



Universitetet
i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY


MASTER'S THESIS

Study programme specialisation: Spring / Autumn semester, 2017

Industrial Automation and Signal Processing

Open Confidential: Open

Author: Eirik Aflekt Thomessen


(signature of author)

Programme coordinator: Ivar Austvoll

Supervisor(s): Ivar Austvoll, Erlend Tøssebro

Title of master's thesis: Advanced vision based vehicle classification for traffic surveillance system using neural networks

Norwegian: Avansert visjonbasert kjøretøyklassifisering for trafikkovervåkingssystem ved bruk av nevrale nettverk

Credits: 30 ECTS

Keywords: Traffic surveillance, background subtraction, Kalman filter, Deep learning, Neural networks

Number of pages: 118

+ supplemental material other: 23

Stavanger, 15.06.2017
date year

Preface

This is a master's thesis in automation and signal processing at the University of Stavanger. The topic is advanced vision based vehicle classification for traffic surveillance systems using neural networks. I would like to thank my supervisors, Ivar Austvoll and Erlend Tøssebro, for valuable discussions and support to successfully accomplish my MSc Thesis.

Also great thanks to Bjørn Inge Lerang from Roxel, Håkon Kjerkreit and other guys from KVS, and prof Trygve Thomessen for great help, ideas and enthusiasm. And thanks to Bjørn Fossåen and Ståle Freyer for acquisition of cameras.

A special thanks to Ingrid, to always supporting me during the work, and for your endless patience.

**Advanced vision-based vehicle
classification for traffic
surveillance systems using neural
networks**

Summary

During the last years, the use of vision-based traffic system has increased in popularity, both in terms of traffic monitoring and control of autonomous cars.

This master thesis focus especially on traffic monitoring, which are of importance to fulfill planning and traffic management of road networks.

An important requirement is data interpretation accuracy to provide adequate characteristic data from the acquired vision-data. A vision-based system has been developed, using new methods and technologies to achieve an automated traffic monitoring system, without the use of additional sensors.

The thesis is based upon Erik Sudland's master thesis from 2016, which investigated available literature containing adequate algorithms for traffic monitoring. However in the current master thesis, methods have been further analyzed and experimentally optimized on vision-data from real traffic situations. In addition, a new classification method based upon neural networks has been implemented and verified with successful results.

The system has undergone a comprehensive experimental verification, with analysis of more than 20000 images. The experimental results verify a successful implementation of both the detection and object classification routines, and demonstrate the system's capability of determining characteristic traffic data, like

- Velocity distribution
- Density of vehicles
- Traffic congestion
- Vehicle class frequency

Sammendrag

I løpet av de siste årene har bruken av kamerabaserte trafikksystemer økt i popularitet, både når det gjelder trafikkovervåkning og kontroll av autonome biler. Denne masteroppgaven fokuserer spesielt på trafikkovervåkning, noe som er viktig for å oppfylle planlegging og trafikkstyring av veinett.

Et viktig krav er at datatolkningen er nøyaktig nok til å gi tilstrekkelig karakteristiske analyser. Et kamerabasert system er utviklet ved hjelp av nye metoder og teknologier for å oppnå et automatisert trafikkovervåkingssystem uten bruk av tilleggs-sensorer.

Avhandlingen er basert på Erik Sudlands masteroppgave fra 2016, som undersøkte tilgjengelig litteratur om algoritmer for trafikkovervåkning. I denne masteroppgaven er metodene videre analysert og eksperimentelt optimalisert på data fra ekte trafikksituasjoner. I tillegg er en ny klassifikasjonsmetode, basert på nevralt nettverk, implementert og verifisert med vellykkede resultater.

Systemet har gjennomgått en omfattende eksperimentell verifisering, med analyse av mer enn 20000 bilder. De eksperimentelle resultatene bekrefter en vellykket implementering av både deteksjons- og objektklassifikasjonsrutiner, og demonstrerer systemets evne til å bestemme karakteristiske trafikkdata, slik som:

- Hastighetsfordeling
- Trafikkfrekvens
- Køsituasjoner
- Frekvens av kjøretøyklassene

Contents

Preface	I
Summary	III
1 Introduction	1
1.1 Background	1
1.2 Introduction to object identification	3
1.3 Current available technology	5
1.4 Problem formulation	7
2 Theory behind implemented methods	10
2.1 Background subtraction	11
2.1.1 Morphological operations	15
2.2 Kalman filter	18
2.3 Artificial neural networks	25
2.4 Classifying images with neural networks	35
3 Implementation of the vision-based traffic system	40
3.1 Hardware components	42
3.2 Implementation in Python	45
3.2.1 System setup	47
3.2.2 Detection module	49
3.2.3 Tracking module	53
3.2.4 Classification module	59
3.2.5 Graphical Interface	62

4	Experimental results	64
4.1	Data acquisition	65
4.2	Testing neural networks architectures	70
4.2.1	Testing the classifier	75
4.3	Arranging the experimental setup	78
4.4	Initial considerations of the acquired data	82
4.4.1	Determining a reasonable vector space	83
4.4.2	Initiating the Kalman filter	87
4.4.3	Hit rate during sunny days	89
4.4.4	Hit rate during cloudy days	92
4.4.5	Recordings from three different locations	94
4.4.6	Multiple lanes	96
4.5	Analysis and presentation of traffic data	98
4.5.1	Velocity distribution	99
4.5.2	Density of vehicles	101
4.5.3	Traffic congestion	103
4.5.4	Classifying the dataset	106
4.5.5	Vehicle class frequency	108
5	Discussion	110
6	Recommendations for further work	115
7	Conclusion	118
	Bibliography	120
	Appendices	130
A	Python libraries	131
B	Datasheets	133
C	Neural network models	137

<i>Chapter 0</i>	<i>Contents</i>
D Source code	140
E User manual	141

Chapter 1

Introduction

1.1 Background

In recent years, big data applications utilized in real-time traffic operation and safety monitoring has gained interest. The IP-based surveillance segment is expected to witness high growth over the next years. According to a new report from the US Market Research Institute, Grand View Research [36], states that the world market for video surveillance and video surveillance to will grow to as much as 49 billion dollars by 2020, which is equivalent to approximately NOK 390 billion.

The primary purpose of a surveillance camera is for security and statistics purpose, such as open data governmental surveillance of traffic, and planning new solutions to reduce risks, and increase traffic flow. Furthermore, this data can be used for risk assessment purposes through analysis providing predictions to current risk levels, and possibly enable proactive mobilization of emergency units.

Traffic surveillance is a method that improves traffic management and flow, and is often referred to as an intelligent transportation system (ITS) [41]. ITS are used in several applications, such as to identify vehicles traveling over the legal speed limit, detect vehicles in the wrong direction, driving on red light or vehicles crossing railways that grade illegally. These camera systems are often used in

combination with a range of sensors to recognize vehicles.

A vision-based approach is introduced. It has the advantages of easy maintenance and high flexibility in traffic monitoring and compact hardware and software structure which enhanced the mobility and performance. The deployed cameras utilize a vehicle detection algorithm that detects cars and performs analyzes based on the data mining. Applying camera systems provides additional benefits as they can detect people in the way, alarm if someone is driving the wrong way or if is a traffic jam.

Autonomous cars are expected to be an important part of our everyday life in just a few years. Combining the vehicles technology system with a robust surveillance system could prevent traffic jams, accidents or other events and drastically improve congestion.

1.2 Introduction to object identification

Motion detection is often the first step of a multi-stage computer vision system. The problem to recognize and monitor vehicles is normally separated into three main operations; *detection, tracking and classification*.

Detection is the process of localizing objects in the scene. A survey on object detection and tracking methods[19] proposed that background subtraction can be a simple method providing complete information about an object compared to optical flow and frame difference for detecting objects.

Tracking is the problem of localizing the object in consecutive frames. Erik Sudland [43] proposed an interesting algorithm for object tracking based upon a Kalman filter to estimate the unknown states of the objects. Even though the Kalman filter has some weaknesses when the background is varying [2], promising results were demonstrated in conjunction with background subtraction [38].

Classification is the process of categorizing the objects. Vehicle classification is an inherently difficult problem, because many vehicle types do not have any distinct signatures. The traffic situation in the real world is constantly changing and cameras will be challenged by occlusions, shadows, camera noise, changes in illumination and weather, etc. In addition, each vehicle category (car, van, bus etc) contains multiple variants of geometry, colour, size, styling etc. which makes it difficult to classify based on simple parameters [10]. This task becomes even more challenging, when subcategories are included.

The development of artificial neural networks, has contributed to a significant improvement of the computer vision tasks during the recent years. The neural networks are frequently used for machine learning and has a special strength in the object classification. The neural network method is based upon offline training on object with known properties, and has the capability to extend this trained

knowledge, to detect unknown object properly. In addition, a big strength with the neural networks, is the small consumption of computation power.

1.3 Current available technology

Erik Sudland[43] presented in his MSc Thesis, an overview of the existing technology and algorithms for vehicle recognition. Among these are inductive loops, pressure sensitive sensors, radars, lazer, ultrasound and infrared light. The overall drawbacks were either high cost or low reliability due to wear and tear and sensibility to challenging weather conditions.

Image processing has been used for traffic monitoring and -surveillance since the 1970s in the USA, Japan and France. Typically, video cameras with image processing are used for vehicle detection. However, the availability of new technology and especially computation power during the last few years have opened for implementation of new and advanced algorithms for even real-time analysis.

Today a fair number of tunnels around the world have camera systems that automatically alerts about abnormal traffic conditions like pedestrians in the tunnel, vehicles in the wrong lane, slow traffic, smoke, dropped load and overheating in vehicles. The system goes under the name Automatic Incident Detection(AID).

The technology also makes is possible to detect traffic in several lanes inside of the field of view. Statens Vegvesen is using an ATK (Automatic traffic control) system which matches cars over a certain distance to measure the average speed [47]. Key data, like time of passage, wheelbase, weight and license plate, are sent from "photo box A" to "photo box B" and is used to recognize the vehicles.

Even though this method is based on basically, simple license plate detection, it has some important challenges related to flaws like, dirt on the plate, weather conditions (rain, snow) etc, making the license plate unreadable, even though the cameras high resolutions images. Because of these flawed factors, the system also has installed pressure sensitivite sensors which detect speed, weight and wheelbase. In total, this leads to high costs and sometimes low performance and need for fre-

quent maintenance. However, currently, there are interesting projects running for video based traffic monitoring and surveillance. The project VITUS (Video Image analysis for Tunnel Safety) was completed in Austria. Even though the project acquired important knowledge about vision technology, it unfortunately, concluded final results it did not achieve a satisfactory level.

Another project, Robust Sensor Systems for Advanced Traffic Applications (ROSSATA), applied more advanced methods, like passive 3D vision and flow analysis to examine 3D-scenes. This project is running.

However, this MSc thesis, has the ambitions to investigate and experimentally, verify, partly new algorithms and methodology. This has implied a considerable risk. However, motivation have occurred of the opportunity to make a possible break-through within vision based traffic surveillance systems.

1.4 Problem formulation

This master thesis is enabled and motivated by a collaboration between the university in Stavanger and Statens vegvesen. The target has been twofold;

- Develop a powerful, reliable, traffic surveillance system to detect traffic situations like:
 - Velocity distribution
 - Density of vehicles
 - Traffic congestion
 - Vehicle class frequency
- Develop, adapt and implement new detection and classification algorithms, and make a representative verification of them through testing on a huge source of real traffic data

Thus, this thesis basically, try to successfully fulfill the expectations from two “customers”. The system should be based upon vision and be able to anonymously detect and determine different classes of vehicles. “Anonymous” is used in the terms of not using any identification features, such as code chip, vehicle registration plates etc. The case is described in figure 1.1.

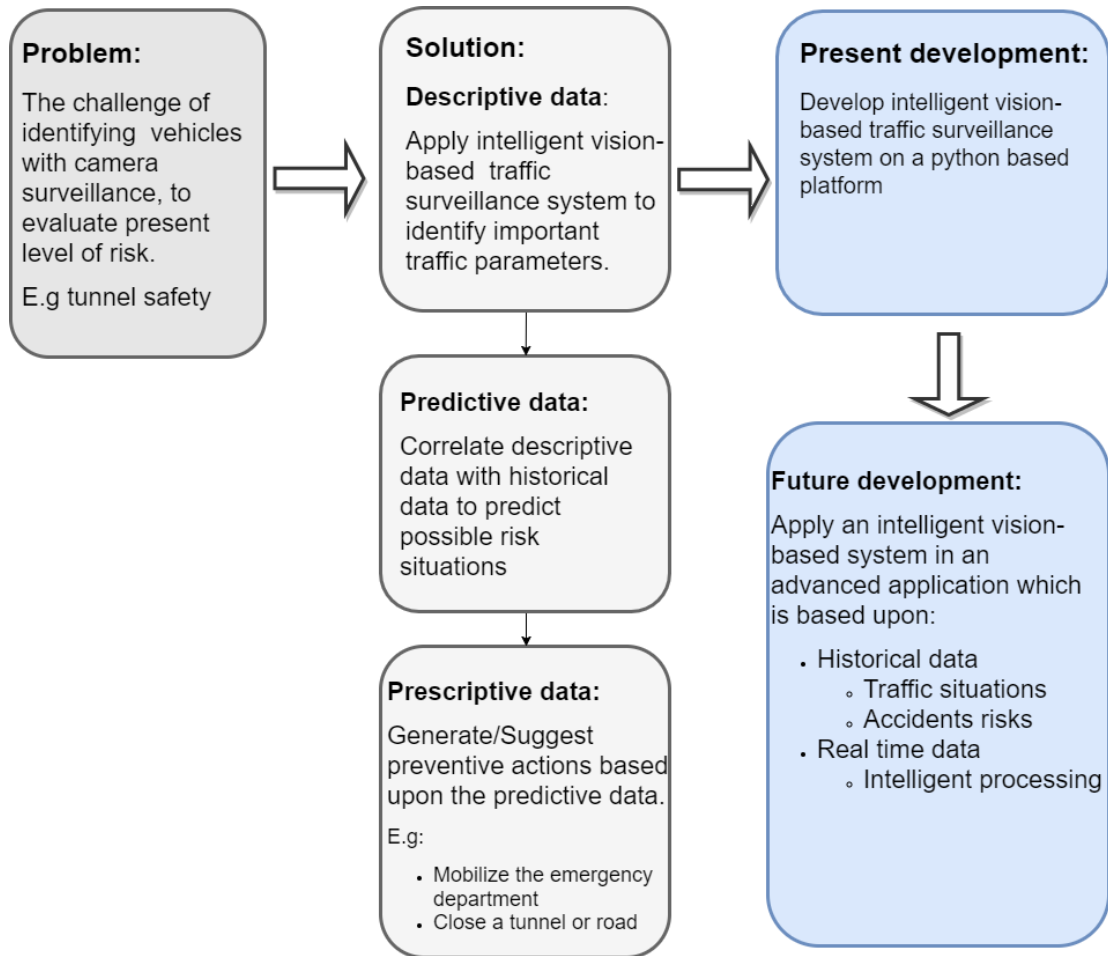


Figure 1.1: Problem, solution and development for a traffic surveillance system

The problem description has during close correspondence with the supervisors, been somewhat adjusted during the development process, to comply with the acquired knowledge during the development process.

Thus, the target has been move away from tunnel safety, toward a more general traffic surveillance system, with vision and neural network classification as core methodologies.

The thesis is organize with the following main chapter:

1. Introductory study to determine the need for and challenges of traffic surveillance systems
2. Introductory study of the detection and classification methods
3. Develop a module for detection, tracking and classification respectively
4. Develop a complete system structure as a surveillance system
5. Collect data for testing
6. Experimental testing for all modules
7. Documentation of the experimental setup, including hardware and software
8. Documentation of the user functions to operate the experimental setup.
9. Documentation of the experimental results

Thus, the reader is brought through basic theory, principles and methodology before diving into the comprehensive experiments to verify the results.

Chapter 2

Theory behind implemented methods

The traffic surveillance system in this thesis utilizes a range of image processing methods and machine learning algorithms. The processes are placed in separate modules, where the entirety of the system is presented in figure 2.1.

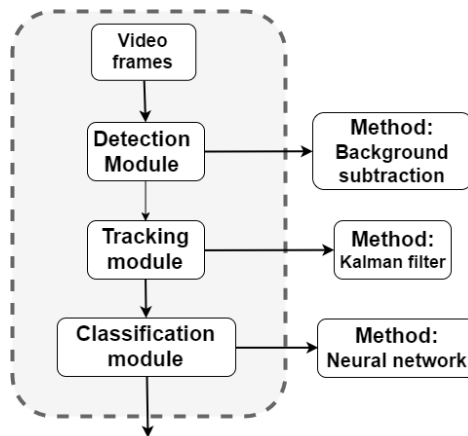


Figure 2.1: Modules in the system.

The *detection module* is solved by implementing a background subtraction method. The *tracking module* is solved with a Kalman filter. The *classification module* utilizes a neural network. The theory behind the implemented methods in each module are explained separately in the next chapter.

2.1 Background subtraction

Background subtraction, also referred to as *foreground detection*, is a method to detect a dynamic foreground in a static background image without any prior knowledge about the objects [19]. Background subtraction is a widely used approach for detecting moving objects in videos from static cameras, as well as other monitoring applications [22] [45]. The method segments moving objects by thresholding a pixel distance between the current frame and static background image.



Figure 2.2: Extract foreground based on movement in the background image

The basic approach is to maintain a background image as a cumulative average of the video stream. "The simplest process" is explained by the equation 2.1:

$$|frame_i - frame_{i-1}| > Threshold \quad (2.1)$$

Where $frame_i$ is the current frame, and the estimated background, $frame_{i-1}$, is the previous frame. Objects are segmented by thresholding a pixel distance between the current frame and the background image. This is very sensitive to the global threshold, and works only under special conditions, where the background is unaffected by uncontrolled environments [26]. A simple subtraction difference with global threshold is a weak solution, because the background subtraction method must deal with problems as illumination changes, motion changes and changes in background geometry [12].

Illumination variations may both be gradual and sudden, such as clouds or blinking lights. Motion changes may be camera oscillations or background objects that are moving frequently. Changes in the background geometry may be moving vehicles. To counteract uncontrolled environments, most of the state of the art algorithms uses a sequence of previous images to create a historical and probabilistic model. Several background methods bound to different probability models have been proposed, and a paper conducted by Massimo Piccardi[33], reviews and compares the state of the art background subtraction methods. The experiments concludes that all methods outperforms the basic method, especially with additional image processing, such as morphological operations. One highlighted and promising method is the Gaussian Mixture Model.

Gaussian Mixture Model (GMM) is a model proposed by Stauffer and Grimson to tolerate environmental changes [42]. Considering that environmental factors contributes to the background pixel values, the GMM uses probability of occurrence of a color at a given pixel, as a mixture of K numbers of Gaussians [8], as illustrated in figure 2.3.

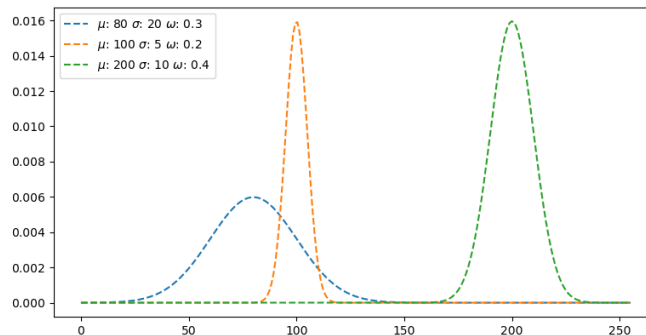


Figure 2.3: Threshold area for a pixel, $I_{(s,t)}$, given by a mixture of Gaussian distributions. Values within the distributions are considered background.

The probability of a given pixel being a part of the background is given by the equation:

$$P(\mathbf{I}_{s,t}) = \sum_{k=1}^K \omega_{k,s,t} \cdot \mathcal{N}(\boldsymbol{\mu}_{k,s,t}, \boldsymbol{\Sigma}_{k,s,t}) \quad (2.2)$$

$P(\mathbf{I}_{s,t})$ is probability of the color $\mathbf{I}_{s,t}$ at *time* t and *pixel* s . $\mathbf{I}_{s,t}$ may be one-dimensional, such as gray scale, 2D (normalized colour space) or 3D (colour, RGB space). $\mathcal{N}(\boldsymbol{\mu}_{k,s,t}, \boldsymbol{\Sigma}_{k,s,t})$ is the k^{th} Gaussian model and $\omega_{k,s,t}$ are the corresponding weights. All weights ω are updated for each consecutive frame. $\boldsymbol{\Sigma}_{k,s,t}$ is the covariance matrix. The RGB components are assumed to be uncorrelated and share the same variance, hence the covariance matrix given by $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$, where σ^2 is the variance. The parameters in the model are updated as following:

$$\omega_{k,s,t} = (1 - \alpha)\omega_{k,s,t-1} + \alpha \quad (2.3)$$

$$\boldsymbol{\mu}_{k,s,t} = (1 - \rho)\boldsymbol{\mu}_{k,s,t-1} + \rho\mathbf{I}_{s,t} \quad (2.4)$$

$$\sigma_{k,s,t}^2 = (1 - \rho)\sigma_{k,s,t-1}^2 + \rho(\mathbf{I}_{s,t}, \boldsymbol{\mu}_{k,s,t}) \quad (2.5)$$

where α and ρ are learning rates, which decides how fast the model should adapt to changes in the background. Faster learning rates results in a more sensitive model. To achieve decay in the background, the weights of unmatched distributions are reduced over time:

$$\omega_{k,s,t} = (1 - \alpha)\omega_{k,s,t-1} \quad (2.6)$$

A pixel is set to be background with higher probability if it occurs frequently (high ω_k and does not vary much (low σ^2). At every frame, some of the Gaussian pdfs are matching the current value. For the matching values, ω_k and σ_k is updated.

If the color $I_{s,t}$ does not match any apriori probabilities, the distribution with lowest weights are replaced by a Gaussian with:

- mean of $I_{s,t}$
- small weights ω
- large initial variance, σ^2

All of the K distributions are ranked by the criterion ω_k/μ_k , which is proportional to the peak amplitude of the weighted distributions. Detected objects are in motion and a distribution representing the foreground will have greater variance and less road factor, and therefore the B most reliable distributions are chosen as part of the background.

$$B = \underset{h}{\operatorname{argmin}} \left(\sum_{i=1}^h \omega_k > \tau \right) \quad (2.7)$$

The distributions that exceeds the threshold τ are set as background, and the rest are default foreground. The foreground is extracted followed by morphological operations, which is presented in next section.

2.1.1 Morphological operations

Morphological means, in mathematics, shape, form or structure. It's a set of non-linear methods related to the shape or of features in an image [35]. It has been used for image processing since the early 1960, and was introduced by Georger Matheron and Jean Serra [35]. When first introduced it was only applicable for binary images, but later developed for grayscale as well. Morphological image processing has a lot of applications, but it's especially useful for extracting and describing image regions. It is based on a set of basic methods, which are applied in different ways. Morphological operations are based on structuring elements (shortened to strel). A strel is a small set of pixels or subimage, used to probe for structure.

Morphological operations may be applied to the extracted foreground to prevent noise and false positives. One of the advantages of morphological operations is that they require little computing power, and won't affect the real time processing. Morphological operations are often combined to enhance specific features in an image. The basic operations are explained below.

Erosion is a set of points Z , so that the structuring element, translated by Z , fits fully inside A . A is the original set, B is the structuring element, given by the equation:

$$A \ominus B = \{Z | B_z \subseteq A\} \quad (2.8)$$

The outcome of erosion will always give a subset of A . It can be seen as a peeler, it removes thin lines and isolated dots, but leaves gross details.

Dilation find pixels in which the shifted strel has overlap with the original set, A. In other words, it fatten things up.

$$AB = \{Z|\hat{B}_z \cap A \subseteq A\} \quad (2.9)$$

Dilation is both commutative (2.10)

$$A \oplus B = B \oplus A \quad (2.10)$$

and associative (2.11):

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C \quad (2.11)$$

Both *dilation* and *erosion* are changing the size of the objects. A mix of these two gives operators that opens and/or fill holes, but does not change the original size. They are called Opening and Closing

Opening is the result of eroding, then dilating, with the same structuring element.

$$A \circ B = (A \ominus B) \oplus B \quad (2.12)$$

The eroding breaks bridges and eliminate thin structures. The dilating adds size to the object, so that it keeps its original size. Typically used to separate regions.

Closing

$$A \bullet B = (A \oplus B) \ominus B \quad (2.13)$$

Result of dilation, followed by erosion. Union of all translates of B that does not intersect with A. This method fuses narrow breaches and eliminates small holes.

The basic morphological operators structuring element contains foreground pixels and zeros. The operators are deduced from combinations of dilation and erosion. They are used to remove noise to either suppress or enhance features in a given image.

More advanced operators, such as *hit and miss*, applies erosion with a pair of disjoint structuring elements, where a pixel is set to foreground if the background pixels corresponds exactly to the structuring element. e.g searching for a corner can be done with a kernel with the structure:

$$\begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \quad (2.14)$$

where -1 corresponds to the values for the first structuring element, and 1 disjointed one. This structuring element gives hit when it finds left corners that are exactly 90 degrees.

One of the benefits with this operation is that it also takes the background pixels into account. The pixel is set to background if there is no match. Operators like hit-or-miss are used to simplify the structure of an object, while preserving its structure.

2.2 Kalman filter

Kalman filter is a linear optimal filtering for computer vision system, which applies to stationary as well as nonstationary environments [14]. The word *filter* is used because it filters out the noise to find the best estimate, and projects the measurements onto the state estimate. It is a recursive filter, since current state depends on previous state. It is known from the theory that the Kalman filter is optimal with the following requirements[16]:

- The model fits perfectly with the underlying system (motion model)
- The noise is normally distributed
- The covariance of the noise are known

The Kalman filter proposes advantages in vision based tracking when tracking congested traffic scenes because it tolerates small occlusions. A limitation of the Kalman filter is the ability to only process *linear, discrete-time dynamical systems*. Complex dynamic trajectories cannot be modeled by linear systems, thus, constant velocity is assumed in the implementation of the Kalman filter.

If the model is a linear motion model, and process and measurement noise are Gaussian-like, then the Kalman filter represents the optimal solution for the tracking problem. These conditions are satisfied for a vast majority of applications [14].

An example of the advantageous of the Kalman filter is when a vehicle is occluded.

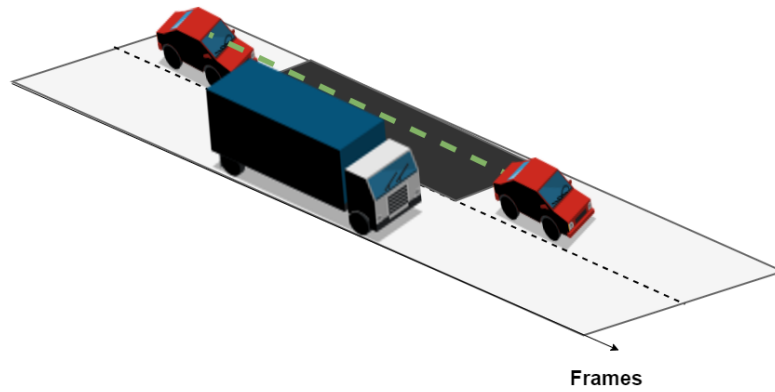


Figure 2.4: Vehicle tracking with occlusion. The dark spot is the occluded area

The vehicle is occluded in figure 2.4, in the dark area. The green line is the Kalman estimated position. The estimate is the weighted average of the predicted state and the measurement. When the necessary measurements are not available, the estimation will fully depend on the prediction of the vehicle motion model.

The vehicle position estimate is obtained by three parameters; *Object motion model*, *Measurements noise* and *Process noise*. These parameters are decisive in the practical application of the Kalman filter, which can be explained in three steps;

- Initial state
- Predicting
- Correction

where *initial state* is the parameters of the filter before it is initiated.

Predicting and *correction* are filtering the measurements, where *predicting* might be seen as a time update and *correction* seen as a measurement update, explained in figure 2.5:

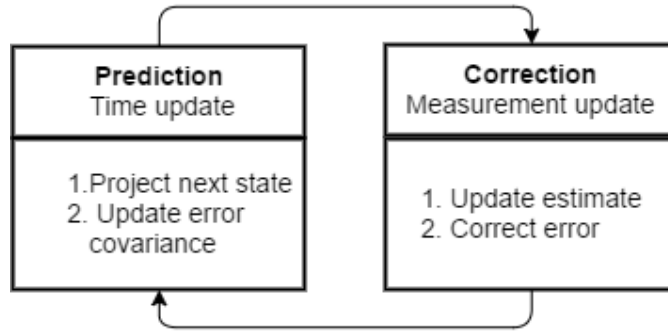


Figure 2.5: Prediction and correction steps in the Kalman filter

The steps are explained in detail below.

Initial state The *object motion model* is considered a *constant velocity model*, represented by

$$x_k = x_{k-1} + v_{k-1} * t \quad (2.15)$$

where x_k is x at step k , and v is velocity. How these parameters affects the estimate is described The state of a constant velocity model includes both the position and velocity in x and y directions. The state vector is presented as:

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} \quad (2.16)$$

where the x, y are position coordinates. \dot{x}, \dot{y} are the velocity in the x and y direction respectively. \dot{x}, \dot{y} are derivatives of the position.

The initial uncertainty is expressed by the covariance matrix P . The uncertainty in position is illustrated in figure 2.6.

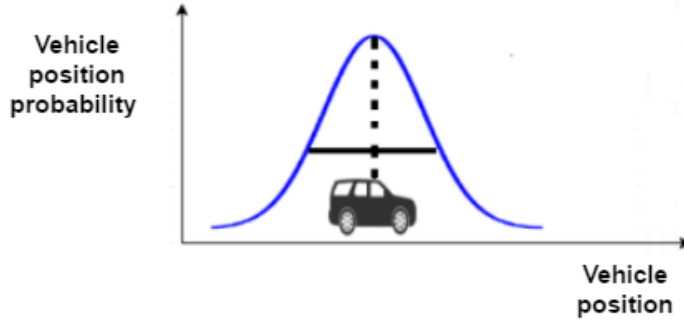


Figure 2.6: Uncertainty of the vehicle position in subsequent frames is given by the blue probability density function

The initial covariance matrix is assumed to have uncorrelated components, and uncertainty I for each component.

$$\mathbf{P} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \quad (2.17)$$

Predicting the next state includes evaluating both position and velocity. The optimal estimate of the vehicles positions is calculated by combining the measurement and the prediction of the prior vehicle position. Predicting the next state:

$$\bar{\mathbf{x}}(k) = \mathbf{\Phi}\hat{\mathbf{x}}(k-1) + \mathbf{\Gamma}\mathbf{u}(k-1) \quad (2.18)$$

where $\bar{\mathbf{x}}$ is apriori state, and $\hat{\mathbf{x}}$ is aposterior state. $\mathbf{\Phi}$ is the state transition matrix. The state transition matrix is represented by the linear dynamical system, constant velocity model, given by equation 2.19:

$$p(t) = p(t-1) + v * p(t-1) \quad (2.19)$$

where \mathbf{v} denotes velocity and \mathbf{p} position. The model is represented by the state transition matrix Φ .

$$\Phi = \begin{bmatrix} I & 0 & \Delta t & 0 \\ 0 & I & 0 & \Delta t \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \quad (2.20)$$

where Δt is given by the time step. The covariance, P , matrix is predicted with equation 2.21:

$$\bar{P}(k) = \Phi \hat{P}(k-1) \Phi^T + Q \quad (2.21)$$

where Q is the process noise, and k is the k 'th frame. The prediction is used to localize and detect the vehicle in subsequent frames. If the vehicle is occluded, the algorithm will predict position purely based on the previous prediction. As the vehicle position becomes "more" uncertain, the covariance matrix would get larger. In addition an increasing acceleration would result in a larger covariance matrix because the model assumes constant velocity (implies zero acceleration).

Correction step is performed when new measurements are observed. After the *predicted* state, the Kalman filter is *correcting* the error covariance, based on the input measurements. The measurement(observation) model is 2.22:

$$\mathbf{z} = \mathbf{D}\mathbf{x} + \mathbf{v} \quad (2.22)$$

\mathbf{v} has zero mean with covariance R , and \mathbf{z} is:

$$\mathbf{z} = \begin{pmatrix} x \\ y \end{pmatrix} \quad (2.23)$$

The x and y coordinates are acquired from the center of the foreground blob.

Furthermore, the model selection matrix, \mathbf{D} , is:

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (2.24)$$

The kalman gain, K , is computed to correct the prediction, 2.25:

$$K = \frac{\bar{P}(k)D^T}{D\bar{P}(k)D^T + R} \quad (2.25)$$

The gain is a relation between the filter's use of predicted state estimate and measurement.

In this application the measurement noise, R , is small, which entails predictions are weighted less than measurements, and the Kalman gain decreases. The weighted prediction is illustrated below:

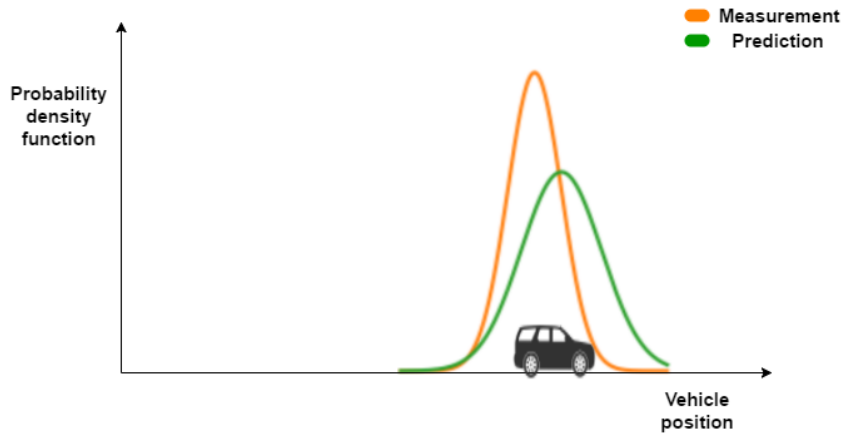


Figure 2.7: Uncertainty of the measurement and predicted position.

The variance of prediction is bigger than the measurement

The green Gaussian probability density function represents prediction, and shows a bigger variance than the orange (measurements), because the process noise, Q , is higher than the measurements noise, R . Analyze possible sources of disturbances and assume them to be white Gaussian.

The state prediction is corrected:

$$\hat{x}(k) = \bar{x}(k) + K(k)[y(k) - D\hat{x}(k)] \quad (2.26)$$

and the covariance matrix by:

$$\hat{P} = (I - K(k)D)\hat{P}(k) \quad (2.27)$$

2.3 Artificial neural networks

Artificial neural networks (ANN) are a method dating back to 1940 (4.40), but in recent times have gained renewed attention in conjunction with increasing data availability and computing power. Neural networks refers to a way of approximating mathematical functions inspired by the biology of the brain, and hence the name neural. The method is used in different applications, including classification. Classifying is the problem of identifying which category a given input belongs to. In newer times when data is to a greater extent stored digitally and IoT (Internet of Things) is introduced, availability of data is greater than before. In addition, road cameras are installed in a greater extent, which in turn increases traffic monitoring capabilities.

Several major companies have already taken advantage of this, including Tesla. Tesla has, in collaboration with Nvidia, based core technology on Neural Networks (NVIDIA). Neural networks are built to solve problems in the same way as the human brain, with several layers of neurons and synapses that forms a network. The number of input and output neurons in the network is determined by the number of input parameters and size of the desired output.

Figure 2.11 is an network structure with two inputs, three artificial neurons and two outputs.

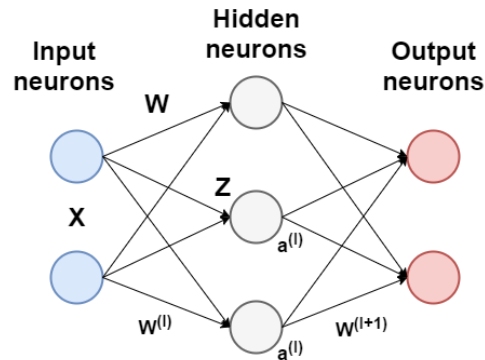


Figure 2.8: An artificial neural network with two inputs, one hidden layer and two outputs

This network can take two-value inputs, and classify into two different classes. Given some input vector, the neural net is trained to compute a desired output by adjusting its weights. $W^{(l)}$ and $W^{(l+1)}$ are the weights, respectively, in the grid from the entry neurons to the hidden neurons, and the grid from hidden neurons to the output neuron. The activation within a single neuron is illustrated below.

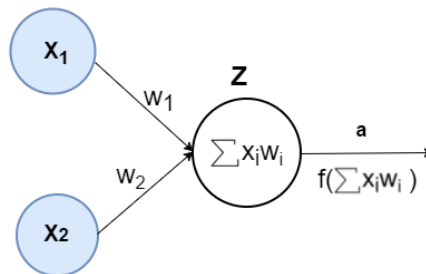


Figure 2.9: Activation's inside a single neuron in the neural network.

where x_1 and x_2 are the input neurons, denoted by \mathbf{X} \mathbf{w} , are the weights, Z is weighted sum of the inputs signals, and a is the output activation of the neuron.

The weights, W , in the network are initialized with random values. The weighted inputs, $Z^{(l)}$ are added together at each node. The activation function is applied to the sum of the weighted input signals, and provides the activity, $a^{(l)}$, of the hidden layer. The activation function is necessary to obtain a non-linear model. There are several activation functions to choose from, where a common one is the sigmoid function, given by:

$$S(t) = \frac{1}{1 + e^{-t}} \quad (2.28)$$

and plotted, with its derivative, in 2.10:

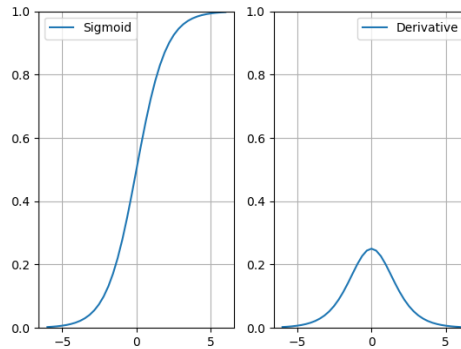


Figure 2.10: The sigmoid function and its derivative

The sigmoid activation function limits the output of a given neuron to a value between 0 and 1. The output, $a^{(l)}$, is multiplied by the corresponding weights. The output will at first give a poor prediction in relation to the expected response, because the weights in the network are initialized with random values. In order for the neural network to improve the classification results, the weights in the network must be updated. This is referred to as training the network.

Training a neural net is done by back propagating its weights. The weights are updated by minimizing a cost function with respect to the weights in the network. The cost function is computed by comparing the predicted value to the desired output. There are several cost functions, where mean squared error is the most frequently used one:

$$J = \sum \frac{1}{2} \times (y - \hat{y})^2 \quad (2.29)$$

y is the target vector or desired output for the input x and \hat{y} is the predicted value.

The goal of backpropagation is to compute the partial derivative, or gradient, $\frac{\partial E}{\partial w}$ of a loss function J with respect to any weight w in the network. This is called *stochastic gradient descent*ⁱ.

Partial gradients, of the loss function with respect to the weights, are used to update the weights and minimize the cost. The weights are updated with an optimizing algorithm called gradient descent.

Gradient descent can be explained as a linear approximation to the cost function, J , and then moving downwards toward the weights, W , that gives the lowest cost, where the hidden layer gradient matrix for the weights in layer l , is given by the matrix:

$$\frac{\partial J}{\partial W^{(l)}} = \begin{bmatrix} \frac{\partial J}{\partial W_{11}^{(l)}} & \cdots & \frac{\partial J}{\partial W_{1n}^{(l)}} \\ \cdots & \ddots & \vdots \\ \frac{\partial J}{\partial W_{n1}^{(l)}} & \cdots & \frac{\partial J}{\partial W_{nn}^{(l)}} \end{bmatrix} \quad (2.30)$$

The sum of the cost function adds the error from each example which creates an overall cost:

$$\frac{\partial J}{\partial W^{(l)}} = \sum (y - \hat{y}) \quad (2.31)$$

ⁱThe gradient descent can also be computed using the whole dataset. This is called batch gradient descent. The batch approach is great for convex, or relatively smooth error manifolds. Additionally, batch gradient descent, given an annealed learning rate, will eventually find the minimum located in it's basin of attraction. Small batches of the dataset may also be used, this is called mini-batch gradient descent

Where \hat{y} is the sigmoid activation function of $Z^{(l)}$, $f(Z^{(l)})$.

To find the gradients with respect to the weights in all layers, the backpropagation algorithm is used to compute the overall cost of function J . This is done by applying the chain rule to 2.31:

$$\frac{\partial J}{\partial W^{(l)}} = -(y - \hat{y}) \frac{\partial \hat{y}}{\partial Z^{(l+1)}} \frac{\partial Z^{(l+1)}}{\partial W^{(l)}} \quad (2.32)$$

The back propagation error with respect to the weights decides where the cost function should move:

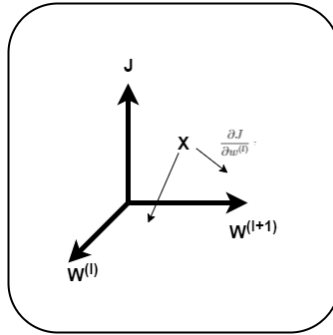


Figure 2.11: Back propagation error of J , with respect to W .

Figure 2.11 illustrates the error, which is moving towards the weights that contributes more to the overall cost, which means that synapses with large error will gain more correction in the next training epoch.

The back propagation starts from the last hidden layer in the network, where $\frac{\partial Z^{(l+1)}}{\partial W^{(l)}}$ is the change of Z , last layer activity, in respect to the weights in the second last layer. $\frac{dZ}{dW}$ is the activation for each synapse. The error-terms is back-propagated to each synapse, by multiplying by each weight. The weights that contributes more to the overall error will have larger activations, yield larger to the next backpropagation layer and yield larger $\frac{dZ}{dW}$ values.

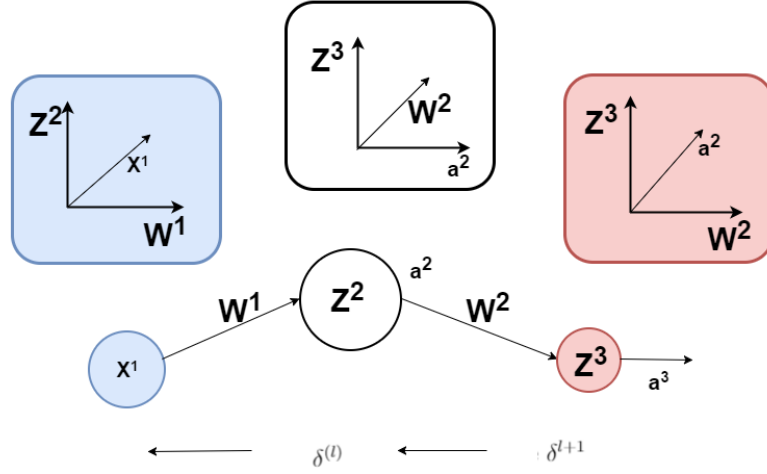


Figure 2.12: Backpropagation error from output to input neuron, with one hidden layer

Figure 2.12 is the backpropagation for one hidden neuron, with one hidden layer. δ^{l+1} is the backpropagation error from Z^3 to Z^2 , and δ^l is the back propagation error from Z^2 to x^1 . The error back propagated from Z^2 and backwards until the input of the network, x^1 . The backpropagation error from Z^3 is the derivative with respect to the weights. The backpropagation error for hidden layers, Z^2 , is computed as derivatives across the synapse, a . The back propagation error is computed, and multiplied by the activity in the hidden layer:

$$\frac{\partial J}{\partial W^{(l)}} = -(y - \hat{y}) f'(Z^{(l+1)}) \frac{\partial Z^{(l+1)}}{\partial W^{(l)}} \quad (2.33)$$

$y - \hat{y}$ is the true label, $f'(Z^{(l+1)})$ is the activation function. The equation 2.33 may be expressed as:

$$\frac{\partial J}{\partial W^{(l)}} = \delta_j^{(l+1)} a_{i,j}^{(l)} \quad (2.34)$$

where δ^{l+1} is the back propagation error, and $a_{i,j}^{(l)}$ the activations.

If the network is built with multiple layers, the back propagation is done with the derivatives across the synapses instead of the derivative in respect to the weights. \mathbf{X} is a vector holding the n numbers of input neurons in the network:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{1+n} \end{bmatrix} \quad (2.35)$$

thus, the cost function in respect to the weights:

$$\frac{\partial J}{\partial^{(l)}} = \mathbf{X} \delta^{(l+1)} \quad (2.36)$$

where \mathbf{X} is the input vector to the neural network, and $\delta^{(l)} = \delta^{(l+1)}(W^{(l)} f'(z^{(l+1)}))$, where f' is the activation function of the sum of weighted inputs, $z^{(l)}$, in each neuron.

Overfitting is a problem of machine learning algorithms, where the algorithm does not reflect the real world [7]. The algorithm is built on observations of the real world, and these observations are composed of signal and noise. The model should capture the underlying process or features of the input, but the signal will always be obscured by noise, therefore the algorithm must be convinced to fit the signal and not the noise to prevent overfitting.

This problem comes apparent if the neural network is too deep, trained with too many iterations, or has a small dataset. A rule of thumb is that it's required ten times more data than degrees of freedom in a model. Each weight is one degree of freedom [6].

Consider a two-class problem, illustrated in Figure 2.13, classifying red and blue dots. The dots are placed randomly. The x- and y coordinates for the red and blue dots represent input data to the neural network.

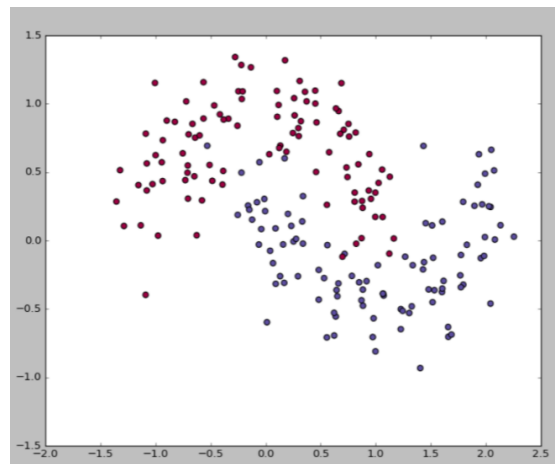


Figure 2.13: A two class problem presented with blue and red dots, randomly placed in a predefined area. The axes represent the position.

In the two class problem, the coordinates are fed as pairs of x,y coordinates, therefore two nodes are being used. One node for each x- and y-coordinate. The output is classifying between red or blue, thus, two output neurons.

Figure 2.14 and 2.15 shows the decision boundary after the neural network has been trained.

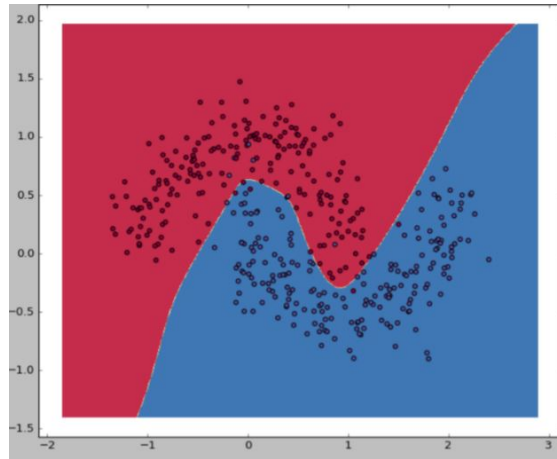


Figure 2.14: Illustration of a properly trained neural network output with 10 hidden neurons after 100 training iterations.

Plot 2.14 is a good fit to the dataset, while plot 2.15 is overfit.

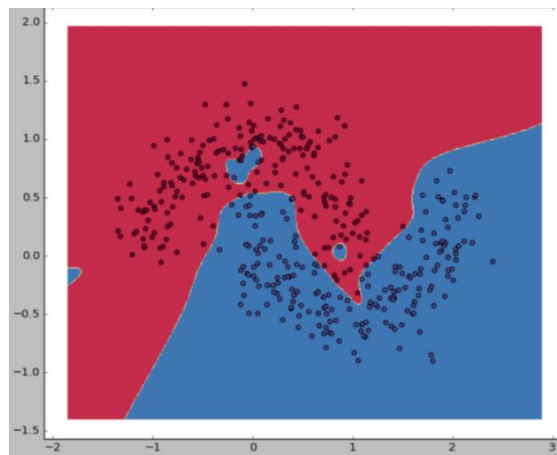


Figure 2.15: Illustration of a overfitted neural network output with 100 hidden neurons after 1000 training iterations

Figure 2.15 is overfit because it does not find the general decision boundary, but locates small patterns that may be considered noise in the dataset. The overfit

model will achieve a best classification score for that particular dataset or case, because it correctly classifies all the data, but will fail when new data is presented to the classifier.

To overcome overfitting, the data is normally split into three sets:

- Test set
- Training set
- Validation set

It is a indication of overfitting if the accuracy of the training dataset scores higher than the accuracy of the test or valididation dataset. The accuracy is calculated as the overall correct classifications.

$$\text{Accuracy} = \frac{\text{Correct classified}}{\text{Total number of samples}} \quad (2.37)$$

With less training data, the neural network parameter estimates have greater variance. With less testing data the performance statistics will have greater variance. The training set is used to update the weights in the network, while the cross-validation set used to measure the accuracy during training, but not used to update the weights. The validation set is not used to update the weights. The test set is used to validate the network after the training is completed.

2.4 Classifying images with neural networks

Conventional artificial neural networks as described in last section does not scale well to large images, because they process the image as a flattened vector. Each of the hidden layer has a set of neurons, and each neuron are fully connected with the neurons in the previous layers. These layers are called dense layers, or fully connected layers. Each layer is fully connected to its previous layer, but neurons in a single layer function completely independently and do not share any connections. E.g an $128 \times 128 \times 3$ image would give 49,152 weights in the first layer, and the weights will add up as the network is getting deeper. The amount of parameters to update and tune the network would lead to overfitting and need of heavy computational power.

Convolutional neural networks takes advantages of the vector input being an image, and limits the number of neurons in the network, without losing information about the feature vector. The neural network is build in three layers: height, width and depth, where depth is referring to the activation volume [27]. The images are input activation volume, with dimension height,width and color channel. Only a part of the image is connected to the previous layer, instead of having a fully connected network. Only the last layers in a convolutional neural network are fully connected. The convolutional neural net architecture is build from three main types of layers: Convolutional layer, pooling layer and fully-connected layer.

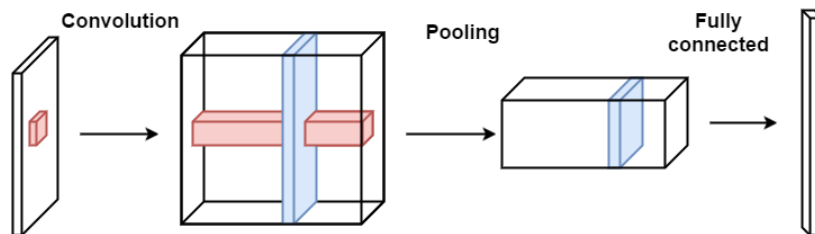


Figure 2.16: Convolutional network structure, with convolution layer, pooling layer and fully connected layers.

Convolutional layers consist of learnable filters. During training, these filters are convolved across the input vector. For each pixel position the dot product summation between the filters and values around the center pixels are computed. The filter is convolved across the entire image, which allows the neural network to respond to visual features such as edges. Each convolutional layer will have a set of filters, which is stacked in the depth dimension of the network. The depth is presented as the red box in figure 2.16. An example of a typical filter that reacts on edges is the Laplacian, with spatial size 3x3

$$F = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (2.38)$$

Figure 2.17 is the result of convolving a Laplacian filter around a grayscale image.

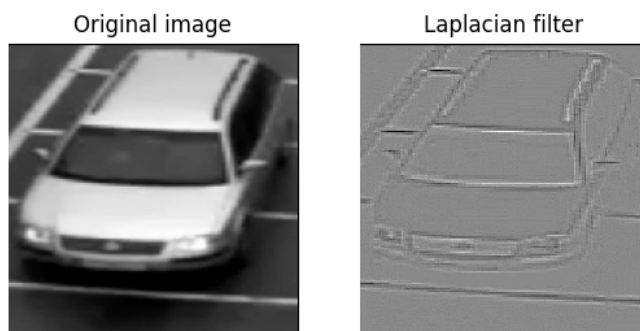


Figure 2.17: Edge detection with Laplacian filtering

In contrast to dense layers, each neuron are connected to a local region of the image, illustrated in 2.18.

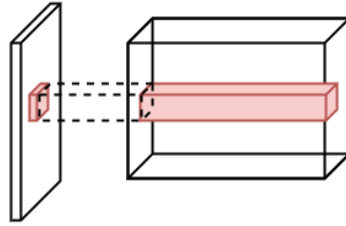


Figure 2.18: All neurons along the depth are looking at the same region in the input.

Each neuron in a depth slice are using the same weights, such that each slice can be computed as a convolution of the feature matrix. This allows the network to localize the features in an image.

The input matrix are divided into small tiles, based on the filter size. Each of the input tiles are processed in the neural net, and the output size of the convolutional layer is given by the equation

$$O = \frac{W - F + 2P}{S} + 1 \quad (2.39)$$

where W is the input volume, F is the receptive field (filter size), S is the stride and P is the zero padding.

The zero padding are used to ensure that the input and output has the same spatial volume.

Pooling layers are often inserted in-between successive convolutional layers. Pooling, or subsampling is a technique to reduce the size of the feature matrix, which leads to less memory use and faster training. The most common one is the max pool. Maxpool applies a filter, normally of size 2x2, to the input volume, and outputs the maximum value in the every region that the filter convolves around.

The pooling layer samples all the highest activation values, reduces the spatial size by a half, and keeps the relative locations between the features.

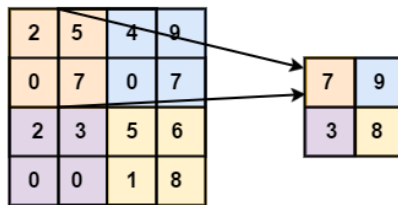


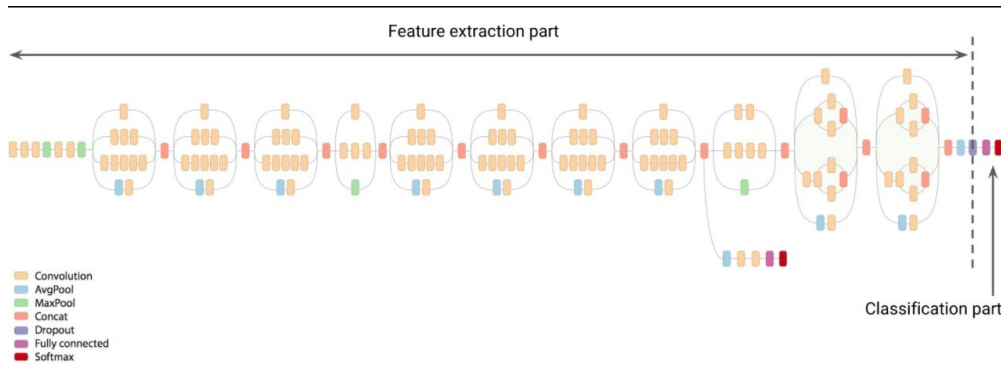
Figure 2.19: Downsampling an image with max-pooling with a 2x2 filter

Figure 2.19 demonstrates the pooling on an 4x4 image, which is down-sampled to 2x2. The filter size is 2x2, and the stride is 2. The filter convolves from the upper left corner, illustrated by orange. The highest activation from the box is chosen, and the filter is subsequently moved to the right, with a stride of two boxes, illustrated with blue, and keeps the highest activation for the region. The process is repeated for the whole image.

Dropout are layers composed to process the overfitting. The layer drops out a random set of activations in hidden layers, by setting them to zero in the forward pass. The dropout forces the network to be redundant, because the activations are removed randomly. In practice, the drop-out is equal to training the data on many different networks, and the result becomes a more robust network.

Fully connected or dense layers is as explained in 4.2. They are the last layers of convolutional networks, and outputs the classification score. The volume from a convolutional layer is flattened into a vector and passed into fully connected layers. Fully connected layers constricts the classification of an image to a single variable for each class(classification score),which is unattainable for convolutional layers because they output a volume.

Transfer Learning Transfer learning is the process of training an already pre-trained model. It transfers the weights and parameters from a network that has been trained on a large volume of images, and continue the training on a custom dataset. The last layers of the pre-trained net are removed, and retrain the last layers on a different dataset. Transfer learning is illustrated in 2.20:



[13]

Figure 2.20: Transfer learning from the Inception net

The layers from the pre-trained net are not updated, and is not affected by the gradient descent. A common model to transfer from is the ImageNet. This is a dataset with 14 million images, classified in 1000 classes [21]. The first layers are discovering edges and curves, which is often needed in all classification task. With the exception of datasets that differ significantly from the classes in the ImageNet, the network will benefit from transfer learning.

Chapter 3

Implementation of the vision-based traffic system

In this chapter, details of the practical implementation of the vision-based traffic system will be presented. The system is divided into four modules:

- Preprocessing
- Detection
- Tracking
- Classification

The system is distributed so that all *training* of the neural network is performed on the Unix server because of the necessity of computational power. The real-time part of the system is run on a local computer.

The framework for the system in terms of software and hardware is presented first. Figure 3.1 shows a simplified overview of the software- and hardware components of the system.

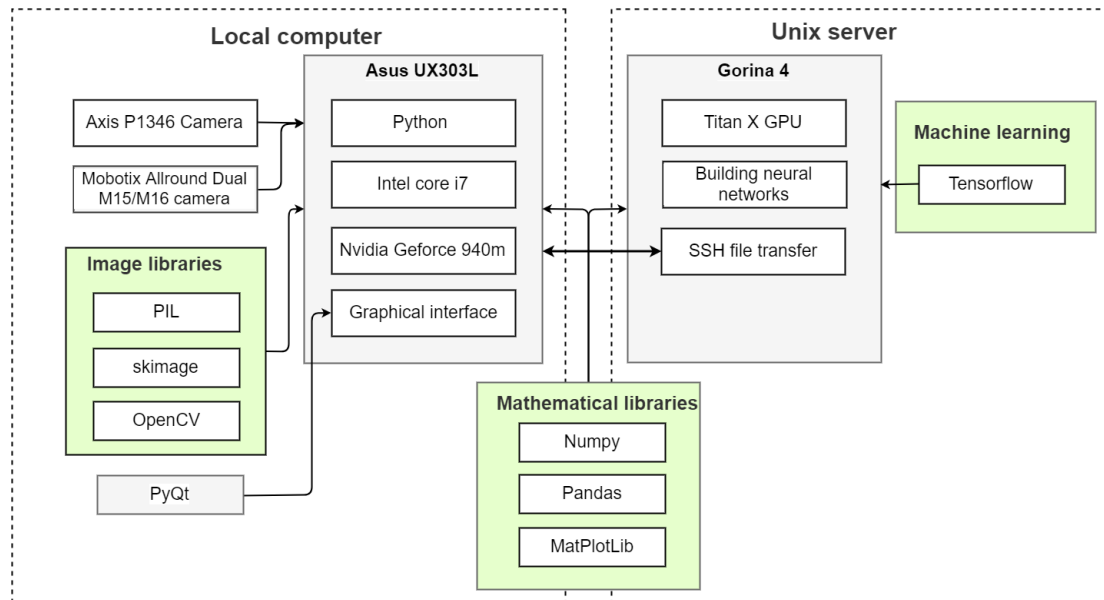


Figure 3.1: Software and hardware components of the system

The system relies on parameters from the location it processes, which makes it more practical to explain parts of the implementation in details in the experimental section. Comprehensive information about the Python libraries are attached in A.

3.1 Hardware components

This section will present various hardware components that has been implemented in the system.

	Camera	System	Machine learning
Hardware	AXIS P1346	Asus UX303L	Unix server
	Mobotix Allround Dual M15/M16	Nvidia 940 M	Tesla P100

Table 3.1: Hardware components table

The recordings from the web cameras are, according to Statens Vegvesen[46], a set of Axis P12346 cameras.



[5]

Figure 3.2: Cameras used by Statens Vegvesen, Axis P1346

The webcams are set to give users an impression of traffic conditions such as congestion, weather and driving conditions. The cameras should not take pictures of individuals, and it should not be possible to identify persons or registration numbers of vehicles on the images, due to privacy regulations. The drawback of the anonymous filming is a limited image resolution and low frame frequency. The images are free for use and Statens vegvesen does not demand any allowance for the usage [46].

Another camera has also been used for testing. This camera is provided by Bjørn Fossåen from Statens vegvesen, and is a combined optical and thermal camera.



Figure 3.3: Mobotix Allround Dual M15/M16

It is a combined day/night camera for 24-hour use, used for surveillance. More comprehensive technical information can be found in B.



[31]

Figure 3.4: Tesla P100 video card

The Unix server is a server that is available to students at the University of Stavanger. The server is a Linux based web server, where heavier applications can be run. The server has installed three Tesla P100 video cards (Figure 3.4). The

server is intended for machine learning purposes such as training neural network. These were purchased in the context of master's thesis work based on machine learning. Comprehensive information about the Tesla P100 may be found in [31].

3.2 Implementation in Python

This chapter will present programming modules that has been developed and implemented in the final system. The Python-system is divided into four modules:

- System setup
- Detection module
- Tracking module
- Classify module
- Analyze and interface

The program is object-oriented, where an object stores vehicle attributes in fields to ensure systematic structure of all the passing vehicles. The objects can be modified and maintained independently of other objects, and once created, the object can be easily modified inside the system. From here on, **objects** refers to an data-object.

Implementation of the *system setup* describes necessary readjustments when the program is utilized at new locations.

The *detection module* involves the background subtraction and blob detection. This module describes both implementation of background subtraction and the image processing. The *classify module* describes how neural networks are implemented into the system.

Figure 3.5 shows the program flow, where the *system setup*, *detection module* and *classify module* are located in the left box, and *Analyze and interface* is at the right:

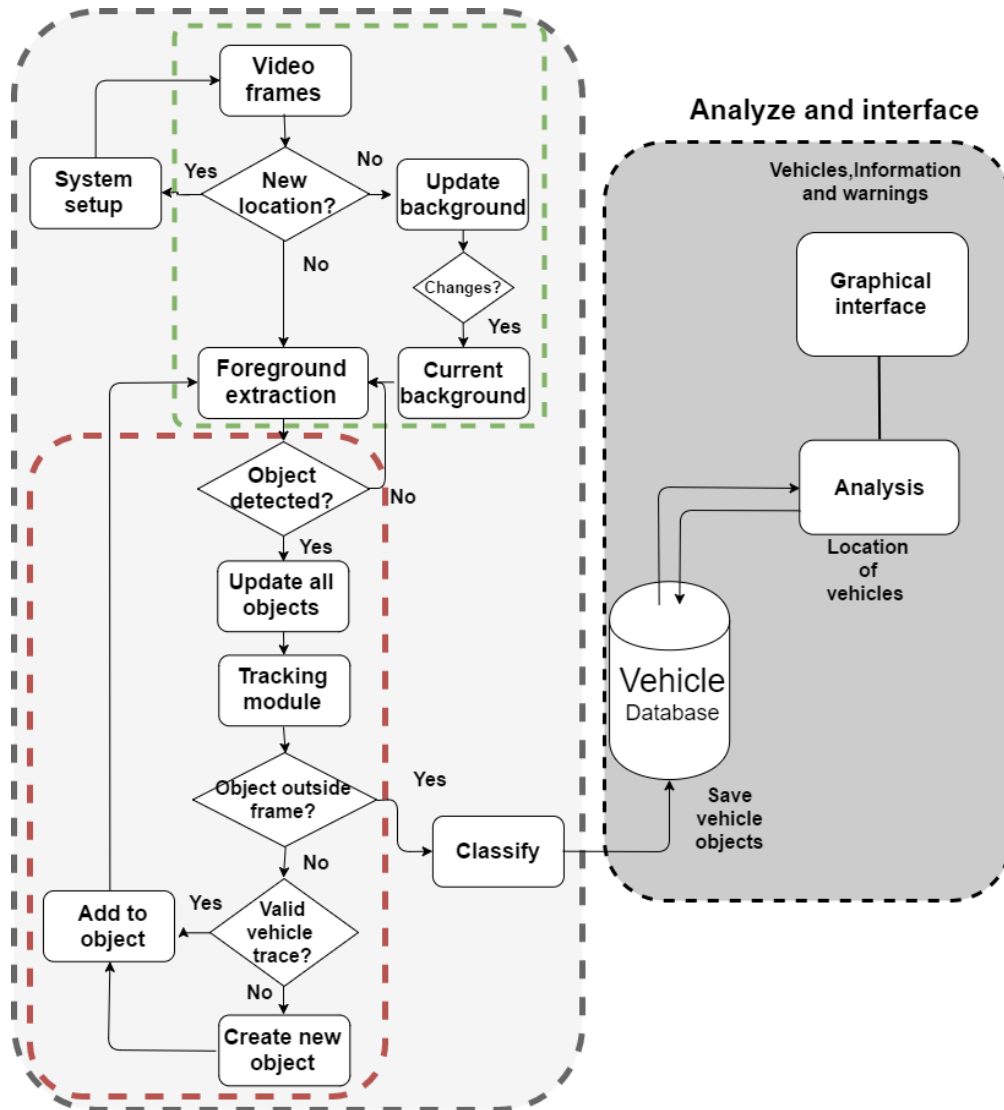


Figure 3.5: The flow chart of the program. The left module is acquiring traffic data, while the right module is processing the data into useful information.

The *detection module* is represented with the green area, the *tracking module* by red and the *analyze and interface* by the right box.

The following sections are describing the implementation of each module.

3.2.1 System setup

The *system setup* allows the user to make adjustments to the system to accommodate geometric changes to changing locations. The reasoning for implementing a *system setup* part into the system is two-fold:

- Adjust the system to a new locations
- Eliminate false positives

The steps are specified in figure 3.6:

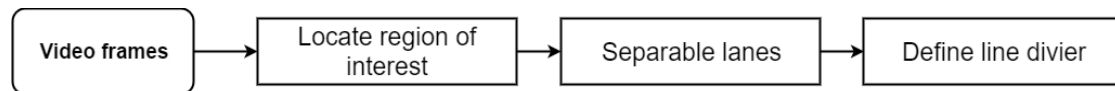


Figure 3.6: Methods used in *system setup*

The *region of interest* is defined by the corners of the road, such that all traffic is captured, but irrelevant information and noise is filtered out.



Figure 3.7: Region of interest defined by the corner the lanes

The divider line is placed to cover the entire road. Objects that pass the divider is counted and saved to the database.



Figure 3.8: Divider line, in red, determines where the vehicles are counted

3.2.2 Detection module

The *detection module* processing the incoming video frames to extract movement in the background and determine the position of the vehicles. The extracted area is then passed forward for further processing. The flow diagram of the detection:

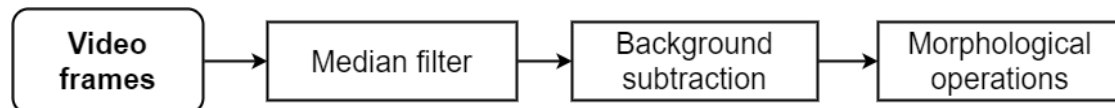


Figure 3.9: flow in the the detection module

The raw video stream is processed with a median filter to reduce noise points of the image ⁱ. The image becomes smoother and the filtering has little effect on the edges of the vehicle and other details [23]. The image processing is followed by a mixture of Gaussian background subtraction(BS) algorithm, provided by opencv. For the BS to adapt faster to the background, an initial background image is chosen. The initial background is set to a frame with no moving objects in the background. The background is updated based on the history of previous frames.

ⁱMedian filtering should be a well known method for the audience of this thesis. Information may found in [15]

In this application, the method is set to form a background based on the 50 last frames. The raw output of the background subtraction is shown in figure 3.10:



Figure 3.10: Unprocessed foreground mask

The foreground mask is then cleaned up with morphological operations, provided by opencv. A structuring element of size 3x3 pixels is used to perform closing, opening and dilation in that respective order.

The result of performing the morphological operations is the removal of noise and filling out details on the remaining objects while retaining the size of the detected objects. The result is shown in figure 3.11.

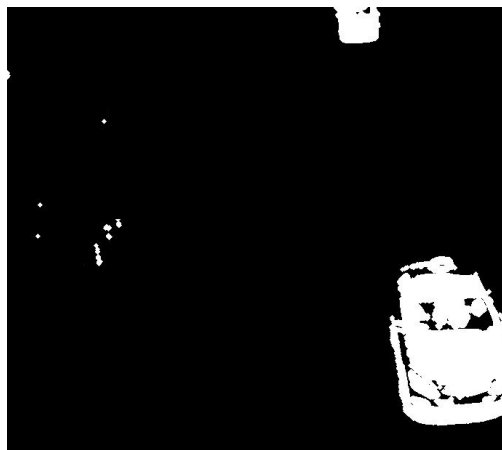


Figure 3.11: Foreground mask processed by morphological operations

Shadows caused by vehicles are removed by thresholding a value relative to the vehicle. The idea behind shadow extraction is that shadow has a slightly darker color than the road, while vehicles are clearly discernible. This approach fails when the illumination is weak, and there is no clear contrast between vehicle and shadow.

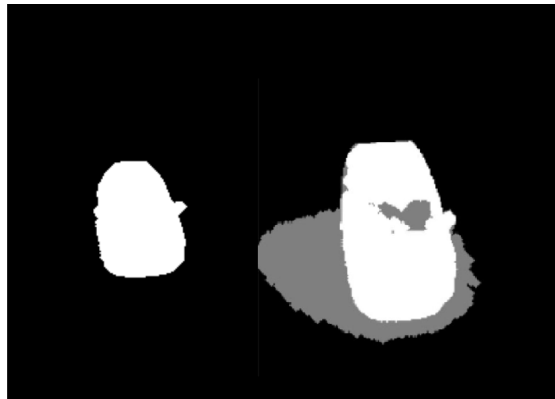


Figure 3.12: Background subtraction without(left) and with shadow(right)

As an object is detected in the background, the center coordinates of the contour is computed and plotted for each consecutive frame. The tracked center of the vehicle is found by dividing the detection box width and height by 2.

Detected objects are marked in relation to its extracted background blob.

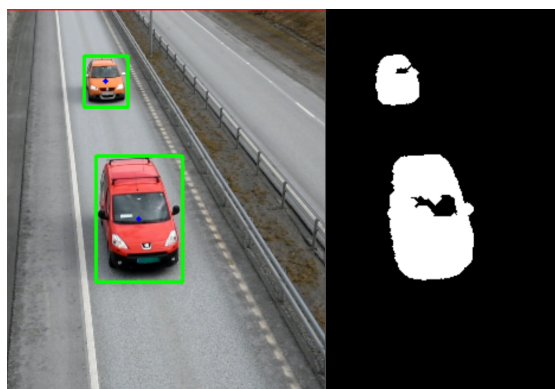


Figure 3.13: Detected blobs are marked with a green square

The extracted region is defined within the lane markers. The lanes are localized by evaluating the blob coordinates over a period of time.



Figure 3.14: The lanes becomes apparent by tracking the movement in the foreground mask

Figure 3.14 displays object positions over time. The three lanes are easily recognized, and manually divided into separate regions. Traffic in opposite direction is removed from the processed region. Each blob are saved as a separate object, with information about size, coordinates and time at last detection. When the coordinates indicates that the object has passed a divider, an image of the vehicle is saved and stored in its respective object and saved to the vehicle database.

3.2.3 Tracking module

Tracking is the process of matching vehicles in subsequent frames. The tracking provides the system with abilities to prevent true negatives values and measure velocity, size and driving pattern.

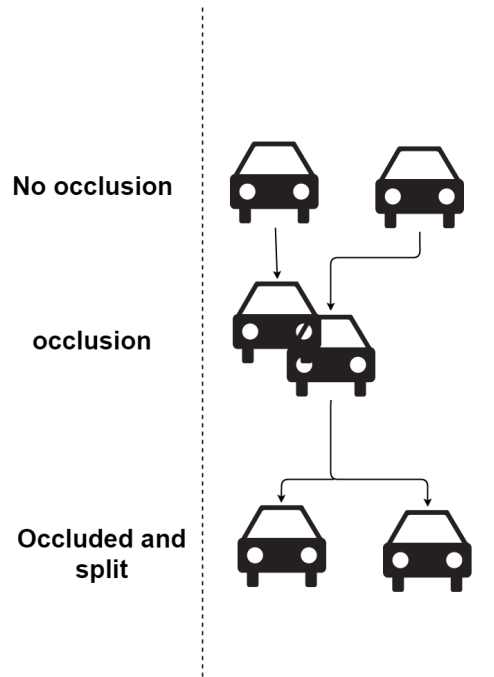


Figure 3.15: Detection and tracking scenarios

The tracking has basically three scenarios per lane, listed in relation to the degree of difficulty:

One lane, ideal case: When there is no occlusion or separation. This is the simplest tracking scenario, where a detected object is assigned a new tracker. This tracker is deleted when the vehicle is leaving the frame.

One lane, object is split: When occluded objects are split, they share the corresponding occluded tracking values until they are split, and assigned separate

values afterwards.

One lane, with occlusion: The blobs are overlapping, and one object contains two or more vehicles. They share the same coordinates, and are counted as one. When objects passes the divider, they are counted and classified.

While taking cognizance of the above, a two part system is proposed to obtain robust tracking:

1. Manually deciding a vector space
2. Predict position with Kalman filter

Figure 3.16 shows the tracking system. The first state, *Acquire object*, is acquired from locating a vector space.

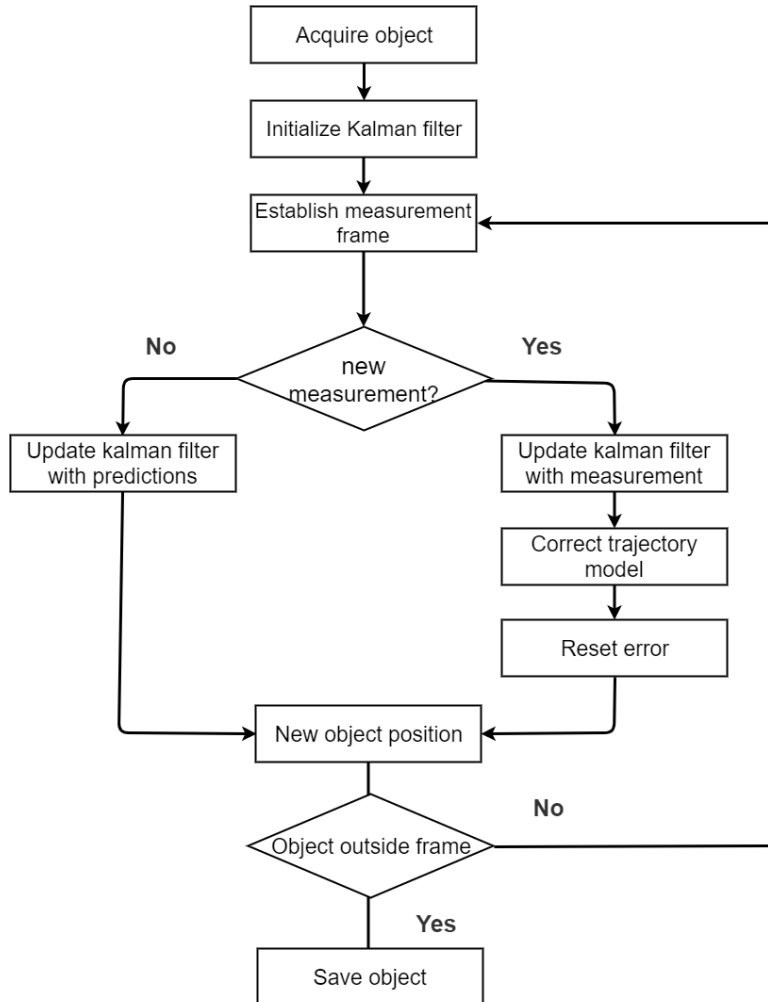


Figure 3.16: Flow chart for the Kalman filter

A region-based tracking method are tracking the regions that are segmented from the foreground extraction. The vehicle assessment involves features as geometry and number of measured positions. The geometric traits are used to eliminate false segmented areas. The module is assigning coordinates to objects, and retained in later frames to ensures that objects are counted.

The first step when the algorithm is applied to a new scene is computing a vector

space, established from the coordinates. The vector space ensures that an moving vehicle-object acquires at least two points, which is required for the Kalman filter to predict the next position. This process is described more in comprehensive details in the experiment section 4.4.1.

The Kalman filter implemented in the system is based upon the constant velocity model, as described in 2.2. It is used to predict the next spatial and temporal state of the vehicle. All measurements for an object up to the current time are used to estimate the next position. A minimum of two positional coordinates are required to describe the dynamical behavior of the system, and to predict its future state.

Initially the noise covariances matrices for the measurement noise and the process noise must be obtained. The covariance of the measurement noise is denoted as R , and assumed to be Gaussian. In the context of this application, this means the detection error. The R matrix describes how uncertain the position around the location of the centroid of the bounding box is. In this case for the x,y coordinates the corresponding diagonal values of R should be a few pixels, assuming that the measurements are relatively reliable. The state includes velocity, thus, the need to guess the uncertainty of the velocity measurement, and take the units into account. The position is measured in pixels and the velocity in pixels per frame, so the diagonal entries of R must reflect that.

Q is the covariance of the process noise. The Q specifies how much the actual motion of the object deviates from the assumed motion model. The constant velocity model should be reasonably good when tracking the vehicles, which implies small entries of Q . If the vehicles are driving with constant velocity, the prediction will deviate from the constant velocity model, and yield larger error. In general the Q matrix will be full matrix, not a diagonal, because there is correlation between the state variables. For example, if there is a change in velocity due to bumps, there will also be a change in position. They are correlated, and so the off-diagonal

elements will be non-zero. But even a relatively simple process model can produce acceptable results if there is enough uncertainty in the Q matrix, but selecting an overly large Q , then Kalman filter would not be well-behaved.

The model is tuned by setting the measurement noise matrix is set as constant, and treating the process noise as a tuning parameter to adjust the gain of the Kalman filter. The tuning is done by plotting the predictions to see how much they deviate from the detections. Since the R matrices is considered reliable, the Q is tuned until the predictions and detections is right. The source code to tune the filter is attached in Appendix. The kalman filter is implemented by coding the equations from 2.2 into Python. If a pair of coordinates fails to match any objects currently tracked by the algorithm, the coordinates are assign to a new object. An object is deleted from the *algorithm* if no new coordinates are assigned to the object within a given time frame. The object is kept in the *database* if the coordinates have passed the divider line.

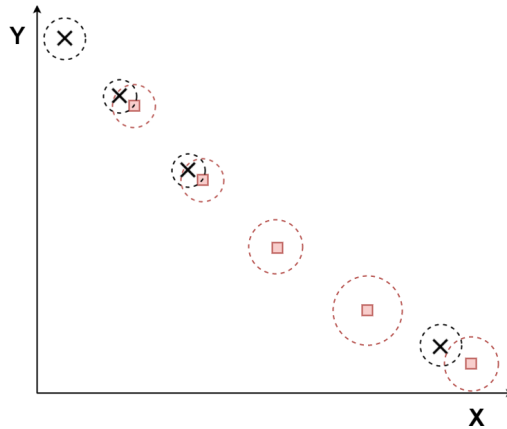


Figure 3.17: Vehicle moving from upper left corner to right bottom corner. Red square is prediction, x is measurements.

Figure 3.17 shows the tracking of a vehicle. The X is measurements, and the squares are predictions. X and Y along the axes are position over time. With no new measurements the uncertainty of the prediction grows, denoted by the circle around the prediction.

Figure 3.18 shows the tracking of a vehicle.

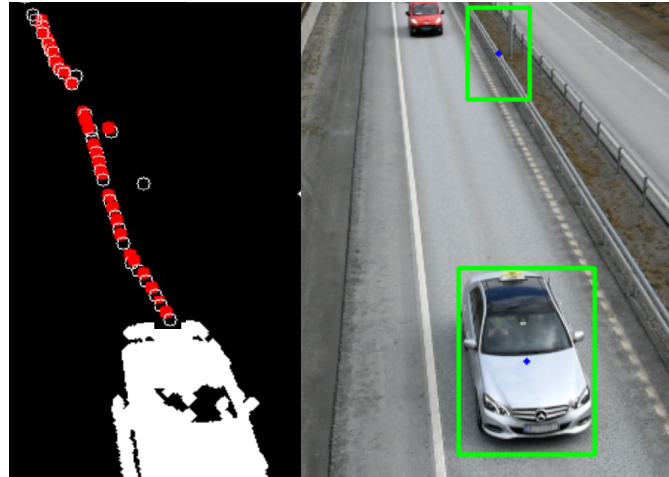


Figure 3.18: Background blob tracked with Kalman filter. The dots highlights trajectory points. Red is measured position, and white is predicted position.

The white circles denotes the kalman predicted position. The red circles are the true measured position. The measurements are weighted more heavily than prediction, but if no measurement is registered, the kalman filter is computed entirely from the last predicted position. Predict the last estimation to the time of the new measurement using the propagation model, and update the co-variance accordingly.

3.2.4 Classification module

A convolutional neural network is trained as a classifier, with the purpose of classifying incoming objects from the detection module. The classifier is built with the Python package Tensorflow. The classification module structure is described in figure 3.19.

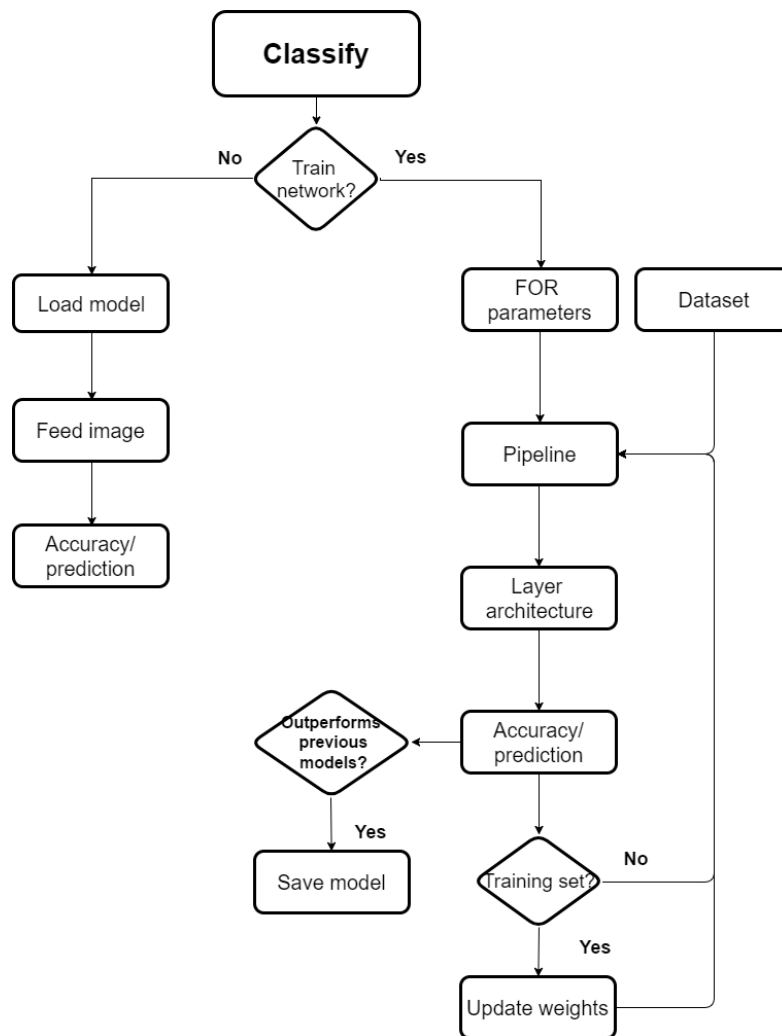


Figure 3.19: Flow chart of the *Classify* module, both with and without training

Training of the classifier is initiated with different hyperparameters that defines the complexity and learning capacity of the model. These parameters cannot be learned directly from the data in the standard model training process and need to be predefined.

By combining different *parameters*, one can programmatically set a competent network architecture, with the number and type of neuronal layers and the number of neurons comprising each layer, and choosing the values that test better.

The hyperparameters for the convolutional neural network are image input volume, the learning rate in the gradient descent, image batch size, convolutional filter size and depth, and fully connected layer size.

An input pipeline were constructed to feed the network with image batches. A script is reading filenames from the dataset folder, and subsequent shuffles the filenames to prevent overfitting. According to the initially defined batch size , a batch of filenames are placed in a queue. An image decoder reads the filenames and pass on the image queue to the network. The pipeline limits the ram usage of the computer by allocating a batch of images to the memory, and not the entire dataset. The dataset structure is describes in next section.

The layer architecture is a sequence of layers, defined by the hyperparameters. Three types of layers are combined, convolutional layers, pooling layers and fully-connected layers, in that order. A number of each layer type may be stacked, keeping in mind that a deeper network is more prone to overfitting. Each convolutional layer is holding a number of trainable weights, according to the spatial input volume and the filter depth.

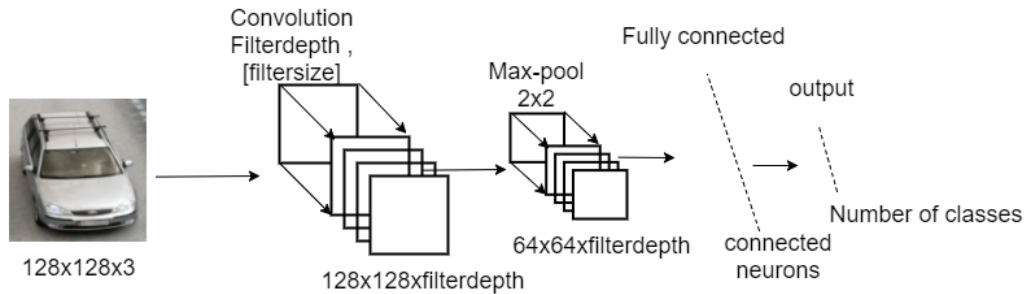


Figure 3.20: Convolutional neural network with fixed hyperparameters, filterdepth, filtersize and connected neurons.

Convolutional layers, fully-connected layers and pooling layers have all hyperparameters that were tested. For each training epoch the model is saved if the accuracy of the validation set is higher than previous recorded.

To use the classifier the weights and the network architecture from the best recorded model is initialized. Loading the model is time consuming, but it is a one time cost. Using the classifier may be done by both the locale computer and the Unix server. The classifier is represented to the left in the flow chart 3.19.

3.2.5 Graphical Interface

A graphical interface was developed to illustrate the activity in the vehicle database. The interface gives feedback about vehicle class, velocity(in pixels per frame), last detection of the vehicle and number of counted vehicles.

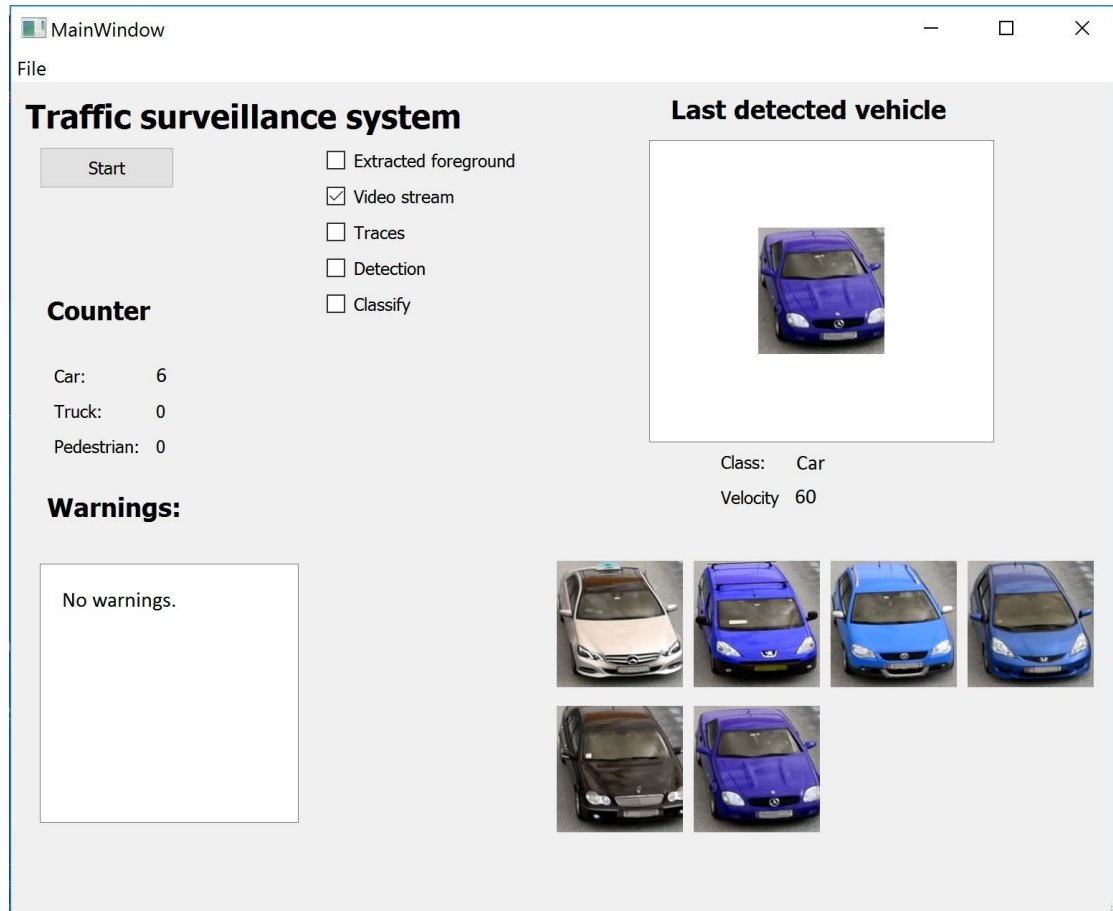


Figure 3.21: Information from the database is shown in a Graphical interface

Start starts the system at a predefined location. The check boxes presents a number of different processing options:

- Extracted foreground
- Show video stream
- Show detected blobs
- Detect and track vehicles
- Classify incoming objects

The *counter* keeps track of the classified vehicles in the database. If the classify check box is unchecked all objects are counted as cars. The neural network model is loaded into the program by pressing *file* and *insert model*.

The upper right image shows the last detected vehicle and its class and velocity. The images below is the last vehicles stored inside the database.

Chapter 4

Experimental results

The following chapter presents the experiments that were carried out and the results they gave. The chapter is divided in four sections:

- Collecting data for the neural network
- Testing architectures for the neural networks
- Initial test of the detection module
- Data analysis of the dataset acquired from the initial tests

All detection experiments were run with a axis P1431 camera and Asus UX303L. All neural network tests were performed on the Unix server on a Tesla P100 GPU. The neural network were trained to classify three different classes; *pedestrian*, *cars* and *trucks*.

4.1 Data acquisition

This section describes the dataset acquisition to meet the large data requirements for deep learning, as discussed in 4.2. The *pedestrian dataset* and *cars and trucks datasets* are described separately.

Pedestrian dataset

The Daimler dataset [18] is made freely available to academic and non-academic entities for non-commercial purposes such as academic research, teaching, scientific publications, or personal experimentation. *The dataset involves a large training and test set. The training set contains 15,560 pedestrian samples (image cut-outs at 48x96 resolution) and 6744 additional full images not containing pedestrians for extracting negative samples. The test set contains an independent sequence with more than 21,790 images with 56,492 pedestrian labels (fully visible or partially occluded), captured from a vehicle during a 27 min drive through urban traffic, at VGA resolution (640x480, uncompressed) [18]*

Unbalanced datasets has a significant impact on the performance of convolutional neural networks [40], thus, to prevent a unbalanced datasets relative to the car and truck, 7000 random images were chosen from the Daimler dataset.

Cars and trucks dataset

In order for the network to recognize vehicle fronts, it is a prerequisite that the data set also consists of vehicle fronts. The data collecting were resolved by processing recording with the detection algorithm. The detection algorithm were utilized at different locations at various times of the day. There are several reasons why the recordings are made under different environments and locations:

- Variance of illumination
- Images of multiple angles of the vehicles
- Changes in background

Performing all data retrieval from a particular location may cause overfitting, as the neural network may find insignificant features such as borderlines and colors of the road.

The collected data were labeled by hand as either *car*, *truck* or *no class*.

The recordings gave a total of 4870 annotated cars and 4993 annotated trucks.

Recording position	Time	Personal vehicle	Truck
Maritim E18	12:00-20:00	2499	1245
Sandvika E16 Nord	12:00-18:00	543	261
Sandvika E18	07:00-11:00	1562	1583
Rv. 150 Ullern	10:00-20:00	266	1905
Sum:	28 hours	4870	4995

Table 4.1: Gathered data at given locations

Data augmentation

To increase the amount of data, various augmentation methods were applied to the dataset. Figure 4.1 shows images from the original dataset, and figure 4.2 shows the augmented dataset.



Figure 4.1: Original data, directly cropped from the object detection module

Different augmentation methods were applied to the dataset, such as cropping, flipping, change in hue, contrast, brightness and saturation. The augmentation simulates different scenarios that occurs during the day, e.g illumination and vehicle orientation.

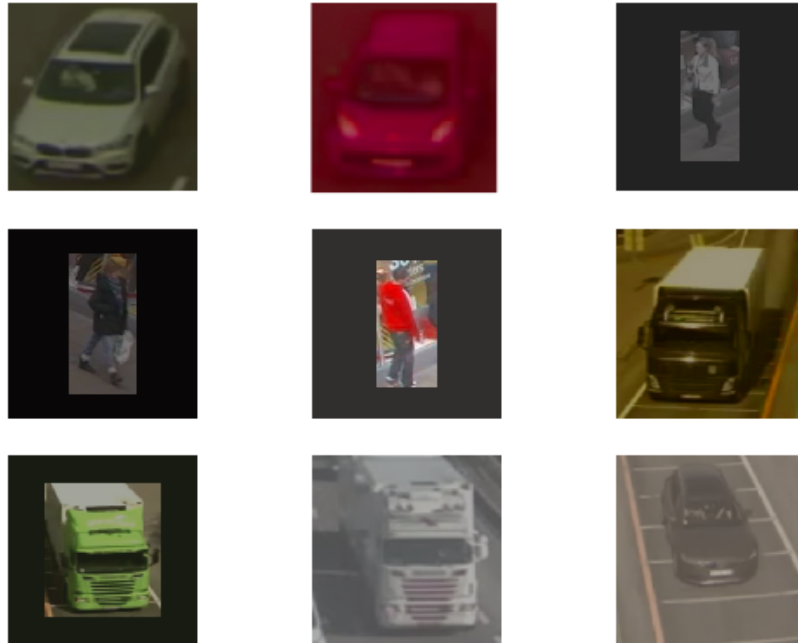


Figure 4.2: Data augmentation of the original dataset in figure 4.1

With augmentation the dataset were increased to 23800 images for both cars and trucks. Splitting the dataset into training, validation and testing datasets proposes two challenges: with less training data, the parameter estimates will have greater variance, and on the other hand, with less test data, the performance statistical will have greater variance. Ideally, that variance should be as small as possible for both.

Ideally, the networks should be trained with different sizes of test, validation and training data sets, to evaluate the combinations that offers the best results. However, an assessment has been made that the dataset is split into 80:20 for the training and test data sets, and then the training set is again divided into 80:20 for the training and validation set respectively. It provides a test data set of 4750

images, training data set of 15,232 and validation set of 2808.

4.2 Testing neural networks architectures

In this section testing and evaluating of various network architectures are explained. The networks architectures are composed with a combinations of different input size, batch size, learning rate and combinations of convolution, dense layers and pooling layers. All input images that does not meet the input size of the neural network are zero padded. The network is designed to predict three classes considered important in traffic surveillance. The three classes are cars, trucks and pedestrians. Detecting pedestrian proposes important features to the system in regards to safety [20].

There are no generalizing rules for building the convolution neural network, but Xudong Cao[11] conducted a paper aiming to find theories for designing very deep convolutional neural networks. The paper concluded that an image size of 128x128 pixels are sufficient, in addition to using small convolutional filters. There are several examples of small filters being applied with success, one of the is the VGG Net [40] , which is one of the most successful neural networks for image classification. The pooling size is fixed to 2x2 with stride 1 (equals down-sampling the volume by a half).

The mini-batch gradient descent algorithm(as described in forms the optimization of the neural network. Mini-batch gradient descent usually operates with batches with size of 32–512 images[30]. The main problem with large batch sizes are sharp local minima which leads to overfitting [25] and GPU memory requirements. According to Yann Lecun [28] is a batch size of one theoretically the best approach, but requires far longer time to train the network. The VGG net was trained with a batch size of 256. To limit the GPU memory use, and optimize the training time, the batch size is fixed to 128 images. In most cases a single validation set of respectable size substantially simplifies the code base, without the need for cross-validation with multiple folds [24].

The structure of the architectural testing is illustrated in figure 4.3.

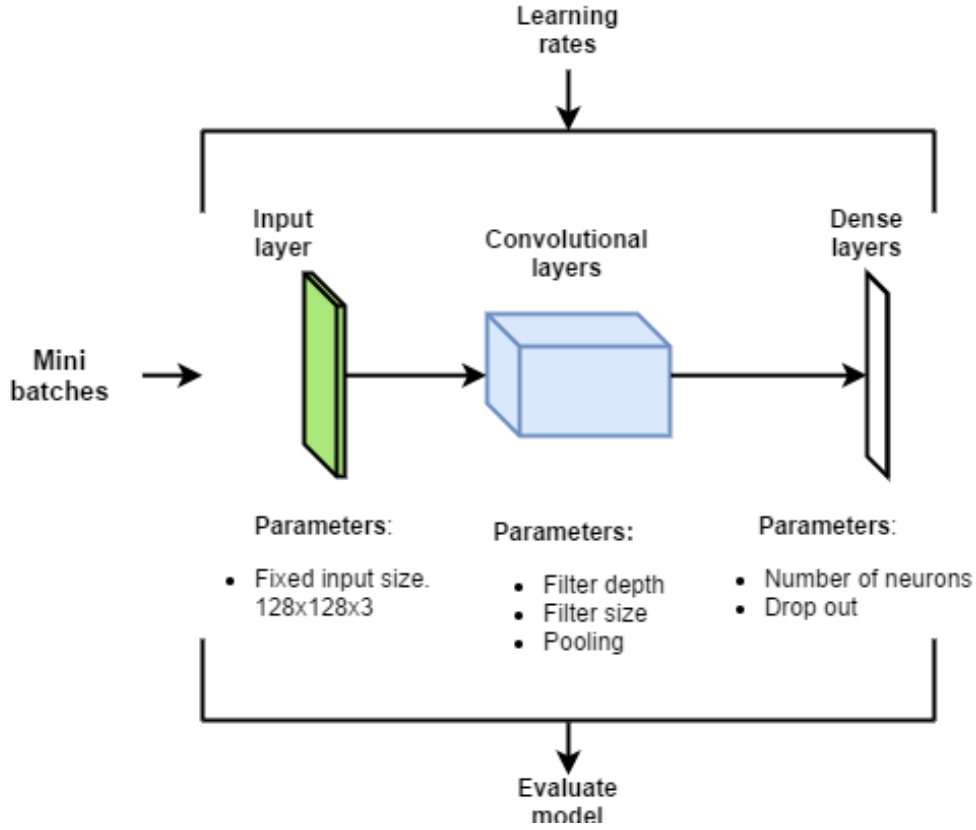


Figure 4.3: Experimenting on different architectures of the neural network

The mini batch size and input size is fixed to 64 and 128x128x3 respectively. The datasets are split into minibatches and fed into the network. The images in each mini batches are randomly picked, and the network is updated according to its gradient descent.

For the mini-batch gradient descent it is for efficiency of the estimator that each example or mini-batch be sampled approximately independently. Faster convergence has been observed by shuffling the dataset[9].

The remaining hyperparameters were compromised within fixed ranges to limit possible architectural structures for the model. The learning rate is defined as either 10^{-6} or 10^{-5} . The convolutional filter size is defined between 3x3 to 8x8 and

filter depths between 32-512, depending on the size of the network. The pooling layer may or may not be used, but with fixed parameters. The neural network takes a lot longer to train without pooling, but some features in the image may be lost using it. Outputs from convolutional layers are zero padded to preserve the same spatial volume as the input.

Dense layers were given input neurons ranging from 512 to 4048, with and without drop out. Note that drop out is just used during training. Each model is trained for 100 epochs and saved when the validation accuracy outscores previous models.

A total of 20 models were trained, varying from 4 to 12 convolutional layers, and two to four dense layers.

Figure 4.4 shows examples of models that were disposed of due to no accuracy improvement. These models were all constructed with four convolutional layers with filter depth 32,64,64,64 respectively, but different pooling and filter size. Regardless of various pooling and filter parameters the model fails to improve its validation accuracy 4.4 (model 2,3,4,7 in appendix C).

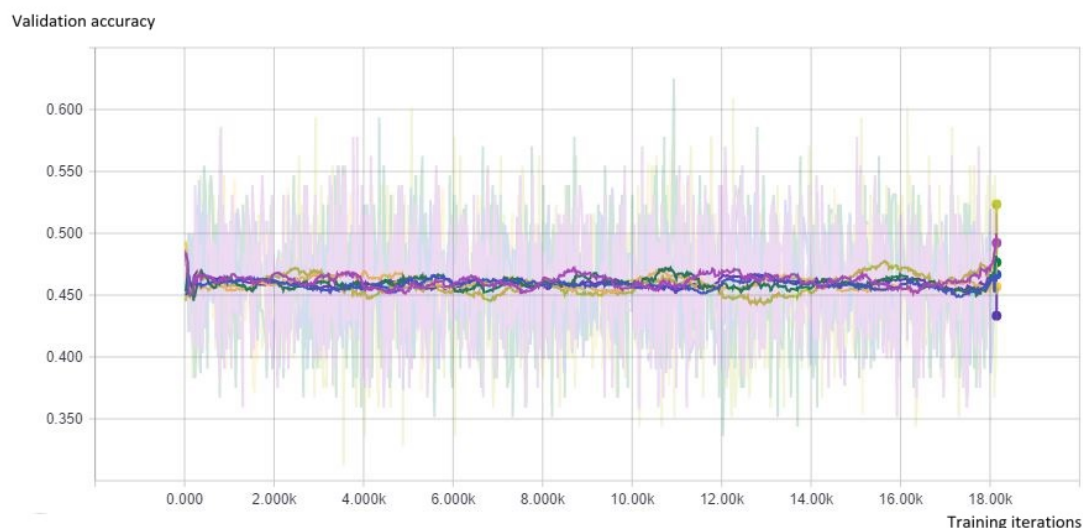


Figure 4.4: Accuracy of neural networks plotted vs epochs. The colors represent neural network models with different architectures.

Figure 4.4 shows no sign of improvement of accuracy, even though the accuracy is floating between 0.35 and 0.55. The accuracy of the validation testing for the top 5 most promising models are shown in figure 4.5. These models were constructed with deeper architectures and more complex filter depth(model 1,4,5,6,10 in appendix C).

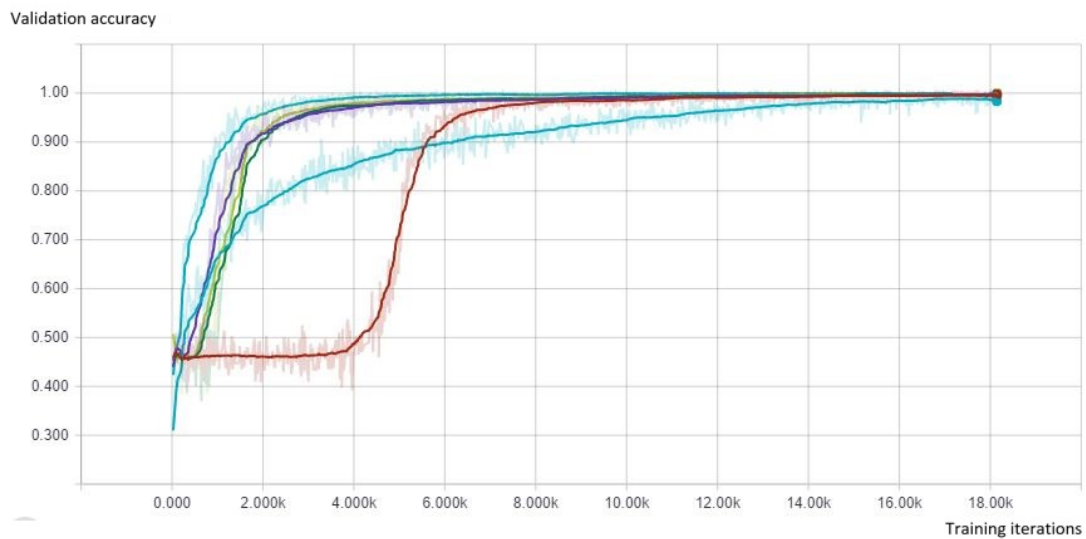


Figure 4.5: Neural network architectures with good accuracy. The accuracy is plotted vs training iterations.

All models that shows high accuracy are saved and used for testing on the *test dataset*.

The best architecture from the validation tests is shown in figure 4.6.

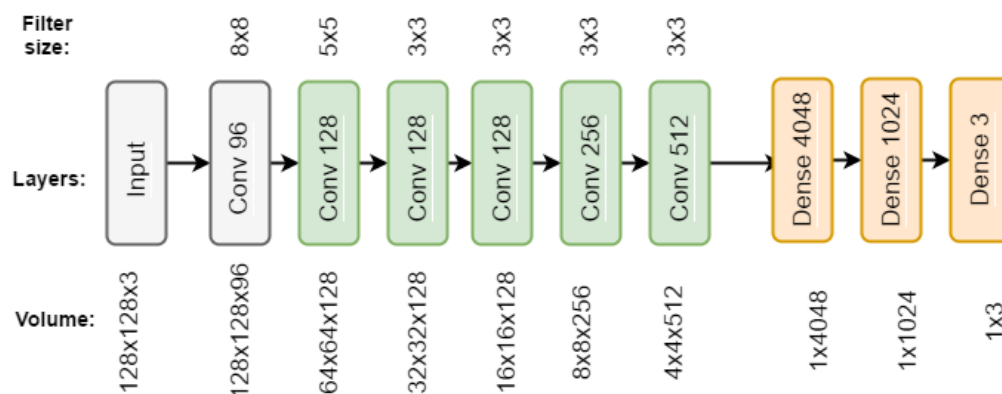


Figure 4.6: The architecture of the neural network with the highest accuracy score

The first convolutional layer is without max pooling. The volume is the number of output neurons from each layer. The last convolutional layer represents the feature vector as a 4x4 image with 512 filters. The model achieved 99,9% validation accuracy.

The validation score itself is not very interesting in regard of network performance, because it only proposes an *indication* of how the model will perform. The test that determines how the network will behave in real is carried out on the *test dataset*.

4.2.1 Testing the classifier

The best architectures from the training was tested on the *test dataset (TD)*. The results for the TD were promising. The result of the test is presented in a confusion matrix, where the *predicted* class are at the top, and the *true* class is at the left. The model with the best classification score is shown in table 4.7.

		Predicted class			
		Car	Truck	Pedestrian	All
True class	Car	2207	23	3	2233
	Truck	0	1828	34	1862
	Pedestrian	1	0	1904	1905
	All	2208	1851	1941	6000

Figure 4.7: Confusion matrix after classifying the *test dataset*. Achieved 99.0% accuracy

E.g of 2233 images *cars* were misclassified as *trucks* 23 times. The confusion matrix shows the biggest error is due to truck being classified as pedestrian, followed by cars being classified as pedestrians.

The normalized confusion matrix is shown in 4.8.

		Predicted class			
		Car	Truck	Pedestrian	All
True class	Car	0,985	0,005	0,011	2273
	Truck	0,001	0,968	0,032	1860
	Pedestrian	0,001	0,003	0,996	1867
	All	2241	1816	1943	6000

Figure 4.8: Confusion matrix after classifying the test set. 99% accuracy

Finding how each class has been classified The matrix gives the percentage of elements of real class classified as each class. The matrix is achieved by dividing each element by the sum of the elements in each row. The matrix gives a better indication of the error rate. From the current dataset the accuracy is approximately 99%.

The error rate of a truck classifying as a pedestrian is 0.03 %, which means the network misclassifies a approximately three in a hundred images. The second largest error rate is a result cars being classified as a pedestrian. This error rate is 0.01%, which gives about one error prediction out of a hundred images. Pedestrians, on the other hand, are classified almost flawlessly, with only 7 misclassifications out of 1867 images, which is equivalent to around 3 misclassification per thousand images. This may be a indication that the network is overfitting the pedestrian class. Ideally, the classifier would avoid misclassifying vehicles as pedestrians because this misclassification proposes the biggest safety challenges.

Figure 4.9 shows some of the misclassifications.

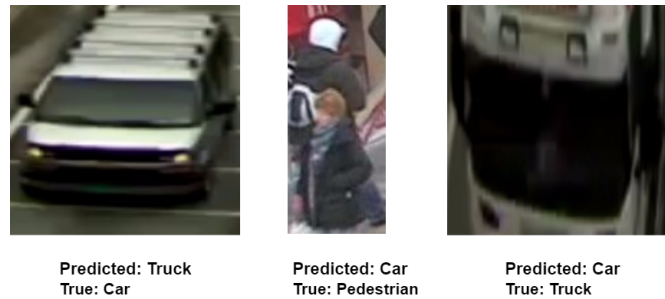


Figure 4.9: Misclassifications of the dataset

Some of the misclassifications are understandable, e.g the pedestrian image shows more than one person, and the image of the truck is very close up, which changes the features of the input images.

4.3 Arranging the experimental setup

All cameras are in a fixed position, filming a static background. The neural network is trained to classify the vehicle front, which implies that the camera must be aligned to capture incoming traffic.

Cameras from Statens vegvesen are mounted on poles or structures above or adjacent to the roadway, capturing incoming traffic, illustrated in figure 4.10.

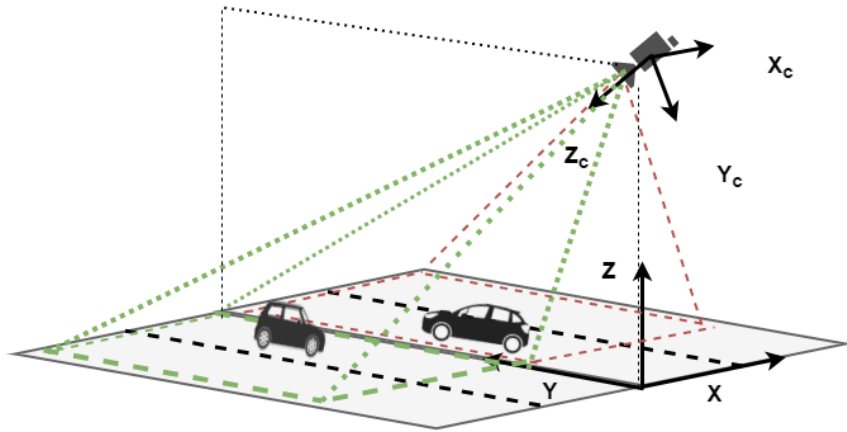


Figure 4.10: The camera may be mounted anywhere along the x-axis, as long as the green area is observable.

The all tests were performed with Axis P1346 cameras with a top-down view. The

recordings are from the destinations in figure 4.11.



Figure 4.11: Recording destinations. 1 is E18 Maritim, 2 is E16 Sandvika, and 3 is E18 Sandvika

The image resolution from the cameras are 800x600 pixels, with a frame rate of 1 frame per second. The video stream are both composed of objects and noise, thus, the result of the detection has four outcomes, presented in table A.1.

	Vehicle present	Vehicle absent
Detected	Hit	False positive
Not detected	Miss	Correct rejection

Table 4.2: Detection table

Multiple detections of the same vehicle are counted as hits in the initial tests. All images that contains recognizable objects are considered a positive, illustrated in figure 4.12 and figure 4.13.

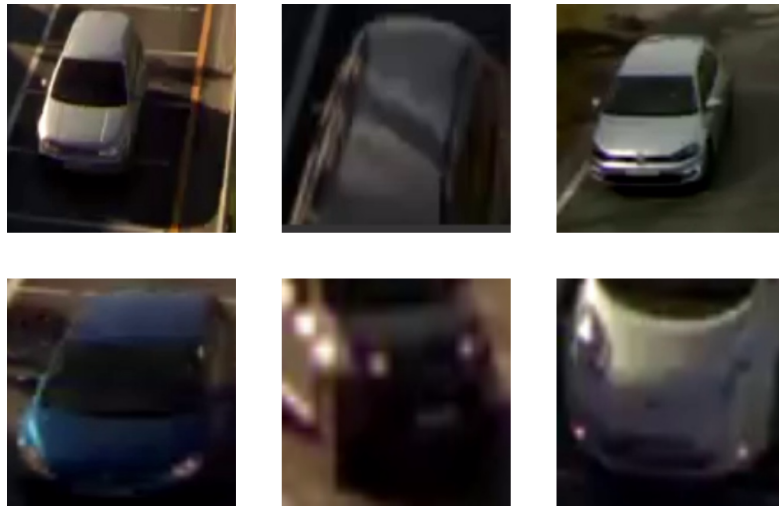


Figure 4.12: Hits detected

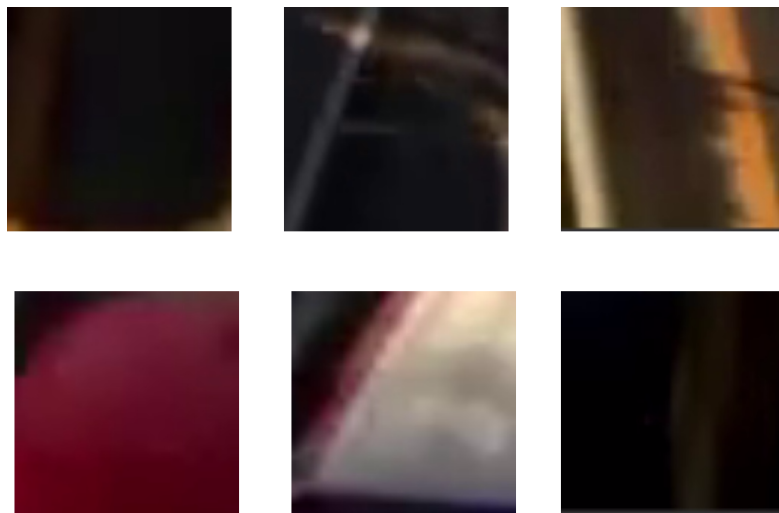


Figure 4.13: False hits detected

The performance of the model is measured with a hit rate. The hit rate is computed by dividing the sum of valid detections on all detections, in epochs of five minutes.

$$hitrate = \frac{\sum_i^n i}{\sum_i^m i} \quad (4.1)$$

where n is the number of detections, and m is the number of true detections. The detections are manually sorted as either false positive or true detections. Detections of non-vehicles are considered false detections.

4.4 Initial considerations of the acquired data

Datasetet var hentet på forskjellige tider og ved forskjellige værforhold. Dette skapte en innledende arrised questions i forhold til validity på den acquired data. thus initial consideration were done to investigate the data and to skille ut de reliable datasets. Several steps were taken to achieve this.

1. Determining a reasonable vector space
2. Initiating the Kalman filter
3. Hit rate during sunny days
4. Hit rate during cloudy days
5. Hit rate at different locations
6. Multiple lanes

4.4.1 Determining a reasonable vector space

Deciding a reasonable vector space must be done every time new locations are used by the system, because the geometry of the lane changes, which implies different driving pattern of the vehicles in different locations. The location of this test is shown in figure 4.14.



Figure 4.14: The system is adjusted according to the location. The black boxes are blocking personal properties, due to privacy regulation.

The adjustment is completed by computing vectors between coordinates. Figure 4.15 illustrates how the vectors are computed. The origin of the coordinate system is in the upper left corner.

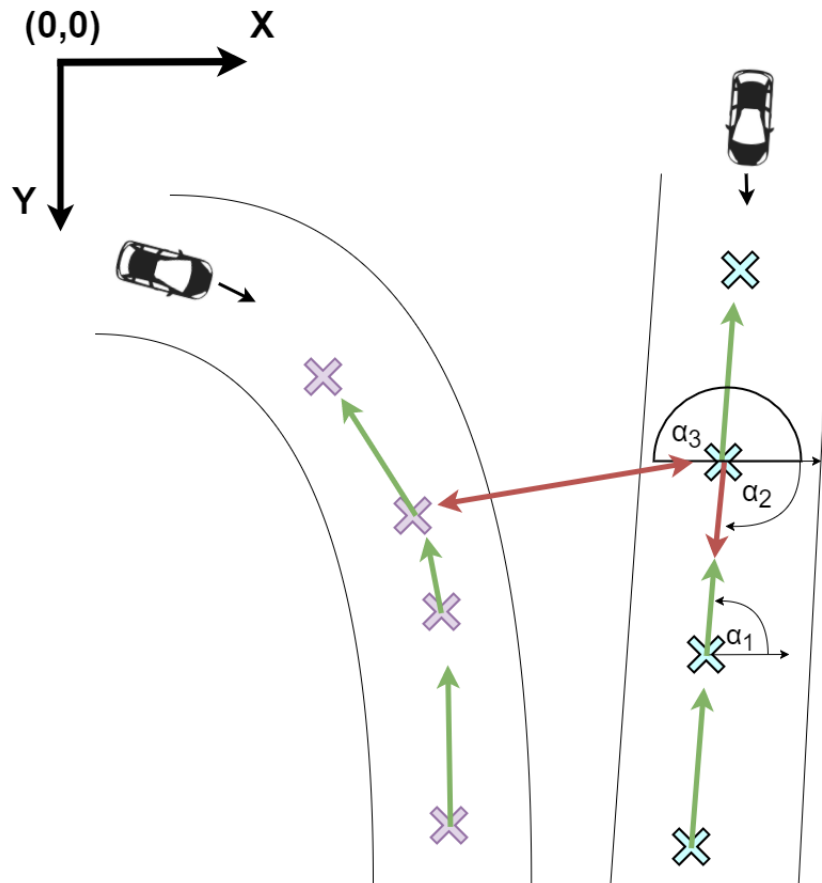


Figure 4.15: Vectors between vehicle coordinates

The purple and turquoise marks represent two separate vehicles driving in the direction of the Y-axis. The valid vectors are the ones that point backwards, because each new measured coordinate is looking to join an existing object. If the vector points downwards, a vector pointing downwards is not valid, because it implies the vehicle is looking for points ahead of itself (e.g. the vehicle in front), which is undesirable. The angle between the points is measured in relation to the x-axis.

α_1 is a valid vector in figure 4.15. α_2 is the opposite vector of α_1 , which is $\alpha_1 + 180$. The vectors are defined between -180 to 180 , which implies angles exceeding 180 , will continue at the negative axis. For each valid vector α_1 , a corresponding invalid α_2 will exist. α_3 is a vector between the two lanes and will be invalid in both directions, because no correspondence is desired between the lanes.

Figure 4.16 shows the result of computing the vectors between all measured coordinates in a one minutes interval in relation to each other.

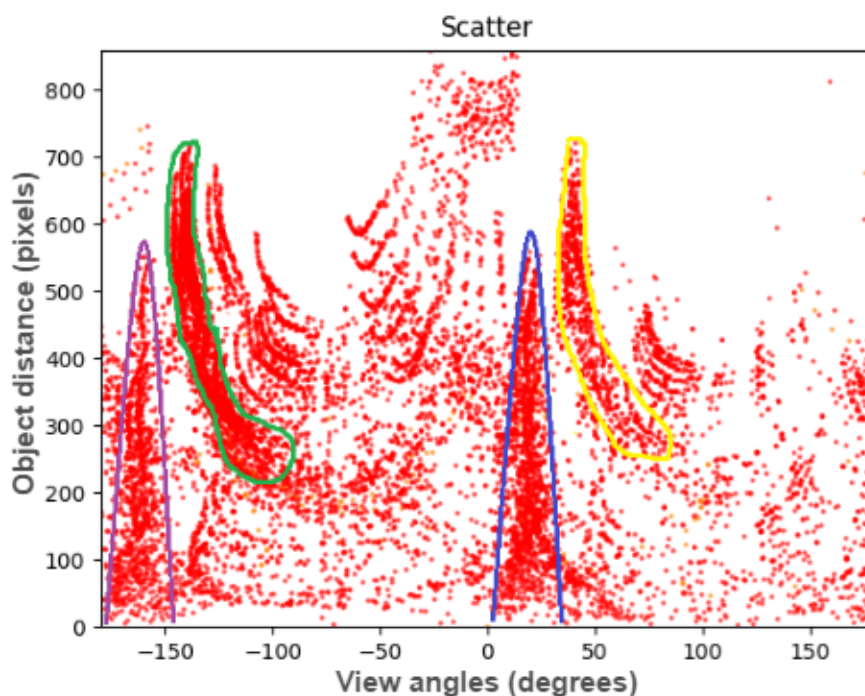


Figure 4.16: Distance between measured positions based on frames per second

The blue and green area are represent the valid space for the two lanes. Since the blue and green are valid vector space, they will project two equal punctures that are skewed 180 degrees. This is reflected in figure 4.16, where the purple parabola has a offset of 180 relative to the blue parabola. The yellow area has an offset of 180 relative to the green. Both purple and yellow are invalid vector area. The seemingly random dots in the plot are coordinates in the two lanes and noise in the extracted foreground.

How the pixel distance changes as a result of various frame frequency is illustrated in figure 4.17

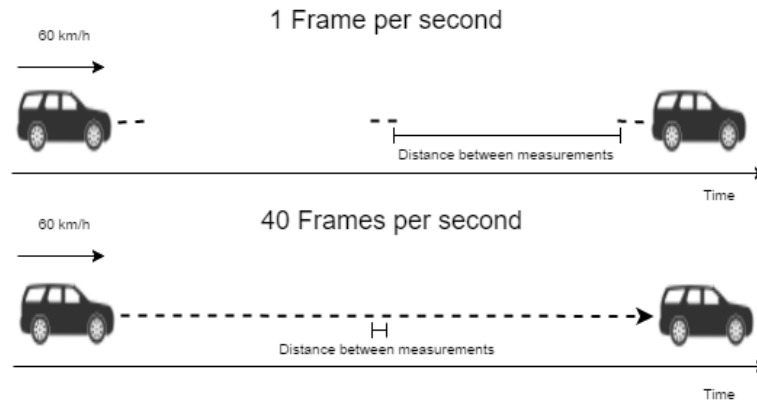


Figure 4.17: Distance between measured positions based on frames per second

Only one lane were used in this test, thus the blue line in figure 4.16 constrains the vector space. The area is manually chosen. The parabola is computed by the following equation.

$$y = a(x - x_o)(x - x_1) \quad (4.2)$$

Where x_o and x_1 is left and right, respectively, intersection between the parabola and the angle-axis. a is solved by inputting the maximum distance value for y and $\frac{x_o+x_1}{2}$ for x , in equation 4.2.

4.4.2 Initiating the Kalman filter

The coordinates for a single vehicle from the background subtraction are plotted. The model is considered constant velocity, and the noise sources is assumed to be white Gaussian distributed. The measurements are considered real values. The velocity is given as $v = \frac{\sqrt{((x(t)-x(t-1))^2+(y(t)-y(t-1))^2)}}{\Delta t}$, where t is a given frame, and x and y are pixels coordinates. The time dependent terms of the state transition matrix are updated every time the time step is changed.

$$\Phi = \begin{bmatrix} I & 0 & \Delta t & 0 \\ 0 & I & 0 & \Delta t \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \quad (4.3)$$

The initial error covariance of the state vector is set to $I \cdot 0.1$.

$$P = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \quad (4.4)$$

Which means the standard deviation of each initial guess is the square of 0.1, = 0.1. So for all cases, 68 % of the time the initial inputs are within +/- 0.3 units and 95% of the time within +/-0.6 pixels. This statement is reasonable because the initial position of the vehicle is relative accurate, which gives an initial error covariance for x and y that is close to zero, and a 95% confidence that the initial position is within +/-0.6 pixels. And the vehicles is moving with +/-0.6pixels/dt. The effect of initializing a measurement noise covariance matrix close to zero means the proportion of error in the correction step is going to be nearly 100% attributed to the model.

The filter is manually tuned until the prediction fits the measurements.

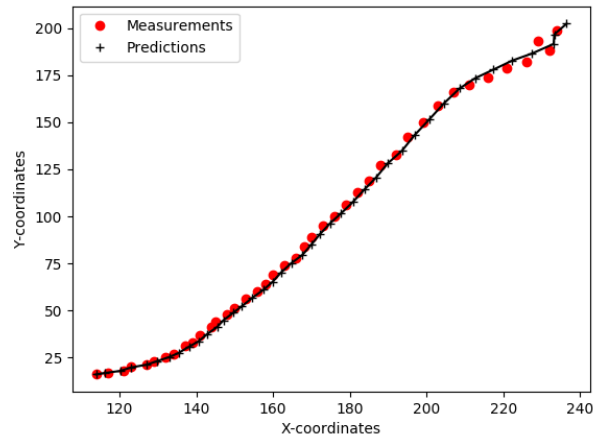


Figure 4.18: Measured coordinates in pixels and the corresponding predictions

The norm of the state covariance matrix is shown in figure 4.19.

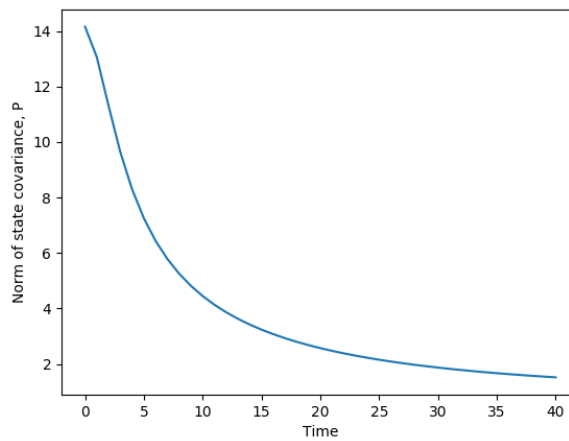


Figure 4.19: The norm of the state covariance matrix plotted vs time

The state covariance matrix converges rapidly, which means a lower uncertainty. The values for Q and R was retained for the following tests.

4.4.3 Hit rate during sunny days

The following initial test were conducted to see how the system responds to the sun changing position during the day. This is to determining if utilizing the system at different times of the day will be decisive for the result.

Recording position	Time	Day	Weather	Lanes
Sandvika E18	08:00-09:00	15.05.2017	Sun	one
Sandvika E18	11:00-12:00	15.05.2017	Sun	one
Sandvika E18	14:00-15:00	15.05.2017	Sun	one
Sandvika E18	16:00-17:00	15.05.2017	Sun	one
Sandvika E18	19:00-20:00	15.05.2017	Sun	one

Table 4.3: Recordings from Sandvika E18

These results are within the practical scope of this project, thus some time gaps are present.

Figure 4.25 and figure 4.21 shows hit rate during the time interval.

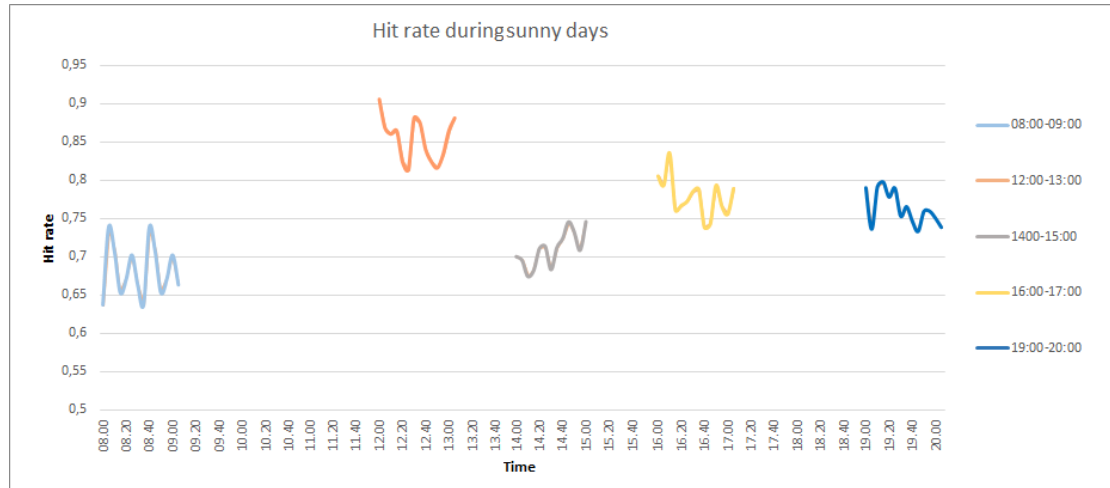


Figure 4.20: Hit rate at different *time of day*

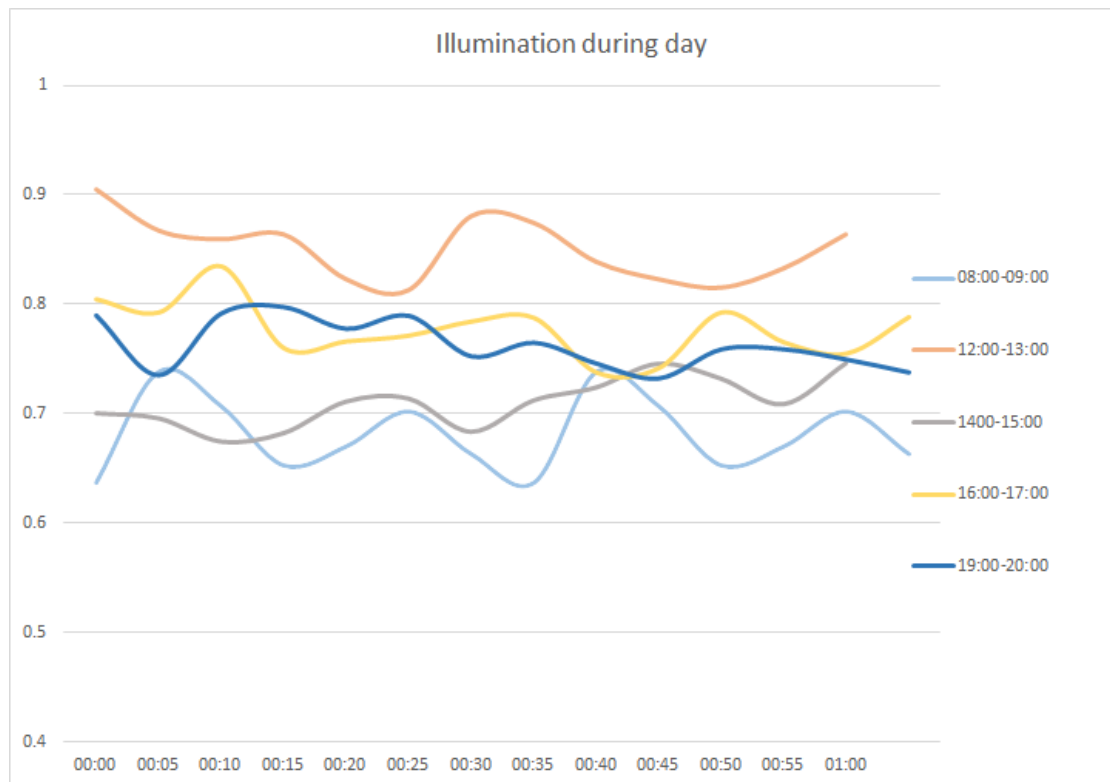


Figure 4.21: Hit rate at different *time of day*, parallel view of figure 4.25

The test shows significant difference in hit rate in relation to the *time of day*, where the average hit rate is varying from approximately 0,70% to 0,85% for 08:00 and 12:00-13:00 respectively.

The test shows significant difference in hit rate in relation to the *time of day*. The position of the sun has an impact on the projected shadows from the cars, which in turn may lead to false positives. Figure illustrates how the sun provokes shadow.

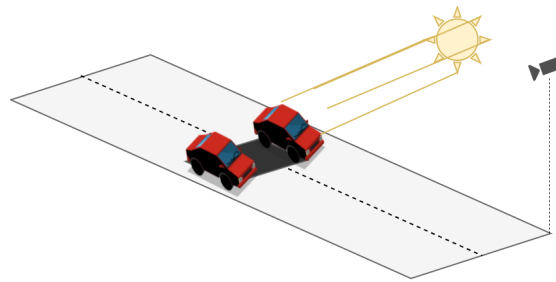


Figure 4.22: Shadow projection caused by the sun

The reason is two-folded, shadow from the surroundings is detected as vehicles, and shadow from vehicles that are not filtered out. This shadow causes vehicles to be detected as the same blob, and thus difficult to differentiate and classify. It can be concluded that illumination proposes considerable error on the hit rate.

4.4.4 Hit rate during cloudy days

The impact of *cloudy days* is measured by filming the same locations, with cloudy weather conditions, in different time intervals between morning and evening.

The recordings are listed in the table 4.4:

Recording position	Time	Day	Weather	lanes
Sandvika E16 Nord	08:00-09:00	15.05.2017	cloudy	one
Sandvika E16 Nord	14:00-15:00	15.05.2017	cloudy	one
Sandvika E16 Nord	19:00-20:00	15.05.2017	cloudy	one

Table 4.4: Recordings from Sandvika E16

The results of detecting and tracking in various times during the day is presented in a graph in figure 4.23 and figure 4.24.

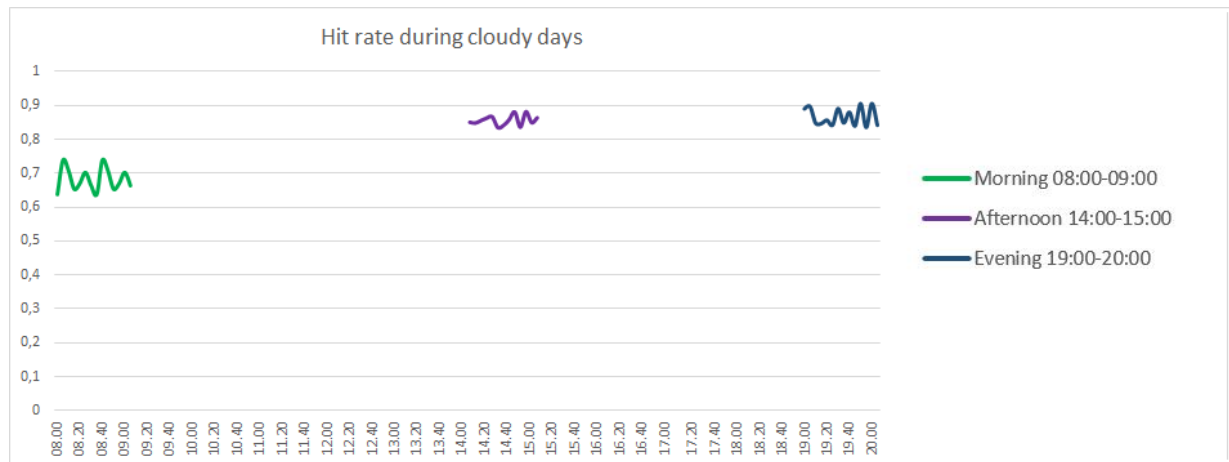


Figure 4.23: Hit rate during cloudy days, plotted between 08:00 - 20:00

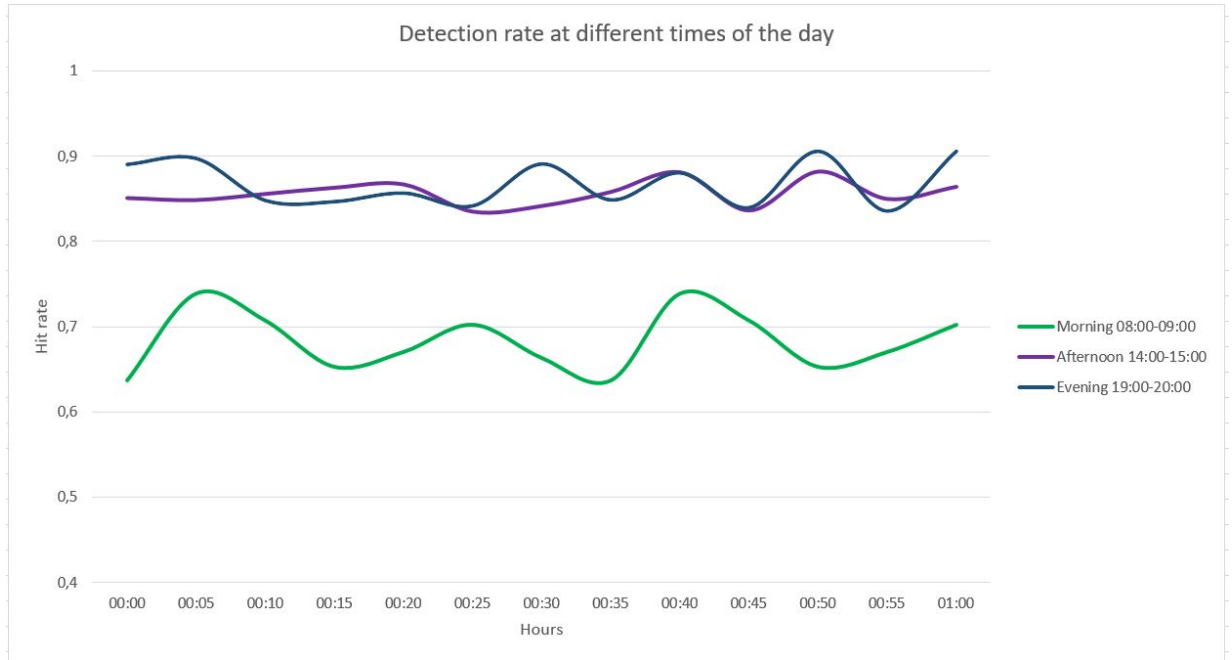


Figure 4.24: Hit rate during cloudy days, parallel view of figure 4.23

The experiment indicates that performance of the system is lower in the morning, compared to measurements later in the day. One apparent reason for the lower hit rate in the morning is higher frequency of vehicles. Higher frequency of vehicles entails occlusion, which in turn may confuses the tracking module. Backed by this results, further analysis will use cloudy data because its more reliable than sunny weather.

Due to the considerable amount of data needed for the tests, a representative time interval had to be selected to make further analysis within a reasonably time frame.

The interval is selected from the initial tests, where the plot indicates a stable hit rate around 14:00 with cloudy weather. Furthermore, the time at 14:00 will be defined as zero point.

4.4.5 Recordings from three different locations

The impact of *different locations* are measured by recording three different locations at the same time of day. The recordings are listed in table 4.5.

Recording position	Time	Day	Weather	lanes
Sandvika E16 Nord	14:00-15:00	15.05.2017	Cloudy	one
E18 Maritim	14:00-15:00	15.05.2017	Cloudy	one ^a
E18 Fiskvollbukta	14:00-15:00	15.05.2017	Cloudy	one

Table 4.5: Recordings from Sandvika E16

^aThis destination has originally multiple lanes, but only one of the lanes are recorded in this test

The result of the test is shown in figure 4.25.



Figure 4.25: Hit rates for the three selected locations

The quality of the data from this three locations are very similar, with a average hit rate of approximately 0,85 %. Thus the data from these three locations will be merged for further analysis. In addition it can be assumed that the system is representative of roads that are relatively straight and with single lanes.

The colors in graph represents the different locations, and by further analysis, the locations will be represented by the same colors.

4.4.6 Multiple lanes

The *multiple lane* experiment aims to provide results to which extent the hit rate is affected by detecting multiple lanes. The recordings are from at same time at day, but different locations.

Recording position	Time	Day	Weather	lanes
Maritim E18	14:00-15:00	03.05.2017	cloudy	one ^a
Maritim E18	14:00-15:00	09.05.2017	cloudy	three

Table 4.6: Recordings with multiple lanes

^aOne lane cropped out

The results from the test lane testing is presented in Figure 4.26.

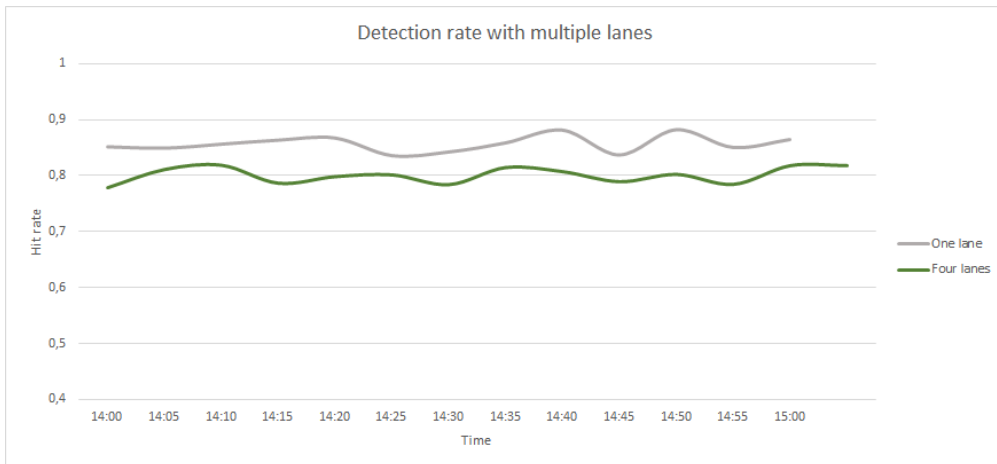


Figure 4.26: Multiple lanes

As expected is the hit rate for multiple lanes poor compared to single lane detection. Multiple lanes proposes challenges as several objects are tracked simultaneously, as multiple object tracking increases the likelihood of mixing vehicle coordinates as they change lanes. In addition the vehicle frequency is higher with multiple lanes, as more cars pass the camera over the same time period compared to a single lane.

Based on the initial tests, it can be assumed that future tests must be conducted at the same time of the day and with the same weather conditions, and with one lane.

4.5 Analysis and presentation of traffic data

The essential part of the data analysis is to present the data in informative format to simplify the decision making and the planning for the end user(Statens vegvesen). Data from the initial tests were used for further analysis.

The three locations within the given prerequisites(cloudy, 14:00-15:00 and a single lane) gave a dataset of approximately 3000 images.

A report from Statens Vegvesen[4] sums up the desired target data from the analysis.

- Velocity distribution
- Density of vehicles
- Traffic congestion
- Vehicle classification
- Vehicle class frequency

These analysis will be presented in the next sections.

4.5.1 Velocity distribution

This analysis shows how the relative speed between the cars are distributed. The database stores information about the *pixels per second* velocity for each moving objects. The real distance of the road is necessary to convert from pixels per second to meter per second.

Figure 4.29 shows how the distribution of the velocity of the vehicles are. Pixels per frame may also be seen as pixels per second, because of the frame frequency is 1. In the plots, 9-25 means the interval between 9 pixels per second to 25 pixels per second and so forth.

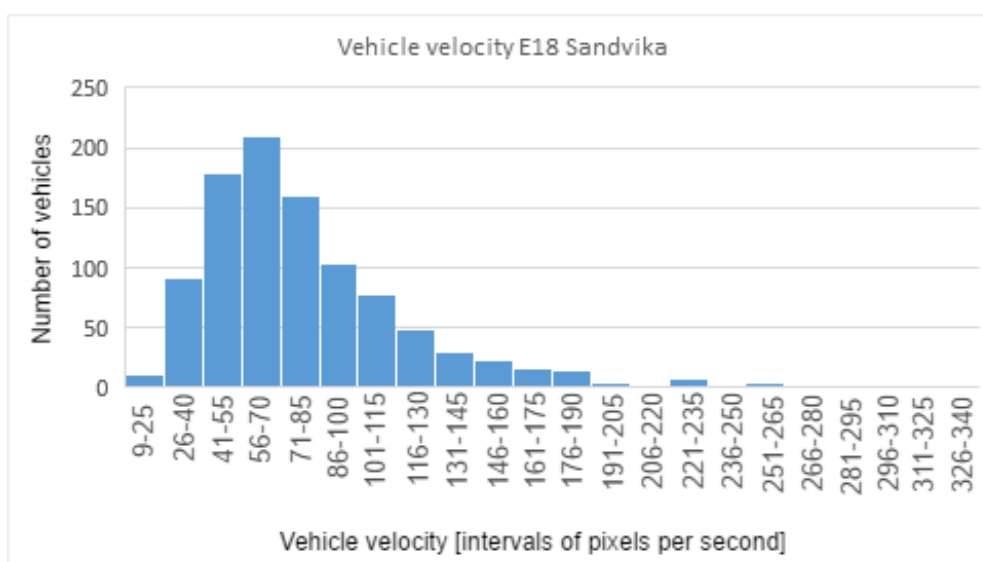


Figure 4.27: Vehicle velocity at E18 Sandvika.

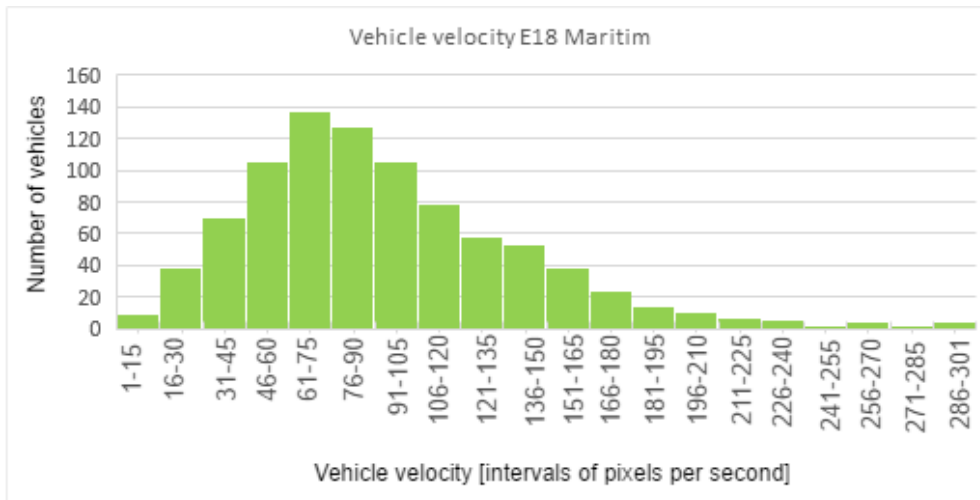


Figure 4.28: Vehicle velocity at E18 Maritim.

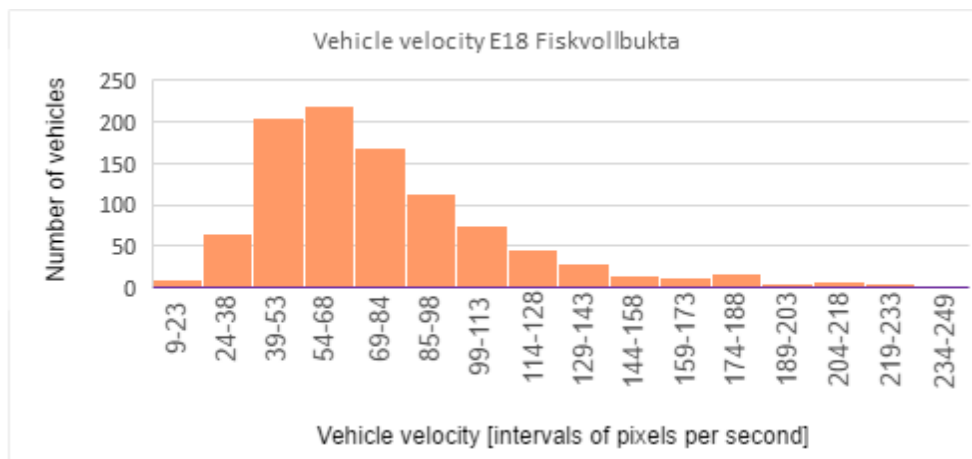


Figure 4.29: Vehicle velocity at E18 Fiskvollbukta.

All the plots indicates a normal distribution with expectancy value of approximately 60 pixels per second. The three locations have a similar speed limit, which indicates that the speed measurements are relatively consistent.

4.5.2 Density of vehicles

This analysis gives information about the amount of vehicles passing through the system for given periods. Information about detection time of the objects is stored in the database. Figure 4.30, 4.31 and 4.32 shows the density of vehicles in time intervals of 6 minutes. The data is supported by graphs showing the typical traffic flow pattern. In the plots, 0-5 means the interval between minute 0-5 and so forth.

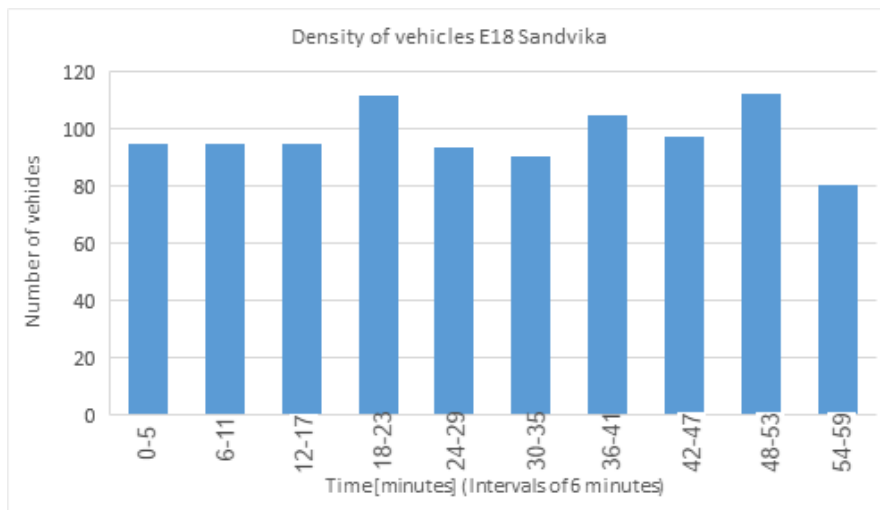


Figure 4.30: Vehicle density at E18 Sandvika

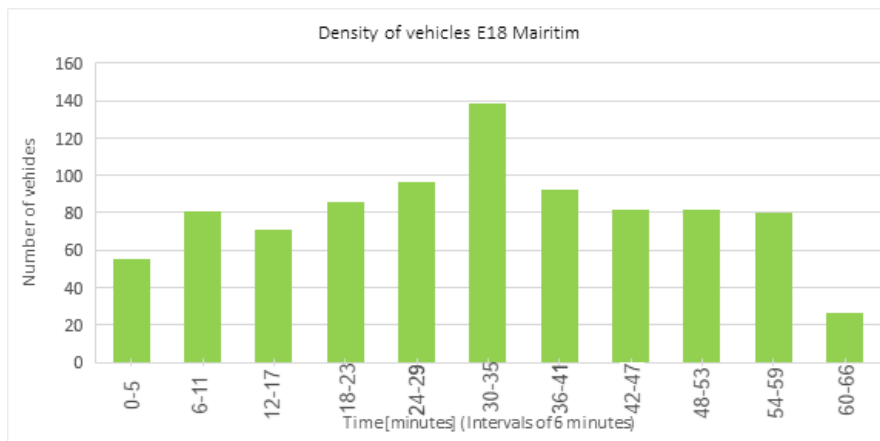


Figure 4.31: Vehicle density at E18 Maritim

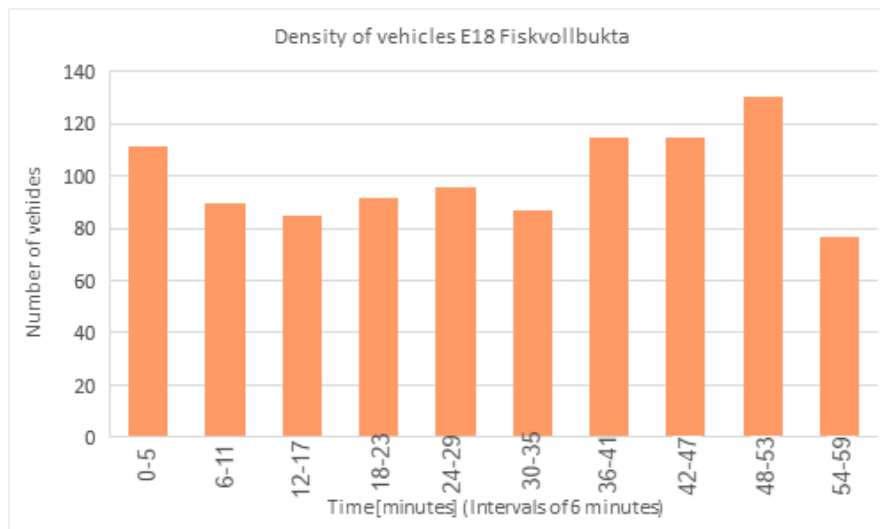


Figure 4.32: Vehicle density at E18 Fiskvollbukta

The plots show relatively continuous flow of vehicles, but there is a somewhat larger variance on the E18 maritime (the interval 60-66 is small because the recordings ended after 62 minutes).

4.5.3 Traffic congestion

This analysis is an result of deriving information from the two previous tests. By combining both analysis one could obtain information about the congestion.

We define the congestion to be high if the velocity is low and the distance between vehicles are short, defined by χ Multiplying the time difference between the detections and the velocity at a given time

$$\chi = v_{o_n} * \Delta t \quad (4.5)$$

where v_o is velocity for object o_n , and Δt is time difference between object o_n and o_{n-1} . χ , is between 0 and 100 where a total traffic jam is represented by 0. Values outside 2 standard deviations of the normal distribution around the mean queue factor, for each measured location, is assumed to be noise, and removed from the plots. The running average of six vehicles are plotted as the purple line.

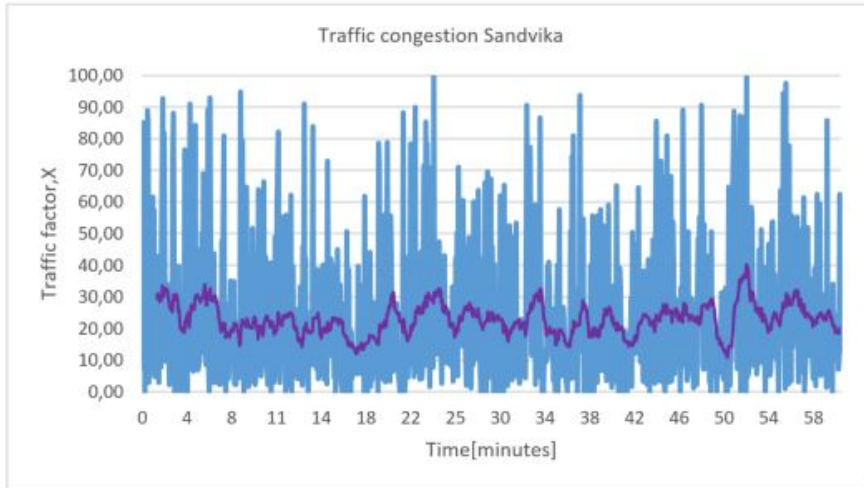


Figure 4.33: Congestion at E18 Sandvika

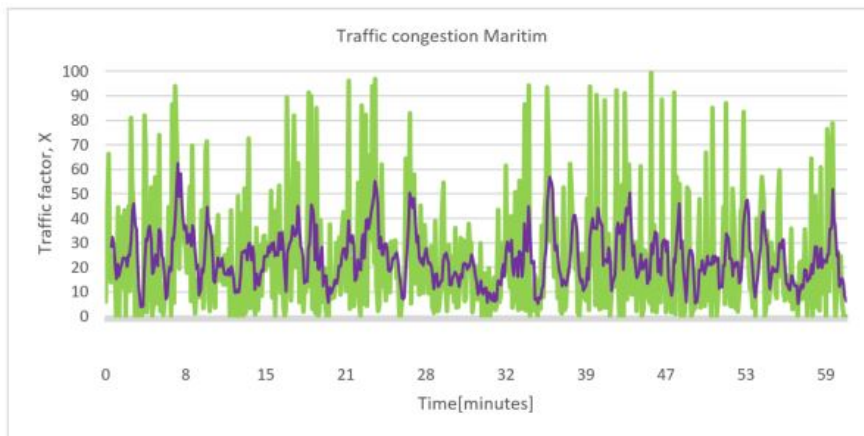


Figure 4.34: Congestion at E18 Maritim

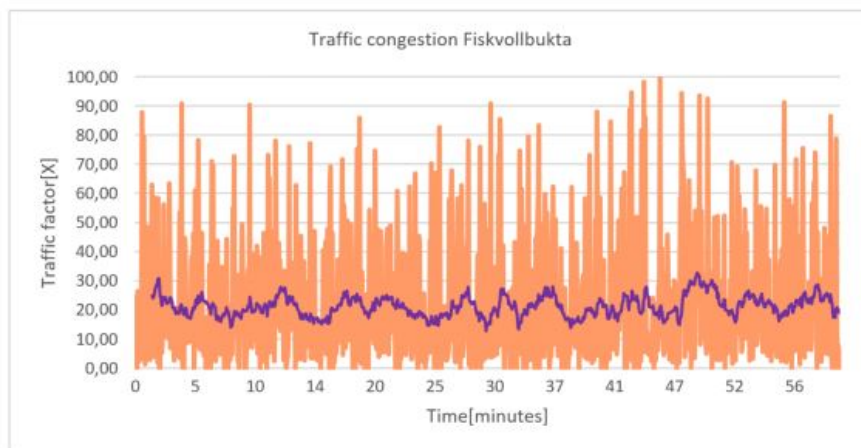


Figure 4.35: Congestion at E18 Fiskvollbukta

The queue situation is determined by low velocity and close distance, and several vehicles have to present at the same time to be able to determine congestion or traffic jams. Each discrete measurements in Figure 4.33– 4.35, does not present congestion by itself. However, if look at the plot of average represented by the purple line, this will be a picture of the moving average, determined by ten objects, will easily give an impression of the current congestion situation. The χ has an

average around 20, however fluctuating between 10 and 40. This makes it difficult to conclude any considerable traffic jam situation. However, this might be natural, due to the fact that video recordings were acquired between 14 and 15 o'clock, thus before people normally go home from work, and in general efficient movement of traffic and minimal traffic congestion problems. It is however an exception in Maritime, where traffic is lower than otherwise. This is discussed later.

4.5.4 Classifying the dataset

The classification of all data from the three destination are performed in the following test. The best model from the initial neural network test is used to classify. The data was first labelled into *car*, *truck* and *pedestrians*, which gave a total of 3000 labelled images.

Unrecognizable data were removed, thus, the classification test implies that the detection algorithm has a 100% hit rate of the detections. This is discussed in 5. This data set will have a large imbalance between classes as a result of frequency of the different types varies. The results of the classification is shown in the confusion matrix 4.36.

		Predicted class			
		Car	Truck	Pedestrian	All
True class	Car	2234	82	0	2316
	Truck	38	643	0	681
	Pedestrian	0	0	3	3
	All	2272	725	3	3000

Figure 4.36: Confusion matrix of the classified dataset

$$Accuracy = \frac{2234 + 643 + 3}{3000} = 0,96 \quad (4.6)$$

and the normalized confusion matrix in figure 4.37.

		Predicted class			
		Car	Truck	Pedestrian	All
True class	Car	0,96	0,04	0,00	2316
	Truck	0,06	0,94	0,00	681
	Pedestrian	0,00	0,00	1,00	3
	All	2159	838	3	3000

Figure 4.37: normalized confusion matrix of the classified dataset

Again, the accuracy is calculated as the overall correct classifications, which is which gives a overall classification accuracy of 96%. All the pedestrian are rightly classified, and in addition none of the other classes are misclassified as pedestrian.

96% of all cars are rightly classified, and the remaining 4 % are classified as trucks. For truck is 94 % rightly classified as trucks, and the remaining 6 % are classified as vehicles.

4.5.5 Vehicle class frequency

This analysis gives information about the frequency of each vehicle class. Applying the results from the classification test in conjunction with the *vehicle density* provides a detailed information about the frequency of each vehicle class are accounted for.

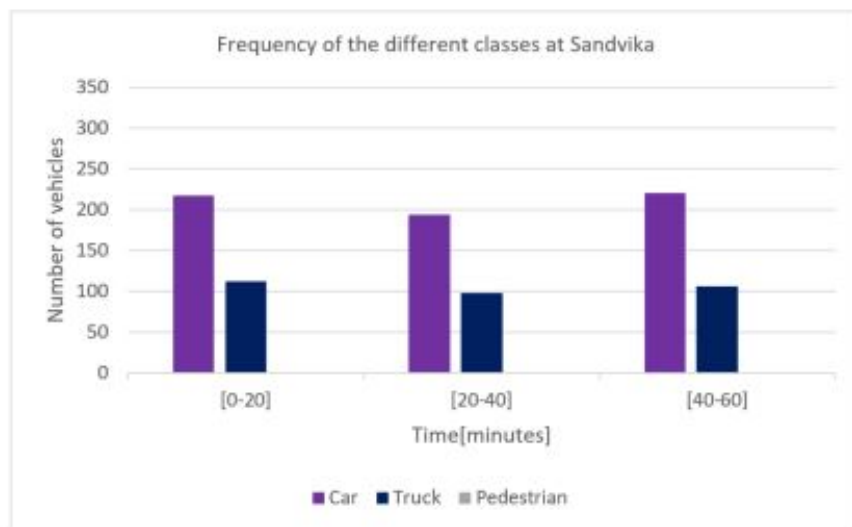


Figure 4.38: Frequency of the three classes at E18 Sandvika in intervals of 20 minutes (no pedestrians were detected)

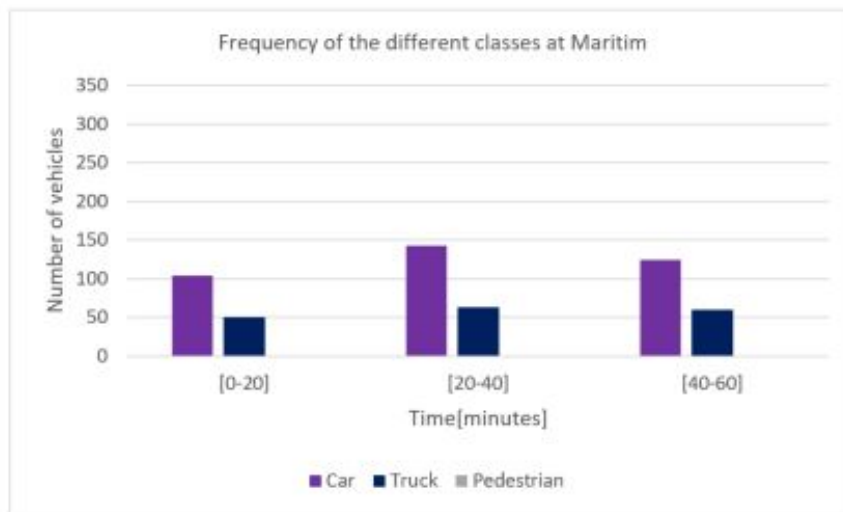


Figure 4.39: Frequency of the three classes at E18 Maritim in intervals of 20 minutes (no pedestrians were detected)

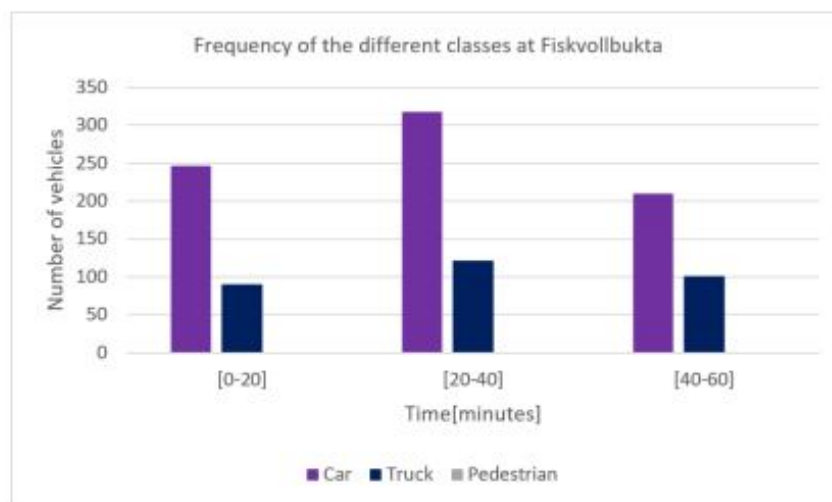


Figure 4.40: Frequency of the three classes at E18 Fiskvollbukta in intervals of 20 minutes (no pedestrians were detected)

These results can be used to determine the amount heavy traffic driving on the road, and thus maintain the road according to needs.

Chapter 5

Discussion

This section considers the experimental results related to the following topics:

- Neural network
- Detection and tracking
- Data analysis

Neural network

The neural network shows good ability to distinguish between the different classes, but is dependent on the data from the detection module being reliable. There is a performance fall when the network is used in new locations, that may be a result of the training data and test data are retrieved from the same location, while the new data set is retrieved from a new locations. Another explanation is the different in frequency of the three classes. There are, for example, only three 3 pedestrian who have been moving in front of the camera during the recordings. This affects the classification results if the *car class* classifies with less accuracy than the *pedestrian class*.

In the test, unrecognizable data was removed before classification. This is because unrecognizable data randomly classified, which induces noise in the result,

which in turn makes the result not representative for the classifier. A solution is to introduce a background class that classifies unrecognizable data as a background, and removes it from the system.

The network is competitive in comparison with other methods performed by classification. In a comparative study by S. Munder and D.M. Gavrilu [29] with pedestrian classification achieved the best method, support vector machine, a classification accuracy of 95.0%, which is lower than the 99.6% neural network accuracy.

Another study by Ambar Dekar [3] compared different classification algorithms, achieving a classification accuracy of 99.25% when classifying between cars and trucks, using a support vector machine. This result is better than the neural network, however, these are methods that only classify between two classes. To extend the methods in order to distinguish between several classes, it must integrate multiple support vector machines, as well as being more resource-intensive than neural networks.

Overall the neural network proves to be a good classifier, and classifies the three classes exact, even though the training dataset is relatively small.

Detection and tracking

Ideally the traffic should have been acquired 24 hours intervals for several days, but however this was not achievable in the time frame of this MSc thesis. The initial test with sunlight indicated that the light affects the rate apparently randomly, thus, results based on data from sunlight recordings will be difficult to conclude from. The recordings from cloudy weather, on the other hand, have shown relatively constant hit rate within given intervals. These tests should be quantified to conclude for which environmental parameters the results are valid. Initial tests have shown that the selected measurement range has provided reliable data sets, which in turn has produced reliable analysis results.

However there is a surprising but interesting deviation of the experimental data, as marked with the dotted lines in figure 5.1, where there is a detected drop around 14:25-14:35. The reason is not fully understandable, but seen in conjunction with plot 4.35, a following reasonable interpretation can be given.



Figure 5.1: Drop in hit rate,

The plot 5.1 shows a drop in hit rate, higher density of vehicles and a lower queue factor (higher congestion) in that time interval. There is a significant increase in the number of vehicles passing the system during this period, which gives reason to believe there is a correlation between congestion and hit rate, due to the fact that vehicles driving closer to each other are more difficult to detect.

Data analysis

Based on reliable data from the initial tests analysis represents useful traffic information.

The congestion analysis should be conducted through a larger time interval in which you will have both morning and evening rush. However, this requires more analysis and experimentation around different lighting conditions, to achieve a smooth hit rate throughout the day.

The relative velocity plot for three locations has clear similarities, and all may be expressed by weibull distributions, and with an assumed mean equal to the speed limit. This means that the extremes of the plot indicates that some vehicles are driving approximately 6 times as fast as the speed limit, which is unrealistic, and assumed to be error detections, thus may be removed from the system. These error detections can be associated with the Kalman filter, which is optimal only for linear models, and the motion model may be unlinear. As a result, when multiple objects are detected, confusion between the objects may occur, which makes coordinates from one object believe it belongs to an object further on the road, which in turn gives higher measured velocity.

In turn, the relative speed is also useful information in connection with traffic queue detections, where low relative speed and small distance between vehicles give indications of queue. However, it was not possible to detect queues during the recordings, so the analysis was omitted. Furthermore, continuous analyzes of the collected data has shown to be useful in determining different traffic factors.

Chapter 6

Recommendations for further work

The thesis holds several elements that could be further investigated and expanded upon. For main topics are recommended for further work:

- Improve and expand the neural network
- Implement other detection and tracking algorithms
- Perform real-time analysis

Neural network

There are several improvements and modifications that can be made to the neural network. The most obvious ones are to train the network on both vehicle fronts and vehicle rears and train with additional classes, e.g a background class can be added to eliminate false positives from the system.

In order to streamline the training process and reduce the need for data, *transfer learning* can be used. Source code for implementing the transfer learning of the Inception Network is attached.

Detection and tracking

As the tracking and detection module seems to be the weak link in the system, it should be considered to investigate other possible improvement of the detection and tracking algorithm. Perform a comparative study solely on detection and tracking algorithms. Several methods may be implemented in the *detection module*:

- Region proposal
- Haar cascade
- SIFT
- SURF

However, none of the methods has shown in general to be superior to each other [39].

There should also be considered using a Thermal camera. The detection algorithm was tested with a thermal camera, which gave good results in terms of filtering out shadows in the video frame. One apparent advantage with thermal camera is its ability to detect at both day and night. The tests were not quantified, and are therefore not a part of the results. The data and source code are attached.

The *tracking module* can be improved by using other methods [19].

- Particle filters
- Support vector machine
- Multiple hypothesis
- Recursive Monte-carlo

An extended Kalman filter could be applied to handle unlinear motion models, but requires more tuning and can be difficult to optimize [32].

It should be considered using video with higher frame rate when tracking objects.

Tests were performed on 40 FPS video, which gave indications of an improved tracking, but these tests were not quantified.

Continuous

analysis

There are several examples of how further analysis can be used in conjunction with forecasting or to establish a relationship between traffic volume and accidents to determine the probable occurrence. A report based on vehicle frequency and classification data shows how data analysis can detect weaknesses in road structure and how road capacity carrying the heavier traffic needs maintaining to extend its durability [4]. By conducting this analysis, the economic aspect can be investigated by distributing the economy where the need is greatest, such as structural upgrading, strengthening or capacity expansion.

Chapter 7

Conclusion

A vision-based traffic system is developed to detect the traffic situation at three different locations in Norway. The data was acquired using a camera, and utilizing a background subtraction algorithm to detect vehicle, and subsequently classifying the vehicles in a neural network. The system has undergone a comprehensive experimental verification, with analysis of more than 20000 images. The results shows the following:

- The algorithm detects 85% of the vehicles with certainty, but there is some uncertainty around the remaining 15%
- Given valid data from the detecting algorithm the neural network were able to classify the type of vehicle with approximately 96%. .

Through analyzes of the acquired data, the following traffic data was determined:

- Velocity distribution
- Density of vehicles
- Traffic congestion
- Vehicle class frequency

This shows that a modern vision-system combining background subtraction and neural network achieves a performance that is capable of acquiring characteristic

data for a modern traffic surveillance system. Finally, recommendations for further improvements are presented.

Bibliography

- [1] Article @ Fluidsengineering.Asmedigitalcollection.Asme.Org.
- [2] V. K. Agarwal, N. Sivakumaran, and V. P. S. Naidu. Six object tracking algorithms: A comparative study. *Indian Journal of Science and Technology*, 9(30), 2016.
- [3] a. Ambardekar, M. Nicolescu, G. Bebis, and M. Nicolescu. Vehicle classification framework: a comparative study. *Eurasip Journal on Image and Video Processing*, 2014(1):1–13, 2014.
- [4] M. o. W. and Transport. Traffic Data Collection and Analysis. (99912 - 0 - 417 - 2):1–54, 2004.
- [5] Axis. Axis Communications.
- [6] L. J. Ba and R. Caruana. Do Deep Nets Really Need to be Deep? pages 1–9, 2013.
- [7] N. Basant. Evils of Overfitting – and How to Minimize them. 2015.
- [8] Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger. Comparative study of background subtraction algorithms. *J. Electron. Imaging*, 19, 2010.
- [9] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTU:437–478, 2012.

-
- [10] S. Bhattacharyya, C. Shen, D. Dawadi, and B. Panja. Detection and Classification of Vehicles Using Wireless Sensor. *Analysis*, 3(1):37–47, 2002.
- [11] X. Cao. A practical theory for designing very deep convolutional neural networks.
- [12] S.-c. S. Cheung and C. Kamath. Robust techniques for background subtraction in urban traffic video. page 881, 2004.
- [13] Codelabs. codelabs.developers.google.com.
- [14] E. Cuevas, D. Zaldivar, and R. Rojas. Kalman filter for vision tracking. *Measurement*, (August):1–18, 2005.
- [15] P. Delmas. Median Filtering Lecture Slides - examples. pages 1–8, 2010.
- [16] R. Faragher. Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]. *IEEE Signal Processing Magazine*, 29(5):128–132, 2012.
- [17] Foswiki. SecureShell @ wiki.ux.uis.no, 2017.
- [18] F. F. Gavrilu. and D. M. PedCut: an iterative framework for pedestrian segmentation combining shape models and multiple data cues. 2008.
- [19] U. K. J. . Himani S. Parekh¹, Darshak G. Thakore². A Survey on Object Detection and Tracking Methods. *International Journal of Innovative Research in Computer and Communication Engineering*, 2(2):2970–2978, 2014.
- [20] R. Hughes, H. Huang, C. Zegeer, and M. Cynecki. Evaluation of Automated Pedestrian Detection at Signalized Intersections. (August), 2001.
- [21] Imagenet. image-net.org.
- [22] R. Javadzadeh, E. Banihashemi, and J. Hamidzadeh. Subtraction Technique and Prewitt Edge Detection. 6(10):8–12, 2015.
- [23] E. Jaynes and F. Cummings. Stamp @ Ieeexplore.Ieee.Org, 1963.

-
- [24] A. Karpathy. CS231n Convolutional Neural Networks for Visual Recognition. 2016.
- [25] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. 2016.
- [26] P. Kumar and N. S. Bindu. A Comparative study on object detection and Tracking in video. 2(12):1784–1789, 2013.
- [27] LeCun, Yann, and M. Ranzato. Deep learning tutorial. *Tutorials in International Conference on Machine Learning (ICML'13)*., pages 1–29, 2013.
- [28] Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. Efficient backprop. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTU:9–48, 2012.
- [29] S. Munder and D. M. Gavrilu. An experimental study on pedestrian classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1863–1868, 2006.
- [30] A. Ng. 1. Supervised learning. *Machine Learning*, pages 1–30, 2012.
- [31] Nvidia. tesla-p100 @ www.nvidia.com.
- [32] N. Obolensky. Kalman Filtering Methods for Moving Vehicle Tracking. 2002.
- [33] M. Piccardi. Background subtraction techniques: a review. 2004.
- [34] PythonWare. www.pythonware.com.
- [35] A. M. Raid, W. M. Khedr, M. A. El-Dosuky, and M. Aoud. Image Restoration Based on Morphological Operations. *International Journal of Computer Science, Engineering and Information Technology (IJCSEIT)*, 4(3):9–21, 2014.

- [36] G. V. Research. Video Surveillance And VSaaS Market Analysis By Product (IP-Based, Analog), By Component (Hardware, Software, Services), By Application (Residential, Retail, Transportation, Government, Corporate, Hospitality, Industrial, Healthcare, Stadiums) And Segment. 2015.
- [37] Riverbank. riverbankcomputing.com.
- [38] J. Scott, M. A. Pusateri, and D. Cornish. Kalman Filter Based Video Background Estimation. *IEEE Applied Imagery Pattern Recognition Workshop*, pages 1–7, 2009.
- [39] M. Shao, D. Tang, Y. Liu, and T. K. Kim. A comparative study of video-based object recognition from an egocentric viewpoint. *Neurocomputing*, 171:982–990, 2016.
- [40] A. Simonyan, K. and Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1, 2014.
- [41] Z. Sitavancová and M. Hájek. Intelligent Transport Systems Thematic Summary European Commission. page 81, 2009.
- [42] C. Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. *Proceedings 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Cat No PR00149*, 2(c):246–252, 1999.
- [43] E. Sudland. Gjenkjenning av kjøretøy ved inn- og utkjøring av tunneler. 2016.
- [44] Tensorflow. Index @ Www.Tensorflow.Org, 2017.
- [45] A. Varghese, G., and Sreelekha. Background Subtraction for Vehicle Detection, 2015.
- [46] S. vegve. Om+webkamera @ www.vegvesen.no.
- [47] S. Vegvsen. Automatisk trafikk-kontroll ATK.

List of Figures

1.1	Problem, solution and development for a traffic surveillance system	8
2.1	Modules in the system.	10
2.2	Extract foreground based on movement in the background image . .	11
2.3	Threshold area for a pixel, $I_{(s,t)}$, given by a mixture of Gaussian distributions. Values within the distributions are considered background.	12
2.4	Vehicle tracking with occlusion. The dark spot is the occluded area	19
2.5	Prediction and correction steps in the Kalman filter	20
2.6	Uncertainty of the vehicle position in subsequent frames is given by the blue probability density function	21
2.7	Uncertainty of the measurement and predicted position. The variance of prediction is bigger than the measurement	23
2.8	An artificial neural network with two inputs, one hidden layer and two outputs	26
2.9	Activation's inside a single neuron in the neural network.	26
2.10	The sigmoid function and its derivative	27
2.11	Back propagation error of J, with respect to W.	29
2.12	Backpropagation error from output to input neuron, with one hidden layer	30
2.13	A two class problem presented with blue and red dots, randomly placed in a predefined area. The axes represent the position.	32

2.14	Illustration of a properly trained neural network output with 10 hidden neurons after 100 training iterations.	33
2.15	Illustration of a overfitted neural network output with 100 hidden neurons after 1000 training iterations	33
2.16	Convolutional network structure, with convolution layer, pooling layer and fully connected layers.	35
2.17	Edge detection with Laplacian filtering	36
2.18	All neurons along the depth are looking at the same region in the input.	37
2.19	Downsampling an image with max-pooling with a 2x2 filter	38
2.20	Transfer learning from the Inception net	39
3.1	Software and hardware components of the system	41
3.2	Cameras used by Statens Vegvesen, Axis P1346	42
3.3	Mobotix Allround Dual M15/M16	43
3.4	Tesla P100 video card	43
3.5	The flow chart of the program. The left module is acquiring traffic data, while the right module is processing the data into useful information.	46
3.6	Methods used in <i>system setup</i>	47
3.7	Region of interest defined by the corner the lanes	47
3.8	Divider line, in red, determines where the vehicles are counted	48
3.9	flow in the the detection module	49
3.10	Unprocessed foreground mask	50
3.11	Foreground mask processed by morphological operations	50
3.12	Background subtraction without(left) and with shadow(right)	51
3.13	Detected blobs are marked with a green square	51
3.14	The lanes becomes apparent by tracking the movement in the foreground mask	52
3.15	Detection and tracking scenarios	53
3.16	Flow chart for the Kalman filter	55

3.17	Vehicle moving from upper left corner to right bottom corner. Red square is prediction, x is measurements.	57
3.18	Background blob tracked with Kalman filter. The dots highlights trajectory points. Red is measured position, and white is predicted position.	58
3.19	Flow chart of the <i>Classify module</i> , both with and without training .	59
3.20	Convolutional neural network with fixed hyperparameters, filter-depth, filtersize and connected neurons.	61
3.21	Information from the database is shown in a Graphical interface . .	62
4.1	Original data, directly cropped from the object detection module . .	67
4.2	Data augmentation of the original dataset in figure 4.1	68
4.3	Experimenting on different architectures of the neural network . . .	71
4.4	Accuracy of neural networks plotted vs epochs. The colors represent neural network models with different architectures.	72
4.5	Neural network architectures with good accuracy. The accuracy is plotted vs training iterations.	73
4.6	The architecture of the neural network with the highest accuracy score	74
4.7	Confusion matrix after classifying the <i>test dataset</i> . Achieved 99.0% accuracy	75
4.8	Confusion matrix after classifying the test set. 99% accuracy	76
4.9	Misclassifications of the dataset	77
4.10	The camera may be mounted anywhere along the x-axis, as long as the green area is observable.	78
4.11	Recording destinations. 1 is E18 Maritim, 2 is E16 Sandvika, and 3 is E18 Sandvika	79
4.12	Hits detected	80
4.13	False hits detected	80
4.14	The system is adjusted according to the location. The black boxes are blocking personal properties, due to privacy regulation.	83
4.15	Vectors between vehicle coordinates	84

4.16	Distance between measured positions based on frames per second	85
4.17	Distance between measured positions based on frames per second	86
4.18	Measured coordinates in pixels and the corresponding predictions	88
4.19	The norm of the state covariance matrix plotted vs time	88
4.20	Hit rate at different <i>time of day</i>	90
4.21	Hit rate at different <i>time of day</i> , parallel view of figure 4.25	90
4.22	Shadow projection caused by the sun	91
4.23	Hit rate during cloudy days, plotted between 08:00 - 20:00	92
4.24	Hit rate during cloudy days, parallel view of figure 4.23	93
4.25	Hit rates for the three selected locations	94
4.26	Multiple lanes	96
4.27	Vehicle velocity at E18 Sandvika.	99
4.28	Vehicle velocity at E18 Maritim.	100
4.29	Vehicle velocity at E18 Fiskvollbukta.	100
4.30	Vehicle density at E18 Sandvika	101
4.31	Vehicle density at E18 Maritim	101
4.32	Vehicle density at E18 Fiskvollbukta	102
4.33	Congestion at E18 Sandvika	103
4.34	Congestion at E18 Maritim	104
4.35	Congestion at E18 Fiskvollbukta	104
4.36	Confusion matrix of the classified dataset	106
4.37	normalized confusion matrix of the classified dataset	107
4.38	Frequency of the three classes at E18 Sandvika in intervals of 20 minutes (no pedestrians were detected)	108
4.39	Frequency of the three classes at E18 Maritim in intervals of 20 minutes (no pedestrians were detected)	109
4.40	Frequency of the three classes at E18 Fiskvollbukta in intervals of 20 minutes (no pedestrians were detected)	109
5.1	Drop in hit rate,	113
C.1	138

C.2	139
-----	-------	-----

List of Tables

3.1	Hardware components table	42
4.1	Gathered data at given locations	66
4.2	Detection table	79
4.3	Recordings from Sandvika E18	89
4.4	Recordings from Sandvika E16	92
4.5	Recordings from Sandvika E16	94
4.6	Recordings with multiple lanes	96
A.1	Table of implemented software	131

Appendices

Appendix A

Python libraries

This section presents software components that has been implemented in the system.

	Image processing	Mathematical operations	Machine learning	GUI
Python package	Open cv skimage PIL	Numpy Pandas matplotlib	Tensorflow	PyQt

Table A.1: Table of implemented software

Python package **OpenCV**[7] , **skimage** [7] and **PIL**[] are used for different image processing tasks. Comprehensive information is attached in 4.40.

OpenCV is open source and open for use both private and for commercial use. scikit-image is a collection of algorithms for image processing. It is available free of charge and free of restriction.

PIL, Python Imaging Library, is a library with image processing capabilities. This library supports many file formats, and provides powerful image processing and graphics capabilities [34].

Tensorflow is an open-source software library for machine learning. It's a computational graph, and may be seen as a program consisting of two discrete sections :

- Building the computational graph.
- Running the computational graph.

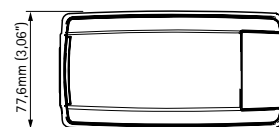
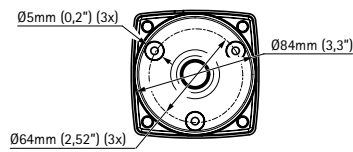
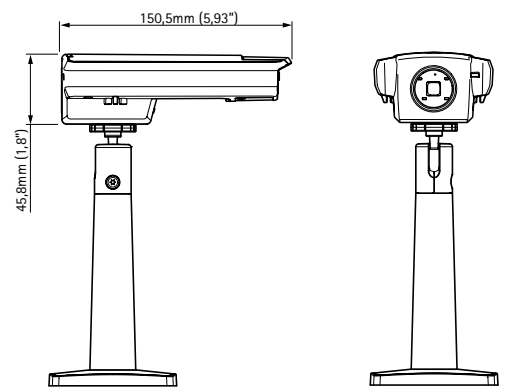
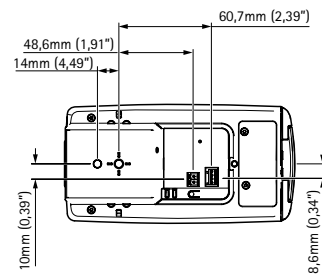
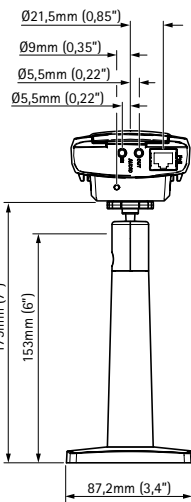
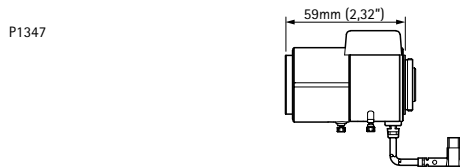
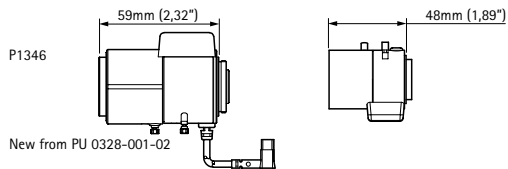
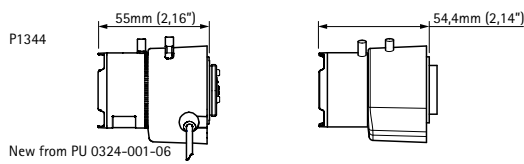
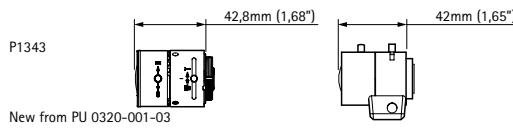
A series of operations or functions are arranged into graph of nodes. Each node takes an input and produces an output. Computational graph's are a technique for calculating derivatives quickly. It can make training of neural networks as much as ten million times faster, relative to a naive implementation . The computational graph is built up of many small units, where each unit is responsible for computing an output based on the inputs to the unit and the gradient of the output with respect to the inputs.[44]

SSH file transfer is a program to transfer files between the local computer and the Unix server [17].

PyQt is a GUI toolkit. Qt also includes Qt Designer, a graphical user interface designer. PyQt is able to generate Python code from Qt Designer. It is also possible to add new GUI controls written in Python to Qt Designer [37].

Appendix B

Datasheets



AXIS P13 Network Camera

1 FEB, 2012





Dual Thermal Technology

Automatic Temperature Alarms • 6MP Moonlight • Reliable





Temperature Events

Thermal radiometry (TR) models M15, S15 and S15 PTMount from MOBOTIX generate automatic alarms, defined by temperature limits or temperature ranges, which is vital to detect potential fire or heat sources. Up to 20 different temperature triggers can be

defined at the same time within so-called TR (Thermal Radiometry) windows or the whole sensor image can be used over the temperature range of -40 to +550 °C. In this way critical situations can be analyzed in the control room in order to plan the next steps for effective fire prevention. Critical assets like emergency generators, wind turbines or radio stations can be cost-effectively maintained and tested remotely. MOBOTIX thermal dual camera systems offer thermal overlay to localize so-called hot spots in the visual image to prevent larger damage. The standard Power-over-Ethernet (PoE) compatibility and the extremely low power consumption of only 6 watts allows operation of MOBOTIX thermal camera systems in every situation.

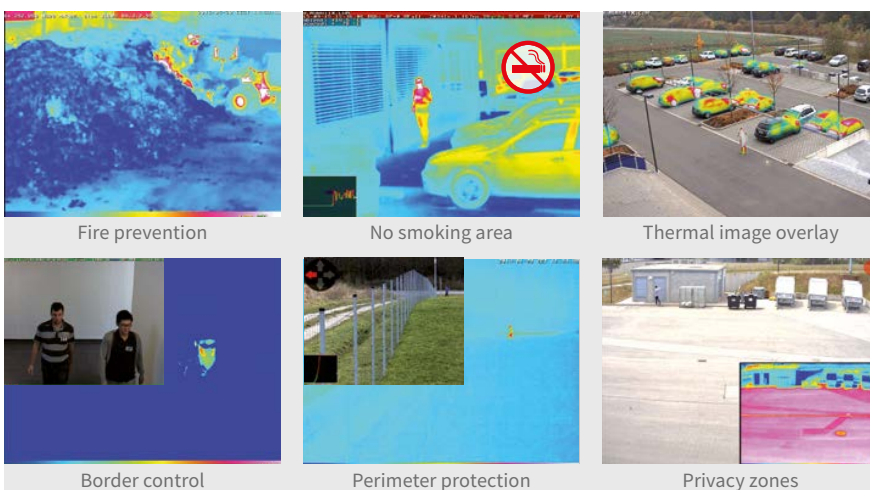
Cost Effective Perimeter Solution

Only one thermal MOBOTIX camera is required to protect a huge outdoor area without the need of any additional illumination – even in complete darkness. The combination of thermal & video sensors and intelligent software based motion detection (MxActivitySensor) are perfectly suited to efficiently cover wide perimeter situations without any secondary equipment like conventional light or infrared illumination.

Respecting Privacy

The detected thermal profile of a thermal camera shows no identifiable details for identification of persons and can therefore guarantee privacy. As soon as an object is moving into the relevant surveillance area, MOBOTIX dual camera system can automatically switch from thermal sensor to the optical sensor, producing visible high resolution video. This unique MOBOTIX feature combines two aspects, respecting the privacy aspect and at the same time optimal video surveillance.

- **Thermal Resolution**
Equivalent to 0.05 °C, range -40 to +550 °C
- **Temperature Alarms**
Up to 20 different automatic temperature events
- **Hot Spot Analysis**
With thermal image overlay
- **Motion Detection**
In complete darkness with thermal image and MxActivitySensor
- **Power**
Lowest energy bill, < 6W, standard PoE
- **Robust and Nearly Maintenance-Free**
Weatherproof, IP66, -30 to +60 °C, MTBF > 9 years



EN_09/16

MOBOTIX AG
Kaiserstrasse
D-67722 Langmeil
Tel.: +49 6302 9816-103
Fax: +49 6302 9816-190
sales@mobotix.com
www.mobotix.com

Appendix C

Neural network models

Models (connected area, filter size, filter depth)					
Model_1	Model_2	Model_3	Model_4	Model_5	
128x128x3	128x128x3	128x128x3	128x128x3	128x128x3	Input
				128x128,5x5,96	Conv1
max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	
64x64, 5x5,16	64x64, 5x5,32	64x64,5x5, 32	64x64,5x5,96	64x64,5x5,128	Conv2
64x64, 5x5,32	64x64, 5x5,32				Conv3
64x64, 5x5,32	64x64, 5x5,64				Conv4
64x64, 5x5,64					Conv5
64x64, 5x5,128					Conv6
64x64, 5x5,128					Conv7
64x64, 5x5,192					Conv8
64x64, 5x5,192					Conv9
max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	
32x32,5x5,256	32x32, 5x5,64	32x32,5x5, 64	32x32,5x5, 256	32x32,5x5, 128	conv10
32x32,5x5,256					conv11
32x32,5x5,256					conv12
32x32,5x5,256					conv13
32x32,5x5,256					conv14
max pool 2x2		max pool 2x2	max pool 2x2	max pool 2x2	
16x16,5x5,512		16x16,5x5,64	16x16,5x5,256	16x16,5x5,128	conv15
16x16,5x5,512					conv16
16x16,5x5,512					conv17
16x16,5x5,512					conv18
				max pool 2x2	
				8x8,5x5,256	conv19
max pool 2x2		max pool 2x2	max pool 2x2	max pool 2x2	
8x8,5x5,512		8x8,5x5,64	8x8,5x5,512	4x4,5x5,512	conv20
1x4048	1x4048	1x4048	1x4048	1x4048	
1x2024	1x2024	1x2024	1x2024	1x2024	
1x3	1x3	1x3	1x3	1x3	

Figure C.1

Chapter 3

Model_6	Model_7	Model_8	Model_9	Model_10	
128x128x3	128x128x3	128x128x3	128x128x3	128x128x3	Input
			128x128,8x8,32	128x128,5x5,16	Conv1
max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	
64x64, 8x8,96	64x64, 8x8,32	64x64, 8x8,64 64x64, 8x8,64	64x64,3x3,64	64x64,3x3,32 64x64,3x3,32 64x64,3x3,64 64x64,3x3,64 64x64,3x3,128 64x64,3x3,128 64x64,3x3,192 64x64,3x3,192	Conv2 Conv3 Conv4 Conv5 Conv6 Conv7 Conv8 Conv9
max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	
32x32,5x5,256	32x32, 3x3,64	32x32,5x5, 64	32x32,3x3, 128	32x32,5x3, 256 32x32,5x3, 256	conv10 conv11
				32x32,5x3, 256 32x32,5x3, 256	conv12 conv13 conv14
max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	
16x16,3x3,256	16x16,3x3,64	16x16,5x5,64	16x16,3x3,128	16x16,3x3,512 16x16,3x3,512 16x16,3x3,512 16x16,3x3,512	conv15 conv16 conv17 conv18
			max pool 2x2	max pool 2x2	
			8x8,3x3,256	8x8,3x3,256	conv19
max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	max pool 2x2	
8x8,3x3,512 1x4048 1x2024 1x3	8x8,3x3,64 1x4048 1x2024 1x3	8x8,5x5,64 1x4048 1x2024 1x3	4x4,3x3,512 1x4048 1x2024 1x3	4x4,3x3,512 1x4048 1x2024 1x3	conv20

Figure C.2

Recordings

Destinations:	Web:
E18 Fiskvollbukta	http://www.vegvesen.no/trafikkinformasjon/Reiseinformasjon/Trafikkmeldinger/Webkamera?kamera=329272&video=true&zoom=9
E16 Sandvika	http://www.vegvesen.no/Trafikkinformasjon/Reiseinformasjon/Trafikkmeldinger/Webkamera?kamera=602663&video=true
E18 Maritim	http://www.vegvesen.no/Trafikkinformasjon/Reiseinformasjon/Trafikkmeldinger/Webkamera?kamera=424667&video=true
E18 Sandvika	http://www.vegvesen.no/Trafikkinformasjon/Reiseinformasjon/Trafikkmeldinger/Webkamera?kamera=418039&video=true

Appendix D

Source code

Source code: 

Systemmain

The main file of the system, including the GUI file. The use is explained in the user manual.

CNN

The convolutional neural network source code. The use is explained in the user manual.

Kalmanfilter

The implemented Kalman filter, used by the main file.

Tensorflowdatasetbatches

The input pipeline. Input parameter is folder destination of the dataset folder. Used by the CNN file. Make sure to follow the user manual.

Appendix E

User manual

The Python files are embedded as python.zip

Using the surveillance system

1. Download the python.zip file
2. Download PyCharm from
<https://www.jetbrains.com/pycharm/>
3. Download Anaconda 4.4.0
<https://www.continuum.io/downloads>
Python 3.6 64 bit-installer
4. Download all necessary Python packages. Open cmd and write pip install followed by (if several conda environments are installed, write "activate" followed by desired Python version, e.g py35):
numpy, scikit-image, opencv-python, PyQt5
tensorflow, pandas_ml, PIL, pickle
If any of those fails to download, replace *pip install* with *conda install*.
5. Open PyCharm

Press file, default settings, project interpreter

In *Project interpreter* choose the desired Python version from Anaconda (python.exe)

6. Open the *system_mainfileinPycharm* and press *Run*

optional: The neural network model may be changed in line 687, by choosing the default directory of the model

optional: The videofile may be changed in line 659 by choosing the default directory.

optional: Modify the vector space in line 218 and 219.

7. Press *Start* in the GUI.

Training a neural network

1. Follow the 4 first steps from *Using the surveillance system*

2. Make a new *main* folder

Make a new *class* folder for each desired class, inside the *main* folder, and add the training data

Make a new folder inside the *class* folders, and put the test data inside

3. Open the Python file CNN in pycharm

change log directory(LOGDIR), debug directory(DEBUGDIR) and desired model directory(FILENAME) in line 22,23 and 24 in line 22,23,24

4. Choose the desired input in line 119

shape of input: [[filter depth],[filter size],[pooling]]

5. Start the program and read the results from LOGDIR.

Graphs can be found in tensorflow by following

Results from testing in Tensorboard

1. open cmd
2. Write cd followed by the *directory* of the FILENAME folder
3. Write "tensorboard -logdir = *directory*"
4. Copy the ip-address from the cmd into a Google Chrome browser