TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Sander Aasaväli 176771 IAPM

NEAR REAL-TIME INTERSECTION CONGESTION PREDICTION USING CONVOLUTIONAL NEURAL NETWORKS

Master's thesis

Supervisor: Martin Rebane
MSc

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Sander Aasaväli 176771 IAPM

REAALAJALÄHEDANE RISTMIKE KINNISÕITMISTE ENNUSTAMINE KASUTADES KONVOLUTSIOONILISI NÄRVIVÕRKE

Magistritöö

Juhendaja: Martin Rebane
MSc

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Sander Aasaväli

07.05.2019

Abstract

The main subject for this thesis is to develop a system, which predicts traffic blockages in near real-time with convolutional neural network from camera stream on the intersection. When a blockage is predicted, the system sends a signal to the traffic light controller so it will stop the green light cycle proactively. This should reduce traffic congestions, because the green light will not go to waste. This thesis focuses only on the Sõpruse-Tammsaare intersection, but the system can later be expanded.

Previously Epp Mauring developed a model in TensorFlow. In this thesis the model was converted to PyTorch. A whole new dataset for training the model was obtained, because Tondi intersection was rebuilt between this and Epp Mauring's thesis. The prediction time was increased to ten seconds. The architecture of the model was also improved, which in turn increased the accuracy. The final achieved accuracy was 83%.

Raspberry Pi 3 B+ was chosen as the hardware for the system. It is relatively cheap, but offers enough computation power and has the ability to run multiple tasks at once.

Although a thorough system design was developed, it could not be tested on the real intersection.

This thesis is written in English and is 52 pages long, including 6 chapters, 26 figures and 7 tables.

Annotatsioon

Reaalajalähedane ristmike kinnisõitmiste ennustamine kasutades konvolutsioonilisi närvivõrke

Töö põhieesmärgiks oli arendada süsteem, mis ennustab ristmiku kinnisõitmisi ette peaaegu reaalajas. Ennustamiseks kasutatakse konvolutsioonilisi närvivõrke, kuhu antakse sisendiks ristmikul olevast kaamera voogedastusest välja lõigatud pilt. Kui mudel ennustab ristmiku kinnisõitmist, siis saadetakse signaal ristmiku foorikontrollerile, mis omakorda paneb koheselt rohelise tule vilkuma. See peaks vähendama liiklusummikuid, sest roheline tuli ei lähe teiselt suunalt tulevatel autodel raisku. See töö keskendub eelkõige Sõpruse-Tammsaare ristmikule.

Eelnevalt on Epp Mauring enda bakalaureusetöö raames arendanud närvivõrgu mudeli antud probleemi lahendamiseks kasutades TensorFlow raamistikku. Antud töö raames vahetati raamistikku ja lõplik mudel treeniti PyTorch raamistikuga. Kuna Tondi ristmik ehitati kahe töö vahel ümber, oli vaja ka uuesti koguda treening- ja testandmestik mudeli treenimiseks. Mudeli ennustamise aega suurendati neljalt sekundilt kümne sekundini. Samuti täiendati närvivõrgu arhitektuuri, mille tulemusena paranes ka mudeli täpsus. Lõplikuks täpsuseks saavutati 83%.

Töö raames töötati välja ka süsteemi riistvaraline pool. Mudeli käivitamiseks valiti välja Raspberry Pi 3 B+, sest see on suhteliselt odav, kuid suudab korraga paralleelselt jooksutada mitmeid programme.

Kahjuks süsteemi ristmikule siiski ei olnud võimalik paigaldada ja seetõttu ei olnud võimalik seda reaalajas testida.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 52 leheküljel, 6 peatükki, 26 joonist, 7 tabelit.

List of abbreviations and terms

Blocked intersection	The intersection is read as blocked, when at least half a car stops at the pedestrian crossing
CNN	Convolutional Neural Network
CPU	Central Processing Unit
Epoch	One cycle of training
F1 score	A value for model accuracy by it's recall and precision
Hyperparameters	Parameters in machine learning, which values are set before the learning process begins. For example learning rate, number of epochs, etc.
IoT	Internet of Things
IP	Internet Protocol
L2 regularization	Neural network regularization method – restrains the weights of the neurons
Learning rate	The rate in which the neural network neuron updates its weights
Near real-time	Refers to the time delay introduced in an automated process.
Neuron	Elementary unit of a neural network
OS	Operating System
Over-fitting	When a Machine Learning model highly adjusts to training data and the accuracy on test data is too low
RPi	Raspberry Pi
RTSP	Real Time Streaming Protocol
SSH	Secure Shell
SWA	Stochastic Weight Averaging
TalTech	Tallinn University of Technology
TCA	Tallinn City Administration
Test data	Data used to validate the model
Training data	Data used to train the model's weights
Traffic lights cycle	Total time necessary to complete a sequence of signalization for all movements
VDSL cable	Very-high-bit rate Digital Subscriber Line cable

Table of Contents

List of Figures

List of Tables

1 Introduction

Regulating intersection traffic in big cities can be a challenge. Even though traffic lights regulate intersections, they are not sufficient to guarantee an optimal traffic flow. Drivers do not always notice, when a section becomes full ahead of a green light and thus have to stop in the center of the intersection. As a consequence, when another direction gets a green light, that direction may not cross the intersection, because of the blocking car. A full traffic lights cycle may be wasted for that direction. It does not only affect other cars, the pedestrian crossings are blocked as well. If people try to cross the intersection between the cars, it is potentially dangerous to them if the driver suddenly starts moving.

Using machine learning to predict ahead of time is one way of solving the problem. The neural network model can predict whether the intersection gets blocked before the end green cycle. By shortening the green cycle for some seconds, the blocking could be prevented. If the prediction is correct and the blocking of the intersection is prevented, the other direction can cross the intersection successfully instead of having to wait with a green light without being able to move.

Previously Epp Mauring has collected some test/training data from a camera in Sõpruse and Tammsaare intersection [1]. In her thesis she stated that convolutional neural networks (CNN) are the best fit for this problem [1]. As the result of her thesis, she managed to get the algorithm to predict congestion 4 seconds ahead of time [1]. Epp Mauring's neural network test set accuracy was 80% [1].

In this thesis CNN will still be used, but the possibility of using sequence models will be researched. The main goal is to increase accuracy and prediction time of the current model. It isn't currently known, what is the best prediction time to predict the congestion. This will be analyzed. Also getting a higher accuracy can be challenging, because the quality of the stream from the intersection camera can vary [2].

Alternatively, there may arise a need to adapt a third party model. The need and possibilities will be researched in the thesis.

The output for the main goal will be a neural network model, that predicts blockage of the intersection before it happens. The trained model needs to be as lightweight as possible, because this is a real-time and time-critical problem. It also needs to have at least 85% accuracy and the false prediction rate should be balanced.

The secondary goal of the thesis will be to build an installable computer system that is able to communicate with a traffic light controller. The computer system needs to be able to access streams from the intersection cameras through a secure network. The system needs to read input and also write output to a relay.

This system runs the trained model near real-time. The hardware receives the camera stream through a network cable and outputs the probability of the blockage. Total time, in which the inference takes place, should be less than a second. If it takes longer, the efficiency of the prediction drops, because the cut of the green light will not be earlier than it was statically designed.

Thesis output will be dated and then installed in the real intersection. The number of congestions before and after the use of the new system will be compared. Tallinn City Administration (TCA) is supporting this research. Because of that, the outcome of the thesis is permitted to be tested on the Tammsaare and Sõpruse intersection.

The author thanks Tallinn City Council for granting the "Raestipendium" for this work, and Sulev Sirkel and Raimond Nõugast for giving a lot of input for the thesis.

2 Background

2.1 Intersections

The blockage of intersections is a world-wide problem and real-time solutions have not completely solved the problem. In Tallinn, most of the intersections have cameras, that stream video [3], which can be analysed. In this thesis, the stream is used to predict, whether the intersection gets blocked.

Traffic light cycle is the total time necessary to complete a sequence of signalization for all movements [4]. Normal length of the traffic light cycle could go from 30 seconds to 120 seconds [5]. In rush hour, the length of the traffic light cycle in Tallinn is 90 seconds and outside the rush hour it is 72 seconds [3]. This means, that in this 90 or 72 seconds, every traffic light on the intersections has a moment, where all the lights will light up (one at a time).

In this thesis, only one direction is taken into further research – Tammsaare from Õismäe side to Järvevana road (seen on figure 1). This direction causes the most blockages on the intersection [3]. Within this thesis, the author was granted access to three cameras on the intersection. As a result of this work, a final configuration used streams from two cameras to train the models – one that is facing the Järvevana direction (marked black on figure 1) and the other that faces the Õismäe direction (marked blue on figure 1).

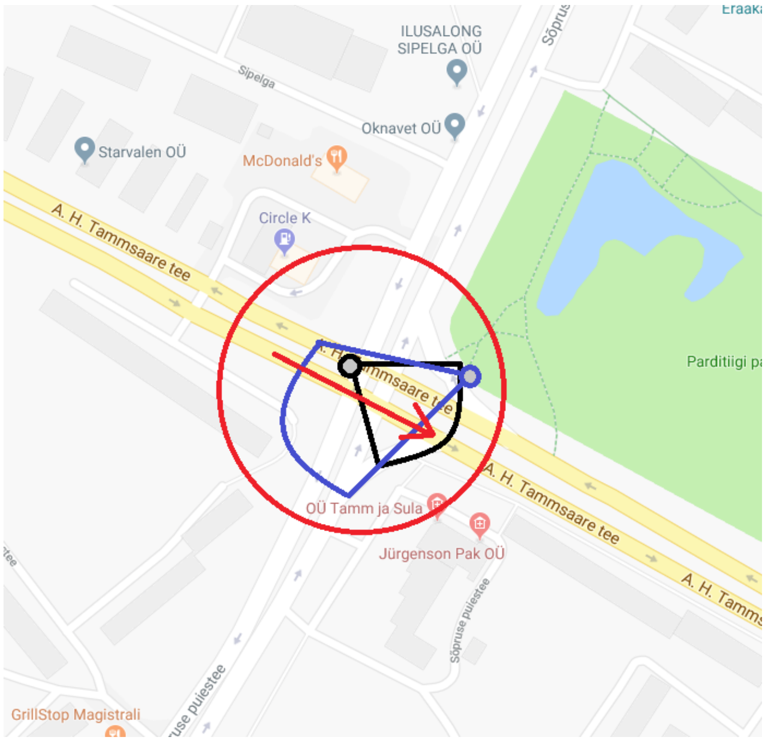

Figure 1. Intersection visualization and camera placements. The red arrow represents the traffic direction, which is optimized. The gray dots are cameras. The blue and black fields represent the field of view of the cameras. (Author's drawing. Base map: Google Maps)

Currently the traffic light cycle is defined to the second [3]. It means that if any traffic light duration is extended or shortened by even a second, it changes the traffic of the intersection remarkably [3], [5]. Also the cycle length of the whole intersection can not fluctuate, because this will cause synchronization issues with other intersections and may cause big blockages in rush hour [3], [5]. The length of every traffic light is calculated from a conflict matrix, which is constructed for every intersection [5] in Tallinn [3].

2.1.1 Current intersection technology

Currently there are cameras placed to the intersection so that every direction can be seen. The cameras are connected to a switch with a VDSL cable [3]. The switch and traffic light controller are in a weather proof box next to the intersection itself [3]. The cameras are also connected to the police network [3], which means it can be accessed outside, but as this is a very safe line, the authorization is not granted lightly. Due to the thesis, access was granted to the network for a static IP, so that test data could be collected from some of the cameras.

All the hardware have special relays in between. This is because when lighting strikes on the equipment, it will not damage everything that is connected [3]. This ensures that even if one intersection may malfunction, others will not go down with it.

2.2 Real-time traffic optimization

Static traffic light cycle configuration is not optimal in every situation [6]. Dynamic regularization not only saves time, but in the long run, it saves also fuel [6], [7], which is more eco-friendly. Models and simulations have been created to solve this issue [6]–[9], but a fully fitting solution to every situation is yet to be discovered.

One way of optimizing traffic lights is to create a multi-agent system [8], [9]. A solution, which would fit this kind of problem really well in theory, because it is a larger system, which is meant to be run at all times and situations [9]. In practice, this is a huge and expensive system to build into a city, which needs a lot of computational power [8], [9].

There is proposed an Internet of Things (IoT) solution as well [10]. Internet connection is needed between intersections, which try to apply green wave theory [10]. The green wave theory is when multiple intersections allow a continuous traffic flow in a given direction [11]. This solution seems promising, but enhancing most of the cars to communicate with the intersections, can get tricky. It seems like the multiagent systems also need some sort of communication between them, but that is not mentioned on the research paper yet.

Another way is to train a model that detects moving vehicles on the intersection [12]. If some lanes have high density of vehicles, then the lane is prioritized over others and the green light is switched on [12]. Vehicle movement detection had a very high accuracy (at daylight) [12], but the system itself is very complex and requires a lot of computing and decision making logic. Moving the same logic to different intersection is going to be troublesome, because of all the business logic in the intersection. Also as this takes control of the whole intersection, it may result in synchronization issues of adjacent intersections.

The IoT solution and the multiagent solution require large investments as well as years of research. These solutions are unlikely to go into production in the coming years. The solution provided in this thesis is easier to put into work, because it does not require any additional devices than the hardware the system runs on.

2.3 Convolutional neural networks

This work will use convolutional neural networks (CNN) which were introduced decades ago [13], [14] and has become the main technique to tackle many image recognition problems [13]–[18]. CNN works by extracting features from the input image [13]. Every subsequent convolutional layer extracts more complicated features [13].

For example the first layer may extract lines in a certain angle, the second layer extracts the interaction of those lines. Finally the last layer may extract some really complicated features or shapes like a flower or a human face. The feature extraction is done via kernels – a weight matrix with defined dimensions [13]. By moving the kernel over an image and doing computations, the spatial relationship between pixels are preserved and thus are the features extracted [13]. Since every pixels is taken into account, the more pixels the image has, the more computations are done. This means, that the smaller the image, the faster the model runs on it.

Thanks to the accessibility of huge number of annotated datasets, the CNN has progressed a lot [15], [19]. While it is better to train on larger datasets, the minimal necessary dataset to train a successful model doing image classification has decreased significantly [19], [20]. Depending on the given problem, that the model tries to solve. Also the efficiency compared to a normal neural network is higher [17], [20], [21], because it usually has parameters and connections [15].

Getting the right architecture for a neural network is crucial. Even if one layers is modified or removed, it may significantly decrease the accuracy of the model [15].

Processing high-resolution images takes longer time, but does not give any accuracy boost [22]. Running the model on a low computational hardware may get advantages from this.

2.3.1 Optical flow

An alternative would be to analyze optical flow. Optical flow is a method for detecting object inter-frame movements [1]. In our research group, Epp Mauring proved with her research, that it does not provide better results, but raises the complexity [1]. One of the reasons would be that the motion does not help by much. To understand and explain this, images were visually assessed in this work. While not rigidly provable, our explanation is that a lot of short-term motion information can be obtained only by reading a picture as well. For example, if cars have a longer cap between them, it means that they are moving with a higher speed, if a very short cap, then they might be standing still. This can be seen from figure 2.

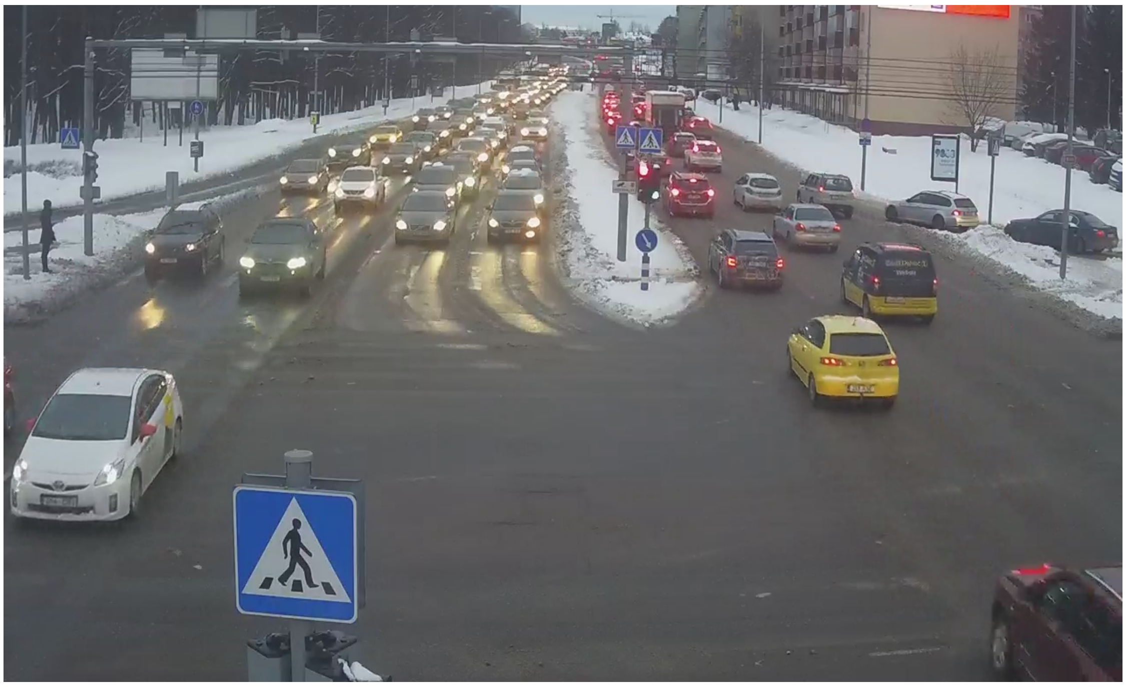
Figure 2. An example image that author used for a movement analysis to explain why optical flow did not have any effect

Our explanation is that the spacing between the cars also encodes also the speed of movement.

2.4 Deep learning platforms

Currently there are many frameworks for machine learning. Epp Mauring started previously her research on Tensorflow [1], [23], but this work uses PyTorch [24]. Both frameworks are documented well, but since PyTorch uses dynamic graph definition, it makes it easier to debug the network [25]. It is also integrated more to Python language, which makes it feel more native [25]. Debugging in Tensorflow requires more effort, because it requires special tools, which don't really debug python code, only the model itself [25].

The model needs to run on a smaller hardware at the end, but training on it is extremely slow [26], because the hardware that is meant only for inference is relatively weak [27]. In terms of porting the model to production hardware, both Tensorflow and PyTorch have sufficient tutorials and community behind it [28], [29].

2.5 Weather effects

Sometimes the picture cut out from the stream may be blurry or some weather effect has altered it. There are some ways of cleaning out the image. Unblurring an image can be done with a model [30]. There is also ways to remove shadow[31] or rain[32], [33].

To use these methods, the traffic image needs to be ran through these models developed in these papers. This can be considered, if enough training data can not be found and the model fails to classify the images on those circumstances. Before using these methods, the compute time needs to carefully considered. If the calculations take too long on the hardware, it is not beneficial.

3 Model development

This chapter describes the implementation of the model training, which consists of three parts:

1. Data processing to create datasets for training and validating the model
2. Framework conversion to PyTorch
3. Optimizing accuracy and prediction time of the model

3.1 Parsing training and test data

Getting the training and test data for machine learning project is a very important step, because the more diverse is a training (and of course the test set), the better results the neural network will give. Parsing this data in this project can be divided into multiple steps.

3.1.1 Camera stream parse

TCA has authorized a certain Tallinn University of Technology (TalTech) Internet Protocol (IP) address to access the camera streams of Sõpruse and Tammsaare intersection. The stream comes over Real Time Streaming Protocol (RTSP) – a network protocol designed for use in entertainment and communications systems [34]. Over the protocol, it is possible to record media streams [34].

A script was created to download record these streams into TalTech server. Ffmpeg is a software that takes RTSP stream URL as input and some additional parameters can be declared, like the duration of the stream to save and the format of the output file. To automate this process, crontab was used: the shell script is executed daily at rush hour.

Another alternative to parse streams, that was tested, is OpenRTSP, but it does not provide a comfortable interface to save streams for user-defined durations, so it does not work well with the setup of this project.

The default rtsp transport channel does not work for downloading streams - '-rtsp_transport tcp' needs to be added to the command. The basic script to parse stream can be seen on figure 3.

```
#!/bin/bash
now=$(date +"%d_%m_%Y")
ffmpeg -rtsp_transport tcp -i "$1" -c:v copy -t 3600
"/path/to/videos/$2/traffic-$now.mp4"
```

Figure 3. Stream parsing – base script

The basic script defines the basic flags to use ffmpeg. It accepts two parameters: first parameter is the stream rtsp url and the second one is identifier, which will be used as a folder name to save the video files into. The example base script usage can be found on figure 4.

```
#!/bin/bash
/path/to/stream_parse/parser.sh "rtsp://rtsp_url" "to_jarvevana"
```

Figure 4. Stream parsing – base script usage

The connection is not always stable and because of that, the ffmpeg script is interrupted. To ensure that we get a full hour of training data from each day, another script, that polls, whether ffmpeg is still running, was necessary. It uses ps aux and grep to check, whether ffmpeg is running and starts the stream parsing script again in case it is not. The example stream parse polling can be found on figure 5.

```
#!/bin/sh
ps_out=`ps aux | grep '[f]fmpeg' | grep 'to_jarvevana'`
if [ "$ps_out" = "" ];then
    /path/to/parsing/script.sh
fi
```

Figure 5. Stream parse polling

3.1.2 Stream slicing

The automated stream parser saves around 200MB chunks of videos (one hour of material per file). This can not be used for training data directly, so the data needs to be sliced and proper images need to be cut out for the neural network algorithm.

A python script using framework called moviepy was developed for this purpose. It provides an interface to extract subclip from videos by defining start time, end time and of course the file name itself.

Every video's name needs to be in exact format: {*first_cycle_start_in_seconds*}-{*label*}-{*date*}.mp4. The *first_cycle_start_in_seconds* is used to pinpoint the start of the first clip. If the *label* contains "NV", the cycle length is considered to be 72 seconds, otherwise it will be 90 seconds. The *date* is necessary to distinguish the clips afterwards.

Every clip generated, will have the name in format: {*annotation*}-{*date*}-{*id*}.mp4. *Annotation* is defaulted to unknown so the unannotated clips can be found quickly, if needed. The *date* is just for identification, and the *id* is for file name uniqueness. For every date, the *id* starts from number 1.

Configurable parameters in the script are the following:

- large_clips_path: Path where the slice-able video is located
- videos_path: Path where to save the sliced videos

3.1.3 Cutting images from sliced videos

The images were cut using opencv framework for python. There is a very straightforward function for doing so. By giving the method type (which can be by milliseconds or by frames) and the time unit of where to do the cut. For this project, cutting by milliseconds made more sense and it provided the results needed.

Before cutting the images out of the clips, all the sliced videos need to be renamed/annotated properly. Every clip needs to be in format {*start_of_red_cycle*}-{*annotation*}-{*date*}-{*id*}.mp4. The *annotation*, *date* and *id* are just for identification and are not as important in image cutting. The most important parameter in the name is *start_of_red_cycle*, because this is used to calculate the correct moment, when the image should be cut.

In the created script, there are some configurable parameters.

- videos_path: Path from where to read the video clips
- images_path: Path to where the script saves the cut images
- time_before_red_cycle: Time in milliseconds before the red cycle to cut out the image
- with_validation: Whether the validation mode is active

The with validation cuts out three images of each video with a 500 millisecond gap. When *time_before_red_cycle* is set to 0, it can be used to validate, whether the *start_of_red_cycle* in the video file name was correct. At least one of the three images needs to be yellow and one of them red. The third one can be either one.

3.1.4 Dataset parsing

The dataset parser, which constructs hdf5 files for model training and validation, was developed by Epp Mauring in her thesis [1]. In this thesis, another similar algorithm was created for multiple image input datasets to test different network configurations.

In these scripts, the image was resized. The final image needs to be 64 pixels on both width and height. It needs to be as small as possible for the model to run fast on low performing hardware [2.3]. If it would be too small, the traffic would not be distinguishable and the model may not do the correct conclusions.

In a hdf5 file, one has to define, which data types will be used for the saved values. In the previous parser, the data type used for pixel values was int8. The problem with it is, that the int8 allows numbers from range -128 to 127 [35], but pixel values go from 0 to 255. Because of the wrong data type, the colours of the loaded images from the created hdf5 files were distorted. When training and testing the model, it did not come out, because the model adjusted to the new colours. But when the model was exported and used on hardware, it did not give correct output on the original images, which were directly read into the program. After changing the data type for pixel values to uint8, which allows values from 0 to 255 [35], recreating datasets and retraining the models, the problem is fixed.

3.2 Converting the base model from TensorFlow to PyTorch

The existing Epp Mauring's prediction model [1] will be used as a baseline to assess the performance and quality of the new model. The baseline model was converted from Tensorflow neural network training code to PyTorch to allow comparison in directly comparable conditions and eliminate possible effects that may be caused by the framework.

3.2.1 Project structure

Unlike PyTorch, Tensorflow does not really encourage object oriented programming. Because of that, it is really easy to write all of the code in to the same file instead of encapsulating the classes in to separate files for better reading.

By converting the previous code into PyTorch, instead of one single file per training, there came three smaller files, which encapsulate the logic a lot better. One file for the network structure, one for dataset loading and one for training the actual model. Having this structure is also highly recommended by PyTorch documentation and examples. It makes the future development a lot easier and the code is a lot more maintainable.

3.2.2 Network

The network file contains a single class, which extends PyTorch's Model class, which is the base class for all neural network modules. In the constructor, one can define all the different layers of the network.

For this project five different layers were needed: two convolutional layers, two (max)pooling layers and one fully connected layer. A convolutional layer needs to know the number of inputs, the number of outputs and the kernel size. A (max)pooling layer only needs to know the kernel size. Both of those layers can receive some additional parameters like stride or padding. The last used layer is the fully connected layer, which needs only the number of inputs and outputs to construct the layer.

All the functional layers, like “sigmoid” or “softmax” for example, can be used as well, but these do not need to be pre-constructed in the network.

The initial Pytorch model after conversion from TensorFlow can be seen on figure Error: Reference source not found and the convolutional layer is described on figure 7.

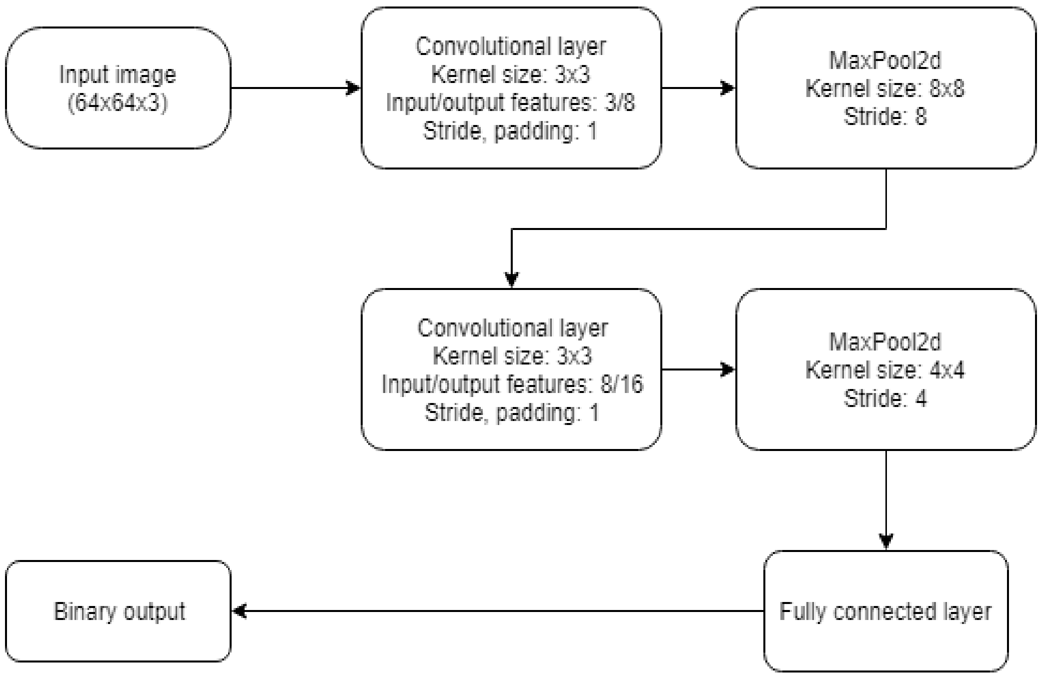

Figure 6. Model visualization after converting to PyTorch. Convolutional layer is described on figure 7.

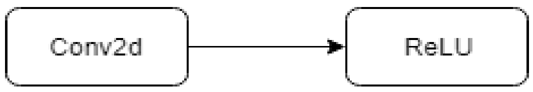

Figure 7. Initial model convolutional layer

3.2.3 Dataset

Parsing data files with PyTorch is preferably done in a separate class to enable the usage of batch processing[1]. The framework offers multiple ways to do this. One way is

1 Batch processing in machine learning means that a group of training samples will be used at once during the learning and optimization process [36]

extending the Dataset class. By extending the class, one has to overwrite two main methods. One, which returns the size of the whole dataset, and the other, which returns one certain item from the whole dataset.

Each item of the data set contains an image and the label, whether the intersection gets blocked. Parsing the label was straight-forward, but there was a problem with image parsing. Since Tensorflow uses images in shape of (width, height, channels) and PyTorch uses images in shape of (channels, width, height), some additional helper functions that reshaped the image structure were necessary. All the images from a data file were reshaped at parse time, so iterating speed would be faster.

DataLoader subclass is used to feed datasets into the model. There are some extra configurable parameters like minibatch size, number of workers and whether to shuffle the dataset.

3.2.4 Model training

Finally there is the main file, where training the model happens. In this file, the network and dataset files are used. In the start of the model training function, there is the loss function defined, network model object constructed and the optimization strategy defined. After these three main elements have been defined, the model is ready for training – processing training data for given iterations and updating the network parameters. After the training, model is validated against test data.

Hyperparameters are parameters that can be configured to optimize the neural network. Sometimes better results can be achieved by slightly changing only one parameter. Hyperparameters available for configuration are:

- lambda – regularization parameter (also known as L2 regularization parameter)
- learning_rate – learning rate of the neural network parameters
- minibatch_size – size of training data to be processed at once in every epoch
- num_epochs – number of optimization iterations done in the training

- Learning rate adjustment step – After every # of epochs, the learning rate is adjusted
- Learning rate adjustment multiplier

3.3 Increasing prediction time

Increasing prediction time could only be done by training the model with new images, which were cut out from the video at an earlier time step. One possible negative outcome was that by increasing the prediction time even by some seconds, the accuracy of the model would drop below our criteria. However this was not the case. By using the same model, the prediction even improved in some cases.

Since Tondi intersection, the next intersection after Sõpruse and Tammsaare, was rebuilt last year, this might have affected the traffic flow. Hence, it was decided to use only newly collected data to train the new model with increased prediction time. As previously Epp Mauring accessed only one camera and in this thesis, three were accessed. This opened up the possibility to train the model using three images from three different streams.

By using different images from different angles, it could be possible to increase the accuracy of prediction. As one of the three cameras was on an angle, that showed us only a glimpse of the direction that needs to be optimized, the stream was not used.

There were two ways to group the data: by test and training data and by input data. The input data groupings are:

- single image – only one image facing Tondi intersection
- horizontal image positioning – two images are sided horizontally, where the left one is facing Tondi intersection (as seen on figure 8)
- vertical image positioning – two images are sided vertically, where the upper one is facing Tondi intersection (as seen on figure 9)

The distribution of training and test data can be found on table 1.

Table 1. Distribution of training and test data.

	Training data	**Test data**
Blockage of intersection	100	22
Successful crossing	100	22

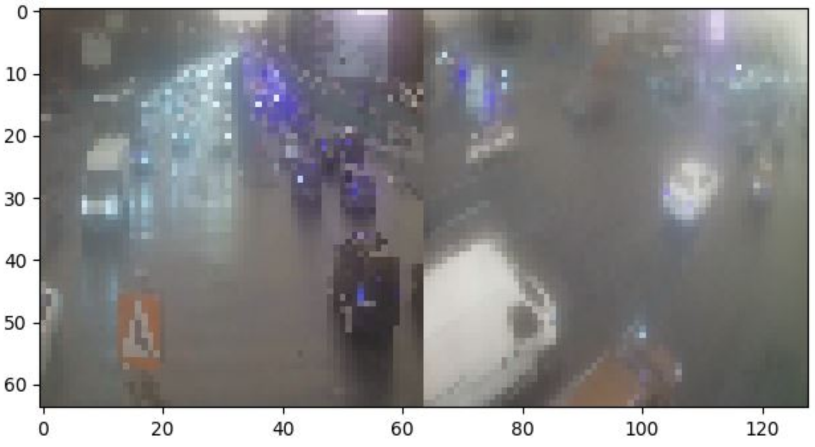
Figure 8. Example of horizontal image positioning.

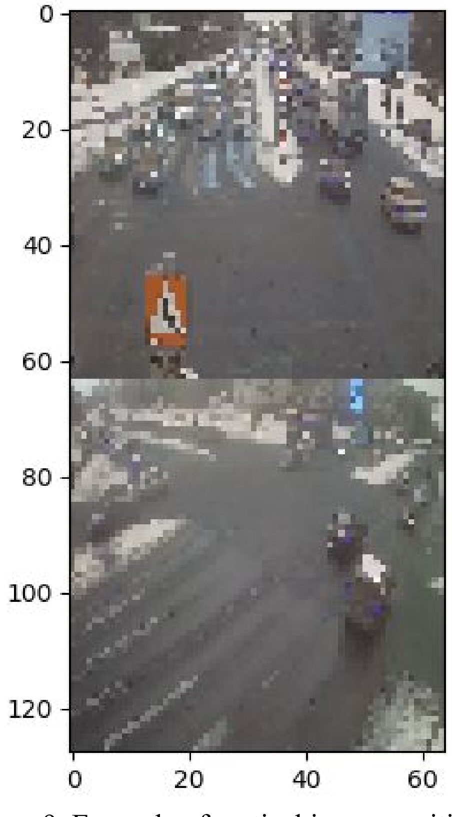
Figure 9. Example of vertical image positioning.

3.3.1 Accuracy

It is one important aspect to measure accuracy on both the training and test data set. Besides showing how likely a model is to output a correct answer, it could also indicate, whether a model may be over-fitting.

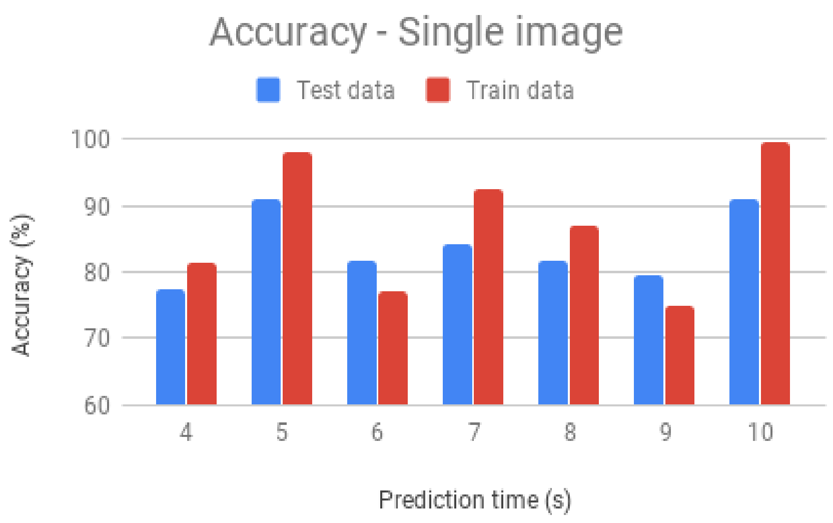

Figure 10. Accuracy – Single image

Analysing figure 10, predicting by a single image shows the best results predicting five and ten seconds ahead. Both the test data accuracy and training data accuracy are over 90%. Surprisingly, predicting four seconds ahead is a lot harder for the model than predicting 5 seconds. The then second model offers both better prediction time and better accuracy.

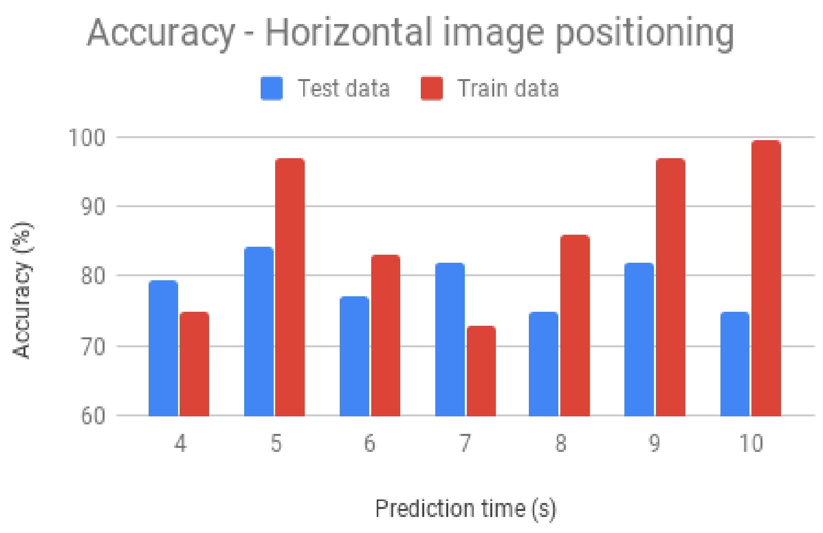

Figure 11. Accuracy – Horizontal image positioning

Analysing figure 11, it seems that putting two images side by side horizontally lowers the accuracy of the model significantly. In many cases (5, 8, 9 and 10 seconds), the model over-fits to training data.

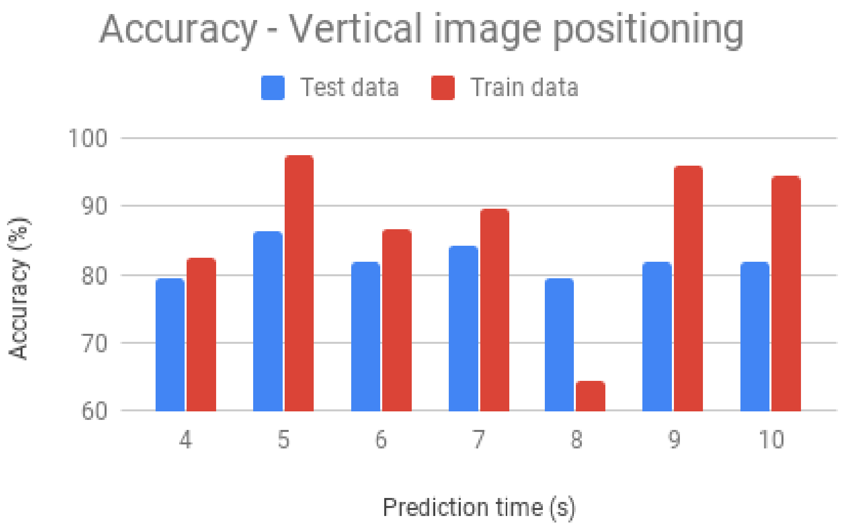

Figure 12. Accuracy – Vertical image positioning

By figure 12 it seems, that placing images vertically has almost 10% better on the test data than the horizontal approach. Over-fitting to the training set is still a problem, but the overall test data accuracy has improved.

Based on these figures, predicting with two images seems to be harder for the neural networks, a single image prediction by ten seconds is the most promising model yet.

3.3.2 False prediction ratio

Comparing the accuracy is important, but making decisions only based on that, could be misleading. The false prediction ratio needs to be satisfying as well. If the model mostly predicts blockage, it may have a great accuracy, but on a real intersection, it means that most of the times, it may cut the green traffic light shorter. This could instead make larger blockages.

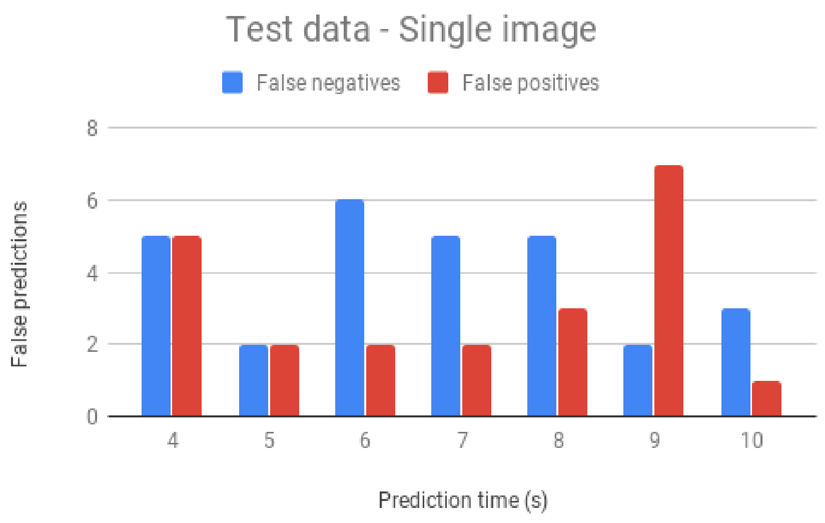

Figure 13. False predictions – Single image test data

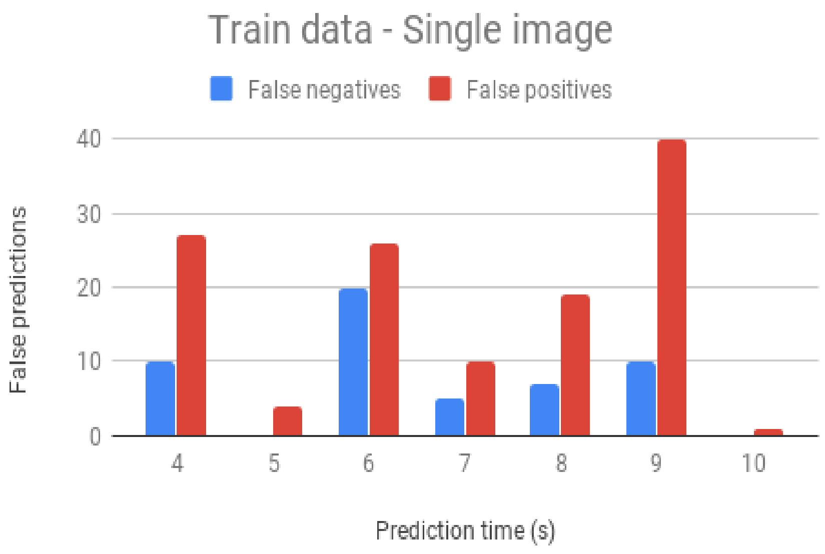

Figure 14. False predictions – Single image training data

From figures 13 and 14 can be read, that the worst model to use, is the nine second model. Both the training data and test data have a lot more false positives. The best models seem to be five and ten second model. They have the overall lowest false predictions. If the single image prediction would be chosen, the ten second model would be the best choice.

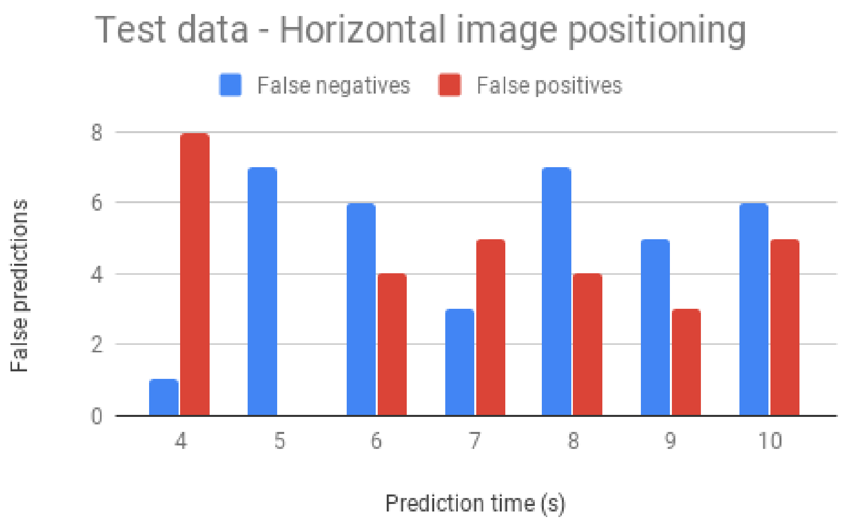

Figure 15. False predictions – Horizontal image positioning test data

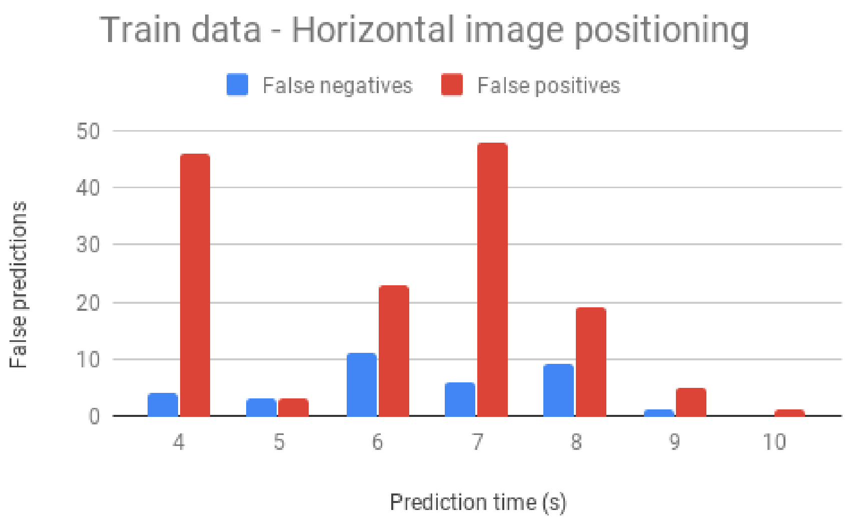

Figure 16. False predictions – Horizontal image positioning training data

From figures 15 and 16 it seems that predicting more time ahead has a better ratio. The four second model and the seven second model produce more false positives and opposingly, the five second model seems to produce more false negatives, but only on the test data. The best ratios are on the nine and ten second models.

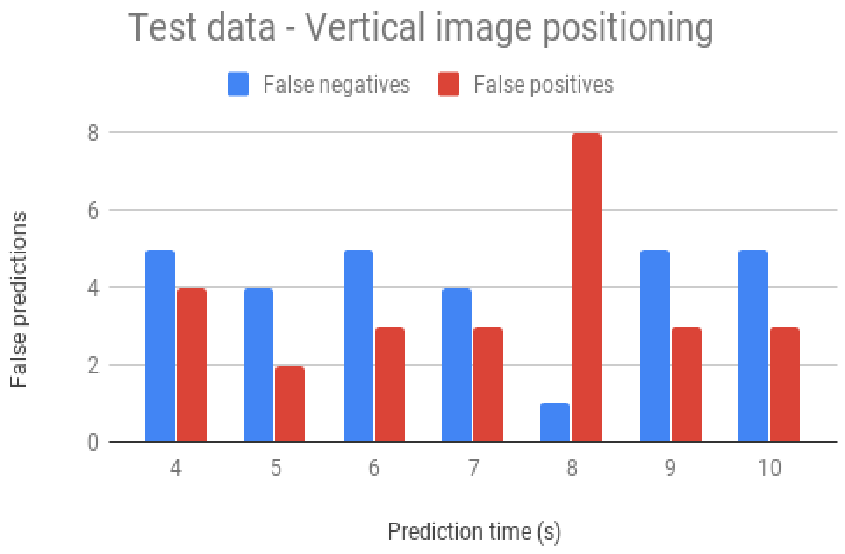

Figure 17. False predictions – Vertical image positioning test data

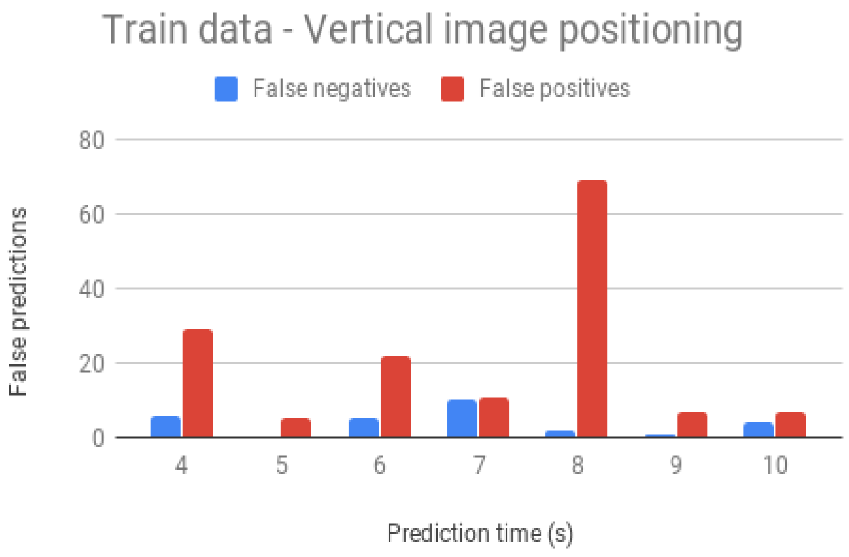

Figure 18. False predictions – Vertical image positioning training data

It can be seen from figure 17 that most of the models have a good ratio on the test data with the exception of eight second model. Figure 18 confirms that the eight second model is the worst performing model with the vertical image positioning. On training data, the four and six second model have a rather bad ratio as well. Since all the other models perform rather well, it is the most beneficial to use ten second prediction.

3.3.3 Conclusion

One of the goals was to increase prediction time to at least seven seconds. Both the single image and the vertical image positioning offer a great model for predicting ten seconds ahead. Both of the models have a good ratio, but the single image model has better overall accuracy on the test data. Another perk of choosing the single image model is that in production, only one image needs to be parsed from the stream. This loses the need to synchronize time between two stream parses. Parsing only one image also is cheaper for the Central Processing Unit (CPU) to compute.

Some of the training and test data were blurry, because of rain on the camera. The model still seems to correctly predict from it. This could be because the red break lights on the cars can still be seen through the blur. The weather effect removal proposed on the chapter 2.5 is not necessary.

The numerical results of figures 10-18 can be found from tables 2-4.

Table 2. Single image results

Single image	**Training data**			**Test data**		
Prediction time (s)	**Accuracy (%)**	**False positives**	**False negatives**	**Accuracy (%)**	**False positives**	**False negatives**
4	81.50	27	10	77.27	5	5
5	98.00	4	0	90.91	2	2
6	77.00	26	20	81.82	2	6
7	92.50	10	5	84.09	2	5
8	87.00	19	7	81.82	3	5
9	75.00	40	10	79.55	7	2
10	99.50	1	0	90.91	1	3

Table 3. Horizontal image positioning results

Horizontal image positioning	**Training data**			**Test data**		
Prediction time (s)	**Accuracy (%)**	**False positives**	**False negatives**	**Accuracy (%)**	**False positives**	**False negatives**
4	75.00	46	4	79.55	8	1
5	97.00	3	3	84.09	0	7
6	83.00	23	11	77.27	4	6
7	73.00	48	6	81.82	5	3
8	86.00	19	9	75.00	4	7
9	97.00	5	1	81.82	3	5
10	99.50	1	0	75.00	5	6

Table 4. Vertical image positioning results

Vertical image positioning	**Training data**			**Test data**		
Prediction time (s)	**Accuracy (%)**	**False positives**	**False negatives**	**Accuracy (%)**	**False positives**	**False negatives**
4	82.50	29	6	79.55	4	5
5	97.50	5	0	86.36	2	4
6	86.50	22	5	81.82	3	5
7	89.50	11	10	84.09	3	4
8	64.50	69	2	79.55	8	1
9	96.00	7	1	81.82	3	5
10	94.50	7	4	81.82	3	5

3.4 Neural network improvement

The network model can not be too complicated or it will not perform on the hardware. In this thesis, a batch2dnorm function was added as a second step to every convolutional layer. The first step is still conv2d and the last step is a ReLU.

Since it was necessary to process the concatenated images, the fully connected layer's number of input features needed to be increased for those training sessions. The number of output features from the first convolutional layer was increased from 8 to 16. When more data was added to the training and test set, it gave higher accuracy.

The final architecture can be seen on figure 19 and the convolutional layer is described on figure 20.

Another regulation method tried in the process of training was Stochastic Weight Averaging (SWA)[1]. It is basically an optimizer wrapper, which should average the weights of neurons at the end of the training [37]. By using SWA, the accuracy of the test set should improve at least a few percentages [37], but it did not work for our model. On the contrary, the accuracy dropped by eight percent.

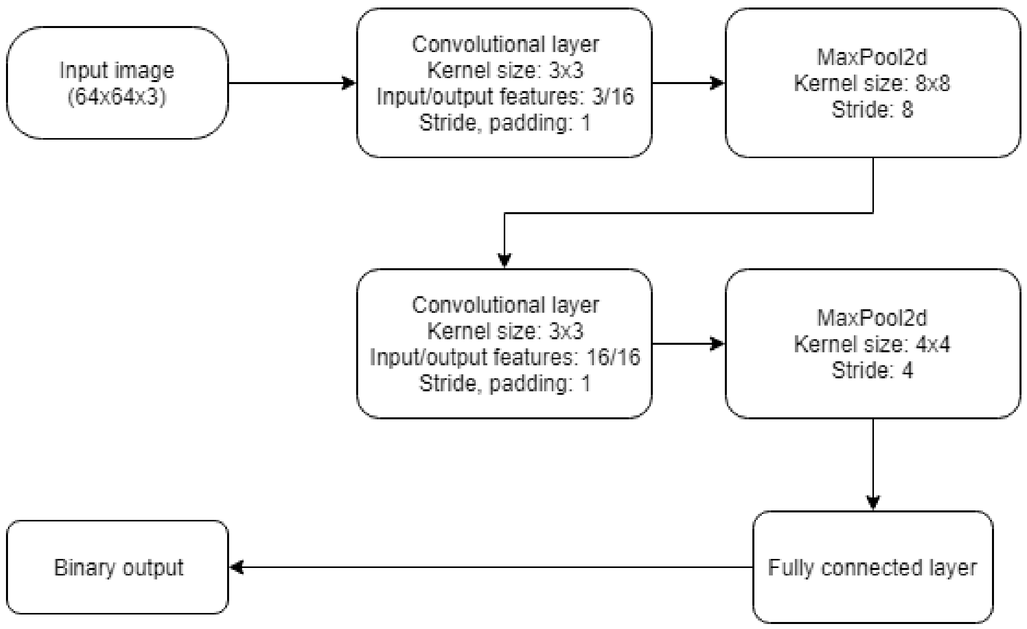

Figure 19. Model visualization after improvements from this thesis. Convolutional layer is described on figure 20.

1 The method is thoroughly described in PyTorch blog found on https://pytorch.org/blog/stochastic-weight-averaging-in-pytorch/

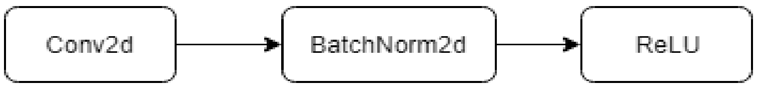

Figure 20. Improved model convolutional layer

Hyperparameter values used for the training of the final model can be seen in table 5.

Table 5. Final model hyperparameters.

Hyperparameter	**Value**
Learning rate	0.001
Number of epochs	108
Minibatch size	30
Lambda	0.0001
LR – adjustment step	10
LR – adjustment multiplier	0.8

3.5 Exporting model

The export of Pytorch model for python use is a straightforward process, but cannot be used in this case as Python is an interpreted language, which adds unnecessary overhead. Since the model will be used on a low performance hardware, a faster approach is necessary to achieve the goal of near real-time predicting. Torch can also be used in C++ [24], which should be a faster language overall. Exporting a model, that can be used in C++ torch library is more complicated. Pytorch's TorchScript offers a way to serialize the model from Python code [24]. This model can then be loaded in another environment, where there is no Python dependency [24].

Pytorch internally reuses some of the weights between neurons, if possible. This functionality created some problems for jit tracking. This was solved by deep copying layers using the copy library – it ensures, that all the weights are copied even tough there may be some identical weights on another neuron.

4 System development and integration

4.1 Board analysis

There are multiple choices for the hardware that will be used. Arduino, Raspberry Pi (RPi) and Jetson will be analyzed in this chapter.

4.1.1 Arduino

Arduino is simple board made for running one program at a time [38], [39]. Arduino is more suitable for repetitive tasks [38] like for example locking and unlocking a door. For more complex projects, multiple different boards may be needed [39]. The community is very active and there are many resources available to get started [39]. The price is relatively cheap for the separate parts.

4.1.2 Raspberry Pi

The RPi is a bit more complex than an Arduino and has the ability to run multiple programs at a time [38], but in terms of computing power they are in the same level [39]. It is more of a full-fledged computer [38] – it has all the components a computer needs already [39], so no extra assembly is needed. The community is slightly smaller than Arduinos [39], but more can be obtained for almost the same price. The Raspberry Pi 3 B+ is the preferred model, because it is the most powerful of the them.

4.1.3 Jetson

The Jetson Nano is well suited for deep learning and neural network tasks [40]. The specs are considerably more powerful compared to RPi, but the Jetson Nano does not have Wi-Fi support and attaching a third-party dongle is not an easy task [40]. While the board is more powerful, it is also larger and it requires even more space for the heat sink [40]. Jetson Nano uses Linux4Tegra for OS, which is a version of Ubuntu 18.04 [40]. The price of the board is almost three times of the RPi.

4.1.4 Conclusion

Arduino and RPi have the same price class and computational power, but since Arduino can not execute multiple tasks at the same time, it is not sufficient for this project. The Jetson Nano on the other hand is a very powerful board, but the price is higher as well. Since price is one of the criteria for the choice and RPi should have enough computational power for our needs, the choice will be the latter. If it turns out that the model needs to run on the device in under a second. It may be necessary to switch to the Jetson Nano instead.

4.2 System design

When placing the RPi on the intersection, it does not just start predicting by itself automagically. One naive method is to execute the algorithm on an interval – the inference happens after every given time. It needs some kind of input signal to know, where the correct time for prediction is.

The traffic light controller receives a signal and can forward it, when any traffic light lights up. Only the info, when a light lights up is known to the traffic light controller, not the countdown, when another light lights up. This signal will be forwarded to an input relay, which is in turn connected to the RPi pins. The pins can be read in a script that was developed in this thesis.

Getting a signal from the controller is a more reliable plan than a simple interval, because it is more prone to different anomalies. Sometimes some lights stay on longer or the cycle length is changed.

To be more immune to cycle length changes, there is a dynamic adjustment solution available that all the pedestrian traffic lights with counters use. The length of the cycle is measured every time the cycle starts. If the cycle length has stayed the same for three times, the length defined in RPi will be updated and further calculations for the inference start will be done with the new number.

If an active signal is received through the input relay, a timeout will be used to execute the inference script given seconds before the red light. Since the RPi is connected to the police network, it can directly access the RTSP stream to download the image of the intersection. With ffmpeg the image can be downloaded with correct dimensions.

Once the image has been downloaded, the model will be ran on it immediately. If the model does not predict congestion, nothing is done. If the opposite happens, a brief signal is sent through the RPi pins to the output relay, which in turn is connected to the traffic light controller's detector. As a result the green light is cut shorter. A simplified circuit can be seen on figure 21.

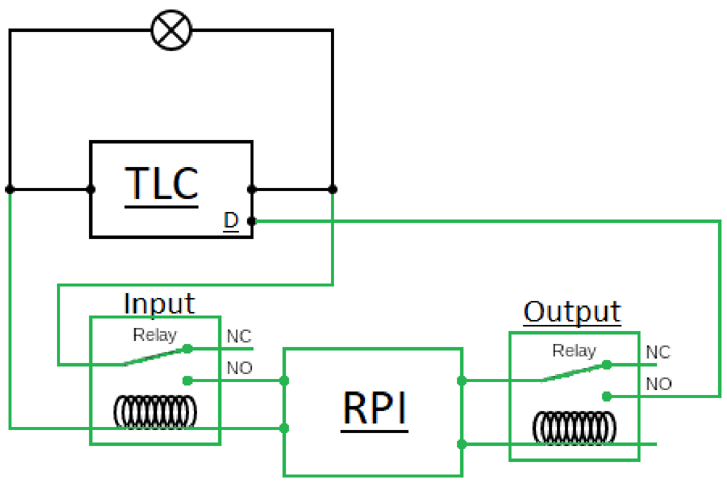

Figure 21. System design. TLC – Traffic Light Controller, RPI – Raspberry Pi, D – detector. Components and wiring marked with green colour are added to the existing system.

Overall there are two relays connected to the RPi – one for input and one for output. There is also the power supply and the network cable. RPi connected to the input and output relays can be found on figure 22.

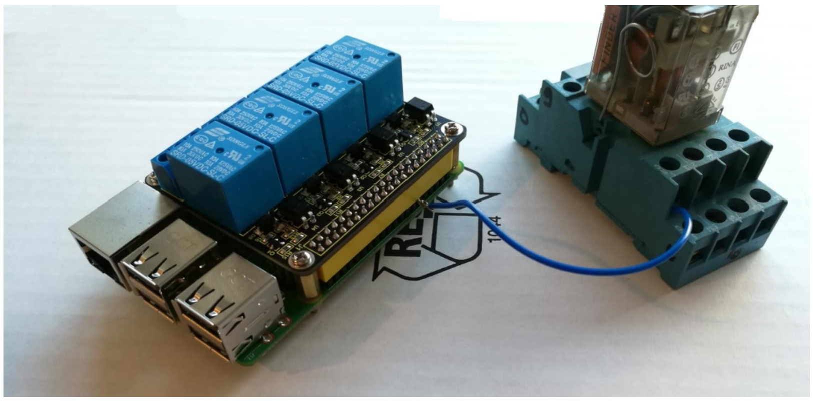
Figure 22. RPi connected with input and output relays. RPi with output relay on top of it on the left and input relay directly connected to the Raspberry's pin on the right. Author's image.

4.3 Setup of Raspberry Pi

For the operating system, Raspbian Stretch Lite was used. It is the most minimal Operating System (OS), that RPi offers. It has is terminal based (no desktop) and has less applications installed at the start. From the user perspective, once the command line is familiar, it is also faster.

The first things that should be done after OS installation are wifi configuration, default password change (for better security over Secure Shell (SSH) connection) and the SSH configuration. Ssh configuration opens the opportunity to connect to the RPi from other devices over network, so further configuration can be done remotely and it does not require a separate screen and a keyboard for RPi.

4.3.1 Pytorch build

The pytorch installation tutorial[1] by Amrit Das has been followed in this thesis and the steps described in this subsection are following it closely [41].

Once remote connection is established, Pytorch can be installed. Since RPi has an ARM type CPU and no pre-compiled PyTorch is available for this type of CPU, it has to be built itself. Before building, some main dependencies need to be installed with a command shown on figure 23. Python 3 is pre-installed on the OS.

1 https://medium.com/hardware-interfacing/how-to-install-pytorch-v4-0-on-raspberry-pi-3b-odroids-and-other-arm-based-devices-91d62f2933c7

```
sudo apt-get install libopenblas-dev libblas-dev m4 cmake cython python3-dev
python3-yaml python3-setuptools
```

Figure 23. Installing dependencies necessary for Pytorch build [41]

Pytorch installation requires more memory than the default Raspbian configuration allows and results in the “Virtual memory exhausted” error. Because of this, the swap memory size of RPi needs to be increased. This can be done by modifying /etc/dphys-swapfile file [42] and changing CONF_SWAPSIZE from 100 to 1024. After that the swapfile service needs to be restarted with commands shown on figure 24.

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start
```

Figure 24. Swapfile service restart [42]

Next step is to clone recursively pytorch’s git repository from https://github.com/pytorch/pytorch. After navigating to the cloned pytorch folder, some environment variables need to be set for correct configuration of the build. Finally the build command can be executed. This takes a long time and may fail in the middle, but executing the same command resumes from almost the same place and does not seem to cause errors in pytorch functionality. The final command validates all the files. All the commands necessary for the installation can be found on figure 25.

```
git clone --recursive https://github.com/pytorch/pytorch
cd pytorch
export NO_CUDA=1
export NO_DISTRIBUTED=1
export NO_MKLDNN=1
export NO_NNPACK=1
export NO_QNNPACK=1
sudo python3 setup.py build
sudo -E python3 setup.py install
```

Figure 25. Pytorch installation [41]

Finally the swapfile configuration needs to be returned to default. When the swap size is left to 1024, it may shorten the life of the SD card significantly. After the change is done to the configuration file, the service needs to be started with the commands shown on figure 24.

4.4 Inference

Exporting a C++ executable model was explained in chapter 3.5. For running the model on RPi, another C++ project was created. The best model found on the Python repository was added to the C++ project. There is a test script, which executes a built C++ file, that loads the model and runs a sample image through it and measures execution time. The biggest worry is that the model doesn't perform fast enough to give prediction on time, which makes the prevention less effective.

4.4.1 Image reading

Most of the solutions for image reading on the web suggested openCV for its simple usage, but as it is very heavy-weight framework, it just does not make sense to include it in the project just for reading capabilities. Torch needs the image in a slightly different format as well, from how it is usually presented. Usually the pixels are in shape of (width, height, channels), but Torch needs the image in shape of (channels, width, height).

Image processing library stb_image met the requirements for this project. It is lightweight and easy to use. All that was needed, was just the header file, which could be added to the project. The header file provided image load function, which returned a char pointer. With the pointer, all the pixel values could be read.

4.4.2 Filling tensor with pixel values

To read image into a tensor, an empty one needs to be created first in the correct shape, which would be (image_nr, channels, width, height). The image_nr comes from training with minibatches; in this file, it will always be size one. To read values into a tensor, an accessor[1] is necessary. Finally through the accessor, the pixel values could be read into the tensor, but in the reverse order. Worth mentioning is also, that the pixel values should be between zero and one, which means that all the values should be divided by 255. Example can be seen on figure 26.

1 An accessor is a wrapper for a tensor, providing a comfortable way to change multidimensional tensor values.

```
r = (int)image[(x + y*width) * comps1 + 0] / 255.0;
g = (int)image[(x + y*width) * comps1 + 1] / 255.0;
b = (int)image[(x + y*width) * comps1 + 2] / 255.0;
accessor[0][0][y][x] = b;
accessor[0][1][y][x] = g;
accessor[0][2][y][x] = r;
```

Figure 26. Reading pixels into a tensor

5 Validation

Validation was done on Sõpruse and Tammsaare intersection in collaboration with Tallinn City Administration. The validation was done theoretically.

5.1 Plan

Neural networks can be validated with proper test data, which in this case are video samples from the camera of the intersection. When those results are promising, then the next step is using neural network on a piece of hardware (In this case the RPi), on the intersection. Altough a real on-site testing was planned, unfortunately testing on the intersection could not be done, because the Tallinn Transportation Agency had some other important last minute tasks to attend to. Although this thesis is highly supported by them and the system will be implemented in the coming months.

The RPi's computation time needs to be measured, to ensure that the prediction is done fast enough to make a difference on the intersection. The faster the scripts runs, the more impact it may have.

Since the real impact of the thesis can not be seen, it is still possible to evaluate how many cars it may save from a "wasted" green light. This result can be achieved by counting the cars that tried to make the left turn to TalTech direction with and without the intersection blockage. From that, we can calculate the average number of cars per some period that the solution provided from this thesis could have saved.

5.2 Model validation

One of the main goal was to improve the prediction time. This was achieved successfully – the former four second prediction was increased to ten seconds. The whole sequence from four seconds to ten seconds were tested and the accuracy change was not linear. The figures can and result interpretation can be seen in chapter 3.3.

The six to nine second predictions are harder to learn for model than the ten second prediction. This may be because it is hard to distinguish the cars that were waiting from last cycle and the cars that just arrived there. This draws out more clearly by predicting ten seconds ahead.

Next the model was tested against a whole new dataset – freshly gathered clips and old unused clips. It is important to note that none of those clips were used in model's training or test set. There were in total 1182 clips. The model's accuracy was 72% on the new test set. This could be because the new clips had a very sunny weather and the network did not have this kind of data before. After re-constructing and updating the training and test datasets, the accuracy on 953 clips was slightly over 83%. The distribution of improved training and test data can be seen from table 6. The f1 score on the training set was 0.99 and on the test set 0.82.

Table 6. Distribution of improved training and test data for ten second prediction.

	Training data	**Test data**
Blockage of intersection	195	23
Successful crossing	195	23

5.3 Computation time

The computation time can be measured with *time* function on RPi. It outputs three numerical values:

- Real – Elapsed time from the start of the call to the finish [43]
- User – Total CPU time spent in user mode [43]
- Sys – CPU time spent in kernel mode [43]

The "Real" parameter is the one, that gives us the most information.

Table 7. Computation time by model

Model type	Real	User	Sys
Single image	~0.640	~1.674	~0.030
Concatenated image	~0.763	~1.791	~0.040

From table 7 we can conclude that both model types match our criteria of running under a second. Running the single image model has a slight advantage over the concatenated image model.

5.4 Thesis potential impact

To analyze the potential impact properly, all work days must be considered, because the number of cars blocked from crossing the intersection could vary from the intensity of the traffic. The footage used for this potential impact analysis was collected 29.04.2019-03.05.2019 from 17:00 to 18:00. At this time the rush hour has it's peak and the traffic is the most intense.

A total of 198 cycles were analyzed on the given time period. 57 of them had a blockage. Most of those blockages only disturbed the pedestrian crossing, but there were 5 larger ones that halted car movement as well. On a successful crossing, the average of 19 cars cross the intersection. On a large blockage the average drops to 11. It means that on the average, 8 cars would not have had to wait for another cycle. Per week it makes a total of 40 cars, that would have not have to wait for another cycle. If the prediction performs well on the intersection, it means that in four weeks, which is roughly a month, an average of 160 cars could avoid waiting for another cycle because of the blockage.

Taken that a car takes roughly five meters with spacings in traffic whilst standing still, it means that the eight cars that did not cross the intersection, take away 40 meters of space from the intersection. 20 meters if there are two lanes like in Tammsaare-Sõpruse intersection. As traffic density in rush hour is already high, the 20 meters make a difference, when new cars are arriving from the previous intersection.

On 29.04.2019 there was car crash on intersection. Although no blockages happened on that day, the model registered a lot of positives. This may be because of the car crash. This had a large negative impact on the total accuracy of the model, which was only 50% on the given week.

5.5 Conclusion

One of the main goals were to increase prediction time and the accuracy of the model. A sub-goal was to increase the accuracy of the model over 85%. The goal of increasing prediction time was successfully met – the new prediction time is 10 seconds. Unfortunately the model accuracy was just under our optimistic goal – 83%, but is still a very good result which can be used in practice.

The secondary goal was to build an installable computer system. This was completed: the system design was developed and the model execution was tested on Rpi. All the relays were connected and the system is ready for on-site installation. The inference script also successfully gives a signal through the output relay. Unfortunately, the full cycle could not be tested, because the on-site installation could not be done due to circumstances that were not under our control.

5.6 Future works

Because the system could not be tested on the real intersection, this will be the main goal for the future works. If the solution works well on the Tammsaare-Sõpruse intersection, another future goal is to expand on other problematic intersections. This probably requires new training and test data parsing and model training, but the tools necessary were developed in this thesis.

6 Summary

This project was done in cooperation with Tallinn City Administration to find a solution for reducing intersection blockages. The goals were to increase the model prediction time, increase model accuracy and develop the hardware system design, so that the solution can be installed on-site.

The solution was to predict the blockage ten seconds before the red traffic light starts. Because the motion to the red light starts 5 seconds before (two seconds of blinking green light and three seconds of yellow light), the ten second prediction only saves five seconds.

Because the Tondi intersection was rebuilt between this thesis and Epp Mauring's thesis, the training and test data had to be re-gathered. To facilitate the expanding to other intersections, the scripts to parse and slice videos, extract images from them and create new datasets were developed.

In this thesis the previous model trained in TensorFlow was converted to PyTorch. The model architecture and input were enhanced considerably. Prediction time was increased from four seconds to ten seconds and the accuracy of the model was increased from 80% to 83 %.

The hardware choices that were compared for the system were: Arduino, RPi and Jetson Nano. The final choice was RPi, because it allows multiple running tasks unlike Arduino and has enough computing power to run the model fast enough. While Jetson Nano is even more powerful, the price goes up as well. It can be considered as a back-up choice or to reduce computation time even further.

While the system could not be installed in this thesis, because the Tallinn Transportation Agency had to reallocate their planned resources, the fully functional system was still developed. A signal is sent on the green light from the traffic light controller through an input relay to the RPi, which starts the inference script and outputs a signal through the output relay back to the traffic light controller. When the traffic light detects the received signal from RPi, it starts the light switch to red traffic light immidiately.

The potential impact of this thesis was theoretically evaluated. On average, 19 crossed the intersection per cycle, when there was no blockage. 11 did cross the intersection on average per cycle when there was a blockage. On the analysis period we had 5 big blockages, which means that 40 could have saved from waiting for another cycle on the given week.

As one car takes roughly five meters with spacings in traffic, those eight cars per cycle took about 40 meters of space away from just one intersection. This is more problematic to longer vehicles as there is already too little space in the rush hour.

Because the system could not be installed on the intersection, this will be left to future works. Given that after the installation the traffic congestions reduce, the expansion to other intersections can be considered. The tools and systems necessary for the expansion were developed in this thesis.

References

[1] E. Mauring, “Ristmiku kinnisõitmise ennustamine fooritsükli optimeerimiseks,” 2018.

[2] S. Zhang, G. Wu, J. P. Costeira, and J. M. F. Moura, “Understanding Traffic Density from Large-Scale Web Camera Data,” 2017.

[3] R. Nõugast, S. Sirkel, M. Rebane, and S. Aasaväli, “Project meeting,” Mar-2019.

[4] “Traffic Signal Timing Manual: Chapter 4 - Office of Operations.” [Online]. Available: https://ops.fhwa.dot.gov/publications/fhwahop08024/chapter4.htm. [Accessed: 26-Apr-2019].

[5] T. Metsvahi, *Ristmike läbilaskvuse arvutamise metoodiline juhend*. 2001.

[6] S. Lämmer and D. Helbing, “Self-Control of Traffic Lights and Vehicle Flows in Urban Road Networks,” *J. Stat. Mech. Theory Exp.*, vol. 2008, no. 04, p. P04019, Apr. 2008.

[7] M. Wiering, J. van Veenen, J. Vreeken, and A. Koopman, “Intelligent Traffic Light Control,” 2004.

[8] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis, “Multiagent Reinforcement Learning for Urban Traffic Control Using Coordination Graphs,” in *Machine Learning and Knowledge Discovery in Databases*, vol. 5211, W. Daelemans, B. Goethals, and K. Morik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 656–671.

[9] L. Wang, “The urban real-time traffic control (URTC) system : a study of designing the controller and its simulation,” p. 171.

[10] “Applying the IoT platform and green wave theory to control intelligent traffic lights system for urban areas in Vietnam,” *KSII Trans. Internet Inf. Syst.*, vol. 13, no. 1, Jan. 2019.

[11] “Green wave - Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/Green_wave. [Accessed: 26-Apr-2019].

[12] Md. F. Chowdhury, Md. Ryad Ahmed Biplob, and J. Uddin, “Real Time Traffic Density Measurement using Computer Vision and Dynamic Traffic Control,” in *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, Kitakyushu, Japan, 2018, pp. 353–356.

[13] ujjwalkarn, “An Intuitive Explanation of Convolutional Neural Networks,” *the data science blog*, 10-Aug-2016. .

[14] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks,” *ArXiv13126229 Cs*, Dec. 2013.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[16] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” *ArXiv14064729 Cs*, vol. 8691, pp. 346–361, 2014.

[17] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” in *Computer Vision – ECCV 2014*, vol. 8689, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 818–833.

[18] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *ArXiv13112524 Cs*, Nov. 2013.

[19] S. Hoo-Chang *et al.*, “Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning,” *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1285–1298, May 2016.

[20] D. Mazzini, M. Buzzelli, D. P. Pauy, and R. Schettini, “A CNN Architecture for Efficient Semantic Segmentation of Street Scenes,” in *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 2018, pp. 1–6.

[21] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, “A convolutional neural network cascade for face detection,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015, pp. 5325–5334.

[22] W. Shi *et al.*, “Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network,” presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 1874–1883.

[23] “TensorFlow,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/. [Accessed: 27-Mar-2019].

[24] “PyTorch,” 06-Mar-2019. [Online]. Available: https://pytorch.org/. [Accessed: 06-Mar-2019].

[25] K. Dubovikov, “PyTorch vs TensorFlow — spotting the difference,” *Towards Data Science*, 20-Jun-2017. [Online]. Available: https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b. [Accessed: 21-Mar-2019].

[26] R. V. Sevilimedu and C. A. Bahon, “Management of Computer Vision Algorithms for Social Robotics on an Embedded Raspberry Pi and an External Processor System,” p. 87.

[27] R. Tang, W. Wang, Z. Tu, and J. Lin, “An Experimental Analysis of the Power Consumption of Convolutional Neural Networks for Keyword Spotting,” *ArXiv171100333 Cs*, Oct. 2017.

[28] “Build from source for the Raspberry Pi | TensorFlow.” [Online]. Available: https://www.tensorflow.org/install/source_rpi. [Accessed: 21-Mar-2019].

[29] “How to install PyTorch v0.3.1 on RaspberryPi 3B - Tutorial · GitHub.” [Online]. Available: https://gist.github.com/fgolemo/b973a3fa1aaa67ac61c480ae8440e754. [Accessed: 21-Mar-2019].

[30] L. Pan, Y. Dai, M. Liu, and F. Porikli, “Simultaneous Stereo Video Deblurring and Scene Flow Estimation,” *ArXiv170403273 Cs*, Apr. 2017.

[31] L. Qu, J. Tian, S. He, Y. Tang, and R. W. H. Lau, “DeshadowNet: A Multi-context Embedding Deep Network for Shadow Removal,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 2308–2316.

[32] X. Fu, J. Huang, D. Zeng, Y. Huang, X. Ding, and J. Paisley, “Removing Rain from Single Images via a Deep Detail Network,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 1715–1723.

[33] W. Yang, R. T. Tan, J. Feng, J. Liu, Z. Guo, and S. Yan, “Deep Joint Rain Detection and Removal from a Single Image,” *ArXiv160907769 Cs*, Sep. 2016.

[34] “Real Time Streaming Protocol,” *Wikipedia*. 11-Mar-2019.

[35] “Data types — NumPy v1.17.dev0 Manual.” [Online]. Available: https://www.numpy.org/devdocs/user/basics.types.html. [Accessed: 13-Apr-2019].

[36] “What is meant by ‘Batch’ in machine learning? - Quora.” [Online]. Available: https://www.quora.-com/What-is-meant-by-Batch-in-machine-learning. [Accessed: 27-Apr-2019].

[37] “PyTorch.” [Online]. Available: https://pytorch.org/. [Accessed: 06-Mar-2019].

[38] “Raspberry Pi or Arduino Uno? One Simple Rule to Choose the Right Board.” [Online]. Available: https://makezine.com/2015/12/04/admittedly-simplistic-guide-raspberry-pi-vs-arduino/. [Accessed: 28-Apr-2019].

[39] “Arduino vs Raspberry Pi: How to Choose the Right Board | All3DP.” [Online]. Available: https://all3dp.com/1/arduino-vs-raspberry-pi/. [Accessed: 28-Apr-2019].

[40] “Nvidia Jetson Nano: the Raspberry Pi of AI?” [Online]. Available: https://www.tomshardware.com/news/jetson-nano-features-price,38856.html. [Accessed: 28-Apr-2019].

[41] A. Das, “How to install PyTorch v4.0+ on Raspberry Pi-3B+, Odroids, and other ARM-based devices,” *Medium*, 12-Dec-2018. .

[42] “How to change Raspberry Pi’s Swapfile Size on Raspbian,” *Bitpi.co*. [Online]. Available: https://www.bitpi.co/2015/02/11/how-to-change-raspberry-pis-swapfile-size-on-rasbian/. [Accessed: 25-Apr-2019].

[43] “Linux Time Command | Linuxize.” [Online]. Available: https://linuxize.com/post/linux-time-command/. [Accessed: 28-Apr-2019].