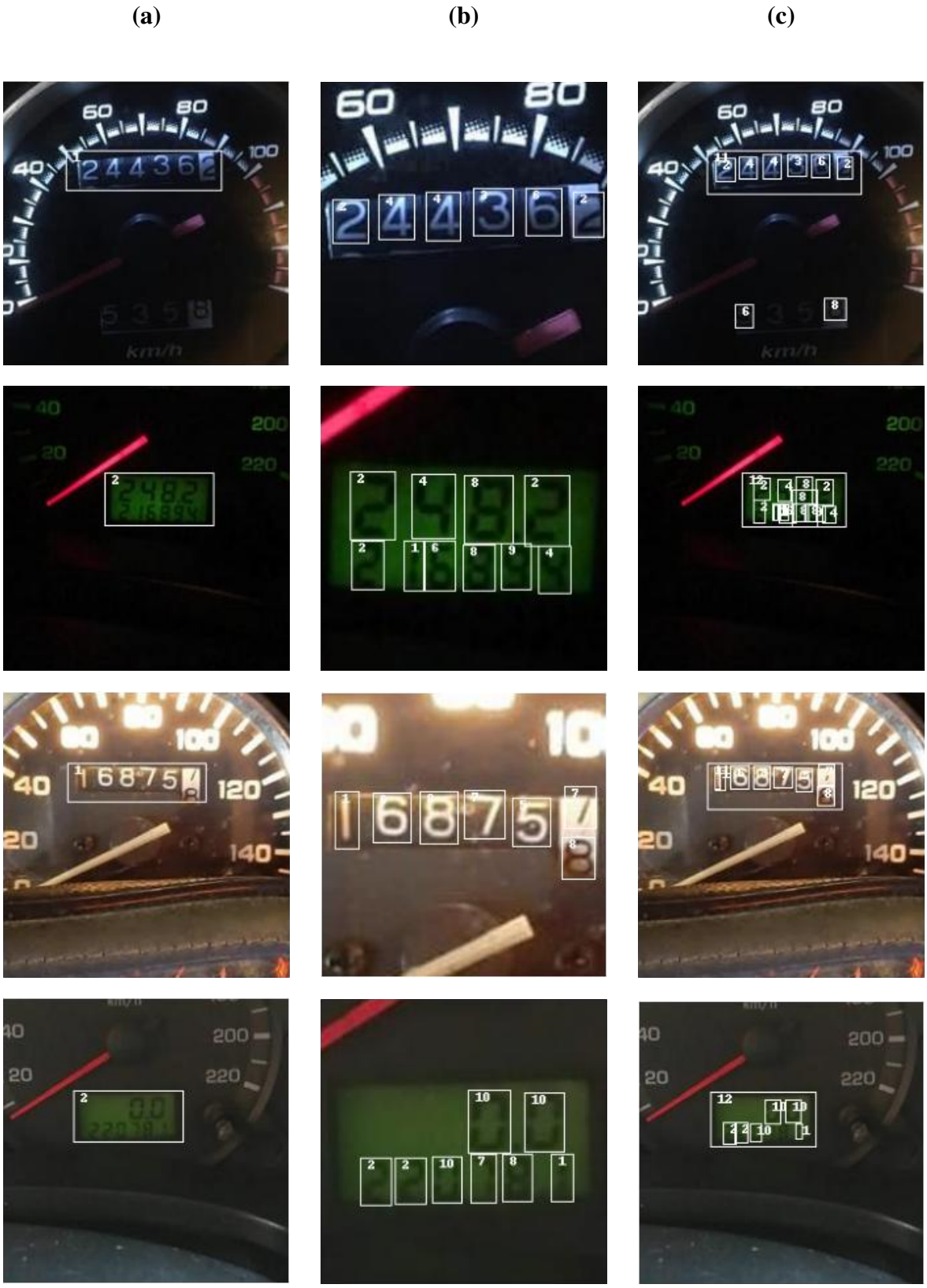


Figure 4.2: Faster R-CNN visualization of Boundary box predictions, and digit classification. (a) Faster R-CNN Singlebox-model. (b) Faster R-CNN Digitbox-model. (c) Faster R-CNN Multibox-model. Observe that in the classification, the class '10', represents the digit '0'. Regarding the odometer classification, '1' stands for analog and '2' represents digital odometer.

Chapter 5

Discussion

In this chapter, we will discuss the result presented in chapter 4. Firstly we will discuss the evaluation of the final models, based on the measurements of Levenshtein Distance and mAP, and then we will discuss that it might be possible to assemble a model pipeline with Singlebox and Digitbox. Furthermore, we will discuss the dataset and its flaws, and finally general limitations.

5.1 Final Models

5.1.1 Levenshtein Distance

As shown in the model evaluation based on Levenshtein distance, which is summarized in table 4.1, the Tesseract has a 0% accuracy. The two models that achieved the highest prediction accuracy are Faster R-CNN Digitbox and YOLO Multibox. Interestingly, Tesseract does not improve when tested on the second dataset, where the images are cropped just around the odometer.

From the result it can be seen that, depending on which measurement of Levenshtein distance, the models have different strengths in terms of classifying digits. The Faster R-CNN Digitbox is achieving a higher accuracy than YOLO Multibox regarding correct observations with 28% correct predictions versus the YOLO algorithm that has an accuracy of 26%. Looking at the mean value of Levenshtein distance, the YOLO algorithm has a slight advantage against Faster R-CNN, which can be explained by the Faster R-CNN have worse accuracy when classifying digital odometers, while performing well for reading the analog odometers.

What we found notable is that the Faster R-CNN and YOLO do not follow the same pattern of performance depending on which dataset it is trained and tested on, which is presented in table 4.1. The Faster R-CNN algorithm benefits from having the area of the odometer as large as possible, while the YOLO algorithm performs better on the entire image. To investigate exactly the reason why this happens falls outside of the scope of this thesis, but it might be because of the different number of parameters in each algorithm. The Faster R-CNN has approximately 20 times more parameters, which means that more complex structures can be described. By cropping the images around the odometer, the image gets fewer irrelevant pixels which with Faster R-CNN means it has less trouble finding

the relevant digits, and several wrongly classified areas are removed, as can be seen in figure4.2. For the Faster R-CNN approaches, it has been difficult to set a good confidence score for the predictions, the algorithm sometimes classifies too many objects as relevant and sometimes too few. The YOLO algorithm also has this problem when classifying automatic images but not to the same extent, as can be seen in figure 4.1. In contrary almost the opposite, where digits go unclassified and could benefit from sorting on a lower confidence score.

One main reason why we see a relatively low prediction accuracy is probably because of the test set consisting of a majority of digital odometers, which has been proven to be more difficult to read. The lowest Levenshtein distance for analog odometers, is achieved by the Faster R-CNN Digitbox followed YOLO Multibox. But for the digital odometers, the YOLO-models is outperforming Faster R-CNN. Tesseract does classify a few digits correctly but never the entire mileage correct, thereby making it an inadequate option for this problem. With zero correct predictions, it does not seem like Tesseract is fit for this problem.

5.1.2 Mean Average Precision, mAP

The final result for the mAP shown in table 4.2, displays as expected, that both the Singlebox models achieve the highest mAP. This is expected since its objective is simply to find and classify the odometer type. The YOLO Singlebox model is performing better, probably because of its simplicity, in regards to the number of parameters, which has made it possible to train for more epochs. Furthermore, Faster R-CNN Digitbox is performing best, followed by YOLO Multibox. Due to the complexity of the Faster R-CNN algorithm, the number of epochs we were able to train it for was limited, both because of limited time and resources.

When comparing each Digitbox-model with their corresponding Multibox-model, we can see that for the Faster R-CNN algorithm, the Digitbox is significantly better than the Multibox. The YOLO-algorithm is, on the contrary, performing slightly better with its Multibox-model. Why so, is not entirely clear. One factor, that increases the mAP, can be that the Multibox has one extra object to classify - the odometer. In which the model has very high accuracy, therefore, it achieves a higher mAP. Concerning the Faster R-CNN Digitbox, the correct classified digits outweigh this factor.

The Faster R-CNN Digitbox-model is achieving the overall highest precision for classifying digits. In this case, Faster R-CNN probably has an advantage because of its complexity and number of parameters, compared to the simpler task when finding and classifying the odometer. The worse precision achieved is also Faster R-CNN but the Multibox-model. The reason for this could be the limited training since the Multibox has the most complex task. We could see that, especially for the digital odometers, this model is very likely to propose many more digits than there are displayed in the image. Unlike YOLO, which is more likely to miss to classify digits rather than classify too many. Which is very clear from the digital odometer predictions in figure 4.2 and 4.1. For some reason the YOLO algorithm has a lower confidence score for the proposed objects when training on the second dataset, i.e the cropped images, which means that YOLO seems to perform worse on larger objects. The underlying reason for this is not known and needs further research and investigation. The inconsistent annotations, for instance, zeros on the analog odometers or the clock for digital odometers, probably had a negative effect on the mAP,

which we will elaborate in section 5.2.

5.1.3 Model Pipeline

Because of the impressive result of the YOLO Singlebox-model, with a threshold of 0.5 mAP got a score of 0.995, see 4.2, and as can be seen in figure 4.1 the predicted boundary box is very precise. Because of this, we argue that we can consider the final result of the Digitbox-models as a potential solution to the main object of this project since a pipeline with YOLO Singlebox-model could be applicable. The pipeline would be arranged where the output of the Singlebox-model is the input to the Digitbox-model. Our proposed pipeline would consist of the YOLO Singlebox-model combined with the Faster R-CNN Digitbox-model because of their high performance compared to the other models.

5.2 Data

The dataset we have used to train our models, TRODO [36], has many properties, which if modified, could improve the result considerably. As described in section 3.2.3, we removed all characters in the annotation file labeled as *X*, but this was not all characters excluded mileage. We believe that the flawed annotation for the digital odometers has negatively affected the result. Because the annotations contain much additional information beyond the mileage, e.g. clock, temperature, kilometers left of fuel, etc, the models get trained on them as well, meaning the models want to predict numbers outside the odometer, which is not the intention of this thesis. This problem could have been avoided if the digital odometer bounding box was annotated just around the actual mileage number and removed all annotations of irrelevant digits, making the models more prone to ignore the irrelevant digits. As done with the Tesseract, digits not presented in long sequence based on their boundary box could be filtered out to improve the result. Since the majority of the wrongly annotated images were on the digital odometers, we suspect that the wrongly annotated images could have a considerable effect on the result, which could be seen in 4.1.

5.3 Limitations

In this thesis, and the practical work behind it, we encountered many limitations, in which we handled, what we thought, was the best way possible. Since training neural networks is both compute and time-consuming. Faster R-CNN was nearly impossible to train locally on our CPU. Because of the long training time, a Virtual Machine (VM) was necessary for the training process. Since running a VM is not free, the models could not be trained for additional epochs. This probably affected the precision and accuracy negatively for our final models. We believe that the models could have been improved, simply by letting them train for more epochs. Looking at figure A.6 and figure 3.9, it is possible to acknowledge that the loss is still improving and the mAP has not started to stagnate when approaching the 15th epoch. Therefore we argue that the Faster R-CNN algorithm would benefit from more training time.

Furthermore, the main limitation of this project was the acquired dataset used to train the data. Mainly because of the inconsistency in the annotations and the limited number of images, as we argued in the previous section.

Chapter 6

Conclusion

From our experiments it can clearly be seen that the niche object detection models are outpreforming the more general OCR model on this application. Because the big variation in how the digits looked the Tesseract had trouble to read them. Overall, we would like to achieve higher precision and accuracy for all the models, except the Singlebox-models - which we are very impressed by. The limited amount of data and the inconsistent data annotation were probably the biggest factors for the models not performing better. However, we believe that a few actions could significantly improve the overall mAP, accuracy, and Levenshtein distance for the models. Either, the removal of all the digital odometer digits which are not included in the mileage, or the adjustment of the digital odometer bounding box, both of these measures would probably increase both the accuracy and the Levenshtein significantly.

Interestingly, the object detection models have similar performance depending on what performance measurement is used, the Multibox YOLO model or the Digitbox Faster R-CNN is the preferred model. We can also observe a relationship between mAP and Levenshtein distance, which support that these measurements are suitable when comparing OCR models with object detection models regarding classification numbers. And that mAP combined with Levenshtein distance are good measurements when further investigating which object detection model to choose. Our belief is that the Faster R-CNN models could, with further training and optimizing the confidence score, outperform the YOLO models regarding accuracy and precision. But with limited time and memory, the better practical implementation is the YOLO models with an optimized confidence score.

For this project, it is important to recognize that one single digit wrong, in the output of predicted mileage, makes the prediction more or less worthless. Since the object is to predict the mileage, one digit wrong does affect the output massively, for example, predicting 9000 km instead of 1000 km, is a big difference. With that in mind, the model that would achieve sufficient accuracy for this objective in a practical application, would be to organize a pipeline of YOLO Singlebox followed by Faster R-CNN Digitbox. This pipeline would achieve an acceptable on analog odometers, but regarding digital odometers, on behalf of Greater Than's objective, there is further work that has to be done.

6.1 Future work

To be able to implement a solution for production, the accuracy of the models needs to be improved. There is a lot that can be done to enhance the accuracy. First and foremost, more data is required, mainly for digital odometers, since those have been the biggest challenge. We are not sure exactly how much additional data is necessary for the models to accomplish an acceptable accuracy. If there is no other dataset available, there are a lot of modifications or pre-processing that can be done on the TRODO dataset, more specifically for the images with the digital odometers. One approach would be to modify the annotations as previously said, e.g. remove all labels for the irrelevant digits. We have reason to believe that this would make the models more resistant to classifying the wrong numbers as the mileage. Another approach would be instead of removing all the irrelevant digits, we classify them with the label X. This approach was tried by the TRODO-dataset creators when they annotated the dataset, unfortunately with limited success, since we found many irrelevant digits that had the wrong classification.

Examine and investigate how the result would be affected by further training of the algorithms, mainly for Faster R-CNN. It might also be a way to build a more robust model. This is because the mAP was still improving when training was stopped for the Faster R-CNN. With more training time, it might be interesting to specialize models for the different types of odometers. Since the Singlebox-predictors are performing very well classifying which type of odometer it is, it would be easy to then use different models, in the model pipeline, depending on which odometer type it is.

Further processing of the object detection model's output. As was done with Tesseract's output, where the predicted mileage was stripped on the single-digit based on its position in the image, and then the longest continuous number line was kept based on the coordinates of the predicted boundary box. By doing this, the models might achieve higher accuracy by filtering digits that are likely to come from other values than the mileage, e.g. temperature or time.

Additional evaluation measurements could be applied to give a deeper understanding of where the models has problems. One measurement that would be interesting, is the difference in the output to the ground truth of what the odometer is showing. If the odometer is showing 1000 the Levenshtein distance is the same for 100 and 1001, but the second prediction might be acceptable while the first one is not. This could provide a broader understanding of the result and challenges for the practical work of this thesis.

Bibliography

[1] Shailesh Acharya and Glenn Fung. "Mileage Extraction From Odometer Pictures for Automating Auto Insurance Processes". In: *Frontiers in Applied Mathematics and Statistics* 5 (2019). URL: https://www.frontiersin.org/article/10.3389/fams.2019.00061.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[3] *Artificiellt Neuronät*. http://www.termado.com/DatatermSearch/?ss=neuralt+n%u00e4tverk. Accessed: 2010-09-30.

[4] Aduen Benjumea et al. "YOLO-Z: Improving small object detection in YOLOv5 for autonomous vehicles". In: *CoRR* abs/2112.11798 (2021). arXiv: 2112.11798. URL: https://arxiv.org/abs/2112.11798.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-yuan Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: (Apr. 2020).

[6] Alexandre Brébisson and Pascal Vincent. "An Exploration of Softmax Alternatives Belonging to the Spherical Loss Family". In: (Nov. 2015).

[7] Jason Brownlee. *Probability for Machine Learning: Discover How To Harness Uncertainty With Python*. Machine Learning Mastery, 2019.

[8] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[10] D.O. Hebb. *The Organization of Behavior*. Wiley: New York, 1949. ISBN: 978-1-135-63190-1.

[11] Mohammad Hossin and Md Nasir Sulaiman. "A review on evaluation metrics for data classification evaluations". In: *International journal of data mining & knowledge management process* 5.2 (2015), p. 1.

[12] John Hunter et al. *Matplotlib documentation*. Accessed: 2022-02-08. 2012. URL: https://matplotlib.org/stable/.

[13] *Hyperparameters YOLOv5*. https://github.com/ultralytics/yolov5/blob/master/data/hyps/hyp.scratch-low.yaml. Accessed: 2022-05-11.

[14] *Improving the quality of the output: Tesseract documentation*. https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html. Accessed: 2022-05-3.

[15] Katarzyna Janocha and Wojciech Marian Czarnecki. "On Loss Functions for Deep Neural Networks in Classification". In: *CoRR* abs/1702.05659 (2017). arXiv: 1702.05659. URL: http://arxiv.org/abs/1702.05659.

[16] Eric Jones, Travis Oliphant, and Pearu Peterson. *SciPy, Open source scientific tools for Python*. Accessed: 2022-02-08. 2001. URL: http://www.scipy.org.

[17] Vladimir I Levenshtein et al. "Binary codes capable of correcting deletions, insertions, and reversals". In: *Soviet physics doklady*. Vol. 10. 8. Soviet Union. 1966, pp. 707–710.

[18] Andreas Lindholm et al. *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022. URL: https://smlbook.org.

[19] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[20] Ravina Mithe, Supriya Indalkar, and Nilam Divekar. "Optical character recognition". In: *International journal of recent technology and engineering (IJRTE)* 2.1 (2013), pp. 72–75.

[21] Kaouther Mouheb, Ali Yürekli, and Burcu Yılmazel. "TRODO: A public vehicle odometers dataset for computer vision". In: *Data in Brief* 38 (2021), p. 107321. ISSN: 2352-3409. DOI: https://doi.org/10.1016/j.dib.2021.107321. URL: https://www.sciencedirect.com/science/article/pii/S2352340921006053.

[22] F. B. Neal and John C. Russ. *The Image Processing Handbook, 7th Edition*. CRC Press, 2018.

[23] Upesh Nepal and Hossein Eslamiat. "Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs". In: *Sensors* 22.2 (2022), p. 464. ISSN: 1424-8220. DOI: 10.3390/s22020464. URL: http://dx.doi.org/10.3390/s22020464.

[24] Michael A Nielsen. *Neural Networks and Deep Learning*. Accessed: 2022-02-10. 2015. URL: http://neuralnetworksanddeeplearning.com.

[25] Travis Oliphant. *NumPy: A guide to NumPy*. Accessed: 2022-02-08. 2006. URL: http://www.numpy.org/.

[26] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. "A Survey on Performance Metrics for Object-Detection Algorithms". In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)* (2020), pp. 237–242.

[27] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f701272Paper.pdf.

[28] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. DOI: 10.48550/ARXIV.1506.02640. URL: https://arxiv.org/abs/1506.02640.

[29] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[30] Adrian Rosebrock. *Intersection over Union (IoU) for object detection*. Accessed: 2022-04-04. 2021. URL: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[31] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[32] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

[33] R. Smith. "An Overview of the Tesseract OCR Engine". In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 2. 2007, pp. 629–633. DOI: 10.1109/ICDAR.2007.4376991.

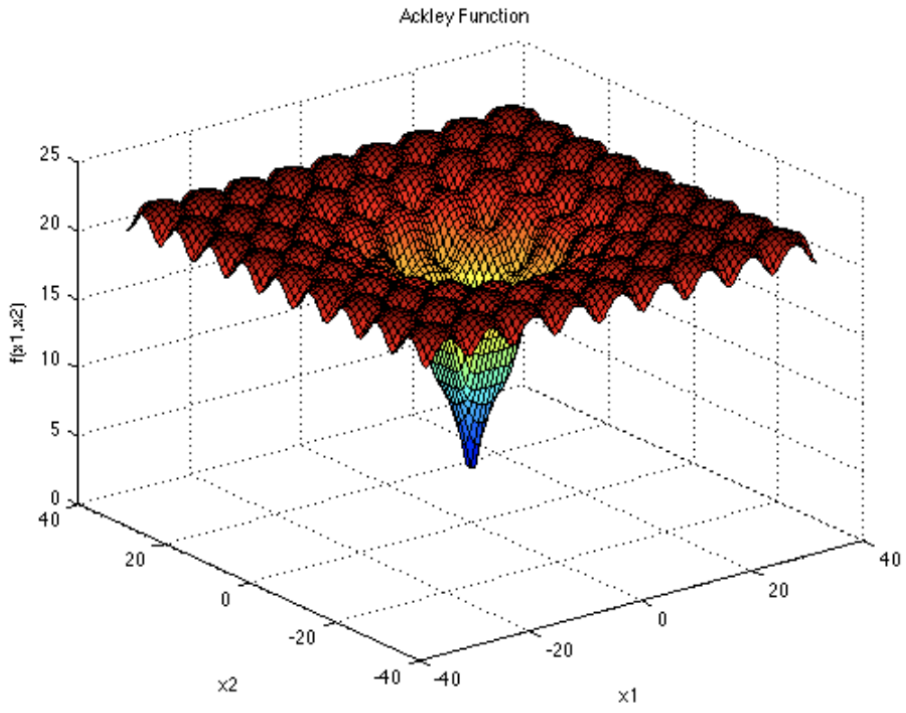
[34] Sonja Surjanovic and Derek Bingham. *Optimization Test Functions - ACKLEY FUNCTION*. Accessed: 2022-02-09. 2013. URL: http://www.sfu.ca/~ssurjano/ackley.html.

[35] *YOLOv5*. https://github.com/ultralytics/yolov5. Accessed: 2022-05-19.

[36] Ali Yürekli, Burcu Yılmazel, and Kaouther Mouheb. *TRODO: a public vehicle odometers dataset for computer vision*. Accessed: 2022-01-20. 2021. URL: https://data.mendeley.com/datasets/6y8m379mkt/2.

Appendix A

A.1 Images to visualize theory explanations



$$f(\mathbf{x}) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

Figure A.1: Example over a complex function that have multiple local minimum and a global minimum [34].

A.2 YOLO training and validation images

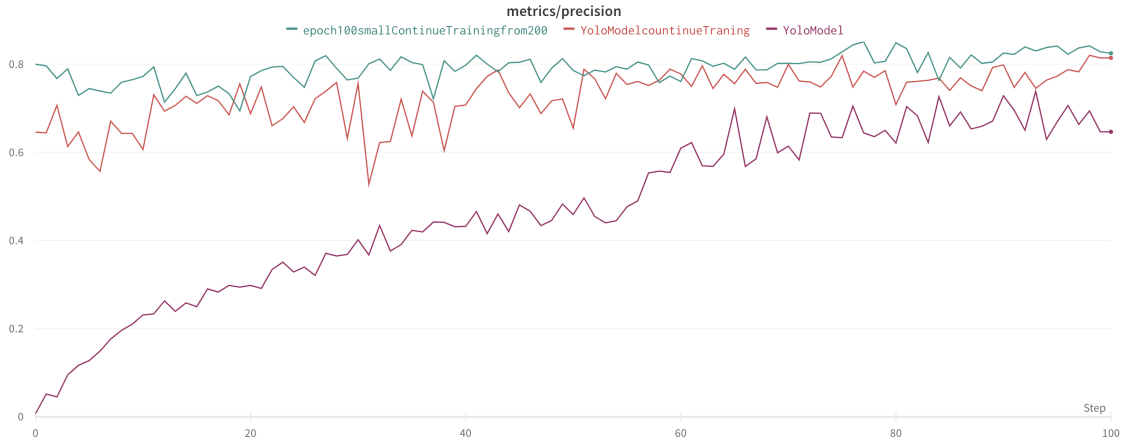


Figure A.2: YOLO-model trained on the first dataset without the odometer boundary box in three rounds, 100 epochs each. With the purple line is the first 100 epochs the red is the epochs between 100-200 and the green line is the last 100 epochs. Horizontal axis: the epoch-steps, vertical axis: the model precision.

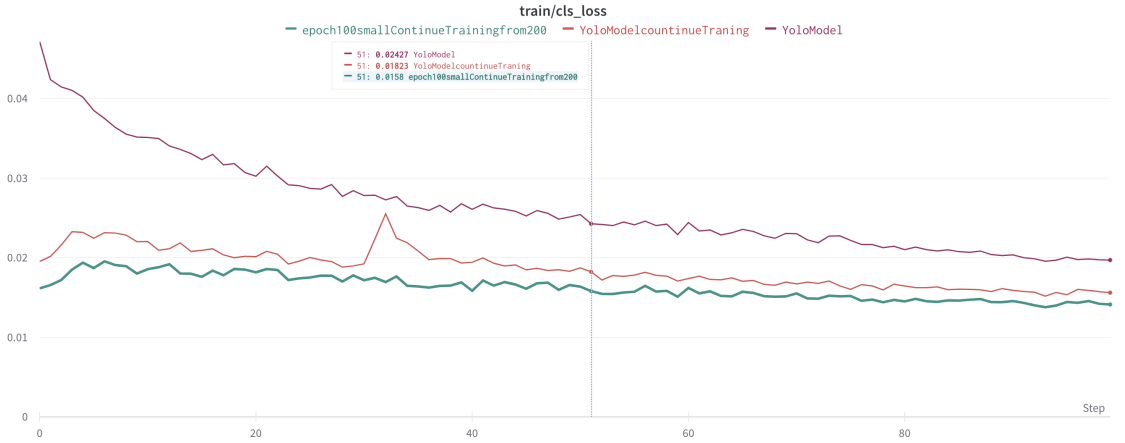


Figure A.3: Multibox Yolo model trained in three rounds, 100 epochs each. With the purple line is the first 100 epochs the red is the epochs between 100-200 and the green line is the last 100 epochs. Horizontal axis: the epoch-steps, vertical axis: The classification loss

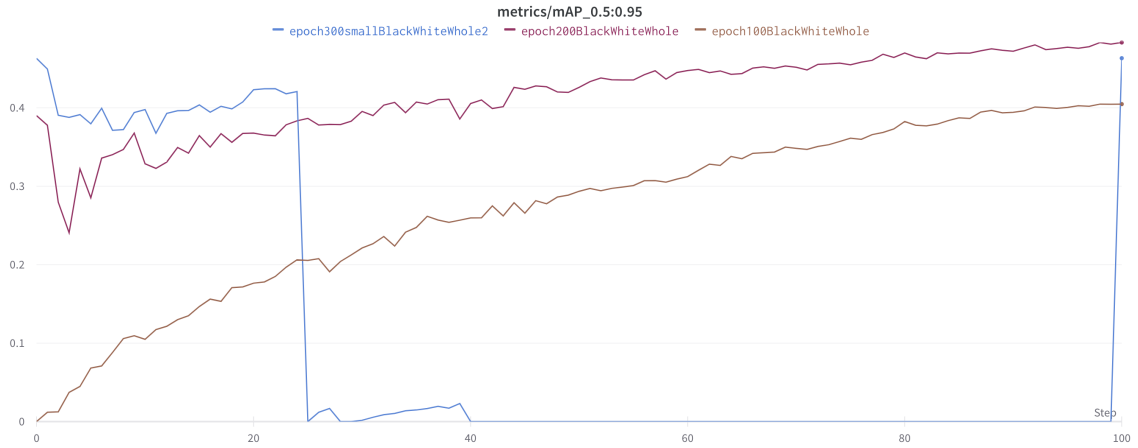


Figure A.4: Multibox Yolo model evaluated on the first data set transformed to greyscale in three rounds 100 epochs each. With the brown line is the first 100 epochs the purple is the epochs between 100-200 and the blue line is the last 100 epochs. Horizontal axis: the epoch-steps, vertical axis: mAP with threshold on 0.5-0.95.

(a) **(b)** **(c)** **(d)**

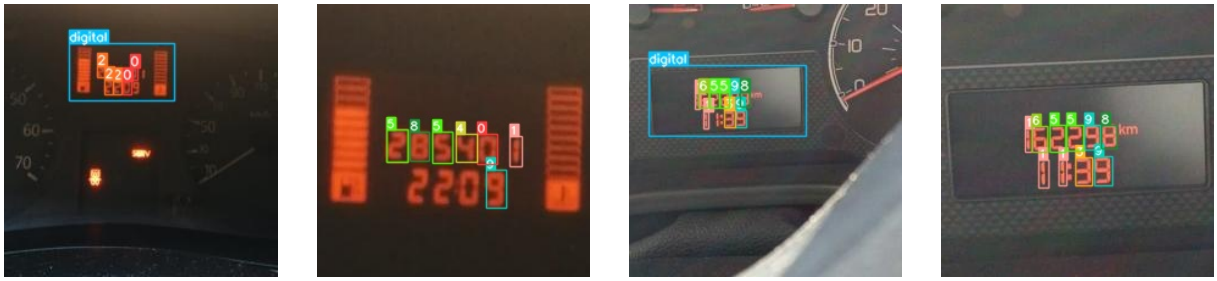


Figure A.5: Selected images on trouble the models have when classifying numbers in a digital odometer. (a) Multibox YOLO-model. (b) Digitbox YOLO-model trained with same image as in (a). (c) Multibox YOLO-model. (b) Digitbox YOLO-model trained with same image as in (c).

A.3 Faster R-CNN training and validation images

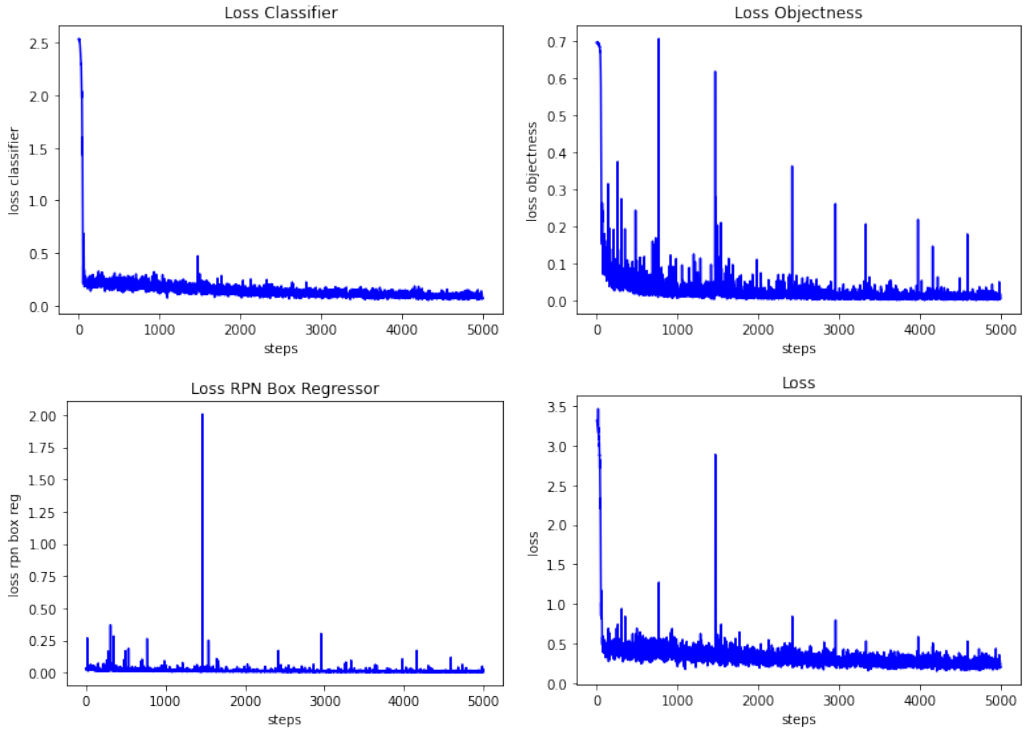


Figure A.6: DigitBox training metrics.

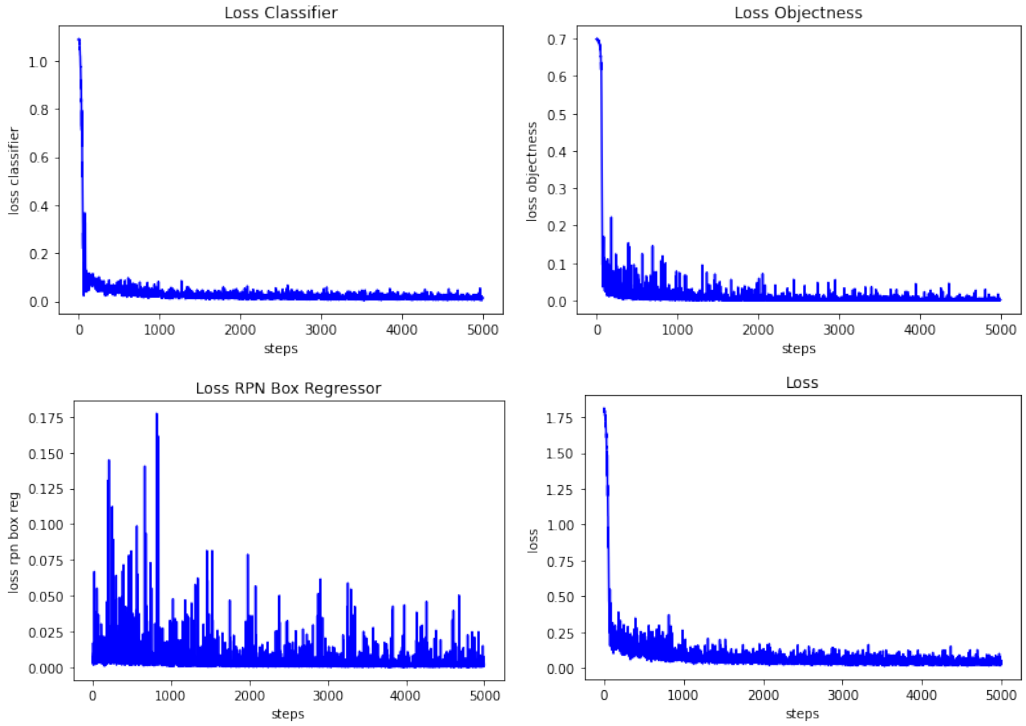


Figure A.7: SingleBox training metrics.

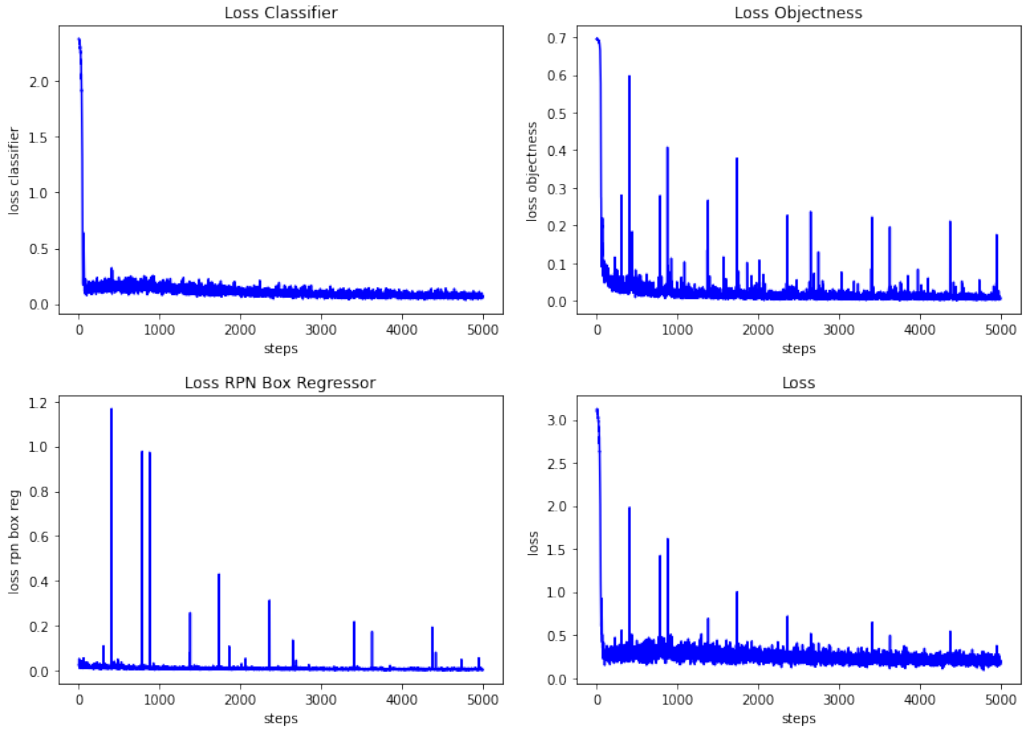


Figure A.8: Multibox training metrics.

(a) **(b)** **(c)** **(d)**

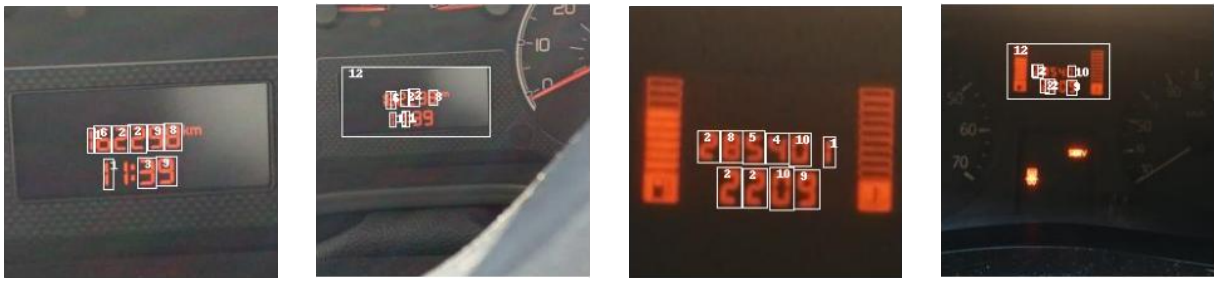


Figure A.9: Selected images on trouble the models have when classifying numbers in a digital odometer. (a) Digitbox faster R-CNN-model. (b) Multibox R-CNN-model trained with same image as in (a). (c) Digitbox faster R-CNN-model. (d) Multibox R-CNN-model trained with same image as in (c).