CHALMERS
UNIVERSITY OF TECHNOLOGY

Vehicle perception with LiDAR and deep learning

End-to-end prediction of the fully unoccluded state of the vehicle surroundings with convolutional recurrent neural network trained in an unsupervised manner

Master's thesis in Systems, Control and Mechatronics

HAMPUS BERG
JOHAN LARSSON

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

Master's thesis 2018:EX019

Vehicle perception with LiDAR and deep learning

End-to-end prediction of the fully unoccluded state of the vehicle surroundings with convolutional recurrent neural network trained in an unsupervised manner

Hampus Berg
Johan Larsson

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

Vehicle perception with LiDAR and deep learning
End-to-end prediction of the fully unoccluded state of the vehicle surroundings with convolutional recurrent neural network trained in an unsupervised manner
Hampus Berg & Johan Larsson

© Hampus Berg & Johan Larsson, 2018.

Supervisor: Peter Nilsson & Joakim Rosell, Volvo group trucks technology
Examiner: Lars Hammarstrand, Department of Electrical Engineering

Master's Thesis 2018:EX019
Department of Electrical Engineering
Division of Signals Processing and Biomedical Engineering
Signal Processing Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of LiDAR scan on highway scenario.

Typeset in LaTeX
Gothenburg, Sweden 2018

Vehicle perception with LiDAR and deep learning
End-to-end prediction of the fully unoccluded state of the vehicle surrounding with convolutional recurrent neural network trained in an unsupervised manner
HAMPUS BERG & JOHAN LARSSON
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous driving is an important area among today's vehicle manufacturers and their suppliers. The development of advanced driver assistance system (ADAS) features that will lead to fully autonomous driving is progressing fast. One of the most important aspects to achieve autonomous driving are through an accurate description of the vehicle's surroundings. As control algorithms, such as trajectory planning, becomes more sophisticated, a comprehensive description of the full surroundings is demanded, including partly and fully occluded objects. In this thesis a solution to accurately predict positions of surrounding objects for highway application using a recurrent neural network is proposed. The system considers raw light detection and ranging (LiDAR) sensor data to compute a final occupancy grid, resulting in an end-to-end solution. Due to the centimeter-level precision of today's LiDARs, we consider during training the LiDAR as ground truth for unoccluded objects and by simulating the input as blocked, and compare the output with a true scan, the network learns to track occluded objects. Hence, no annotated data is required. The LiDAR point clouds are projected into a 2D occupancy grid, which can be seen as an image with one channel, making computer visual techniques such as feature extractions with convolutions possible. The developed networks are built of convolutional and long short-term memory layers and tested on real-world data. We considered five different network structures, where one had the most consistent performance throughout the tests. The obtained result shows that the network is able to accurately predict the location of an object in an occlusion scenario emerged during an overtaking situation. Further, results show that the network performs at approximately 90 % accuracy score after 1.1 s of full occlusion in a highway scenario when the full grid is considered. We conclude that the developed network is able to describe the unoccluded surrounding of the ego vehicle accurately while having the ability to handle shorter times of occlusion.

Keywords: Vehicle Perception, LiDAR, Machine Learning, Artificial Neural Networks, Recurrent Neural Networks, Convolutional Neural Networks, Unsupervised Training.

Acknowledgements

The authors would like to start of by thanking our supervisors at Volvo GTT Joakim Rosell and Peter Nilsson for their help and guidance during this thesis, also our examiner Lars Hammarstrand for his help and guidance concerning machine learning. Furthermore, we would like to thank Arpit Karsolia and REVERE for support and loan of data collection vehicle such that the development could proceed.

Hampus Berg & Johan Larsson, Gothenburg, June 2018

Contents

List of Abbreviations

ADAS	Advanced driver assistance system
BPTT	Backpropagation through time
CNN	Convolutional neural network
ConvLSTM	Convolutional LSTM
LiDAR	Light detection and ranging
LSTM	Long short-term memory
NN	Neural network
RANSAC	Random sample consensus
RNN	Recurrent neural network
ROI	Region of interest
SAE	Society of automotive engineers
SLAM	Simultaneous location and localization
Volvo GTT	Volvo group trucks technology

1
Introduction

Autonomously driven heavy vehicles would make a substantial impact in todays society. As an example, in Sweden 85 % of the food transport is carried out by trucks [1] making heavy trucks common on the public roads. Automating these transport would not only reduce the transportation cost but could also substantially impact the traffic flow and safety. Continuous progress within autonomous driving have shown that realizing such features are not so far-fetched. The first features are expected to handle highway scenarios as the environment is well-structured and the actuation's consist of simpler maneuvers such as maintain lane, change lane, roadway departures and entrances. Even though the scenarios are simpler, comprehensive environment perception, tactical decision making and trajectory planning are required.

Focusing on the environment perception, plenty of research has been conducted reaching from vision based systems [2], [3] to active laser based systems [4]. These consider only the tracking task. Further research have been conducted on combinations of simultaneous localization and mapping (SLAM) and tracking using both passive and active sensors [5]. What is common with these approaches is that all represent road occupants as objects by data segmentation and data association, using almost exclusively variants of Kalman filters. But non of these solutions has yet to meet the challenge for an application on public roads.

An alternative approach that has gotten more focus during the last years is machine learning. It has been shown that for computer visual classification, convolutional neural networks (CNN) have great potential [6]. It is commonly known that plenty of research on machine learning is conducted at vehicle manufacturers, including development of neural networks capable of handling the perception task. To the best of our knowledge little research has been conducted in the field of combining laser sensor and machine learning, which is the topic of this thesis.

More specifically, this thesis will consider predicting the location of road occupants in the area along the sides and in front of a vehicle in highway scenarios using the LiDAR sensors and machine learning.

1.1 Background

The background of this thesis is based on the AutoFreight project [7], which aims on conducting research on core functionality to realize fully automated road freight transports (SAE level 4 [8]) and reviewing what legislation needs to be adapted. AutoFreight also consider how the road network could be modified to improve the accuracy and safety for achieving productive automated road freight transports. The project will make the research on vehicle data collected in the project and validate the research result using simulations, Asta Zero test track driving and fleet operator driving on the highway roads between Gothenburg Harbour and the logistic Hub in Viared, Borås. The ambition of the project and thus the expected result is to make a feasibility study on automation of commercial vehicle transportation with focus on long haul and verify and validate the use in logistic operation. The project vehicle is an A-double combination, meaning a tractor with a semi-trailer in which a dolly carrying another semi-trailer is connected, spanning a total of 32 m, see Fig. 1.1.

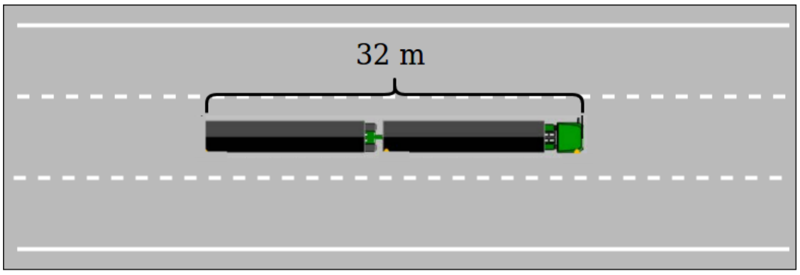


Figure 1.1: Illustration of an A-double combination which spans 32 m. Modified from [9].

In order to perform a study on automation of commercial vehicle transportation, the AutoFreight project will investigate and develop necessary systems in order for the vehicle to be in control. This covers both electrical hardware as well as software development. As mentioned, a fundamental necessity for autonomous driving on public roads is the need for comprehensive environment perception. All actions performed must be based on information such that the manoeuvre is safe and legal, e.g. functions such as trajectory planning depend on accurate information describing where obstacles and other road user are to derive an optimal trajectory.

The vehicle used in the AutoFreight project shall be equipped with numerous high-end sensors and computers to enable this study. Among these sensors are two LiDARs which are meant for the perception task that will be handled in this thesis.

1.2 Problem description

This thesis objective is to develop a neural network based system to accurately predict positions of the surrounding objects of a heavy truck travelling on highway, including temporarily occluded objects. The surroundings corresponds to both sides and in front of the vehicle, see Fig. 1.2 for illustration.

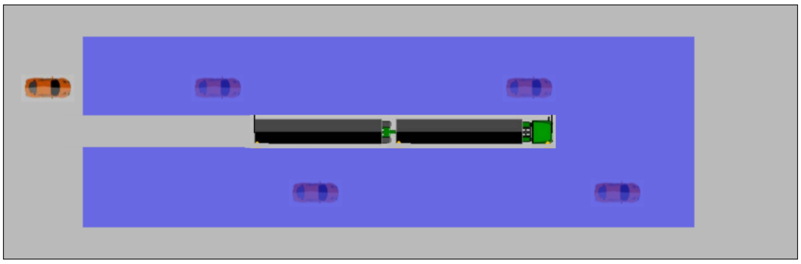


Figure 1.2: Illustration of the surrounding of interest.

The system should be end-to-end such that it considers raw LiDAR data as input and outputs the final representation of the surroundings. To achieve this objective a number of tasks must be addressed. This involves defining a suitable input and output structure to the network and transform raw LiDAR data accordingly. Further, the network must be designed to handle objects types likely to occur in such situations, both dynamic and static objects. Finally a training procedure with appropriate data set must be selected.

1.3 Related work

Interest in machine learning algorithms to solve automotive challenges has increased over the last years. In visual based systems such as camera systems, CNN have shown impressive results [10], [11], [12]. The CNN architecture is also evaluated on sensors with higher accuracy, such as LiDAR systems. In [13] a fully convolutional network is used to detect objects from LiDAR scans. The network both estimate the objects confidence as well as encapsulates it by a bounding box. The network is trained on the KITTI data set [14] and manages to achieve promising result for the car class detection task. Additional work has also considered including the tracking task in the same network. In [15] a novel deep learning method that employs a recurrent neural network (RNN) to capture the state and evolution of the environment is developed. The method is inspired by the Bayesian filtering and takes raw LiDAR data as input and outputs an occupancy grid with binary values. One of the key features with this approach is that it is possible to train the model in an entirely unsupervised manner. The idea is to use LiDAR as ground truth when validating, by simulating the input as blocked and compare the output with a true scan the networks learns to track occluded objects. The authors argue that compared to the more traditional model-free multi-object tracking approach the proposed method is more accurate in predicting future states of objects from current input.

1.4 Scope and limitations

In the transport business tractors change trailers frequently making an autonomous system impractical to implement on the physical trailer. Therefore the required equipment for autonomous driving must be fitted on the tractor, meaning field of view is predefined.

An A-double combination was not available during the course of this project, at our disposal was instead a truck without trailers. The simplification that this brings is that the vehicle will be static in the sensor frame, i.e. the field of view will stay constant. Given that a truck with trailer is used the algorithm must be modified to compensate for rotations along each trailer axis such that the trailer is not misinterpreted as a separate object, this has not been taken into consideration in this thesis.

The truck considered in this thesis is equipped with two Velodyne VLP-32C LiDAR sensors with a range of 200 m and accuracy of up to 3 cm, see specification in Table. A.1. The LiDARs are mounted on each side-view mirror providing a visibility field according to Fig. 1.3 and the LiDARs runs at an update frequency of 8.3 Hz.

Each sensor operates in their respective coordinate system, since there are multiple sensors involved a common reference system must be created. We have defined this system with regard to the truck, the origin of this system is placed on the center of the vehicle front at ground plane and follows the right hand rule. Where x heading in the same direction as the truck, y-direction to the left, and z pointing upwards. Illustration of the involved coordinate system can be seen in Fig. 1.4.

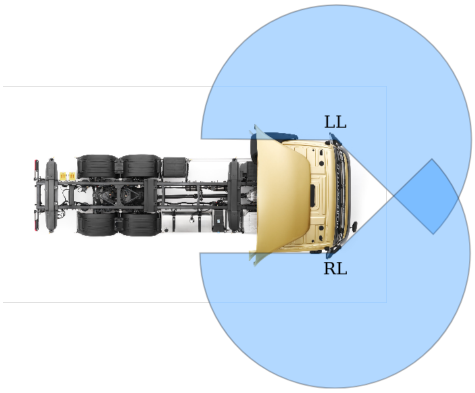


Figure 1.3: Field of view for each LiDAR, note that the range is not made to scale. Figure based on [16].

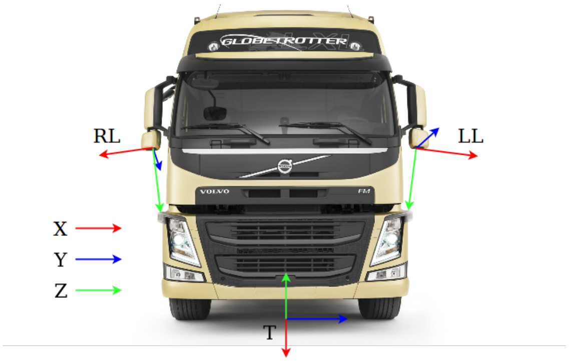


Figure 1.4: Illustration coordinate system for left LiDAR(LL), right LiDAR(RL) and truck(T). Figure based on [16].

Note that each LiDAR x axis points away from the vehicle and that z is aimed towards the ground plane, this is due to simplified wiring and attachment but need to be considered in use.

As only highway scenarios are considered we assume that the road plane is parallel with the trucks in the range of the sensors, illustrated in Fig. 1.5. This simplification is based on [17] where it is stated that vertical curves in Sweden has a minimum concave radii of 2000 m, meaning that changes in slope occur slowly. This simplification allows filtering of point clouds to be performed solely on their height information.

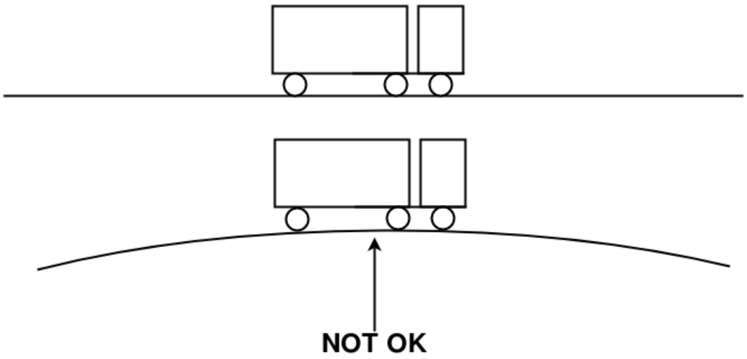


Figure 1.5: Road plane assumed to be parallel to the vehicle.

1.5 Structure of the report

The structure of this thesis will be divided into the following chapters. In Chapter 2 the theoretic parts used in this thesis are presented which are followed by the methodology in Chapter 3. The method Chapter describes how we have transformed the 3D data and how it has been used in neural networks to output a prediction of the surroundings. The results are presented in Chapter 4 which are discussed in the next coming chapter, Chapter 5. Finally, the thesis is concluded in Chapter 6.

2

Theory

To support the implementation and framework used in this thesis the underlying theory is presented in this chapter. This comprises the LiDAR sensor, the probabilistic data representations technique, occupancy grid, that is used to represent the environment, pre-processing techniques to enhance and extend the raw data and lastly the elements of Artificial neural networks and the algorithm and tools to construct and train a network.

2.1 LiDAR sensor

A LiDAR sensor measures the relative position of objects by laser signals. The sensor sends laser signals that reflects off an object and traverses back and is then captured by receive optics in the LiDAR. The distance to the object that the signal reflected on is calculated by time of flight, this information is supplemented with angle to determine the exact position of the reflection. There are different LiDARs, 1D LiDAR that only measures range in one dimension, 2D LiDAR measures the range and the azimuth angle and the 3D LiDAR that measures range, azimuth and elevation angle [18]. The raw output from the 2D and 3D LiDARs are sample distributed point clouds where each point cloud is an array of measured points that is defined in the dimension of the LiDAR, see Fig. 2.1 for an illustration of a 3D point cloud.

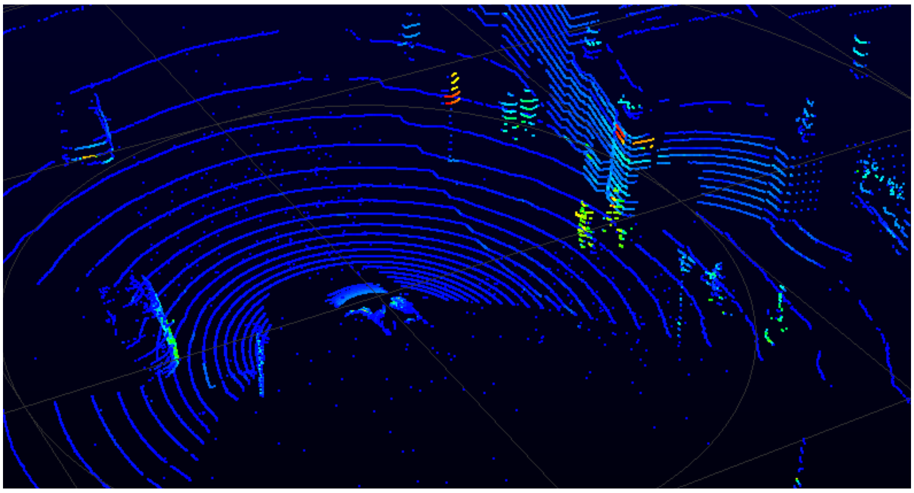


Figure 2.1: Visualization of a 3D point cloud

For a given sample k a 3D LiDAR point cloud is defined in Cartesian coordinates as:

$$P^k = \{p_1, p_2, ..., p_N\}, \ p_i = \{x, y, z\}^T \tag{2.1}$$

where the number of points N are dependent on the capacity of the LiDAR and the number of reflections.

A LiDAR has a range accuracy usually within a few centimeters which therefore can represent the objects position, size and shape in the surrounding precisely [19]. This attribute is popular within the automotive industry, for example in the DARPA Grand Challenge [18] all finishers had at least one LiDAR. However disadvantages for LiDARs are the high price, Velodynes 32 layer LiDARs costs roughly \$30000. Also, they are sensitive to bad weather such as snow, rain and fog [20].

2.2 Occupancy grid

The occupancy grid representation is a cell based representation of a space [21]. Each cell in the grid corresponds to a predefined subset of the space that the grid represent. The grid is thus defined as a multidimensional array, either 2D or 3D. Furthermore each cell stores a probabilistic estimate for the state of the cell, such as the probability of the cell being occupied. A 2D occupancy grid can be introduced having a size of $i \times j$ cells, each cells probability of being occupied can be notated as:

$$P(O_{ij}) = [0, 1], \ \forall \ i, \ j. \tag{2.2}$$

An example of a binary image representation in a 2D occupancy grid can be seen in Fig. 2.2.

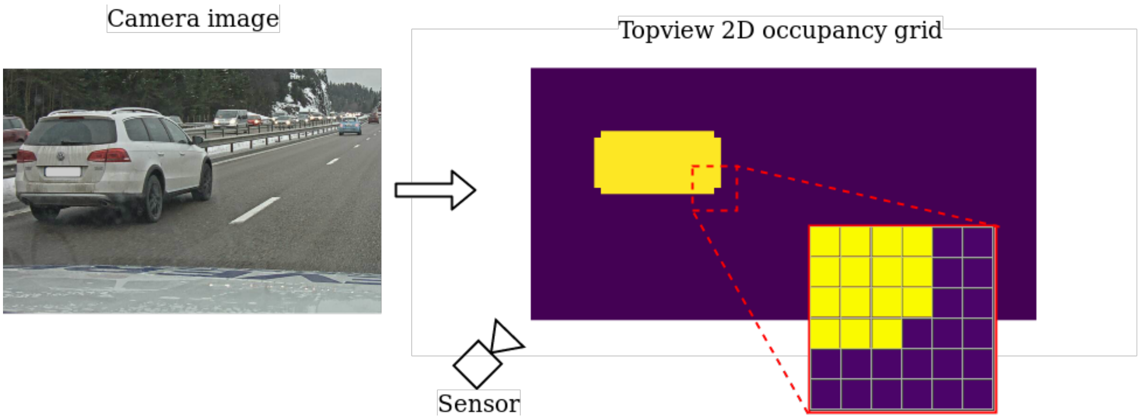


Figure 2.2: Illustration of a 2D occupancy grid. The camera image is used to create a topview occupancy grid for the vehicle in front. Each cell has state occupied, where yellow marks true.

2.3 2D ray tracing

2D ray tracing is a technique that can be used to extract information about which part of the surrounding that is visible or occluded seen from a sensors perspective. This information can be used to create shadows from objects. Ray tracing works in the fashion where a ray is traced from the sensors position through the surrounding until it intersects with an object [22]. In a discrete cell representation, as occupancy grid, the ray is traced by which cells it passes through before intersecting with the object. The cells that the ray passes without intersecting can thus be marked empty.

To determine if a cell in the occupancy grid is visible or not, it is compared against all occupied cells in the grid. If the distance to the centre of the cell of interest is longer than for an occupied cell and the angle to the centre of the cell of interest lies between the edges angles of the occupied cell, it is considered as occluded, otherwise visible. The distances, d, can be calculated with Pythagoras theorem as:

$$d = \sqrt{x^2 + y^2} \tag{2.3}$$

and the angles, α, as:

$$\alpha = \arctan \frac{y}{x} \tag{2.4}$$

where x and y are the distances from origin. Visually, a cell is marked occluded if the ray up to the centre of it has penetrated an occupied cell, see Fig. 2.3. Mathematically, the distance to the centre of the occupied cell(black circle in yellow at (2.5, 3.5)) is 4.3, to green circle at(2.5, 4.5) is 5.1 and to green circle at (5.5, 4.5) is 7.1. The minimum and maximum angle to the occupied cell is 45 respectively 63.4 degrees. The respective angles to the two green circles are 60.9 and 39.3 degrees. Leading to, the cell with green circle at (2.5,4.5) is marked as occluded since the conditions is fulfilled, the distance is larger compared to the occupied (5.1 > 4.3) and the angle lies between (45 < 60.9 < 63.4). The cell with the other green circle is visible since its angle is less than the minimum angle of the occupied cell.

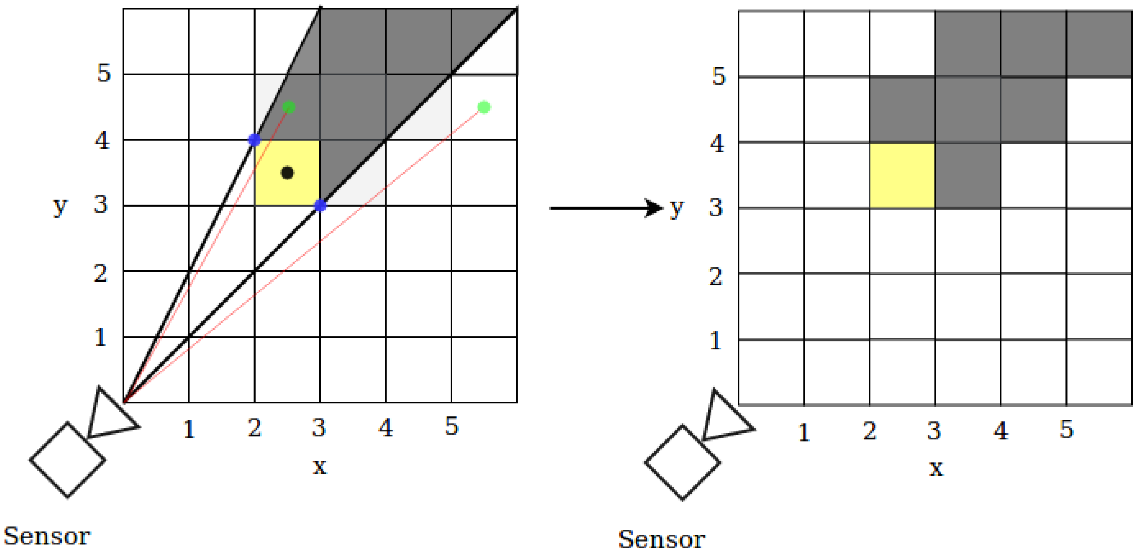


Figure 2.3: Example of determining the visible part and the occluded part in an occupancy grid. Graph to the right illustrating the occluded cells as grey and visible cells as white and yellow.

2.4 Mathematical morphology

Mathematical morphology developed by Matheron and Serra [23] is a technique to analyze spatial structures such as binary images. The technique is based on set theory which in combination can be seen as filters, these filters are commonly used in image processing to eliminate noise but can also enhance images by padding missing pixels.

2.4.1 Morphological operators and filter

Dilation and erosion are morphological operations which expand respectively shrink objects in a binary image. To perform these operations, one binary image and one structuring element, is needed. The binary structuring element B can take any shape and is pre-defined such that relevant structures in the image can be extracted. The mathematical expression for dilation is defined as:

$$A \oplus B = \bigcup_{b \in \check{B}} (A)_b, \tag{2.5}$$

and erosion as:

$$A \ominus B = \bigcap_{b \in \check{B}} (A)_b, \tag{2.6}$$

where A is the binary image and B is the structuring element. The notation $\check{B}$ is the reflection of B, also called transposition, and $(A)_b$ is the translation of A by b. In short, each pixel in the reflected structure element B, translates each pixel in image A by steps equal to the pixels coordinates, i.e. $b_i = (x, y)$. When all translated images are collected, unions and intersection are made with these to obtain the dilated and eroded image. An illustration of dilation and erosion can be seen in Fig. 2.4.

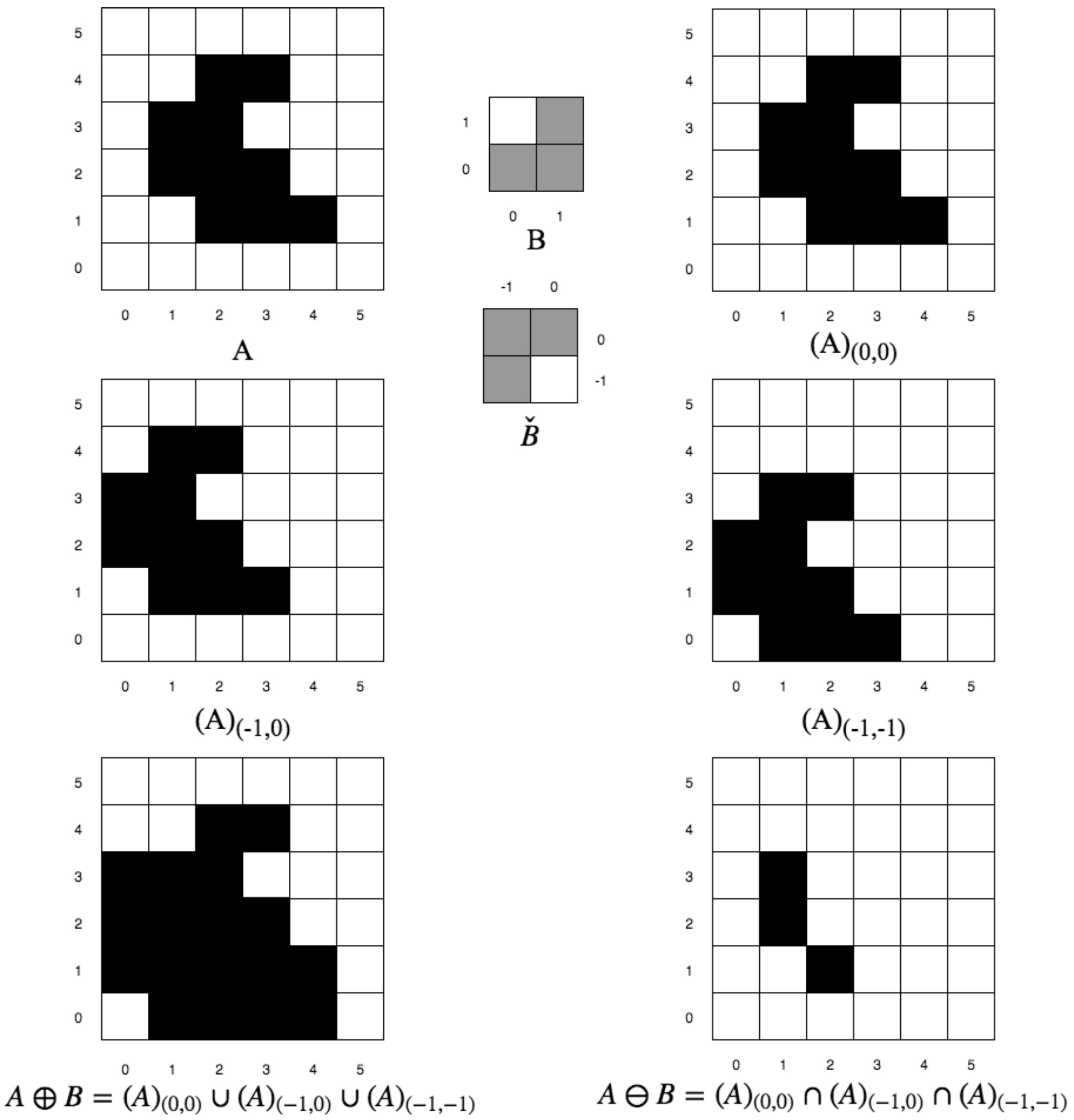

A
B
$\check{B}$
(A)(0,0)
(A)(-1,0)
(A)(-1,-1)
0 1 2 3 4 5
-1 0
$A \oplus B = (A)_{(0,0)} \cup (A)_{(-1,0)} \cup (A)_{(-1,-1)}$
$A \ominus B = (A)_{(0,0)} \cap (A)_{(-1,0)} \cap (A)_{(-1,-1)}$

Figure 2.4: Illustration of the dilation and erosion operations with a binary image A and the corresponding structure element B.

In [23], the definition of a morphological filter is stated as a non-linear transform that locally modify features of image objects. Closing is a morphological filter to smooth altered structures, implemented by one dilation followed by one erosion using the same structuring element. As a mathematical expression closing is defined as:

$$A \bullet B = (A \oplus B) \ominus B \tag{2.7}$$

and one example of closing can be seen in Fig. 2.5.

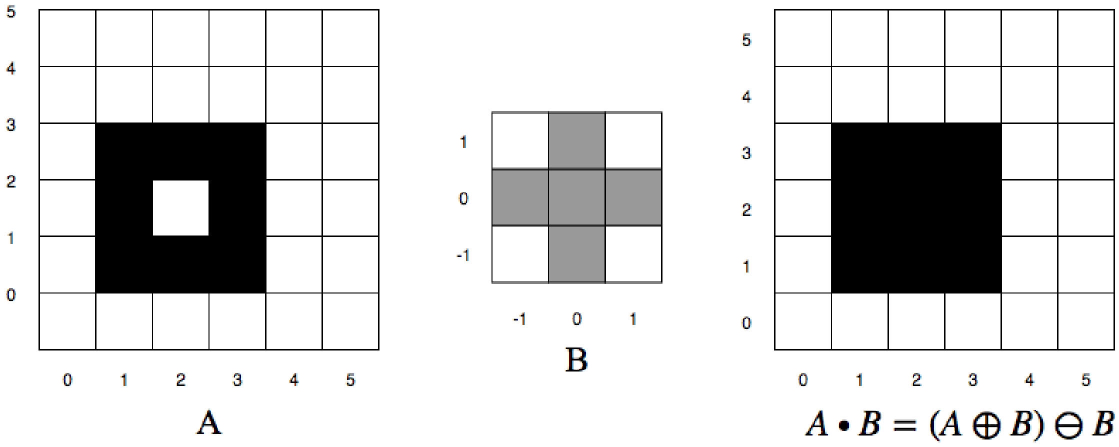


Figure 2.5: Illustration of when the closing filter is applied on a binary image A with structure element B.

2.5 Artificial Neural Networks

Artificial neural networks, also called neural networks (NN), is a commonly used technique in machine learning to solve complex problems in applications such as signal processing and pattern recognition. The technique is inspired by biological structure of the human brain where a task is solved by activation's of neurons, and is learned autonomously. In [24] a classifier for handwritten digits is developed, the input consist of images of 32x32 pixels showing a handwritten number. The network classifies an image by evaluating certain features that correlates to a specific number between 0-9, as the network is trained on several version of the same numbers it has learned to find unique features that is present in all variants, similar to how a human brain works.

The process to solve a specific problem using neural networks are divided into two phases, training- and recall phase. During the training phase the network is trained with examples and learns how to solve the problem. This process is dependent on a training set and a mathematical function, called loss function, to determine the mismatch in the predictions. When the training phase has ended a trained network has been obtained and the network can be used in the recall phase. In the recall phase new data is fed to the trained network and the output is used to solve the task.

2.5.1 Elements of neural network

The neural network computational model consist of two parts, processing elements called neurons and connections between neurons with coefficients called weights [25]. Similar to the brain biology where these parts makes the neural structure. In [25], a neuron is described by the three following parameters:

1. Inputs: x_1, x_2, ..., x_n multiplied with weights w_1, w_2, ..., w_n. A constant bias, b, represented separately from the input is added to each neuron.

2. Input functions f: Calculates the combined net input signal to the neuron $u = f(x, w)$, where x and w are the input and weight vectors. The input function is ordinarily the summation function, $u = b + \sum_i x_i w_i$.
3. An activation function, which calculates the activation level of the neuron, $a = s(u)$, subsequently the output of the neuron.

An illustration of a neuron can be seen in Fig. 2.6.

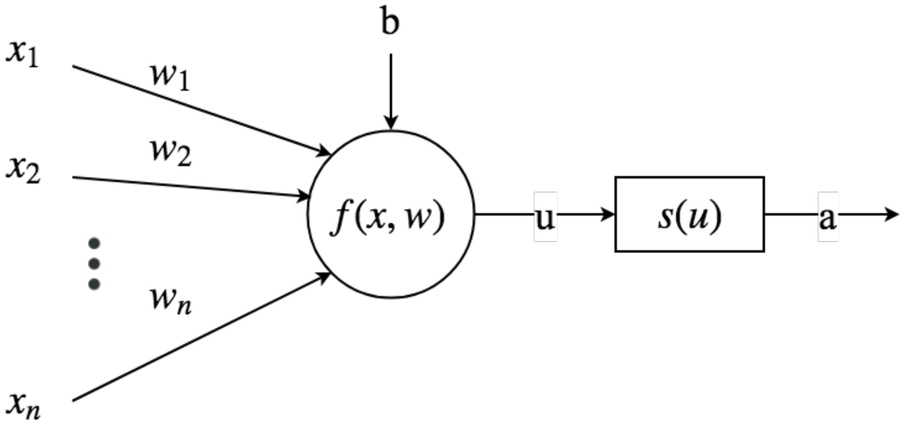


Figure 2.6: Illustration of a neuron model.

Neurons are composed in layers where each layer can have an arbitrary number of neurons. To establish a neural network one must at least have an input layer connected to an output layer such that the information can flow between these layers. Layers between the input and output layers are called hidden layers.

The structure of layers is called connectionist architecture [25]. A network can either be partially connected or fully connected. A partially connected network is a network where an output from a neuron does not connect to all neurons in the next layer, only a part of them. Conversely, a fully connected layer is a network where the output from a neuron connects to every neuron in the next layer. The architecture also includes how many neurons there are in each layer. Furthermore the architecture can be divided into two types as well, it can either be a feedforward or feedback network. A feedforward system do not have any connections from the output layer back to the input layer whilst feedback system have connections from the output layer to the input layer. Considering feedback architectures, the network have memory of the previous state such that the next state depends on both the current input and previous state of the network.

A simple feedforward network with one hidden layer is illustrated in Fig. 2.7. To obtain the output from a network the input is propagated from the input layer through all layers until the output can be retrieved from the output layer. A neural network can consist of more than one hidden layer and such networks are called *Deep neural networks* [24].

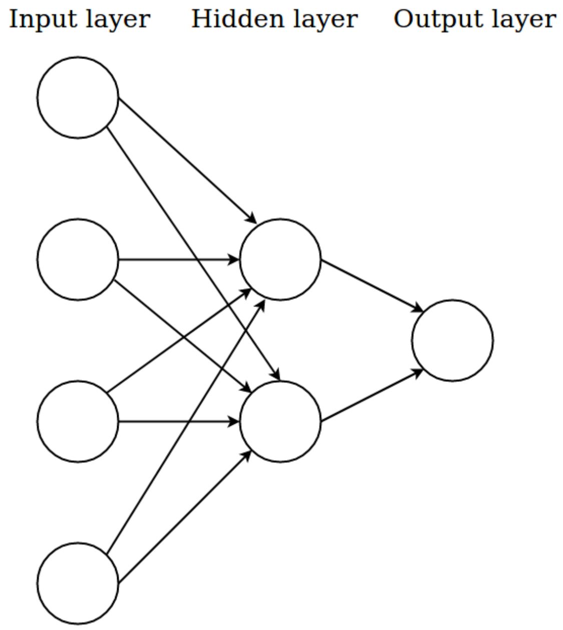


Figure 2.7: Illustration of a simple fully connected feedforward network with 4 input neurons, 2 hidden neurons and one output neuron. The arrows between the neurons represent the connection from the output of one neuron to the input of another neuron.

2.5.1.1 Activation functions

Complex problems often includes some nonlinearity, to learn such behaviours it is necessary to introduce nonlinearity to the network. This is achieved by adding a nonlinear activation function to the calculate the activation level a. Sigmoid and tanh are two commonly used nonlinear activation functions. Tanh is a multiplied and shifted version of sigmoid, see Fig. 2.8.

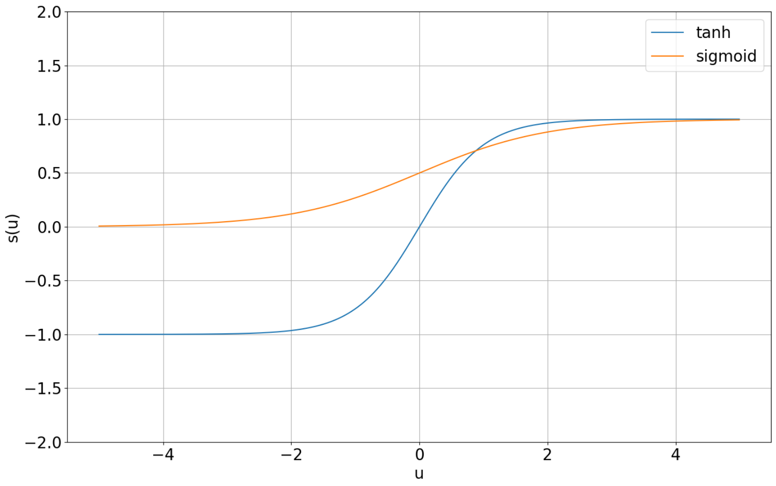


Figure 2.8: Illustration of activation value for each activation function.

In [26] the properties of mentioned activation functions are considered. All of these function introduces nonlinearity to the network. Sigmoid is useful in the final layer if the desired output is a probability, this since the sigmoid range is between 0 and 1. The tanh function is similar to sigmoid but the derivative is stronger which is useful between hidden layers when training. Though, one drawback with these are that they suffer from the vanishing gradient, explained further in section 2.5.2.3.

2.5.2 Training neural network

To train a neural network to deliver the desired behaviour, the weights and biases to all neurons need to be tuned. The training procedure consist of choosing an appropriate data set that will reflect properties desired by the network, next a loss function to calculate the deviations of the predictions must be defined, an optimization algorithm to backpropagate the error and tune parameters of the network must be implemented, finally training parameters such as the number of iteration must be defined. The purpose and theory for each parts will be described further in the following sections. Once an accepted accuracy is achieved the training is halted and the network is set for the recall phase.

2.5.2.1 Training data set

For complex problems extensive amount of data is needed to train a network. In the previously mentioned work [24] where a neural network has been used to classify digits from images, a data set of 70,000 images was used for training the network. As handwritten digits can look very different, the network must be able to generalize, thus needing this large set to find similarities. Further, the data set is often divided into two parts, one training set and one validation set. The training set is used to train the network and learn the task while the validation set is used to provide an unbiased evaluation of the performance.

Depending on available data set a learning algorithm need to be chosen. The most common learning algorithms are *Supervised training*, *Unsupervised training* and *Reinforcement learning*. Where supervised training has labelled data, i.e. the input and correct output vector are supplied to the network such that it can compare the prediction output with the correct output, an example of this is classification of images. In unsupervised training, only input vectors are supplied to the network and the network learns internal features in the data set, it is usually used for clustering of data. Reinforcement learning, also called reward penalty training, the input vector is given and if the output from the network is good, the network is rewarded otherwise penalized. Reinforcement learning uses trial and error to find the actions that get the highest reward given the input. A typical area where reinforcement learning is used are within game theory, the network predicts the best actions to win the game.

2.5.2.2 Loss function

In order to improve the performance of a neural network, one must know how well the network performs. How well the network performs can be calculated with a loss function where the loss should be minimized. The loss is calculated from the deviation in the prediction by comparing the prediction with the target. Several different loss functions are available when training neural networks and one of the most common for simpler networks is the mean squared error. Another common loss function is the Cross entropy function. Cross entropy is a measure of the difference between two probability distributions [27]. The mathematical description of the cross entropy loss function is shown in Eq. 2.8 below:

$$C = -\frac{1}{n}\sum_i [y_i \text{ ln } a_i + (1 - y_i)\text{ln}(1 - a_i)], \tag{2.8}$$

where the sum is over all training inputs n, y_i is the desired output and a_i is the output from the network.

2.5.2.3 Backpropagation

Once the loss is obtained the adjustments of network parameters is performed using backpropagation such that the performance improve. Backpropagation uses the gradient of the loss function to change the weights and biases [28]. To know how much the weights and biases should be changed, the gradients of the loss function is fed into an optimizer. The optimizer ADAGRAD [29], adaptive gradient, is a gradient descent based algorithm that adapts the learning rate to the parameters, where it makes larger updates for less frequent parameters and smaller updates for more frequent parameters. The steps in the backpropagation algorithm are described below:

After the input has been fed to the network and been forward propagated through each layer in the network, the net input signal and the activation level for each layer can be described as two vectors as:

$$\begin{aligned} z^l &= w^l a^{l-1} + b \\ a^l &= s(u^l) \end{aligned} \tag{2.9}$$

where the vectors u^l and a^l for layer l, contain every neurons net input signal and their respective activation level, $s(\cdot)$ is the activation function. Thereafter, the output error vector δ^L with respect to every neuron in the output layer (L) can be calculated as:

$$\delta^L = \nabla_a C \circ s'(u^L) \tag{2.10}$$

where $\nabla_a C$ is the gradient of the loss function with respect to the activation, $\circ$ is the Hadamard product (elementwise multiplication of two vectors) and s' is the derivative of the activation function. When the output error is calculated the error can be backpropagated through each layer up to the input layer, with the following equation:

$$\delta^l = ((w^{l+1})^T \delta l + 1 \circ s'(u^l). \tag{2.11}$$

Where w is the vector corresponding to all weights in a layer. Note that there are not any errors in the input layer when $l = 1$. When all output errors are obtained, the gradient of the lost function with respect to any weight in the network can be calculated as:

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{2.12}$$

where w^l_{jk} is the weight between neuron j to neuron k in layer l. The gradient with respect to each bias is simply:

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j. \tag{2.13}$$

Lastly the gradients are used to update the weights using gradient descent, e.g. the adaptive gradient optimizer ADAGRAD.

There are two common problems connected to deep neural networks, the vanishing and exploding gradient. Both related to the backpropagation. Vanishing gradient means that the gradient vanishes in the earlier hidden layers, especially when sigmoid and tanh are used as activation function. As the gradient is calculated using the derivative of the activation function and sigmoid and tanh have a derivative less then one, except tanh at zero, the output error becomes smaller and smaller for each earlier layer. This in turn leads to that the output error vanishes in the first layers of deep networks which in turn gives a low gradient. A low gradient means that the network will train very slowly. Contrarily if the error signal δ is larger than one, the networks gradients will change in the opposite way, i.e. increase and the weights will find it hard to converge, thus the exploding gradient problem occur.

One benefit of using the cross entropy with the sigmoid activation function is that the local gradient gets cancelled out in backpropagation. This in turn avoids the learning from slowing down [24].

2.5.2.4 Training parameters

Similar to humans, learning is an iterative process, training must be repeated such that network parameters converge and satisfying output is achieved. The number of times the whole data set is iterated is known as *epochs*, the number of epochs required depends on several factors such as desired accuracy, size of training set etc. Further adjustable training parameters are *batch size* and *mini-batch size*, the mini-batch is commonly used in RNN where this value defines the number of samples in each sequence. While the *batch size* defines how large part of the data set that will be considered when changing the parameters in the network, e.g. data set of 100 mini-batches and batch size of 25 will result in four changes of the parameters in one epoch. Batch size of 100 will consider the whole data set resulting in one accurate step in the negative gradient per epoch, lower batch size will result in a less accurate step but as the update will occur more often the weights and biases will converge faster. The size of the step taken in the negative gradient is called *learning rate*, intuitively larger step means faster convergence but also more likely to miss the local minima of the parameters.

Defining an appropriate training setup is important to achieve a good performance, however more training is not necessarily better. Too much training might result in that the network finds co-adaptions in a specific training data, called overfitting, rather then the actual task [30]. Contrarily too little training might result in that the network does not learn, underfitting. To avoid over- and underfitting a number of precautions can be implemented such as dropout, early stopping and input data shuffling.

Dropout is randomly disregarding some neurons in each layer such that the network is forced to rely on more of its neurons. Such that the network learns features that is generally helpful to produce the correct answer, rather than being dependent on several specific features [30].

Early stopping means that the performance of the neural network is constantly evaluated on a validation set that is excluded from the training set, when this accuracy has converged to a user defined level the training is halted [31].

Input data shuffling is shuffling of the batches such that the order in which the batches are evaluated is changing in each epoch.

2.5.3 Neural Network types

There are several different predefined types of neural networks. They differ in the sense of what mathematical operations and what parameters are needed in order to calculate the output. The reason to why they exist is because they all have different properties that are desired dependent on the task at hand. In this chapter a thorough explanation to network types related to the task will be presented.

2.5.3.1 Convolution Neural Network

Structured data such as images comes with the property that its locality is important. When decoding an image it is often not possible to only consider a single pixel by itself since information can be concealed within several neighbouring pixels [32]. In such case a standard feedforward network is substandard and the idea of convolution filters becomes appealing.

A convolutional layer consist of k number of 3-dimensional arrays called filters or kernels of size $m \times n \times c$. Where m and n are smaller than the input shape and c is the same or smaller. The input data is of shape $M \times N \times C$ where M and N is the width and height of the data and C is the number of channels [33], e.g. in an RGB-colored image C will be equal to 3. For a 2D image, $C = 1$, the image can be represented as a two dimensional matrix $x(i, j)$, in which the filters $f(i, j)$ are convolved over to extract features. The output from each filter convolution is a feature map $y(i, j)$, these maps contain information about what each filter extracted from the data. From [34] the discretized mathematical expression for a feature map is derived as Eq. 2.14

$$y(i,j) = f(i,j) * x(i,j) = \sum_k \sum_l f(i-k, j-l)x(k,l) \tag{2.14}$$

In [35] it is explained that stacking convolutional layers have the ability to learn a hierarchy of features such that the first layer learn low-level features such as edges while the final layer learns more high-level features such as vehicle shapes. Leading to convolutional layers being extremely useful in image processing and classification tasks.

As an example a certain filter could be made to extract edges from raw pixels in an image such that the feature map only contains edges from the input data, see Fig. 2.9. If no feature corresponding to the filter is present in the convolution step the output will consist of extremely small values [36].

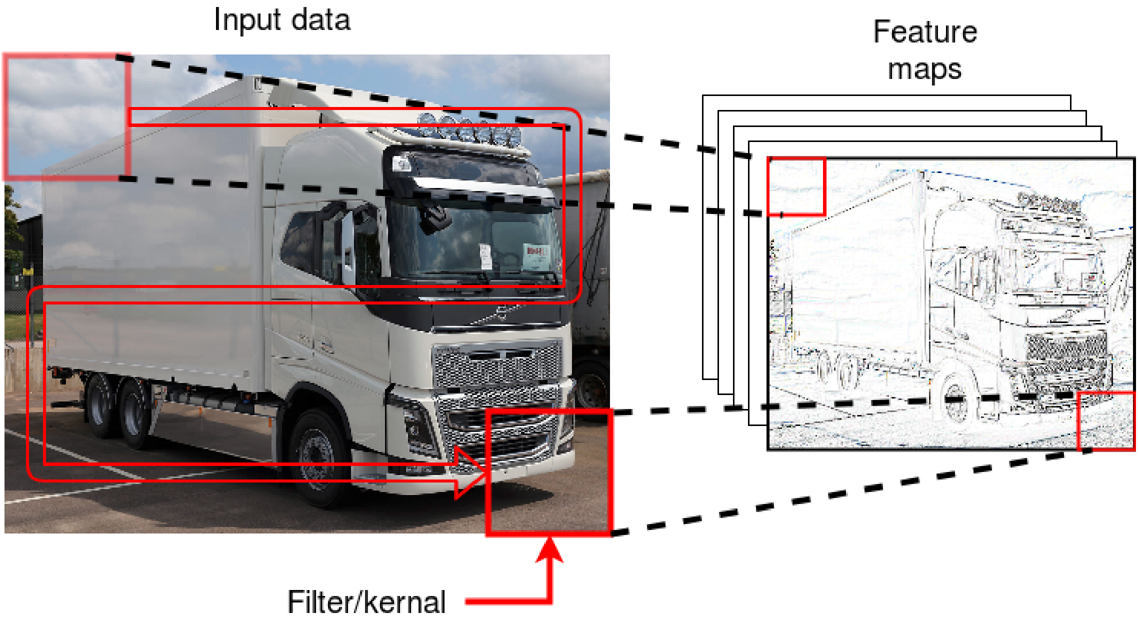


Figure 2.9: Illustration of feature map from edge extraction. The red box illustrates the filter that is convolved with the input image. Figure based on [37].

As mentioned in section 2.5, layers consist of weights and biases. In a convolutional layer these weights correspond to the filters parameters. During training these weights are set to extract the features that improves the network prediction.

2.5.3.1.1 Filter size and dilation As mentioned above the filter size can be chosen arbitrary within the input size. The area of the input data that is evaluated in each convolutional step is called receptive field. For a traditional convolution, the filter size is the same as the receptive field, i.e. the red box in the input data in Fig. 2.9. The receptive field should be chosen with regard to how scattered the information of interest is over the input and also how it might change. Increasing the receptive field could either be done by increasing the filter size or stacking multiple convolutions on top of each other. However for a typical filter, with shape $m \times m$, increasing the filter size results in quadratic increase in variables and a quadratic increase in receptive field, i.e. costly with regard to computational complexity. Stacking convolutions on top of each other result in a quadratic increase of receptive field and linear increase of variables. Another option introduced by [38] is to use stacked dilation. A dilated filter is skipping a number of pixels between each filter component such that the receptive field increases while the filter size remain the same. By combining dilation with stacking it is shown in [38] that the effective receptive field grows exponentially with a linear increase in variables, as illustrated in Fig. 2.10.

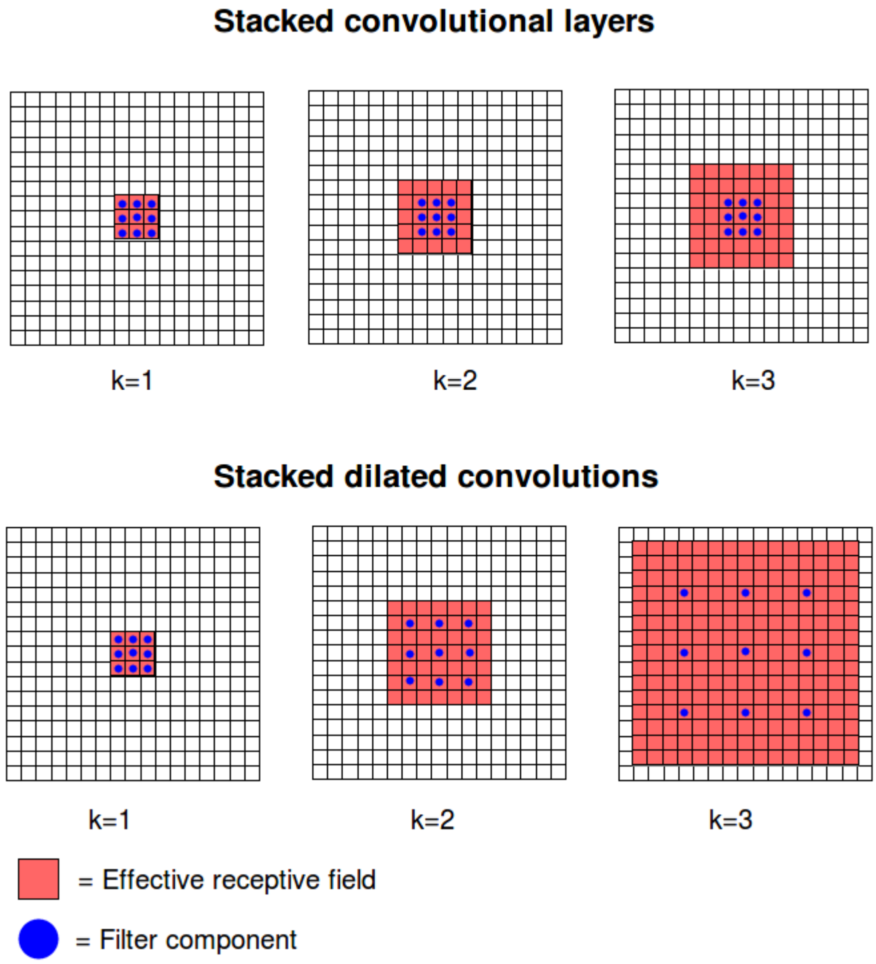

Stacked convolutional layers
k=1
k=2
k=3
Stacked dilated convolutions
k=1
k=2
k=3
= Effective receptive field
= Filter component

Figure 2.10: Effective receptive field of stacked convolution compared to dilated stacked convolution. The receptive field from previous layer is considered at each step. Dilation of $2^{k-1} - 1$ for each layer k in bottom row.

As can be seen in Fig. 2.10 a dilation of one in the second layer corresponds to a effective receptive field of 7×7 compared to 5×5 in conventional stacked convolution. The difference in the third layer truly shows the value of dilation where the receptive field has grown to 15×15 compared to 7×7 for the stacked. Furthermore, in [38] it is shown that the resolution and coverage is preserved.

2.5.3.1.2 Zero padding A result of convolutions is that the filter size will affect the output size. Given a unit step convolution the size of the output will be the number of steps the filter can perform within the data plus 1, such that a 3×3 filter over an $m \times n$ image will have the output size $m - 2 \times n - 2$ [39]. To preserve or control the size it is possible to zero pad the input before convolving, also known as wide convolution. Zero padding the input means adding zeros around the input such that it is possible to e.g. maintain the original size of the data.

2.5.3.2 Recurrent Neural Network

Timestep dependent tasks such as object tracking contain valuable information in its previous states. To make use of such information, RNN can be utilized [40] [41]. In RNN the neural units do not only connects forward to the next layer but also have a feedback connection to the layer itself. This is illustrated in Fig. 2.11, where the RNN considers the input x_t, using the feedback loop and outputs h_t. The loop gives the RNN an internal state or memory that can be useful in sequential tasks.

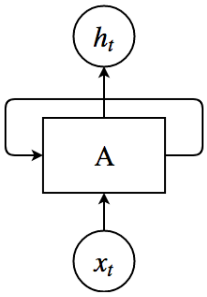


Figure 2.11: A recurrent neural network with input x_t, loop and output h_t

When the input is sequential, the RNN can be more easily visualized by unrolling it, this can be seen in Fig. 2.12 where it has been unrolled t times. Where the input for a specific time is one part of the sequence. RNNs have been used more frequently the last decade and shows satisfying results in areas such as speech recognition, language modeling and image captioning [40][42].

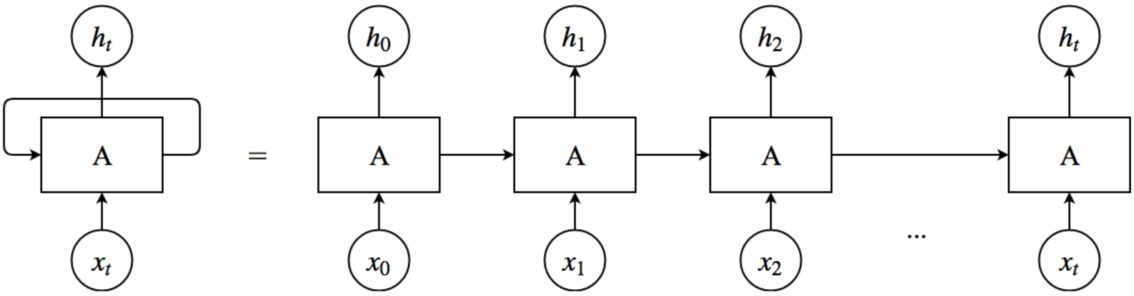


Figure 2.12: An unfolded recurrent neural network

When considering RNN the backpropagation must consider activations at different timesteps. This is handled by the backpropagation through time (BPTT) algorithm

[43]. It works in the fashion that the recurrent part is unfolded, as in Fig. 2.12, to reflect a static version of the network for each time step. This resulting in a deep neural network where one can see that the network have t hidden layers. The standard backprogation algorithm is then applied to each time step and the correction terms are calculated for the weights and biases such that the cost decreases.

In standard RNNs, the repeating module contains a single layer with the hyperbolic tangent function, tanh, as the activation function see Fig. 2.13.

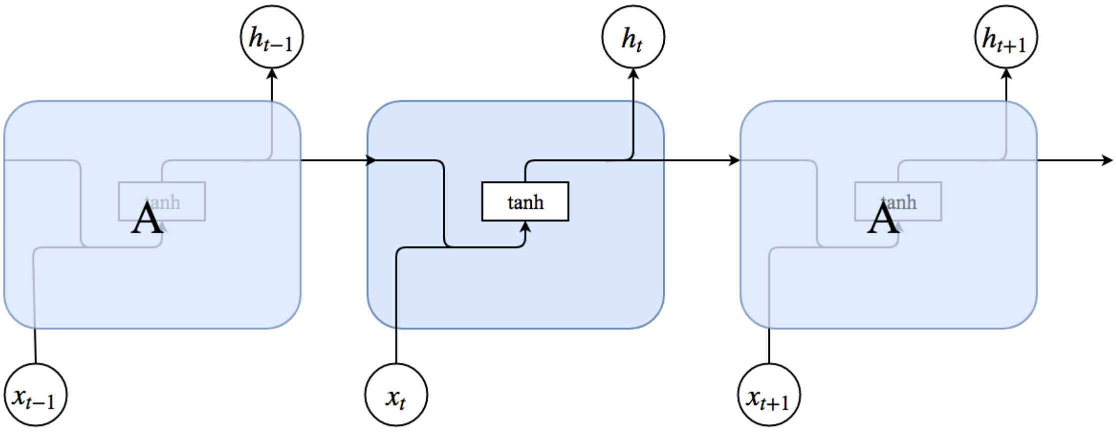


Figure 2.13: Repeating module of a standard RNN

As mentioned earlier, the derivative of the activation function tanh is smaller than 1 for all inputs except when the input is equal to 0. Which means that for longer sequences the error signal decreases during backpropagation, resulting in that weights and biases in early layers is subjected to extremely small changes. The effect of the vanishing gradient is that RNN networks have difficulties learning longer dependencies. According to [44] this behaviour is seen already after 5-10 time steps. To handle the vanishing gradient problem a new version of RNNs called Long Short-Term Memory network (LSTM) was introduced in 1997 [45].

2.5.3.2.1 LSTM A LSTM network is a special type of an RNN which do not suffer from the vanishing gradient and is thus able to learn long-term dependencies better. The difference between a standard recurrent neural network and an LSTM network is the repeating module. The LSTM network saves and updates old information in its cell state which can be seen in Fig. 2.14 as the top horizontal arrow in the repeating module. Since this cell state goes through the repeating module, information from previous time steps are saved. According to several studies [46] [47], it is proven that because of this repeating module the LSTM networks does not suffer from the vanishing gradient in the same extent and are thus able to learn long time dependencies better.

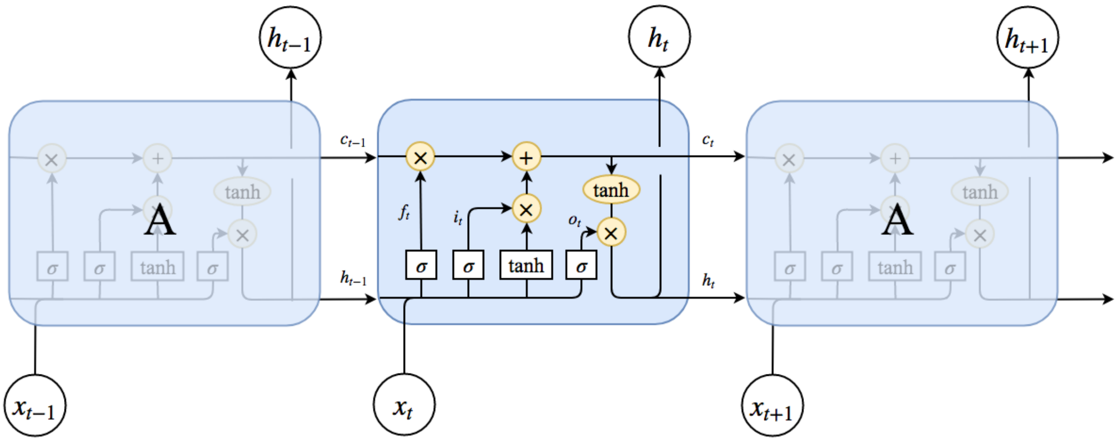


Figure 2.14: Repeating module of an LSTM network

The cell state c_t can be updated in several ways by control gates. How much of the previous cell state c_{t-1} that should be remembered is controlled by the forget gate f_t. Also, the new input will be accumulated to the cell state if the input gate i_t is on. At last, whether the cell state should be propagated to the output depends on whether the output gate o_t is activated or not. By using the cell state and these gates to control the information flow, the gradient will be captured in the cell, which is also known as the constant error carousels [45]. This in turn prevents the gradient from vanishing too fast which is the problem with standard recurrent neural networks. From [47] the equations for the LSTM network are derived and these are shown in Eq. 2.15 below:

$$\begin{aligned}
i_t &= \sigma(w_{\mathrm{xi}}x_t + w_{\mathrm{hi}}h_{t-1} + w_{\mathrm{ci}} \circ c_{t-1} + b_{\mathrm{i}}),\\
f_t &= \sigma(w_{\mathrm{xf}}x_t + w_{\mathrm{hf}}h_{t-1} + w_{\mathrm{cf}} \circ c_{t-1} + b_{\mathrm{f}}),\\
c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(w_{\mathrm{xc}}x_t + w_{\mathrm{hc}}h_{t-1} + b_{\mathrm{c}}),\\
o_t &= \sigma(w_{\mathrm{xo}}x_t + w_{\mathrm{ho}}h_{t-1} + w_{\mathrm{co}} \circ c_{t-1} + b_{\mathrm{o}}),\\
h_t &= o_t \circ \tanh(c_t),
\end{aligned} \tag{2.15}$$

where σ is the logistic sigmoid function, x_t is the input, h_{t-1} is the output from previous time step, $\circ$ denotes the Hadamard product. w is weight matrices, for example w_{hi} corresponds to the hidden-input gate matrix and w_{xo} to the input-output gate matrix and b are biases for the different gates.

2.5.3.3 Convolutional LSTM

Despite the fact that LSTM networks has been proven to handle temporal correlation, it has drawbacks for handling spatiotemporal data. Therefore the Convolutional LSTM (ConvLSTM) layer was introduced in 2015 by Xingjian Shi et. al. [48]. According to the inventors, the ConvLSTM network captures spatiotemporal correlations better and consistently outperforms fully connected LSTM networks. The ConvLSTM network is quite similar to an LSTM network but the input transformation and recurrent transformation are convolutional, leading to all the inputs

$X_1, ..., X_t$, cell outputs $C_1, ..., C_t$, hidden states $H_1, ..., H_t$ and gates i_t, f_t, o_t are 3D tensors where the two last dimensions are rows and columns in a matrix. The equations for the ConvLSTM network are shown in Eq. 2.16 below:

$$\begin{aligned}
i_t &= \sigma(w_{\text{xi}} * X_t + w_{\text{hi}} * H_{t-1} + w_{\text{ci}} \circ C_{t-1} + b_{\text{i}}),\\
f_t &= \sigma(w_{\text{xf}} * X_t + w_{\text{hf}} * H_{t-1} + w_{\text{cf}} \circ C_{t-1} + b_{\text{f}}),\\
c_t &= f_t \circ C_{t-1} + i_t \circ \tanh(w_{\text{xc}} * X_t + w_{\text{hc}} * H_{t-1} + b_{\text{c}}),\\
o_t &= \sigma(w_{\text{xo}} * X_t + w_{\text{ho}} * H_{t-1} + w_{\text{co}} \circ C_{t-1} + b_{\text{o}}),\\
h_t &= o_t \circ \tanh(C_t).
\end{aligned} \tag{2.16}$$

To make sure that the internal states have the same dimensions as the inputs, padding is needed before applying the convolutional operation, see section 2.5.3.1.2 for more information about padding.

3
Method

The task, stated in section 1.2, of rendering an accurate description of the surrounding occluded and unoccluded objects is approached using machine learning. More specifically we considered the Deep learning method proposed in [15] as inspiration to circumvent the issue of annotating data and enable computer visual techniques. The end-to-end approach considers raw LiDAR point clouds as input to compute a posterior occupancy grid using RNN. As the LiDAR is considered ground truth during training, a faulty point-measurement will result in false penalization. To minimize the effect of these a morphological operation to make object shapes more consistent have been implemented. The proposed method is also extended by considering two sensors, that could easily be extended to more.

In this chapter the framework used to implement the proposed solution is presented. Further on the developed system architecture is presented, this work is divided into three main parts where each part is carefully described in the succeeding sections.

3.1 Framework

In the development and implementation of the algorithm a number of different software have been used. The environment have been chosen with regard to simple experimentation rather then the computational efficiency. Below follows a brief presentation to each of them.

3.1.1 Tensorflow

Tensorflow is a Google developed open source library for machine learning. The underlying idea is to think of it as dataflow graphs where the nodes corresponds to numerical operations and the edges, called tensors, corresponds to multidimensional arrays connecting the nodes [49]. The library can be used in the programming languages Python, C and C++.

3.1.2 Keras

Keras is an open source neural network library built for Python development. Keras uses lower level libraries to build the networks. It offers support for Tensorflow, Theano aswell as CNTK (Microsoft Cognitive Toolkit) [50]. Keras is designed to offer fast experimentation by enabling modular, extensible builds that is user-friendly.

The library consists of commonly used building blocks such as layers, activation functions, optimizers etc. Keras is a well known library used for machine learning, according to [50] Keras is one of the most mentioned libraries in scientific papers, see Fig. 3.1. As modularity and fast experimentation is a desired property during development, we chose to work in the Keras environment with Tensorflow as backend.

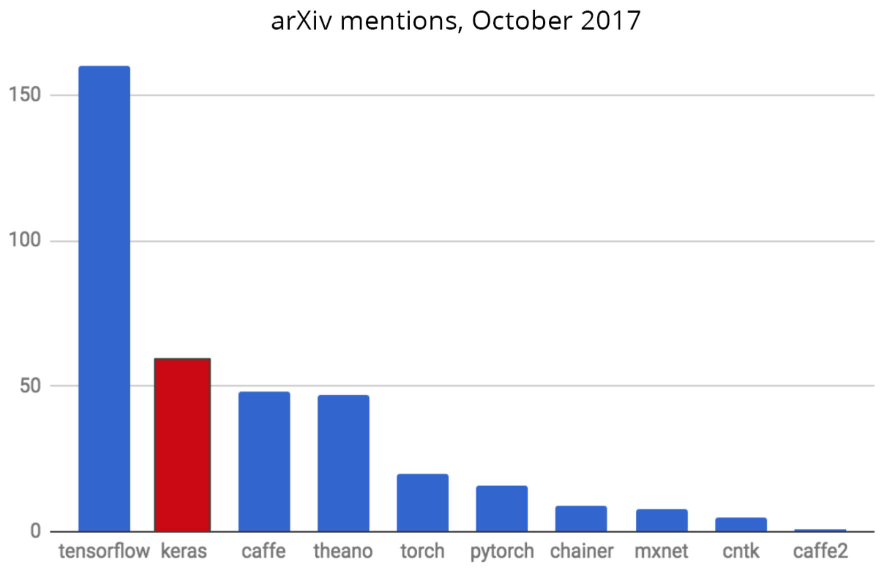


Figure 3.1: Keras mentioning in scientific papers uploaded to arXiv.com

3.2 System architecture

The proposed recall software architecture used in this thesis is presented in Fig. 3.2 where the solution is divided into the following three parts; *Data transformation and filtering*, *Data pre-processing* and *Neural network*.

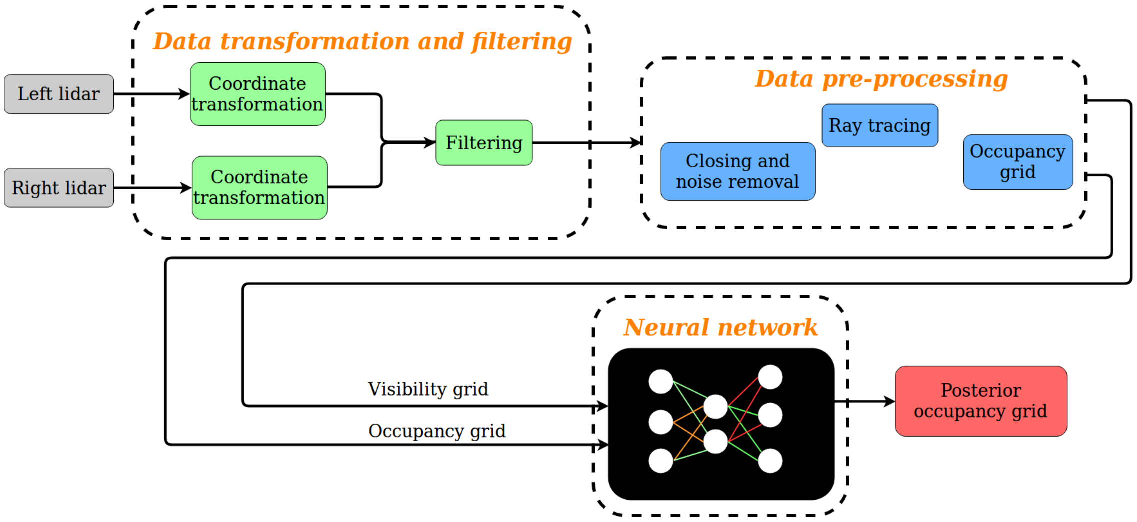


Figure 3.2: Proposed real-time software architecture

The purpose of the data transformation and filtering part is to take raw LiDAR data as input and output relevant measurements defined in a common coordinate system. In the data pre-processing part the purpose is to present the data in the format that the neural network expects. This involves projecting into a occupancy grid, applying a closing filter and calculate the visible areas of the grid seen from the sensor. The last part, the neural network part, is where the sensor data is interpreted such that the output becomes the current unoccluded state of the vehicle surroundings. In the real-time application this loop is iterated at each measurement scan. The implementation of each part is carefully described in the following sections.

3.2.1 Data transformation and filtering

The LiDARs used in this project delivers point cloud data in the format of Cartesian coordinates presented in each LiDARs coordinate system. To instead present the point clouds in the trucks coordinate system, defined in section 1.4, a transformation of the points is needed, this is achieved by a rotation and a translation of each point. The LiDARs orientation and position relative the truck origin is presented in Table 3.1 below:

Table 3.1: Sensor offset relative the truck coordinate system, where roll is the rotation around the x-axis, pitch around the y-axis and yaw around the z-axis.

	Left LiDAR	Right LiDAR
Roll, ϕ [rad]	3.17	3.13
Pitch, θ [rad]	0	0
Yaw, ψ [rad]	1.53	-1.54
x offset [m]	0	0
y offset [m]	1.3	-1.3
z offset [m]	1.95	1.86

To translate each point into the truck coordinate system rotational matrices have been utilized, one matrix for each axis. The rotation matrices are presented in Eq. 3.1-3.3 below:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix}, \tag{3.1}$$

$$R_y = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix}, \tag{3.2}$$

$$R_z = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.3}$$

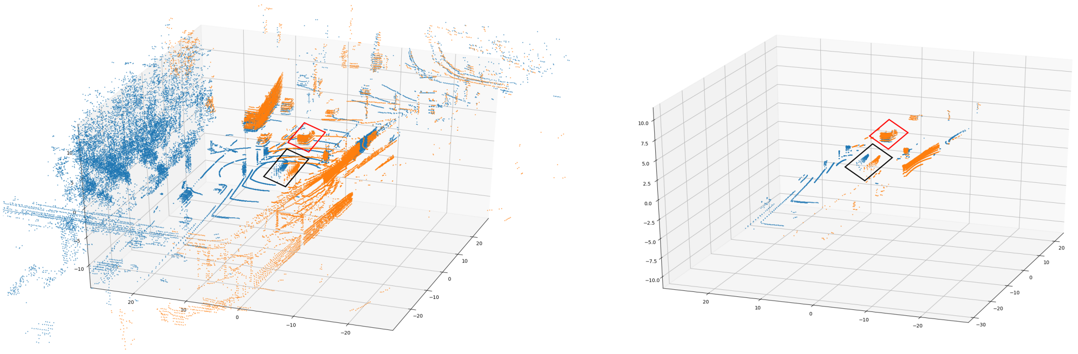
The product of R_x, R_y and R_z is then used to express the total rotational matrix, R, for the LiDARs which is given in Eq. 3.4

$$R = R_x R_y R_z. \tag{3.4}$$

To obtain the resulting transformed point cloud from each LiDAR, every point is multiplied by the corresponding rotational matrix R and translated according to the offsets, see Eq. 3.5 for the mathematical description of a transformation of one point in the point cloud,

$$p' = Rp + p_{\text{offset}} \tag{3.5}$$

where p' is the transformed vector of coordinates, p is the provided coordinates from the sensor and p_{offset} is the offset from the truck origin to the sensor. The translation of points into the trucks coordinate system can be seen in Fig. 3.3a



(a) Unfiltered point cloud

(b) Filtered point cloud

Figure 3.3: Illustration of raw point cloud rotated to truck coordinate system. Orange marks right LiDAR measurements and blue marks left. The black box encircles the ego vehicle and the red box an adjacent vehicle.

When the transformation is completed, a simple filtering is performed of the point p' where it is evaluated if it is inside or outside the region of interest(ROI).

The ROI must be chosen with regard to the application. In this project the route involved in the Autofreight project have been studied. This route consists of primarily two lane highway with some part being single lane and some part being more than two lanes. Furthermore objects positioned more than two lanes away is rarely limiting the driving of the ego vehicle. With regard to this a lateral length corresponding to two lanes in each direction is considered and thus chosen to ±8 m. Longitudinal wise the more is better, however as the LiDAR laser beams are distributed by angle, objects further than ±50 m contain very few reflections. We have therefore limited the longitudinal field of view to ±50 m in each direction. As the task is to find and track objects on road, also the height must be limited. With the assumption that the road plane is parallel to the trucks, stated in section 1.4, we only consider reflections between 0.45 and 1.95 m. The 0.45 limit is chosen to have a margin towards ground reflections and noticed that a upper limit of 1.95 m, is sufficient to get dense reflections of road occupants without the risk of getting road sign reflections. The final ROI seen in the truck coordinate system in meters are thus $-50 \leq x \leq 50,\ -8 \leq y \leq 8,\ 0.45 \leq z \leq 1.95$, see Fig. 3.3b.

3.2.2 Data pre-processing

In this part, the implementation of projecting a point cloud into the expected input for the network, occupancy grid, is explained. In this transformation morphological operations are applied to enhance the raw data where the sensor was unable to reflect the correct state of the surrounding. From the enhanced data, calculation of the visible part of the grid, seen from the sensors is explained.

3.2.2.1 Occupancy grid generation

In our application the idea of projecting 3D point cloud into a 2D occupancy grid is essentially to simplify unsupervised training and enable computer vision techniques. This eases the computational effort as the parameters in the grid will be fewer than with a point cloud, leading to a simpler task to learn for the network.

When projecting points to a grid it is possible to define the resolution by defining the size of each cell. In an automotive application the margins tend to be relatively large, a normal Swedish highway lane is 3.5 m [17] while a standard passenger vehicle stretches 1.9 m. To ease the computational effort we argue that a cell size of $20 \times 20\text{cm}^2$ preserve a sufficient resolution while the computational complexity is significantly lowered compared to using point clouds. Resulting in a resolution of 500×80 cells in the occupancy grid, O_{ij}, i stretches from 0-499 and j from 0-79. The origin of the grid, O_{00}, is placed in the top left corner according to Fig. 3.4. Each point in ROI is translated into a specific cell with equations:

$$\begin{aligned} i &= \lfloor (p'_x + 50)/0.2 \rfloor, \\ j &= -\lfloor (p'_y - 8)/0.2 \rfloor + 1 \end{aligned} \tag{3.6}$$

where $\lfloor \cdot \rfloor$ denotes integer division, p'_x and p'_y are the x and y coordinates for a given point in the point cloud. Equation 3.6 is utilized for all filtered points. A cell is marked as occupied if there are more than two points within the cell. This threshold is introduced to avoid noise from occupying a cell.

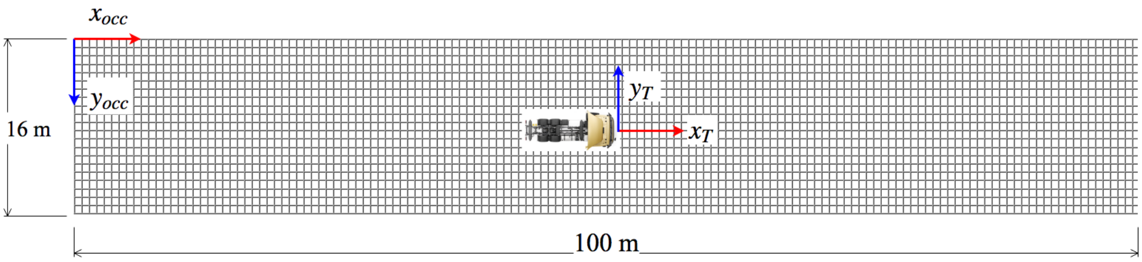


Figure 3.4: Illustration of size and coordinate system for occupancy grid (occ) relative truck (T). The truck origin is placed in the middle of the occupancy grid.

An illustration of a projected occupancy grid of points from ROI is presented in Fig. 3.5.

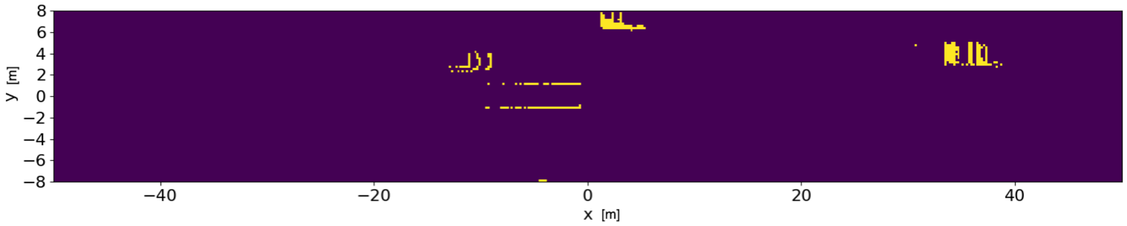


Figure 3.5: Illustration of a projected occupancy grid of points from ROI where yellow are occupied cells. Where the ego-vehicle is located in the origin and reflections from three adjacent vehicle are present.

3.2.2.2 Morphological filtering of occupancy grid

An issue noticed from the grid generation operation is the inconsistency of the objects shape. To compensate for this we introduce morphological filtering. The idea of performing morphological filtering is to fill gaps between cells that correspond to the same object. Practically compensate for sparsely missed reflections on an object occurring when the object surface is close to parallel with the ray, as the first vehicle in Fig. 3.5. By filling these gaps the objects will have a more continuous shape which we hypothesize will ease the object identification, further it will also penalize the network during training more correctly as the LiDAR is used as ground truth. The morphological filtering that are performed consists of two closing filters, where the first fills horizontal gaps whilst the second fills vertical gaps in the grid. The structuring elements used for these two filters are $B_1 = [1, 1, 1, 1, 1]$ and $B_2 = [1, 1, 1, 1, 1]^T$, where T is the transpose and the origin for these structuring elements are placed in the middle. A result when these two filters have been applied can be seen in Fig. 3.6 where adjacent objects have been filled.

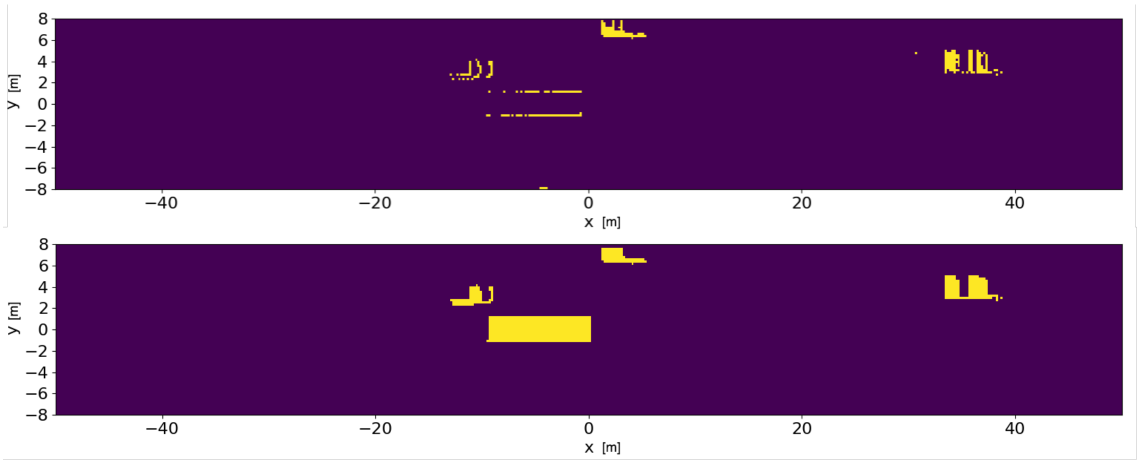


Figure 3.6: Top picture illustrating an occupancy grid before closing, bottom picture after applying closing. Where yellow cells are correspond to a cell being occupied. Note that the ego vehicle is manually marked as occupied, i.e. not a part of the closing algorithm.

As can be seen the objects after filtering has a more continuous "L-shape" that more accurately reflect the actual objects.

3.2.2.3 Visibility grid generation

To be able to keep track on which part of the occupancy grid that are visible or occluded seen from the sensors perspective, ray tracing is performed on the occupancy grid as described in section 2.3. This information is later used as an input to the neural network. To reduce the number of computations, the occupancy grid is divided into three regions, see dashed lines in Fig. 3.7, this since the right LiDARs field of view do not cover the left side of the truck and vice versa, though the region in front of the truck is covered by both LiDARs. A cell in the area where the LiDARs field of view overlaps is only marked as occluded if it is not seen by both LiDARs, i.e. a cell is marked visible if one of the two LiDARs have the cell in line of sight.

The ray tracing is performed only in 2 dimensions, no correction of the height which leads to that if one cell is occupied, all other cells behind it are marked as occluded. This may not always be the case since the LiDARs are mounted approximately 1.9 m from the ground and are multi-layered which in turn may yield reflections on a higher object behind a lower one. Therefore, all cells that are marked as occupied before the ray tracing action are also marked as visible. An illustration of the result after performing ray tracing can be seen in Fig. 3.7.

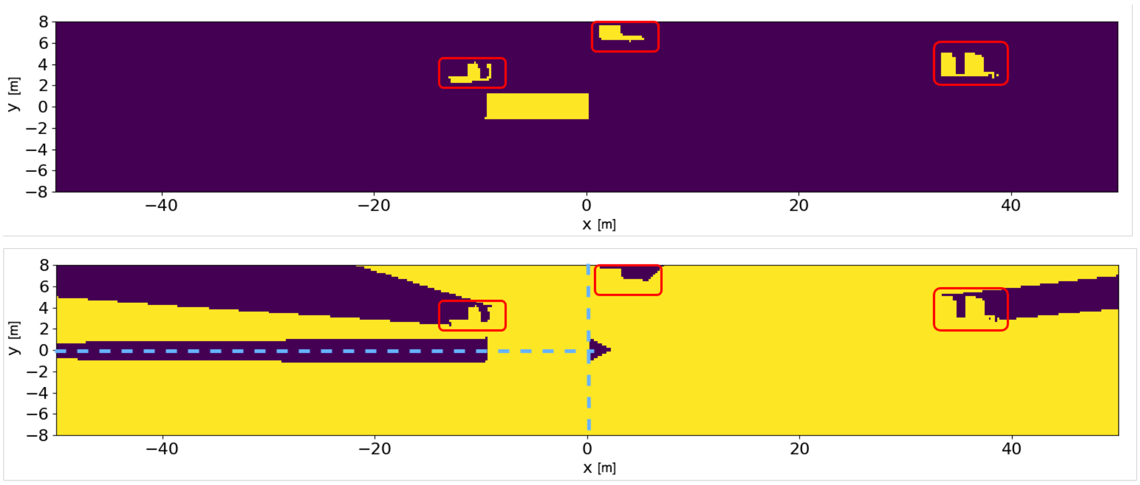


Figure 3.7: Top picture illustrating the occupancy grid before ray tracing is performed, bottom picture illustrates the visibility grid after ray tracing is applied. The blue dashed lines divides the ROI into the three regions. Note the three objects that are marked in red, these are an example of when reflections behind another reflected cell is marked as visible. Yellow cells are visible cells and purple are occluded. The triangle in front of the ego vehicle is an area where the LiDARs are occluded by the ego vehicle itself.

3.2.3 Neural network

At this point a suitable method to pre-process raw data has been derived. In this section the development of a neural network to handle the perception task is proposed. This involves defining network architecture, deriving a training procedure and choosing suitable training parameters.

3.2.3.1 Network architecture

To realize an accurate description of the surrounding, objects must first be detected in the data and thereafter their movement must be followed over time such that they can be tracked during occlusion. It has been shown that convolutional layers handles spatiotemporal data well and that LSTM layers can keep memory and learn long time dependencies. We therefore choose to proceed with the combination of those using the convLSTM layer described in 2.5.3.3.

Memory wise we hypothesize that the time needed to capture the dynamics of the surrounding and predict occluded objects need to be 2.5 s. This length is adequate to cover typical occlusions scenarios observed in the data set. Resulting in mini-batches of 20 samples used as input to the network. The length could be altered to match other scenarios.

Consequently, the input layer is defined to have the shape (20, 80, 500, 2) where it takes 20 consecutive frames with the size of 80 × 500 pixels with two channels. One channel for the occupancy grid projected sensor measurements and another for the visibility grid from ray tracing.

In highway application other road occupants can have different sizes and shapes, e.g. both motorcycles and heavy trucks are expected. We therefore took advantage of the stacking and dilation property of the convLSTM layer that makes the receptive field grow exponentially while the parameters increase linearly. To prevent that the network becomes overfitted to the training data and be more general, dropout is applied to all convLSTM layers.

Finally we want our network to fully connect extracted features. This was achieved by employing a dense layer as the final layer. The information of interest is only the current prediction of the surroundings, the output shape was therefore defined as (80, 500) which correspond to the occupancy grid for the current time. Further, the dense layer uses the sigmoid activation function to introduce non-linearity and represent the output as a probability p of each cell O_{ij} being occupied, i.e. $p(O_{i,j}) \in [0, 1]$.

3.2.3.1.1 Proposed network architectures As the procedure of defining network architecture is trial and error we defined a number of different networks to investigate the effect of changes in number of layers and filters. To simplify comparison we defined a base network in which changes are made from, this network architecture is shown in Fig. 3.8, from here on referred as *Base* network.

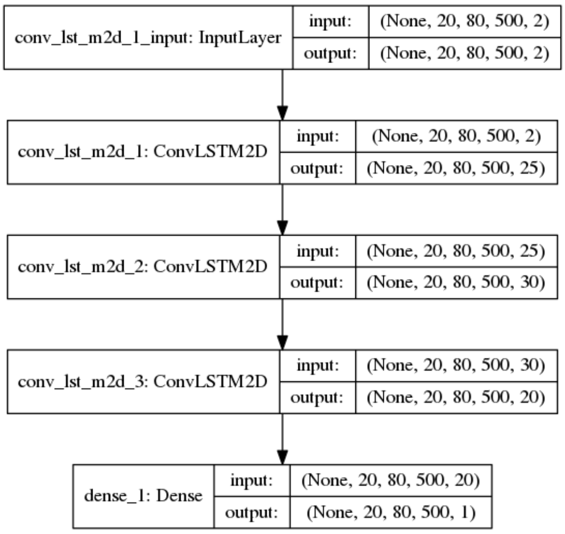
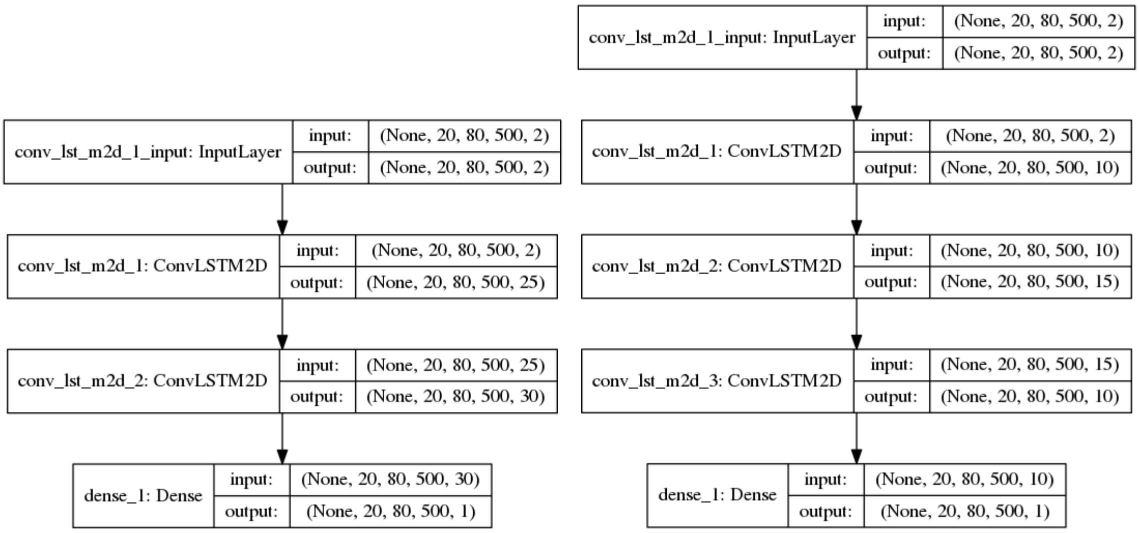

conv_lst_m2d_1_input: InputLayer
input: (None, 20, 80, 500, 2)
output: (None, 20, 80, 500, 2)
conv_lst_m2d_1: ConvLSTM2D
input: (None, 20, 80, 500, 2)
output: (None, 20, 80, 500, 25)
conv_lst_m2d_2: ConvLSTM2D
input: (None, 20, 80, 500, 25)
output: (None, 20, 80, 500, 30)
conv_lst_m2d_3: ConvLSTM2D
input: (None, 20, 80, 500, 30)
output: (None, 20, 80, 500, 20)
dense_1: Dense
input: (None, 20, 80, 500, 20)
output: (None, 20, 80, 500, 1)

Figure 3.8: Architecture of the *Base* network. Three ConvLSTM layers with 25, 30 respective 20 number of filters, followed by a fully connected dense layer to match input shape. Note that "None" is the defined by the batch size during training.

The three hidden layers are of the type convLSTM with 25 filters in the first layer, 30 in the second and 20 in the last. A kernel size of 3 × 3 is used for all layers and the second and third layer has a dilation of 2 respective 4. A dropout value of 0.2 is applied to all hidden layers, resulting in a total of 120.021 trainable parameters.

The three alternative network architectures defined to investigate effects of the number of layers and filters can be studied in Fig. 3.9 and 3.10. Dropout, dilation rate and kernel size remain the same as the *Base* network.


conv_lst_m2d_1_input: InputLayer
input: (None, 20, 80, 500, 2)
output: (None, 20, 80, 500, 2)
conv_lst_m2d_1: ConvLSTM2D
input: (None, 20, 80, 500, 2)
output: (None, 20, 80, 500, 25)
conv_lst_m2d_2: ConvLSTM2D
input: (None, 20, 80, 500, 25)
output: (None, 20, 80, 500, 30)
dense_1: Dense
input: (None, 20, 80, 500, 30)
output: (None, 20, 80, 500, 1)
conv_lst_m2d_1_input: InputLayer
input: (None, 20, 80, 500, 2)
output: (None, 20, 80, 500, 2)
conv_lst_m2d_1: ConvLSTM2D
input: (None, 20, 80, 500, 2)
output: (None, 20, 80, 500, 10)
conv_lst_m2d_2: ConvLSTM2D
input: (None, 20, 80, 500, 10)
output: (None, 20, 80, 500, 15)
conv_lst_m2d_3: ConvLSTM2D
input: (None, 20, 80, 500, 15)
output: (None, 20, 80, 500, 10)
dense_1: Dense
input: (None, 20, 80, 500, 10)
output: (None, 20, 80, 500, 1)

(a) Two layer network architecture.

(b) Less filters network architecture.

Figure 3.9: Model architecture for two of three alternative networks.

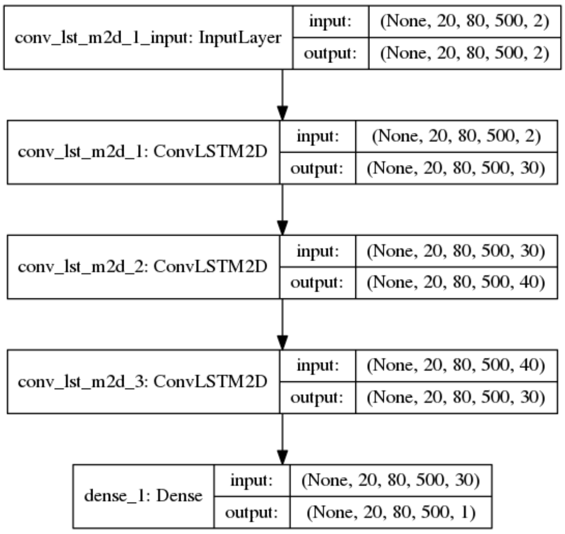


Figure 3.10: More filters network architecture.

The network in Fig. 3.9a, from here on called *Two layer*, only have two hidden ConvLSTM layers with 25 respective 30 number of filter in each layer, resulting in a total of 83.951 trainable parameters. Next network in Fig. 3.9b, from here on called *Less filters*, have 10, 15 respective 10 filters in the respective layers, which is significantly fewer filters than the *Base* network, resulting in a total of 26.971 trainable parameters. The last alternative network architecture, Fig. 3.10, from here on called *More filters*, has 30, 40 respective 30 filters. This is significantly more filters in each layer than for the *Base* network and gives a total of 211.391 trainable parameters.

3.2.3.2 Training

The training procedure used in this thesis is based on the method presented in [15]. As the data collected only contain the visible, seen from the sensor, part of the surroundings, the full ground truth data must be composed. We circumvent the time consuming task of annotating data by instead considering the LiDAR as ground truth for the visible parts. We argue that this is possible due to the exceptional accuracy of the LiDAR measurements, see Table A.1.

As for the occluded parts of the world the ground truth is unknown, we bypass this issue by not penalizing these areas, corresponding to the shadows from adjacent objects in the visibility grid as seen in Fig. 3.7. To train for occlusion we simulate occlusion by hiding inputs in the input. Such that, for a given measurement sequence of $x_t, ..., x_{t+n}$ the network is shown $x_t, ..., x_{t+m}$ where $m \leq n$ and is followed by $n-m$ blank inputs to result in a mini-batch. When comparing the network prediction with the ground truth all frames are considered, which indirectly forces the network to make predictions without sensor input. This can be seen as the sensor being occluded for $n - m$ samples. A graphical implementation of this training architecture can be seen in Fig. 3.11.

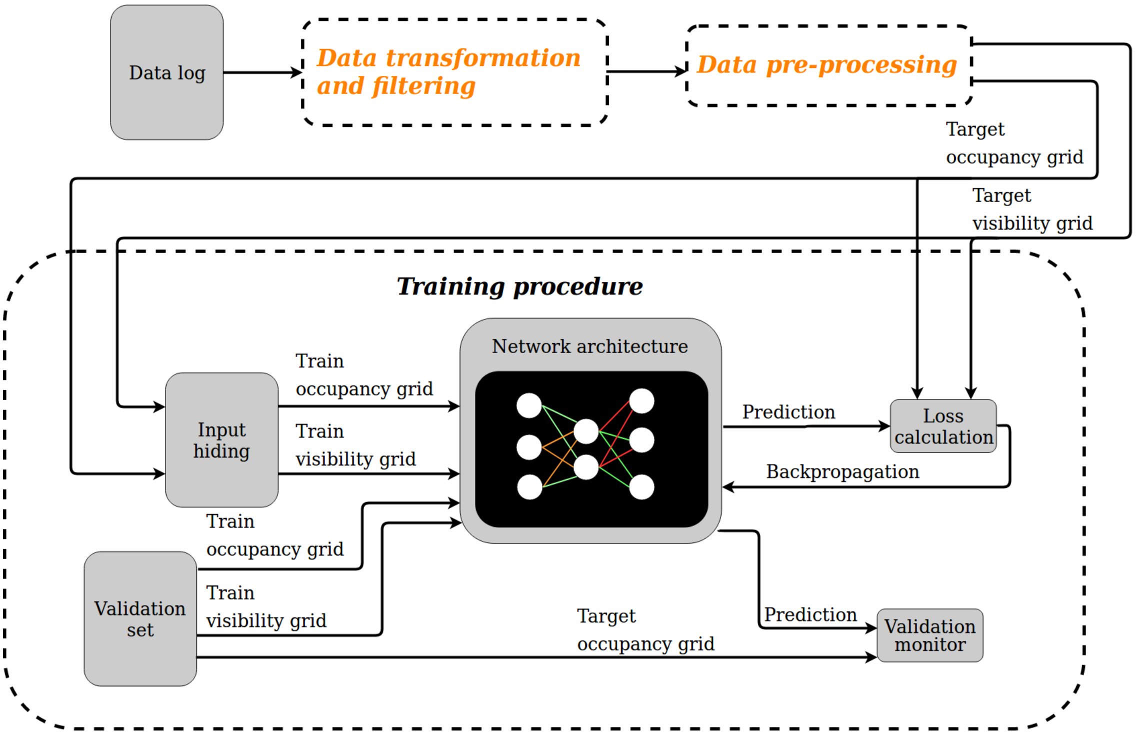


Figure 3.11: Training architecture with the necessary parts to train the network.

As can be seen the logged data is first transformed and pre-processed to output the occupancy grid and the visibility grid for the desired area, these outputs are fed directly to the *Loss calculation* as the ground truth. The target grids are also fed to the *input hiding* block that manipulates the mini-batches to include blank outputs as explained above. In this way the network is tricked to believe measurements are missing while the loss calculation has the ground truth available. The progress is continuously monitored on a separate validation set containing hand-picked scenarios to avoid overfitting.

The hypothesis behind hiding input from the network is to steer it towards establishing and trusting its memory states to accurately predict the movements of occluded objects without sensor input. Which in a conventional model based solution would be the motion model. The ratio between visible and non visible frames must be chosen with regard to sampling rate and the dynamics of the surrounding objects. In this thesis all versions are evaluated with 15 visible followed by 5 blank frames, to investigate the effect of another ratio the *Base* network is also tested using 10 visible followed by 10 blank, from here on called *Frames 10/10*.

3.2.3.2.1 Loss calculation Due to the training method described earlier it is of importance that the calculation of loss is computed such that non-visible areas are not falsely penalized. When calculating loss we only consider parts of the grid that are visible to avoid false penalization. This is achieved by element wise multiplication of the prediction and the visibility grid such that prediction in the non-visible areas are not a part of the loss.

Further, if new objects appear in the region of interest during the blank samples in the mini-batch, the network should not be penalized. Therefore the first and last ten columns in the grid are not considered in the calculation, see Fig. 3.12 for illustration of these part of the grid. The size of ten columns covers most of the typical scenarios when new objects appear in the region of interest as objects rarely travels more than 10 cells during the five samples simulating occlusion.

Furthermore, as trucks typically are overtaken in the lane to the left of the ego vehicle it is important that the algorithm handles those scenarios well. Therefore the loss function is implemented such that it penalizes ten times harder in the region next to the truck, i.e. the lane to the left, see Fig. 3.12.

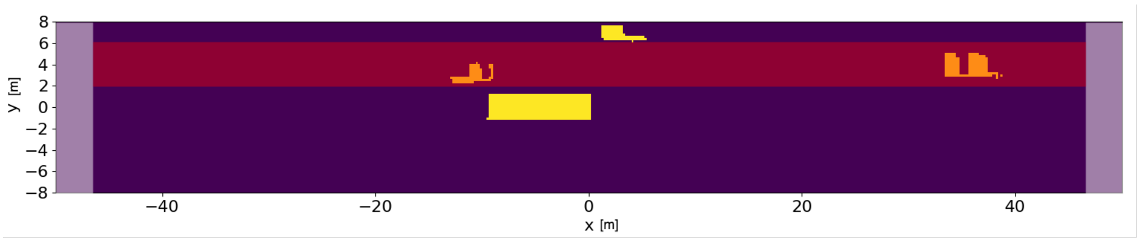


Figure 3.12: Figure showing the considered surrounding where the ego vehicle is placed at origin and three adjacent vehicles present. The edge boxes illustrating the first and last ten columns that are not considered in the loss calculation. The red faded box illustrates the area that are considered more important.

Moreover, the loss function consider whole sequences when calculating the loss, i.e. the neural network does not only compare the last current frame to the ground truth. Thus the network can learn how faulty it has been during a sequence and change the parameters such that the overall tracking performs better. Though, when a mini-batch is fed to the network, the memory states of the network may not be initialized correctly. Therefore, we initialize the network by not penalizing the first five frames.

Finally the prediction is compared with the target using the cross entropy function. The cross entropy function is preferred when comparing probabilities when the output only can take two label, 0 or 1 (occupied or not), of the cell to decide.

3.2.3.2.2 Training parameter setup A representative data set to train and validate the networks was collected along the highway route illustrated in Fig. 3.13. As can be seen the route consist of several different conditions, from single lane to multiple lanes and also tunnel driving, which are representative for the different road environments expected from the route concerned in the AutoFreight project. The training set consist of 1200 frames and the validation set consist of 100 frames. Both sets contains several different overtaking scenarios. Each frame except the first 19 will be predicted in the data set due to the size of the mini-batch, i.e. result in 1181 mini-batches for the training set and 81 mini-batches for the validation set. The mini-batches are shuffled before it is fed to the neural network for each epoch.

During training the networks weights are randomly initialized and biases are initialized to zero. The batch size and number of epochs are chosen to two respectively five. The time spent for one forward and backward pass of a mini-batch was approximately 1 s, i.e. it took approximately 2 hours to train each network. The ADAGRAD optimizer is chosen to utilize the adaptive gradient for fast convergence, the initial learning rate is set to 0.01 and the networks are continuously validated on the validation set. The validation loss is monitored after each epoch and the network parameters is saved if the validation loss has improved.

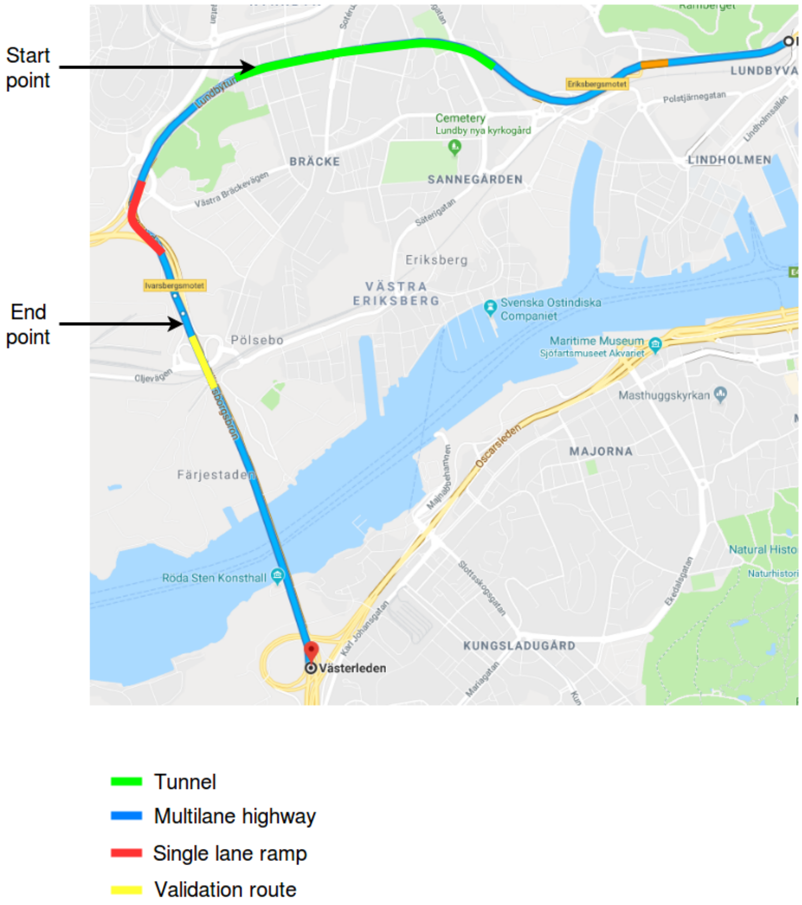


Figure 3.13: Illustration of the route for the training and validation set. The arrows indicates where the training set starts and ends and the validation set is illustrated as yellow.

The networks are trained using a cloud computer from Google equipped with a NVIDIA Tesla K80 GPU with 12 GB ram, the CUDA framework is installed to enable GPU accelerated computations.

4

Results

This chapter will present result from the five proposed network architectures as well as render visual and numerical performance of the networks trained using the method described. Scenarios evaluated in this chapter is manually chosen to be representative for highway driving, such that this thesis objective can be evaluated. As the implementation is specific to the problem, we were unable to find a public data set to benchmark the solution against others approaches. Instead this chapter focuses on establishing result from actual input data to evaluate the proposed method and further compare different model architectures against each other.

4.1 Training results

The validation loss after every epoch for each network can be studied in Fig. 4.1. The relatively high loss obtained is due to amplification of cost in adjacent lane, see the definition of loss function in section. 3.2.3.2.1.

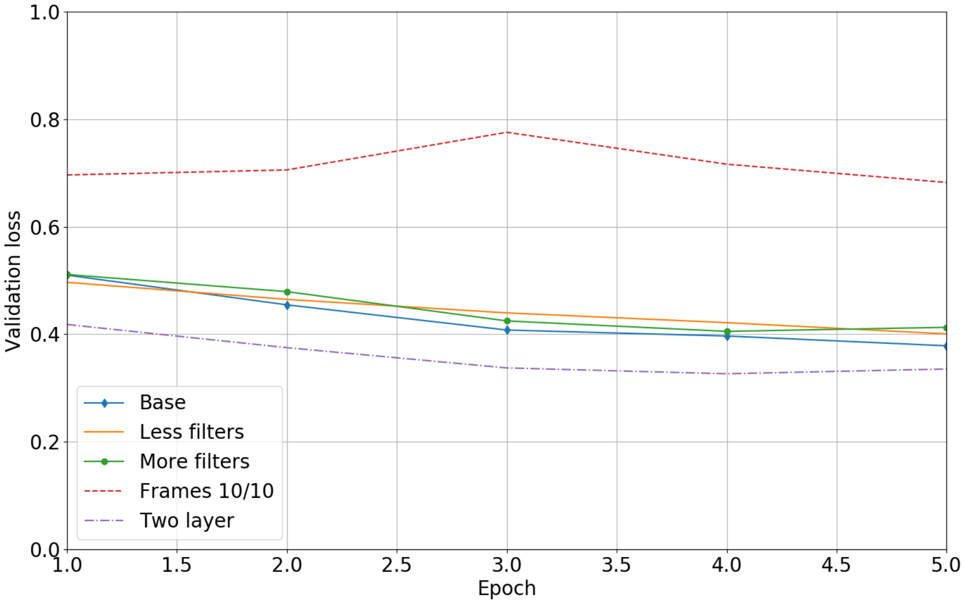


Figure 4.1: Validation loss for each network during training.

During training the loss was evaluated and the weights and biases related to the best score was saved and used in further evaluation.

4.1.1 Filter similarity

As explained in section 2.5.3.1 the fundamental idea with convolutional layer is that each filter is meant to extract a certain feature. To predetermine the number of filters needed in each layer is impossible, it is however possible to evaluate the uniqueness of each filter within the layer after training. Duplicates or extremely similar filters indicates that each filter does not contribute uniquely, which might be due to the number of filters being to high with regard to the complexity of the data.

The result of filter similarity in each networks layer can be studied in Fig. 4.2. The comparison is computed element wise such that all elements in each respective filter is compared, in order for two filters to be considered similar, the largest difference between two elements must be below the tolerance value.

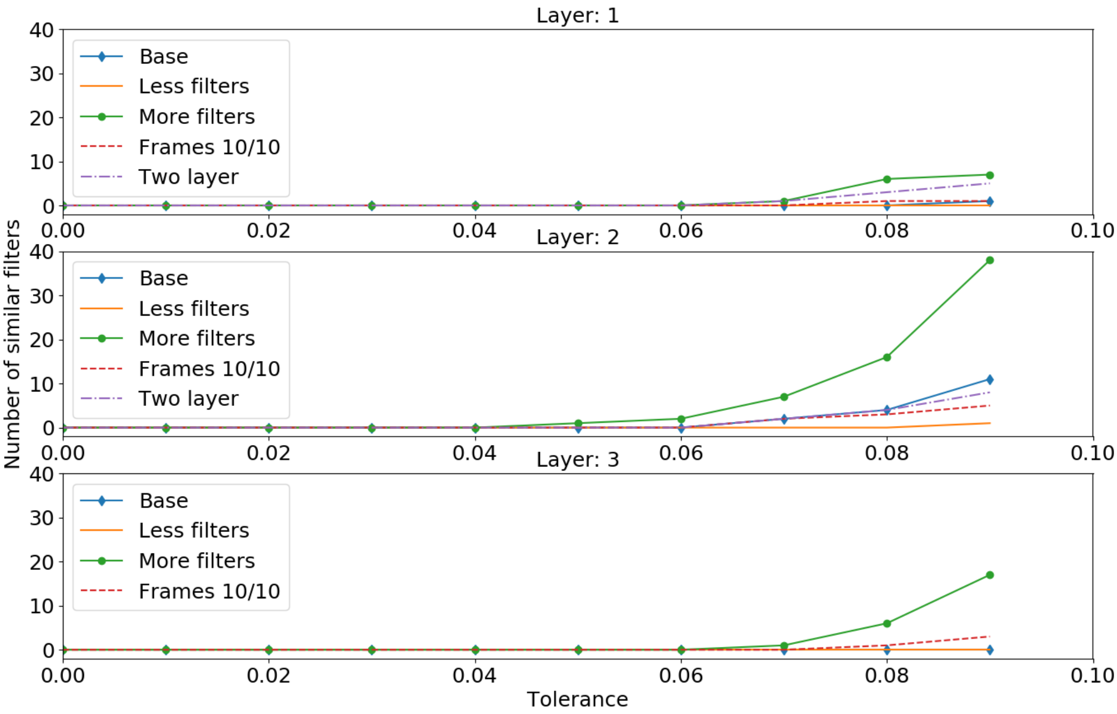
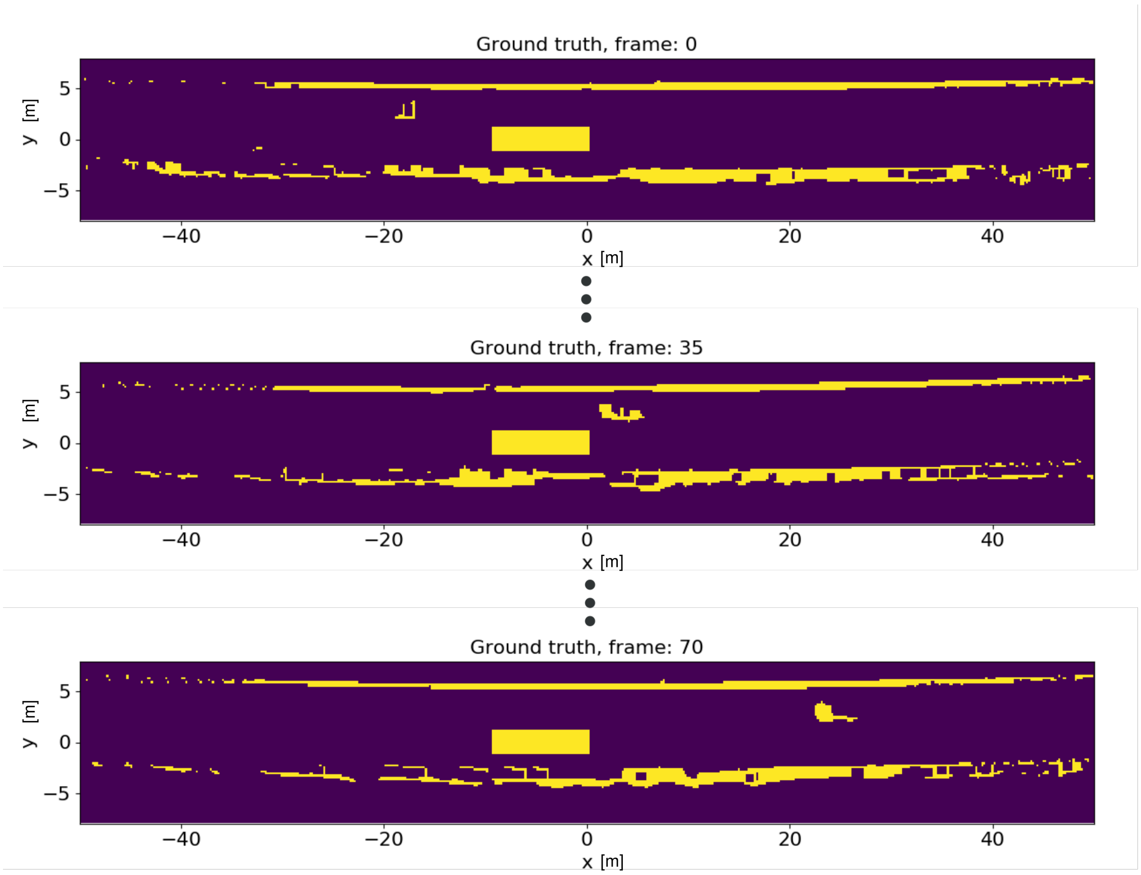


Figure 4.2: Evaluation of filter similarity in each networks layers. The plot shows number of equal filters given tolerance level.

As can be seen there is a relation between the network size and the number of similar filters, i.e. more filters correspond to more filters being similar.

4.2 Simple prediction scenario

A common scenario during highway driving for a heavy duty vehicle is when the ego vehicle is overtaken by another road occupant, one such scenario that was excluded from the training set can be seen in Fig. 4.3. Testing on this scenario will be used to evaluate the performance of the different networks mentioned earlier. The scenario is 70 time frames long (8.5 s), for each frame to predict, the network was shown the prior 19 frames such that the internal memory is initialized. This leads to that a sequence of 89 frames are divided into 70 mini-batches which are fed to the networks.


Ground truth, frame: 0

Ground truth, frame: 35

Ground truth, frame: 70

Figure 4.3: Illustrating an overtake action where it spans over 70 time frames, i.e. a sequence of ca 8.5 s. The overtaking car is behind ego vehicle at the beginning, halfway through at sample 35 it has just passed ego vehicle and in frame 70 it is around 20 m ahead of ego vehicle.

4.2.1 Performance

The performance of each network for the overtaking scenario mentioned above are presented in Fig. 4.4, where the score represents the average accuracy, $1 - \text{loss}$, calculated for all mini-batches fed to the network. This loss is calculated with the same loss function as mentioned earlier except that the adjacent lane is not weighted higher. This since we want a measurement of how many of the visible cells that are correct. The score is calculated for ten different ratios of visible and occluded frames as input to each network, ranging from 0 occluded up to 9 occluded which corresponds to a prediction time from 0 to 1.125 s.

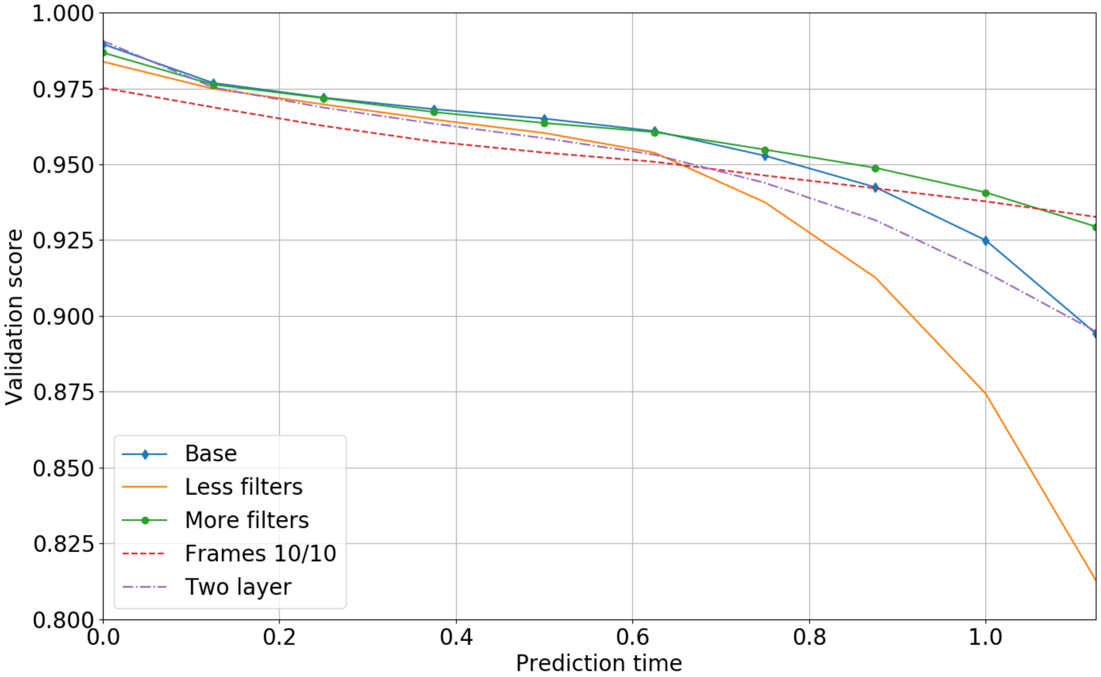


Figure 4.4: Evaluation of the five different convLSTM networks. The prediction time corresponds to how many of the last samples in a mini-batch that are blank where 0 corresponds to no blank samples and 1.125 s corresponds to 9 samples being blank.

The tendency that can be observed is that all networks with architectural differences perform similar up to a prediction of 0.5 s. After that point the performance of the networks differ more. The *Frames 10/10* network performs poorer in the beginning but catches up as the prediction horizon increases.

4.2.2 Visual representation of result

From the overtaking scenario mentioned above, a visual prediction result of 5 blank samples for each of the trained networks are presented in Fig. 4.6. The upper illustration in each subfigure present the raw output from the network. The bottom figure is a comparison against ground truth, *true/pos* means both the network and the ground truth marks the cell as occupied, *false/neg* when ground truth marks the cell as occupied while network predicts it to be empty. *false/pos* is when network predicts the cell to be occupied but the ground truth marked it as empty and true/neg means that both the network and the ground truth marks the cell as empty. In the network prediction, a cell is considered occupied if the probability exceeds 0.5, otherwise empty. The last visible input to the networks and the output ground truth are shown in Fig. 4.5.

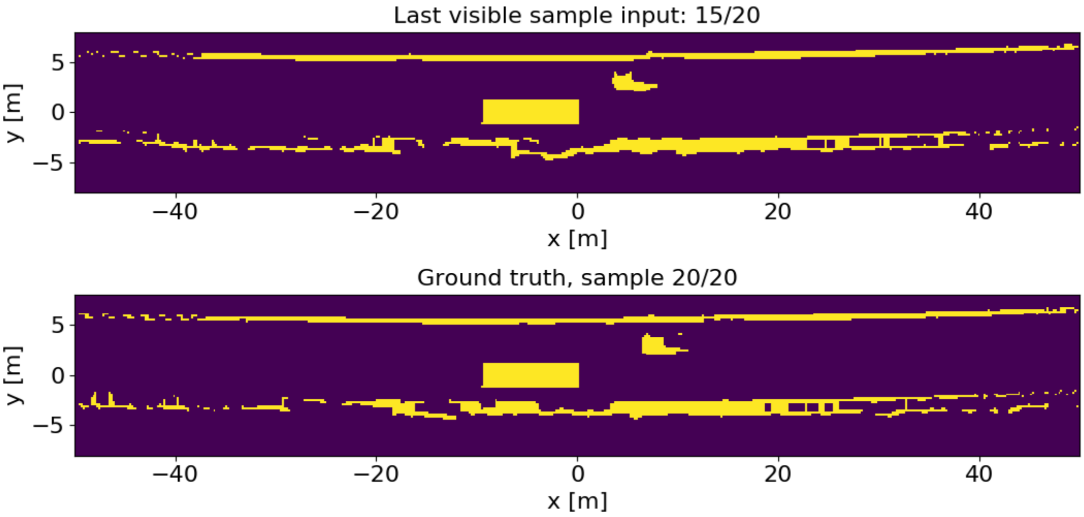
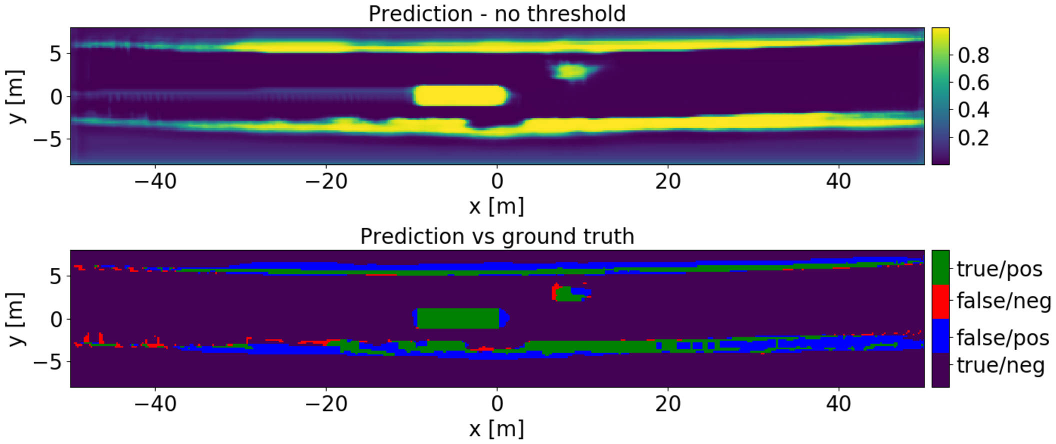
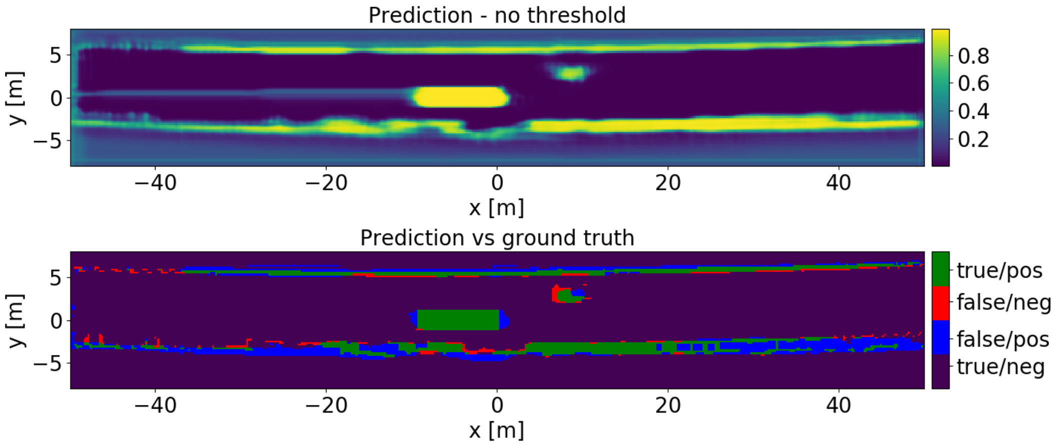


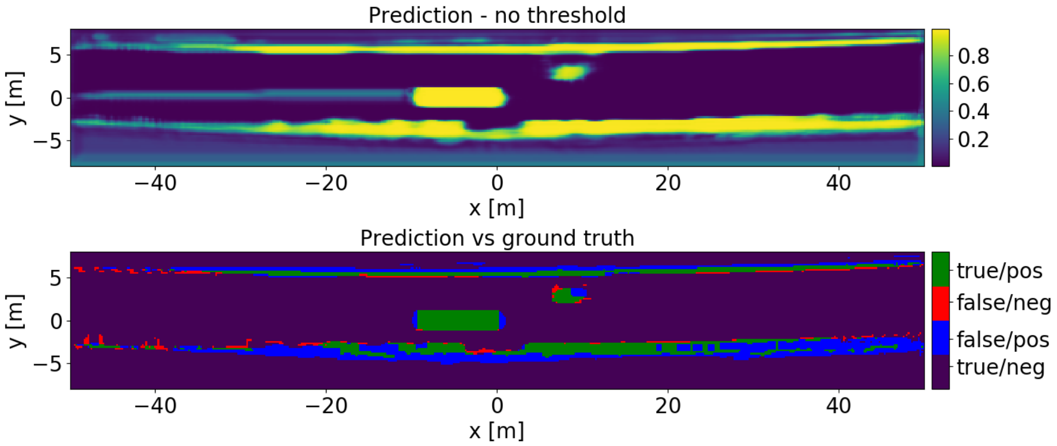
Figure 4.5: Last visible input and ground truth 5 time samples later in the overtaking scenario.



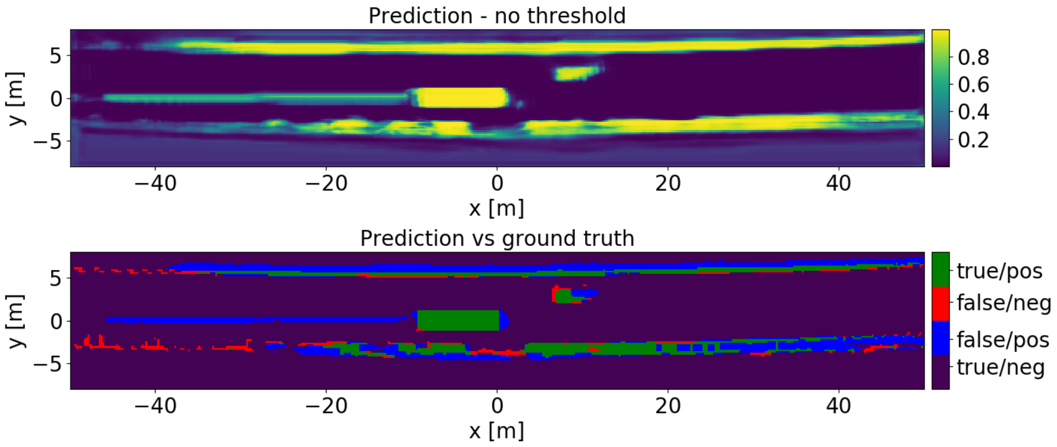
(a) Output from the *Base* network and ground truth comparison.



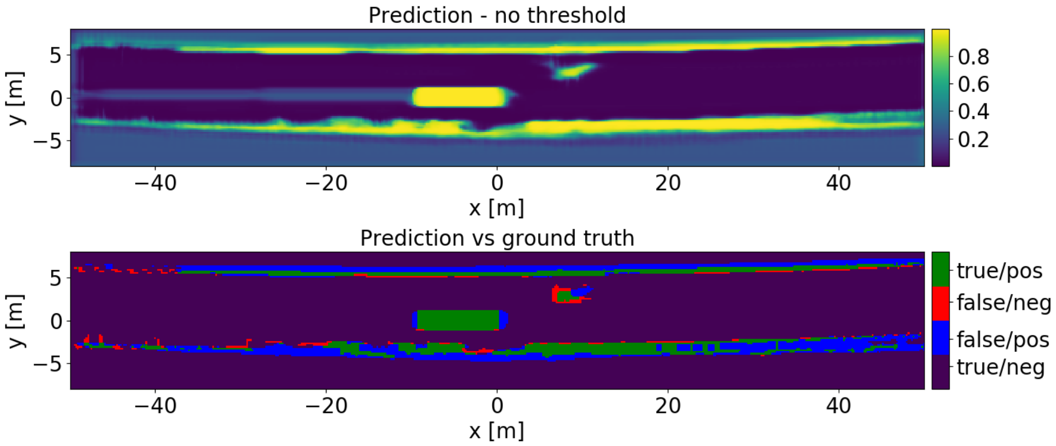
(b) Output from the *Less filters* network and ground truth comparison.



(c) Output from the *More filters* network and ground truth comparison.



(d) Output prediction from the *Frames 10/10* network and ground truth comparison.



(e) Output from the *Two layer* network and ground truth comparison.

Figure 4.6: Output grid for the five trained networks for a 5 sample prediction. The top graph in each subfigure is the output prediction from the network and the bottom illustrating ground truth comparison where each cell from the output is marked as occupied if the probability exceeds 0.5. The colorbars to the right for the prediction is the measure of how probable each cell is to be occupied and the colorbars for the ground truth comparison is color coded to each cell's classification.

The time to compute one output is presented in Table 4.1, the predictions are made on a NVIDIA Quadro M1000M GPU running Ubuntu with i7 CPU. These prediction times do not include the time it takes to construct the occupancy and visibility grid from the LiDARs point cloud message.

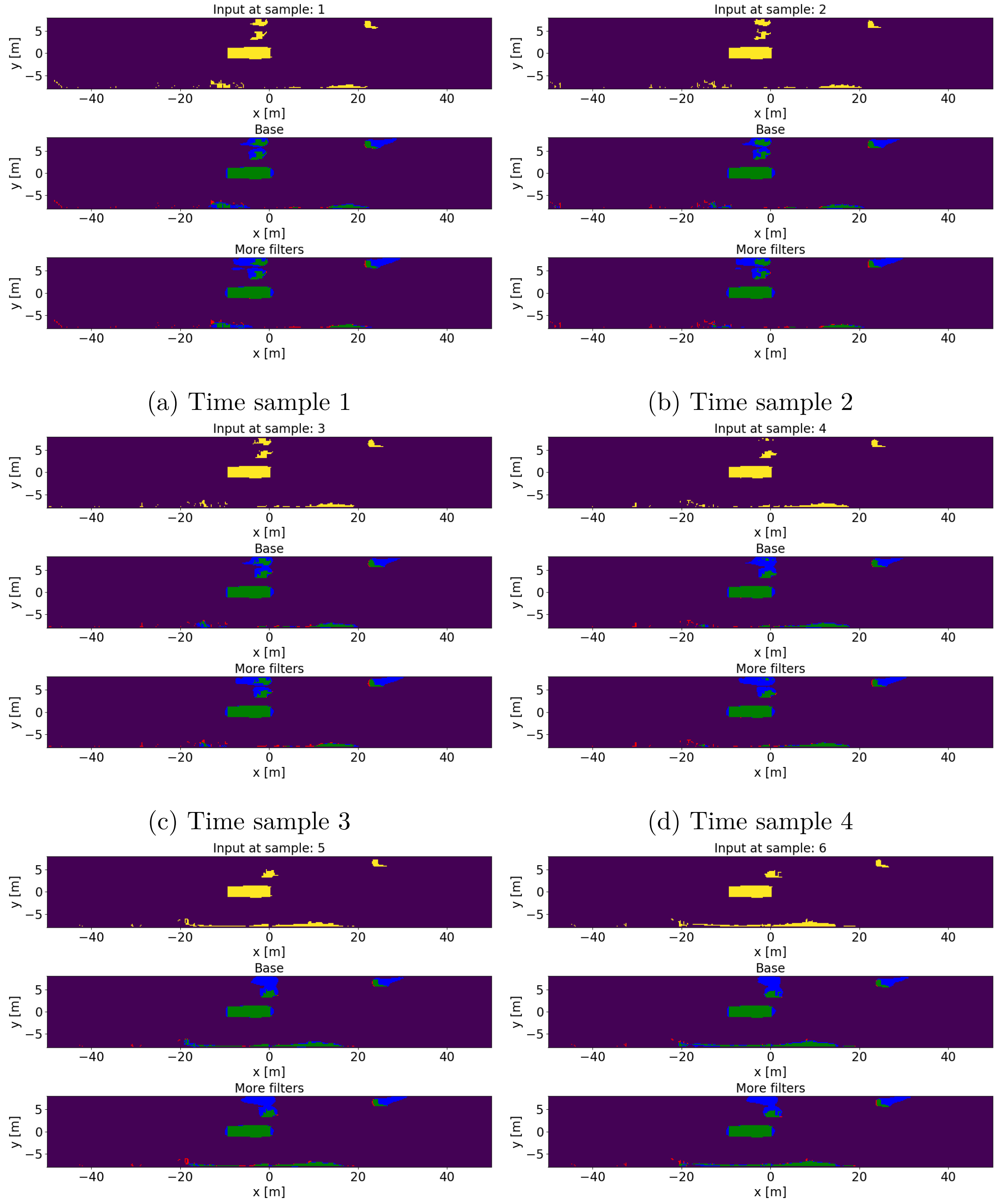
Table 4.1: Time to predict one output frame for the five networks.

Architecture	**Time to predict** [s]
Base	1.081
Less filters	0.583
More filters	1.521
Frames 10/10	1.099
Two layer	0.706

Further evaluation will only be performed on the *Base* and *More filters* network, discussed in chapter 5.

4.3 Occlusion scenario

Ground truth for the fully occluded state of the world is not available, thus forcing evaluation of partly and fully occluded objects to be done manually. One such scenario arise when an adjacent vehicle is occluded by another, i.e. when both vehicles are lined up behind each other seen from the sensor. The *Base* and *More filters* networks are tested on this case and accuracy is evaluated visually. For this test a threshold of 0.5 is used, the result is illustrated in Fig. 4.7 where in each subfigure the top figure is last input, middle is the *Base* network prediction and the bottom figure is the *More filters* prediction. This scenario was not a part of the training set.



(a) Time sample 1

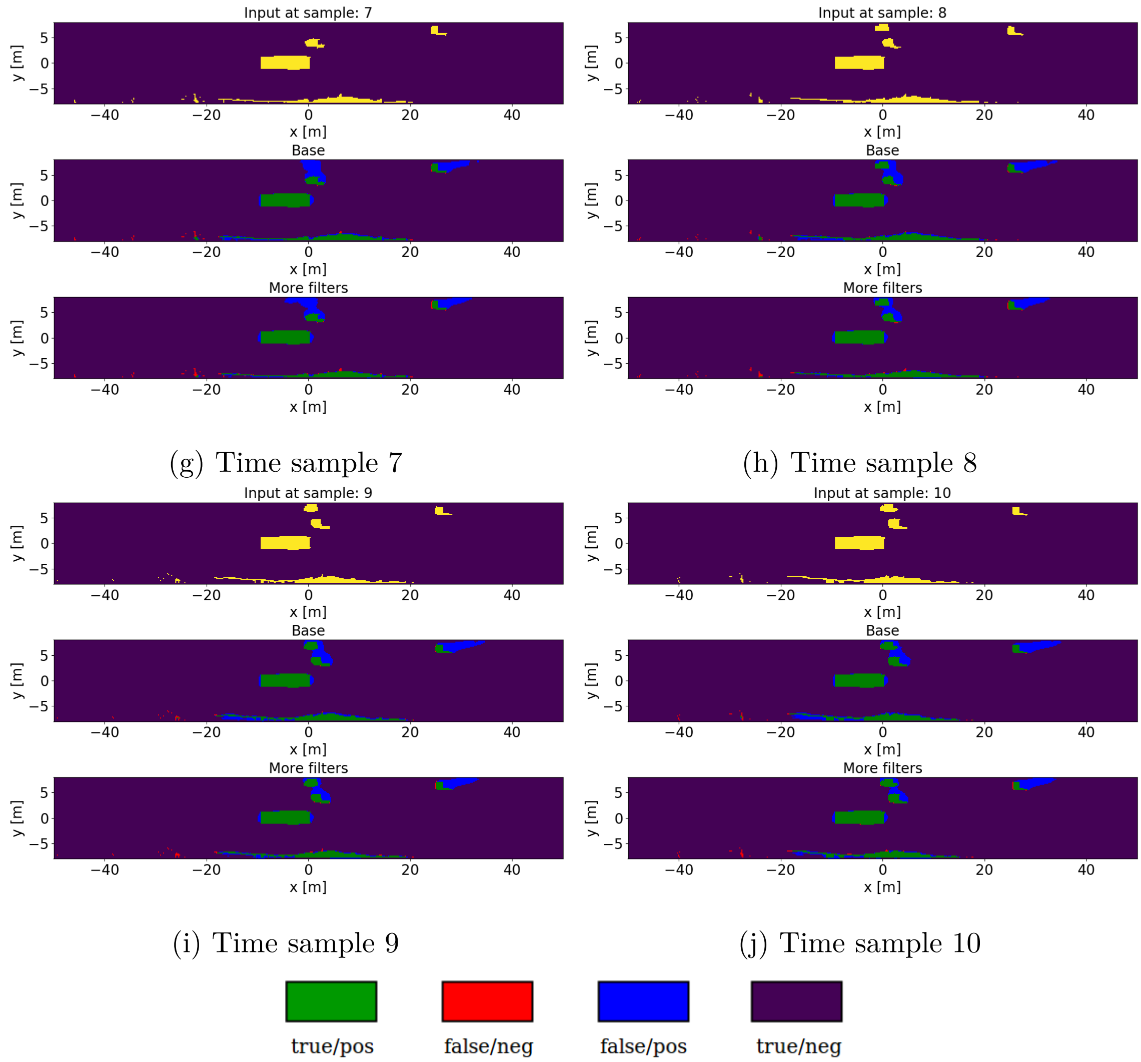
(b) Time sample 2

(c) Time sample 3

(d) Time sample 4

(e) Time sample 5

(f) Time sample 6



(g) Time sample 7

(h) Time sample 8

(i) Time sample 9

(j) Time sample 10

Figure 4.7: Prediction sequence where occlusion between vehicles emerge. The vehicle is partly occluded in sample 4 and fully occluded between sample 5 and 7. The prediction from the *Base* and *More filters* network at each sample, using a threshold of 0.5, is shown below the input. Note that the classification is done with regard to the input sample.

As can be seen in the input sequence, an adjacent vehicle overtakes both the ego vehicle and another road occupant simultaneously by driving in between.

5
Discussion

The result was produced to evaluate how well the networks can describe the surrounding of the truck. In this chapter we will discuss the results and how we interpret different behaviours of the different networks. Each result and scenario will be discussed individually to support the conclusion chapter.

5.1 Method

The approach of using the LiDAR as ground truth has advantages and disadvantages. The obvious advantage is that there is no need for the time consuming task of annotating data. However, this comes at a cost, as the LiDAR is unable to render the fully unoccluded true state of the surroundings the ground truth is not complete. A side effect of this is that some parts are being falsely penalized. This is mostly circumvented by the visibility consideration but as the LiDARs itself has its weaknesses such as missed reflections that not even morphological operations can handle the weakness is not fully managed. Another side effect of this is that seen from the networks point of view, the task is not to interpret measurements to a fixed number of objects, instead there are infinite number of different combinations as the ground truth will change for the slightest difference. What we mean is that the exact same objects ground truth will change at each frame. Hence the network must solve a more complicated problem. However, from the result chapter it is obvious that the approach has potential to solve the task at hand, an analyze of the result will tell whether the result could be considered useful for an automotive application.

An interesting side effect of this training is that it in a real application the network could handle when point cloud messages are lost or not received. As the last part when blank input is given can be seen as when sensor data are missing. Also this could be used to predict future states.

5.2 Network comparison

To obtain as good result as possible we formed neural networks with different attributes, described in section 3.2.3.1.1. The idea of having a *Base* network is that changes in the network architecture can be tracked and thus easily evaluated. From Fig. 4.1 the loss after each epoch during training is visualized. One can see that the loss has started to converge after five epochs for all networks. Since the loss converged the computational time to further train the networks was considered unnecessary, five epochs of training took roughly 2 hours for each network. In mentioned figure, Fig. 4.1, the loss values are noticeably high. Since the loss function has been manipulated we argue that for this result it is more valuable to compare the networks with each other rather than focusing on the values. Comparing the architectures there is no significant difference in performance, except for the *Frames 10/10* network, which indicates that further evaluation is necessary.

In Fig. 4.4 all networks are evaluated on a common scenario for a heavy duty vehicle, being overtaken. One expectation from the score performance is that the network with more filters compared to the *Base* network should have performed better. This since it has more filters that can recognize features such as more specific geometric structures. However, the difference is not that significant, especially for the first 0.6 s of prediction. The explanation for this low improvement we argue could be the similarity of the filter. As can be seen in Fig. 4.2 the *More filters* network has the most similar filters. Similar filters can indicate that the complexity of the problem is lower than what the network can handle, since each filter does not contribute with an unique property. This fact might be an indication that the number of filters in the network could be reduced. Contrarily, the *Less filters* network was expected to perform worse, which in Fig. 4.4 is also shown. It can be seen that the performance up to 0.6 s is quite accurate and similar to the *Base* network, but after that time the performance is significantly worse, this we believe is due to the feature maps providing indistinct information.

Similarly the *Two layer* network was expected to lack ability to handle the scenario compared to the *Base* network. Which is also shown in Fig. 4.4 as the validation score is constantly below the *Base* network. As can be seen in Fig. 4.2 both the *Less filters* and the *Two layer* networks contain few similar filter in their respective layers, indicating that the number of filters might be too low.

The *Frames 10/10* network has almost the same number of equal filter as the *Base* network which is expected since they have the same architectural structure. Seen to the prediction score, it performs worse than the *Base* network up to 0.8 s, but performs better output predictions beyond that time. We argue this is due to that the network rely more on its internal memory, since it has been trained that way. Such behaviour is desired for long term prediction but might impede short term predictions where sensor data is available.

The validation score gives a performance indication for the complete grid, however as smaller objects just occupy a fraction of the grid it is necessary to visually validate the performance. In section 4.2.2 the output for a 5 sample prediction from the simple prediction scenario is visualized, each occupied cell is also classified with regard to ground truth. A common result for all networks is that they output a lot of false positives, however as the ground truth is defined to what the LiDARs can see these are not necessarily false. For example, the thickness of the guardrail depend on the angle of incidence, see Fig. 4.6. Considering the networks performance the most desired feature is minimizing false negatives as those cells are considered empty by the network but is occupied in ground truth. Studying the overtaking scenarios in Fig. 4.6 we believe that the *Base* network and the *More filters* network performs better than the other alternatives, this since they both have quite few false negatives in general but especially at the back of the passing vehicle, all this while maintaining a representative object shape.

It is also of interest to consider the raw prediction of each network, as the certainty about where an object might be is a key for the overall performance. Consider the *Less filters* network and the *Two layer* network, in Fig. 4.6b and 4.6e, the networks shows uncertainty over large portions of the grid, we hypothesize that this behaviour is a result of low confident of the inner states. Further if one study the raw prediction from the *Frames 10/10* in Fig. 4.6d it shows significant uncertainties of the guardrail shape behind the vehicle, which is not a desired behaviour.

Even though network size optimization is not a primarily focus in this thesis it is still of interest to consider the performance gain contra computational complexity. As presented in section 3.2.3.1.1 the number of parameters in the networks differs quite a lot. The computational effect of this can be studied in Table 4.1. As can be seen the prediction time for the network with more filters is almost 50% higher than the *Base* network, while the network with less filter is almost 50% less. However, evaluating the prediction time together with performance one can see in Fig. 4.4 that the *Less filters* performs significantly worse during longer predictions, indicating that there is a trade off between complexity and desired accuracy. Considering the *More filters* network it can be noted that this trade-off fades away as network size increases, this as the prediction time is significantly higher without receiving the corresponding increase in performance as in the *Less filters* contra *Base* network.

At this point it is hard to distinguish whether the *Base* or the *More filters* network is the most suitable for our application. However the other alternative models we believe have shown obvious weaknesses such that they can be disregarded in further evaluation.

5.3 Occlusion scenario

As mentioned in section 1.2, a desired property of the algorithm is to track objects through occlusion. This property is evaluated in section 4.3. The car furthest away in the ground truth gets first partly occluded and then fully occluded in samples 5 to 7, see ground truth in Fig. 4.7d-4.7g. If one look closely to the predictions it can be noticed that the occluded object travels slowly forward relative the ego vehicle. Consequently it is showing that the network has managed to identify the object and its attributes such that it is able to predict its movement through the occlusion. One can also notice that the *More filters* network also manages to track the object but changes the size of the object between samples, see Fig. 4.7g and 4.7h. Our intuitive feeling is that the *More filters* network are more uncertain of the objects shape. Therefore the result from the *Base* network may be of more interest since the shape of the object is kept more constant during the scenario.

From the predictions in the time sequence it can also be observed that the two adjacent objects have grown together, this does not correspond to the true unoccluded state of the world since objects usually have some safety margin to each other. But since the cars lies close to each other the area between them is not driveable since it is too narrow and therefore the problem with concatenation of the objects is not of great importance. Even if the objects have grown together the network consider it as two distinct shapes, the center of these do not move equally which we argue indicates that the network considers them as two independent objects. However, if this output should be used further on for object tracking (calculate speed, acceleration and heading) one would need to split these objects such that they are not perceived as one large object.

5.4 Data set

For the scenarios considered in this thesis it has been shown that the networks performs quite well, this as it has been shown that predictions for a 0.6 s occlusion can be performed with an accuracy of at least 95 %. However, there may be scenarios that it can not handle as good. To be more confident that the network can handle more complex scenarios, one could train the networks on a larger data set compared to the one used in this thesis. The data set size used in this thesis was not so large, only 1200 frames long. The size of the data set was mainly impeded due to that we changed the hyper parameters such as layers, filters, ratio of visible and occluded frames a lot, which resulted in that it took extensive time to train and evaluate the performance of all networks. Furthermore, with a bigger data set the number of similar filters could be reduced since more features can be learned during a longer route. The risk of evaluating on a small data set is that the smaller proposed networks may not be enough to capture all complex features.

6
Conclusion

In this thesis a deep learning approach to identify and track objects in surrounding of the ego vehicle has been considered. The end-to-end deep tracking method have been adopted where the LiDARs are considered as ground truth in training.

As the LiDARs are considered ground truth the implementation of the training procedure was carefully performed. From the first result it was shown that the LiDAR measurements where partly inconsistent, shown by missed reflections along the objects. This issue were addressed and improved by the morphological operations on the sensor measurement grid, resulting in more consistent shapes.

During training, difficulties to learn to accurately track the dynamic objects arose. As the dynamic objects only cover a small portion of the full grid we suspected that their presence where not considered important. By manipulating the loss function and weigh parts of the grid, where objects are common, higher the problem was addressed.

The development could then proceed to construct the network architecture. A large number of architectures with different hyper parameters have been tested to achieve the best result.

From the final result it is clear that the method can be used to develop a perception algorithm for tracking objects surrounding a vehicle. We conclude this as we believe there is no doubt that the results show a network that detects objects and track them.

Considering the different networks we conclude that the best result was given by the *Base* network. As this network had the most consistent performance through the different tests. Seen from the result we argue that this network has learned to detect road occupants and manages to track these through occlusion. Further, the result also show that the network is able to track multiple objects travelling at different speeds. A weakness we see in the network, is that there are uncertainties about the shape of some objects, making it difficult to determine where the objects edges actually are. Another weakness is the prediction time, which is significantly higher than the update frequency of the sensors when using a high performance laptop, depending on the hardware at hand these must most likely improved in order to run online in a vehicle.

The goal of the thesis was to develop an algorithm suitable for accurately describing the surrounding of the tractor using neural networks, see section 1.2. From this thesis work we can conclude that the *Base* network presented in this thesis have shown ability to solve the stated tasks and have potential to be used in the autonomous application of the AutoFreight project.

6.1 Future work

To further improve the result and evaluate the method there are some areas we would consider. One improvement can be to introduce optical flow in each cell. In the highway scenario considered in this thesis the relative ego vehicle movement is not changing significantly, which we hypothesize with support of result in [15] makes it possible for the network to handle without ego movement as input. In a more dynamic environment we believe this movement should be considered as input to the network. Introduction of optical flow we hypothesize would contribute to the overall tracking performance. For example when the truck is driven in more dynamic scenarios where turns appear more often, objects located further away in the grid during turns will move faster than the objects located closer to the ego vehicle.

Another part that could be considered is filtering of points. In this thesis all points are represented in a coordinate system that is defined along the ground when the vehicle is stationary. As there are only small slopes and quite flat roads in highway driving this was sufficient for generating a result. To further improve and allow for slopes and discrepancies of the road a more sophisticated filtering method should be chosen. Some literature study on this topic has been performed and concluded in that plane fitting using *Random sample consensus* (RANSAC) should be evaluated. The property a plane fitting algorithm would give is to get more consistent measurements while filtering ground reflections more accurately.

Finally it would be interesting to benchmark the presented work against other approaches on public data set, this would set the work in a context such that its true potential could be evaluated.

Bibliography

[1] S. Lohmander, Transport av livsmedel (2007).
URL https://www.livsmedelsverket.se/globalassets/produktion-handel-kontroll/krisberedskap/krisberedskap-och-sakerhet---livsmedel/transport-av-livsmedel.-livsmedelsverket.pdf

[2] T. Zielke, M. Brauckmann, W. von Seelen, Intensity and edge-based symmetry detection applied to car-following, in: G. Sandini (Ed.), Computer Vision — ECCV'92, Springer Berlin Heidelberg, Berlin, Heidelberg, 1992, pp. 865–873.

[3] F. Dellaert, C. Thorpe, Robust car tracking using kalman filtering and bayesian templates, in: Conference on Intelligent Transportation Systems, 1997.

[4] A. Petrovskaya, S. Thrun, Model based vehicle tracking for autonomous driving in urban environments (June 2008).

[5] C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, H. Durrant-Whyte, Simultaneous localization, mapping and moving object tracking, The International Journal of Robotics Research 26 (9) (2007) 889–916. arXiv:https://doi.org/10.1177/0278364907081229, doi:10.1177/0278364907081229.
URL https://doi.org/10.1177/0278364907081229

[6] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.
URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[7] Vinnova, Highly automated freight transports (Accessed May 23, 2018).
URL https://www.vinnova.se/en/p/highly-automated-freight-transports2/

[8] SAE International, SAE J3016 Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles, Tech. rep. (2016).

[9] P. Nilsson, L. Laine, J. Sandin, B. Jacobson, O. Eriksson, On actions of long combination vehicle drivers prior to lane changes in dense highway traffic – a driving simulator study, Transportation Research Part F: Traffic Psychology and Behaviour 55 (2018) 25 – 37. doi:https://doi.org/10.1016/j.trf.2018.02.004.
URL http://www.sciencedirect.com/science/article/pii/S1369847817300256

[10] S. Gupta, R. B. Girshick, P. Arbelaez, J. Malik, Learning rich features from RGB-D images for object detection and segmentation, CoRR abs/1407.5736.

arXiv:1407.5736.
URL http://arxiv.org/abs/1407.5736

[11] M. Schwarz, H. Schulz, S. Behnke, Rgb-d object recognition and pose estimation based on pre-trained convolutional neural network features, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 1329–1335. doi:10.1109/ICRA.2015.7139363.

[12] R. Socher, B. Huval, B. Bath, C. D. Manning, A. Y. Ng, Convolutional-recursive deep learning for 3d object classification, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 656–664.
URL http://papers.nips.cc/paper/4773-convolutional-recursive-deep-learning-for-3d-object-classification.pdf

[13] B. Li, T. Zhang, T. Xia, Vehicle detection from 3d lidar using fully convolutional network, CoRR abs/1608.07916. arXiv:1608.07916.
URL http://arxiv.org/abs/1608.07916

[14] A. Geiger, P. Lenz, R. Urtasun, Are we ready for autonomous driving? the kitti vision benchmark suite, in: Conference on Computer Vision and Pattern Recognition (CVPR), 2012.

[15] J. Dequaire, P. Ondrúška, D. Rao, D. Wang, I. Posner, Deep tracking in the wild: End-to-end tracking using recurrent neural networks, The International Journal of Robotics Research.
URL https://doi.org/10.1177/0278364917710543

[16] Volvo, Media gallery (may 2018).
URL https://www.volvotrucks.se/sv-se/trucks/volvo-fm/media-gallery.html

[17] Trafikverket, Krav för vägars och gators utformning (2015).

[18] P. McManamon, Field guide to lidar, Vol. FG36, SPIE Press, 2015.

[19] P. Lindner, G. Wanielik, 3d lidar processing for vehicle safety and environment recognition, in: 2009 IEEE Workshop on Computational Intelligence in Vehicles and Vehicular Systems, 2009, pp. 66–71. doi:10.1109/CIVVS.2009.4938725.

[20] R. H. Rasshofer, M. Spies, H. Spies, Influences of weather phenomena on automotive laser radar systems, Advances in Radio Science 9 (2011) 49–60. doi:10.5194/ars-9-49-2011.
URL https://www.adv-radio-sci.net/9/49/2011/

[21] A. Elfes, Using occupancy grids for mobile robot perception and navigation, Computer 22 (6) (1989) 46–57. doi:10.1109/2.30720.

[22] A. Butterfield, G. E. Ngondi, Ray tracing.
URL http://www.oxfordreference.com/view/10.1093/acref/9780199688975.001.0001/acref-9780199688975-e-4335

[23] Q. Wu, Z. Lu, T. Ji, Protective Relaying of Power Systems Using Mathematical Morphology, 1st Edition, 1612-1287, Springer-Verlag London, 2009.

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

[25] S. Shanmuganathan, S. Samarasinghe, S. O. service), S. (e-book collection), Artificial Neural Network Modelling, 1st Edition, Vol. 628, Springer International Publishing, Cham, 2016.

[26] H. Zhou, Z. Li, Deep networks with non-static activation function, Multimedia Tools and Applicationsdoi:10.1007/s11042-018-5702-5.
URL https://doi.org/10.1007/s11042-018-5702-5

[27] X. Chen, S. Kar, D. A. Ralescu, Cross-entropy measure of uncertain variables, Information Sciences 201 (2012) 53 – 60. doi:https://doi.org/10.1016/j.ins.2012.02.049.
URL http://www.sciencedirect.com/science/article/pii/S0020025512001697

[28] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536.

[29] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (2011) 2121–2159.

[30] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, CoRR abs/1207.0580. arXiv:1207.0580.
URL http://arxiv.org/abs/1207.0580

[31] A. P. Piotrowski, J. J. Napiorkowski, A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling, Journal of Hydrology 476 (2013) 97–111.

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

[33] A. Ng, J. Ngiam, C.-Y. Foo, et al., Convolutional neural network ufldl tutorial (2018).
URL http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[34] F. Piewak, Fully convolutional neural networks for dynamic object detection in grid maps (masters thesis), CoRR abs/1709.03138. arXiv:1709.03138.
URL http://arxiv.org/abs/1709.03138

[35] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks?, CoRR abs/1411.1792. arXiv:1411.1792.
URL http://arxiv.org/abs/1411.1792

[36] S. Leroux, S. Bohez, C. D. Boom, E. D. Coninck, T. Verbelen, B. Vankeirsbilck, P. Simoens, B. Dhoedt, Lazy evaluation of convolutional filters, CoRR abs/1605.08543. arXiv:1605.08543.
URL http://arxiv.org/abs/1605.08543

[37] Volvo fh16 demotruck (may 2018).
URL https://commons.wikimedia.org/wiki/File:2013_Volvo_FH16_540_demotruck.jpg

[38] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, CoRR abs/1511.07122. arXiv:1511.07122.
URL http://arxiv.org/abs/1511.07122

[39] V. Dumoulin, F. Visin, A guide to convolution arithmetic for deep learning, ArXiv e-printsarXiv:1603.07285.

[40] A. Milan, S. H. Rezatofighi, A. R. Dick, K. Schindler, I. D. Reid, Online multi-target tracking using recurrent neural networks, CoRR abs/1604.03635. arXiv:

1604.03635.
URL http://arxiv.org/abs/1604.03635

[41] C. Olah, Understanding lstm networks (aug 2015).
URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[42] K. Rao, H. Sak, R. Prabhavalkar, Exploring architectures, data and units for streaming end-to-end speech recognition with rnn-transducer, CoRR abs/1801.00841. arXiv:1801.00841.
URL http://arxiv.org/abs/1801.00841

[43] J. Mazumdar, R. G. Harley, Recurrent neural networks trained with backpropagation through time algorithm to estimate nonlinear load harmonic currents, IEEE Transactions on Industrial Electronics 55 (9) (2008) 3484–3491. doi:10.1109/TIE.2008.925315.

[44] F. A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: Continual prediction with lstm, Neural Computation 12 (10) (2000) 2451–2471. arXiv:https://doi.org/10.1162/089976600300015015, doi:10.1162/089976600300015015.
URL https://doi.org/10.1162/089976600300015015

[45] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.
URL http://dx.doi.org/10.1162/neco.1997.9.8.1735

[46] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, CoRR abs/1409.3215. arXiv:1409.3215.
URL http://arxiv.org/abs/1409.3215

[47] A. Graves, Generating sequences with recurrent neural networks, CoRR abs/1308.0850. arXiv:1308.0850.
URL http://arxiv.org/abs/1308.0850

[48] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, W. Woo, Convolutional LSTM network: A machine learning approach for precipitation nowcasting, CoRR abs/1506.04214. arXiv:1506.04214.
URL http://arxiv.org/abs/1506.04214

[49] About tensorflow (2018).
URL https://www.tensorflow.org/

[50] Keras: The python deep learning library (2018).
URL https://keras.io/

A

Appendix 1

Table A.1: Velodyne VLP-32C Lidar specifications

Sensor	• Channels: 32 • Measurement Range: 200 m • Range Accuracy: Up to ±3 cm (Typical) 1 • Horizontal Field of View: 360° • Vertical Field of View: 40° (-25° to +15°) • Minimum Angular Resolution (Vertical): 0.33° (non-linear distribution) • Angular Resolution (Horizontal/Azimuth): 0.1° to 0.4° • Rotation Rate: 5 Hz to 20 Hz • Integrated Web Server for Easy Monitoring and Configuration
Laser	• Laser Product Classification: Class 1 – Eye-safe per IEC60825-1:2014 • Wavelength: ∼903 nm
Mechanical/ Electrical/ Operational	• Power Consumption: 10 W (Typical) 2 • Operating Voltage: 10.5 V – 18 V (with interface box and regulated power supply) • Weight: ∼925 g (typical, without cabling and interface box) • Dimensions: See diagram on previous page • Environmental Protection: IP67 • Operating Temperature: -20°C to +60°C • Storage Temperature: -40°C to +85°C
Output	• 3D LiDAR Data Points Generated: - Single Return Mode: ∼600,000 points per second - Dual Return Mode: ∼1,200,000 points per second • 100 Mbps Ethernet Connection • UDP Packets Contain: - Time of Flight Distance Measurement - Calibrated Reflectivity Measurement - Rotation Angles - Synchronized Time Stamps (µs resolution) • GPS $GPRMC and $GPGGA NMEA Sentences from GPS Receiver (GPS not included)