TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Kirill Tihhonov 204704IASM

Vehicle Counting and Direction Determination with Convolutional Neural Network Using Data From a 9.9GHz Low Power Microwave Radar

Master's thesis

Supervisor: Jaanus Kaugerand
PhD
Rene Pihlak
MSc

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kirill Tihhonov 204704IASM

Sõidukite loendamine ja suuna määramine konvolutsioonilise närvivõrguga, kasutades 9,9 GHz väikese võimsusega mikrolaineradari andmeid

Magistritöö

Juhendaja: Jaanus Kaugerand
PhD
Rene Pihlak
MSc

Tallinn 2023

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: Kirill Tihhonov

03.01.2023

Abstract

The increasing urbanization and growth of vehicle fleets in the European Union have led to public transport usage decrease and traffic congestion. These issues have accelerated the implementation of initiatives to improve public transport and more efficiently manage city road networks. Traffic monitoring systems have been suggested as a solution since they can identify and predict congestion while also providing valuable data for traffic management. Additionally, these systems also have the potential to make traffic control systems more efficient.

The purpose of this thesis is to discover the possibilities of vehicle counting and their travel direction detection accurately using data from a 9.9GHz low-power microwave radar with the help of a convolutional neural network. More precisely, the author's aim is to count the number of vehicles passing in front of microwave radar, their direction of movement and evaluate the possibility of classification between vehicle types, using 9.9GHz low-power microwave radar data. Several things make this research work special: building a bridge between the collected data and the data fed into the neural network training algorithm as a dataset; developing a way to balance data during training due to the difficulty of collecting rare classes; creation of a neural network model capable of working with data with low signal to noise ratio.

This research is related to one of the projects carried out in the Laboratory of Proactive Technologies in TalTech. As part of the SmENeTe project, 250 low-cost radar sensing systems have been already developed and deployed in Tallinn. The software developed for these systems is able to count vehicles and detect their speed, but it has a low signal to noise ratio. The authors of [1] proposed that using a convolutional neural network might improve the accuracy of vehicle counting and enable the detection of vehicle type and movement direction due to the high tolerance of convolutional neural networks for noisy data.

The secondary research problem consists in creating a light model which could fit a battery powered embedded device. This means that the computational resource and

memory capacity of the system created as a future solution is limited. The microwave radar transmits real-time time series data, which is continuously converted with sliding-window to spectrogram images and then fed to the neural network input. The neural network model being created should be light enough so that it would be possible to implement it on an embedded system that would be capable of real-time processing of the input data. The motivation of this thesis is to bring the cost of the system down by creating an efficient and light model.

The chosen project topic belongs mainly to the machine learning field of research. The solution is created with a help of the TensorFlow library, and the model could be transferred into an embedded system. Two convolutional neural network models were created as a part of this thesis. The first model is capable of detecting travel directions of vehicles travel directions as well as counting vehicles. On a test set, it has 91% accuracy. Cases when there were no vehicles present in the image were classified the best and for any class precision was not low than 76.73% and recall not lower than 73.65%. Additionally, the second model was created to test the possibility of classifying vehicles by type can differentiate between small and large vehicles.

This thesis is written in English language and is 77 pages long, including 5 chapters, 21 figures and 7 tables.

Annotatsioon

Sõidukite loendamine ja suuna määramine konvolutsioonilise närvivõrguga, kasutades 9,9 GHz väikese võimsusega mikrolaineradari andmeid

Suurenev linnastumine ja sõidukiparkide kasv Euroopa Liidus on toonud kaasa ühistranspordi kasutamise vähenemise ja liiklusummikud. Need probleemid on kiirendanud ühistranspordi parandamise ja linna teede võrgu tõhusama haldamise algatuste elluviimist. Liiklusseiresüsteeme on pakutud lahendusena, kuna need suudavad tuvastada ja ennustada ummikuid, pakkudes samal ajal väärtuslikke andmeid liikluse juhtimiseks. Lisaks võivad need süsteemid liikluskorraldussüsteeme tõhusamaks muuta.

Käesoleva töö eesmärk on välja selgitada, kas 9,9 GHz väikese võimsusega mikrolaineradari andmete abil on võimalik konvolutsioonilise närvivõrgu abil sõidukeid loendada ja nende liikumissuunda täpselt tuvastada. Täpsemalt on autori eesmärk loendada mikrolaineradari eest mööduvate sõidukite arvu, nende liikumissuunda ja hinnata sõidukitüüpide klassifitseerimise võimalust, radarist salvestatud andmete abil. Selle lõputöö teevad eriliseks mitmed asjaolud: kogutud andmetest närvivõrgu treeningalgoritmi sisendandmestiku loomine; haruldaste klasside kogumise raskuse tõttu treenimise ajal andmestiku tasakaalustamise viisi väljatöötamine; närvivõrgu mudeli loomine, mis on võimeline töötama madala signaali-müra suhtega andmetega.

Käesolev lõputöö on seotud ühe TalTechi proaktiivtehnoloogiate laboratooriumis läbiviidud projektiga. SmENeTe projekti raames on Tallinnas juba välja töötatud ja kasutusele võetud 250 madala hinnaga radar-süsteemi. Nende süsteemide jaoks välja töötatud tarkvara suudab loendada sõidukeid ja tuvastada nende kiirust, kuid mitte väga efektiivselt kuna radari väljundsignaalil on madal signaali-müra suhe. Artiklis [1] pakkusid autorid välja, et konvolutsioonilise närvivõrgu kasutamine võib parandada sõidukite loendamise täpsust ja võimaldada tuvastada sõiduki tüüpi ja liikumissuunda, kuna konvolutsioonilised närvivõrgud taluvad kõrge müratasemega andmeid.

Teisene uurimisprobleem seisneb võimalikult väiksese mudeli loomises, mis sobiks akutoitega sisseehitatud seadmega. See tähendab et lõpplahendusena loodud süsteemi arvutuslik ressurss ja mälumaht on piiratud. Mikrolaineradar edastab reaalajas aegrea andmeid, mis jooksvalt teisendatakse libiseva aknaga spektrogrammipiltideks ja seejärel suunatakse närvivõrgu sisendisse. Loodav närvivõrgu mudel peab olema piisavalt väiksemahuline et oleks võimalik valida sardplatvorm, mis suudaks sisendit reaalajas töödelda. Selle töö motivatsiooniks on vähendada süsteemi kulusid, luues tõhusa ja kerge mudeli.

Valitud projektiteema kuulub peamiselt masinõppe uurimisvaldkonda. Lahendus luuakse TensorFlow raamistiku abil ja mudelit on võimalik üle kanda sardsüsteemi kasutades TensorFlow Lite raamistikku. Selle lõputöö osana loodi kaks konvolutsioonilise närvivõrgu mudelit. Esimene mudel on võimeline tuvastama sõidukite liikumissuundi ja loendama sõidukeid. Test andmetel on mudeli täpsus 91%. Kõige paremini tuvastati olukordi kus sõidukeid ei olnud ja ühegi klassi puhul ei olnud kordustäpsus madalam kui 76,73% ja saagis madalam kui 73,65%. Lisaks on loodud teine mudel, et testida võimalust klassifitseerida sõidukeid tüübi järgi, mis võimaldab eristada väikeseid ja suuri sõidukeid.

See lõputöö on kirjutatud inglise keeles ja on 77 lehekülge pikk, sisaldab 5 peatükki, 21 joonist ja 7 tabelit.

List of abbreviations and terms

Adam	Adaptive Moment Estimation
AUC	Area Under Curve
CNN	Convolutional Neural Network
CSV	Comma-Separated Values
CVAT	Computer Vision Annotation Tool
ISC2PT	Intelligent Smart City and Critical Infrastructure Protection Technologies
LiDAR	Light Detection and Ranging
NN	Neural Network
mmWave	Millimetre Wave
ODD	Operational Design Domain
PNG	Portable Network Graphics
ROC	Receiver Operating Characteristics
RNN	Recurrent Neural Networks
RvNN	Recursive Neural Networks
SmENeTe	Smart Environment Networking Technologies
XML	Extensible Markup Language

Table of contents

1 Introduction 13
1.1 Research Background 13
1.2 Research Problem 14
1.3 Research Goals 16
1.4 Thesis Structure 17
2 Related Work 19
3 Method 23
3.1 Data Storage and Dataset Organization 23
3.1.1 Data Collection 23
3.1.2 Data Properties 26
3.1.3 Data Pre-processing 28
3.1.4 Data Augmentation 32
3.1.5 Dataset Building 39
3.2 Model Creation and Training 40
3.2.1 Introduction to Convolutional Neural Network 41
3.2.2 Convolutional Neural Network Architecture 42
3.2.3 Model Regularization 45
3.2.4 Model Optimizers 47
3.2.5 Model Optimization (Loss function) 48
3.2.6 Counting vehicles 50
3.2.7 Model Creation and Training (Vehicle Counting and Direction Detection) . 51
3.2.8 Vehicle Type Classification 55
3.2.9 Model Creation and Training (Vehicle Type Classification) 56
3.3 Model Evaluation 59
3.3.1 Model Evaluation Metrics 59
3.3.2 Model Evaluation Process 61
4 Summary 69
5 Acknowledgements 71
References 72

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis 77

List of figures

Figure 1 Microwave sensor signal, its spectrogram and detected vehicles. 24
Figure 2 Spectrogram of a random 60-second-long time frame..... 25
Figure 3 Three vehicle types (passenger car, commercial vehicle, and large vehicle) .. 26
Figure 4 Output from cvat_converter.py [16] 31
Figure 5 Training dataset without overlayed images..... 35
Figure 6 Image filtering to identify zones of interest 36
Figure 7 All possible combinations when rolling 2 dice [41] 37
Figure 8 Image pair distribution 38
Figure 9 Convolutional Neural Network Architecture [52] 43
Figure 10 Most popular activation functions..... 45
Figure 11 Loss function application [48] 49
Figure 12 Baseline model architecture 54
Figure 13 CNN structure for vehicle type classification 56
Figure 14 Matrix of choices [69]..... 60
Figure 15 Confusion matrix for vehicles coming from left (beginning) 62
Figure 16 Training and validation accuracies (vehicles from left)..... 63
Figure 17 Training and validation accuracies (vehicles from right) 63
Figure 18 Training and validation losses..... 64
Figure 19 Confusion matrix vehicle coming from left 65
Figure 20 Confusion matrix vehicle coming from right..... 66
Figure 21 Confusion matrix of vehicle type detection 67

List of tables

Table 1 Most popular activation functions and their description [47] 44
Table 2 Hyperparameters chosen as a baseline for vehicle counting model 52
Table 3 Hyperparameters chosen as a baseline for vehicle type detection model 57
Table 4 Precisions and recalls for vehicles coming from left..... 62
Table 5 Precisions and recalls for vehicles coming from left (balanced dataset)..... 64
Table 6 Precisions and recalls for vehicles coming from right (balanced dataset) 64
Table 7 Precisions and recalls for vehicle types..... 66

1 Introduction

1.1 Research Background

Vehicle fleet grows in European Union and urbanization is a global trend [2]. Urban populations are growing, requiring more and more transport to meet their needs. Cities are growing in area, but due to the already existing development, it is not always possible to build roads that would be capable to provide round-the-clock non-stop travel. Where the roads are not wide enough, bottlenecks emerge. On the other hand, wider roads would have a negative impact on urban infrastructure, giving large areas of urban territory for transport.

According to [3] the following issues and anxious trends are general in urban transport sector:

- Rapid growth of cities
- Motorization growth
- Growing traffic congestion
- Public transport usage decrease
- Public service substitution by private

Undoubtfully consequences of these listed problems will put a heavy load on the current road network. In contrast to issues listed above the initiatives to improvement of public transport and managing city road network more efficiently began to appear [3]. Thus, it is uncertain if it is possible to use the current road infrastructure more optimally and whether it is viable to save the city public space through solving new demands by creation of a new infrastructure allowing to manage traffic flows.

Traffic problems cannot be solved without traffic data and information which are needed to assess current and past performance and to predict future performance. Over the previous years authorities have been searching for solutions enabling traffic surveillance

and control. Traffic monitoring systems are the solutions to provide valuable data to concerned road departments and other stakeholders related to the traffic management. Additionally, such systems might recognize and forecast congestions, thus enable better traffic flow modelling and congestion prediction [4]. This output could be applied for traffic control systems making latter safer and more efficient. Furthermore, according to the estimation done in [5] vehicle to infrastructure and vehicle to vehicle systems could reduce crash consequences by up to 80%. However, real-time traffic monitoring systems present many challenges, where data processing and cost play significant roles [1].

1.2 Research Problem

Nowadays there exist various traffic monitoring systems but camera-based systems are the most popular [6], [7]. Cameras are easily deployable, and many solutions utilize already installed cameras. On the one hand, a significant advantage of camera system is development ease. While on the other hand, such systems are prone to weather and lighting conditions, which significantly complicates installation locations or reduces availability of such system, making concerned parties loose valuable data. Camera solutions also require high computational resources, which increases overall system cost. These problems have enabled research for other sensor types.

There also exist works and companies showing implementation of LiDAR (Light Detection and Ranging) or radar-based systems. Both sensor types are becoming more available in terms of price due to growing consumption of these sensors by automotive industry. Nonetheless a unit price is still far more expensive than of a camera unit [8], [9]. This has enabled research for low-cost radars for traffic monitoring systems.

This thesis builds its research task on a low-cost microwave radar traffic monitoring system developed in research [1] conducted in cooperation between Tallinn University of Technology Laboratory for Proactive Technologies and Thinnect OÜ company. The research was conducted in terms of SmENeTe project [10]. The focus of this project was to develop the next generation IoT networking components to support applications of smart environment. This thesis is already done within ISC2PT project [11], which is a follow-on project to SmENeTe project. The aim of ISC2PT project was to develop next generation sensor technologies to monitor smart cities and critical infostructure.

By the time author joined the project there were already 250 radar sensing systems solutions developed during SmENeTe project and deployed in Tallinn. These devices were installed at street lighting posts on a height of 3.5 meters. The solution is elegant since it uses already made infrastructure. Considering the amount of used radar sensors, it can be concluded that operational design domain (ODD) for each sensor should be at least slightly different. Some traffic situations are more complex (including intersections/take-overs/multi-lane roads) than others, so there would be required a huge work over data collection and processing in order to fulfil author's aims. Therefore, to exclude complex cases, it was agreed to determine straight non-intersecting roads which consist of only two lanes each facing an opposite direction as thesis solution ODD.

The developed radar systems are capable of counting vehicles and detecting speed of those. However, the system performance was affected by low signal to noise ratio. Therefore, there was an opportunity to improve the results. Since convolutional neural networks (CNN) are known to be noise resistant, authors of [1] have raised and proposed a hypothesis that a CNN implementation might improve vehicle counting accuracy as well as allow vehicle type and vehicle movement direction detection.

Thereby, thesis task was to test the hypothesis and improve already developed solution by using CNN. This thesis should also provide an input on the computational performance and embedded system memory size. It is planned that the future system real-time time series data would be transmitted by the microwave radar, which would be continuously converted with sliding-window spectrogram images and then fed to the neural network input. Author had some general research questions which will be addressed throughout the work. The first issue that had to be resolved was the data storage and processing and the search of the best practices for these tasks. Author studied significant amount of works, most of these stating that dataset size should be large [12]–[14]. However, minority of those could point out precisely what would be an optimal size for the dataset. Dataset class distribution balance was another author's concern due to low presence of image instances where two or more vehicles are captured at the same time.

Data observation showed that collected data is likely to contain less useful data (containing passing vehicles) than useless data (no vehicles present) when data collection is done in locations specified in ODD. Thus, the question arises whether the developed neural network model should generalize more for empty cases (no present vehicles) or

not. On the contrary dense traffic is a rare case, especially when there are more than three vehicles driving in a row one after another in specified ODD areas. Especially within a timeframe of six seconds that was specified as a window time length fed into the future model.

Development of own CNN design was also challenging as there is broad choice of literature illustrating various approaches. Some authors present innovative approaches which seem quite attractive. And there is a lot of ready-made designs which are appealing to draw inspiration from. As diving deeper into deep learning field there appear more details and parameters to tune. Thus, when aiming that each step would be thought through and there would theoretical background behind it, huge number of sources should be read, and development done [12].

1.3 Research Goals

Based on research problem there were identified five research goals which are suitable for scope of this thesis:

- Count vehicles
- Detect vehicle travel directions
- Improve accuracy of existing system
- Detect vehicle types
- Network should be capable to fit an embedded device

As it was previously stated the problem of this thesis originates from previous work conducted in SmENeTe project. Since the dataset used in this research is based also on previous research [1] where signal/noise ratio is not the best, convolutional neural networks implementation having a higher tolerance to noisy data could show better performance compared to other algorithms [15].

Vehicle counting and travel direction detection were the main goals of this research. As it was identified earlier the solution [1] should count up to three vehicles travelling in one direction. Based on ODD there are two travel directions, so a vehicle can approach on a radar or depart from it. Vehicle type detection is not the main aim of this thesis since

amount of collected vehicle type instances imposes limitations on NN model training. These three goals should be achieved on static dataset images, thus providing a solution that could perform on data stream is not in the scope of this thesis.

Even though accuracy is not very precise metric for multi-class classification it is a comprehensible term. Therefore, there was reached an agreement that author will aim highest possible accuracy (and overall quality according to other metrics) for counting and detecting vehicle travel directions, but it needs to be higher than 80%.

Since the developed solution is planned to be deployed with a radar system in the future, which is a battery powered device with limited computing capacity, CNN model developed in this thesis should fit a low-cost embedded devices and be capable of running on it performing first three goals. This thesis will also indicate how powerful embedded platform for classification might be needed.

Based on the works studied in chapter 2 Related Work, there were identified following research gaps. There exist various approaches on how to classify vehicles using microwave radars, however, author did not find any solutions developing convolutional neural networks to count, determine travel directions and vehicle types using low-cost microwave radars installed at already made infrastructure which has high availability in the city environment.

Previously stated problems have to be solved keeping in mind that a model has to be small in size so it could easily fit a low-cost embedded devices. Taking the abovementioned into account the overall motivation is to create a low-cost solution that will allow classifying vehicles by types on the roads with low and medium traffic. However, the scope of this thesis does not include neural network integration on an embedded platform nor optimization for microcontroller.

1.4 Thesis Structure

This thesis consists of three main chapters: Data Storage and Dataset Organization, Model Creation and Training and Model Evaluation. The first data related chapter serves as a description of work done regarding the dataset creation. Starting from an introduction of input data in chapter 3.1.1, moving on to data properties described in chapter 3.1.2. The data part also gives an overview of data pre-processing activities in chapter 3.1.3 and

augmentation in chapter 3.1.4. The first part ends with dataset creation description in chapter 3.1.5.

The second part discusses neural network development. It begins with introduction to neural networks from the thesis task viewpoint and explains why certain decisions have been made. The main research and following development have been made at this point. Chapters 3.2.1 to 3.2.5 are descriptive and form a background relevant for creation of convolutional neural network model. Chapters 3.2.6 and 3.2.8 are analytical and explanatory to the solutions made in chapters 3.2.7 and 3.2.9.

The third part reviews model evaluation metrics and describes how evaluation of developed models was done. There is a brief introduction into evaluation metrics in chapter 3.3.1 and their description as well as consecutive description of model evaluation during the development in chapter 3.3.2.

The following source [16] contains the programs that were used to create this thesis. The program code is publicly available and distributed under MIT license.

2 Related Work

As stated before, this thesis is based on a previous work that targeted the application of low-cost microwave radar to traffic monitoring [1]. In that work authors show a feasible solution though creation of the algorithm that detects a number of vehicles, determines the passing directions and velocities of the vehicles. Detection is performed in two steps. First, spectrogram is created by Fourier transform and secondly changes in spectral power are sought for vehicle detection and direction is identified by spectral power growth direction.

The obvious advantage of the solution provided in [1] is the use of the existing infrastructure, which is confirmed by the use of similar devices throughout Tallinn. In addition, the radars used in the solution are low-cost (price about 30 euros), which again increases the availability of this system. However, the system experienced some difficulties since the signal/noise ratio was not the best and also different at each location.

Conventional traffic monitoring systems are mostly camera-based [6], [7]. This can be also demonstrated by various companies like Aaeon AI [17] and Vision Genius [18] offering their traffic monitoring solutions utilizing cameras. Nevertheless, as other sensor type prices fall due to increasing usage in automotive sector, there appear attempts to monitor traffic utilizing radars and even LiDAR's. Nowadays cameras do not seem like a perfect solution for traffic monitoring system primarily due to computer vision detection being dependent on good weather and lighting conditions. But despite camera is a relatively cheap sensor it requires lens as well as high computational power. Frequently engineers seek for lens, image resolution and vision range balance, so the system would act best in certain environment. Worth to mention that usually camera vision range is limited.

Following author analyses camera implementation solutions in traffic monitoring systems by below mentioned companies. Solution by Aaeon AI does not require any additional infrastructure – cameras are mounted on lighting post and field of view is directed towards the ground. Author assumes Aaeon AI relies on lighting provided by the posts since computer vision capacity is likely to fall during bad lighting conditions. Vision Genius solution called Traffic Genius seems ready for installation in various locations. In the example on the website camera is seeming installed on a bridge observing highway

traffic. Night light conditions might be compensated by available surrounding light sources like lighting posts. Vision Genius claims Traffic Genius solution is capable of vehicle detection counting and classification.

LiDAR's on the other hand are more resistant to weather and lighting conditions. But not completely resistant to environmental conditions and for instance have troubles working in snow and rainstorms. This is primarily cause by LiDAR light reflection and ranging work principle when laser beams are used to grope the surroundings (physical objects). LiDARs are expensive and price reduction comes with significant trade-offs. Low-cost models are prone to false reflections and phantom objects. Evaporation sensing is typical even for high-end models. LiDAR's have limited field of view and changing point density dependent on object proximity from the sensor. Point cloud segmentation or classification may also require significant amount of computational power.

Author has also found two companies implementing radar-based traffic monitoring solutions. Oyla [19] provides a LiDAR and high-resolution camera fusion based solution. LiDAR is reportedly cost-efficient. Even though sensor might be low-cost, this system should have high computational costs. Oyla does not state any specific requirement to installation location of their system. Bluecity [20] proposes 360 LiDAR installation in some centre point of an intersection. System is reportedly able to conduct multimodal classification.

Radars seems the best choice for vehicle traffic monitoring. Metal surfaces are highly reflective with respect to microwaves. However, other traffic participants can be also detected using radars. Absolute advantage of radar is object speed measurement which implementation is done easily. Radars also perform the best in severe lighting and weather conditions delivering almost 100% availability. Thus, information regarding traffic changes during severe weather conditions might be acquired.

Smartmicro [21] traffic monitoring 24GHz radar solutions do not require additional infrastructure and have installation flexibility. Though it is recommended to use those on highways or intersections. Solutions are able to count up to 256 vehicles simultaneously and can classify between vehicle types as well as other traffic participants such as pedestrians and bikes. Cheapest model with a price starting from 3000 euros [22] has 100-degree field of view and has a range up to 219 meters.

Texas Instruments proposed usage of their mmWave radar in [23]. Advantage of mmWave radars lies in their low-cost compared high resolution radars like Smartmicro. In the work authors showed methods of object position and velocity measurement, however, did not focus on classification of vehicles and pedestrians. Authors were able to achieve approximately 60-meter detection distance within 22-degree field of view.

Knowing that camera and LiDAR systems cost considerably more compared to microwave radars [1] and that a radar is more robust when compared to the camera and is not sensitive to weather or light conditions perception-wise it can be proceeded with radar solution evaluation only.

There are several works focusing on vehicle type classification using microwave radar. In [24] authors present a solution based on vehicle height and length and height profiles usage obtained by two microwave radar sensors. The radar is located above the road in such a way it could see profiles of passing by vehicles. Unfortunately, such places are rare to meet, thus installation of such a system might be costly and impossible at some locations. In [7] authors propose a method to monitor traffic lane-by-lane claiming they achieved stable detection and tracking performance. Authors have designed the whole radar system from hardware to software. The system is capable of vehicle detection, speed, lanes up to 300 meters. The system was installed on a pedestrian bridge, which limits the availability of installation locations. However, the authors in [7] do not propose any methods on how to distinguish between vehicle types.

In [25] authors show a method which is designed for vehicle detection, tracking, and computing its speed using radar and camera in one system. Detection is based on dual background subtraction and sliding window for vehicle pose estimation. Radar data is transferred to server where it is processed. Authors claim seven radars require one computer to be processed. However, authors main focus is speeding vehicles detection and camera is used for making a video of speeding vehicles.

In [26] author proposes mmWave radar and camera sensor fusion to detect and track vehicles. Author has chosen a bridge as an installation location. The solution seems to have low performance when detecting small or large vehicles, nevertheless the solution managed to detect most of the passed vehicles in a short test. Work also does not propose a solution to vehicle travel direction identification nor type classification.

There were found also other works related to traffic monitoring utilizing a low-cost radar and CNNs. In [27] authors have proposed a solution for traffic monitoring at intersections. Solution was based on a Texas Instruments mmWave FMCW radar and classified over clutter, pedestrians, and radars. According to the point clouds present in the research the resolution of mmWave radar is comparably low to classify over the vehicle types. Authors of [28] research camera and radar sensor fusion trying to solve image CNN small object size detection problem. Focus is done on the camera perception system improvement classifying vehicles and pedestrians. In both works authors highlight ability of CNNs to show high performance in various scenarios. Thus, implementation of a low-cost microwave radar for traffic monitoring is relevant, whereas CNNs are encouraging result-wise. The application of low-cost microwave radar in traffic monitoring can possibly save costs on monitoring devices, which is the main cost of traffic monitoring systems.

3 Method

3.1 Data Storage and Dataset Organization

Essential and first step when a scientist desires to develop some neural network model or algorithm is data related work. This thesis starts with suitable data collection and data processing under certain rules results in a created dataset. If there is nothing to feed into neural network model when training it is not only useless, but also it would not be developed since neutral network algorithm training process depends on data.

3.1.1 Data Collection

Nowadays in many cases data is already collected and the task of data scientist is often to decide how to develop a neural network on this data that would contribute to business. Author had a similar case when a task within a project was to improve already created solution. Therefore, it would not be handled in scope of this thesis how should be data collected and stored according to a certain development task. Although data collection is not within the scope of this thesis it remains a significant part and needs clarification before moving on to further development.

Locations for data collection there were chosen straight (without intersections) road segments in Tallinn consisting of only two lanes each facing an opposite direction. These segments were chosen specifically. Raw data regarding vehicle types and traveling directions was being collected with microwave radar. The collected data is represented by 60-second-long frames (horizontal axis in Figure 1), containing a sequence of numbers in ASCII encoding, representing a radar signal. The signal which is measured in vols is normalized into the interval [-1,1] (Figure 1, upper subplot). Next, Fourier transform is applied to the signal and the resulting spectrogram is depicted in Figure 1 (middle subplot). Y-axis of Figure 1 (middle subplot) represent vehicle velocity, thus figure height in spectrogram represents vehicle speed. The algorithm described in [1] publication detects a number of vehicles (Figure 1, lower subplot) by total spectral power (y-axis of Figure 1, lower subplot) and attempts to determine the travelling directions and velocities of the vehicles.

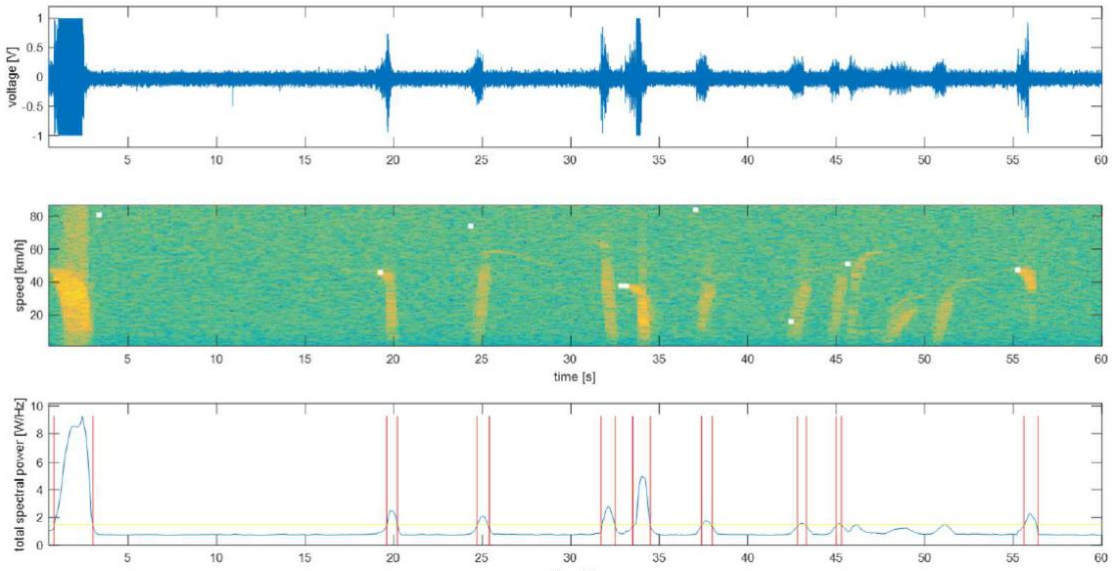

Figure 1 Microwave sensor signal, its spectrogram and detected vehicles.

The Fourier transform applied to the signal results in a spectrogram that is saved as a monochrome image in PNG format. Although, authors task was to work with spectrogram images, there was a freedom to create those himself.

On the Figure 2 a spectrogram based on a 60-second-long time frame is demonstrated. During 60-second-long period the microwave radar captured various vehicle travelling in the opposite directions. The horizontal image axis is responsible for time. The location of the figure representing a vehicle on the spectrogram image along the time axis indicates the time location of a vehicle in a given time interval. The height of the figure is responsible for vehicle velocity. The higher, the faster the vehicle was moving, respectively, the lower - the slower.

While the classes of different vehicles differ mainly in the width of the figures on the spectrograms, the direction is a little easier. In the real world, the radar is located in such a way that the main vectors of motion are directed to the radar or away from the radar. However, on the spectrograms themselves, the figure tips of vehicles are directed to the right or left. Therefore, to simplify and avoid confusion, the movement of vehicles will be considered linearly: to the right or to the left.

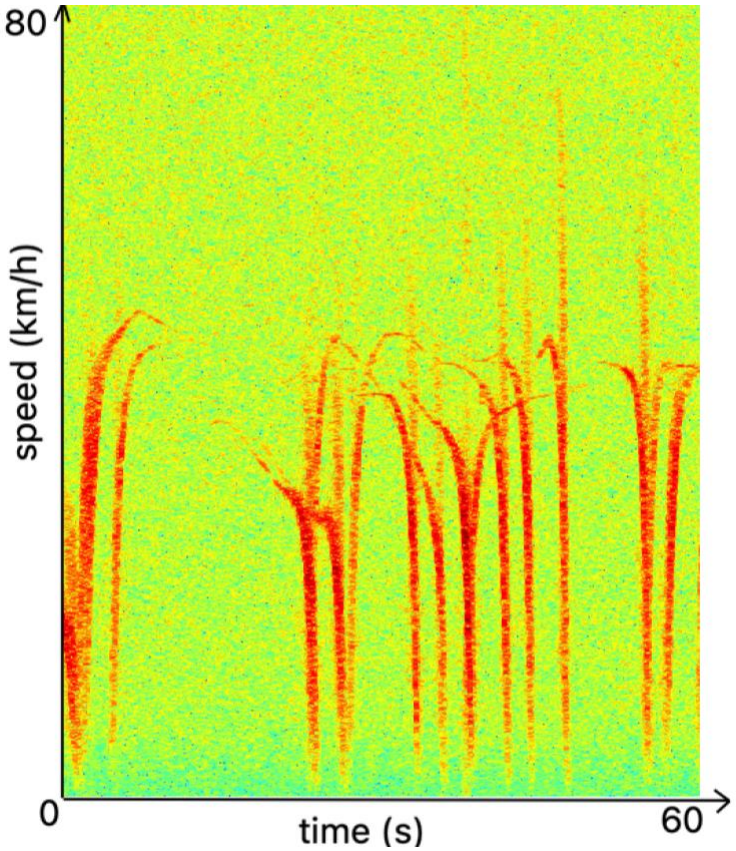

Figure 2 Spectrogram of a random 60-second-long time frame

Further received images were cut so that each image would be split into ten smaller segments. Resulting images are 60 pixels long, which corresponds to a time frame of six seconds. The width of the figure itself is responsible for the size of the vehicle. The larger the car, the wider the figure on the spectrogram will be. Thus, one can distinguish between types of vehicles by the width of the figures on the spectrograms. On the Figure 3 there are demonstrated three vehicle types: passenger car (Figure 3, A), commercial vehicle (Figure 3, B), and a large vehicle (Figure 3, C). These three vehicles travelled with different velocities; large vehicle had the lowest.

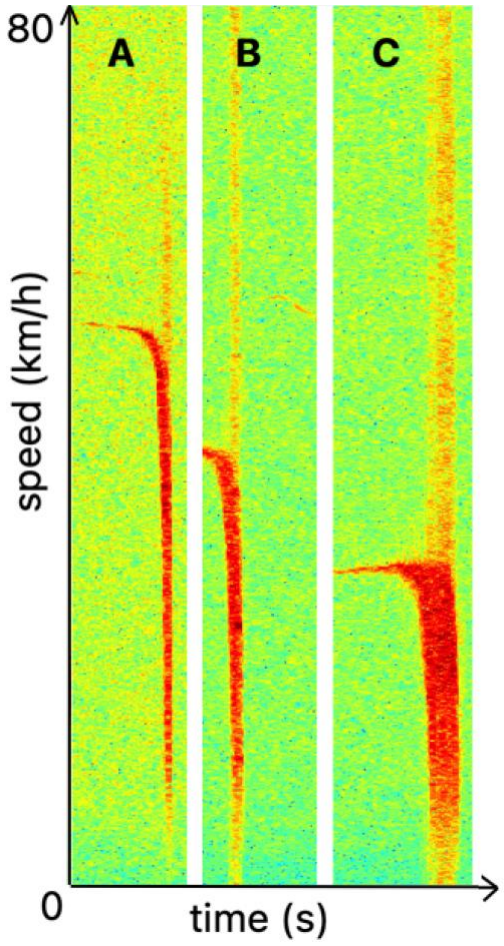

Figure 3 Three vehicle types (passenger car, commercial vehicle, and large vehicle)

A large number of spectrograms was reviewed, and only the most applicable spectrograms were chosen for classification model training. The spectrograms were selected based on the condition that these must be human-readable and be representable as possible.

3.1.2 Data Properties

When data is already collected it needs to be stored somehow to be processed later. Creating a dataset, in turn, happened to be a very difficult task, despite it seemingly being simple. The following question required answers:

- How are datasets structured?
- Which properties/features should data have?
- How should images be linked with labels?
- What is the sufficient size of a dataset?

- Should the dataset be balanced?

Author have sought for answers in various sources. Unfortunately, it was difficult to find capacious answers to questions addressed. To answer what is the sufficient size of a dataset author had to search for quite a while. There were no scientific articles or works focused on that topic. At least it required some effort to find some. Nevertheless, in [29] authors have experimented with various NNs and came to the solution, that each class of a dataset should be represented by 150-500 images. Similar results are also shown in [30]. In [31] author has also found an answer whether classes should be balanced between each other. It is said that in case of a more than 10-time difference in class size training is likely to become problematic. It is also recommended to avoid training when dataset is heavily imbalanced.

"Dataset Definition Standard (DDS)" by DEEL [32] does not answer stated questions in a straightforward manner, it still contains very valuable information on machine learning dataset standardization. As mentioned in the paper collected data should correspond to the following properties:

- Representativeness
- Traceability
- Accuracy and precision
- Reliability
- Consistency
- Integrity
- Unintended bias

These properties are discussed in this chapter from a dataset point of view. So, in scope of a dataset, representativeness means how well are classes of a given task are represented based on the operational design domain (ODD) of a machine learning algorithm created to classify between those classes. This means in order to a certain dataset be representative in case of image classification it must correspond to specified ODD [32].

Since the data can be pre-processed, several times before being fed into developed algorithm during training it is vital to keep track of data modifications and be able to trace from to end point to initially captured data. This explains traceability definition [32]. Thus, it is a good practice to document any data modifications, data usage and results. Amazon Web Services states [33] that data traceability is important since it helps to observe and understand the influence of data on model.

As described in [32] accuracy and precision are measured between the data stored in the dataset and the data produced by the solution when it is functioning. Accuracy is meant as representation quality and precision to a correspondence to ODD of a functional product.

Reliability is a trustworthiness of data in relation of ODD. Dataset should be cleaned out from all unnecessary information [32]. Cleaning the dataset is extremely important so any artefacts or misjudgements would not be present in the neural network model. However, this topic is more related to certification of machine learning algorithms.

Dataset is consistent if it does not contain redundant data. Redundant data may get into both training and test dataset and thus influence the test results. It is vital to highlight that there may be present not only duplicates of data, but also inconsistent labels or images.

Data integrity basically means that there should not be applied any changes to origins contained in the dataset. Data itself could be processed, however it should be done outside of a dataset.

Bias in dataset refers to representation of features observed in dataset images. These features might be contained to a greater or lesser extent. In image classification task some features containing in the images for some class may make the classificator take this bias into account. So, whenever any other image of this class will be introduced to the classificator during test phase there could arise a classification error. Having such an unwanted feature is called unintended bias and must be avoided.

3.1.3 Data Pre-processing

3.1.3.1 Introduction to Data Annotation

Annotation of data means assignation of labels to images (in authors case), considering that the process should ensure no errors are done. However, it is important to keep in

mind that processed data should remain traceable. There exist manual, semi-automatic and automatic means of image annotation.

Manual image annotation is an important but tedious process, since it requires lots of decisions being made and sometimes it is difficult to decide which class should be assigned to an image. However, from the benefits side manual image annotation provides higher accuracy during the process since humans understand the semantic content of the picture better [34].

The purpose of automatic image annotation is to keep humans uninvolved. There are multiple ways to automatically annotate the data. In terms of radar sensor one of examples could be adding a camera system running YOLO object detector in parallel with collection of radar data. Every time an object of interest passes by both sensors a timestamp is saved. However, automatic annotations are potential source of errors. In [34] authors have made a review of methods for the image automatic annotation methods. It is highlighted in the work that there are present issues with extraction of textual characteristic and semantic gap, which are present to observed automatic image annotation methods. In [32] authors recommend avoiding image annotation in case of ambiguous images.

As described in [32] labelling shall be executed bearing following definitions in mind:

- Accuracy
- Consistency
- Absence of annotation bias

Labels should correspond to the information contained on the images in respect to the task. Accuracy refers to the ratio of correctly labelled images to all images.

Even for a single person it is difficult to label the data consistently since humans are not robots a constant threshold for class detection cannot be set. It is inherent for a human to doubt and generally saying other factors like mood fluctuations might influence decision-making process. While data annotation a person learns more details from the data and decisions may be also influenced. It is also a source of annotation bias.

3.1.3.2 Data Annotation

Having a certain strategy and as nearly precise as possible thresholds is vital while data annotation. In this case automatic data annotation would help, however as previously described it is a potential source of labelling errors. Besides, the idea to annotate the images manually was recommended by supervisors since it provides a great knowledge on the data value inside a process as well as helps to understand the process more.

Onwards, the annotation of pre-selected samples was done in a popular image annotation tool used for this task – Computer Vision Annotation Tool (CVAT). Each figure was lead around with polygon shape and then annotations exported in CVAT for images 1.1 format. Manual annotation and data storage in this format also insures data traceability. Annotations were contained into one file, so to proceed it was necessary to have annotations divided by classes. Therefore xml_remove.py [16] was created, which required some manual work, however, was sufficient for infrequent use of annotations file separation. Further actions and data conversions were also done keeping in mind the data properties handled in chapter 3.1.2 Data Properties.

Article [35] shows a nice example of masks creation from annotations file. The annotations were saved in CVAT image format and masks were generated from those. Figure 4 represents the output from annotations converting program (cvat_converter.py [16]). In case of Figure 4 there are shown three vehicles travelling to the right direction. Outputted masks were color-coded so it would be more intuitive to distinct between classes.

Figure 4 Output from cvat_converter.py [16]

The program requires several arguments to operate. Most important of these are input and output paths. The code for mask creation from XML annotations (cvat_converter.py [16]) is based on a legacy CVAT program from CVAT GitHub repository. It is distributed under MIT license and there were done slight modifications by the author to fit his needs more precisely.

3.1.3.3 Dataset creation

Following the further programs were used for image slicing and labelling. Two programs that formed a dataset a described further:

- prep_rene.py [16] – forming slice labels in .csv file
- split_data.py [16] – image slicing and saving with one-hot encoded labels containing filenames

prep_rene.py [16] uses all images and masks prepared earlier without class differentiation as input and outputs a CSV file containing image name, slice and label of figure(s) contained on the image. Masks and images have the same base names, so a base name is used a key to match the right pairs of masks and images.

Images and masks are sliced inside the program. For increasing amount of overall data slicing is done with several offsets (0/15/30/45 pixels) horizontally and with a slicing width equalling to 60 pixels. Then a mask slice is analysed whether it contains a figure. There is also a threshold in pixels given what could be a figure. This threshold was calculated empirically and equals to 700 pixels. It is done since some images contain figures on the borders that would be quite difficult to distinguish and estimate vehicle direction. So, if there is figure size below threshold it is assigned to nothing class. The border problem also made author to consider avoiding padding during neural network models creation so it would not rely on border placed information.

split_data.py [16] is a simpler program. It takes the csv file outputted on the previous step, parses it for image base name, split number, and labels. Then pairs of images and labels are found, images are slitted on the same rule as in the previous step and images are saved with one-hot encoded labels in the filename.

After finishing with the first batch of the images author reassessed the order of dealing with images. It made more sense to slice the images first and then to label those since it was necessary to conduct a lot of manual correction to the labelling done.

3.1.4 Data Augmentation

3.1.4.1 Introduction to Data Augmentation

There are several ideas behind data augmentation. Data augmentation might be the key to solving frequently faced problems of data scientist. First, data augmentation is a solution to poorly represented datasets. More data can be generated with the help of simple and/or sophisticated techniques. It also helps to cut down the data collection costs since depending on the data type, collection might become complex. Augmentation might also help to reduce overfitting by providing diversity to data. Besides, it may also increase overall model generalizability.

Initially to learn about data augmentation methods author turned to TensorFlow data augmentation tutorial [36]. There are given such methods as resizing, rescaling, flipping, rotating, transposing, colour space convert, image adjustments like brightness and saturation and cropping, which may be considered as classic simple ones. Methods introduced in the tutorial are a subject of use since integration of these is comparably easy. Methods were brough down to those that could be applied on this project images. As it turned out at first glance the list of methods could be narrowed down to mirroring and cropping with padding.

Authors in [37] provide the following augmentation methods: geometric transformations, colour space augmentations, kernel filters, mixing images, random erasing, feature space augmentation, adversarial training, generative adversarial networks, neural style transfer, and meta-learning. These obviously overlap with those introduced by TensorFlow; however, some other methods are also shown, including synthetic generation methods. In [38], [39] there also presented additional synthetic data generation methods. Such as deep learning models, stochastic gradient descent, Bayesian networks, variational auto-encoders, differential equations.

Even though augmentation can solve a lot of problems it can create some. Therefore, when augmenting data, it is important to keep in mind the data properties mentioned in chapter 3.1.2 Data Properties. Data consistency must not be violated, and unintended bias creation should be avoided. So, even though augmentation is based on initial data usage it should significantly differ.

In [38] authors also introduce the requirements which could be used for definition of a good data generator algorithm. Thus, a good data generator should be:

- Syntactically accurate
- Statistically accurate
- Efficient

Thereby, there exist methods on generated data validation. A simple method would be accuracy comparison between real and synthetic data. Statistical methods could be also applied. Contradiction between models trained on real and generated data is also possible

as well as combinations when measuring model performance on generated data using a model trained on real data and vice-versa [38].

3.1.4.2 Realization

Initially author planned augment data with TensorFlow methods within one notebook. There was used tf.data API in order to create efficient data input pipelines in train_ver_baseline_v5_classic_arch.ipynb [16] notebook used for CNN training. In [40] authors show how is training time affected by input pipeline performance. Therefore, it seemed a justified API choice for image inputting. Following, datasets of image names were created, and images were mapped accordingly to datasets. As a result, training validation and test datasets were transferred from locally stored as files on physical drive to tf.data.Dataset objects in computer random access memory.

At some point during neural network training, it became obvious that dataset has uneven class distribution, having no balance between classes. At the same time, it was important to raise the performance of the developed model. Empty and one car instance class was dominating over other classes, so the algorithm was learning to classify between those two classes. Not mentioning vehicle types, when a chance to meet a large vehicle on specific road might take from 10 to 20 minutes. As discussed in chapter 3.1.4.1 Introduction to Data Augmentation there are various methods to cope with class imbalance. Nevertheless, author had decided to choose image mirroring and overlaying (image mixing) since these methods seemed easy to implement on a given dataset.

TensorFlow offers various techniques to augment datasets or images [11], however, offered methods are limited. For instance, tf.image module allows only to rotate images by 90 degrees. Therefore, the author had decided to create own augmentation methods. Author decided not to use TensorFlow built-in methods to mirror the images additionally, due to usage of tf.data API for previously created datasets in notebook. tf.data.Dataset objects which datasets were transformed into are not easily traceable and code integration requires more effort. Therefore (MirrorImages.py [16]) was created so augmentation was done before data was inserted to the notebook.

It reads in all images in the specified folders, iterates over these getting labels, slices numbers and base name. Is mirrored with np.fliplr() method and saved with new image

name where left and right labels are swapped. Also, “mir” letters were added to the image names to insure higher traceably.

Once images were mirrored there arose the necessity in class balancing since dominant classes were outperforming other classes during the training and the model could not generalize to non-dominant classes. Image overlaying seemed better than other methods since it could solve several issues at the time.

Therefore (Augmentor.ipynb [16]) was created which aimed to compensate submissive classes and bring down data collection cost. After reading in the images these were filtered down by labels so only appropriate ones were used. As instance it was decided not to use images from validation and especially testing dataset. Images already possessing 3 classes were not required at this point. In Figure 5 dataset class distribution without overlayed images can be observed. Worth mentioning that it is already doubled by mirrored images.

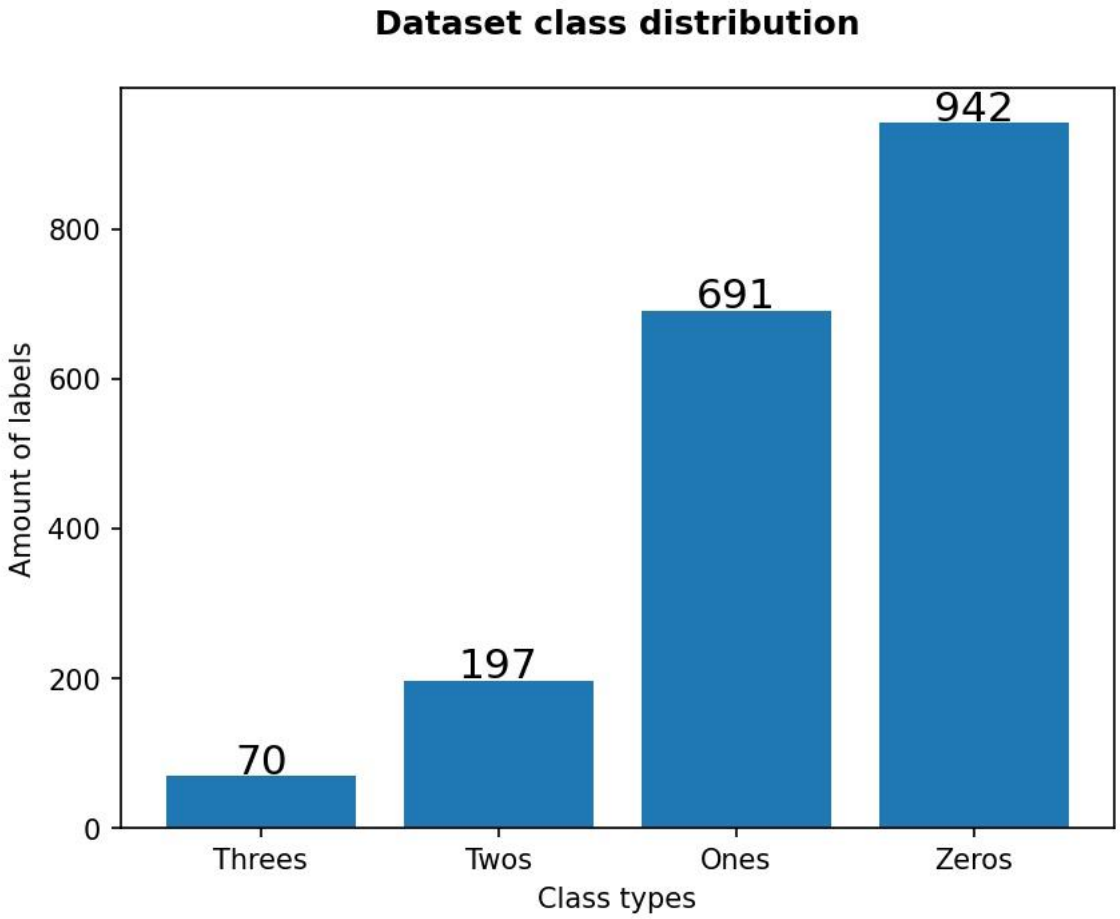

Figure 5 Training dataset without overlayed images

Filtering also included identification of interest zones and highlighting those. This was done with firstly applying cv2.medianBlur() method and then setting all image pixels below threshold to zero with np.where() method. Threshold was calculated as half of delta between image minimal and maximal images pixel values (0..255) added to minimal image pixel value. Result can be observed in Figure 6, where initial dataset image is on the left, blurred image is in the centre and image without background is on the left. Numbers on y and x axis represent image size equal to 740×60 pixels. Image colours are different to ones in chapter 3.1.1 Data Collection due different colormap chosen by the author, so details could be better observed.

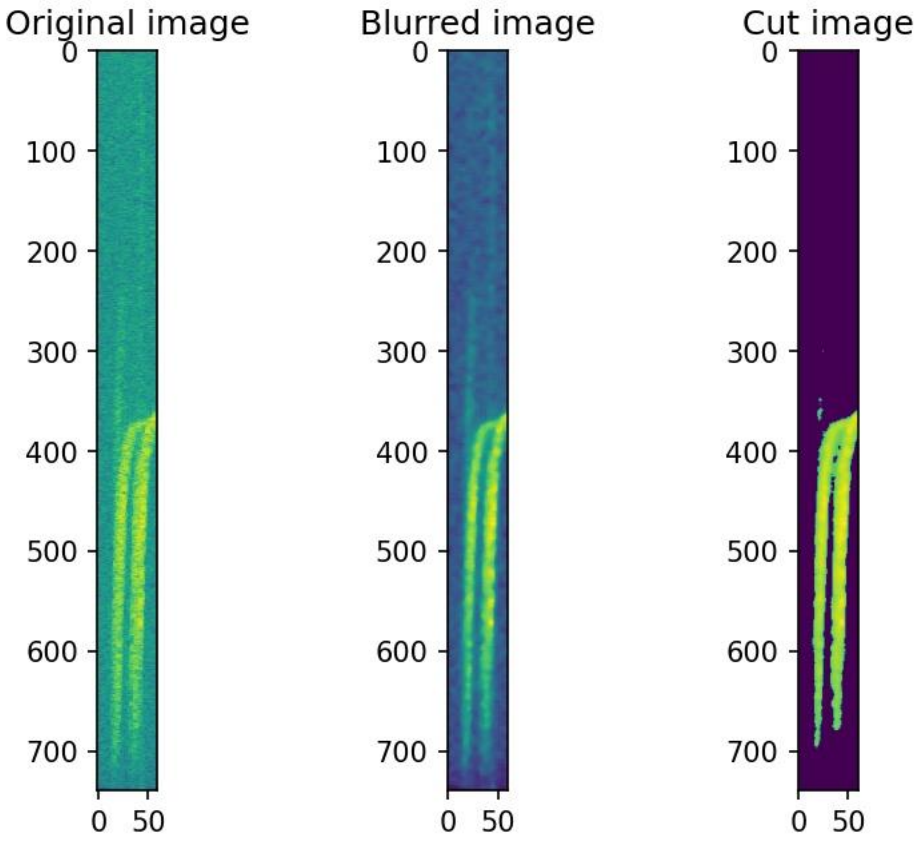

Figure 6 Image filtering to identify zones of interest

Then those images were converted to binary images and multiplied with each other. Multiplication result helped to identify whether figures present on images overlap and how much they do if so. It was also clear that there was no point in multiplication of all images by themselves since a lot of duplicated would be produced. The easiest way to illustrate this problem is shown in Figure 7, since multiplication of an image list by itself would result in a similar combination set.

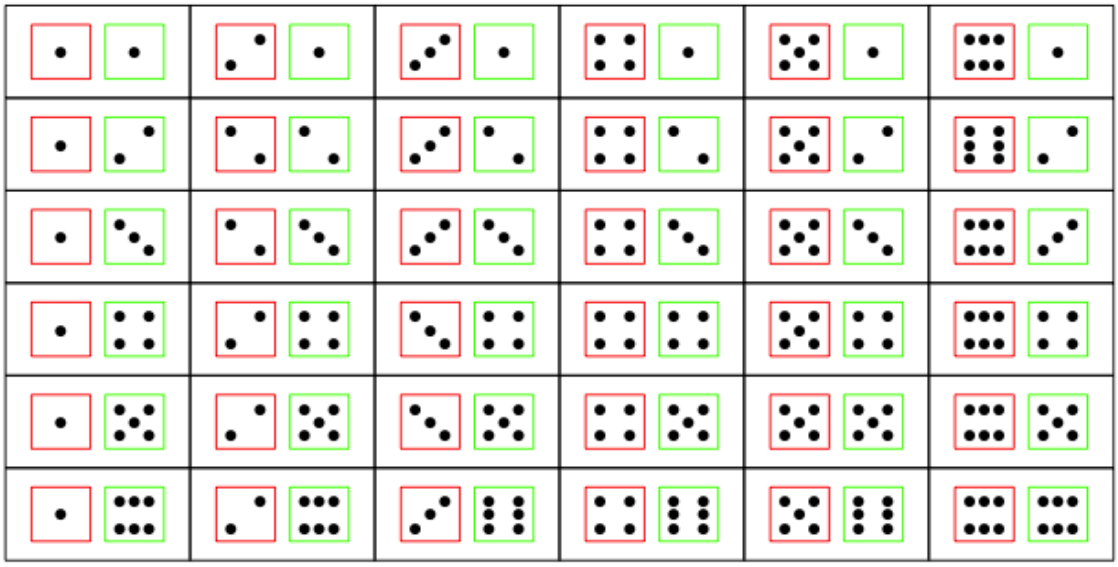

Figure 7 All possible combinations when rolling 2 dice [41]

So, as it can be observed, the main diagonal represents combinations of the same numbers. Below and above the main diagonal there are dice combinations resulting in the same product but having the opposite multiplication order. Translating that to image overlaying means that only a part below or above main diagonal offers an interest, since image multiplication by itself will result in the end in the same image. As well as the multiplication order is not important as the product is the same.

So, to overcome this issue an elegant solution was created by the author. Since multiplication was done in nested loops one iteration over image list and second iteration over its copy, an image was popped out of a copied image list before the second loop began. Thus, second list popping was done with every iteration of the first loop. Multiplication-wise the Figure 8 shows images plotted against the amount of their pairs.

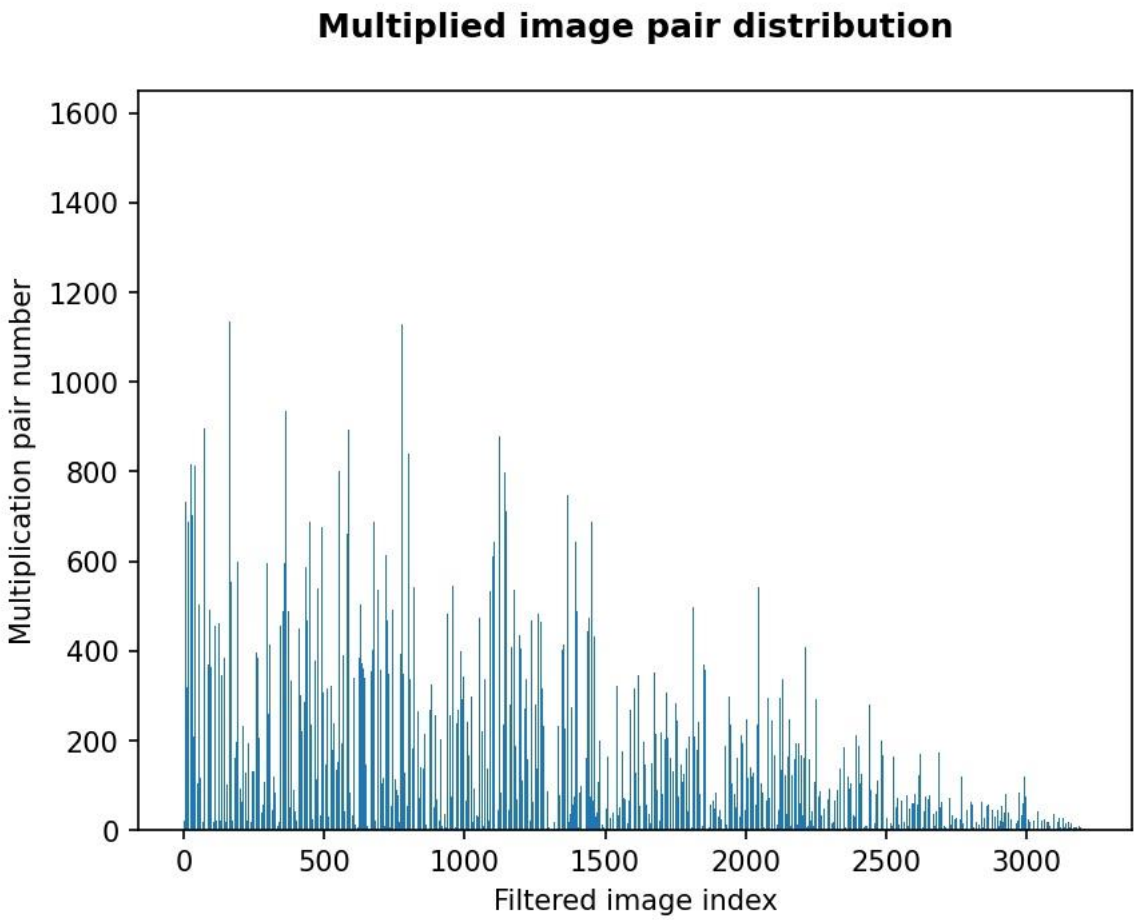

Figure 8 Image pair distribution

The bar chart present in the figure is not linear since all images must pass condition defined before. Nevertheless, a tendency of images having higher indexes to match lower number of pairs shows that designed approach fulfils its task.

The overlapping grade was inspected by checking sum of all figure pixels. In case images have passed the condition not to overlap and have sums of pixels greater than 700 their positional indexes and corresponding pair images positional indexes in image array were saved to the dictionary as key and items respectively.

Following images were randomly shuffled using seed 43 so each time shuffling was reproducible. It was important to shuffle in two steps, first, items and then order of keys in the dictionary, so items would still correspond to a certain key.

Further it was only required to overlap images and their corresponding pairs using PIL.Image.blend() method. As final number of generated images was several times greater than initial dataset, it was decided to set a limit on number of images being actually

saved. Thus, since the main purpose was to balance whole dataset, number of saved images was tuned to balance classes.

3.1.5 Dataset Building

When building a dataset, it very important to divide all pre-processed data into three groups. These three groups are called training, validation, and testing datasets. Training set is used simply to train the model. Validation set is used to validate the model. Validation is usually done via comparison of training and validation errors. More about this is described in chapter 3.2.5 Model Optimization (Loss function). Test set is used for model performance evaluation. Evaluation metrics are described in the chapter 3.3.2 Model Evaluation Process.

As addressed previously the structure and content of dataset has a valuable effect on neural network model performance. In order to have a efficiently performing model is vital to find and have balanced splits of training, validation and test sets. In [42] authors state that optimal size for training set is in the range of 40-80%. In [43] authors state it is a common practice to split data into sets bay ratio of 70%, 15% and 15% for training, validation and testing sets respectively. In [44] it is described that a training set must be at least 40% for convolutional neural networks to achieve good validation results. In [31] is recommended to have 80% of images for training, 10% for validation and evaluation. All three datasets must be independent have no inter-redundant data. Evaluation of neural network performance with the same data as in training set is useless [32].

As described in [32] in case of continuous data like video stream the data should be shuffled in a specific way to avoid duplicates and ensure consistency. However, when shuffling data and splitting it into sets data representativeness and traceability must not be violated.

It often is a subject to happen that the data is collected from different sources and locations. If there is some unintended bias in training data introduced neural network model may not generalize for some sources or locations since the data may be not so representative for specific ODD. This also concerns validation in some way and touches upon testing set. Therefore, it is recommended to perform validation and training on the same data as used during collection. The data in the testing set should be representative

for any ODD in scope of project realization and thus should include samples from every collection source [32].

3.2 Model Creation and Training

At the beginning of this chapter, it is necessary to remind the goals set for this thesis. The end solution had to be able to count vehicles, detect vehicle directions and in addition differentiate between three different vehicle types and do it with certain accuracy. Also as previously mentioned, this thesis was carried out on the technical basis of the solution presented in the framework of the project. The solution should fit and run on an embedded device that is attached to a street lighting pole, and it is natural that this kind of solution should be compact.

The results of this thesis might indicate that a how powerful embedded platform for classification might be needed. Since within the framework of the project the task was to create a low-cost solution, embedded platform and selected radar had their own technical limitations. To achieve these goals, it is important to select the right techniques and algorithms that would be able to perform their tasks most appropriately in the given conditions.

Vehicle type classification turned out to be a difficult task comparing to first two tasks, being dependent on the training data as any other machine learning algorithm. In given set of data there were two options how to solve vehicle type classification.

First option would be a creation of a single multi-class classification model that apart from classifying between vehicle travelling directions and counting these would need to classify between passenger cars and large vehicles. In total it would make up 25 different classes, which would require collection of a huge amount of data. Due to the high price of manual data collection, it would result in high price of a dataset. In case of proceeding with the amount of data collected during this project, some of 25 classes would be heavily imbalanced leading to model overfitting to some classes due to dataset limitations if even possible to train on. Other issue with this approach is contained in high risk decrease of model performance. Latter is important in case of embedded devices.

The second option is to divide the goals into two models. One for vehicle counting and direction estimation and the second one for vehicle type classification. Such an approach

is questionable in terms of integration complexity and resulting performance. Nevertheless, it is a plausible way to implement to test whether it is possible to distinguish vehicle types with CNN. However, since the model was training on a rather small amount of data, resulting model may experience classification difficulties. Author had decided to proceed with second option due to limited dataset.

Since the work of the author is planned to be used in an embedded device, this imposes restrictions on the proposed solution. For example, there are a sufficient number of ready-made neural networks that are widely used in smartphones. However, it should be clear that current smartphones have significantly more computing resources, as well as memory, in comparison with the battery powered embedded devices.

3.2.1 Introduction to Convolutional Neural Network

Despite the seemingly simple task, it turned out to be quite difficult to find a clear guide for choosing or comparing machine learning algorithms. Especially when compared to deep learning. Remembering that images are high-dimensional data, author turned to works considering high-dimensional data in machine learning. Deep learning methods are well performant in complex pattern or structure identification in high-dimensional data. As a result, deep learning outperforms other machine learning methods in image recognition [45]. However, it was necessary to choose a deep learning architecture before going on with practice.

It was difficult to find all or even majority of neural network types grouped in one place. However, in [46] is shown a chart of as author calls it most complete chart of neural networks, uncovering architecture designs. Deep learning includes architectures such as: convolutional neural network, recurrent neural network, distributed neural network, autoencoder and generative adversarial neural network. There is also provided a brief overview and typical application of mentioned neural networks.

Authors of [47] state that most popular types of neural networks are recurrent neural networks (RNN), recursive neural networks (RvNN) and convolutional neural networks (CNN). It is hard to compare those popular types directly, since many comparisons are task based and consider very specific problem. According to the review presented by authors main application areas of RNNs are speech and natural language related tasks, RvNNs are exploited also in natural language processing task. Same authors consider

CNN to be more performant than RNNs and report CNNs are applied in various field including computer vision. Authors of [48], [49], [50] also confirm that typical CNN implementations tasks are image related.

According to authors of [47] CNN usage advantages over two other neural network types mentioned in the previous chapter are as follows:

1. Implementation simplicity
2. High organized and high reliance on mode output and on features respectively

On the CNN property side authors of [47] and [48] highlight weight-sharing as well as outline a new property called sparse connectivity. Due to weight sharing CNNs are able to reduce network parameters, improve generalization and in some way, it even helps avoid overfitting. Sparse connectivity means that kernels are used for whole image. No need to learn parameters from every region, instead can learn group of parameters. Sparse connectivity can be described as each neuron has only few connections, which also saves memory. Having carefully studied the strengths and weaknesses of all, as well as weighing the recommended areas of application, the author decided to focus on the convolutional neural network architecture [51].

3.2.2 Convolutional Neural Network Architecture

Convolutional Neural Networks consist of convolutional layer, pooling layer, activation function, fully connected layer, and loss functions. A basic architecture of a CNN can be observed in the Figure 9. Input layer in this figure is an image matrix. Convolutional Neural Network

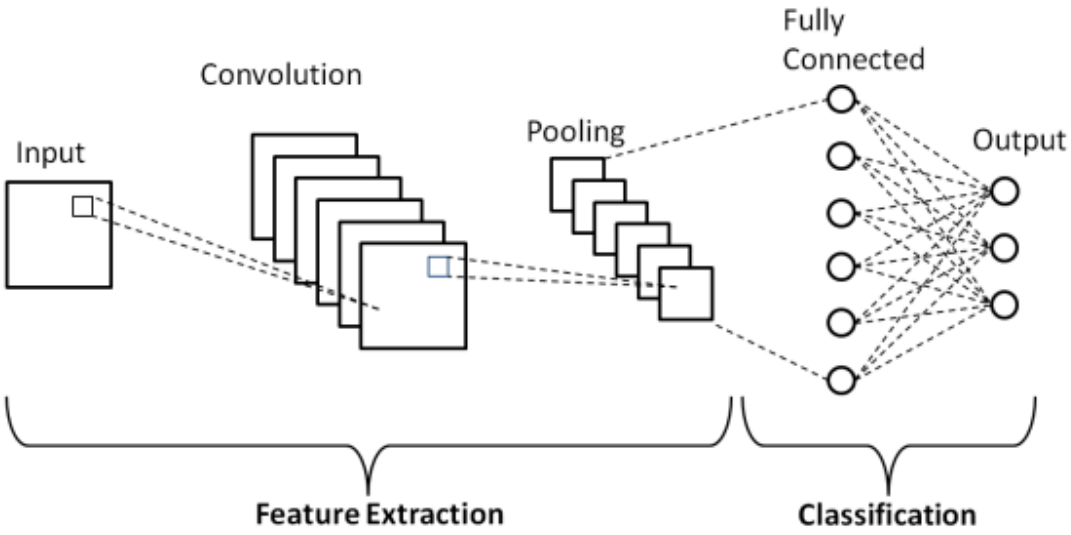

Figure 9 Convolutional Neural Network Architecture [52]

3.2.2.1 Convolutional layer

As it can be assumed by the name the most important layer of a CNN is convolutional layer. This layer is formed by convolutional kernels. Kernel is a 2-dimensional array consisting of integers, where an integer is a weight. These kernels move through an image represented by a pixel matrix with a step defined by a stride parameter. Sometimes padding is done to an image, so the information from border regions would not be lost during convolution. If padding is not done focus will stay in the centre, and features on the borders would not perform in less kernel computations as centre ones. Through training process these integers are adjusted from random numbers to weights helping to extract features [47]. Convolutional layer mainly deals with feature extraction. In TensorFlow there is Conv2D layer available acting as convolutional layer for 2-dimensional arrays [53].

3.2.2.2 Pooling layer

Pooling layers task is to perform down sampling of convolutional layer output without significant feature losses. The purpose of pooling operation is to helps CNN to focus on feature location (feature mapping). However, there is a risk to miss relevant information. Pooling layer does not learn anything since there are not used weights or/and biases. Pooling operation also helps to improve performance since the lower feature size is the higher becomes performance.

According to TensorFlow [53] there are following pooling types available in the library:

- average pooling (AveragePooling2D) – takes average value in a kernel of specified size
- max pooling (MaxPooling2D) – takes maximal value in a kernel of specified size
- global average pooling (GlobalAveragePooling2D) – takes global average value in a kernel of specified size
- global max pooling (GlobalMaxPooling2D) – takes global maximal value in a kernel of specified size.

Author of [54] states that in case object position is not important max pooling can be a better choice. Global average and max pooling are used in the task when it is important to reduce the size of a feature map and are applied instead of flatten or dense layers [54].

3.2.2.3 Activation Function

Activation function is usually placed after every element with weights. In context of CNN these layers are convolutional and fully connected. It is responsible for neuron activation and can take two Boolean values: true or false. Thus, it decides whether the feature is important or not [47].

TensorFlow a large variety of activation functions already available in the library. Author will describe the most popular of those (popularity was judged according to literature author went through during this thesis). Following, in Table 1 are listed most popular activation functions and corresponding description is given and in Figure 10 graphs of the same activation functions are illustrated.

Table 1 Most popular activation functions and their description [47]

Activation Function	Definition
Sigmoid	converts real numbers to rational numbers between 0 and 1
Tanh	converts real numbers to ratio between 1 and 1
ReLU	converts integers to positive numbers

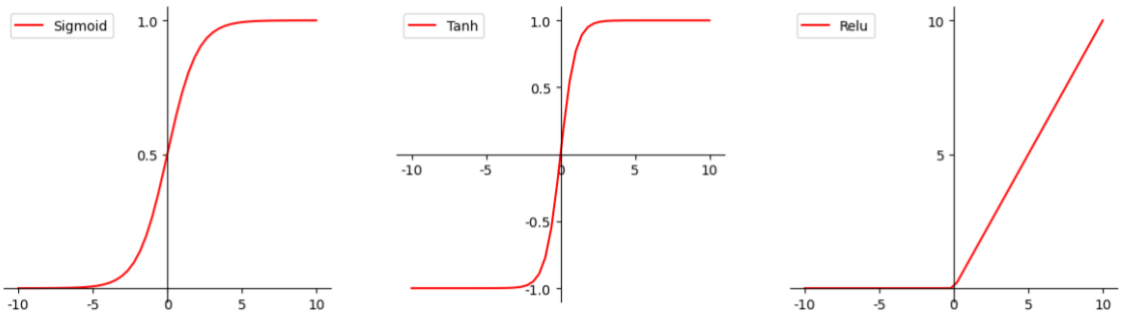

Figure 10 Most popular activation functions

ReLU is most widely used in CNNs. Its advantage is that it requires less computational resources [47]. Subtypes of ReLU are leaky and noisy ReLU, which are out of scope since are not offered in TensorFlow library. It was also decided to use ReLU as activation function due to wide use in CNN applications.

3.2.2.4 Fully Connected Layer

One may face fully connected layer very often to be the last layer in CNN. As it is stated in the name all the neurons of this layer are connected with every neuron of previous layer. Usually flatten or dense layers are used as previous layer. The purpose of flatten or dense layers is to bring feature map outputted by previous convolution or pooling layers into vector form. Fully connected layer is fed with extracted features in vector form and outputs predictions. This layers as convolutional layer has also weights and biases. Fully connected layer is likely to overfit, so it is recommended to be regularized [47], [48].

3.2.2.5 Loss Functions

Loss functions are also used in the last layer. Loss functions calculate the prediction error which is equal to distance between prediction and label. Further, model is optimized as described in chapter 3.2.5 Model Optimization (Loss function).

3.2.3 Model Regularization

There is a term known as bias-variance trade-off which mainly deals with finding optimal model complexity. High bias is responsible for underfitting and high variance for overfitting. Regularization deals with search of optimal model complexity [48].

Overfitting on the other hand occurs when neural network starts to memorize the data and performs very well on training data, however, fails on testing data. Overfitting problem

is often faced in deep learning. Underfitting is just simply when the neural network did not learn well enough and shows poor performance [48].

To cope against overfitting there, exist popular methods to overfitting reduction [48]:

- Data amount growth
- Regularization
- Model simplification (feature decrease)
- Data dimensionality reduction (where regularization cannot be done)

Data amount growth can be simply explained as increasing the dataset size. This can be done through new data collection or augmentation as described in chapter 3.1.4 Data Augmentation.

Regularization can be explained as penalty for overfitting (training too well). Following regularization techniques exist [47]:

- Dropping
    - Dropout
    - Drop-weights
- Data augmentation
- Model simplification
- Batch normalization

Dropout and drop-weight methods were added under dropping group as these contain similarities. The idea behind dropout is to randomly after each training iteration (user defined probability) drop neurons during training. Such an approach makes neurons equally act in to select and learn features; neural network becomes more general since does not rely on neurons activations so much. Ready network does not have dropout anymore [47], [48]. In case of drop-weight technique weight is preserved, however connections are cut. Both dropout and drop-weights are executed each training iteration and final model does not contain these layers (if implemented as a layer).

Data augmentation is widely covered in chapter 3.1.4 Data Augmentation. Batch normalization is done normalize inputs using Gaussian distribution and considered to improve training duration and allow choosing higher learning rate as well be less careful during initialization [55]. It can also regularize model and is available as a layer in TensorFlow and can be done after each layer.

Model simplification is self-explanatory. Model might overfit if it is too complex for dataset, since there is a lot of decision space in the model. Whereas data dimensionality reduction is also partly dependent from model complexity. Anyway, if there are too many classes to predict on model might overfit [47], [48].

3.2.4 Model Optimizers

Weight updating for error minimization of a neural network requires some kind of an algorithm that would utilize this task. Model optimizers deal with this task. According to authors of [47], [56] following are popular optimization algorithms:

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-batch Gradient Descent
- Momentum
- Adaptive Moment Estimation (Adam)

Batch Gradient Descent – weights are updated only once every training epoch. This technique requires a lot of resources and is mainly suitable for small datasets.

Stochastic Gradient Descent – weights are updated on each training sample. This technique faster and more efficient compared to batch gradient descent. However, due to its peculiarity of frequently updating weights is very noisy.

Mini-batch Gradient Descent – implements advantages of previously mentioned techniques – more efficient.

Momentum – once training reaches local minimum momentum helps to find the global minimum.

Adaptive Moment Estimation (Adam) – was specifically created for deep learning networks training and is cutting-edge optimizer. This technique is very efficient compared to listed above.

It was decided to proceed with Adam optimizer when configuring model metrics and losses. TensorFlow offers model.compile() method for this purpose. It is required to provide Adam optimizer with learning rate value. Learning rate was initially set to 1^{-5} according to authors previous experience working on MNIST dataset. According to [48] when choosing inappropriate learning rate cost function might overshoot the global minimum. So, this hyperparameter was tuned through the process. In [57] author shows practical approach to learning rate tuning.

3.2.5 Model Optimization (Loss function)

Loss function is used in supervised machine learning (and not only) for solving an optimization problem. Optimization problem can be simply explained as a search for best solution in set of given conditions among other possible solutions. Simplistically said when solving an optimization problem there persist a goal to find a such a value which will lead the optimization problem translated into minimized function or loss function to be minimal. Value during which a loss function is minimal is considered as best value [58].

It is also vital to understand the difference between loss and cost functions. Various sources like Wikipedia [58] state that loss and cost functions are synonyms therefore mean the same. However, author of this research has spent significant amount of time in terms of such a simple problem to identify that the is a difference between loss and cost functions. Loss function is a difference between predicted and actual values, also known as error. While cost function is a sum of loss functions. So, while judging by loss function values scientist can see if an error is being minimized, cost function can be used to evaluate a performance of a model over entire training set [59].

In supervised machine learning certain set of samples (training set) and those samples' values (labels) are provided to the learning algorithm which tries to connect those training samples with their labels (supervised machine learning algorithms catch to fit into data). So, during each training iteration algorithm tries to minimize loss function value [48].

Loss function is being used as a way to estimate prediction performance of a model during training. Generally, it is considered, that the lower the cost function value is the better. So, it represents the distance between predicted and actual value. There exist following types of loss functions [59]:

- Regression loss functions (example: Euclidean)
- Binary classification loss functions (example: Hinge)
- Multi-class classification loss functions

Since in terms of this research a problem of multi-class classification has been solved further are presented loss functions typical for abovementioned problem. Thus, multi-class classification loss functions are s designed for labels in one-hot encoded format and sparse categorical cross in a integer format.

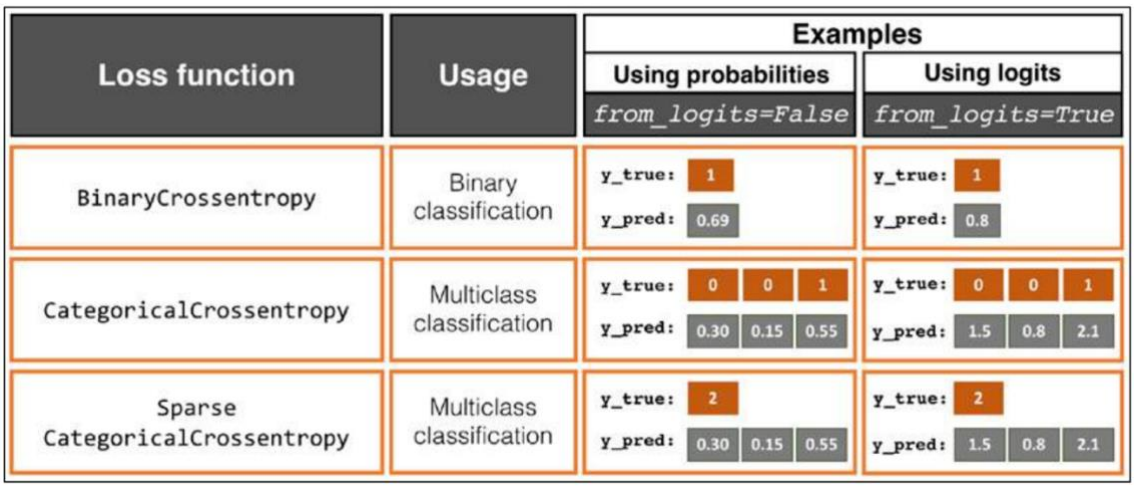

Figure 11 Loss function application [48]

However, according to [60] on the practical side an author of a model is often limited to libraries person is working with. Author of the article also highlights that a loss function is considered good if it works efficiently in machine learning algorithm. So, some authors end up with those functions that are easier optimization-wise. As well as is being pointed out that model evaluation metrics should be considered first in model evaluation since the purpose of a loss function lies in mode optimization. Thus, the model was developed in TensorFlow library, the author of this research was limited to tf.keras.losses module and was keeping in mind mentioned in this chapter.

TensorFlow model.compile() method also requires loss function as input when configuring model for training. Based on mentioned above categorical cross entropy was chosen.

3.2.6 Counting vehicles

As it was found out in 2 Related Work camera-based solutions to traffic monitoring are popular. One of the reasons is ease of development. A typical approach could be implementation of object detection NN and setting some region of interest, so a vehicle is counted when a vehicle enters specified region. The current system handled in this thesis differs from camera system in many ways, however there is a significant difference from vehicle counting viewpoint. Whereas on camera stream vehicles travel in lanes and there is often a gap between vehicles, which is reasonable to use for most accurate vehicle detection. On current radar system images vehicles travelling in opposite direction are in one plane and are likely to overlap. There was lack of confidence that an object detector will handle complicated overlapping cases. Additionally, object detector creation is a long work since there are no ready-made models pretrained on similar datasets. Nevertheless, the main concern of current work was object detection model performance since object detector not only requires classification being done but also object localization done first.

Thus, it was decided to proceed with vehicle classification also to count vehicles. It was already defined by ODD that the developed system itself is not designed for dense traffic routes. As it can be seen on the Figure 2 some figures overlap, making it difficult to observe these as some number of different cases. The approach was to consider there could be up to six vehicles concurrently (three for an each travelling direction) on a one six-second-long time frame. Thus, three vehicle constraint means that in case there get more than three overlapping vehicle figures travelling into one direction on a one six-second-long time frame, these would be considered as three vehicles by design. This decision is due, among other things, to the fact that hitting four or more cars in a time frame is extremely rare.

Number of vehicles per travelling direction was considered as class. Thus, adding a case when there are no vehicles present on an image there will be eight classes in total (empty class is taken for each travelling direction). Since in the initial scope of this thesis was performing vehicle counting on static images (frames) there is no solution proposed for labels that get in between of two consequent frames.

3.2.7 Model Creation and Training (Vehicle Counting and Direction Detection)

The next step was a neural network model creation. The development was done in python programming language using mainly TensorFlow, Pandas, NumPy and OpenCV libraries. TensorFlow website provides a great tutorial [61] of related information as well as real use examples.

Given the theoretical knowledge gained during research, author began with neural network creation model (train_ver_baseline_v5_classic_arch.ipynb [16]). As soon as it was clear with input format author decided to create the architecture. 1 channel was chosen as input since dataset images were monochrome.

During training, the author had experimented with various methods to increase the general accuracy and decrease size of a model. As a baseline there were chosen common practices and default values learned in chapters 3.2.1 Introduction to Convolutional Neural Network to 3.2.5 Model Optimization (Loss function). Several different neural network designs were considered during the model creation phase. Among them were both custom and developed designs like MobileNet. Majority of already developed designs were large so they would barely work on embedded device. Some had restrictions such as requiring a specific input image size.

It is essential to point out that modelling a neural network is a task requiring thorough knowledge of the topic. Custom designs included experimenting with and without dropout layers, average or average global or maximal pooling layers application. At first steps, self-created designs were less efficient and larger than pre-designed solutions. Nevertheless, the author had used as a standard CNN architecture (Figure 9) as a baseline. The sequential structure was formed of convolutional and max pooling layer pairs. In current case there were five consequent layer pairs where the last pair was connected to flatten layer which as previously described converts feature maps into vector form. Five pairs are chosen since it is a maximal amount of convolutional max pooling pairs that could be applied for the image size of the dataset (taking into account kernel size, stride size and padding type). Following the dense layer connected to flatten layer is connected to two parallel dense layers. This split is performed so to categorize left and right moving vehicle predictions. The structure can be observed in Figure 12.

There were various efforts concerning architecture to train model with imbalanced classes, however the result was pretty much the same every time: model was generalizing too well for dominant classes and experienced difficulties classifying on classes with two or three vehicles. Google Cloud [31] recommends avoiding training with imbalanced data and if so, claim training is problematic if there is more than ten times difference between number of class instances. This can be confirmed by one of the results in Figure 15, where three vehicles are not identified at all.

Author considers worth mentioning the results acquired by using different hyperparameters and changing the architecture. While Figure 12 illustrates baseline CNN structure, the baseline hyperparameters are listed as follows in Table 2.

Table 2 Hyperparameters chosen as a baseline for vehicle counting model

Hyperparameter	Value
Learning rate	1^{-5}
Batch size	32
Epochs number	500
Drop rate	0.33
Activation function	ReLU
Kernel size	5x5
Strides	1x1
Padding	Same

Following will be described what results were acquired when tuning hyperparameters listed above and if author decided to proceed with those. It is important to mention that the random seed was fixed when shuffling the dataset. That means the order of images remained the same each time.

At first it was decided to observe how dropout regularization may influence the training process. There are two options how to proceed with dropout. One way would be adding more layers and second one would be increasing dropout value. To implement the first way author added dropout after each convolutional and padding layer pair. It did not result in any valuable result since model started recognizing three vehicles travelling left with increased precision. However, at the same time classification of two vehicle moving to the left was done with lower precision. Other than that, the results were comparable. Increasing dropout size to 0.5 did not bring any valuable result as well. The second way was just to increase dropout value from 0.33 to 0.5 in single layer of baseline CNN model. And the model showed slightly increased accuracy and precision during evaluation. No noticeable changes were introduced when increasing dropout value to 0.75.

Learning rate reduction to 1^{-6} resulted in less abrupt accuracy and loss jumps compared to 1^{-5} learning rate and required more epochs to reach to stable values. However, during the evaluation model showed less precise results on a training set. Learning rate equal to 1^{-4} on the other hand did not provide any good results. Model loss and accuracy remained almost constant throughout the training process, and it was able to predict only the empty class classifying every other class as empty.

The kernel size of the convolutional layer was reduced to 3×3. As a result, accuracy and loss values spiked more frequently. Additionally, padding being changed from same to valid had no effect on accuracy. The number of convolutional and padding layer pairings had to be lowered to four after changing padding from same to valid. Precision and recall values revealed that model overall performance did not change. Compared to baseline CNN model both models still made lots of classification errors, which is consistent with 80% accuracy. To try with larger kernel size, it was necessary to reduce the convolutional and padding layer pairs to four in order to increase the kernel size to 7×7. Model did not reach optimized level.

Changing stride value to 2×2 required to reduce convolutional and padding layers to three and resulted in accuracy about 80%, however the model showed 10% precision on class representing two vehicles travelling to the right.

Although in [54] it was recommended to replace flatten and dense CNN layers with global average pooling layer, author was not able to achieve any superior result by changing

structure as recommended. The result was rather inaccurate, and it was tried to increase and reduce learning rate as well as tune dropout value, however models did not optimize to comparable to baseline structure level.

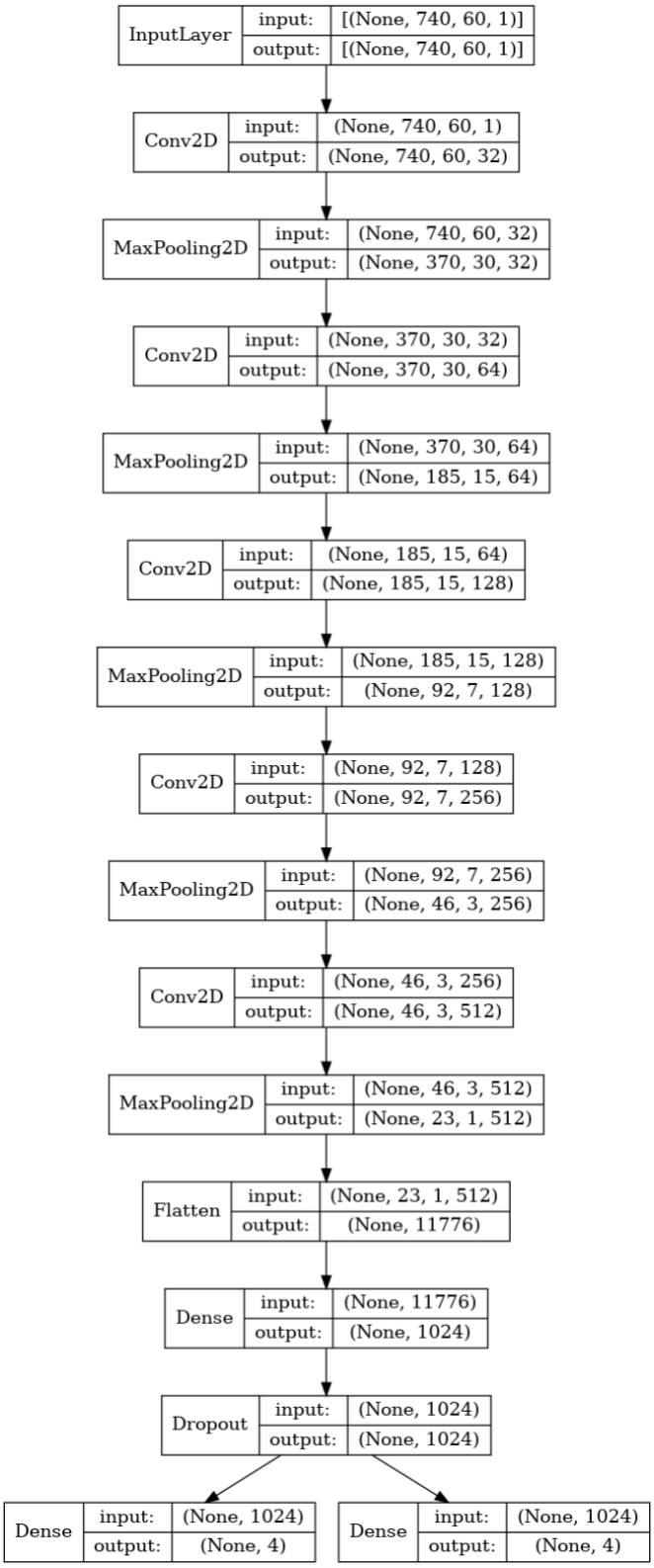

Figure 12 Baseline model architecture

Considering the results mentioned above it was decided to proceed with the network structure as shown in Figure 12 and among all hyperparameters show in Table 2 increase the drop rate to 0.5.

3.2.8 Vehicle Type Classification

Differentiation between vehicle categories belongs to the problem typical to classification, since a classification task can be stated as a task of identifying what class an observed object belongs to [62]. Since datasets and tasks differ, there is no single algorithm capable of solving a particular problem. In view of this, it was necessary to choose the most appropriate method of classification.

As mentioned in chapter 3.2 Model Creation and Training since the dataset was not balanced author had decided to create a separate model for vehicle type classification. The idea was to create a proof of concept that vehicle types could be classified by CNN using spectrogram images of radar data. Besides passenger car and large vehicle images there were two separate images each containing a van and a tractor in collected dataset. The author chose to omit the last two because they were both underrepresented and weren't the goal of the thesis. Since data contained samples of vehicles travelling in both directions it was decided to address direction in the model as well. The dataset was manually filtered to images containing only large vehicles or passenger vehicles. In given ODD the chance of getting two large vehicles one after another is considered very low. Therefore, it can be said that the main task of the model would be to distinguishing between large vehicle or not a large vehicle, since mostly other vehicles passing by the radar will be passenger cars anyway (as classification is done between a large vehicle and a passenger car). Thus, for simplification reasons author had decided to choose for model training images where only a passenger car or a large vehicle were present.

Since together with augmentation author managed to get about 30 images containing only large vehicle instance, passenger car image amount was narrowed down to the same amount to keep data balanced. About 60 images were divided into three sets: eight for test set, eight for validation set and rest to training set.

3.2.9 Model Creation and Training (Vehicle Type Classification)

3.2.9.1 Direction Detection and Vehicle Type Classification Model

In following, author had begun with model creation (train_ver_veh_class_v3.1.ipynb [16]), which had to classify between four classes consisting of vehicle type (large vehicle or car) and travel direction (left or right). Model structure that can be observed in Figure 13 was chosen as a baseline to begin with.

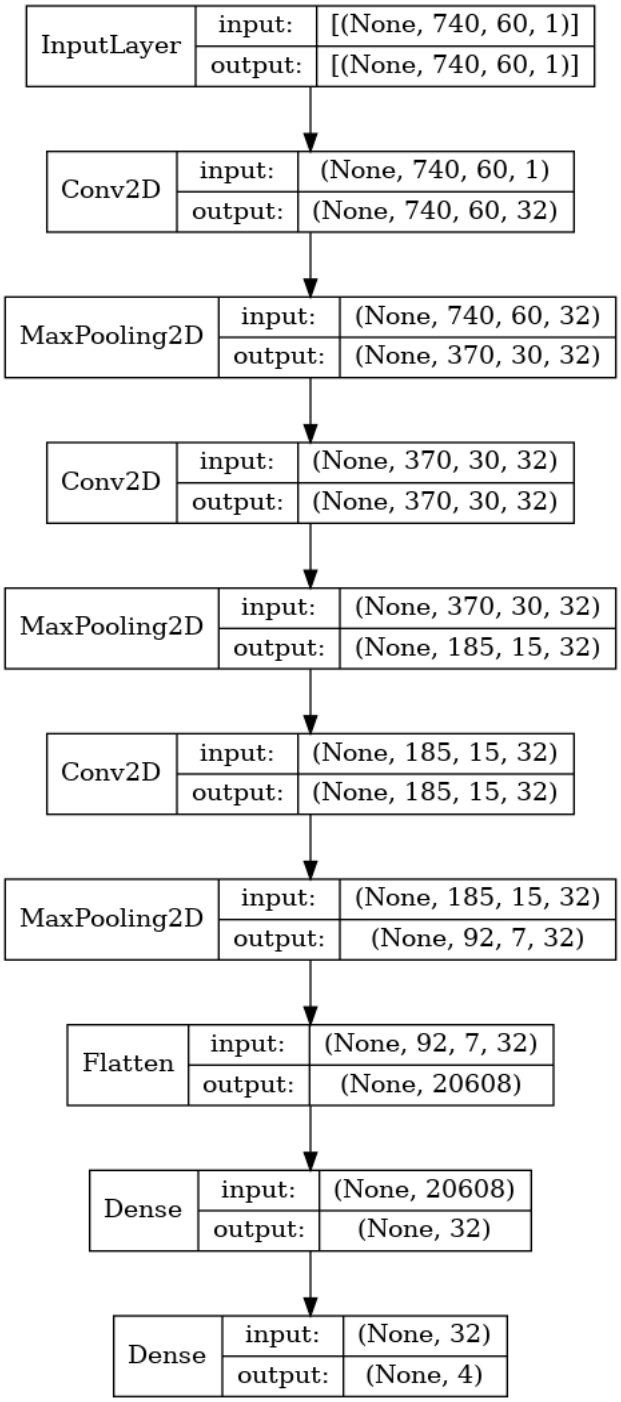

Figure 13 CNN structure for vehicle type classification

Even during early stages of hyperparameter tuning it became clear that 100 epochs should be considered as maximum for training this model. Having previous experience with model created in chapter 3.2.7 Model Creation and Training (Vehicle Counting and Direction Detection) author picked similar hyperparameters as a baseline adjusting only those which from author's viewpoint would be unsuitable. Epochs number higher than 100 would lead to overfitting. Dropout was also set to 0 since it would decrease accuracy on small dataset [63]. In Table 3 baseline hyperparameters could be observed.

Table 3 Hyperparameters chosen as a baseline for vehicle type detection model

Hyperparameter	Value
Learning rate	1^{-5}
Batch size	16
Epochs number	100
Drop rate	0
Activation function	ReLU
Kernel size	5x5
Strides	1x1
Padding	Same

Dropout layer was left out since the model could not generalize when it was regularized with dropout. It was also tried to conduct training with batch normalization layer, which required also tuning learning rate parameter. However, CNN models trained with the same hyperparameters showed different evaluation results each time. Since the dataset for vehicle type classification was significantly smaller than for vehicle counting and detection estimation task it was decided to change batch size to optimize training speed.

The training was conducted with various batch sizes such as 8, 16, 17, 18, 19 and 32. Apart from training duration changes author did not notice any correlation between the

batch size and model performances during validation. Batch size equal to 16 remained being optimal for training speed.

Most of the time author had spent for learning rate tuning since it was primarily influencing the training results. Many learning rates in between 1^{-4} and 1^{-7} were tried out. Learning rate equal to 1^{-4} turned out to be the most optimal in this task judging by the results achieved during model evaluation. As an optimizer it was decided to choose Adam due to its efficiency. Categorical cross entropy was chosen as loss function. label_smoothing parameter of this loss function TensorFlow representation was set to 0.0 since author designed the model to be confident on the predictions. This was confirmed during model evaluation since model optimized better and showed higher accuracy and precision on a test set.

Author had also tried different CNN structures. As it can be seen in Figure 13 there are three pairs of convolutional and max pooling layers in the structure chosen as baseline. Structure was modified to contain one, two and four pairs of these layers with each time doubling output filters value in convolutional layer. The lower number of pairs the more model was overfitting, the higher number of pairs the more model was underfitting. Thus, three pairs of convolutional and max pooling layers stayed as optimal option.

3.2.9.2 **Vehicle Type Classification Model**

Details on model evaluation will be later shown in chapter 3.3.2.2 Evaluation of vehicle type detection model. However, based on evaluation data it was decided to try without direction detection and classify only between large vehicle and passenger car. The idea behind doing so was based on observation that CNN model was able to differentiate between vehicle types but had troubles with direction detection. Hyperparameters were already tuned in chapter 3.2.9.1 Direction Detection and Vehicle Type Classification Model, so it was only needed to adjust the code.

It was tried various ways to achieve accurate and precise training results. Unexpectedly, author got lower accuracy and precision than with direction as two additional classes. To overcome this learning rate was adjusted between 1^{-4} and 1^{-7} and the number of consequent convolutional and max pooling layer pairs was changed to one, two, three and four. But unfortunately, models had high inaccuracy when classifying vehicle type.

3.3 Model Evaluation

3.3.1 Model Evaluation Metrics

There exist various methods on neural network performance evaluation. Since most of the development was done with TensorFlow library and know-how was read alongside with development. TensorFlow seemed a first source of valuable information regarding evaluation methods.

Thanks to a "Training and evaluation with the built-in methods" [64] author got knowledge on general performance metrics as well as realization techniques of those with build-in tools.

Initially, during first stages of neural network development the history object served as the main mean to access training loss and metric values and evaluate() method was called to evaluate the model based on test set. Gaining confidence in the model architecture, author got acquainted with such metrics as AUC, precision and recall, true positives.

Author have also read several articles on model evaluation metrics generally popular in machine learning. Many authors prefer to divide these articles into several segments beginning to describe from classification related metrics, then describing regression related metrics and going into more specific applications like ranking related metrics. It was sightly unclear whether regression related metrics could be applied with the scope of this thesis. However, after reading literature addressed to this problem like [65], [66] it became clear that although classification and regression address similar problems from authors viewpoint, practical difference between these two lies in the fact that classification result are categories or some discrete values while regression result is some continuous value [65]. Since regression problem is not related to this thesis classification related metrics stay exclusively applicable to this thesis. Scikit learn provides a great overview of classification related metrics [67]. However, further are provided popular classification related metrics according to [66], [68]:

- Confusion matrix
- Accuracy
- Precision

- Recall
- F1-Score
- Receiver Operating Characteristics (ROC) Curve
- Area Under Curve (AUC)

Before figuring out what is the application area and case of these metrics author had to become acquainted with 4 new terms, since misunderstanding of these may be misleading in understanding of the logic behind and use of abovementioned metrics. So, these definitions are true positives, false positives, false negatives, true negatives. True positives stand for the correct predictions, while false positives are incorrect predictions identified as true to class. False negatives are incorrect predictions as false to class and true negatives are correctly predicted as false to class. The matrix of choices in Figure 14 is illustrative and provides an overview of terms mentioned in this paragraph.

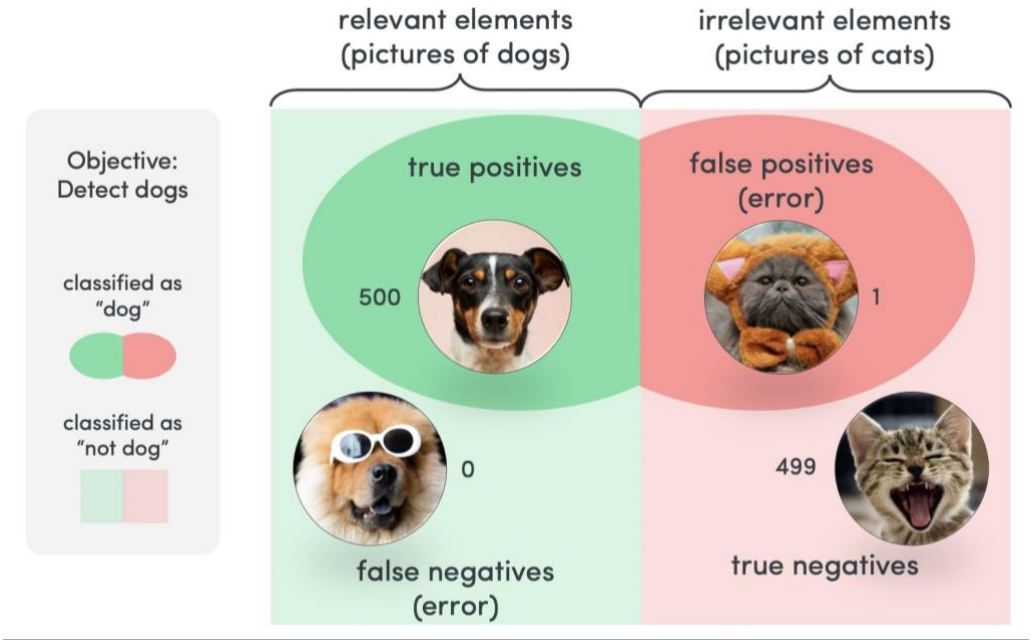

Figure 14 Matrix of choices [69]

It is vital to mention here that confusion matrix is not technically a metric. It rather helps to understand in visual way a class distribution between predictions and ground-truth labels as well helps to observe false guesses and their distribution. Example of confusion

matrix sampled in one of the early stages of model development could be observed in Figure 15. Explain where predicted and actual data is in the figure

Accuracy metric is pretty much self-explanatory. In machine learning accuracy is calculated as ratio between correct predictions and all predictions multiplied by 100.

Precision is a metric showing the amount of relevant (correctly assigned to class) predictions relatively to all positive predictions and is calculated as a ratio between true positives and all positives. Precision is considered to be helpful in problems where is necessary to determine imbalanced class distribution. Recall shows the number of relevant predictions and is calculated as a ratio between trues positives and relevant (true positives plus false negatives) [69], [70] . Precision and recall values of one of the models during early stages of development can be observed in Table 4. F1-Score term is densely bound with precision and recall since it a harmonic mean of these [71].

ROC curve shows performance of a classification model as function of its classification threshold. AUC is related to ROC curve graph and measures the are under ROC curve. AUC provides a summary measure of performance across all possible classification thresholds. One way to interpret the AUC is as the probability that the model places a random positive sample higher than a random negative sample [72].

3.3.2 Model Evaluation Process

3.3.2.1 Evaluation of vehicle counting model

In the very beginning of development when there was no augmented data, the model was evaluated on left travel direction labels. Later it became clear that it would be wise for both directions. With self-developed architecture, it was possible to achieve 81% accuracy on the test set. And it clearly explains why it is wrong to purely rely on one evaluation metric. In Figure 15 it can be observed the model did not correctly classify three vehicles.

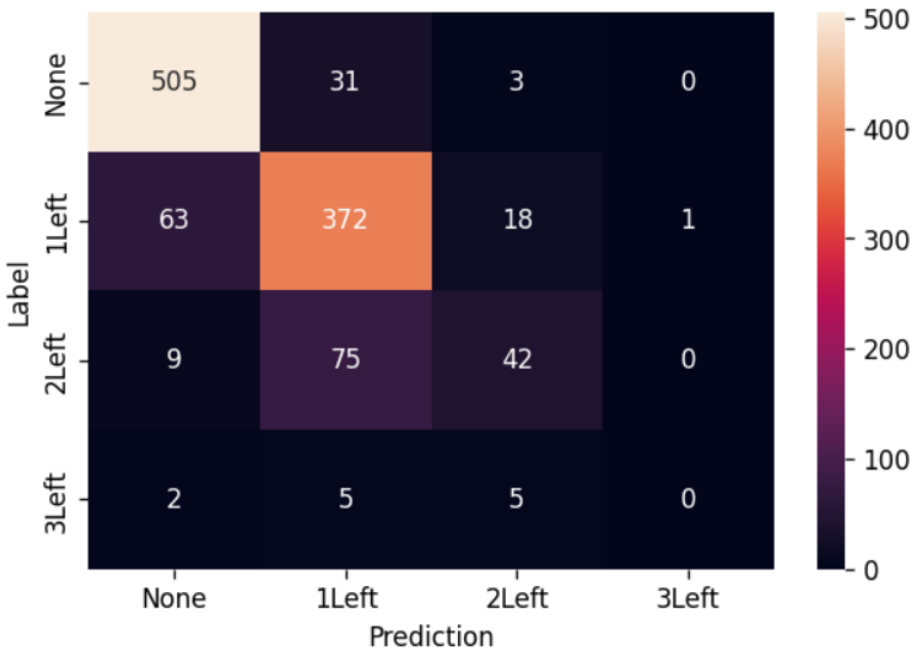

Figure 15 Confusion matrix for vehicles coming from left (beginning)

It can already be observed in a Figure 15 that accuracy drops with an increase in vehicle count. Therefore, the mode evaluated on other parameters such as recalls, precisions. Precision and recall confirm results observed in contusion matrix. As it can be seen in Table 4 model classification capacity drops when the amount of approaching cars raises.

Table 4 Precisions and recalls for vehicles coming from left

Metric/Label	None	1 Left	2 Left	3 Left
Precisions	93.69%	81.94%	33.33%	0.0%
Recalls	87.22%	77.02%	61.67%	0.0%

On the other side when dataset was balanced the results became significantly better. At this point author also started doing evaluation for both vehicle travelling directions. Test set accuracy was 90% on left movement direction and 92% on right moving direction. Training and validation accuracies as well as losses was also plotted the graphs. As it can be seen in Figure 16, Figure 17 (accuracy plotted against epochs numbers) and Figure 18 (loss plotted against epochs number) validation is noticeably oscillating. Author has initially thought it could be a learning rate related problem since dataset size was already increased by augmentation. This behaviour is assumed to happened due to differences

between training and validation sets. Other issue is a gap between training and validation accuracies/losses, which might indicate overfitting happening. Nevertheless, the network model converges, and result is consistent with accuracy goal.

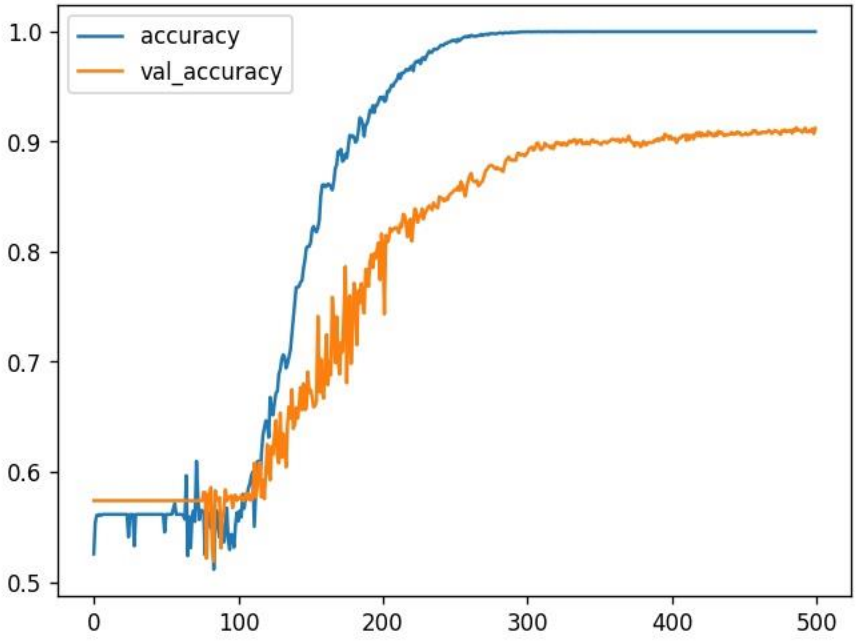

Figure 16 Training and validation accuracies (vehicles from left)

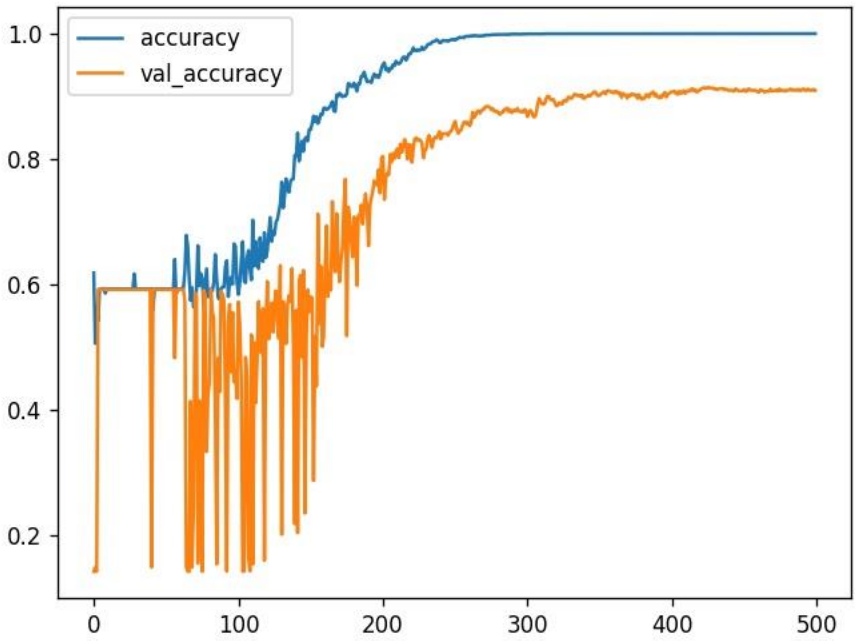

Figure 17 Training and validation accuracies (vehicles from right)

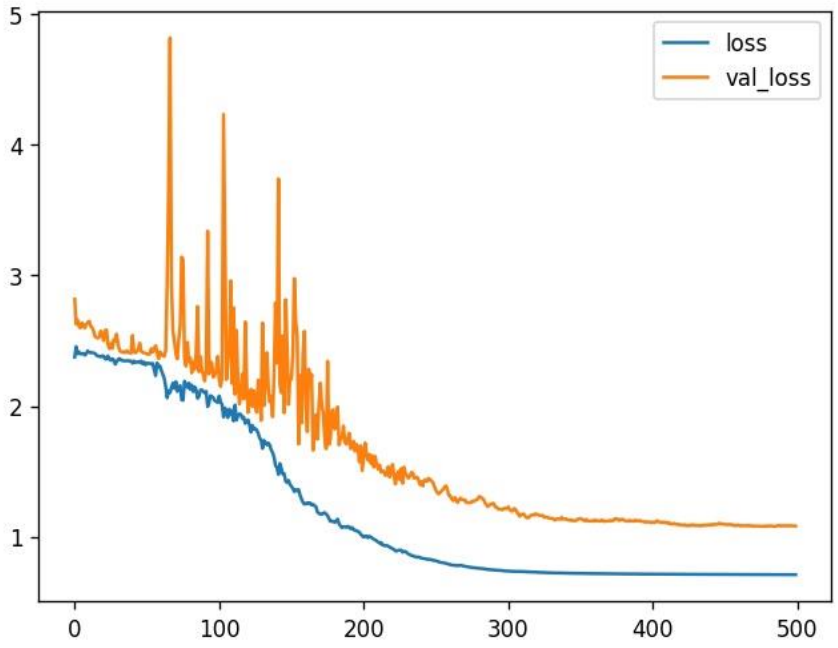

Figure 18 Training and validation losses

Following in the Table 5 and Table 6 precision and recalls for vehicles approaching from left and right respectively can be observed. Precision and recall values are significantly higher.

Table 5 Precisions and recalls for vehicles coming from left (balanced dataset)

Metric/Label	None	1 Left	2 Left	3 Left
Precisions	99.59%	77.56%	80.44%	76.96%
Recalls	95.44%	87.12%	73.65%	90.06%

Table 6 Precisions and recalls for vehicles coming from right (balanced dataset)

Metric/Label	None	1 Right	2 Right	3 Right
Precisions	99.35%	79.08%	76.73%	88.35%
Recalls	96.19%	90.1%	82.45%	82.5%

It can be also confirmed when looking on confusion matrixes in the Figure 19 and Figure 20. There are still classification error present, however classification is done more precise compared to the previous result in the Figure 15.

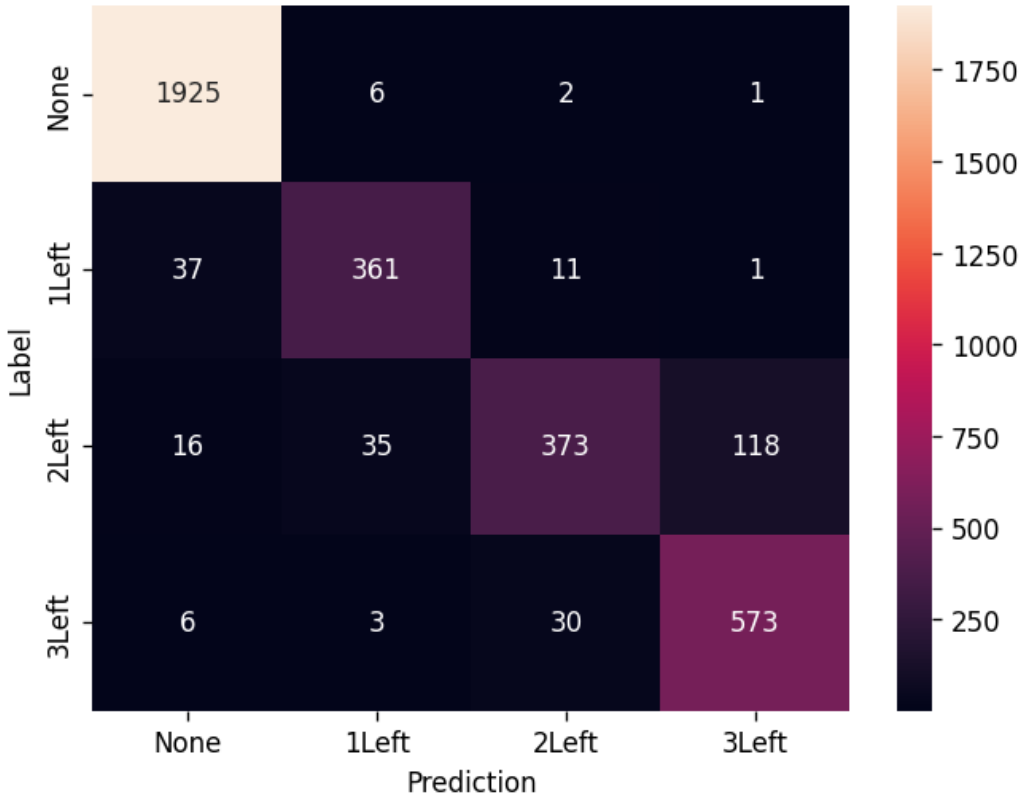

Figure 19 Confusion matrix vehicle coming from left

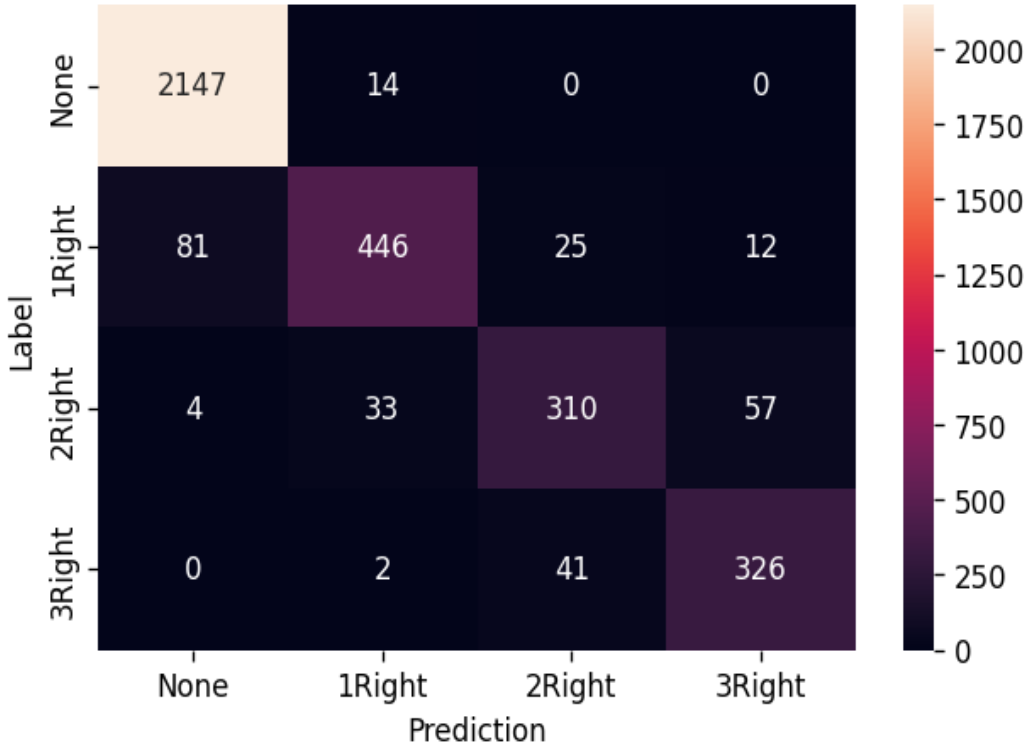

Figure 20 Confusion matrix vehicle coming from right

3.3.2.2 Evaluation of vehicle type detection model

Vehicle type detection model evaluation was limited due to test set size and is not precise due to this limitation. As mentioned earlier test set was formed out of eight images of large vehicle or passenger car. Even though having such a small test set, some interesting results were observed. Most of trained models delivered not repeatable results apart from the latest. 75% accuracy on a test set was considered a rather good result. Table 7 illustrates precisions and recalls for developed vehicle type and direction detection model. Values for "Large Vehicle Left" class are missing since it was not present in a test set.

Table 7 Precisions and recalls for vehicle types

Metric/Label	Large Vehicle Left	Car Left	Large Vehicle Right	Car Right
Precisions	-	100%	100%	0%
Recalls	-	50%	100%	0%

Figure 21 shows confusion matrix for vehicle type and direction detection model. L stands for vehicle travelling to left direction and R analogically to right. Among various developed model designs there was a common similarity. Models distinguished well

between vehicle types, however had troubles with determining vehicle travelling directions.

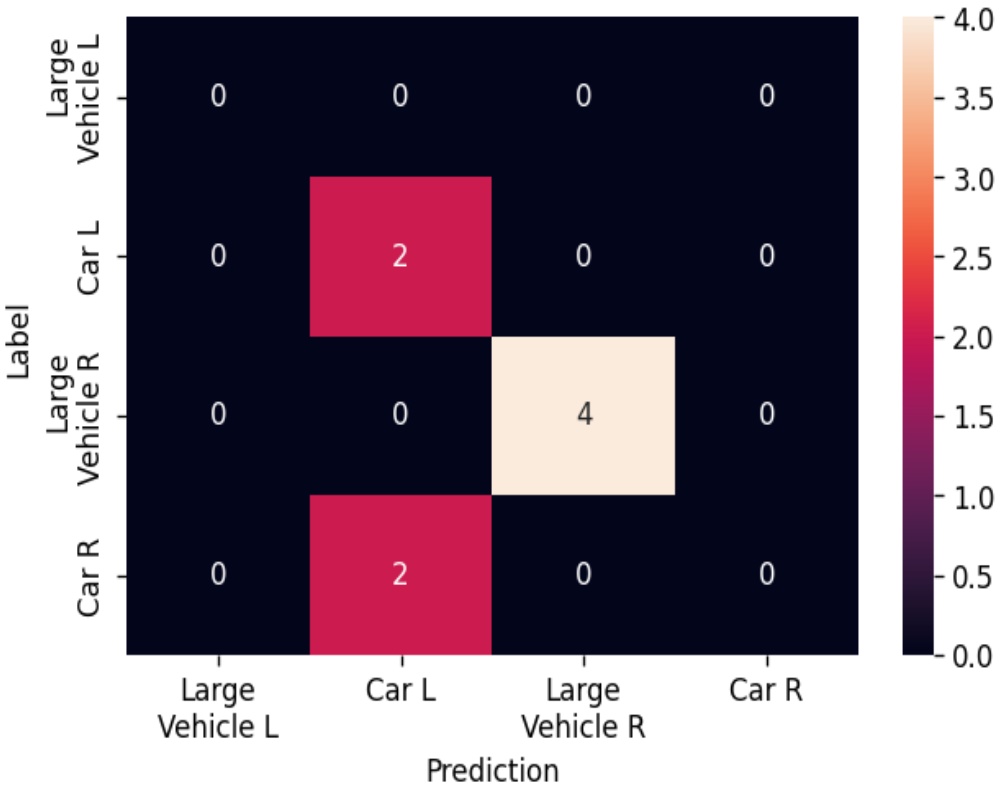

Figure 21 Confusion matrix of vehicle type detection

Due to model inaccuracy in vehicle travelling direction detection, author decided to try creating a model that would only classify vehicle types. However, model did not optimize.

3.3.2.3 Model choice

Although it was not known at the time of development which embedded device will the created model run on, author considered to take care of that as much as possible. TensorFlow Lite is designed to help developers with integration of created models on mobile and embedded devices. Model integration to an embedded device is not a part of this thesis and is a part of future work; however, the model should be theoretically executable on a battery powered embedded device. TensorFlow Lite for microcontrollers and embedded systems was considered as well, however it has a set of limitations on choice of microcontroller as well as model development and execution side.

As far as author is concerned there are several ways how to evaluate model ability to fit an embedded device. These include model size, meaning how much device memory will a certain model use, computational performance, and power consumption. The last two

points were difficult to estimate without a decision made on an embedded device, therefore it was decided to concentrate on memory usage even though the size of memory could be only estimated.

The best solution in this case seemed to take models that have delivered the desired accuracy and showed better metric values listed in chapter 3.3.1 Model Evaluation Metrics compared to other trained models and fulfilled the goals. Then convert each of these TensorFlow models into TensorFlow Lite and then pick a best one. To pick the best one author needed to compare model scores to weight. The size of finally selected vehicle counting and travelling direction detection model is 65.7MB when converted to TensorFlow Lite format. Vehicle type and direction classification model weight is 192.5kB.

4 Summary

Within the framework of SmENeTe project there were developed and deployed 250 low-cost radar sensing systems solutions in Tallinn. The software developed for these radar systems during the project is capable of counting vehicles and detecting their speed, however, suffers from low signal to noise ratio. Authors of [1] have raised and proposed a hypothesis that a CNN implementation might improve vehicle counting accuracy as well as allow vehicle type and vehicle movement direction detection, by virtue of neural networks having high tolerance to noisy data [15], which is highly attractive considering radar noisiness. Thus, the purpose of this thesis was to develop a CNN model that would confirm the hypothesis using collected data from 9.9GHz low-cost microwave radar.

Author faced many difficulties throughout the research and development process. Finding answers to some, especially data related questions, was difficult and required a lot of effort. This is due to the machine learning field peculiarities that do not have general best-practices for various problems and keys to the problems are usually established in the process. However, this issue was overcome, and all vital answers were established.

Any machine or deep learning model is highly reliant on data since it is being trained on it. Therefore, dataset quality is one of the key factors to success in machine learning field. Due to highly imbalanced classes and modest dataset author had to create a data augmentation solution suitable for current task. It consisted of image mirroring to double the size of a dataset as well as image overlaying to balance the dataset. Such an approach was effective since it helped to overcome the issue when the model was generalizing only for the dominant class.

The search for the model structure that could deliver accurate and precise result and be lightweight required thorough theoretical background research and besides a lot of trial and error. Inspired by popular convolutional networks like MobileNet author began development with complicated designs. However, the need for a compact design allowed author to stick to much simplified and more classical design, which is more likely to fit a compact embedded device.

To understand whether a certain model performs well and to score it author used a test dataset to get such metrics as accuracy, precisions and recalls and confusion matrix. Evaluation methods were developed continuously, though in the very beginning author evaluated on accuracy and loss curves as the model design developed, need for more complex metrics arose. Additionally, the author learned the value that a particular metric provides.

Although it was not technically possible to achieve vehicle type detection goal within one model due to limitations imposed by the dataset, no class in the provided vehicle counting and travel direction detection model has a precision or recall lower than 76.73% or 73.65% respectively. The best was recognized class containing no vehicle instances. In addition to the vehicle counting and direction determination model the vehicle type detection model was created as a proof of concept. The vehicle type detection model performed well only in case of single vehicles and demonstrated clearly that more training data was required to achieve necessary accuracy.

Summarizing the work done for this thesis author can state that this thesis shows the potential of CNNs in combination with low-cost microwave radar to count vehicles, determine their direction and type. The goals set in the beginning of the thesis were achieved. Author considers that both developed models show impressive results taking into account the compactness of those. It is confirmed that vehicles could be counted, travel direction and vehicle types could be detected using CNN on spectrogram images of radar real-time time series data. Presented solutions bring traffic monitoring systems one step closer to solving city traffic problems by increasing the traffic data quality valued by authorities.

5 Acknowledgements

Author would like to his supervisors Jaanus Kaugerand for giving a possibility to work in a project, Rene Pihlak for help with developing and both for supervising the development.

References

[1] A. Riid, J. Kaugerand, J. Ehala, M. Jaanus, and J. S. Preden, "An application of a low-cost microwave radar to traffic monitoring," *Proceedings of the Biennial Baltic Electronics Conference, BEC*, vol. 2018-October, Jan. 2019, doi: 10.1109/BEC.2018.8600962.

[2] ACEA, "Report – Vehicles in use, Europe 2021," Jan. 2021. Accessed: Dec. 14, 2022. [Online]. Available: https://www.acea.auto/publication/report-vehicles-in-use-europe-january-2021/

[3] Molnar Eva and Alexopoulos Constantinos, "ITS in urban transport: the challenges for the UNECE Transport Division", Accessed: Apr. 16, 2022. [Online]. Available: www.eurotransportmagazine.com

[4] D. B. Nguyen, C. R. Dow, and S. F. Hwang, "An Efficient Traffic Congestion Monitoring System on Internet of Vehicles," *Wirel Commun Mob Comput*, vol. 2018, 2018, doi: 10.1155/2018/9136813.

[5] B. Thomas, "Proposed rule would mandate vehicle-to-vehicle (V2V) communication on light vehicles, allowing cars to 'talk' to each other to avoid crashes," Dec. 13, 2016. https://one.nhtsa.gov/About-NHTSA/Press-Releases/ci.nhtsa_v2v_proposed_rule_12132016.print (accessed Dec. 15, 2022).

[6] B. Maurin, O. Masoud, and N. P. Papanikolopoulos, "Tracking all traffic: Computer vision algorithms for monitoring vehicles individuals, and crowds," *IEEE Robot Autom Mag*, vol. 12, no. 1, pp. 29–36, Mar. 2005, doi: 10.1109/MRA.2005.1411416.

[7] H. S. Lim, H. M. Park, J. E. Lee, Y. H. Kim, and S. Lee, "Lane-by-lane traffic monitoring using 24.1 ghz fmcw radar system," *IEEE Access*, vol. 9, pp. 14677–14687, 2021, doi: 10.1109/ACCESS.2021.3052876.

[8] L. C. Kristoffersen Theimann, "Comparison of depth information from stereo camera and LiDAR," 2020. Accessed: Dec. 20, 2022. [Online]. Available: https://folk.ntnu.no/edmundfo/msc2019-2020/LinaTheimannStereoVision.pdf

[9] S. Iftikhar, Z. Zhang, M. Asim, A. Muthanna, A. Koucheryavy, and A. A. A. El-Latif, "Deep Learning-Based Pedestrian Detection in Autonomous Vehicles: Substantial Issues and Challenges," *Electronics 2022, Vol. 11, Page 3551*, vol. 11, no. 21, p. 3551, Oct. 2022, doi: 10.3390/ELECTRONICS11213551.

[10] Thinnect, "SMENETE." https://thinnect.com/mist-computing-2/smenete-next-generation/ (accessed Dec. 18, 2022).

[11] "Intelligent Smart City and Critical Infrastructure Protection Technologies | Thinnect." https://thinnect.com/intelligent-smart-city-and-critical-infrastructure-protection-technologies/ (accessed Dec. 18, 2022).

[12] A. R. Ajiboye, R. Abdullah-Arshah, H. Qin, and H. Isah-Kebbe, "EVALUATING THE EFFECT OF DATASET SIZE ON PREDICTIVE MODEL USING SUPERVISED LEARNING TECHNIQUE," *International Journal of Computer Systems & Software Engineering*, vol. 1, no. 1, pp. 75–84, Feb. 2015, doi: 10.15282/IJSECS.1.2015.6.0006.

[13] J. Hestness *et al.*, "Deep Learning Scaling is Predictable, Empirically," Dec. 2017, doi: 10.48550/arxiv.1712.00409.

[14] J. Gu *et al.*, "Recent Advances in Convolutional Neural Networks".

[15] P. Wlodarczak, *Machine learning and its applications*. 2020. Accessed: Dec. 14, 2022. [Online]. Available: https://www.routledge.com/Machine-Learning-and-its-Applications/Wlodarczak/p/book/9781032086774

[16] K. Tihhonov, “MasterThesis GitHub” 2022. https://github.com/kitihh/MasterThesis (accessed Jan. 01, 2023).
[17] AAEON, “AI Traffic Monitoring System.” https://www.aaeon.ai/us/solutions/ai-traffic-monitoring-system (accessed Dec. 14, 2022).
[18] Vision Genious, “Traffic Genius.” https://visiongenius.ai/product/traffic-genius (accessed Dec. 14, 2022).
[19] “Oyla.” https://oyla.ai/ (accessed Dec. 14, 2022).
[20] Bluecity, “Real Time Traffic Monitoring Ai & Lidar Technology.” https://bluecity.ai/traffic-monitoring-solutions/ (accessed Dec. 19, 2022).
[21] Smartmicro, “Traffic Sensor.” https://www.smartmicro.com/traffic-sensor (accessed Dec. 19, 2022).
[22] Level Five Supplies, “Smartmicro UMRR-11 Type 132.” https://levelfivesupplies.com/product/automotive-radar-sensor-umrr-11-type-132/ (accessed Dec. 19, 2022).
[23] K. Garcia, M. Yan, and P. Alek, “Robust traffic and intersection monitoring using millimeter wave sensors.”
[24] I. Urazghildiiev *et al.*, “Vehicle classification based on the radar measurement of height profiles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 245–253, Jun. 2007, doi: 10.1109/TITS.2006.890071.
[25] J. Sánchez-Oro, D. Fernández-López, R. Cabido, A. S. Montemayor, and J. J. Pantrigo, “Radar-based road-traffic monitoring in urban environments,” *Digital Signal Processing: A Review Journal*, vol. 23, no. 1, pp. 364–374, 2013, doi: 10.1016/J.DSP.2012.09.012.
[26] J. Aslani, “Radar Based Object Detection and Tracking for Critical Infrastructure,” Tallinn, 2021.
[27] S. Cao and Y.-J. Wu, “Development of Intelligent Multimodal Traffic Monitoring using Radar Sensor at Intersections.” https://nitc.trec.pdx.edu/research/project/1296/Development_of_Intelligent_Multimodal_Traffic_Monitoring_using_Radar_Sensor_at_Intersections (accessed Dec. 14, 2022).
[28] S. Chadwick, W. Maddetn, and P. Newman, “Distant vehicle detection using radar and vision,” *Proc IEEE Int Conf Robot Autom*, vol. 2019-May, pp. 8311–8317, May 2019, doi: 10.1109/ICRA.2019.8794312.
[29] S. Shahinfar, P. Meek, and G. Falzon, “‘How many images do I need?’ Understanding how sample size per class affects deep learning model performance metrics for balanced designs in autonomous wildlife monitoring”, Accessed: Dec. 25, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1574954120300352
[30] B. Barz and J. Denzler, “Deep Learning on Small Datasets without Pre-Training using Cosine Loss,” *Proceedings - 2020 IEEE Winter Conference on Applications of Computer Vision, WACV 2020*, pp. 1360–1369, Jan. 2019, doi: 10.48550/arxiv.1901.09054.
[31] Google Cloud, “Preparing your training data.” https://cloud.google.com/vision/automl/object-detection/docs/prepare (accessed Dec. 26, 2022).
[32] DEEL Certification Workgroup IRT Saint Exupéry, “DATASET DEFINITION STANDARD (DDS),” Dec. 2020. Accessed: Nov. 11, 2022. [Online]. Available: https://arxiv.org/pdf/2101.03020.pdf

[33] AWS, “Traceability - Machine Learning Best Practices in Healthcare and Life Sciences.” https://docs.aws.amazon.com/whitepapers/latest/ml-best-practices-healthcare-life-sciences/traceability.html (accessed Nov. 12, 2022).

[34] M. M. Adnan, M. S. M. Rahim, M. Hasan Ali, K. Al-Jawaheri, and K. Neamah, “A Review of Methods for the Image Automatic Annotation,” *J Phys Conf Ser*, vol. 1892, no. 1, May 2021, doi: 10.1088/1742-6596/1892/1/012002.

[35] O. Sheremet, “Extract annotations from CVAT XML file into mask files in Python | by Oleksii Sheremet | Towards Data Science,” Aug. 10, 2020. https://towardsdatascience.com/extract-annotations-from-cvat-xml-file-into-mask-files-in-python-bb69749c4dc9 (accessed Dec. 05, 2022).

[36] Anonymous, “Data augmentation” *TensorFlow Core*, 2022. https://www.tensorflow.org/tutorials/images/data_augmentation (accessed Nov. 14, 2022).

[37] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J Big Data*, vol. 6, no. 1, Dec. 2019, doi: 10.1186/S40537-019-0197-0.

[38] J. Jordon *et al.*, “Synthetic Data -- what, why and how?” May 2022, Accessed: Nov. 16, 2022. [Online]. Available: http://arxiv.org/abs/2205.03257

[39] F. K. Dankar and M. Ibrahim, “Fake it till you make it: Guidelines for effective synthetic data generation,” *Applied Sciences (Switzerland)*, vol. 11, no. 5, pp. 1–18, Mar. 2021, doi: 10.3390/APP11052158.

[40] D. G. Murray, J. Šimša, A. Klimovic, and I. Indyk, “tf.data,” *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 2945–2958, Jul. 2021, doi: 10.14778/3476311.3476374.

[41] “Rolling the dice II” https://sasandr.wordpress.com/2012/05/04/rolling-the-dice-ii/ (accessed Dec. 12, 2022).

[42] K. K. Dobbin and R. M. Simon, “Optimally splitting cases for training and testing high dimensional classifiers,” *BMC Med Genomics*, vol. 4, 2011, doi: 10.1186/1755-8794-4-31.

[43] B. Genç and H. Tunç, “Optimal training and test sets design for machine learning,” *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 27, no. 2, pp. 1534–1545, 2019, doi: 10.3906/ELK-1807-212.

[44] J. JBarry-Straume, A. Tschannen, D. W. Engels, and E. Fine, “An Evaluation of Training Size Impact on Validation Accuracy for Optimized Convolutional Neural Networks,” *SMU Data Science Review*, vol. 1, no. 4, p. 12, 2018, Accessed: Nov. 14, 2022. [Online]. Available: https://scholar.smu.edu/datasciencereviewAvailableat:https://scholar.smu.edu/datasciencereview/vol1/iss4/12http://digitalrepository.smu.edu.

[45] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature 2015 521:7553*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[46] F. van Veen, “The Neural Network Zoo - The Asimov Institute.” https://www.asimovinstitute.org/neural-network-zoo/ (accessed Nov. 24, 2022).

[47] L. Alzubaidi *et al.*, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *J Big Data*, vol. 8, p. 53, 2021, doi: 10.1186/s40537-021-00444-8.

[48] S. Raschka and V. Mirjalili, “Python machine learning: machine learning and deep learning with python, scikit-learn, and tensorflow 2,” p. 741, 2019, Accessed: Nov. 08, 2022. [Online]. Available: https://www.worldcat.org/title/1135663723

[49] “(PDF) An Overview of Convolutional Neural Network: Its Architecture and Applications.”

https://www.researchgate.net/publication/329220700_An_Overview_of_Convolutional_Neural_Network_Its_Architecture_and_Applications (accessed Nov. 20, 2022).
[50] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A Survey of Deep Neural Network Architectures and Their Applications".
[51] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, no. 3, pp. 685–695, Sep. 2021, doi: 10.1007/S12525-021-00475-2.
[52] "Binary Image classifier CNN using TensorFlow | by Sai Balaji | Techiepedia | Medium." https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697 (accessed Nov. 28, 2022).
[53] "Module: tf.keras.layers | TensorFlow v2.11.0." https://www.tensorflow.org/api_docs/python/tf/keras/layers (accessed Nov. 28, 2022).
[54] C. Versloot, "What are max pooling, average pooling, global max pooling and global average pooling, ," *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, Feb. 2022, doi: 10.1109/ICDAR.2019.00177.
[55] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *32nd International Conference on Machine Learning, ICML 2015*, vol. 1, pp. 448–456, Feb. 2015, doi: 10.48550/arxiv.1502.03167.
[56] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, "Review of Image Classification Algorithms Based on Convolutional Neural Networks," Nov. 2021, doi: 10.3390/RS13224712.
[57] R. Pramoditha, "How to Choose the Optimal Learning Rate for Neural Networks," Sep. 21, 2022. https://towardsdatascience.com/how-to-choose-the-optimal-learning-rate-for-neural-networks-362111c5c783 (accessed Dec. 16, 2022).
[58] Anonymous, "Optimization problem - Wikipedia." https://en.wikipedia.org/wiki/Optimization_problem (accessed Nov. 08, 2022).
[59] M. Z. Mulla, "Cost, Activation, Loss Function." https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de (accessed Nov. 08, 2022).
[60] C. Kozyrkov, "What's the Difference Between a Metric and a Loss Function?" Jun. 2022, Accessed: Nov. 08, 2022. [Online]. Available: https://towardsdatascience.com/whats-the-difference-between-a-metric-and-a-loss-function-38cac955f46d
[61] Tensorflow, "Image classification," 2022. https://www.tensorflow.org/tutorials/images/classification (accessed Dec. 14, 2022).
[62] E. Alpaydin, *Introduction to Machine Learning, Third Edition*, no. course 395.
[63] A. Pauls and J. A. Yoder, "Determining Optimum Drop-out Rate for Neural Networks".
[64] "Training and evaluation with the built-in methods." https://www.tensorflow.org/guide/keras/train_and_evaluate (accessed Nov. 03, 2022).
[65] S. Shukla, "Regression and Classification | Supervised Machine Learning - GeeksforGeeks." https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/ (accessed Nov. 07, 2022).

[66] S. Minaee, “20 Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics,” Oct. 2019, Accessed: Nov. 03, 2022. [Online]. Available: https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce

[67] Anonymous, “Metrics and scoring: quantifying the quality of predictions — scikit-learn 1.1.3 documentation.” https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics (accessed Nov. 07, 2022).

[68] İ. Tanrıverdi, “Model Evaluation Metrics in Machine Learning,” Apr. 2021, Accessed: Nov. 03, 2022. [Online]. Available: https://medium.com/analytics-vidhya/model-evaluation-metrics-in-machine-learning-928999fb79b2

[69] T. Huellmann, “Precision vs Recall in Machine Learning.” https://levity.ai/blog/precision-vs-recall (accessed Nov. 07, 2022).

[70] D. M. W. Powers, “Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation,” Adelaide, 2007. Accessed: Nov. 07, 2022. [Online]. Available: https://web.archive.org/web/20191114213255/https://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf

[71] Anonymous, “F-score,” Nov. 06, 2022. https://en.wikipedia.org/wiki/F-score (accessed Nov. 07, 2022).

[72] Anonymous, “Classification: ROC Curve and AUC,” 2022. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc (accessed Nov. 07, 2022).

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Kirill Tihhonov

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Vehicle Counting and Direction Determination with Convolutional Neural Network Using Data From a 9.9GHz Low Power Microwave Radar”, supervised by Jaanus Kaugerand and Rene Pihlak
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

03.01.2023

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.