CHALMERS
UNIVERSITY OF TECHNOLOGY

Deep Path Planning Using Images and Object Data

Master's thesis in Engineering Mathematics and Computational Science

MARIA BERGQVIST

Master's thesis in Complex Adaptive Systems

OSKAR RÖDHOLM

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018:EX020

Deep Path Planning Using Images and Object Data

MARIA BERGQVIST
OSKAR RÖDHOLM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signals Processing and Biomedical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

Deep Path Planning Using Images and Object Data
MARIA BERGQVIST
OSKAR RÖDHOLM

Supervisor: Nasser Mohammadiha, Zenuity
Christopher Innocenti, Zenuity
Lennart Svensson, Department of Electrical Engineering

Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2018:EX020
Department of Electrical Engineering
Division of Signals Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Deep Path Planning Using Images and Object Data
MARIA BERGQVIST
OSKAR RÖDHOLM
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous vehicles are currently being heavily researched with the purpose of finding a manner in which a vehicle can drive safely. An approach for doing this is imitation learning by which human behavior is imitated, and which recently has found success in steering vehicles using images. For this thesis, the purpose is to investigate whether imitation learning can be applied to path planning for autonomous vehicles using several sources of input data. We also want to investigate if merging different types of information will improve the safety and robustness of the vehicle. The aim is therefore to design and implement a neural network which merges input data from several sources and predicts the path of the vehicle 5 seconds into the future.

This thesis compares a variety of neural network models used to predict the path. Some models use data from only one source, while other merge the data. The results show that it is possible to use deep learning to predict the path for autonomous vehicles in some situations. They also show that including images as input to the model does not significantly improve the performance. Instead, using only object data make the networks performs as good as merging the inputs. However, more research are needed in order to find a more accurate and stable model.

Keywords: autonomous driving, deep learning, neural networks, machine learning, path planning, convolutional neural networks, recurrent neural networks.

Acknowledgements

We would like to thank our supervisors at Zenuity, Nasser Mohammadiha and Christopher Innocenti, for their help and support. Their input in discussing aspects and potential solutions to the task has been very helpful throughout the thesis. Lennart Svensson, our supervisor at Chalmers, also deserves a special thanks for his insight and for steering us in the right direction in the project. We would also like to everyone at Zenuity who has helped us continuously and Volvo Cars for providing us with good data. Last but not least, we want to thank all our friends, who have supported us throughout and provided much needed distractions.

Maria Bergqvist and Oskar Rödholm, Gothenburg, June 2018

Contents

1

Introduction

This chapter covers the background on autonomous vehicles and deep learning. It also presents some previous work and the objective of this thesis. Finally the outline of the thesis is presented.

1.1 Background

Since the early twentieth century automobiles have been the main form of transportation. Today, most vehicle manufacturers include software for autonomous driving. These vary from cruise control on highways to autopilot capable of steering and properly adjusting speed. The advances in this area have shown that it might be possible to one day have vehicles which are completely autonomous.

A study by RethinkX predicts that only 20 % of Americans will own a car in 15 years and 95 % of all passenger miles traveled will be done by Transportation-as-a-Service providers using autonomous vehicles [1]. In order to meet this estimation, the software for autonomous driving will need to become more intelligent so the vehicles can drive autonomously and safely.

The first big movement of autonomous cars, came in the form of "The Grand Challenge". The competition was organized by DARPA and was organized multiple years between 2005-2012. The first installation was set in the desert, with not a single vehicle finishing the race. However, during the second installation in 2006, 6 cars finished the race before the time limit. The winner, Stanley, was designed with carefully implemented control systems, based on sensor data collected by the car [2].

Nowadays, companies such as Waymo [3], NVIDIA [4], Volvo Cars and Zenuity [5] are using deep learning to develop their autonomous vehicles. In 2015, Waymo were able of driving a fully autonomous car on public roads and in 2017 they began trials in Arizona with no driver behind the wheel in their autonomous cars [6]. NVIDIA also showed in 2016 that it was possible to do lane and road following using deep learning [7]. In Gothenburg, the Drive Me project by Volvo Cars is working towards fully autonomous commercial vehicles by 2021 [8].

An autonomous vehicle is typically controlled using a carefully designed hierarchical decision control system and data collected by sensors on the vehicle [9]. The vehicle has to fully explore the surrounding area and only once it has been fully explored is the vehicle capable of driving in a safe manner. While this method seems to work on small scale, it is expensive and time-consuming to do this at all times while driving.

A method for avoiding the process of exploring the whole environment is to imitate the behavior of a human. This can be done by a form of deep learning called imitation learning. In this, the actions of a human are recorded when solving the task considered. A neural network is then taught to imitate this behavior and hopefully be capable of understanding the underlying thinking of the human it is imitating. By doing so the network should be able of making good decisions in situations similar to those it has been shown.

Imitation learning can be applied to driving a vehicle. This has been done mostly to predict the immediate actions which should be taken by the vehicle, such as decide the steering angle [7, 10, 11]. Outputting the immediate actions is however potentially dangerous as a neural network may output something incorrect. Further, the neural networks function as a black box solution, which means there is little to no understanding about why a certain prediction is made. For this thesis we instead want to predict the future path of the vehicle. A controller can then be influenced by the predicted path and together with other constraints decide what the actual steering commands. This would introduce an understanding behind the vehicle's behavior. Thereby, the autonomous driving functions less as black box compared to when the neural networks output the immediate actions.

1.2 Previous Work

In vision-based autonomous driving, the industry has different approaches to how this should be done. There are two approaches today which are common; mediated perception and behavior reflex. Currently, mediated perception is the most common approach. This approach first extracts information about the surrounding environment, e.g. bounding boxes of surrounding objects and lane marking estimations. The information is then used to operate the vehicle. Meanwhile, behavior reflex does not extract any specific information. Instead, this approach uses only the raw data to operate the vehicle.

The first approach, mediated perception, extracts information about the surrounding environment. This information can then be used to compute how the driving should be done. Using this approach, the control system is presented a set of extracted information, which is commonly helpful for driving and can improve the performance. However, the networks rely on the necessary information being extracted in an appropriate manner before being able to decide how to drive.

Mediate perception was used by a team from New York University and Net-Scale

Technologies to predict traversability for off-road driving [12]. They used a deep hierarchical network to detect and classify objects and thereby determine whether it was possible to drive there. They claimed that their classifier was able to see obstacle 5 to 100 meters away. More recently, it has also been shown that convolutional neural networks can be used to estimate the lane-markings and surrounding cars in a highway situation [13].

On the other side lies the behavior reflex approach. Here, the information about the surrounding environment is not extracted but instead has a direct mapping between the input and output. That is, the surrounding environment is represented by images which is the only available information for the neural network to perform the task.

The first big success in behavior reflex autonomous driving came in the form of ALVINN (Autonomous Land Vehicle In a Neural Network) in 1989 [14]. ALVINN is a 3-layer neural network architecture used to control NAVLAB, the Carnegie Mellon autonomous test vehicle. The network was trained on simulated road snapshots and it managed to predict the steering direction with 90 % accuracy in a real-world environment.

In 2016, NVIDIA presented their model PilotNet. This model uses a behavior reflex system to predict the steering angle of a vehicle and has shown success in this field [7]. A team from Peking University has had more recent success using only images to predict the steering angle with their model Deep Steering [11]. They claim their model beats other models on known benchmarks. Both PilotNet and Deep Steering were trained by imitation learning to choose the same steering angle as human driver did.

In 2017, a team from the University of California, Berkeley, also using imitation learning, created an end-to-end network given images from a video feed and information about previous motion using their fully convolutional network - long short-term memory (FCN-LSTM) network structure [15]. The network predicted the desired motion at the current time and therefore successfully combined raw images with past motion, where it predicted different discrete actions, i.e. steer left, right, forward, brake. The model was trained on the Berkeley DeepDrive dataset which consists of crowdsourced driving videos and IMU data. They evaluate two different ways to do motion, one where they have a set of specific actions and one where the actions are continuous.

There is an ongoing discussion whether mediated perception or behavior reflex should be used in vision-based autonomous driving. The approach by the Berkeley team uses a behavior reflex approach combined with information about past motion [15]. By doing so, some information may be highlighted and easier to access for the network instead of having to deduce the information by the images alone. This approach can be taken a step further by combining behavior reflex approach with more information about the surroundings.

Further, most studies to date have focused on what the autonomous vehicle should do at the current time. This means that the neural networks do not consider what future actions but instead consider what should happen now. By also looking into the future, it is possible that a different action is better to perform right now. In addition, by not predicting the steering angles and instead consider a path, it is possible to use a controller to compute the actual steering commands. This allows for a more transparent system and understanding for the vehicle's behavior.

1.3 Thesis Objective

The objective with this thesis is to further the planning from the neural networks. Instead of only considering what steering angle or velocity the vehicle should have now, the network should predict the path for an autonomous vehicle for the upcoming seconds. The neural network is provided a sequence of gray-scale images together with information about past ego motion, detections of surrounding objects and lane marking estimations during the last second. The aim is to investigate whether neural networks are capable of predicting the path 5 seconds into the future. An additional aim is to investigate whether merging different forms of input data will improve the performance, and thereby the reliability and safety of the autonomous vehicle.

The hope is that by merging both types of input data, the neural network will be able to use both sets of information. It would hopefully then, in situations where the information about the surrounding environment is lacking, use the image to compensate for the missing information in a manner similar to the behavior reflex approach.

The predicted path is given by points of coordinates. Each coordinate represents the predicted position of the vehicle after a fix time interval. These predicted coordinates can then in another future project be given to a controller which can use them to compute the desired steering angle and acceleration for the vehicle in the upcoming seconds.

1.3.1 Scope

The focus of this project lies in driving in rural Europe. Almost all available data has been collected in Europe. The network is not supposed to handle decision-making and thus the training data does not include situations where there were multiple paths. Thus, urban traffic is excluded as there often exists several valid paths there.

Another limitation is to only consider right-hand traffic. This limitation is done as what constitutes an appropriate path differs largely between left- and right-hand traffic.

1.4 Thesis Outline

The remainder of the thesis is outlined as follows. Initially, chapter 2 describes the data available, its source and how it was extracted. Afterwards, theory about how artificial neural networks work is presented in chapter 3. This includes the various types of network structures which are used in this thesis. The chapter also contains theory on how the training of a neural network works. Then, once the theory is presented, chapter 4 will describe the design of the evaluated models. Each model is described in general and also the architecture of their parts. The chapter ends with a description of the implementation and how they were trained. Chapter 5 then presents the result obtained in the thesis. These results are then discussed in chapter 6. It is also discussed what potential explanations may be for why the particular results were obtained. The chapter then ends with some potential future work which can be done in the area. Finally, chapter 7 presents the conclusions drawn in the thesis.

2

Data Source and Description

Good data is essential when implementing and training a neural network. In order to handle many types of scenarios it is important that the data is versatile and covers many different situations. This chapter covers the data used in this project.

2.1 Data Source

The data used in the project is real-world data collected by Volvo Cars during expeditions with the purpose of testing implemented functions. While on the expeditions the vehicles also recorded data which can be used for future development of new functions.

The stored data was collected from many sources. The data used in this project was mainly collected from a front-looking camera and on-board sensors. The front-looking camera was used to record videos of the road ahead and the on-board sensors were used to detect objects and road information. While moving, the vehicle also recorded its own movement in terms of yaw rate and how far it has moved in a time step.

2.2 Data Extraction and Synchronization

In the log files, the object detections and road information are recorded with the same frequency as the vehicle's movements. However, the camera used a different frequency, which was approximately 2.5 times slower. In order for the network to be trained properly, the data needed to be synchronized so that the inputs provided together were recorded simultaneously.

The extraction from the dataset was made so that the sample rate was 10 samples per second. For this, some reference times were used. The object detections and the videos were synchronized to the reference times. The data was synchronized by taking the sample with time stamp closest to that of reference time in consideration.

In some sequences the object detections and/or the videos could not be synchronized in a satisfactory manner and the sequence was then not considered in the project.

2.2.1 Image Extraction

The log files contained the video-feed from the camera. From this video-feed, images were extracted by sampling each frame to a 640×480 gray-scale image. Examples of what these images look like can be seen in figure 2.1.

Figure 2.1: Examples of images extracted from the front-looking camera's video feed.

2.2.2 Object Data Extraction

The object data used in this project includes information about detections of surrounding objects and estimates for the lane markings. In addition, the vehicle's movement is included in the object data.

At each given time, the 32 most relevant object detections were extracted and used. As there were not always 32 objects surrounding the vehicle not all of these were real detections. If there were fewer than 32 detections the list of detections was filled with zeros. Additionally, if a detected object was too small the detection's position was set to zero.

A similar filtering was done for the lane markings estimations. From the log files a third degree polynomial estimate was fitted and used. In some cases, one or more actual lane markings did not exist or were not detected. In these cases, the coefficient for that lane marking estimate was set to zero.

2.3 Data Formatting

While most of the data was not modified after extraction from the logs, some formatting had to be done. This section presents how the ground truth was computed given the vehicle's motion and how the data was pruned in order to avoid biased input.

2.3.1 Computation of Ground Truth Path

The data collected in the expeditions did not contain explicit information of the driven path. This information had to be computed using the yaw rate and the distance moved at each time step. While computing the ground truth path many variables had to be computed, these are visualized in figure 2.2 and explained in the following paragraphs.

The coordinate system was decided by the vehicle's position and direction at the current time (t_0). The time between two sequential observations was constantly 0.1 seconds and is here denoted Δt.

In order to compute the full path, the distance moved and direction at each time had to be known. The distance moved in the time interval $[t_{n-1}, t_n]$ was δ_n and the yaw rate during the same time period was ω_n. Using the yaw rate it was possible to compute the change in direction during the interval as

$$\Delta\theta_n = \omega_n \times \Delta t.$$

The direction at time t_n in comparison to time t_0 thus became

$$\theta_n = \sum_{i=1}^{n} \Delta\theta_i.$$

The next step was to compute the lateral and longitudinal movements made. First the movements made during each time step needed to be computed. During the time $[t_{n-1}, t_n]$ the lateral movement was

$$\Delta x_n = \delta_n \times \sin(\theta_n)$$

while the longitudinal movement was

$$\Delta y_n = \delta_n \times \cos(\theta_n).$$

The position at time t_n was computed by summing all movements made up to that point. Thus, the lateral position was

$$x_n = \sum_{i=1}^{n} \Delta x_i$$

while the longitudinal position was

$$y_n = \sum_{i=1}^{n} \Delta y_i.$$

Once the lateral and longitudinal positions were computed, the full path could be created. The position of the vehicle at time t_n was

$$p_n = (x_n, y_n).$$

The ground truth path was a vector $\mathbf{p} = [p_1, \ldots, p_N]$ where N was the desired number of positions. For this project $N = 50$ as the prediction were made 5 seconds into the future with a sample rate of 10 samples per second.

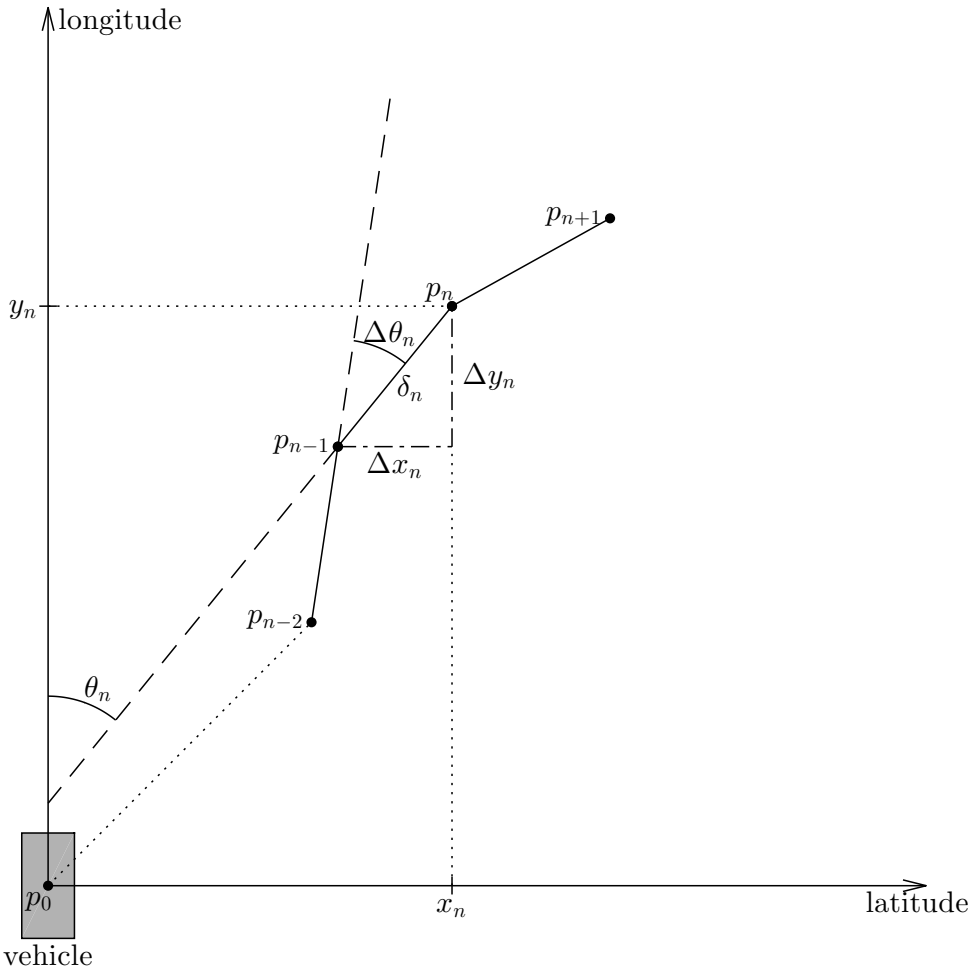


Figure 2.2: The image shows the measurements used in order to compute the ground truth path $\mathbf{p} = [p_1, \ldots, p_N]$. The coordinate system was determined by the vehicle's position and direction at t_0. During the time interval $[t_{n-1}, t_n]$ the vehicle moves a total distance of δ_n in the direction θ_n. At time t_0 the vehicle was moving purely in the longitudinal direction.

2.3.2 Data Pruning

The performance of a network largely depends on the quality and distribution of the data. If the data contains an uneven distribution the network will be biased towards the more common outputs. In order to alleviate this problem, the data was pruned so no type of output was overly dominant.

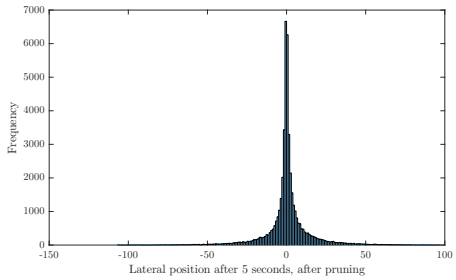
The first step of pruning was simply to remove overlapping data. Time series of 1 second were used as input to the network. Therefore, permitted sequences were created so the inputs had no overlap between samples. After that the ground truth target path was computed for each of the sequences. The lateral and longitudinal positions after 5 seconds were then extracted from the target path. The pruning

of data was then done with respect to the lateral and longitudinal positions after 5 seconds.

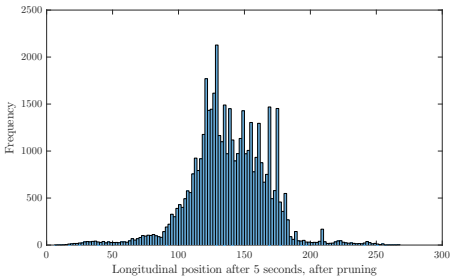
Figure 2.3a shows that most observations of the lateral position lie in the range $[-30, 30]$ meters. Similarly, figure 2.3b shows that the range $[90, 180]$ meters covers most observations of the longitudinal position, which corresponds to a velocity range of $[65, 130]$ kilometers/hour. Therefore, to avoid tails on the distribution of data, sequences with values outside these ranges were excluded when pruning.

The next step in the pruning process was to create bins for the ranges in question. One by one, the target path was computed for the sequences, and the lateral and longitudinal positions were put into their respective bin. If either of the bins for lateral or longitudinal position was full, the sequence was discarded, otherwise it was added to the list of permitted sequences. The cap for each bin was set to 2000 for the lateral position and to 800 for the longitudinal position.

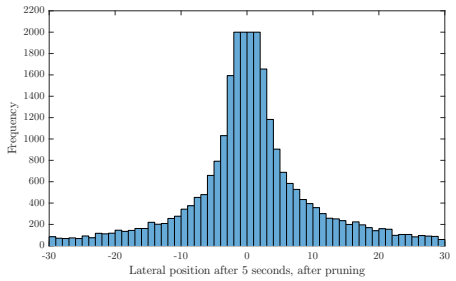
Figure 2.3 shows the histograms of the lateral and longitudinal positions after pruning, with the remaining data representing 7 hours of driving. From the histograms it is visible that the pruning reduced the domination of one bin for both the lateral and the longitudinal position. Thereby, the network should have lesser risk of being biased and always producing the same output.



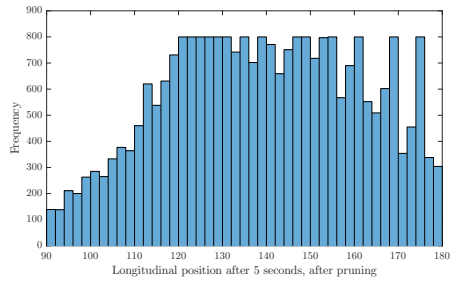
(a) Histogram of lateral position before pruning.



(b) Histogram of longitudinal position before pruning.



(c) Histogram of lateral position after pruning.



(d) Histogram of longitudinal position after pruning.

Figure 2.3: The figures show the histograms of the ground truth position's x and y coordinates after 5 seconds. Figures 2.3a and 2.3b are before pruning while 2.3c and 2.3d are after pruning.

3

Artificial Neural Networks

In 1943 Warren McCulloch and Walter Pitts released their work on how the nervous system could be modelled by using a mathematical model of a neuron [16]. With Frank Rosenblatt's work "The Perceptron" building on their work the concept of artificial neural networks was introduced [17]. However, neural networks had to wait for another 60 years until the computational power and availability of data made them useful. This chapter covers the types of networks used in this thesis along with some theory about how neural networks are trained.

3.1 Feedforward Neural Network (FFNN)

The feedforward neural network (FFNN) is the first and most basic neural network. Each neuron in the network is similar to the McCulloch-Pitts neuron as it receives and transmits a value and can be seen as a direct and acyclic graph. The neurons are separated into layers which are fully connected.

The fully connected layers get their name from the fact that a neuron gets input from all neurons in the previous layer. The value of a neuron is determined by computing a weighted sum of all neurons in the previous layer and also adding a bias. The sum is usually passed through an activation function in order to create non-linearity (see section 3.4). The value then becomes

$$z_i^{(n)} = f\left(\sum_{j=1}^{N_{n-1}} w_{i,j}^{(n)} z_j^{(n-1)} + b_i^{(n)}\right)$$

where $z_i^{(n)}$ is the value of neuron i in layer n, f is the activation function, $w_{i,j}^{(n)}$ is the weight between neuron j in layer $n-1$ and neuron i in layer n and $b_i^{(n)}$ is the bias of neuron i in layer n. This is computed for each neuron in layer n. Once the values are computed they are used to compute the values of the neurons in layer $n+1$.

The input to the network is typically a vector of values. This vector can be used as layer 0 when computing the values of the first layer. Finally, the values of the last layer are used as the output from the network.

Figure 3.1 shows a small FFNN with two fully connected layers. The network has two inputs, $z_1^{(n)}$ and $z_2^{(0)}$, which are used to compute the values of the next layer according to

$$\begin{aligned} z_1^{(1)} &= f\left(w_{1,1}^{(1)} z_1^{(0)} + w_{1,2}^{(1)} z_2^{(0)} + b_1^{(1)}\right) \\ z_2^{(1)} &= f\left(w_{2,1}^{(1)} z_1^{(0)} + w_{2,2}^{(1)} z_2^{(0)} + b_2^{(1)}\right) \\ z_3^{(1)} &= f\left(w_{3,1}^{(1)} z_1^{(0)} + w_{3,2}^{(1)} z_2^{(0)} + b_3^{(1)}\right). \end{aligned}$$

Once these have been computed, they are used to compute the values of the next layer which is also the output layer. The values of the output neurons is

$$\begin{aligned} z_1^{(2)} &= f\left(w_{1,1}^{(2)} z_1^{(1)} + w_{1,2}^{(2)} z_2^{(1)} + w_{1,3}^{(2)} z_3^{(1)} + b_1^{(2)}\right) \\ z_2^{(2)} &= f\left(w_{2,1}^{(2)} z_1^{(1)} + w_{2,2}^{(2)} z_2^{(1)} + w_{2,3}^{(2)} z_3^{(1)} + b_2^{(2)}\right). \end{aligned}$$

and thereby the full network has been passed through.

Commonly, these computations are computed in vector form. For this the values of the neurons in one layer are placed in vectors. For the small FFNN, the calculations then become

$$\begin{aligned} \mathbf{z}^{(1)} &= f\left(\mathbf{W}^{(1)} \mathbf{z}^{(0)} + \mathbf{b}^{(1)}\right) \\ \mathbf{z}^{(2)} &= f\left(\mathbf{W}^{(2)} \mathbf{z}^{(1)} + \mathbf{b}^{(2)}\right) \end{aligned}$$

where $\mathbf{W}^{(n)}$ is the weight matrix and $\mathbf{b}^{(n)}$ the bias vector of layer n. The output from the network is the vector $\mathbf{z}^{(2)}$.

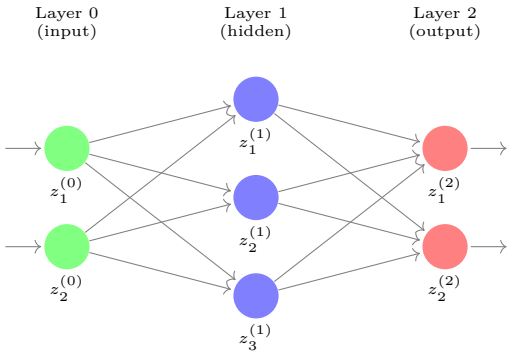


Figure 3.1: The image shows a small FFNN with two fully connected layers.

3.2 Convolutional Neural Network (CNN)

In 1989 Yann LeCun showed that the convolutional neural network (CNN) were able to recognize hand-written characters and thereby showed the potential of the convolutional neural network [18]. The architecture of the neural network consisted

of convolutional layers, pooling layers and non-linear activation functions, which later became the key building blocks of the convolutional neural networks. The most recent successes in image recognition and natural language processing have been done using convolutional neural networks [19, 20].

The success of the CNN comes from its structure. Unlike other neural networks, a CNN assumes its input is 3-dimensional and has a width, a height and a depth. A common input is an image, where the width and height are determined by the size of the image and the depth represents the number of channels in the images.

CNN usually use a combination of convolutional and pooling layers. These layers aim to reduce the size of the image in terms of width and height, with some exceptions such as done in GoogLeNet [21]. Usually the depth increase in the convolution layers as channels are added to the image. In doing so, the channels can find different features of the images.

Depending on the desired output, a CNN can have some fully connected layers after the convolutional and pooling layers. This is common when the desired output is a feature vector. In this case the image remaining after convolutional and pooling layers is flattened into a vector. The fully connected layers then work as described in section 3.1.

3.2.1 Convolutional Layer

The convolutional layers rely on local connectivity of the units, meaning that close-by pixels are most likely correlated. This means that the value of a neuron does not depend on the value of all neurons in the previous layers. Instead, it is only dependent on a local area, as seen in figure 3.2. The resulting value depends on the values of all the channels for the local area in the previous layer.

All neurons in one layer use the same weights when computing its value. Thus, the weights serve like a filter which slides over the entire layer trying to distinguish features. The purpose of this is to make the CNN space-invariant, as a feature may be anywhere in the layer.

The convolutional layers depend on some hyperparameters which influence its behavior, where the main ones are kernel size, stride and padding. The kernel size determines the size of the filter containing the weights. The stride is the step size with which the filter is moved as it slides over the image. Padding is done to increase the size of the image by adding values on either side of the layer. This is typically done for the image to remain the same size in terms of width and height.

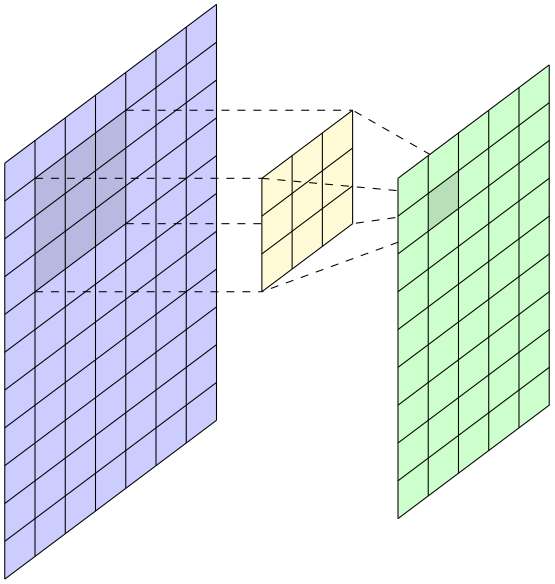


Figure 3.2: Example of a convolutional layer from the current layer (blue) to next (green). The filter (yellow) has 3×3 kernel size, 1×1 stride and no padding.

3.2.2 Pooling Layer

The pooling layers, like the convolutional layers, do not look at the complete previous layer when computing the values of a neuron. However, the pooling layers works on each channel separately.

A pooling layer pools together neurons in the previous layers into clusters. It then performs an action on the cluster to determine the value of the neuron. The most common is max pooling, which means that the largest value in the cluster becomes the value of the neuron. Figure 3.3 illustrates how a 2×2 max pooling layer with 2×2 stride works.

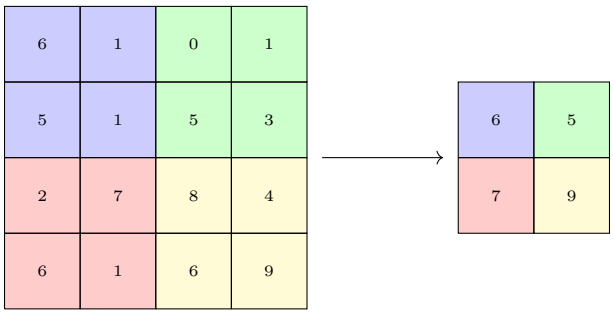


Figure 3.3: Illustration of how a 2×2 max pooling layer with 2×2 stride works.

3.3 Recurrent Neural Network (RNN)

In contrast to the FFNN and CNN which only consider the current input, the recurrent neural network (RNN) is capable of keeping a memory between inputs. Therefore, the RNN is capable of accessing temporal information within sequential inputs. This in turn makes the RNN suitable for tasks in which the input data is a time series.

A RNN can be seen as a unit which is passed through several times, with a hidden state kept in its memory between each passage. Figure 3.4 shows how a sequence of inputs is fed to a RNN unit. At each time point t the network is fed input $\mathbf{x}_t$ and it computes the hidden state $\mathbf{h}_t$ which functions as both the output at the given time and is kept as memory until time $t + 1$.

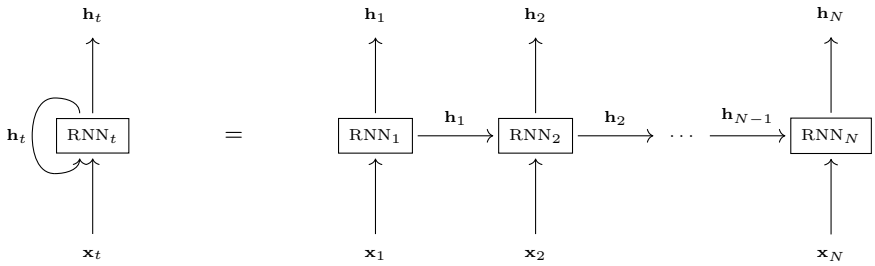


Figure 3.4: The image shows the input and output of a RNN unit over time. To the left a folded RNN is shown, in which it is clear that the hidden state is kept in the RNN. To the right it shows the RNN unfolded, where each time step is visualized as a separate unit which receives two inputs at each time $t \in 1, \ldots, N$. The inputs at time t are the actual input is $\mathbf{x}_t$ and the hidden state from the previous time h_{t-1}. The output is $\mathbf{h}_t$, and this is also kept in memory until the time $t + 1$.

The internal structure of a RNN can vary greatly. In a simple form the RNN unit, seen in figure 3.5, considers the hidden state from the previous step $\mathbf{h}_{t-1}$ as an input alongside the actual input $\mathbf{x}_t$. The two vectors are concatenated and and the new values of the hidden state are computed as

$$\mathbf{h}_t = \tanh\left(\mathbf{W}[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}\right)$$

where $[\cdot, \cdot]$ denotes vector concatenation and $\mathbf{W}$ and $\mathbf{b}$ are the weight matrix and bias vector respectively.

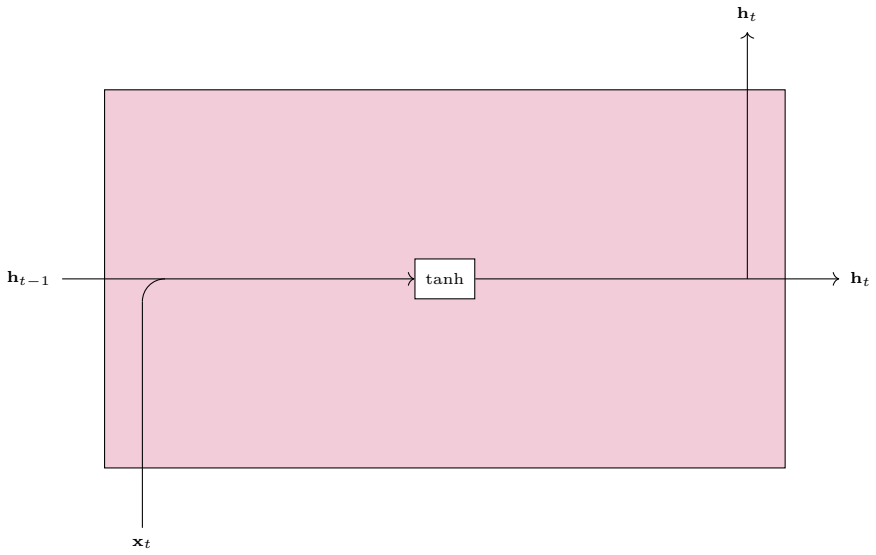


Figure 3.5: The image shows a schematic view of the process inside a simple RNN unit. The unit concatenates the current input with the hidden state from the previous time step. This is then passed through a fully connected layer with hyperbolic tangent activation.

3.3.1 Long Short-Term Memory (LSTM)

The most successful RNN is called long short-term memory (LSTM), introduced in 1997 by Hochreiter and Schmidhuber [22]. It has found success in e.g. natural language processing and speech recognition. Figure 3.6 shows a schematic view of an LSTM unit.

In addition to the hidden state $\mathbf{h}_t$ of the RNN, the LSTM unit keeps the cell state $\mathbf{c}_t$ in memory at time t. Updates to the states are done in each time step, and these updates are controlled by four gates in the unit; the forget gate (f), the input gate (i), the modulation gate (g) and the output gate (o). Each of the gates has a weight matrix $\mathbf{W}_n$ and a bias vector $\mathbf{b}_n$ for $n \in \{f, i, g, o\}$.

When an input vector $\mathbf{x}_t$ is given to an LSTM unit it is immediately concatenated to the hidden state vector from the previous time step $\mathbf{h}_{t-1}$. The first gate it then encounters is the forget gate which controls how much should be forgotten from the previous cell state and is computed as

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f\right)$$

where $[\cdot, \cdot]$ again represents vector concatenation and σ is the sigmoid function.

In order to update the cell state, the LSTM needs to decide which new information to add. This is done in two steps. The first step is to calculate which states to update. This is done by the input state which also guards against noisy input data

which will hurt the cell state. The computations are given by

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right).$$

The second step is computing potential candidates for new values to be given to the states which are being updated. These values are given by

$$\mathbf{g}_t = \tanh\left(\mathbf{W}_g[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_g\right).$$

After finding which states should be updated and by how much, the cell state is ready to be updated according to

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathbf{g}_t$$

where $\circ$ represent the Hadamard product.

With the cell state updated, an output from the LSTM can be computed. This is done with the output gate which determines how much of each cell state should be used. The output gate is computed as

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right)$$

and is used to update the hidden state by

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh\left(\mathbf{c}_t\right).$$

Thereby the cell and hidden states have been fully updated and the hidden state is provided as output.

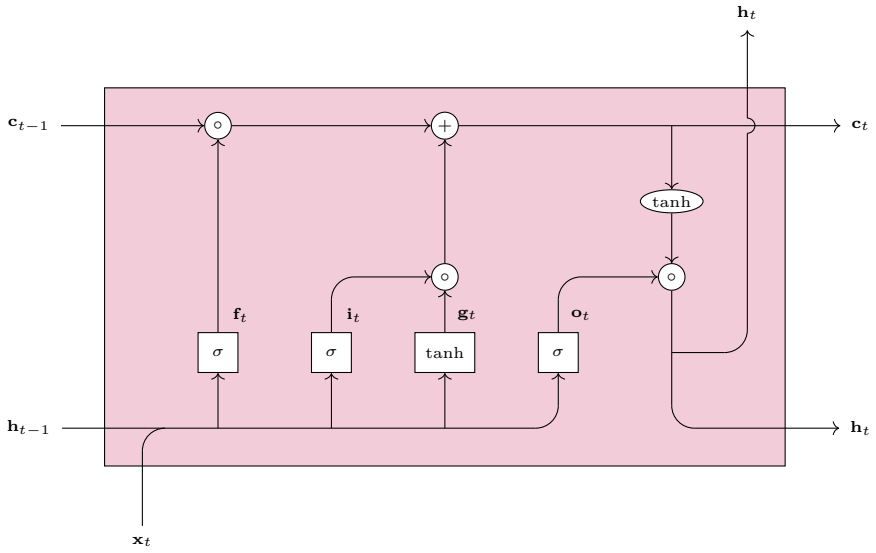


Figure 3.6: The image shows a schematic view of an LSTM unit. The rectangles represent gates in the units which pass the input through a fully connected layer with either sigmoid or hyperbolic tangent activation. The circles signify operations between two vectors, $\circ$ for the Hadamard product and $+$ for addition. Ellipse represent that a function is applied to a vector.

3.4 Activation Function

Activation functions are used in order to introduce a non-linear dependency between the input and output of the network. In theory any function can be used as an activation function, but there are some functions which are commonly used. These functions are listed in equations (3.1)-(3.6) and their respective graphs are shown in figure 3.7.

$$\text{Identity:} \qquad f(x) = x \tag{3.1}$$

$$\text{Binary step:} \qquad f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{3.2}$$

$$\text{Sigmoid:} \qquad f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

$$\text{Hyperbolic tangent:} \qquad f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.4}$$

$$\text{Rectified linear unit:} \qquad f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{3.5}$$

$$\text{Leaky rectified linear unit:} \qquad f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{3.6}$$

The binary step is one of the oldest activation functions and it decides whether a signal is on (1) or off (0). This function has derivative zero at all places where it is differentiable. Therefore, this function will always give zero-gradients and is not suitable to use in combination with optimization methods which use the gradients.

The sigmoid function (also known as logistic function) is a smoother version of the binary step. It has continuous values in the range $[0, 1]$ and is differentiable at all points. Historically, this function has been used widely in neural networks along with the hyperbolic tangent function. The hyperbolic tangent function is similar to the sigmoid in shape but has a larger range which is centered in 0. This makes it similar to the identity function for inputs close to 0. However, the sigmoid and hyperbolic tangent become saturated for large values.

The last two functions, rectified linear unit (ReLU) [23] and leaky ReLU do not saturate for large positive values. This allows for large activations and thereby a sensitivity towards large inputs. Leaky ReLU has a parameter $\alpha < 1$ which is a small value to allow for the output to not become zero, but still have a small value. Currently, ReLU is the most commonly used activation function in deep neural networks. However, other linear units such as leaky ReLU are becoming more popular due to their non-zero derivative for negative inputs.

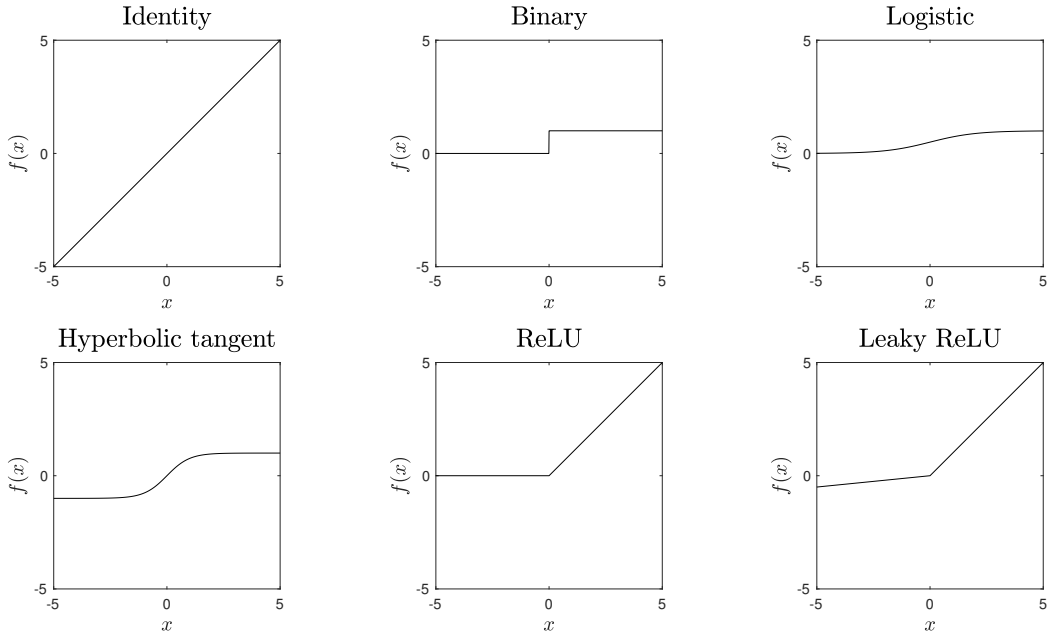


Figure 3.7: The figure shows the graph for some of the common activation functions. For leaky ReLU the parameter in equation (3.6) is set to $\alpha = 0.1$ for visualization purposes.

3.5 Training a Neural Network

In order to train a neural network there is a need for a method in which a network is taught to execute a task. This is usually done by letting the network compute the value of the function it represents and then providing feedback. The network can then use the received feedback in order to change the parameters so that its output improves.

There are two common types of learning; supervised and unsupervised learning. The main difference between these forms of learning is that in the former the desired output of the network is known. This means that the feedback provided to the network is a form of error determining how much the output of the network differs from the target. On the other hand, when performing unsupervised learning there is no definitive desired output. The feedback can instead consist of how well the density of the inputs is captured or some other measurement which reflects back on the input data.

3.5.1 Supervised Learning

As supervised learning has a target for each input it gives the possibility to compute a loss of how much the computed output differs from the target. Given an input x the network computes the output $\hat{y} = f(x|\theta)$ where θ are the network's parameters. The loss is computed with a loss function $L(y, \hat{y})$ which uses the target output y and the actual output $\hat{y}$. Depending on the problem at hand, different loss functions are used. Common loss functions are mean square error for regression type problems and cross entropy for classification type problems.

Imitation learning is a form of learning where the neural networks are trained to imitate the behavior of humans. For supervised imitations learning this means that a human has performed the task which the network is supposed to learn. The human's solution is then used as the target output y while training the network.

When using supervised learning it is common to use backpropagation to update the parameters of the network. This is an algorithm which works iteratively to minimize the loss. The basic steps of the algorithm are:

1. Compute "forward pass" of the network, i.e. compute the network's output $\hat{y}$.
2. Compute gradients $\nabla_\theta L(y, \hat{y})$ with respect to parameters θ.
3. Update parameters with an optimization algorithm. A commonly used optimization method is gradient descent which updates the parameters according to $\theta \leftarrow \theta - \eta \nabla_\theta L(y, \hat{y})$, where η is the learning rate.
4. Has the network converged?
   Yes: Stop algorithm
   No: Repeat from step 1.

3.5.2 Optimization Methods

A common optimization method used to minimize the loss is to use gradient descent. The method works iteratively and the idea is to at each iteration take a step in the direction in which minimizes the loss function the greatest. This is done by computing the gradient of the loss with respect to each parameter and taking a step in the opposite direction.

Stochastic gradient descent (SGD) is a slightly modified version of gradient descent. The difference comes in when the gradients are computed. In gradient descent the function is computed for all the input data in the training set. In SGD the gradients are computed after each mini-batch. The purpose for this is that gradient descent easily finds a local minimum and has difficulties improving further. By computing the gradients after each mini-batch the specific mini-batch may have gradients which differ from the overall and thereby leave the local minimum and hopefully find a better solution.

A method to further improve the gradient descent is to use SGD with momentum. By doing so, the update of parameters does not only depend on the gradient of the current values. Instead it also remembers the step taken in the previous iteration. This leads to a modified update rule $\theta \leftarrow \theta - \eta \nabla_\theta L(y, \hat{y}) + \beta \Delta\theta$ where $\Delta\theta$ is the update in the previous iteration and β a scalar forgetting factor.

A more recent optimization method which is also an offspring of gradient descent is adapted moment estimation (Adam) [24]. Adam uses both running averages of the gradients and second moments of the gradient. Adam updates the parameters according to

$$
\begin{aligned}
m_\theta &\leftarrow \beta_1 m_\theta + (1 - \beta_1) \nabla_\theta L(y, \hat{y}) \\
v_\theta &\leftarrow \beta_2 v_\theta + (1 - \beta_2) \left(\nabla_\theta L(y, \hat{y})\right)^2 \\
\hat{m}_\theta &= \frac{m_\theta}{1 - \beta_1} \\
\hat{v}_\theta &= \frac{v_\theta}{1 - \beta_2} \\
\theta &\leftarrow \theta - \eta \frac{\hat{m}_\theta}{\sqrt{\hat{v}_\theta} + \varepsilon}
\end{aligned}
$$

where β_1 and β_2 are scalar forgetting factors and ε is a small number used to avoid diving by 0.

3.5.3 Regularization

In order to have a robust model it is not enough to only have a low training error on the training set, but also on the validation and test set. A model, given enough parameters, can in theory gain near-perfect loss on the training set, but still not learn the underlying structure of the data. Therefore, a model has to be able to generalize the behavior of the dataset, instead of only fitting the unimportant noise in the training set. The purpose of regularization is to use techniques which lower the validation loss, but not the training loss. Two commonly used techniques are early stopping and dropout.

Early stopping is one of the most fundamental regularization methods used when training a neural network [25]. The idea is to stop training the network as soon as the validation loss stops decreasing. This means that the network has stopped generalizing and started overfitting to the noise in the training set.

Dropout is a regularization technique which randomly disconnects neurons (their value is set to zero) from a computation graph during training [26]. By forcing a network to not completely rely on a specific set of weights to produce the wanted output, the network learns to use more of its weights to produce the output. This has been shown to increase the robustness of neural networks and has increased the accuracy on difficult tasks such as image classification [19, 27]. Dropout has the added benefit of being very computationally cheap and can be applied to most

networks and problems [28]. However, since all the weights are not used during training the overall model size and number of training iterations may need to be increased to compensate [28].

4

Model Designs

This chapter presents the main models used to predict the path. A total of four models are presented; a CNN which uses only images, an LSTM which uses only object data and two different models which merge images and object data. Finally, the chapter presents how the models were implemented and trained.

4.1 CNN Model

The first model is a CNN model which uses a behavior reflex approach, using images as the only input. The presented model has two similar versions, with the difference lying in the input data. One version uses 1 image and the other version uses 10 images. When 10 images are used, there is some temporal information within the data as all the images are taken within a fixed time-frame.

4.1.1 Image Preprocessing

The image extraction described in section 2.2.1 provides images of size 640×480. In order to reduce computation time while running the model, the images were cropped and down-sampled. Initially the images are cropped by 168 rows at the top and 72 rows at the bottom, as the top rows shows the sky and the bottom rows shows the vehicle's hood. These parts are deemed unnecessary for the network to process as they are seldom beneficial for driving. After the cropping a 640×240 image remains. This is further down-sampled to a 182×68 image in order to reduce computation time.

After cropping and down-sampling the image, per image normalization is performed. This is done by subtracting the mean of the entire image from each pixel value and then dividing the remaining value by the image's standard deviation. The purpose of the normalization is to make the image input less dependent on the brightness of the surrounding environment. This is necessary as the data was collected both daytime and nighttime.

The version of the model which uses only one image uses a 3-dimensional tensor with dimensions $1 \times 68 \times 182$ as input. For the version using 10 images, the input to the network is a sequence of the 10 images. This is done by stacking the images from the sequence by using each individual image as a channel of the input. Thereby the input in this version is given as a 3-dimensional tensor with dimensions $10 \times 68 \times 182$.

4.1.2 CNN Architecture

This model is given an input which consist of either 1 or 10 images, preprocessed as described in section 4.1.1. The CNN has five convolutional layers and three max pooling layers, which are illustrated in figure 4.1. All the pooling layers have 2×2 kernel size and 2×2 stride. The first convolutional layer has 5×5 kernel size while the subsequent convolutional layers have 3×3 kernel size. All convolutional layer use 1×1 stride and are followed by leaky ReLU activation.

The depth of the first convolutional layer is usually used for RGB of a single image, but can have any size. In the version which uses stacked images, the time aspect is used as each image represents a channel of the input.

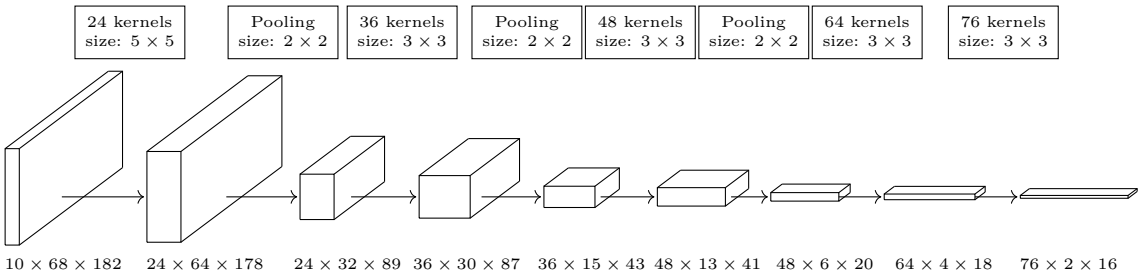


Figure 4.1: The image shows the convolutional and pooling layers used in the CNN with 10 images as input. The convolutional layer all have 1×1 stride and are followed by leaky ReLU activation. All pooling layers have 2×2 stride and perform perform max pooling. The version of the network which uses only image instead has input with dimensions $1 \times 68 \times 182$.

Finally, the feature remaining after the convolutional and pooling layers is flattened into a vector. This vector then goes through three fully connected layers which aim to format the output. Each of the layers decrease the dimensionality of the vector. All layers are followed by leaky ReLU activation, except the last which is followed by identity activation.

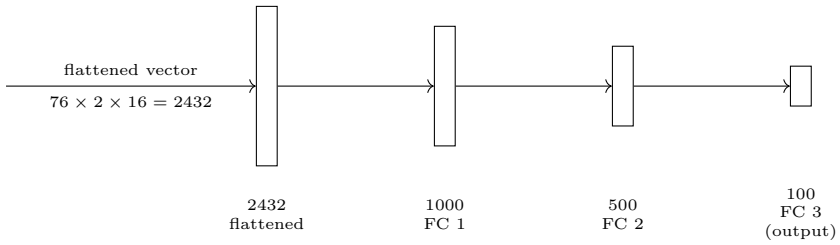


Figure 4.2: The image shows the structure of the FFNN in figure 4.4. All layers use leaky ReLU activation, except the last which uses identity activation. The values below each layer list how many neurons are in each layer.

4.2 LSTM Model

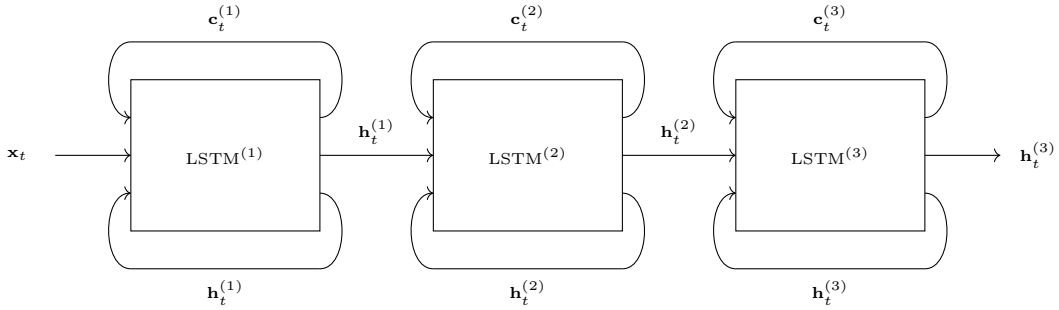
The second model is an LSTM network which uses only the object data as input. The object data is provided sequentially to the LSTM in order to make use of the temporal information.

4.2.1 Object Data Preprocessing

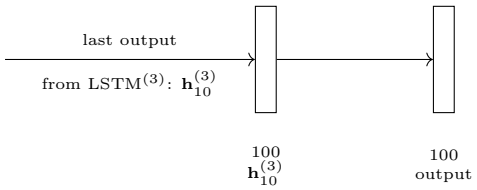
The object data consists of the object detections, the lane estimations and the vehicle's movements. At each time step these are placed into a vector containing the lateral and longitudinal positions of the detections, the coefficient for the lane estimations, the distance the vehicle has moved during the time step and the yaw rate during the time step. This vector then contains 82 elements and the input then consists of a sequence of 10 vectors.

4.2.2 LSTM Architecture

The architecture of the LSTM is visualized in figure 4.3. The network uses a sequence of object data as described in section 4.2.1. The network has three LSTM units after one another, each functioning as a layer and containing a cell state and hidden state of size 100. In order to format the output from the model the final output from the LSTM layers is passed through a fully connected layer with identity activation.



(a) The image shows how information is passed between the three LSTM units used at each time point t.



(b) The output from the third LSTM unit for the tenth and final time step is passed through a fully connected layer. The values below each layer list how many neurons are in each layer.

Figure 4.3: The two subfigures show a schematic view of the LSTM network. The input passes through three LSTM units for all samples in the sequence, which is visualized in figure 4.3a. Once all have gone through the the units, the final hidden state of LSTM$^{(3)}$ is passed through a fully connected layer with identity activation to format the output and allow for a non-saturated output, seen in figure 4.3b.

4.3 Merging Model 1

The third model uses both the images and the object data. In order to do this the model is constructed using a combination of networks. The resulting model has three inner networks as shown in figure 4.4.

The inner networks allow for the images and the object data to be processed separately in a manner suitable to the data type. The images are processed by the CNN described in section 4.1, the version which stacks 10 images. Meanwhile, the object data is processed by an LSTM of the type described in section 4.2.

Both the CNN and the LSTM output a vector each. The vectors are then concatenated and passed to a FFNN. The FFNN consists of four fully connected layers as seen in figure 4.5. The first layer increases the dimensionality of the data. The subsequent layers then decrease the dimensionality of the data until the last layer which produces the output vector with 100 elements. All layers except the last are followed by leaky ReLU activation, while the last uses identity activation. The output from the FFNN is then also the output from this model.

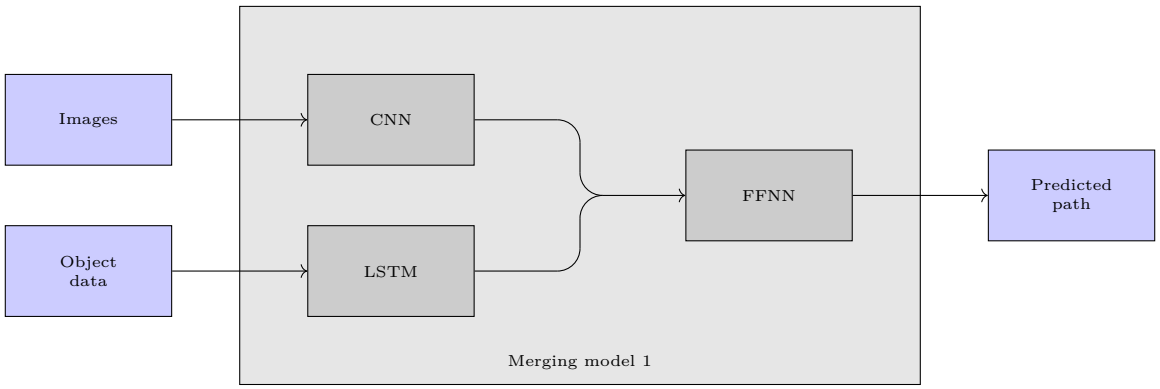


Figure 4.4: The image shows a schematic view of the neural network, which has three inner networks. The CNN processes the images while the LSTM processes the object data. Afterwards the outputs from these networks are concatenated and passed to the FFNN.

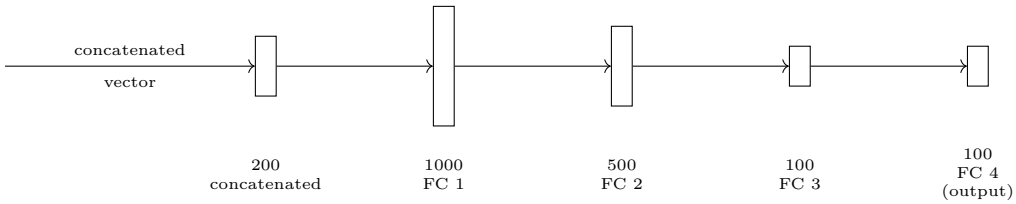


Figure 4.5: The image shows the structure of the FFNN in figure 4.4. All layers use ReLU activation, except the last which uses identity activation. The values below each layer list how many neurons are in each layer.

4.4 Merging Model 2

The fourth model also uses both the images and the object data. The structure of the model is similar to the model implemented at Berkeley for predicting steering angles [15]. The model first lets each image pass through a CNN to extract a feature vector. These feature vectors are then merged with the object data and passed to the LSTM. A schematic view of this model can be seen in figure 4.6.

The images are processed by a CNN which uses 1 image as input, as described in section 4.1. This provides a feature vector for each of the 10 images. Each of the features vectors is then concatenated with the corresponding object data, which has been preprocessed as described in section 4.2.1. This creates a time series of 10 vectors, each with 182 elements.

The concatenated vectors are then given to an LSTM of the type described in section 4.2, but the input vectors have 182 elements. The output from the LSTM is also the output from this model.

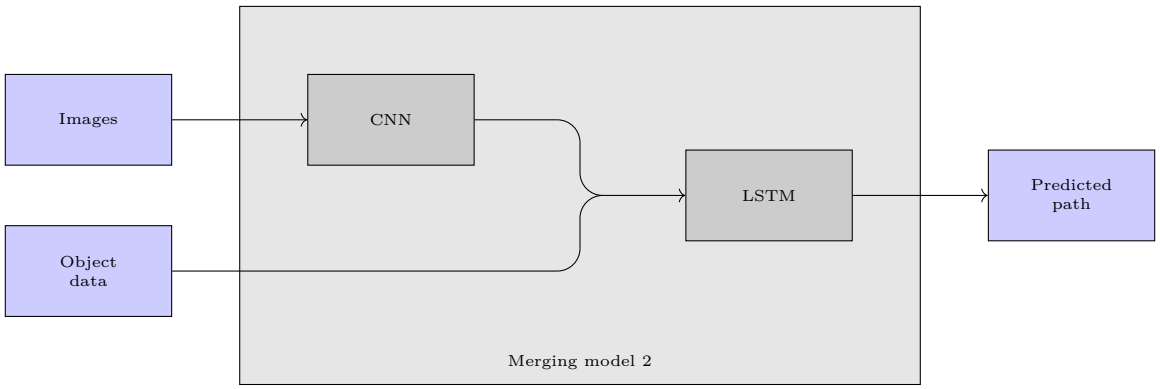


Figure 4.6: The image shows a schematic view of merging model 2, which has two inner networks. The CNN processes each individual image and outputs a feature vector. The feature vectors are then concatenated with the corresponding object data and passed to the LSTM as a time series.

4.5 Output Format

The models are trained to predict 5 seconds into the future. With a sample rate of 10 samples per seconds this means that the path consists of 50 future positions. Each position is 2-dimensional and therefore the models must output 100 values to account for the lateral and longitudinal positions. This is the reason the models output vectors with 100 elements. These were then grouped into points such that every two output elements corresponds to a 2-dimensional position.

4.6 Implementation

The models were constructed, trained and evaluated using PyTorch [29], which is a machine learning library developed mainly by Facebook. While training, Adam was used as the optimization algorithm with the default parameters.

The CNN (with both 1 and 10 images as input) and the LSTM models were trained first. These models did not require any other models while training. While training merging model 1, pretrained CNN and LSTM models were used, which means that the output vectors from these two networks were also predicted paths. Merging model 2 used a pretrained CNN model while training. Thereby, the feature vector from the CNN was a predicted path.

While training the models, early stopping and dropout were used. Early stopping was used by interrupting the training once the validation loss had stopped improving. Dropout was used in the training of all models. For the CNN, dropout was used on the second and third fully connected layers. In the LSTM, dropout was applied

to the output from the first two LSTM units. Merging model 1 had dropout in its FFNN part, on the first and second fully connected layers. Merging model 2 had dropout on its LSTM part in the same manner as the LSTM model. For all models, dropout probability was 0.5, which is the common dropout probability for linear layers [26].

4.6.1 Normalization of Paths

While driving, the vehicle moved further longitudinally than laterally, as could be seen in section 2.3.2. In addition, the position after 5 seconds is likely to have larger values than the position after 1 second. Due to this, each output value will have a different range in which its potential values lie. This difference in ranges will cause the data to be scaled badly and thereby be unbalanced.

To balance the data when computing the loss, the path had to be normalized. This was done for each position c_i individually, with $c \in \{x, y\}$, $i \in \{1, \ldots, 50\}$. The mean and standard deviation of c_i, μ_{c_i} and σ_{c_i} respectively, were computed over the full training set. Then all the values of the ground truth path were normalized as

$$c_i' = \frac{c_i - \mu_{c_i}}{\sigma_{c_i}}.$$

Similarly, the predicted path also had to be normalized. This was done using the mean and standard deviation computed for the ground truth paths. Thus, a predicted value $\hat{c}_i$ is normalized to

$$\hat{c}_i' = \frac{\hat{c}_i - \mu_{c_i}}{\sigma_{c_i}}.$$

4.6.2 Loss Function

While training the network, a mean square error was used on the normalized values. The loss is thus defined as

$$L = \frac{1}{100}\left(\sum_{i=1}^{50}(\hat{x}_i' - x_i')^2 + \sum_{i=1}^{50}(\hat{y}_i' - y_i')^2\right)$$

where (x_i', y_i') is the normalized true position at time t_i and $(\hat{x}_i', \hat{y}_i')$ is the normalized predicted position at the same time.

5

Results

This chapter covers the evaluation of the different model. The models are evaluated in a few different manners. First, the distribution of errors which occurred on the test set. Second, the predicted paths for some scenarios are shown alongside the target path. Third, predictions and target through one whole video are shown. Last, the models are compared to an interpolation which can be used to predict the path.

5.1 Error Distribution

This section shows box plots for how the errors on the test set are distributed for the models. A box represents the values between the first quartile q_1 and the third quartile q_3 while the line inside represents the median. The whiskers above and below the box are $q_1 - 1.5(q_3 - q_1)$ and $q_3 + 1.5(q_3 - q_1)$ long respectively and corresponds to the 99.3 percentile if the data were normal distributed. Box plots are shown first for the CNN model and then for LSTM model. Finally, the box plots of the CNN and the LSTM are compared to the box plots of the two merging networks are shown.

5.1.1 CNN Model

Figure 5.1 shows the distribution of the errors for the CNN with 1 image and the CNN with 10 images. The figure shows the errors as they are after 1, 2, 3, 4 and 5 seconds for the three models. It is visible from the plot that the CNN model with 10 images as input performs best of these models.

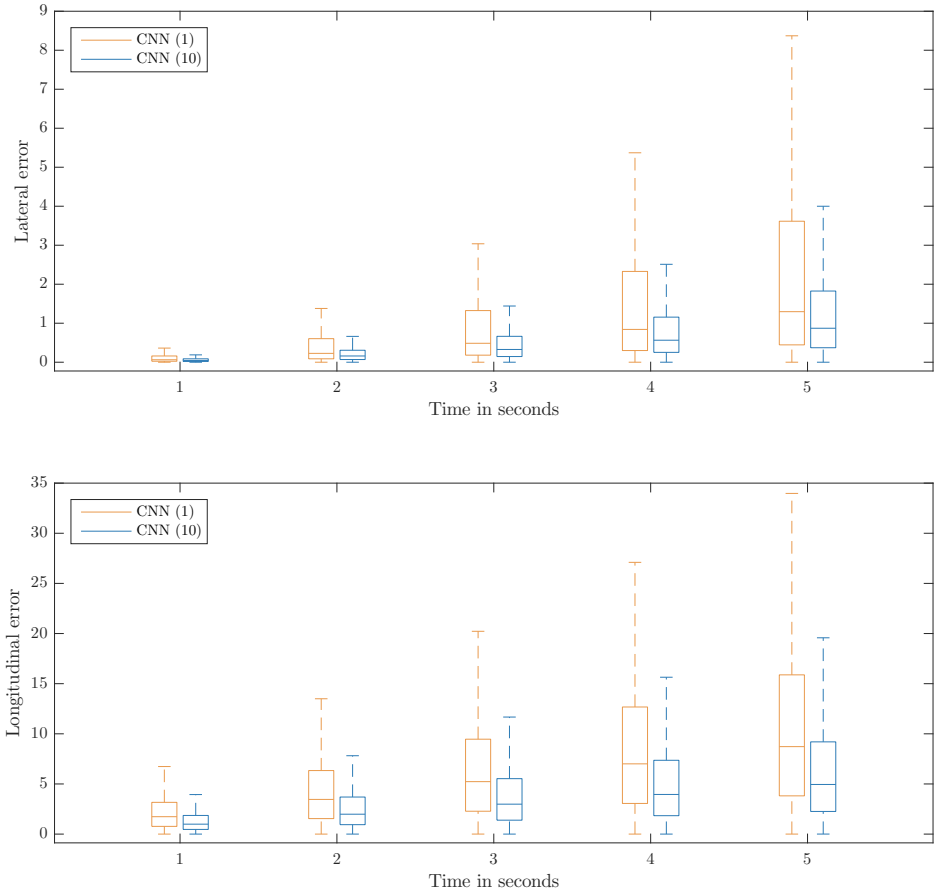


Figure 5.1: The box plots contain information about how the distributions of the absolute error for the lateral and longitudinal position over time. The plots represent the models which only use images as input; CNN with 1 image and CNN with 10 images.

5.1.2 LSTM Model

The LSTM model was tested in three different ways. The first is when the input data works just as it should, the second is when lane marking estimates are missing and the final manner is when object detections are missing. Figure 5.2 shows the distribution of errors for these three cases after 1, 2, 3, 4 and 5 seconds. The plots show that excluding lane marking estimates is harmful while excluding object detections does not make a large difference.

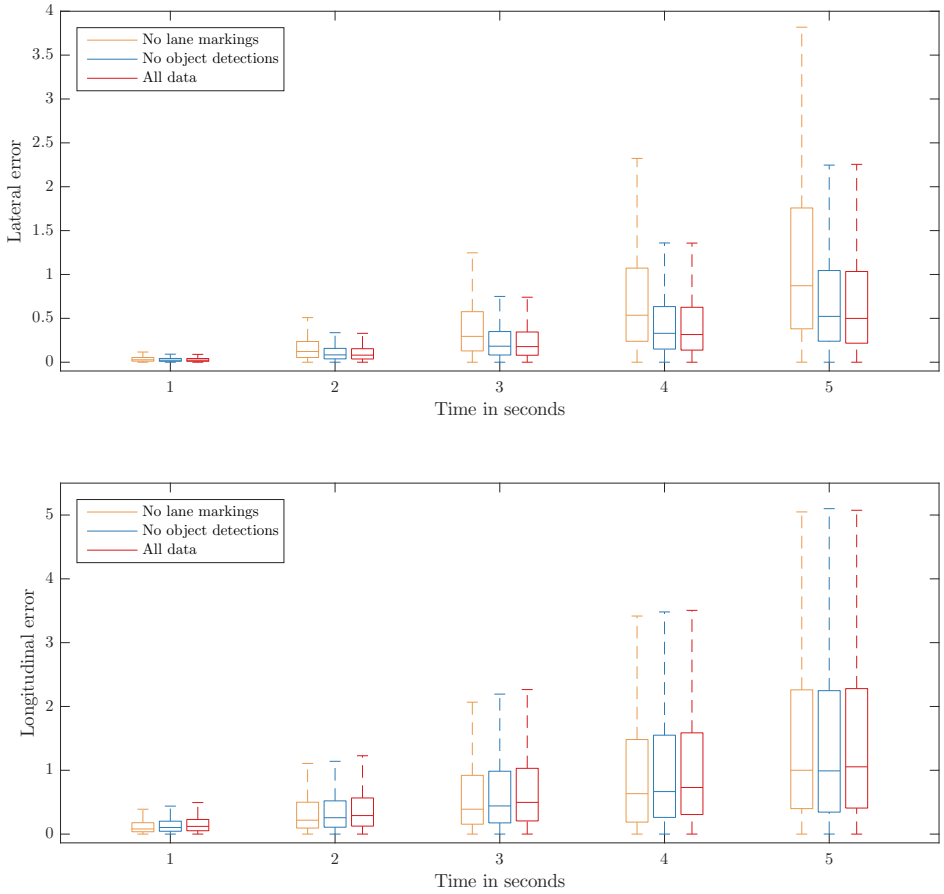


Figure 5.2: The box plots contain information about how the distributions of the absolute error for the lateral and longitudinal position over time. The plots all represent the LSTM model tested in three different manners; with all data, with lane marking estimates missing and with object detections missing.

5.1.3 All Models

Figure 5.3 shows the distribution of the errors for the best CNN model and the remaining models; the LSTM, merging model 1 and merging model 2. Again, the errors are shown as they are after 1, 2, 3, 4 and 5 seconds for these models. It is visible from the plot that the errors are larger for the CNN, while the other models have similar error distributions.

Figure 5.4 shows how the errors build throughout the predicted path. The box plots shown at 2 seconds display the difference between the error after 1 second and the error after 2 seconds, and similarly for the box plots after 3, 4 and 5 seconds. The corresponding measure after 1 second is equal to the error after 1 second.

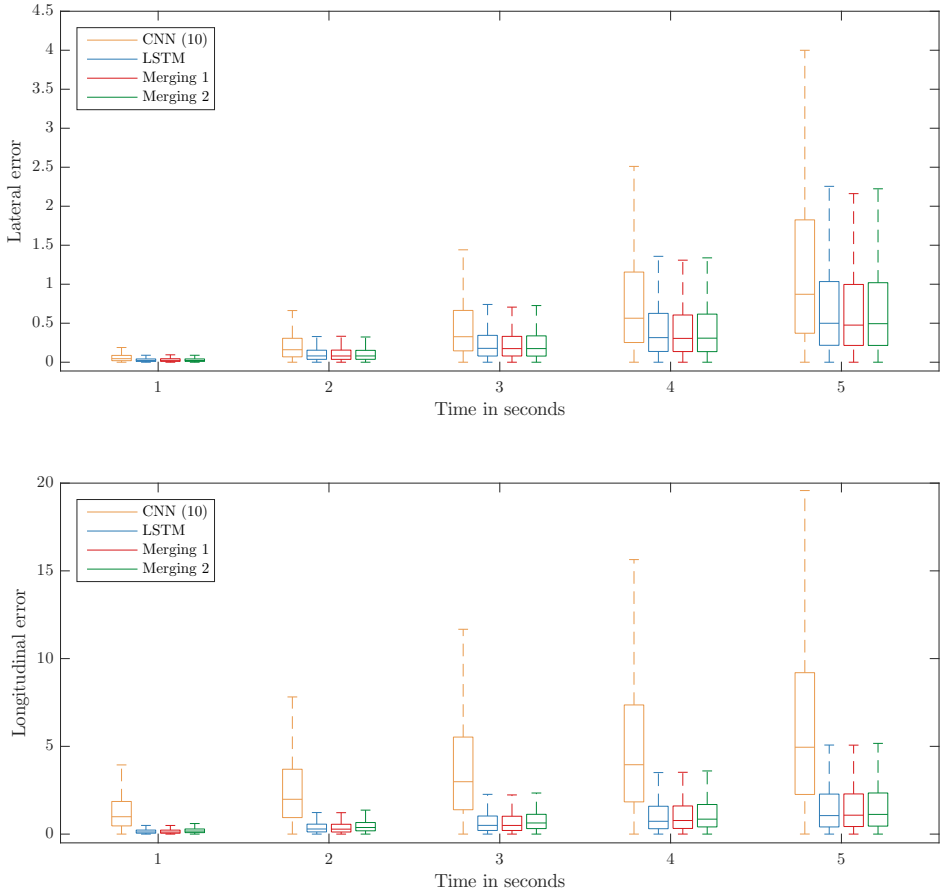


Figure 5.3: The box plots contain information about how the distributions of the error for the lateral and longitudinal position over time. The network shown in the box plots are the CNN, the LSTM and the full network.

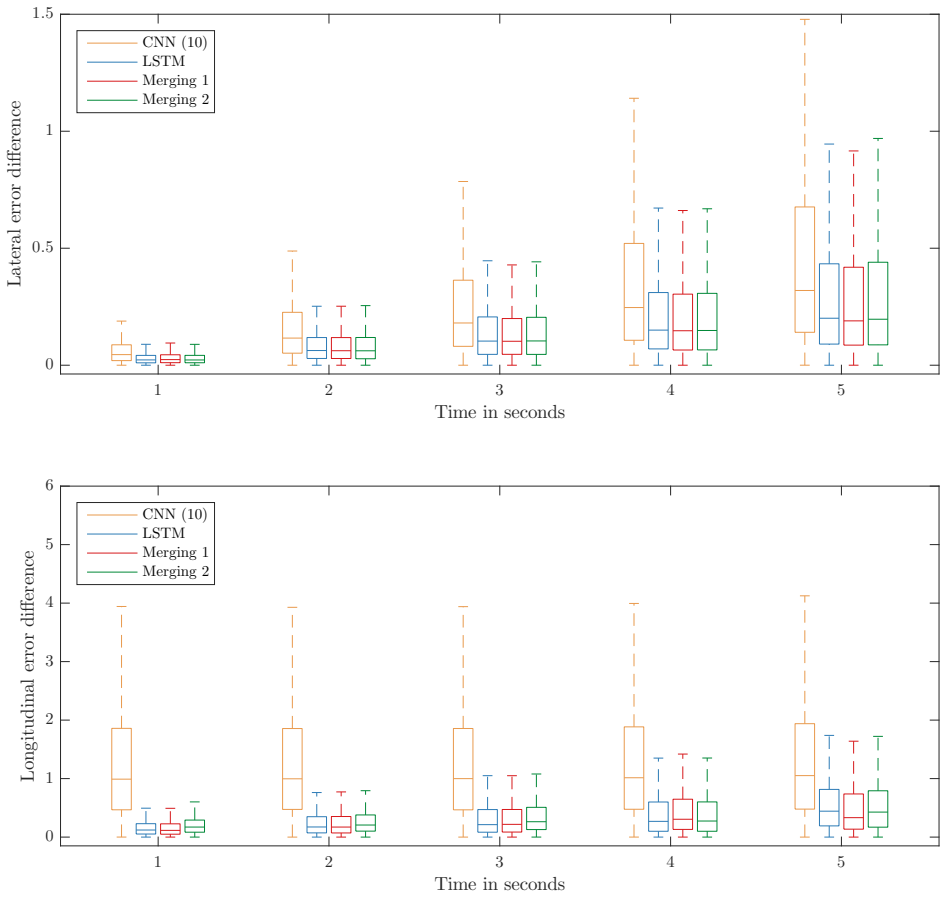


Figure 5.4: The box plots contain information about how the distributions of the error difference for the lateral and longitudinal position over time. The network shown in the box plots are the CNN, the LSTM and the full network.

5.2 Example Paths

A different method for evaluating the performance of the models was to plot the predicted paths alongside the target path. Figures 5.5-5.7 shows examples of what this may look like for the CNN with 10 images, the LSTM. The image shows the input from the camera to the left, the detected objects and third degree polynomial lane marking estimates in the middle and lastly the predictions alongside the target to the right.

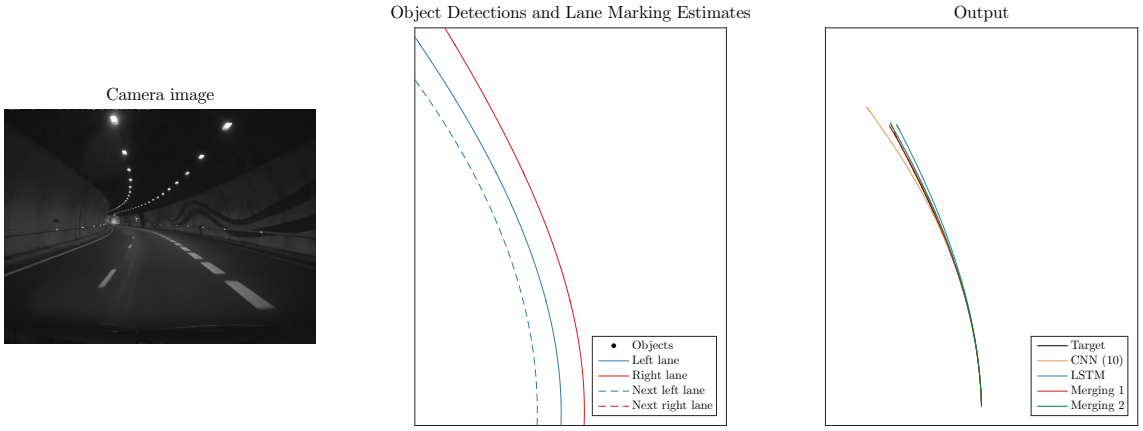


Figure 5.5: Examples of input and output from the network. The sample was recorded in a tunnel and the road appears to be turning left, with no surrounding objects. The predicted paths are similar with the exception of the CNN which performs the worst. The best prediction is the merging model 1 which manages to catch the curvature of the path but has misjudged the speed slightly.

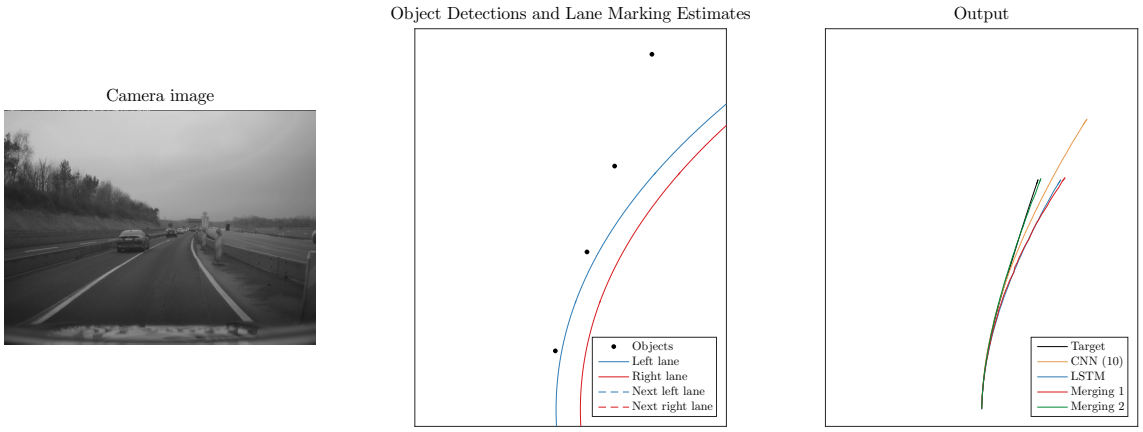


Figure 5.6: Examples of input and output from the network. The sample was recorded daytime and the road appears to be turning slightly right. In the middle image the estimated lane markings differ in comparison to how the detected objects are aligned. The predictions vary a lot in this case, with the best prediction in done by merging model 2 which manages to follow the ground truth. The CNN misjudges the path a lot but has figured it should be a right turn. The LSTM and merging model 1 have very similar paths and has estimated the longitudinal position good, but overestimated the lateral position.

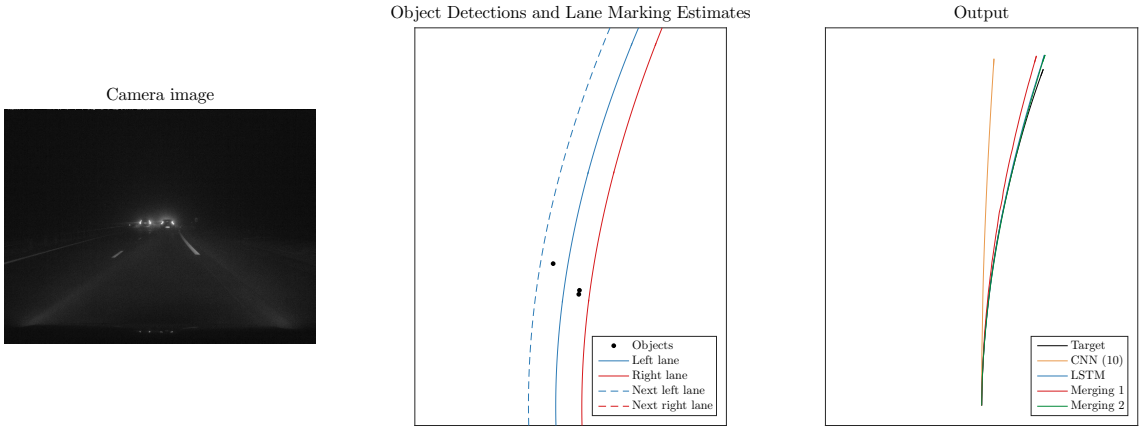


Figure 5.7: Examples of input and output from the network. The sample was recorded nighttime and the road appears to be turning right, but is hard to estimate from the image alone. The predictions from all models deviate from the ground truth, with the best prediction done by the LSTM and the merging model 2, which coincides. The CNN misjudges the scene by a lot and the merging model 1 finds a right curve but is not accurate to the ground truth.

5.3 Consecutive Sequences

Figure 5.8 shows the target and predictions in terms of lateral and longitudinal position after 5 seconds for 77 consecutive sequences. The input and output for sequence 8 in the figure are shown in figure 5.7. These sequences are among the toughest to estimate in the test set, as they vary much in terms of lateral and longitudinal position. Additionally, the consecutive sequences are recorded during the night.

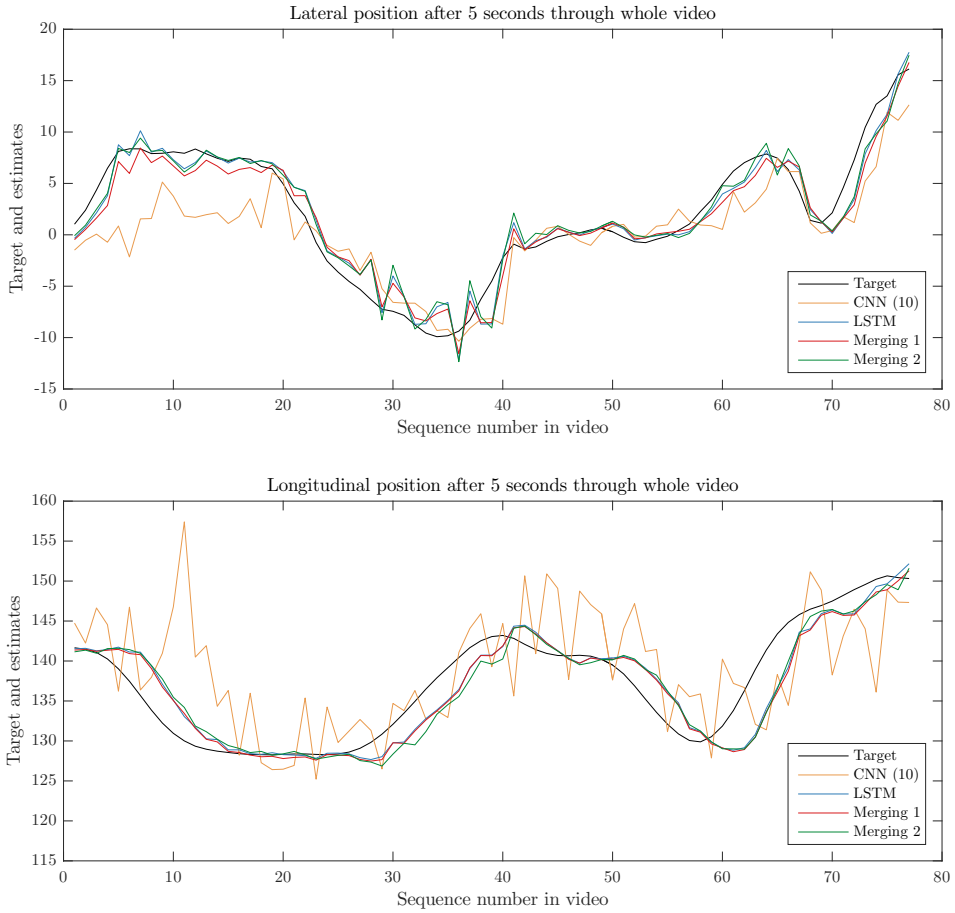


Figure 5.8: The lateral and longitudinal position after 5 seconds 77 different consecutive sequences. The predictions are done by the CNN, the LSTM and the full model in comparison to the ground truth. The x-axis shows the numbering of the sequences through the video.

5.4 Comparison to Interpolation

To evaluate the performance of the LSTM, merging model 1 and merging model 2, they were compared with an interpolation from the input data. The interpolation was done by using the longitudinal movement recorded in the last value of the input sequence. By assuming that the vehicle would not change its current speed the longitudinal position at each future time step could be computed. The lateral position was calculated from the lane marking estimations, such that the vehicle's lateral position was placed between the lane marking estimates. If either of the lane marking estimates did not exist, the path was placed alongside the existing lane marking estimate with the same offset as the vehicle currently has. If neither of the lane marking estimates existed, the path was to drive straight ahead. However, this occurred very seldom and had no significant effect on the result.

The comparison between the models and the interpolation is shown in figure 5.9. From the figure it is visible that the models perform similarly to the interpolation. In terms of the lateral movement the models are better than the interpolation, while the interpolation is better at predicting the longitudinal movement.

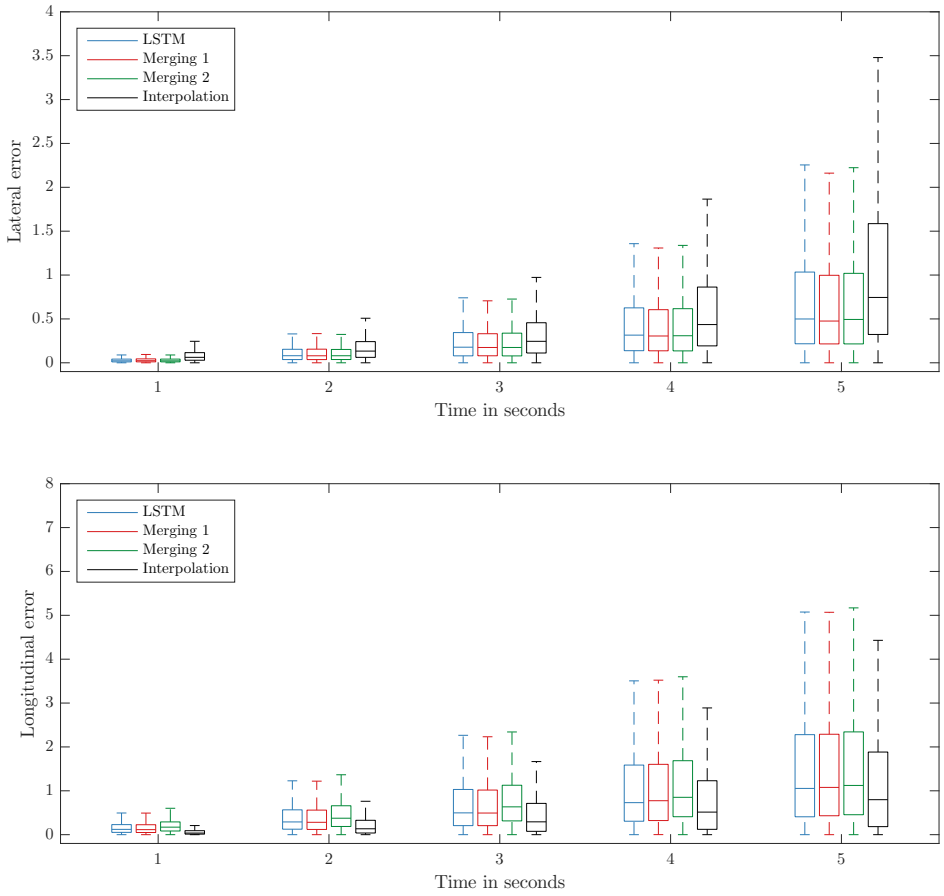


Figure 5.9: The box plots contain information about how the distributions of the error for the lateral and longitudinal position is over time when the data is faulty. The boxes represent the LSTM, merging model 1, merging model 2 and the interpolated path.

5.4.1 Accuracy

Figure 5.10 shows two paths which follow the same curvature but at different speeds. This causes the lateral and longitudinal positions of the two paths to differ largely. Therefore, looking at only the distance between the target position and the predicted position does not show the full performance of a model.

A different method of evaluating the predicted path is to not look at how far the vehicle has moved. Instead, one can use a third degree polynomial representation of the target path. The predicted path can then be evaluated by comparing the

predicted lateral position at a certain longitudinal position to the target position at the given longitudinal position. One can then look at how often this difference is larger than a threshold. The threshold was set to 30 centimeters, as a deviation smaller than this will likely not affect the comfort of the passengers when driving on the roads considered in this thesis.

Table 5.1 shows the frequency with which the models and interpolation were within 30 centimeters of their target. This is shown after 1, 2, 3, 4 and 5 seconds. The table shows that it is more common to be within the acceptable area early along the path. Table 5.2 shows the median of the deviations larger than 30 centimeters for the models and table 5.3 shows the largest deviation for each model. The tables show that the LSTM, merging model 1 and merging model 2 perform similarly and that the interpolation has lower accuracy and larger deviations.

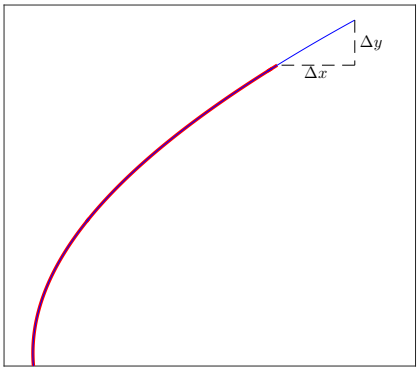


Figure 5.10: The image shows two paths which follow the same curvature, but at different speeds. Due to the difference in speed, the positions in the paths differ both laterally and longitudinally.

Table 5.1: The table shows the percentage of times the models were within 30 centimeters of a third degree polynomial representation of the target path after 1, 2, 3, 4 and 5 seconds.

Model	**Accuracy**				
	1 second	**2 seconds**	**3 seconds**	**4 seconds**	**5 seconds**
LSTM	99.92 %	91.65 %	70.44 %	48.44 %	33.00 %
Merging model 1	99.86 %	91.60 %	70.62 %	49.19 %	33.69 %
Merging model 2	99.97 %	92.12 %	70.66 %	49.09 %	33.62 %
Interpolation	97.04 %	82.08 %	58.13 %	36.73 %	23.34 %

Table 5.2: The table shows the median of the deviations (in meters) which were larger than 30 centimeters after 1, 2, 3, 4 and 5 seconds.

Model	Median of deviations larger than 30 centimeters				
	1 second	**2 seconds**	**3 seconds**	**4 seconds**	**5 seconds**
LSTM	0.32	0.41	0.49	0.61	0.79
Merging model 1	0.36	0.41	0.49	0.60	0.75
Merging model 2	0.42	0.41	0.49	0.61	0.79
Interpolation	0.38	0.42	0.52	0.70	1.04

Table 5.3: The table shows the largest deviation (in meters) from the target path representation after 1, 2, 3, 4 and 5 seconds.

Model	Largest deviation				
	1 second	**2 seconds**	**3 seconds**	**4 seconds**	**5 seconds**
LSTM	0.39	2.21	5.75	10.59	15.77
Merging model 1	0.51	2.31	5.70	10.10	15.06
Merging model 2	0.54	2.52	6.15	10.95	15.91
Interpolation	1.27	4.38	9.58	17.61	30.48

6

Discussion

This chapter discusses the work done in the thesis and the obtained result. The chapter starts by discussing the idea behind each of the presented models. Then the obtained results are discussed along with potential explanations for why the results are as they are. Additionally, the chapter presents thoughts on why the images influence the merging models so little, and also alternative methods tried to increase the influence from them. Lastly, ideas for what could be done in the future to further improve the results are discussed.

6.1 Model Selection

The models presented in chapter 4 were designed such that the network structure suited the network type. The LSTM model was designed to use the temporal information within the object data, while the CNN was to process the images. The main difference between the two merging models lies in where the recurrency appears. We have investigated two forms of recurrency; early recurrency on the merged inputs separately versus late recurrency on the merged inputs.

The model which was most straight forward in its implementation was the LSTM. The object data was separated into sequences. The sequences were then provided as input so the model was always trained to use 10 sequential input vectors before computing an output. After each input vector has passed the temporal information is used as the cell state and hidden state of the each LSTM unit is updated.

The CNN was the model which required most time to design and implement. The purpose with this model was to find a model which used only the images to predict the path. The presented CNN model has two versions as we wanted to investigate whether the model could make use of temporal information in sequential images. Stacking the images turned out to be the most effective manner of accessing this information.

Merging model 1 were designed such that images and object data were processed separately. In this model, the access to temporal information was done before merging the two inputs and was a part of the processing done on each input separately. Once

each set of input is processed, the information obtained from them is merged. The hope was that the model would be able to decide which information was most useful or combine them in order to make a more accurate prediction using two different sources of data.

Merging model 2 combines a single processed image with the object data. This was done to merge the information before trying to access the temporal information. The main reason to try to implement this model was due to a similar architecture having had success in predicting the steering angle [15]. Therefore, we wanted to investigate whether this type of structure could be used to predict the path.

While training merging model 1 and merging model 2, pretrained models were used, as mentioned in section 4.6. This was done for a couple of reasons. In the case of both the models, the main reason was to force a larger dependency on the output of the early inner networks, especially the CNN. Additional benefits were that the training time was reduced and the total performance of the models improved.

6.2 Discussion of Results

This section discusses the results presented in chapter 5. The results and their significance are presented. In addition, explanations for the obtained result are discussed.

6.2.1 CNN Model

The first comparison was between models which only used the image. Figure 5.1 shows how the errors compare for the CNN model described in section 4.1 when it uses 1 and 10 images. From the figure it is clear that the model performs better when it receives 10 stacked images instead of only 1 image. This means that the CNN with 10 images is better at finding a relation between the images and the path.

Research done by NVIDIA when they presented PilotNet, showed an ability to predict the steering angle using a single image [7]. However, our results show that using a single image does not work well when predicting an entire path. The result instead show that including several images into the CNN will improve its ability to predict the path.

Further in the discussion, only the CNN using 10 images is compared to the other models. This choice was done as using 10 images was clearly better, and including both versions of the CNN made the comparisons more difficult due to the different scales.

6.2.2 LSTM Model

The LSTM model was trained using object data, which contained positions of object detections, lane marking estimations and past movements of the vehicle. Figure 5.2 compares the performance of the LSTM to when the LSTM does not use either the object detections or the lane marking estimates. From the box plots it is clear that not using the object detections had a very small effect on the performance of the LSTM. However, not using the lane marking estimates had a large negative effect on its performance.

This indicates that the LSTM does not rely much on the object detections to predict the path of the vehicle. Instead, the LSTM relies mostly on the past movements of the vehicle and the lane marking estimations. Therefore, in situations when the lane markings are not available the LSTM will not perform as well. The hope was that in these situations, merging the object data with images would increase the reliability of the predicted path.

The object detections contain the information how vehicles ahead move in relation to the ego vehicle and therefore include information about the surrounding vehicles' change in velocity. This could be used to predict an acceleration or brake. However, in the used data there were very few situations where the vehicle brakes or accelerates and even fewer where the speed change is made due to the surrounding environment. The object detections therefore do not contribute much to the overall performance as shown in figure 5.2. Therefore, continuing driving in the same speed and following the lane markings will be a good approximation.

6.2.3 Performance of the Models

The box plots in figure 5.3 compare the errors of the CNN and LSTM to the errors obtained for the models which merge different types of inputs. In addition, figure 5.4 shows how the error build through the path. From these figure it is visible that merging model 1 and merging model 2 perform very similar to the LSTM. They also show that the errors of the CNN are much larger than for the other models.

Additionally, figure 5.3 shows that the obtained errors are worse the further into the future the prediction is done. This was expected, as the predicted path will have had more time to deviate from the target path. However, figure 5.4 shows that the errors do not increase linearly. Instead the errors get progressively worse the further into the future the predicted position is. A reason for this is that further into the future, there exist more possible paths which the model can choose between.

Figures 5.5-5.7 show examples of what the predicted paths may look like. While training the models, it was important to look at this type of plot to make sure the networks were not simply trained to always drive straight forward. In the figures it is clear that this is not the case, and they also show that the predicted paths are

rather smooth. It was also important to investigate whether the predicted paths were smooth, which they were. If the path had been jagged, the passengers in the vehicle would have been uncomfortable while traveling.

From figures 5.5-5.7 it is visible that the path predicted by the CNN differs the most from the target path, while the other predicted paths are quite similar to each other and the target. It is also visible that the models are capable of understanding how the road is turning. The behavior and performance shown in the figures is representative for the full test set. Commonly, the CNN was the worst estimate by far, while the other three models performed similarly and changed which was closest to the target path.

The graphs in figure 5.8 show the lateral and longitudinal positions for the models after 5 seconds for 77 consecutive sequences. In the figure it is immediately visible that the CNN does not perform as well as the other models. The graphs show that the output from the model is quite volatile and not robust, especially in terms of the longitudinal positions. Two consecutive sequences have similar input, but the output from the CNN can vary greatly between these. The figure shows that the other models perform very similarly. These models have relatively smooth curves, especially in terms of the longitudinal position. This indicates robustness in the models as small changes in the input will result in small changes in the output. The curves representing the lateral movements are also quite smooth for these models, except for sequences 25 to 40 where some sudden jumps appear. A look at the input data for these cases showed that in these cases the lane marking estimates were flawed.

In the graphs for the longitudinal position in figure 5.8 it is visible that the LSTM, merging model 1 and merging model 2 curves have a delay compared to the curve of the target. This indicates that the models are not capable of predicting a change in speed, however they can react to a change in speed provided in the input data. The tendency is not visible in the graphs for the lateral error. This again indicates that using the lane marking estimates is helpful when trying to predict how much the vehicle should turn.

One likely explanation for why the LSTM is performing much better than the CNN is that the relevant information is provided in a more accessible and extractable manner. This is not very surprising as the LSTM is provided values for where objects are and how the lane markings are estimated, while the CNN has to deduce this from the images as well as how the vehicle has previously moved. Thereby, the LSTM can focus on just computing the path given this information, while the CNN might not be able to extract the data in the images.

The CNN being much worse than the LSTM is a probable reason for why the merging models are very similar to the LSTM. When predicting the path of the vehicle, the merging models will likely find the information provided by the object data to be much more relevant than the information provided by the images. Therefore, the merging models treat the information from the images as mainly noise. This causes

the merging models to not be influenced by the images and perform as the LSTM.

6.2.4 Comparison to Interpolation

Section 5.4 shows the results of comparing the LSTM, merging model 1 and merging model 2 with an interpolation. Figure 5.9 shows box plots for how the errors are distributed after 1, 2, 3, 4 and 5 seconds. From the figure it is visible that the interpolation is better at predicting the longitudinal position than the models. However, the models are better in terms of predicting the lateral position. This is a sign that the models have a difficult time predicting the speed with which the vehicle is moving, but are capable of understanding when and how the vehicle should turn.

While looking at examples of predicted paths we noticed that in some instances the predicted path seemed to follow a trajectory similar to that of the target path, but either ended to soon or continued to far. Therefore, in order to investigate how ell the predicted path and target follow the same trajectory, a different evaluation metric was used which is described in section 5.4.1.

The comparison between the models and the interpolation with this metric can be seen in table 5.1 for the positions after 1, 2, 3, 4 and 5 seconds. In the table it is clear that for all it is much easier to stay within 30 centimeters early on along the path than later. In the table it is visible that the LSTM, merging model 1 and merging model 2 perform very similar in terms of accuracy, while the interpolation performs much worse.

Table 5.2 shows the median of the deviations which where larger than 30 centimeters and table 5.3 shows the largest deviation obtained. From these tables it is visible that the interpolation has the largest deviations. Meanwhile, the LSTM, merging model 1 and merging model 2 have rather similar deviations.

From the tables it is visible that the models do well in predicting the curvature of the path one second into the future; the accuracy is high and the deviation which lie outside are not very large. While the threshold of 30 centimeters was set as to when the passenger might experience discomfort, deviation of up to 70 centimeters will remain in the lane. Given this, after one second all models are still within the lane. After two second the accuracy is still good, but the larger deviations are quite high which make the models unreliable as they may predict something which is potentially very dangerous. Later, after 3, 4 and 5 seconds the accuracy has become quite low and the largest deviations very high. This makes the models unreliable after that far into the future. In order to use these models, the predicted path should likely not be followed for more than one second in order to remain in the lane.

6.2.5 Identified Outliers

In the data there are some cases where the predictions from the models are much worse than commonly. This can be seen in most plots in chapter 5. It is further seen in table 5.3 where the largest deviations after 2, 3, 4 and 5 seconds is much worse than the medians shown in table 5.2.

These specific scenarios were shown to be recorded within a few seconds of each other on a very curvy road. During this time the lane marking estimations did not correlate with the actual lane markings and the vehicle drove at the lower limit of included velocities in the dataset. Additionally, the view from the front-looking camera was very limited.

Looking at the input data, both images and object data, it is clear why the models cannot use the input data to predict something useful. The road is extremely curvy, with most target paths including an S-curve. This S-curve is not visible in the images, and only the first turn is seen. This causes the lane marking estimates to assume there to only be one sharp turn, while in fact it is followed by more turns. The output from all the models also assume this, and the predicted paths each include one very sharp turn.

6.3 Lack of Contribution From Images

As could be seen in section 5.1.3 and is discussed above, the CNN performed much worse than the other models. When looking at the performance and output of merging model 1 and merging model 2, it is noticeable that these models appear to ignore the images. This is likely due to the object data being consider much more relevant than the images. But why is this the case? There are a variety of reasons which could explain this.

One of the main reasons for this is the data used in the project. In a majority of cases the road is very good and nothing drastic is happening. Therefore it is beneficial for the models to simply continue with the same speed and direction as the vehicle currently has. This information is not provided by the images, rather by the information of how much the vehicle is moving and the yaw rate. Therefore, even without the object detections and lane markings the models benefits from ignoring the information in the images.

This issue could be alleviated by expanding the scope of the thesis. By including urban driving in the project, there is a large chance that the many scenarios contained either braking or acceleration. Thereby, continuing driving as right now will not be as good as in the available situations. This could increase the relevance of the images and also of the object detections.

A potential improvement would be to use more data while training the CNN. Currently, the training data represents approximately 7 hours of driving. We tested using less data, about half and a quarter of the total training set. The results show that using more data increases the model's ability to generalize to new test data. This indicates that increasing the amount of data even more would be beneficial to the CNN. One solution to this would be to extract more data. However, it is also possible that doing data augmentation can help. This can done in a variety of manner, for example cropping and scaling of already existing images. This approach remains unexplored in this project.

Another factor which is likely harmful to the images, is that in many situations the vehicle is following a truck. The truck fills a large part of the image and cover the view of the road ahead. Thereby, the images will not contain information about how the road turns in the future. In this cases it would be unreasonable to try to predict the path as the image provides no information about how the road is turning.

Additionally, it is possible that the model with images should be more modular, in a fashion which is more similar to mediated perception instead of letting the images use full behavioral reflex. This means that the CNN should not be trying to predict the path from the images. Instead, the images could be used to estimate the lane markings or detect objects. Thereby the images would contribute to creating the object data, and thereby potentially improve its reliability. This object data could then be used in the LSTM. Thereby the images would contribute in an indirect manner to the prediction of the path. This approach was not explored as the dataset did not allow for this type of training.

It it also possible that the models which only consider the images are too shallow compared to the state-of-the-art models. Structures such as the ones used in Deep Steering [11] are much more complex. These types of structured could however not be attempted in this thesis due to time and resource limitations.

The above-mentioned reasons lead us to think that either the images do not include the information to predict a suitable path or the relation between image and path is too far-fetched for our networks. A reason for the relation being to far-fetched is that it is necessary to create a relation between the image and the real world. This relation is not elementary and may be difficult to estimate. An alternative could be to predict the path in the image plane and then translate to the real world. However, the information necessary for this translation was not available for this thesis.

Finally, it is possible that using one front-looking image is not enough to perform this task. The surrounding environment can change a lot over time. In cases where the sight is blocked by other vehicles or curves, the images may not contain the required information to predict the path. It is also plausible that the images do not contain reliable information about how the road is 5 seconds into the future.

6.3.1 Usage of Only Images and Vehicle Movements

While driving a vehicle, it is possible that either the camera or other sensors start malfunctioning. In this cases, pieces of the input may be lost. However, what will likely not be dropped is the information about the vehicle's past movements. In order to see the effect from the images when the object detection or lane marking estimates do not exist, we did a trial in which these were not used.

Models similar to merging model 1 and merging model 2 were implemented where the object data used only consisted of the past movements of the vehicle. This means that the object detections and lane marking estimations were excluded from the models. In addition, an LSTM was trained using only the yaw rate of the vehicle and the distance moved at the previous time steps, to use for comparison.

We believed that in this case the images would be beneficial to the performance of the merging models, as the images could contribute crucial information about the surrounding environment. However, the results in this trial showed that the CNN performed slightly worse than the other models in terms of the lateral position, and much worse in terms of the longitudinal position. In this case the merging models also performed very similarly to the LSTM. This indicates that the CNN is not capable of aiding the merging model by providing more relevant information than that provided by the speed and yaw rate.

6.3.2 Alternative Image Models

The models using the object data performed better than the models using solely the images. Therefore, most of the other models tried were models which used the solely the images.

The CNN presented in section 4.1 makes use of a series of convolutional and pooling layer before some final fully convolutional layers. A different model was constructed which was inspired by PilotNet [7]. The model did not include any pooling layers, instead only using convolutional and fully connected layers. This model also had two versions, one with 1 image and one with 10 images. This model performed in a similar manner to the presented model but with a slightly larger loss.

We also tried using models which made use of the full image instead of the down-sampled version. These models were similar in structure to the presented CNN and the CNN which used only convolutional and fully connected layers. The models showed no improvement by including a larger image, with only difference being an increase in the amount of computation time needed.

As an alternative approach to the image model, a pretrained ResNet-18 [30] was loaded and had its last linear layer reset to predict a path. Since ResNet-18 is originally trained using 224×224 RGB images, the images had to be resized to that

size and the RGB channels were filled with the gray-scale values. This pretrained network did however perform similarly to the CNN with one image and is therefore not shown. In addition, the CNN performed the best when 10 stacked images instead of one. ResNet-18 was trained using 3 channel originally and therefore, attempts at stacking the images could not be done.

An extension of the image models was also implemented and trained which consisted of a CNN followed by the LSTM. This extension was done with both the presented CNN and ResNet-18, where neither showed and ability to learn the underlying behavior. Both implementations got stuck in the local minimum of driving straight ahead with a fixed speed with only small deviations. We attempted to use a pretrained CNN to predict a path and then continue training it follow by an LSTM, but the same behavior as before was observed.

6.4 Usage of Models

One of the aims with this thesis was to investigate whether neural networks can be used to predict the path of an autonomous vehicle. Therefore, it is relevant to consider whether the predicted path can be given to a controller to determine the speed and steering angle of the vehicle.

In general, neither of the models can be used to predict the path of a fully autonomous vehicle. The models have been trained on a specific set of situations. Therefore, it is unrealistic to expect that the model could perform well in any other situation.

Considering situations which are similar to the training situations, the LSTM, merging model 1 and merging model 2 can likely be used as lane-keeping assistance. The models are quite capable of predicting an appropriate path in terms of understanding the curvature of the road. However, they struggle with choosing the correct velocity and cannot predict rapid changes in velocity. It is also important that the path is updated at least once every second, as the path should not be followed much further than that.

6.5 Future Work

In order to fully evaluate the ability of the neural networks more research should be done to improve the performance of the models. They should also be generalized to include more scenarios than the scope of this thesis.

For the model to act as more than a lane-keeping assistance, it needs to be capable of predicting changes in velocity. The data used in the thesis was limited to rural

areas where the mean speed of a sequence was within $[65, 130]$ kilometers/hour. As a consequence, the number of situations where a large change of speed occurs were few and only a limited connection to the surrounding environment was found. By including a larger variety of situations, it is possible that the model will become better at predicting large changes in speed.

An added benefit of considering a larger variety of situations is that there is a possibility that the images might contribute. As mentioned, it appeared that the in most scenarios it was beneficial to simply continue in with the same speed and stay between the lane marking estimates. In scenarios where this is not the case, it is possible that the models might require more knowledge about the surrounding area, which could then be provided by the images.

A further extension to the project would be to train in scenarios where the vehicle has to choose between several valid paths. In this cases it is important that the vehicle chooses one of the paths and not a middle road as this will have devastating consequences. To handle this type of behavior, the models have to be structured in a different manner and consider different input. Besides the information about the environment, information about intent is needed so the model knows where it should be heading.

Additionally, before implementing the models in a real vehicle the results should be tested and verified in a simulation environment. This would give an insight to how the model might perform in reality. This would also reveal how the model performs if the vehicle has deviated from the target path. In addition, it would show what the model outputs when the historic data is not provided from a drive with an experienced human driver.

A different approach, which should be tried for the images is to use them in a mediated perception manner. This means that the images could be used to find surrounding objects as well as estimate the lane marking. These estimates may then be better than the ones used as object data, and thereby the images could help improve the performance.

The better models appear capable of predicting the path decently 1-2 seconds into the future. Further than that the predictions become very flawed and unreliable. It would then be reasonable to make models which do not predict further into the future. As the model stands right now, it is possible that the attempts of predicting 5 seconds into the future hurt the models ability of predicting 1-2 seconds very well.

Finally, a controller should be implemented which makes use of the predicted path. This would force the vehicle to follow a specified set of constraints to improve the safety. The models have a difficult time estimating the speed of the vehicle, something which a controller could potentially help with.

7

Conclusion

The goal of the thesis was to predict a vehicle's position for the upcoming seconds and investigate whether combining different forms of input improved the performance. A total of four models are presented; one of the models used only images, one used only object data and two used both images and object data. All models were trained by imitation learning using approximately 7 hours of collected driving data.

The presented models perform very similarly, with the exception of the CNN which shows a lesser performance in all areas. In the model structures used and with the available data, the images appear to not contribute in any significant manner to increase the reliability of the network.

The models show an ability to predict a path with a somewhat high accuracy. The predicted path is smooth and feasible in many situation, with a more accurate prediction laterally than longitudinally. The models also beat the benchmark of an interpolation when following a path. The results also show that the models are reliable for 1-2 seconds before the accuracy drops.

However, the models need to be tested and verified, for example by letting a vehicle follow the path predicted by the model in a simulated environment. Before this is done, it is uncertain whether the models are capable of predicting viable paths for a vehicle.

Bibliography

[1] J. Arbib and T. Seba, "Rethinking transportation 2020–2030: The disruption of transportation and the collapse of the internal-combustion vehicle and oil industries", in *RethinkX Sector Disruption Reports*, RethinkX California, US, 2017.

[2] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the darpa grand challenge", *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[3] Waymo, *Technology*, `https://waymo.com/tech/`, Accessed: 2018-01-26.

[4] NVIDIA, *Autonomous Car Development Platform from NVIDIA DRIVE PX2*, `https://www.nvidia.com/en-us/self-driving-cars/drive-px/`, Accessed: 2018-01-26.

[5] Zenuity, `https://www.zenuity.com/`, Accesssed: 2018-05-14.

[6] Waymo, *Journey*, `https://waymo.com/journey/`, Accessed: 2018-01-26.

[7] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars", *CoRR*, vol. abs/1604.07316, 2016. arXiv: `1604.07316`. [Online]. Available: `http://arxiv.org/abs/1604.07316`.

[8] Volvo Cars, *Swedish families help Volvo Cars develop autonomous drive cars*, `https://www.media.volvocars.com/global/en-gb/media/pressreleases/217555/swedish-families-help-volvo-cars-develop-autonomous-drive-cars`, Accessed: 2018-01-26.

[9] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, *et al.*, "Towards fully autonomous driving: Systems and algorithms", in *Intelligent Vehicles Symposium (IV), 2011 IEEE*, IEEE, 2011, pp. 163–168.

[10] C. Innocenti, H. Lindén, G. Panahandeh, L. Svensson, and N. Mohammadiha, "Imitation learning for vision-based lane keeping assistance", *CoRR*, vol. abs/1709.03853, 2017. arXiv: `1709.03853`. [Online]. Available: `http://arxiv.org/abs/1709.03853`.

[11] L. Chi and Y. Mu, "Deep steering: Learning end-to-end driving model from spatial aand temporal visual cues", *CoRR*, vol. abs/1708.03798, 2017. arXiv: `1708.03798`. [Online]. Available: `http://arxiv.org/abs/1708.03798`.

[12] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun, "Learning long-range vision for autonomous off-road driving", *Journal of Field Robotics*, vol. 26, no. 2, pp. 120–144, DOI: 10.1002/rob.20276. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20276. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20276.

[13] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng, "An empirical evaluation of deep learning on highway driving", *CoRR*, vol. abs/1504.01716, 2015. arXiv: 1504.01716. [Online]. Available: http://arxiv.org/abs/1504.01716.

[14] D. A. Pomerleau, "ALVINN: An Autonomous Land Vehicle in a Neural Network", in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed., Morgan-Kaufmann, 1989, pp. 305–313. [Online]. Available: http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf.

[15] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets", *CoRR*, vol. abs/1612.01079, 2016. arXiv: 1612.01079. [Online]. Available: http://arxiv.org/abs/1612.01079.

[16] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943, ISSN: 1522-9602. DOI: 10.1007/BF02478259. [Online]. Available: https://doi.org/10.1007/BF02478259.

[17] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.", *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, ISSN: 0018-9219. DOI: 10.1109/5.726791.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257.

[20] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa, "Natural language processing (almost) from scratch", *CoRR*, vol. abs/1103.0398, 2011. arXiv: 1103.0398. [Online]. Available: http://arxiv.org/abs/1103.0398.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", *CoRR*, vol. abs/1409.4842, 2014. arXiv: 1409.4842. [Online]. Available: http://arxiv.org/abs/1409.4842.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[23] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks", in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., ser. Proceedings of Machine Learning Research, vol. 15, Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *CoRR*, vol. abs/1412.6980, 2014. arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980.

[25] L. Prechelt, "Early stopping — but when?", in *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67, ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_5. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_5.

[26] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", *CoRR*, vol. abs/1207.0580, 2012. arXiv: 1207.0580. [Online]. Available: http://arxiv.org/abs/1207.0580.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *CoRR*, vol. abs/1409.1556, 2014. arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[29] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch", in *NIPS-W*, 2017.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.