CHALMERS
UNIVERSITY OF TECHNOLOGY

Prediction of Driver Actions With Long Short-Term Memory Recurrent Neural Networks

Master's thesis in Biomedical Engineering

Martin Torstensson

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

Master's Thesis:EENX30

Prediction of Driver Actions With Long Short-Term Memory Recurrent Neural Networks

Martin Torstensson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

Prediction of Driver Actions With Long Short-Term Memory Recurrent Neural Networks

MARTIN TORSTENSSON

Supervisors:
Cristofer Englund — Research Manager, RISE Viktoria AB
Boris Durán — Senior Researcher, RISE Viktoria AB
Artur Chodorowski — Department of Electrical Engineering, Chalmers University of Technology

Examiner:
Artur Chodorowski — Department of Electrical Engineering, Chalmers University of Technology

Master's Thesis:EENX30
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000
Gothenburg, Sweden 2018

Abstract

The most prominent factor behind traffic accidents today is human errors. There are many ways, in which problematic behaviors such as inattention can be mitigated. One of the tools used for this purpose is warning systems. These warning systems needs to give warnings both at a time where the driver needs it as well as give the driver a sufficient window to react. In order to achieve these goals information of the state of the driver can be vital to know how curtain situations should be handled.

A prediction of future events could be used in order to increase the amount of time between the warning and the dangerous event. This report explores possibilities of using recurrent neural networks with long short-term memory for prediction of eight different driver actions inside of a vehicle, such as glancing and reaching inside of the vehicle among others.

Other studies in the domain of predictions of driver actions have had the focus on car movements such as breaking and lane changing, while this study concentrates on the state of the driver. There is potential for these predictions to improve a warning system and give a driver more time to react to a given situation. The predictions are based on sequences of actions, which are generated from sequences of images with a convolutional neural network.

A dataset consisting of sequences of images used in the report was gathered at RISE Viktoria AB. The hyperparameters of the recurrent neural network, such as the number of hidden units and amount of layers, was chosen with Bayesian optimization. An addition of a parallel input of optical flow created from the input images was found to improve the performance of the convolutional neural network. The complete network achieved an average prediction accuracy of 80% for the next frame predictions and 62% after 20 frames. A comparison where the predictions were set to the last element in the input achieved an accuracy of 79% for one frame ahead and 49% after 20 frames.
Keywords: Machine learning, LSTM, RNN, Optical flow

Table 1: Abbreviations used in the report

CNN	Convolutional neural network
RNN	Recurrent neural network
LSTM	Recurrent neural network with long short-term memory
MSE	Mean squared error
GPS	Global Positioning System
CAN	Controller Area Network

Acknowledgements

I want to thank my supervisors Cristofer Englund and Boris Durán at RISE Viktoria AB, who have provided a great deal of support and ideas during the project. My supervisor and examiner at Chalmers University of Technology, Artur Chodorowski, also deserves thanks for the information and help he has given me. I am grateful to David Lindström for taking the time to answer all of my questions and Joachim Andersson for his expertise in academic writing. Finally, I want to thank the employees at RISE Viktoria for creating a great working environment.

List of Figures

List of Tables

Contents

1 Introduction

Traffic accidents are a large problem today with many casualties worldwide every year. In the vast majority of the cases, human errors play a large role in creating the situations leading up to the accidents [2]. Hence there is a great potential benefit to traffic safety by finding ways to erase or mitigate this source of traffic accidents. Among current technology used for this purpose is forward collision warnings, which are designed to detect and inform the driver of a danger and thus giving the driver an opportunity to take an appropriate action. For this purpose, a sophisticated system is required to detect danger in situations where it is needed but not to excessively alert the driver of situations where there is little or no danger present. If the system does warn the driver to frequently it can affect the driving experience for the worse and even lead to the driver not responding to a warning as it is assumed to be a false positive.

There are several ways to detect and assess dangers such as using cameras or radars to canvass the area and detect the surrounding traffic as well as pedestrians. Other options could be to instead direct the focus inside the car and consider the state of the driver. Based on the driver's posture, gaze, and several other factors the focus of the driver can be estimated as well as whether the driver is able to respond to current traffic conditions or not. An indication of danger can potentially be detected by comparing the posture and actions of the driver to other scenarios and draw conclusions based on thier outcomes. There are cases when a good understanding of the surroundings of the car is needed to warn the driver effectively. This study on the other hand deals with another important factor in creating a good warning system and that is the driver. Depending on the state of the driver the system needs to adapt to make relevant warnings ahead of time. As an example the same warning can to an alert driver or a distracted driver be a nuisance or lifesaving respectively. Therefore, to create an effective system the state of the driver needs to be understood to handle a specific situation.

1.1 Aim

The intention of this report is to explore the possibilities of training recurrent neural networks with long short-term memories to predict the actions of a driver based on sequences of images. One goal was to implement networks to classify and predict driver actions based on images. Another goal

was to evaluate the networks on a data set.

1.2 Limitations

The dataset used in this report was collected with eight different participants and one car. For generality, it could be beneficial to use a larger dataset with more participants in different driving environments.

In this study, the number of actions of the driver used for classification and prediction is limited to eight. These actions are all a part of a sequence of picking up an object. For practical use, more of these actions would probably be required in order to cover all driving scenarios, in addition to other settings, such as time of day. However, a single case is used in this study to show proof of concept.

1.3 Related work

There are cases of when recurrent neural networks with long short-term memory (LSTM) have been used in order to predict actions in traffic situations. In [3] a LSTM was used to predict driver actions such as lane change and turns. The predictions were based on among other information from the face and head orientation. Also, other types of sensors were used where one camera was directed forward on the road as well as GPS data and data collected by the car in addition to the information of the driver's head. This type of sensory fusion could be beneficial as a future prospect for this thesis as well. Even though these types of predictions could provide a great benefit for among other warning systems there are other factors that could further improve this type of technology. It could also be out of interest to more in-depth study of the driver's behaviour inside of the car. A warning system could be designed to both make use of predictions of what is happening outside of the car and its surroundings but also consider what is happening on the inside. For that purpose this thesis instead considers information from the full body posture of the driver to make predictions of the drivers actions and distractions.

The images of the drivers face and head orientation used in [3] was processed by detecting and following specific parts of the images with faces. Another possible way of processing the images could be to use convolutional neural networks (CNN). There are many examples of when these

types of networks have performed well in the domain of image classification. A competition named ImageNet Large Scale Visual Recognition Challenge, has had many candidates using CNNs performing well in the classification of images. One example of a CNN that have won the challenge is GoogLeNet [4]. A CNN can both produce complete classifications of images or by removing the last parts of the network, create features.

The classification of images can be further improved with the use of optical flow. In [5] optical flow images were created by methods such as optical flow stacking and trajectory stacking. The optical flow images and the original images were then processed separately. The network consisted of five convolutional layers, three max-pooling layers, two fully connected layers and one softmax layer. Using optical flow in combination with the original images can be benefical as CNNs commonly do not consider changes in between images. There are several situations where it can be beneficial to also use information that is stored in a sequence of images rather than just in single images. In the case of a driver, it can as an example, be important to know in which direction the driver is moving, which can be hard to gather from a still image.

In another article by [6] there is an approach of using a sensory fusion with among other information from a facial camera in the input. There were also other types of sensors used as well, such as the GPS similarly to the previous case. Also data gathered from the car, CAN bus data and so on was used among other sensors. In this case the LSTM used was instead a bidirectional LSTM. As in [3] the focus was on predictions of movements of the car such as breaking and lane changes. For the case of predictions of driver actions inside of the car the use of bidirectional LSTMs is an interesting choice and has been used in this thesis.

A model of how drivers behave, which also used sensory fusion was presented in [7]. The behaviours was whether the driver was aggressive or not in three different surroundings. To make these identifications a combination of different input data was used, with a focus on the CAN-bus and GPS. This method allowed for an identification that could be performed with cheap and easily accessible equipment. One of the downsides on the other hand was a time requirement for the process to adapt, affecting the results for the first few seconds. With a CNN-based classification the time frame needed can potentially be reduced as a CNN can be created with

only input from a few time points back.

In the domain of predicting driver actions [8] used head orientation to create long time predictions of lane changing maneuvers. The predictions were based on glances as a way to detect intentions of lane changing at a time well before the action is taken. This resonates well with the concept of this thesis, which assumes that there are signs, such as glances, leading up to certain actions of the driver. The classifications were made based on a model, that compared the angle and position of the drivers head compared to where the gaze was directed. The classes used were several areas inside of the car, such as windows and mirrors. Based on the training data, normal distributions of the classes were created as a function of the orientation of the head. This meant, on the other hand, that even though some choices of angles had a more favored class there was an overlap between the different classes. That type of classification might lose accuracy when a range of angles can not be directly associated with a specific class. There is a potential to further improve the classifications by using machine learning instead of using a model of assuming a normal distribution of the classes. Using the orientation of the head can provide valuable information to understand what coming actions a driver is about to take but it can be considered as a part of using the full body posture. Therefore, it could be of interest to expand the input to full body posture to gain more information of the driver. Another approach for long time predictions of driving behaviour was explored in [9], based on GPS coordinates. The data used was recorded with 30 second intervals, over long periods of time. A LSTM was used to perform the analysis of the data and determine changes in behaviour. As these predictions are on a larger scale time-wise it can potentially be hard to draw conclusions from shorter sequences of actions of the driver inside of the car, which is explored in this thesis.

Others have instead of concentrating on the actions of a driver looked further into awareness. In [10] a method similar to machine learning, in the sense that it was trained to achieve a gathered experience of previous scenarios, was used. It did on the other hand not cover predictions of future events.

This study proposes a method using a combination of a CNN and a LSTM to classify and predict future actions of the driver based on sequences of images with the drivers full body posture. No other study using this type of method on images of full body posture to predict actions of

the driver, rather than movements of the car, have been found.

2 Proposed model

The network that has been proposed in this study could be divided into several components and is shown in Figure 1. The inputs to the system were sequences of images, which were downsampled to reduce size and also annotated to create a ground truth, which can be considered labels to the images. The smaller downsampled images were then converted into grayscale. Once the preprocessing was done optical flow vector fields was calculated. One of the two major components in the network was a CNN which took as an input the grayscale images, the optical flow vector fields and the ground truth. The CNN makes classifications of the images, which could be compared to the ground truth to see how correct they were. These classifications were then sent to the second major part of the system namely the LSTM. In this component the sequences of classifications was extrapolated in time to create predictions of future classes. The ground truth could then once again be compared to the created predictions to find out how well the LSTM performed. Alternatively the LSTM was tested on the output labels from the CNN. The ground truth was also used in order to train both the CNN and LSTM.

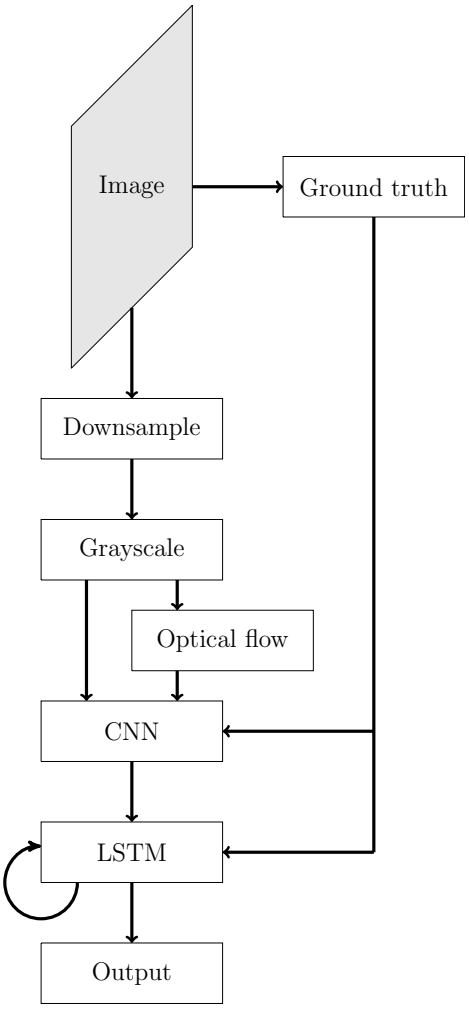

Figure 1: A schematic overview of the network proposed in this thesis.

3 Classification

This section will start by giving a brief explanation of convolutional neural networks, CNNs, and then go through the CNN used in this report.

3.1 Convolutional neural networks

Convolutional neural networks usually consist of several different types of components, which performs a certain task. A common form of input to a CNN is an image. The components used are in general built out of a set of nodes where the value in each node is calculated based on parts of the elements from the input or in the case of when these components are stacked after each other the values from the previous component. Due to how these components are lined up after each other and that their elements in many cases are in the same shape as the input they are for CNNs called layers and the nodes are sometimes called neurons in an analogy to the brain [11]. Three different types of layers will be explained further and these are convolutional layers, pooling layers, and dense layers.

3.1.1 Convolutional layers

In CNNs features are extracted using convolutional layers. A visualization of a convolutional layer can be seen in Figure 2. In the case of this figure layer 2 is the convulotional layer and the circles are the elements or nodes of that layer. The square represents a window, which is applied to the previous layer i.e. the input to the convolutional layer. This window has a height h and width w that determines how many elements the window covers. In the case of three dimensional input the window can also be considered to have a depth. The lines are connections between the layers also know as weights.

As explained by [11] every element in the convolutional layer is connected through the weights of the window to several elements in the layer before it. A weight in the case of CNNs refers to a variable of the network, which can be changed when training the network to improve the performance. The output of these neurons are merged as an element-wise product [11]. It is then further explained that the next element is calculated by moving the window one or several steps, this process is repeated for every element in the case of movements of one step. This type of layer is shown in Figure 2 where the window covers two elements in both directions.

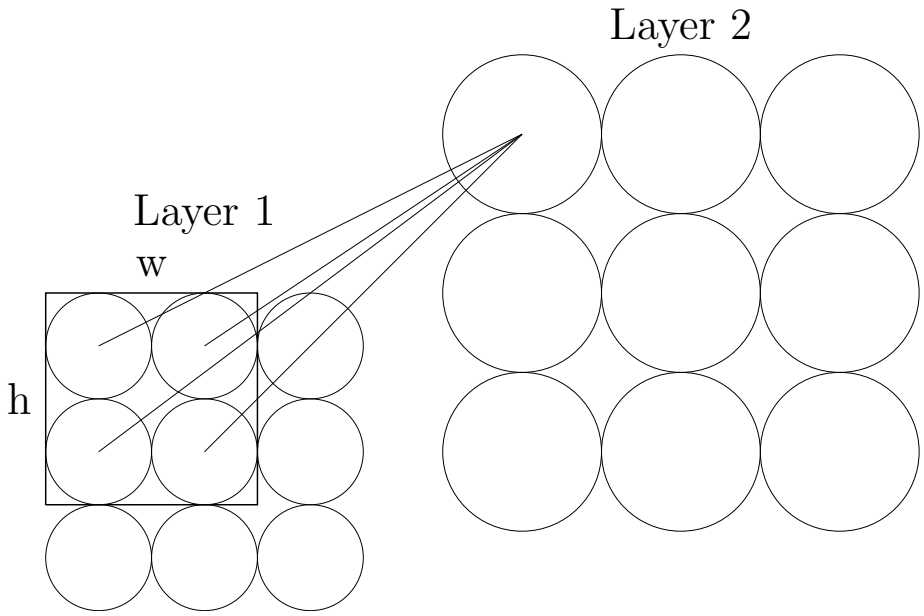

Figure 2: A convolutional layer in a neural network which has a window that covers two elements in both directions. Here Layer 2 is the convolutional layer. The circles represent the nodes and the lines are the connections between the layers with their individual weights. For the sake of a clearer visualization the connections from all neurons have not been drawn.

3.1.2 Pooling layers

The pooling operation, performed by a pooling layer, consists of applying a function to a window, which is moved over the nodes of the input. The major properties of a pooling layer are width and height of the window as well as the stride and the type of the pooling layer. Depending on the stride, the window is moved a different amount of pixels in every step. Examples of these types of layers are max as well as average pooling. These types affect the window function. Max-pooling extracts only the value from the node within the window with the highest value, i.e. out of all the node values in the current window only the highest is chosen. Average pooling, on the other hand, computes the average of all node values in the window [12].

As illustrated in Figure 3, when using max-pooling the resulting layer will be smaller than the previous layer, given a large enough stride. In this example, the pooling is over two elements in both the lateral and vertical direction. This is not always the case and can vary depending on the use. In this scenario, on the other hand, this will result in 16 elements in the first layer transforms into four in the second layer thus reducing the number of elements by a factor of four. Notice that there is no overlap between the windows in this case, which contrasts the example of the convolutional layer.

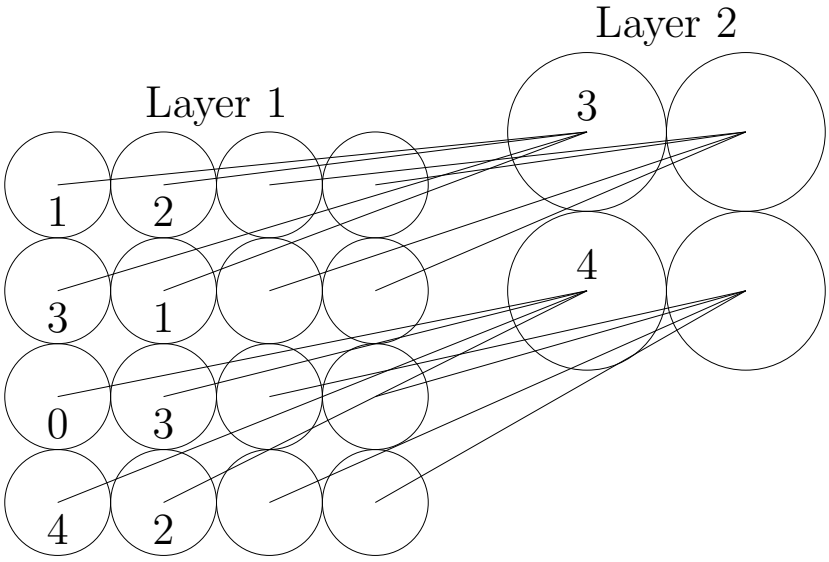

Figure 3: Layer 2 depicts a maxpooling layer, which covers two elements in both directions. Circles represents neurons and the lines shows which elements are used to create the element in Layer 2. From the four corresponding elements in Layer 1 the highest value is given to the element in Layer 2.

3.1.3 Dense layers

A dense layer, also called fully connected layer, is a layer where every node is connected to every node in the previous layer. The connection consists of a weight, which is multiplied by the value of the connected node in the previous layer. The value of the element in the dense layer is then calculated by summation of all these weights and previous node values as well as the addition of a bias [11]. An example is shown in Figure 4 where Layer 2 is a dense layer with four units, which are all connected to the previous layer, which also contains four units.

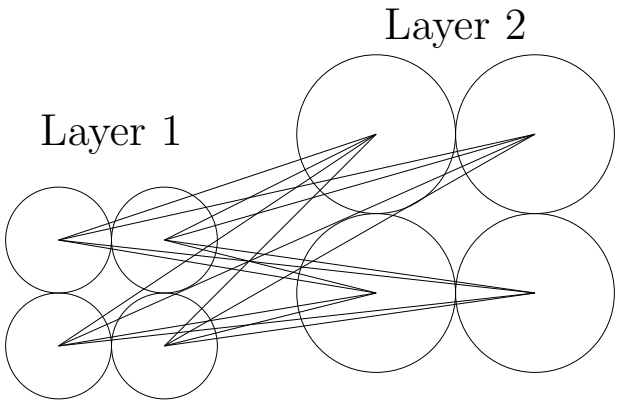

Figure 4: Here Layer 2 is a dense layer. The circles represent the units and the lines the connections between the layers with their respective weight. Dense in this case refers to how all the elements are connected to all the elements in the previous layer.

3.2 Convolutional neural network model

To do the initial classification of the images, a CNN has been used which can be seen in Figure 5. It consisted of four different types of layers; convolution, max-pool, dense and dropout. The input image was connected to a convolutional layer, which extracted features. This layer, in turn, was connected to a max-pooling layer where the size of the layers was reduced. The combination of a convolutional and max-pooling layer was repeated four times. After the last max-pooling layer was a dense layer, which was created by flattening out the output from the max-pooling layer. This dense layer was in turn connected to a dropout layer. In the same manner, as before the combination of dense and dropout layers were repeated three times.

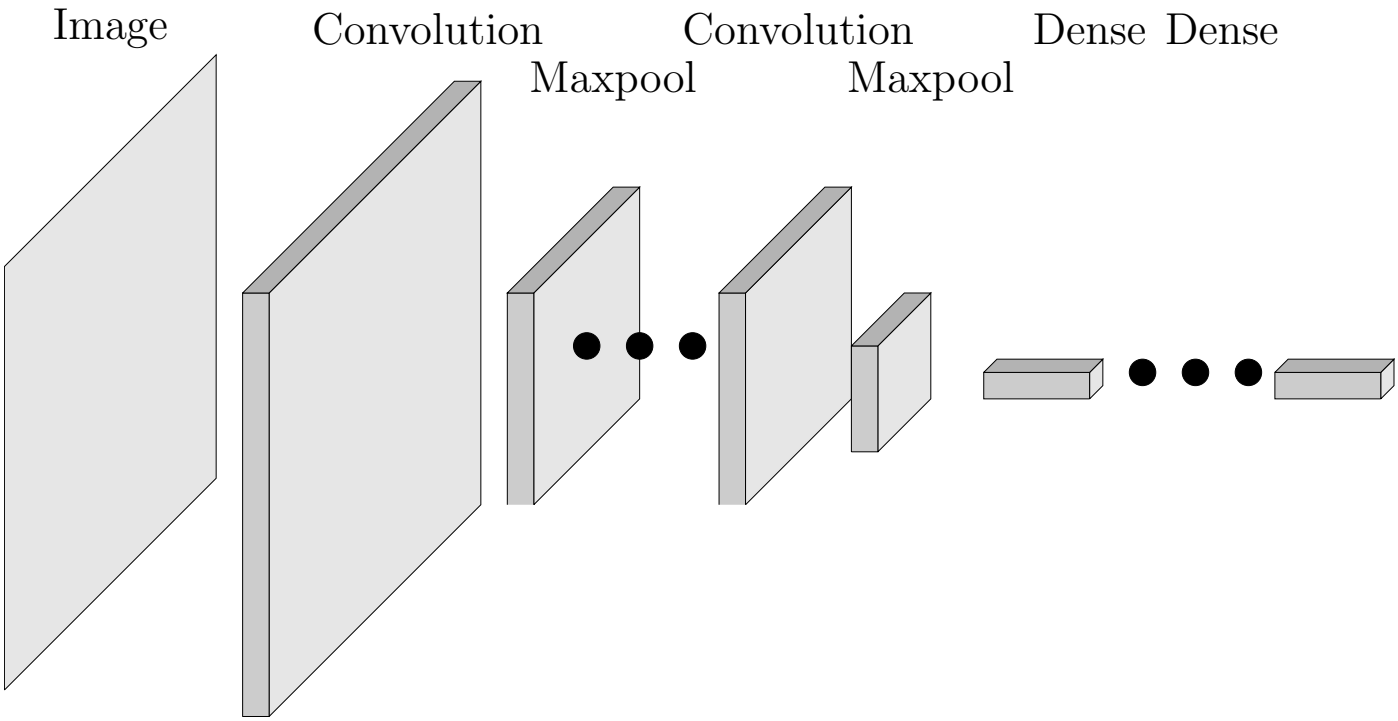

Figure 5: The input from an image is processed through a convolutional layer and then max-pooled. This process is repeated four times. After the last max-pooling follows three dense layers.

When working with images there are several important factors in order to recognize and classify objects or scenes. To start with, there is the overview of the image with rough shapes. This overview in turn is built up out of small fragments of the image that can vary from whole objects to simple shapes. By using several of these fragments together more interesting things can be detected. The convolutional layers can be trained to detect different types of these fragments [13].

In a small image, there can be several thousand pixels so with only one layer where every element is connected to a few pixels with one weight and bias in every connection it is easy to realize that there is a need for a large number of weights and biases resulting in many computations and large capacity for storage. In networks with many layers it can be beneficial to reduce the size of the layers to make it less computationally heavy. Therefore, pooling layers can be beneficial [11].

Because two synchronized cameras were used during data collection the dataset contains two images for each time point. The images from both cameras could have been separately classified but that on the other hand would result in problems when sending the information to the RNN and

process it there. Instead, both images were processed with this type of pipeline with convolutional, pooling and dense layers as shown in Figure 6. In the later dense layers, they were concatenated resulting in the last dense layers being shared. This meant that both the images could have features extracted from them and gradually reduced in size while still being able to affect the classification. This setup did on the other hand demand more computational power and memory usage compared to only using one image.

The measurement of the performance of the CNN used in this study was accuracy. Where the accuracy, for the purpose of this thesis was defined as the amount of correctly classified images on the test set divided by the total amount of classifications made on the test set. This gave an average of how successful the classification was.

3.3 Optical flow

When working with sequences of images there is not only information in the images themselves but also in the changes in the sequence. Due to this, it can be of interest to, instead of training the network based on pixel values, use the gradient values in time. In order to achieve this, optical flow can be used. As described in [14], optical flow is based on the movements between separate frames and can be based on two assumptions. The first is that the pixel values in one image to the other might be rearranged but not removed. This is of course not always the case and can cause issues but does also in many cases work well even though it might not completely hold. The second assumption is that nearby pixels have a similar rate of change. This is in many cases true, for example when an object moves it is likely that all the pixels inside a small area representing parts of the object are moving at a similar phase. On the other hand, in the transition between objects and the background, this might not be the case. Depending on the method used there are ways to mitigate these issues, such as separation of objects and background. It is not likely to be able to find solutions which can completely fulfill these assumptions and therefore rather than solving this problem it is formulated as a minimization problem to optimize.

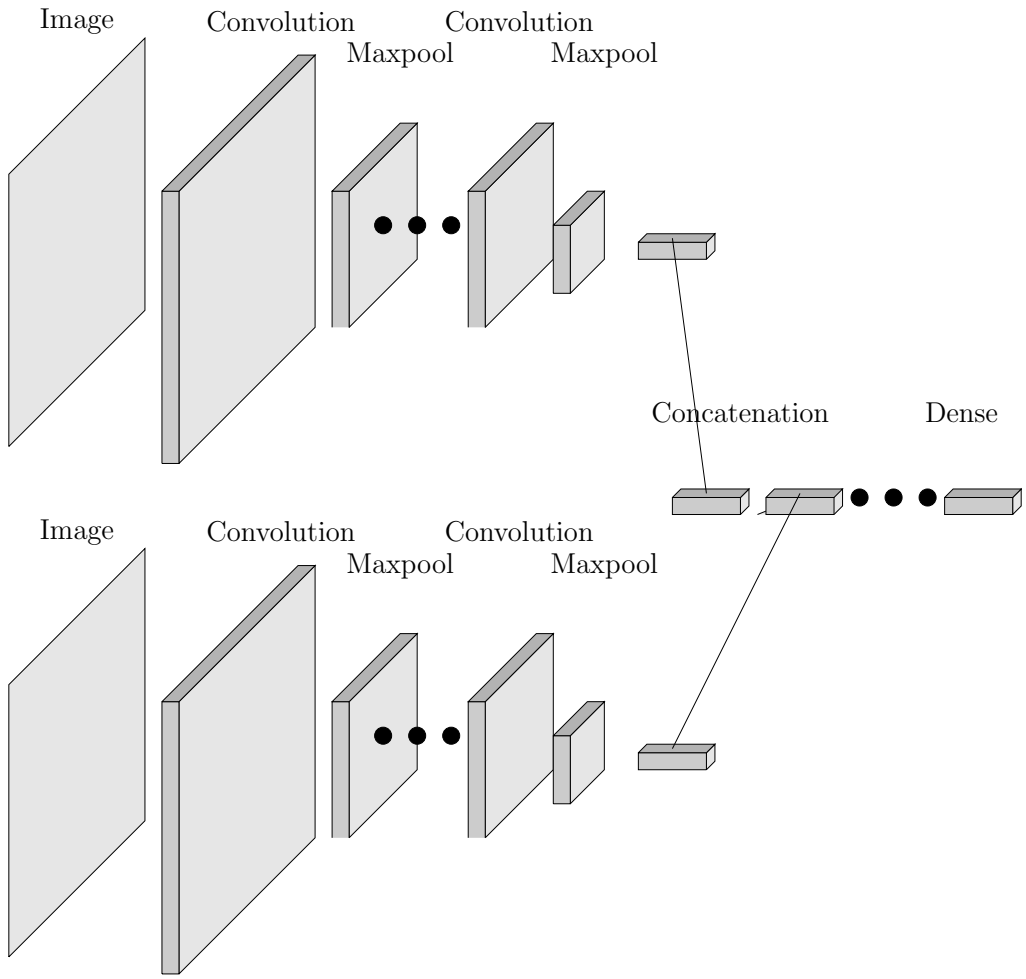

Figure 6: The input from an image is processed through a convolutional layer and then max-pooled. This process was repeated four times. After the last max-pooling followed a dense layer. This sequence of layers were done for both input images in parallel. Then both the dense layers were concatenated into one. Lastly, there were three dense layers.

In a similar way to how the original CNN model was created, a network which also used optical flow in an attempt to improve the network's performance has been implemented. The program used to create optical flow vector fields was created by [15] and [16]. Two input images were first used to calculate the optical flow between the previous image and the current. This was done to implement a sense of time in the CNN. When using optical flow the network had information about time as it creates gradients between the two frames used. The method used in this report used two images as input. Convolutional layers were then used on both images in parallel and then connected them to the dense layers. The same thing could be done with optical flow fields. A dense vector field of optical flow from a 2D image can be considered as a 2D image with two channels one for the vectors in the x-axis and one for the y-axis. The resulting vector field was then passed through a pipeline of convolutional, pooling

and dense layers and concatenated to the dense layers corresponding to the original image. By calculating the optical flow between the last two frames and applying convolutional layers to it in parallel to the input from the original images, a system with four different inputs could be created. The resulting system does not only consider the current state of the images but also an approximation of their motion.

4 Prediction

In this chapter, a brief introduction to recurrent neural networks and long-short term memory cells is given. There is also a description of the LSTM used in this project.

4.1 Recurrent neural networks

There are networks which are focused around sequences rather than fixed time points, such as the recurrent neural network (RNN). Such a network uses information from previous time steps of an input sequence. This gives an advantage while processing data where the sequence is of importance, such as videos. What separates RNNs from other types of neural networks such as CNN is that the output is calculated using a memory, which is based on the calculation of the previous element in the sequence. This element in turn is also based on the previous elements [17].

The introduction of the memory components has a drawback, which is usually called vanishing and exploding gradients. This is due to a problem where components corresponding to long-term memory are affected by weights from every step of the sequence. If these weights are large enough a small change in the long-term components can completely change the current components. For small weights, the opposite can be true where there is close to no contribution at all from the long-term components [17].

4.1.1 Mathematical description

As explained in [3] a RNN can be described as follows. Call the input $\mathbf{x}_t$ where the use of bold text is to show that it is a vector and the t that it is for a time t in the sequence. With the output for time t as $\mathbf{h}_t$ the equation can be written as

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{H}\mathbf{h}_{t-1} + \mathbf{b}) \tag{1}$$

where all capital letters represent variables which are determined during training [3]. There is also a bias added called b which is determined through training as well. The σ is a function used to determine whether there is an activation or not and is set before calculation. In the case of classification, the output can, in turn, be used to calculate the probabilities, $\mathbf{y}_t$, of the specific classes at a time t. It is further explained in [3] that this can be

done by using a softmax function after further weighting and addition of a bias

$$\mathbf{y}_t = softmax(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \tag{2}$$

4.2 Recurrent neural networks with long short-term memories

The exploding and vanishing gradient problem for RNNs can be handled in different ways. One of these is to use a modification to the RNN called LSTM, where the memory unit has been changed. One of the main components of a LSTM is the memory cell, which stores the information while the gates are used to control what information should be inserted or extracted from the memory cell. The first of the gates is the forget gate, as the name suggests, it is used to decide what information in the memory cell should be discarded. The second gate is the input gate that takes care of how much of the input should be stored in memory. Lastly, the output gate affects the hidden state based on the memory cell [3].

4.2.1 Mathematical description

It is also described in [3] how an LSTM work in a mathematical sense. Let the hidden state be called h_t where t as before symbolizes that it is the hidden state for a time t and call the memory cell c_t. Further, let the input be called x_t and the input gate can then be written as

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{V}_i \mathbf{c}_{t-1} + \mathbf{b}_i) \tag{3}$$

Here the σ is an activation function, W, U, V, b are variables that are given values through training [3]. To be more specific b is a bias. In the same manner according to [3] the forget gate can be calculates as

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{V}_f \mathbf{c}_{t-1} + \mathbf{b}_f) \tag{4}$$

It is further explained in [3] that the calculation of what will be forgotten and what will be learned can be written as

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_t + \mathbf{b}_c) \tag{5}$$

where $\circ$ is the multiplication of every element seperately. The update on what is forgotten from the previous time step is calculated by the term $f_t \circ c_{t-1}$ and the new information to learn from the term $\mathbf{i}_t \circ tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_t + \mathbf{b}_c)$ where the right hand of the $\circ$ is the information, which can be

learned [3]. In [3] it is also described that with the calculated memory cell the output gate to the next time iteration can be calculated as

$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{V}_o\mathbf{c}_t + \mathbf{b}_o) \tag{6}$$

and finally the hidden state as

$$\mathbf{h}_t = \mathbf{o}_t \circ tanh(c_t) \tag{7}$$

A LSTM cell is illustrated in Figure 7. In this figure, the cell state C_{t-1} is the top-left input and the cell state C_t the top-right output. In the same way, the bottom left input is the hidden state h_{t-1} and the bottom right output the hidden state h_t.

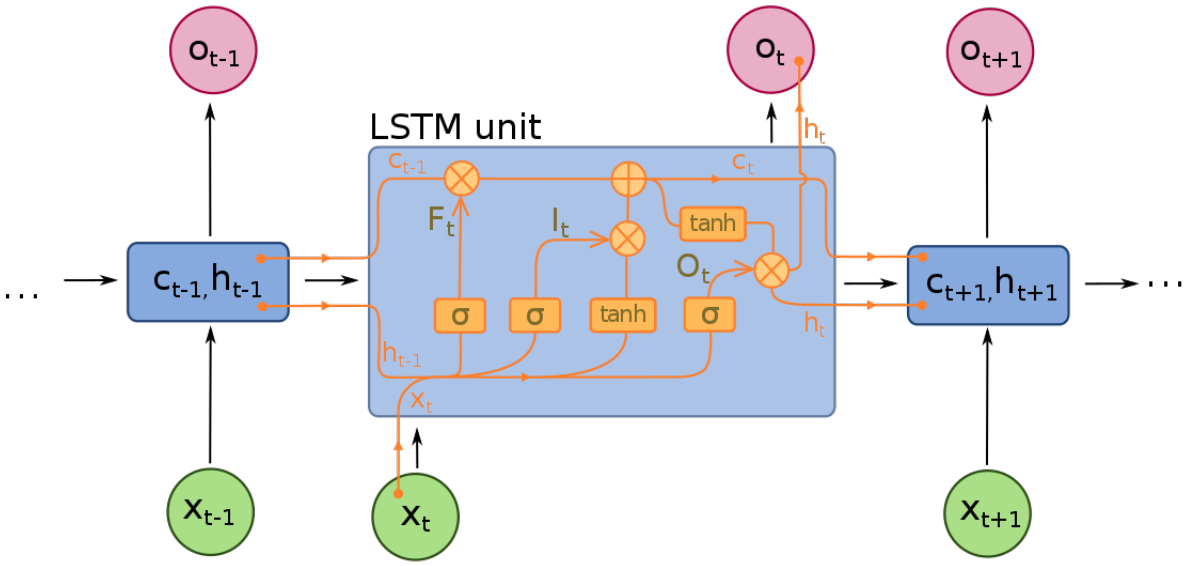

Figure 7: Long Short-Term Memory.svg. From [1], CC BY-SA 4.0.

4.2.2 Bidirectional LSTM

A LSTM processes the input from one side to the other meaning that it only processes the sequence in one time direction. A bidirectional LSTM, on the other hand, creates two LSTMs where one has the original input and the other the reversed input. The results from the two cells are then combined, creating one output [18].

The use of bidirectional RNNs and LSTMs are easy to motivate in the case of text recognition as grammar is in many cases not only depend on the past but also on the coming words. An example of this would be the use of a and an, which also depends on what the following words in the sentence are. Bidirectional RNNs and LSTMs are useful when there is a

need for context in order to make a good prediction. It could be argued in the case of driver action prediction that usually an action such as reaching for an object has to happen to some degree in a specific order and thus later events affect previous ones. For example, a driver can not grab an object without first removing at least one hand from the steering wheel.

4.3 Prediction model

An interesting subject in neural networks is sequence prediction. This where RNNs shine as they can keep information about previous states in memory giving them a better understanding of sequences. There are several different ways in which RNNs can be used in order to make a prediction based on sequence data.

The method used in this study is based on using a bidirectional LSTM which used a sequence of classifications of images as input. The bidirectional LSTM will be referred to as a LSTM for short. This LSTM predicted the next element in the sequence. The newly predicted element could then be appended to the previous input and the element representing the earliest timepoint was removed. This created sequence was the same as, if the prediction was correct, the input sequence to the LSTM one frame after the current frame. Therefore by sending this created sequence to the LSTM a prediction of two frames ahead could be achieved. This process could be repeated indefinitely, giving a prediction an arbitrary amount of frames ahead. One problem with this method was the assumption that the predictions were accurate, which is an assumption that rarely holds indefinitely. If one prediction were to be inaccurate then the input to the LSTM used to predict the next frame would not be accurate meaning that future predictions are affected by the previous ones. The measurement of how well the LSTM performed in this study was accuracy, which has been defined as the number of correct predictions on the test set at a specific time point divided by the total amount of predictions on the test set made at that time point on the test set.

4.4 Choice of model

The intended inputs to the complete system are sequences of images of drivers in a car. Images are not easy to handle as a sequence in a RNN, at least not when predicting sequences by concatenating the predictions to the input sequence to calculate further predictions. If every prediction

at this point would be an image, that would mean that the prediction of the next time step would have to be another image, which in turn can be used for further predictions. There are methods for video prediction such as PredNet presented in [19]. This type of network does on the other hand work mostly with one frame predictions. There are also examples of predictions made several frames forward with some fine-tuning. These predictions perform well but do suffer from an increasing amount of blurriness with further predictions. It is reasonable for the blurriness to occur when errors in one image propagate forward, which in turn leads to errors in subsequent images. When making predictions further ahead in time then it could be beneficial with a method which is less affected by errors made in previous predictions. Therefore, the method in this study is based on sequences of numbers rather than images with the hypothesis that the lowered dimensionality would also lower the number of errors introduced in every prediction step.

4.5 Implementation

In order to implement the neural networks created in this thesis, the programming language Python has been used. The reason for this is that Python has a library called Tensorflow, which provides many functionalities in the subject of machine learning.

The code for the implementation of the CNN has been based on the tutorial by [20]. The implementation of the LSTM network has been based on the tutorial by [21].

5 Training

This section covers other aspects of the networks used in this report such as the dataset used, the choice of optimization method and optimization of hyperparameters.

5.1 Dataset

Training the network requires a dataset where the images already have been classified. This project specifically requires a dataset with sequences of images of drivers performing actions inside of a car. As datasets are often costly and time-consuming to produce in addition to the constraints due to the privacy of the participants, most companies do not publish their datasets.

Among the few public datasets that could be used is the dataset created by Y. Abouelnaga, H. Eraqi, and M. Moustafa [22]. It contains around 13000 training images and around 4300 test images. There are ten different classifications in this dataset: Drive Safe, Talk Passenger, Text Right, Drink, Talk left, Text Left, talk Right, Adjust Radio, Hair & Makeup and Reach Behind. The dataset was originally used to classify the driver's actions based on single images and therefore is not focused on sequences. The classifications are that of people who are distracted in different ways, while this project requires classifications of actions leading up to the point where the driver is distracted. Therefore this dataset cannot be used for this project.

Another possibility was to create a new dataset. This required resources both in the form of equipment as well as participants. The upside was that the dataset could be tailored to the needs of the project, which made it possible to create a dataset where the focus is on the time before and the process of when the action is taken.

The collection of the dataset used in this report was performed at RISE Viktoria. It consists of video films of people sitting in the front seat of a car performing several tasks. There were a total of eight different people and one car used. Due to safety reasons, the car was parked while the data was gathered. There were 13 tasks, which each of the participants performed. The first two tasks were to start off by driving normally and then in an as natural way as possible pick up a mobile phone or a water

bottle. There were then three tasks where the driver was to go from driving safely to remove a hand from the steering wheel, glancing or leaning into the car respectively and then going back to driving safely. The last eight tasks consisted of going from driving safely to either picking up a bottle or a mobile phone but performing it in a specific sequence for example: drive safe, glance, lean, remove hands, pick up the bottle. Each task was performed five times by each participant and lasted for around five seconds. For the first six participants, the data was collected at 30 fps and the last two at 25 fps due to technical problems. A total of nearly 90000 frames was collected. In order to get clearer data, the participants were told to neither move to fast or excessively slowly.

During annotation, eight different classes were chosen: Drive safe, glance, lean, remove hands from the steering wheel, reach, grab, retract and holding an object. It is worth mentioning at this point that every frame where only given one classification i.e. that one frame could as an example not both be classified as glance and lean. In practice, it is not uncommon that more than one of these actions were performed simultaneously. This was a problem during annotation and was solved by implementing a priority system. In this system among the classes drive safe, lean, glance and remove hands from the steering wheel the action that was taken last was chosen as the current class. Also lean had priority over glance as they very often appeared together. The classes reach, grab, retract, hold object had priority over drive safe, glance, lean and remove hands from steering wheel.

During the training and evaluation processes of the networks, the dataset was divided into 2 parts; one for training and one for testing. The reason for this is that the data used for testing should be new to the model. If the test data has been used for training as well the network will already have adapted to this data and likely performs better than it would in a scenario when used in practice.

5.2 Backpropagation and gradient descent

The intention of training neural networks is to update the weights and biases of the network in every training iteration to better mimic the behaviours ingrained in the inputs. One method to perform this training step is to use some form of optimizer such as gradient descent with backpropagation. This method uses a cost function, which gives an indication of

how close the given output is to the targeted output. It is common to use a mean squared error (MSE) based type of cost function [23].

Consider Figure 8, which depicts three dense layers where the last is the output of the network. Given a target output, a cost function and a received output from the network the cost can be calculated. With this in mind, the cost could then in many cases be reduced by changing the output. In the case of a MSE cost function, the cost can be reduced if the output from the network becomes closer to the targeted output. The question which arises from this, on the other hand, is how the output could be changed.

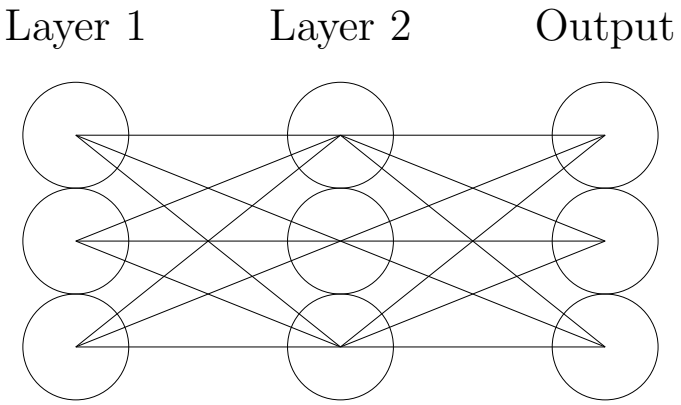

Figure 8: A simplified neural network which only shows three dense layers where the last is the output. The circles are the neurons and the lines the connections between the neurons with their individual weights.

As mentioned before an optimizer such as gradient descent is used to perform the training. This is done by optimizing the cost function with an iterative update to the weights. The thought behind gradient descent is to move along the opposite direction of the gradient for the function [24]. Backpropagation is used to calculate said gradients and works by moving backwards through the network and describing the gradient of the cost function in each step as a function of a specific weight [23].

It is explained in [25] that when the number of weights grows large and there is a need to save computational power, another type of gradient descent called stochastic gradient descent can be used. In this method, the update rule is changed to update with one partial gradient. With this update rule, only one partial gradient needs to be calculated before iterating. On the other hand, there is no guarantee that the weights will completely

reach their minimum, not even a local minimum.

Another similar method is Adam. One of the main differences is that this method does not only use the gradients but creates a momentum. This algorithm was proposed in [26]. The momentum is created by also taking into account previous iterations.

5.3 Activation function

Gradient descent depends on the weights and biases of the network, it can also depend on activation functions of the network. The activation function is meant to separate low-intensity inputs from high-intensity inputs. One function that can do this is a step function, which makes a binary separation i.e. all values below a certain threshold value will not result in an activation and all above will give an activation [27].

For gradient descent to work there needs to be gradients of the cost function. In turn, this requires gradients of the weights and activation functions. Therefore the intuitive choice of a step function as activation function does not work. Other options that are similar but have gradients are the sigmoid, tanh and ReLU functions [27].

5.4 Regularization

Due to training data being a finite resource, there is a risk of overfitting when using larger networks with many hidden units. Hence there is too much room for the weights to adapt to the specific case of the training data. Therefore, the network can perform well on the test data but when it is subjected to new test sets the lack of generality results in a poor performance [28]. In order to mitigate the effects of overfitting, a dropout layer for every dense layer has been used in the CNN. This type of layer randomly chooses neurons from the previous layer and set their output to zero. This process is repeated for every training step, resulting in different nodes being temporarily disconnected. The dropout is only used during training as it could have a negative effect if used while testing.

5.5 Bayesian optimization

Bayesian optimization is a method which can be used to make optimizations of black box functions. Rather than making an optimization of the function itself, a model of the function is made. This model can be created in different ways but the one used in this report is based on Gaussian processes. This means that the model is not a perfect replica of the black box function but is meant to give a representation of likely values for the function to take. At first, a few points of the function is evaluated, potentially randomly chosen. Then the areas between the known points are estimated as a Gaussian process. The next step is to choose an acquisition function, which is used to calculate the point on the model that is the best to evaluate next. Whether the point is the best or not is a balancing between giving more information by evaluating at a point close to previous optimum or unexplored areas [29].

5.6 Hyperparameters

When creating or training a neural network, there are some parameters that unlike the weights and biases has to be set rather than learned. These parameters are usually referred to as hyperparameters and consists of among others the structural parameters such as the number of hidden units and layers or training parameters [29].

The hyperparameters of the LSTM that have been optimized in this thesis are the learning rate, amount of units in each layer and the number of layers. Where the learning rate determines how large steps, for each iteration, the optimizing algorithm will take during training. Larger learning rates result in larger step sizes, which makes the algorithm reach a minimum faster. If the learning rate is too large there is a risk that the algorithm will not converge to a minimum value instead it moves around the minimum point [24].

The amount of hidden units in the LSTM determines how many weights there will be and thus increases the complexity. This increase leads to more ways for the network to adapt to the training data as mentioned in [28]. One downside, on the other hand, is that the network will have a higher risk of overfitting. The amount of layers in the LSTM also affects the complexity of the network. This is due to how each layer gives more

possibilities to store data as well as processing previous data [30].

5.7 Preprocessing

Due to restrictions on memory usage, it was not preferable to perform an extensive optimization of the training process of the actual full-size training data. Therefore, the size of the input images was converted into grayscale and down-sampled. The network was then trained and tested on the down-sampled data. The original images from the dataset are 1240x720 pixels in size. After down-sampling, the size was set to 512x256 pixels. This is not the exact ratio between width and height as the original image. The max-pooling layers divide the number of elements from the input to the output by four therefore for every max-pooling layer the total amount of pixels from the input image will be divided by four. Problems can arise when the number of pixels can not be evenly divided and thus it is preferable to have sizes, which are powers of two.

6 Results

The CNN was tested for a fixed set of hyperparameters both for the basic CNN as well as the one with the addition of optical flow. The RNN with LSTM was tested for a variety of hyperparameters and the ones giving the best results were then tested again with a longer training period. Where tested means that the network was trained on a data set and evaluated on the corresponding test set. Additionally, the RNN with LSTM was tested on the ground truth data for comparison. Different lengths on the predictions were also tested.

6.1 Performance: classification

The classification was first tested with the model described in the method, which consisted of convolutional, max pooling, dense and dropout layers. The model with the addition of optical flow vector fields was then tested as well. The dataset was divided into a training and a testing set. The testing set was created out of the first ten out of 65 sequences from each person. These sequences corresponded to the tasks where the participants were told to start by driving safely and then in a way that felt most natural to them pick up the object and bring it to them. These sequences were made to create scenarios where the sequence was supposed to be closer to the way it would be performed in a real-life case.

6.1.1 Plain model

The best performance for the first model achieved an accuracy of 69% after two epochs of training out of a total of ten. Where an epoch is a training iteration for the network. In Figure 9, the y-axis represents the accuracy of the CNN and the x-axis the number of epochs trained. The hyperparameters used can be seen in Table 2. The sizes of the convolutional layers were inspired by AlexNet [31]. As can be seen in the figure, there is very little improvement after ten epochs of training.

Table 2: The hyperparameters used when computing Figure 9. The first column is the hyperparameters the second column are the used values.

Parameter	Used
Learning rate	0.01
Convolutional layers	4
Dense layers	3
Size of dense layers	1000
Dropout	0.5
Filters	5, 10, 20, 20
Filter size	11, 5, 3, 3

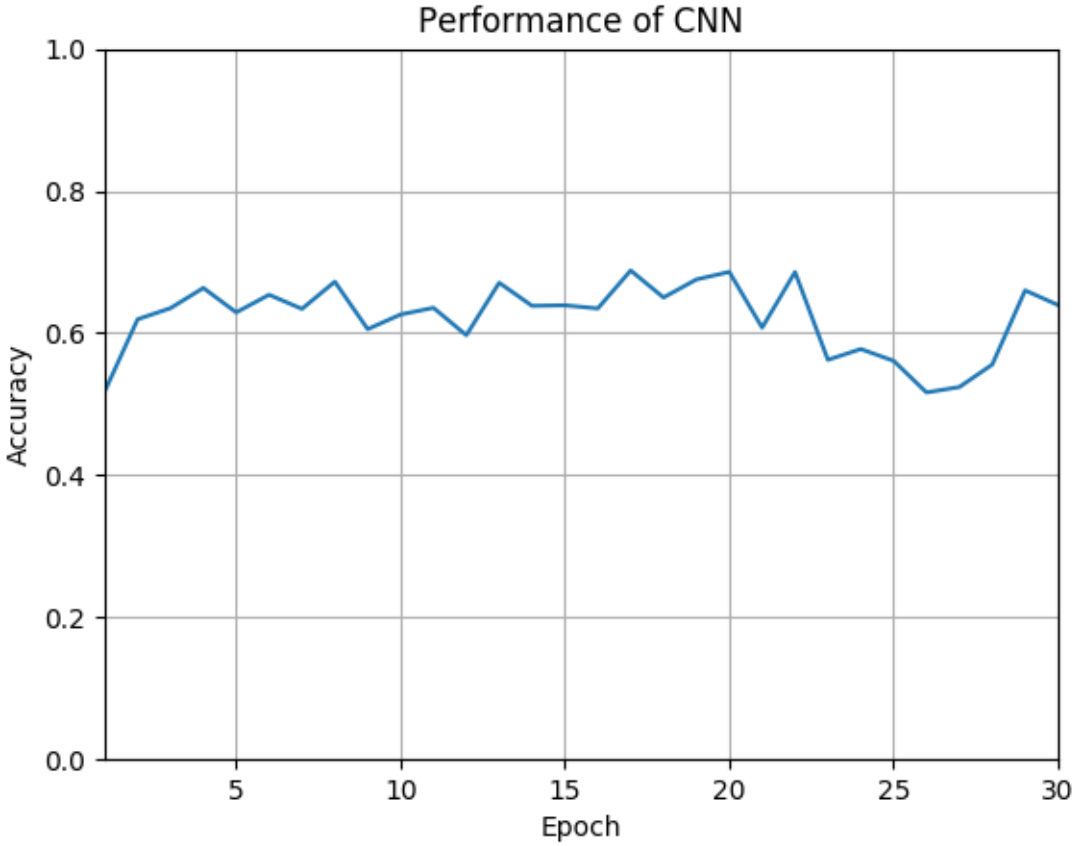

Figure 9: The performance of the CNN. The y-axis is the accuracy of the classifications and the x-axis is the number of epochs of training. Hyperparameters used for the CNN can be found in 2

6.1.2 Optical flow model

The second model which used optical flow, as well as the images, managed to get an accuracy of 80% after twelve out of 30 epochs of training. The performance can be seen in Figure 10, where the y-axis represents the accuracy and the x-axis the number of epochs trained. The hyperparameters

were the same as for the first model and can be seen in Figure 2.

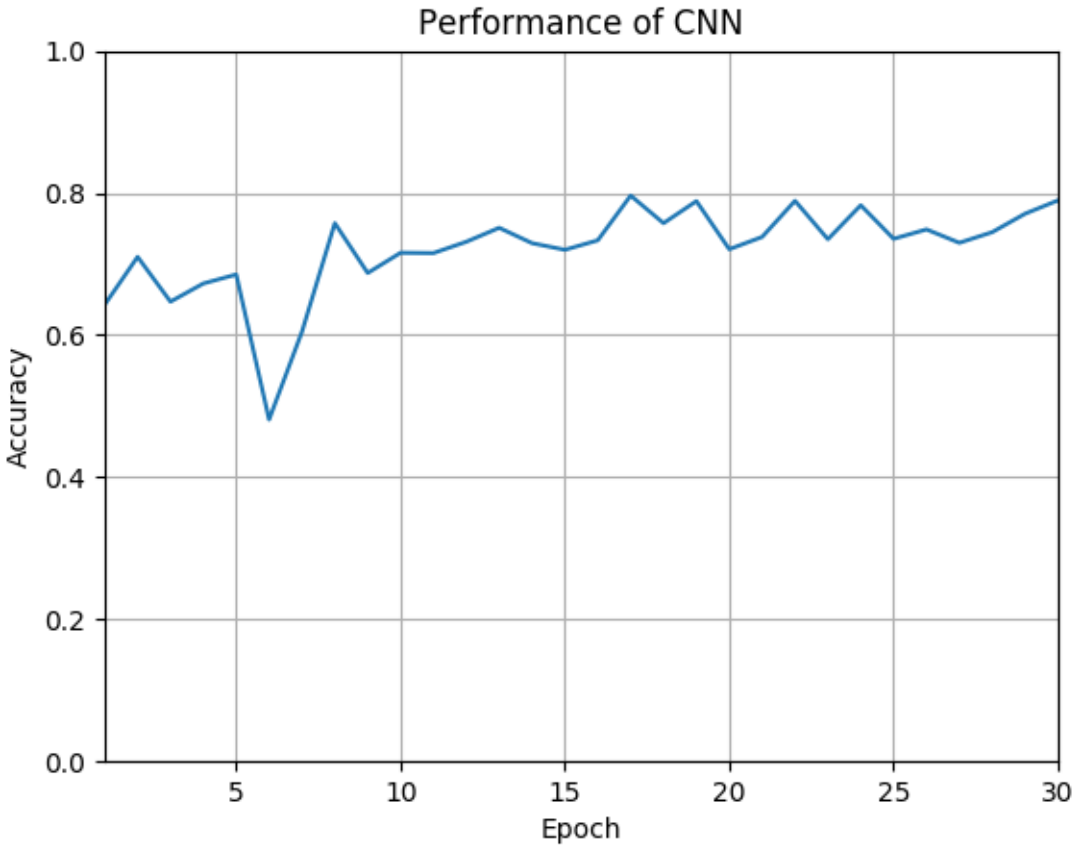

Figure 10: The performance of the CNN with the addition of parallel optical flow input. The y-axis represents the accuracy of the classifications and the x-axis the number of epochs of training.

6.2 Performance: Prediction

The CNN model using optical flow performed better than the plain model and was thus the one used in the testing of the LSTM. This LSTM was trained on the same ground truth data that the CNN was and then tested on the classifications of the test images generated by the CNN.

6.2.1 Prediction with CNN data

To see the performance of the whole network from image to predictions, the network was trained on the same ground truth data as the CNN and then tested on the labeled data from the CNN. The results from the respective best epochs of three sets of hyperparameters are shown in Figure 11. Where

the y-axis is the accuracy and the x-axis the amount frames ahead of time the prediction is. Bayesian optimization was used with four random sets of hyperparameters and then six optimization steps. Each optimization consisted of five epochs. The best performance in the sense of highest average accuracy on the interval 1 to 20 frames prediction was with the hyperparameters shown in Table 3.

Table 3: The hyperparameters used when computing Figure 11. Parameters is the hyperparameters. Set:1 to 3 are the hyperparameters in that respective set in 11. Min is the lowest values possible in the optimization and max are the highest possible values in the optimization.

Parameter	Set:1	Set:2	Set:3	Min	Max
Hidden units	79	50	150	50	150
Layers	2	1	1	1	2
Learningrate	0.02	0.00001	0.00001	0.00001	0.02

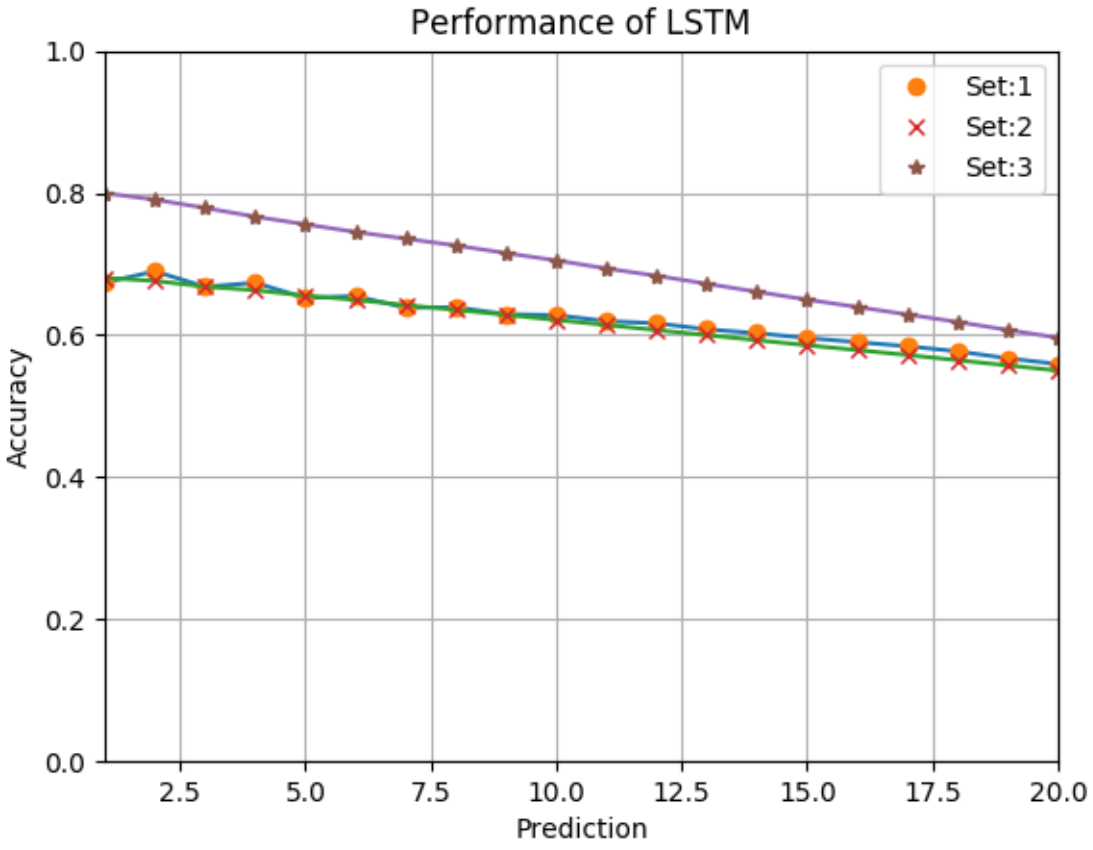

Figure 11: The performance of the LSTM network for different sets of hyperparameters. The y-axis is the accuracy of the predictions and the x-axis is the predicted amount of frames ahead of time. The network was trained on the same ground truth data as the CNN and then tested on the output classifications from the CNN. The three sets of hyperparamters show in the firgure, which include the best performing set, as well as the range can be found in Table 3

6.2.2 Extended training

The best set of hyperparameters were then trained for 20 epochs and Figure 12 shows intermediate results, where the y-axis is the accuracy of predictions and the x-axis the number of frames predicted. There are in total four functions, the straight line is for comparison and represents the case if the predictions would be to set the same as the previous value in the sequence. The other three lines represent 7, 10 and 14 epochs of training where 14 was the maximum. As can be seen in Figure 12 the best performance was from seven epochs of training with an accuracy after one frame of 80% and 62% after 20 frames. The LSTM network does better than the comparison case for every frame and the margin grows for longer time predictions.

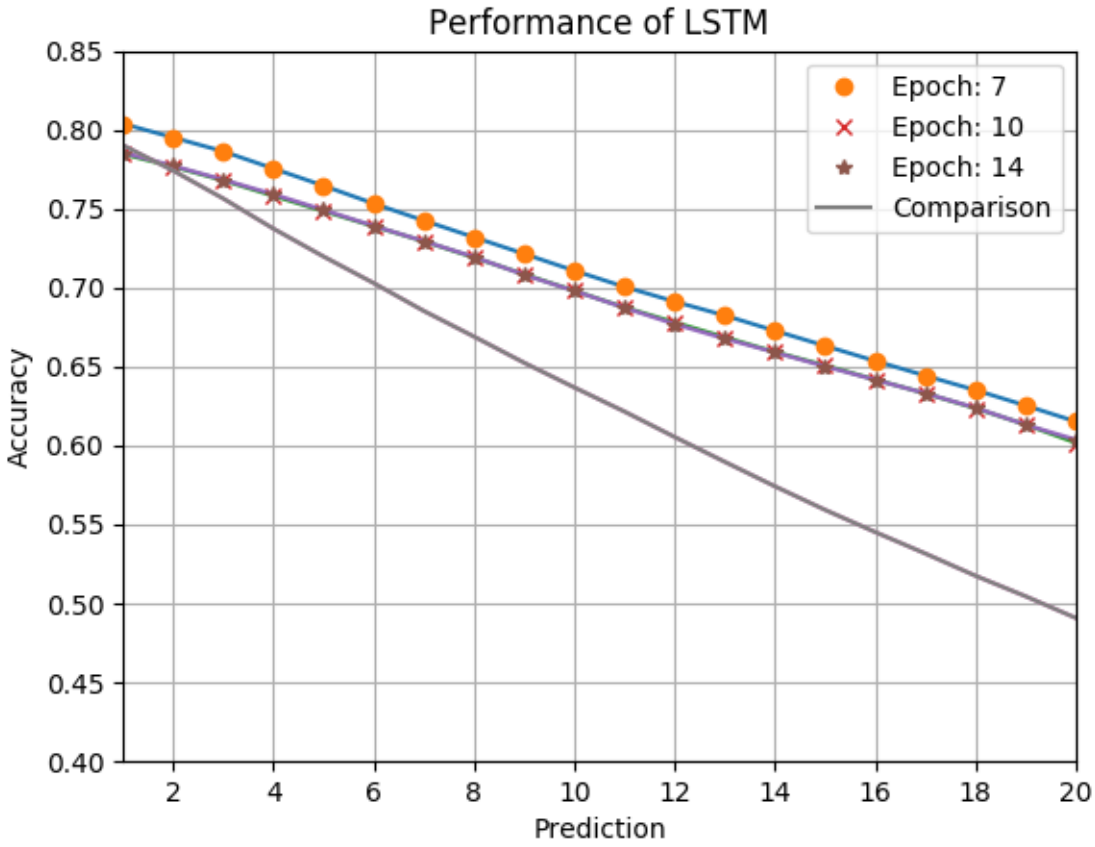

Figure 12: The performance of the LSTM network. The y-axis is the accuracy of the predictions and the x-axis is the predicted amount of frames ahead of time. In this case, the hyperparameters are not set for each of the predicted sequences. Instead, the figure represents the predicted sequences for three different amounts of epochs of training between 1 and 14. Also, the straight line represents the case where the predictions would have been set to the last value of the input sequence. The performance is for input data from the classifications from the CNN.

6.2.3 Prediction with ground truth data

For comparison purposes, the LSTM was also tested with the ground truth data. In this test, Bayesian optimization was used to optimize the hyperparameters and four random combinations were calculated followed by six optimization steps. The results can be found in Figure 13. The accuracy in this case compared to the previous one is significantly higher for predictions of earlier frames but rather similar for long time predictions. The hyperparameters giving the best result for average accuracy can be seen in Table 4.

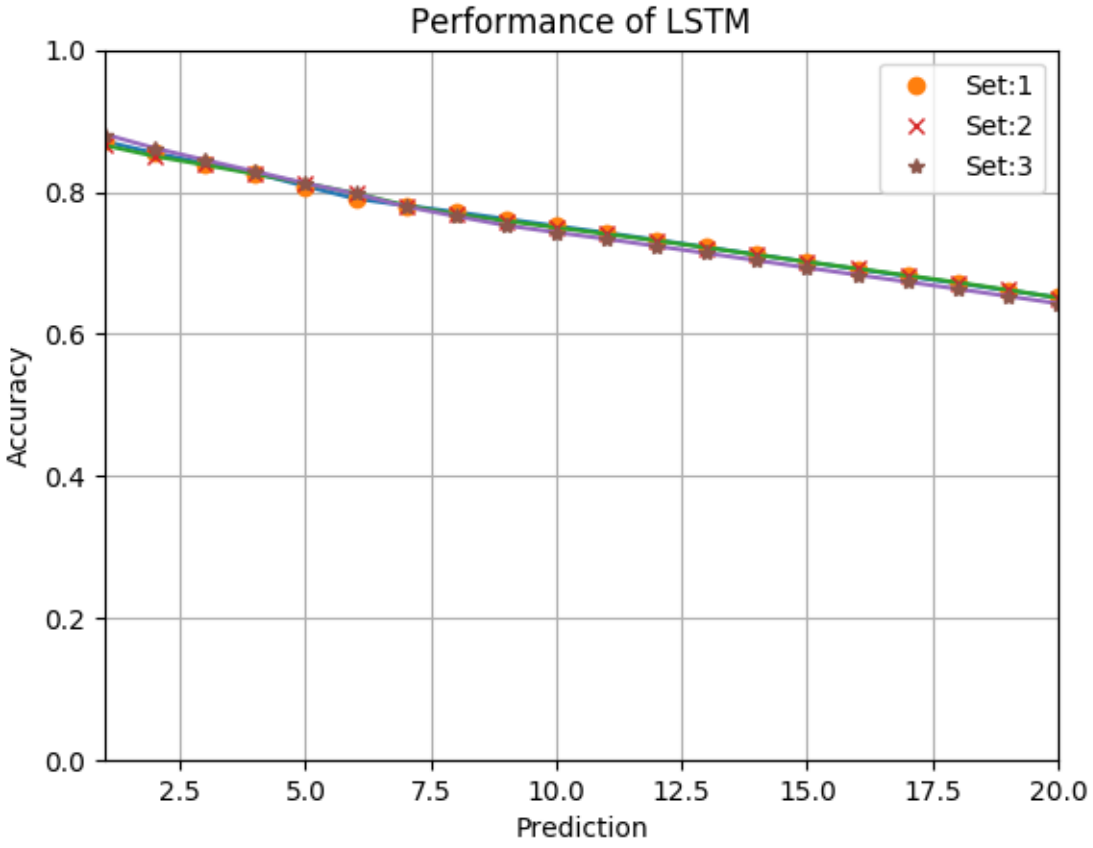

Figure 13: The performance of the LSTM network. The y-axis is the accuracy of the best performing epoch for each of the three sets of hyperparameters used for the predictions and the x-axis is the predicted amount of frames ahead of time. The different lines represent different sets of hyperparameters. The performance is for input data from the ground truth. The hyperparameters used for these three results as well as the range can be found in Table 4.

Table 4: The hyperparameters used when computing Figure 13. Parameters is the hyperparameters. Set:1 to 3 are the hyperparameters in that respective set in Figure 11. Min is the lowest values possible in the optimization and max are the highest possible values in the optimization.

Parameter	Set:1	Set:2	Set:3	Min	Max
Hidden units	50	144	149	50	150
Layers	1	2	1	1	2
Learningrate	0.00001	0.00839	0.00700	0.00001	0.02

6.2.4 Extended training with ground truth data

The best performing set of hyperparameters for the LSTM tested on ground truth data was also trained for 20 epochs and the results can be seen in Figure 14. In this figure, the y-axis is the accuracy and the x-axis the number of frames predicted ahead of time. The straight line as before represents the case when the predictions are set to the last class of the input sequence. The other lines are the networks performance for 10, 15 and 20 epochs of training, where 15 epochs of training yielded the best result. The best performance was an accuracy of around 87% for one frame predictions and 65% at 20 frames, with an average of 76% over all frames. In this case, the comparison performs better for the very first frames but significantly worse for later predictions.

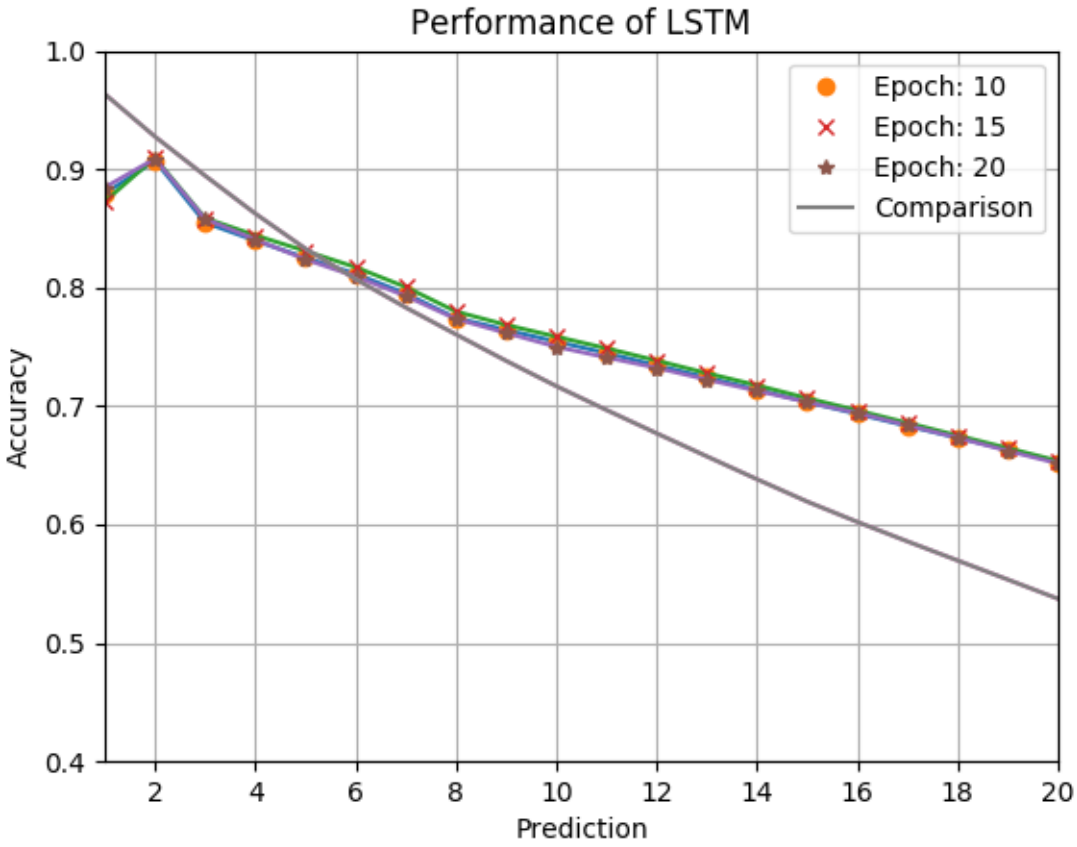

Figure 14: The performance of the LSTM network. The y-axis is the accuracy of the predictions at each specific time frame and the x-axis is the predicted amount of frames ahead of time. In this case, the hyperparameters are not set for each of the predicted sequences. Instead, the figure represents the predicted sequences for 10, 15 and 20 epochs of training as well as a straight line representing the case where the prediction is set to the last element in the input sequence. The performance is for input data of the ground truth, not the CNN.

6.2.5 Variations of output lengths

The LSTM was also tested with three different lengths of the output, namely 10, 20 and 30 frames ahead of time and is illustrated in Figure 15. The best performing hyperparameters in Table 4 was used and each of the networks were trained for ten epochs. The results are that of the best performing epoch of training. The y-axis represents the accuracy and the x-axis the number of frames predicted. Observations made during training seemed to suggest that the oscillatory pattern occurred due to an insufficient amount of training epochs.

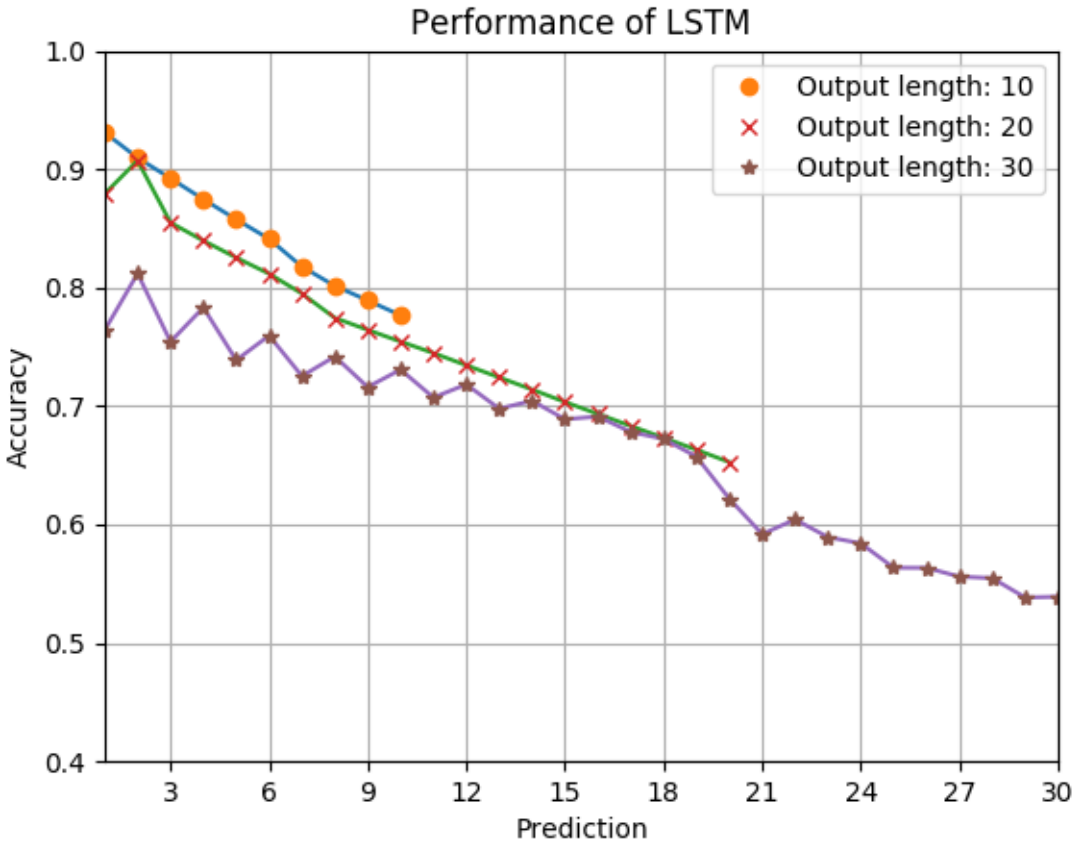

Figure 15: The performance of the LSTM network. The y-axis is the accuracy of the predictions and the x-axis is the predicted amount of frames ahead of time. The lines represent the best performance for three different cases of lengths of outputs namely 10, 20 and 30 frames predicted ahead of time. These results are that of the best epochs during ten epochs of training on the best performing hyperparameters in Table 4.

6.3 Confusion matrices

In order to study the interaction between different classes of the LSTM three confusion matrices were created. The rows in these confusion matrices represents different predictions for the LSTM, while different columns represent the same classes for the ground truth. Each element in the matrix thus belongs to a class for the prediction and a class for the ground truth. Each element represents the number of times that given class of the ground truth has been predicted as the class from the prediction. Both the ground truth and the prediction starts with the class drive safe then glance and so on meaning that the diagonal of the matrix represents the cases where the prediction is correct. A perfect performance of the network would be illustrated with a value of 100 in every element of the diagonal. The matrices used here are normalized along the column direction. Each

of the confusion matrices in Tables 5, 7 and 8 were generated from the best performing epoch of the LSTM used in Figure 14. The number of labels for each class used when calculating the confusion matrices can be found in Table 6.

The first matrix is shown in Table 5 was created by comparing the results of the prediction with the ground truth for one time frames prediction. For many of the different classes, the diagonal is the majority with the exception of grab. Most of the predictions are either correct or in an adjacent class. The best performing classes are drive safe and reach, while grab and retract performed worst.

		Ground truth							
		Drive safe	Glance	Lean	Remove hand	Reach	Grab	Retract	Hold
Predicted	Drive safe	0.98	0.09	0.07	0.12	0.0	0.01	0.01	0.37
	Glance	0.0	0.84	0.0	0.05	0.0	0.0	0.0	0.0
	Lean	0.0	0.0	0.8	0.0	0.0	0.0	0.0	0.0
	Remove hand	0.0	0.02	0.13	0.8	0.02	0.01	0.0	0.0
	Reach	0.0	0.05	0.0	0.02	0.97	0.25	0.0	0.0
	Grab	0.0	0.0	0.0	0.0	0.0	0.25	0.0	0.0
	Retract	0.0	0.0	0.0	0.0	0.0	0.48	0.6	0.01
	Hold	0.02	0.0	0.0	0.0	0.0	0.0	0.39	0.62

Table 5: A confusion matrix over the predictions after one time frame. The elements in the off-diagonal represents the wrong classifications of a class as the class of the corresponding row. Each column represents another class for the ground truth and each row another class for the prediction. The predictions are from the best performing epoch in Figure 14.

The second matrix, which can be seen in Table 7, from predictions ten time frames ahead. The values are now starting to shift away from the diagonal with the exception of drive save and reach. Glance, Lean, retract and remove hand now performs significantly worse.

Table 6: The amounts of labels for each class when determining the confusion matrices.

Drive safe	Glance	Lean	Remove hand	Reach	Grab	Retract	Hold
4904	265	15	332	1782	186	536	1180

		Ground truth							
		Drive safe	Glance	Lean	Remove hand	Reach	Grab	Retract	Hold
Predicted	Drive safe	0.97	0.68	0.7	0.6	0.1	0.0	0.04	0.36
	Glance	0.01	0.26	0.3	0.15	0.05	0.0	0.0	0.0
	Lean	0.0	0.01	0.0	0.0	0.01	0.0	0.01	0.0
	Remove hand	0.0	0.04	0.0	0.21	0.1	0.02	0.03	0.0
	Reach	0.0	0.0	0.0	0.02	0.74	0.96	0.47	0.02
	Grab	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Retract	0.0	0.0	0.0	0.0	0.0	0.0	0.01	0.0
	Hold	0.02	0.0	0.0	0.03	0.0	0.02	0.43	0.61

Table 7: A confusion matrix over the predictions after ten time frames. Each column represents another class for the ground truth and each row another class for the prediction. The predictions are from the best performing epoch in Figure 14.

The third matrix in Table 8 was created from predictions 20 time steps ahead. Only drive safe and reach remains with a majority in the diagonal while hold is slightly below. The performance of the other five classes is at this point very low or 0.

		Ground truth							
		Drive safe	Glance	Lean	Remove hand	Reach	Grab	Retract	Hold
Predicted	Drive safe	0.96	0.87	0.43	0.82	0.32	0.0	0.0	0.4
	Glance	0.01	0.05	0.57	0.04	0.09	0.01	0.0	0.0
	Lean	0.0	0.01	0.0	0.0	0.01	0.0	0.0	0.0
	Remove hand	0.0	0.06	0.0	0.11	0.07	0.03	0.02	0.0
	Reach	0.01	0.0	0.0	0.02	0.51	0.97	0.87	0.19
	Grab	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Retract	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Hold	0.02	0.0	0.0	0.02	0.0	0.0	0.11	0.41

Table 8: A confusion matrix over the predictions after 20 time frames. Each column represents another class for the ground truth and each row another class for the prediction. The predictions are from the best performing epoch in Figure 14.

7 Discussion

The following section gives a deeper explanation of the different factors affecting the results, such as the dataset. Also, some possible areas of future work, which potentially could improve the networks used in this report, are presented.

7.1 Dataset

As can be seen in the results the network struggles to maintain a high accuracy for longer time predictions. One possible way to improve this is to instead of using the structure of priority, in the dataset, to use additional classes. These extra classes could cover combinations of the classes used. The solution does have its advantages. As an example, it could likely make the CNN network perform better as the CNN merely looks at frames and not sequences of actions, which makes it hard to determine what action has been taken last. By creating hybrid classes and abandoning the priority system, the CNN would therefore likely perform better. With more classes, on the other hand, the complexity of the LSTM increases. This increase in complexity could be a good thing as more classes give rise to more different combinations of sequences for the network to learn, giving it more information to make better decisions. Therefore, it could potentially increase the performance of the whole system (both CNN and LSTM). An extreme example of this would be to consider a case where there are only two classes; drive safe and distracted. Given this little information, the system will likely consider the transfer from drive safe to distracted as random unless the drivers are very periodic in the way they are distracted. With this in mind, it is easy to see that with an increase in the number of classes used, there are more and more patterns arising. These patterns, in turn, can be of use for the LSTM to improve the accuracy of the predictions. The downside on the other hand with this increase in complexity is that there is a greater need for training data for the network to perform well. The problem also arises that the classes would become more similar to each other and thus harder to separate.

By making another extreme example, if there would be an almost ”infinite” amount of classes there would with limited training data come sequences of classes, which has been in very few of the training examples. These sequences will be almost unknown to the system. As there were limited amounts of public datasets with sequences of drivers, the amount of

data used in this project was limited and therefore the priority system was used rather than a hybrid class system as it was expected to give better performance with a small dataset. It can also be mentioned that the more classes used, the less likely it is to make a correct classification or prediction as there are more classes to chose from. This can be problematic for the predictions since they, as mentioned before, are susceptible to a cumulative error where one wrong prediction can lead to further predictions also being wrong.

7.2 Hyperparameters

There are more hyperparameters in addition to those already mentioned which have not been optimized. Some of these are batch size and the type of optimizer. The batch size could potentially change the performance of the network but was chosen to be a set value due to how it greatly affects the memory usage while training.

The hyperparameters used to get the best performing LSTM networks in Figure 7 and Figure 13 were in the first case close to the largest possible values for the number of units. Due to this, there might be a possibility to further increase the performance by testing more units and layers. This increase will on the other hand also increase the memory usage, which might lead to the reduction of other parameters to compensate such as batch size. On the other hand, a decrease in batch size could also affect performance. In the second case the performance was very similar for different sets of hyperparameters.

7.3 Performance: CNN

It was clearly demonstrated that the first model of the CNN which did not use optical flow was, in this case, inferior to the model utilizing optical flow. A plausible explanation for this is that the dataset was created with functionality for sequences in mind, not classifications. The use of optical flow, in this case, gives information to the CNN of movements, i.e. the CNN is also working in the dimension of time. It could, therefore, be possible that it is easier to detect which action has been taken last when using optical flow.

It was mentioned before that the CNN performs poorly with this type of dataset, which is created for sequences rather than classification. One

reason for this could be due to how hard it can be to distinguish between a reaching person and a retracting person from a still image. In the case of optical flow, on the other hand, these scenarios are close to opposites making them easy to separate. This extension of the model does based on the results in Figure 9 and Figure 10, enhance the classification accuracy of the CNN.

There was also the problem arising from a dataset of sequences captured with a moderate or high fps that there comes a point in almost every transition from one classification to another where two almost identical images are given different classes. Due to this, it is not only hard for the network to make correct classifications but also for the human who labels the ground truth data.

7.4 Performance: LSTM

The accuracy of the predictions, as can be seen in Figure 14, was as expected almost dropping for every frame, which represents a lowered accuracy for longer time predictions. As mentioned before, this decrease in accuracy was probably due to a cumulative error where one incorrect prediction gave the following predictions an input, which deviated from the true input and thus increased the risk of further incorrect predictions.

The difference in performance between the LSTM tested on ground truth data and the one tested on the labels from the CNN was larger for shorter predictions and diminished with time. The large difference for shorter predictions might be due to earlier predictions being more influenced by the accuracy of the last elements of the input sequence.

From Figure 15 it is hard to draw any conclusions but the performance of the model using an output length of 10 appeared to perform best. This could possibly be due to either faster training or better performing initialization of weights. The performance with longer output sizes might be improved with more training epochs as an increase in the output size also increases the complexity of the network.

The result shown in Tables 5, 7, 8 indicate a trend of deteriorating performance for all classes in longer predictions. Some of these classes became more incorrect faster than others and some go as low as 0% accuracy. The three classes that performed the best drive safe, reach and hold were the

ones appearing with the highest frequency in the dataset. One possibility for why the other classes tended to be predicted as one of these three classes could be that the over-representation led to a bias in the prediction. This bias, in turn, increased the possibility of a prediction to become one of those three classes for each prediction. Therefore the underrepresented classes would gradually be predicted as the over-represented classes. This, in turn, led to an increase in the bias.

In order to solve this problem potentially, the bias could be corrected but this will also lead to problems. The reason for the bias is that some of the classes are more likely to appear than others. By removing it the underrepresented classes might perform better over time but it is also likely that the performance overall will be worse. With a more even spread of predictions over the classes the accuracy of the classes that perform well might become worse and as those are the classes with the largest impact it can have a negative effect overall.

7.5 Future work

In this section, some possible future work is discussed for the CNN, LSTM as well as for the complete network.

7.5.1 CNN

The CNN network used to classify the images could probably be improved by implementing some of the features of the networks in ImageNet Large Scale Visual Recognition Challenge. GoogLeNet introduced a structure of parallel layers with different structures. These layers were ordered so that there was convolutional layers of size one, three and five parallel to each other and one combination of a pooling layer and a convolutional layer. These blocks of layers followed after each other with softmax layers in between. These blocks are called inception modules. There were other types of blocks as well in GoogLeNet such as larger ones where the convolution layers forked out into two more. Within the inception module, there was also a convolutional layer of size one in each of the parallel layers. The addition of the one size convolutional layers was to reduce the number of filters in each layer among others [32]. This type of structure has proven to be very useful as it won the ImageNet competition in 2015 [4].

Another source of potential improvement is to further process the input

images. This study has used downsampling and conversion to grayscale but could also use processing methods such as normalization and whitening. Using these types of tools could increase the performance of the feature extraction [33].

7.5.2 LSTM

The LSTM network could possibly be improved with the combination of different models of networks. One example is to combine several networks, which have varying frame rate. It could be argued that there are benefits in reducing the frame rate as it provides fewer prediction steps resulting in fewer opportunities for errors to occur. The combination could be performed before the softmax. In that case, either the model's probabilities for the different classes could be added together or the highest one across all models could be chosen.

7.5.3 Extensions of the networks

One future prospect would be to in a similar fashion as in [3], [6] not only use video data but also make use of other types of information such as cameras directed forward out on the road. In the source articles, it was used to predict major movements of the car such as the driver changing lane or breaking but these types of information can also be useful when trying to predict what is happening inside of the car. One example of when it could be useful is for example that the information gained could help understand why a driver is distracted.

8 Conclusion

A LSTM has been tested for the use of prediction of driver actions inside of a vehicle. The network uses sequences of previous actions in order to predict future ones. The previous actions are generated by a CNN, which takes as an input images of the driver. The input images are from sequences from two different cameras located inside of the vehicle. The images from these two cameras are processed in parallel convolution layers and then concatenated in the dense layers. The dataset used was divided into two parts a training and testing set.

Also, another model of CNN has been tested, which uses optical flow. From the input images, the flow between two images is calculated. The optical flow input is also processed in parallel convolution layers resulting in a total of four parallel sets of convolutional layers, which are concatenated before the dense layers.

The LSTM network was trained on the same training set as the CNN's were and then tested both using the classifications generated by the CNN's and also on the ground truth. For comparison, the accuracy of a network, which would predict every frame as the last element of the input sequence were also calculated.

The performance of the CNN that used optical flow compared to the model without performed significantly better. Where the first model reached an accuracy of 69% and the one with optical flow 80%. The LSTM network outperformed the comparison network for all frames predicted in the case of input data from the CNN. With accuracies of 80% and 62% for one and 20 frame predictions respectively. In the case where the LSTM was tested on the ground truth data, the accuracies were 87% and 65%, which outperformed the comparison for longer time predictions.

One possible improvement of the CNN could be to test some of the structures of large classification networks such as AlexNet and GoogleNet. The overall prediction accuracy could also potentially be improved by utilizing more types of input data from the vehicle or using combinations of different models.

References

[1] F. Deloche, "Long short-term memory.svg," https://commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg#file, 2017. [Electronic image], accessed: 2018-07-05.

[2] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Washington, DC: National Highway Traffic Safety Administration, February 2015, (Traffic Safety Facts Crash•Stats. Report No. DOT HS 812 115).

[3] A. Jain, A. Singh, H. S. Koppula, S. Soh, and A. Saxena, "Recurrent neural networks for driver activity anticipation via sensory-fusion architecture," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 3118–3125.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[5] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 568–576. [Online]. Available: http://papers.nips.cc/paper/5353-two-stream-convolutional-networks-for-action-recognition-in-videos.pdf

[6] O. Olabiyi, E. Martinson, V. Chintalapudi, and R. Guo, "Driver Action Prediction Using Deep (Bidirectional) Recurrent Neural Network," *ArXiv e-prints*, Jun. 2017.

[7] J. Carmona, F. García, D. Martín, A. d. l. Escalera, and J. M. Armingol, "Data fusion for driver behaviour analysis," *Sensors*, vol. 15, no. 10, pp. 25 968–25 991, 2015. [Online]. Available: http://www.mdpi.com/1424-8220/15/10/25968

[8] T. Pech, P. Lindner, and G. Wanielik, "Head tracking based glance area estimation for driver behaviour modelling during lane change execution," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct 2014, pp. 655–660.

[9] J. S. Wijnands, J. Thompson, G. D. Aschwanden, and M. Stevenson, "Identifying behavioural change among drivers using long short-term memory recurrent neural networks," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 53, pp. 34 – 49, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1369847817301183

[10] O. L. Georgeon, T. Bellet, A. Mille, D. Letisserand, and R. Martin, "Driver behaviour modelling and cognitive engineering tools development in order to assess driver sitation awareness," in *International Workshop on Modelling Driver Behaviour in Automotive Environments.*, C. Macchi, Re, Ed. Ispra, Italy: Official Publication of the European Communities, May 2005, pp. 236–241. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01536639

[11] A. Karpathy, "Cs231n convolutional neural networks for visual recognition," http://cs231n.github.io/convolutional-networks/#overview, accessed: 14 April 2018.

[12] M. Sun, Z. Song, X. Jiang, J. Pan, and Y. Pang, "Learning pooling for convolutional neural network," *Neurocomputing*, vol. 224, pp. 96 – 104, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231216312905

[13] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba, "Object detectors emerge in deep scene cnns," *CoRR*, vol. abs/1412.6856, 2014. [Online]. Available: http://arxiv.org/abs/1412.6856

[14] D. Fleet and Y. Weiss, *Optical Flow Estimation.* Boston, MA: Springer US, 2006, pp. 237–257. [Online]. Available: https://doi.org/10.1007/0-387-28831-7_15

[15] C. Liu, "Beyond Pixels: Exploring New Representations and Applications for Motion Analysis," Ph.D. dissertation, Massachusetts Institute of Technology, 2009. Accessed on: Aug. 29, 2018. [Online]. Available: https://people.csail.mit.edu/celiu/Thesis/.

[16] D. Pathak and T. Alldieck, 'Fast, accurate and easy to run dense optical flow with python wrapper', 2017. Available: https://github.com/pathak22/pyflow. [Accessed: 22- May- 2018].

[17] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the

exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012. [Online]. Available: http://arxiv.org/abs/1211.5063

[18] Z. Yu, V. Ramanarayanan, D. Suendermann-Oeft, X. Wang, K. Zechner, L. Chen, J. Tao, A. Ivanou, and Y. Qian, "Using bidirectional lstm recurrent neural networks to learn high-level abstractions of sequential features for automated scoring of non-native spontaneous speech," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, Dec 2015, pp. 338–345.

[19] R. Mahjourian, M. Wicke, and A. Angelova, "Geometry-based next frame prediction from monocular video," *CoRR*, vol. abs/1609.06377, 2016. [Online]. Available: http://arxiv.org/abs/1609.06377

[20] H. Kinsley, 'Deep Learning with TensorFlow - Creating the Neural Network Model', 2016. [Online]. Available: https://pythonprogramming.net/tensorflow-deep-neural-network-machine-learning-tutorial/. [Accessed: 20- May- 2018].

[21] M. Ezhov, 'Dynamic seq2seq in TensorFlow, step by step', 2017. [Online]. Available: https://github.com/ematvey/tensorflow-seq2seq-tutorials. [Accessed: 20- May- 2018].

[22] Y. Abouelnaga, H. M. Eraqi, and M. N. Moustafa, "Real-time Distracted Driver Posture Classification," *ArXiv e-prints*, Jun. 2017.

[23] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[24] A. Ng, "Cs229 lecture notes," http://cs229.stanford.edu/notes/cs229-notes1.pdf, accessed: 7 April 2018.

[25] G. Roughgarden, .T Valiant, "Cs168: The modern algorithmic toolbox lecture 6: Stochastic gradient descent and regularization," http://theory.stanford.edu/~tim/s16/l/l6.pdf, accessed: 7 April 2018.

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

[27] R. Rojas, *Neural Networks : A Systematic Introduction*. Springer, 1996.

[28] 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. [Online]. Available: http://arxiv.org/abs/1207.0580

[29] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 2951–2959. [Online]. Available: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf

[30] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 190–198. [Online]. Available: http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf

[31] 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. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[32] 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. [Online]. Available: http://arxiv.org/abs/1409.4842

[33] Ufldl, "Data preprocessing," http://ufldl.stanford.edu/wiki/index.php/Data_Preprocessing, accessed: 22 May 2018.