Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Measurement

Offloading of road sign recognition from autonomous vehicle to edge serves in mobile networks

Master Thesis

Supervisor: prof. Ing. Zdeněk Bečvář, Ph.D.

Field of Study: Cybernetics and Robotics

Bc. Jan Daněk

May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Daněk Jan** Personal ID number: **491847**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Offloading of road sign recognition from autonomous vehicle to edge serves in mobile networks

Master’s thesis title in Czech:

Rozpoznávání značek pro autonomní vozidlo se zpracováním informací na hraně mobilní sítě

Guidelines:

Study principles and possibilities of computation processing for autonomous vehicles at the edge of the mobile network (multi-access edge computing, MEC) and with usage of mobile networks for vehicles to infrastructure communication.
In a model of the autonomous vehicle, implement a neural network to discover road signs in a vehicle camera image and to identify the sign using existing datasets.
Test the possibilities of neural network processing directly in the vehicle with limited computing resources with a solution allowing to transfer at least a part of the neural network processing to the MEC server.
Compare a delay and a success rate of road sign detection and identification for both solutions and compare the energy consumption of both solutions considering, for example, an availability of resources for communication and for processing in the vehicle and in the MEC server.

Bibliography / sources:

[1] P. Mach and Z. Becvar, “Mobile Edge Computing: A Survey on Architecture and Computation Offloading,” IEEE Communications Surveys & Tutorials, volume 19, no. 3, 2017.
[2] T. Verschoor, V. Charpentier, N. Slamnik-Krijestorac and J. Marquez-Barja, "The testing framework for Vehicular Edge Computing and Communications on the Smart Highway," IEEE Consumer Communications & Networking Conference (CCNC), 2023.
[3] J. Dolezal, Z. Becvar and T. Zeman, "Performance Evaluation of Computation Offloading from Mobile Device to the Edge of Mobile Network," IEEE Conference on Standards for Communications & Networking (IEEE CSCN 2016), 2016.
[4] Z. Q. Zhao, P. Zheng, P., S. T. Xu, X. Wu, "Object detection with deep learning: A review," IEEE Transactions on Neural Networks and Learning Systems," volume 30, no. 11, pp. 3212-3232, 2019.

Name and workplace of master's thesis supervisor:

prof. Ing. Zdeněk Bečvář, Ph.D. Department of Telecommunications Engineering FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **06.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until:
by the end of summer semester 2024/2025

prof. Ing. Zdeněk Bečvář, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to express my sincere gratitude to prof. Ing. Zdeněk Bečvář, Ph.D., for his invaluable guidance and advice throughout the course of my thesis. His expertise and insights have greatly contributed to the success of this work.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

In Prague, 24. May 2024

.....

bc. Jan Daněk

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 24. května 2024

.....

bc. Jan Daněk

Abstract:

This thesis is focused on road sign detection and investigates the possibility of the road sign detection and classification locally in an autonomous vehicle and in the edge of mobile network, represented by multi-access edge computing (MEC) server. First, the thesis focuses on the analysis of existing methods for road sign detection and classification to select the most suitable techniques for this task. Due to the lack of suitable datasets for a complex neural network, the detection and the classification of the road signs are handled via two independent neural networks, one for the road sign detection and the other for the road sign classification. Both neural networks are designed and implemented for road sign detection and classification on a model of the autonomous vehicle and in the MEC server. Dataset for neural network training for detection purposes is created, as the existing datasets are unsuitable and do not provide sufficiently accurate results for own created road signs. The experiments are performed on real data collected from a model of the autonomous vehicle equipped with a camera. Classification accuracy, processing time, and energy consumption are evaluated for both local processing and offloading to MEC server via mobile network. The results of experiments demonstrate that neural network-based processing of images from the vehicle on the MEC server significantly reduces the time required to identify the road sign compared to local data processing on the autonomous vehicle. Additionally, the findings reveal that low communication channel quality increases the vehicle's energy consumption for recognition of one road sign.

Key Words:

Edge computing, mobile network, autonomous vehicle, road sign, detection, classification, neural networks.

Abstrakt:

Tato práce se zaměřuje na detekci a klasifikaci dopravních značek a zkoumá možnosti lokální detekce a klasifikace dopravních značek v autonomním vozidle a na hraně mobilní sítě, která je reprezentovaná multi-access edge computing (MEC) serverem. Práce se nejprve zaměřuje na analýzu stávajících metod detekce a klasifikace dopravních značek s cílem vybrat vhodné techniky pro tuto úlohu. Vzhledem k nedostatku vhodných datasetů pro naučení komplexní neuronové sítě je detekce a klasifikace dopravních značek řešena prostřednictvím dvou nezávislých neuronových sítí, jedné pro detekci dopravních značek a druhé pro klasifikaci dopravních značek. Obě neuronové sítě jsou navrženy a implementovány pro detekci a klasifikaci dopravních značek na modelu autonomního vozidla a v MEC serveru. Je vytvořen dataset pro trénování neuronových sítí pro účely detekce, protože stávající datasety jsou nevhodné a neposkytují dostatečně přesné výsledky pro značky použité v diplomové práci. Experimenty jsou prováděny na reálných datech získaných z modelu autonomního vozidla vybaveného kamerou. Přesnost klasifikace, doba zpracování a spotřeba energie jsou vyhodnoceny jak pro lokální zpracování, tak pro přenesení na server MEC prostřednictvím mobilní sítě. Výsledky experimentů ukazují, že zpracování obrázků pomocí neuronových sítí na MEC serveru významně zkracuje dobu pro určení dopravní značky oproti lokálnímu zpracování dat v autonomním vozidle. Kromě toho zjištění ukazují, že špatná kvalita komunikačního kanálu zvyšuje spotřebu energie vozidla potřebnou k rozpoznání jednoho snímku.

Klíčová slova:

Výpočty na hraně sítě, mobilní síť, autonomní vozidlo, dopravní značka, klasifikace, detekce, neuronové sítě.

Překlad názvu:

Rozpoznávání značek pro autonomní vozidlo se zpracováním informací na hraně mobilní sítě

Table of Content

Table of Figure

Table of Table

1. Introduction

In today's digital era, automatic object detection and classification in images and videos introduce significant challenges and opportunities in various fields. This includes detection and classification of road signs, crucial for enhancing road safety and advancing autonomous vehicles [1]. Detection is the process of finding the bounding box of road sign, classification is process get information about road sign type and recognition is the process of detection and classification the road sign. Autonomous vehicles, or self-driving cars, have an immense potential to revolutionize transportation globally. The autonomous vehicles rely on a sophisticated array of sensors, such as cameras, lidar, or global positioning system (GPS), to perceive their surroundings and navigate without human intervention [2]. However, achieving reliable autonomous driving poses significant challenges. A key challenge is a precise object detection and classification. This encompasses classification road signs, pedestrians, vehicles, and obstacles, crucial for safe navigation through dynamic environment [1]. Furthermore, the robustness of autonomous systems should be ensured across diverse conditions like changing weather, varying lighting, or unpredictable road traffic situations [3]. Overcoming these obstacles demands advanced and computationally complex algorithms capable of interpreting a wide range of real-world scenarios with adaptability and resilience.

The problem computational complexity of road sign detection and classification on autonomous vehicles can be facilitated via edge computing. The edge computing enables fast processing of data closer to its source [4]. While centralized clouds offer a high computing power, transmitting data to and from the centralized cloud incurs significant latency [5]. In contrast, edge servers positioned at the network edge, albeit with lower computing power, mitigate latency issues. In the realm of the autonomous vehicles, edge computing can reduce latency, optimize resource utilization, and enhances scalability and reliability [6]. This capability enables the autonomous vehicles to execute real-time decisions efficiently, even within the confines of limited onboard computational resources, thereby enhancing overall performance and responsiveness.

This thesis aims to explore the efficacy of modern machine learning and deep learning techniques for detection and recognition of road signs for the application in the autonomous vehicles. Leveraging the evolution of mobile networks and edge computing, the thesis focuses on implementing neural networks for road sign detection and classification in the autonomous vehicle. The data for neural networks are obtained by camera mounted on vehicle.

The goal of this thesis is to test performance of neural network for road signs detection and classification in the scenario with available connectivity of the vehicle via mobile network and availability of multi-access edge computing (MEC) server to process data using neural networks.

The possibility of controlling autonomous vehicles using deep neural networks is explored in [7]. The authors create a small neural network to control vehicles steering by recognition of the guidelines and specific road signs. We do not aim to directly control our vehicle using a neural network. Instead, our objective is to gather the necessary data for autonomous driving systems and preprocess it by using neural networks. Notably, the main innovation in this diploma thesis lies in incorporating mobile networks and edge servers, a facet not addressed in [7].

In [8], the authors work on self-driving car in simulation by cloning the behavior of human driver by neural network. The car is able to clone this human behavior and can detect objects like traffic lights, guidelines etc. The challenge addressed in this work stems from its reliance solely on

simulation. Our objective is to evaluate the performance of our autonomous vehicle, particularly its road sign recognition capabilities, under real-world conditions, where computational resources are limited, and sensor data is corrupted by a measurement noise, as in the case of blurred camera images.

Furthermore, in [9], researchers focus on classification of road signs using neural networks. Unlike our approach, which involves processing images of the surrounding environment, the authors exclusively work with isolated images of individual road signs extracted from larger scenes. While we adopt a similar classification method, we also include a detection process for classification road signs using an additional neural network making the whole processing significantly more demanding in terms of computation.

For the offloading purposes, the autonomous vehicle should communicate with the gNB to transfer the data to be processed to the MEC server. The latency and performance of the 5G network with MEC server is investigated in [10] in testbed in Antwerp. The testbed is the highway with several gNBs close to the road. Researchers in the study work with real data and 5G network infrastructure, but the data does not have an impact on vehicle and researchers measure the latency with communication to MEC server. In our case, we evaluate the possibility to offload neural networks for road sign recognition. Unlike in [10], we evaluate on MEC server and we investigate latency of the local and MEC server processing related to the time to react to road signs.

In practice, the detection and classification parts of neural network are solved together. But there is not sufficient dataset for complex neural network. Therefore, we solve both problems with two separate neural networks for which the required datasets are available [11], [12], [13]. The detection phase is commonly solved using Faster R-CNN [14], Single Shot Detection (SSD) [14], You Only Look Once version 3 (YOLOv3) [14], [15]. The Faster R-CNN introduces a Region Proposal Network (RPN) to efficiently generate region proposals area. Detectable objects are searched for in a region proposal area. The object detection process is divided into two stages. First, the algorithm identifies region proposal areas, and then it detects objects within these areas. This two-stage approach allows Faster R-CNN to achieve a higher accuracy than other neural networks. Single Shot Detection (SSD) network, unlike two-stage methods, adopts a single-shot approach directly predicting object bounding boxes and class probabilities from a single pass through the network. Look Once version 3 (YOLOv3) is also a one-stage method. Like many other neural networks, YOLOv3 is based on DarkNet-53 [16]. The principle of object detection and classification of YOLOv3 is to divide the input image into a grid of cells and predict bounding boxes and object class probabilities for each cell simultaneously. The image classification is commonly solved via, for example, Residual Network (ResNet) [17] Visual Geometry Group-16 (VGG-16) [18], Inception Networks (GoogLeNet) [19], or AlexNet [18].

The main contributions of this work include:

- Road sign detection and classification are implemented to the autonomous vehicle along with the support for offloading of the computation to MEC server.
- The possibility of processing related to the road sign detection and recognition directly on the autonomous vehicle or in the MEC server is investigated with real data.
- New approach to road sign recognition with separated detection and classification parts is explored.
- Experiments with offloading and local processing of road sign detection and classification demonstrate an energy saving on autonomous vehicle when we reduced

the power consumption to 1/7 per evaluated image. During offloading, we aim to transfer the entire amount of camera data to the MEC server and evaluate the data using neural networks on it.

- All developed codes and software are publicly available at Gitlab [1].

The rest of this thesis is structured as follows. First, in Chapter 2, we introduce background and details of aspects related to the thesis, including mobile networks, MEC, and YOLOv3. Then, in Chapter 3, we outline data flow from autonomous vehicle to MEC server and a high-level design of the system. Chapter 4 provides an overview of the autonomous vehicle on which we conduct experiments and its capabilities for communication via the mobile network. The used datasets is described in Chapter 5 along with the process of creation new datasets for the detection purposes. Subsequently, we describe the process of implementing neural networks, choosing the YOLOv3 for the detection part and a custom-made neural network for the classification part in Chapter 6. The performance is evaluated in Chapter 7. Last, the thesis is concluded, and future directions are outlined in Chapter 8.

[1] https://gitlab.fel.cvut.cz/mobile-and-wireless/autonomous-driving

2. Background

This chapter introduces background for main components considered in the thesis including mobile network used for the offloading of computation to the MEC servers, MEC concept, and neural network YOLOv3 for detection part of road sign recognition, and principle of distance measurement using camera.

2.1. Communications and edge computing for sensor data processing

In this section, we describe the mobile network and concept of multiple access edge computing for processing of data from an autonomous vehicle. The mobile network ensures communication between autonomous vehicle and MEC server. Communication and edge computing are key elements enabling efficient processing and analysis of data close to the source, leading to faster and accurate decision-making with a high scalability in terms of computing resource availability allowing to reduce cost of the vehicles.

2.1.1. Mobile network

This chapter provides an overview of the fifth generation (5G) mobile networks [20], [21], adopted for experiments in the thesis. The 5G network promises high data rates, low latency, and massive device connectivity, towards application in various industries [22]. Due to performance improvements, 5G networks are suitable also for sensor data processing on MEC server in scope of autonomous vehicle.

The 5G networks offer several key features that are particularly beneficial for autonomous vehicles. These include low latency in the order of ms, ensuring minimal delay in data transmission and enabling responsiveness for critical driving decisions [23]. High reliability ensures consistent connectivity, crucial for the continuous exchange of data between vehicles and infrastructure. Wide bandwidth allows for the efficient transmission of large volumes of data, such as video streams from onboard sensors.

The architecture of 5G networks comprises two parts [24]: Radio Access Network (RAN) and Core network. At the forefront of 5G, the RAN consists of base stations known as gNodeB (gNB), which communicate wirelessly with user equipment (UE), such as smartphones, IoT devices, or autonomous vehicle. The core network is responsible for routing data between various network elements and providing services such as authentication, billing, and policy enforcement.

The radio interface in the 5G network, also known as the RAN [21], enables fast and reliable communication between UE and the mobile network infrastructure. Channel quality in the 5G radio interface is reflected in the level of interference, noise, and signal reflections, which can affect the stability and value of datarate. Higher channel quality means less interference and better conditions for data transmission with lower losses and a higher datarate. Modulation and coding (MCS) [25] are techniques used for data transmission in radio networks. Higher MCS allows transmission of a larger amount of data using the same amount of spectrum or vice versa, transmission of the same amount of data with less spectrum. Selecting the right MCS depends on the current channel conditions and the required bitrate value.

2.1.2. Multi-access edge computing server

The MEC server plays a crucial role in enabling efficient and responsive autonomous driving applications [4]. The MEC is facilitated via a distributed computing architecture that brings computational resources closer to the UEs, typically to the gNBs or aggregation points. At its core, the MEC server consists of high-performance computing resources, including processors, memory, and storage.

In terms of functionality, the MEC server hosts software applications and services that leverage its computational resources to perform various tasks related to autonomous driving [26]. These tasks may include real-time sensor data processing, object detection and classification, path planning, and decision-making algorithms.

One of the key features of the MEC server architecture is its support for both partial and full offloading of computational tasks from autonomous vehicles [4]. Partial offloading involves offloading specific tasks or data processing to the MEC server while retaining some processing capabilities onboard the vehicle. Full offloading, in contrast, involves delegating the entire computation to the MEC server, allowing the vehicle to focus solely on data acquisition and communication. In our case we only do full offloading or local data processing.

2.2. Description of You Only Look Once version 3

In this chapter, we discuss the YOLOv3 [14], [15] neural network as it is used in this thesis to detect road signs. We focus on YOLOv3 model and loss function used in training process. YOLOv3 is a deep convolutional neural network used for object detection in images. YOLOv3 is a one-stage method such as SSD. Like many other neural networks, YOLOv3 is based on DarkNet-53 [16]. One of the key aspects of YOLOv3 is its ability to detect objects in the real time with a high accuracy.

The main principle of YOLOv3 is to divide the image into a grid and apply a single convolutional neural network to the entire image. This division creates cells. The division is repeated three times at grid sizes of 13x13, 26x26, and 52x52 cells [15]. The network produces predictions for various classes and bounding boxes simultaneously for every grid division and every cell separately.

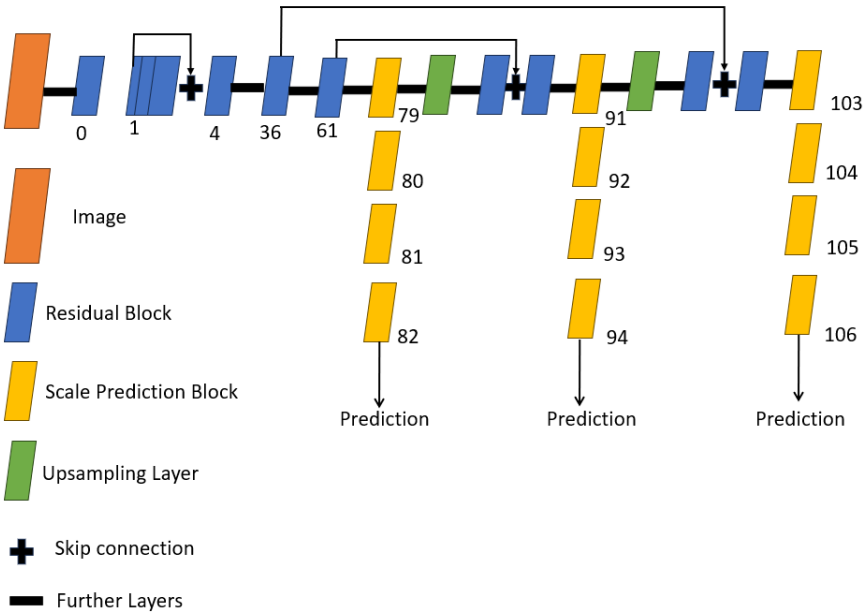

Figure 1: Architecture of You Only Look Once v3

Model of YOLOv3 is in Figure 1. Residual blocks, also known as residual connections, are fundamental building blocks of deep neural networks that enable the training of deep models. Residual blocks address the problem of vanishing gradients [27] by adding skip connections, which transmit information directly from one layer to distant layers of the model. This prevents information loss and allows for more efficient training and higher performance in deep networks. Residual blocks are made from two convolutional blocks. The first block halves the number of channels and the second layer increases the number of channels to the original value. So, no information is lost.

In the scope of this thesis, the convolutional block includes a convolutional layer, a batch normalization layer, and a leaky ReLU layer [28]. Leaky ReLU is an activation function that helps neural networks react to nonlinear features and inputs, without this function neural networks can react only to linear problems.

The scale Prediction layer is responsible for generating predictions at different spatial scales, enabling the detection of objects of various sizes and locations in the image. Having predictions at multiple scales allows for detecting objects at different sizes and distances from the camera and capturing various details in the image. The scale prediction layer combines outputs from different layers with different resolutions to provide comprehensive object predictions. In this thesis, we used blocks called Scale Prediction blocks. These blocks are made up of one residual block, one convolution block, and a scale prediction layer.

The prediction layer is followed by upsampling. Upsampling is a method to increase the spatial dimensions of an input feature map. Upsampling involves interpolating the values of the input feature map to generate a larger output feature map with a higher spatial resolution.

YOLOv3 also uses anchor boxes [14], which are bounding boxes with fixed size that help to find objects with different sizes. They work like sliding windows that scan the image and if they see the search object, they mark the cell where they see this object. There are different sizes of anchor boxes because we want to search for objects of different sizes and proportions. The size of anchor boxes is different in each grid size and is obtained by experiments and has the size representing the proportion of the search objects.

The loss function of YOLOv3 combines object detection with object classification. This function aims to minimize the difference between the actual and predicted bounding boxes, classify objects in the image and is used in the process of learning. The loss function is divided into four categories, namely no object loss, object loss, box candidates loss and class loss, which are added together using weights [15] , [29].

The no object loss L_{noobj} is assigned to cells that do not have a search object in them and penalizes YOLOv3 if YOLOv3 finds an object that is not there. It is calculated by following formula:

$$L_{noobj} = \frac{1}{N \sum_{a,i,j} \mathbb{1}_{n=1}^{noobj}} \sum_{n=1}^{N} \sum_{a,i,j \in \mathbb{1}_{n=1}^{noobj}} -\left[y_{n,a,i,j}^{obj} \times \log \sigma\left(t_{n,a,i,j}^{obj}\right) + \left(1 - y_{n,a,i,j}^{obj}\right) \times \log\left(1 - \sigma\left(t_{n,a,i,j}^{obj}\right)\right)\right], \quad (1)$$

where i, j are the coordinates of concrete cell in the grid, the variable a is the index of the anchor box and N is the batch size, $\mathbb{1}_{n=1}^{noobj}$ is at the binary tensor, which has a value of 1 if the anchor box

is not assigned to an object and zero otherwise, the output of the network for the correct cell is denoted as t and the correct value is given by y. Function σ is sigmoid function and is given by:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2}$$

The object loss L_{obj} penalizes the neural network if it does not find an object that is in the cell and is calculated by:

$$L_{obj} = \frac{1}{N \sum_{a,i,j} \mathbb{1}_{n=1}^{obj}} \sum_{n=1}^{N} \sum_{a,i,j \in \mathbb{1}_{n=1}^{obj}} -\left[\hat{y}_{n,a,i,j}^{obj} \times \log \sigma\left(t_{n,a,i,j}^{obj}\right) + \left(1 - \hat{y}_{n,a,i,j}^{obj}\right) \times \log\left(1 - \sigma\left(t_{n,a,i,j}^{obj}\right)\right)\right]. \tag{3}$$

The previous equation is very similar to the equation for no object loss. The only difference is in the binary operator $\mathbb{1}_{n=1}^{obj}$, this binary tensor has ones when the anchor box is assigned to a target bounding box and $\hat{y}_{n,a,i,j}^{obj}$ is an intersection over union (IOU), between the output value and computed bounding box by following formulas:

$$b_x = \sigma(t_x) + c_x, \tag{4}$$

$$b_y = \sigma(t_y) + c_y, \tag{5}$$

$$b_w = p_w e^{t_w}, \tag{6}$$

$$b_h = p_h e^{t_h}. \tag{7}$$

where p_h and p_w are anchor box sizes c_x and c_y is offset given by cell, t_x, t_y, p_w and p_h is the position of the bounding box related to the concrete cell, see Figure 2.

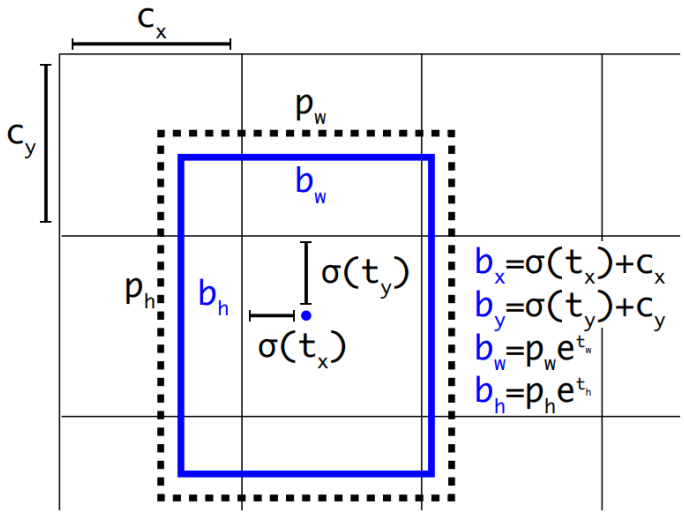

Figure 2: Bounding box location [15]

Box coordinates loss L_{box} penalizes the found bounding box, respectively its dimensions and position, it is calculated by:

$$L_{box} = \frac{1}{N \sum_{a,i,j} \mathbb{1}_{n=1}^{obj}} \sum_{n=1}^{N} \sum_{a,i,j \in \mathbb{1}_{n=1}^{obj}} \Big[\big(\sigma(t_{n,a,i,j}^{x}) - y_{n,a,i,j}^{x}\big)^2 + \Big(\big(\sigma(t_{n,a,i,j}^{x}) - y_{n,a,i,j}^{y}\Big)^2 + \Big(\big(\sigma(t_{n,a,i,j}^{w}) - t_{n,a,i,j}^{\widehat{w}}\Big)^2 + \Big(\big(\sigma(t_{n,a,i,j}^{h}) - t_{n,a,i,j}^{\widehat{h}}\Big)^2 \Big], \tag{8}$$

where indexes x and y denote the dimensions and w and h are the height and the width of bounding boxes, t denotes true bounding boxes and $t^{\widehat{w}}$ and $t^{\widehat{h}}$ are computed by:

$$t^{\widehat{w}} = \log\left(\frac{y_w}{p_w}\right), \tag{9}$$

$$t^{\widehat{h}} = \log\left(\frac{y_h}{p_h}\right). \tag{10}$$

Class loss L_{class} is denoted for the classification of object and incurs penalized when the object is classified incorrectly, calculated by:

$$L_{class} = \frac{1}{N \sum_{a,i,j} \mathbb{1}_{n=1}^{obj}} \sum_{n=1}^{N} \sum_{a,i,j \in \mathbb{1}_{n=1}^{obj}} -\log\left(\frac{\exp\left(t_{n,a,i,j}^{c}\right)}{\sum_k \exp\left(t_{n,a,i,j}^{k}\right)}\right), \tag{11}$$

where t^c is the predicted class and t^k is the true class.

Complete loss L is found in next formula:

$$L = \lambda_{noobj} L_{noobj} + \lambda_{obj} L_{obj} + \lambda_{box} L_{box} + \lambda_{class} L_{class}, \tag{12}$$

where λ_{noobj}, λ_{obj}, λ_{box} and λ_{class} are constants that gives a weight to losses. In the original paper [15] are constants set to ones.

The loss calculated in (12) is propagated backward through the YOLOv3 model during training. This backpropagation step involves adjusting the model's weights based on the calculated loss, which iteratively refines the network's ability to make accurate predictions. This iterative process of updating the model's parameters based on the computed loss is fundamental to training YOLOv3, allowing it to learn and improve its performance over time.

2.3. Camera principle

We adopt following principle to measure the distance d between the vehicle (camera) and the detected object, in our case road sign. An illustration of the principle is shown in Figure 3.

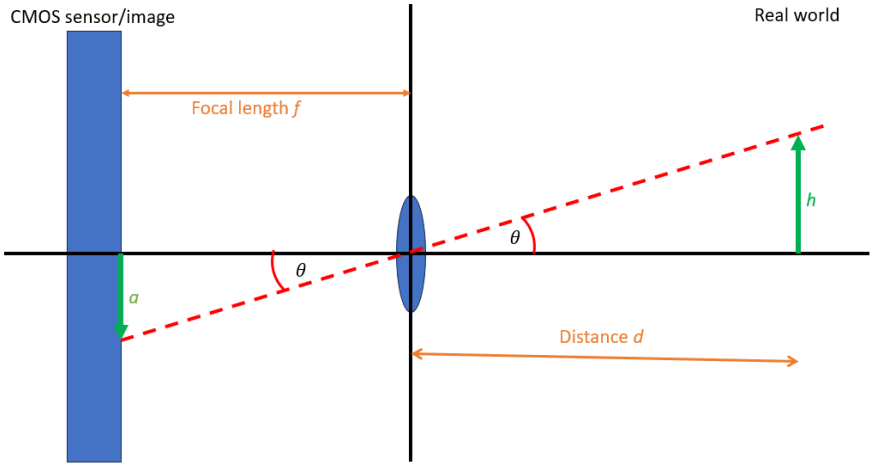

Figure 3: Proportional measurement of distance of road sign

The camera's lens projects the scenery onto the CMOS sensor, which converts light into a digital representation of the image. By considering several parameters, we can estimate the distance of a real object from the camera. The θ angle is computed as follows:

$$\frac{a}{f} = \tan\theta = \frac{h}{d}, \tag{13}$$

where a is captured object size in pixels, f is focal length also in pixels, distance d denotes the distance between the camera and the real object and h is its height, both in centimeters. Focal length f is known value or is determined through experiments. Based on this we write equation for d as follows:

$$d = \frac{h \times f}{a}. \tag{14}$$

3. High level design

In this chapter, we introduce the setup of MEC server in our laboratory, as well as discuss why we need to separate the detection and classification parts of neural networks.

3.1. Setup of MEC server and 5G network

In this section, we describe the configuration of our MEC server and the 5G network setup used in our experiments. MEC server is created by Intel NUC11PHKi7CAA2 (Intel NUC) with processor i7-1165G7, 16 GB RAM and graphic card Nvidia RTX 2060. Every task, such as a neural network, has a separate Docker container [30] on the MEC server. This ensures that each research participant can simply encapsulate their codes into individual containers that do not interact with each other. The only exception is the transfer of data between them using Apache Kafka. The flow of camera data to the MEC server and processed data back to autonomous vehicle follows the path described in Figure 4. The local data flow in case of local computation is in bottom part of Figure 4. The function of each node in this figure is described below:

- Camera: The input data originates from a camera Niceboy STREAM.
- Camera Node: This Robot Operating System (ROS) node captures images from the camera and adjusts their resolution as required and sends this image to ROS environment as ROS message. Robot Operating System is a flexible framework for writing robot software.
- Gateway Sender: This ROS node Intercept ROS message with image and sends camera data to the 5G network from autonomous vehicle. This node also encodes the image to reduce the transmitted size using .jpg quality reduction.
- Modem: The Quectel RM500Q modem is designed to encode and decode messages according to the 5G standard. When transmitting data, it encodes the message and broadcasts it to the surrounding area using an antenna. On the receiving end, the antenna picks up the signal, and the modem decodes the message back into its original form. In both cases accordance with the 5G standard.
- Software Defined Radio (SDR): The SDR USRP N310 is used together with a computer as a gNB in RAN. The gNB receives data, which is then decoded. On the way from the server, the data is encoded and broadcasted to the surrounding environment.
- Core Network: In this stage the data is only forwarded to the MEC server or to the RAN. We use OpenAirInterface (OAI) [31] to emulate 5G network in our laboratory. OAI is an open-source software framework for research and development of next-generation networks, including 5G and its successors. OAI is a flexible and extensible tool that allows researchers and developers to conduct advanced experiments and test new technologies in next-generation networks. For our network, we used a bandwidth of 20 MHz.
- TCP-Gateway: The first part is the MEC server. This gateway only receives data from the core network and forwards them based on their type to specific Docker containers or collects data from Docker containers and sends them to the core network.
- CNN-Container: The final destination of the camera data is a Docker container running YOLOv3 for road signs detection and custom make convolutional neural network for road signs classification. Information about classified road signs is then sent back to TCP-Gateway.
- Control algorithms: Algorithms for vehicle autonomous driving.
- CNN node: ROS node that runs the neural network for local image processing.

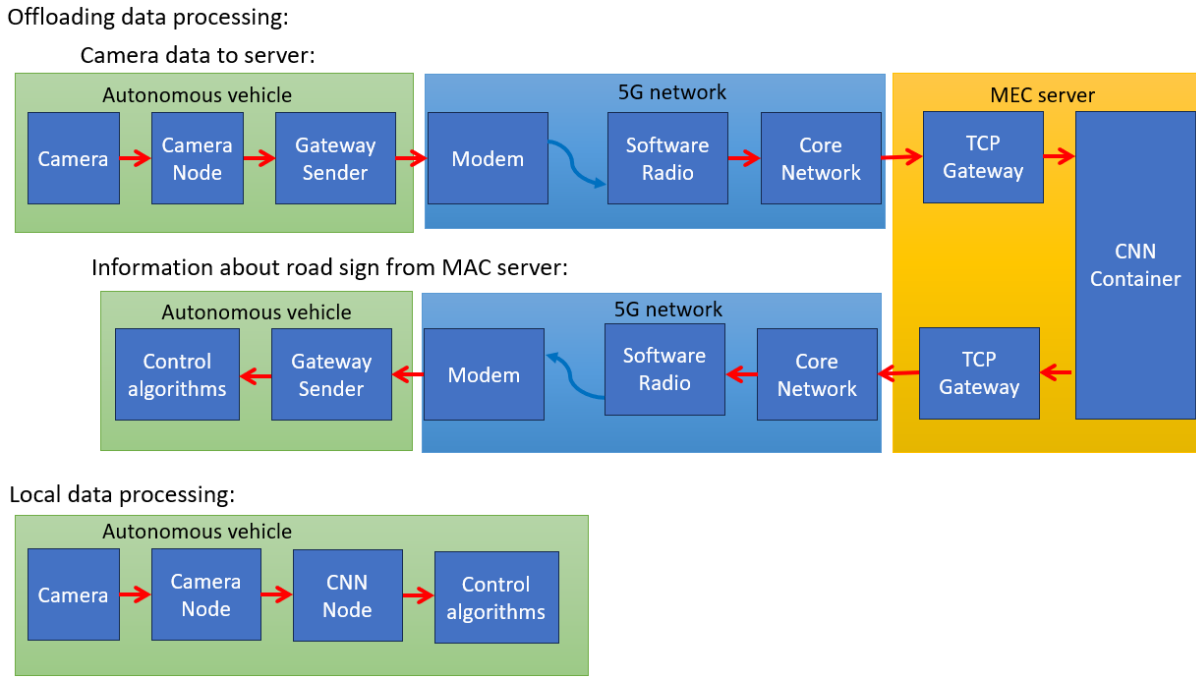

Figure 4: Flow of camera data to MEC server and back

3.2. Road sign detection and classification using neural networks

In this thesis, it is essential to separate the detection and classification parts of the neural network. In this section, we examine the motivation behind splitting the detection and classification functionalities of the neural network in this thesis. Detection is the process of find the bounding box of road sign, classification is process get information about road sign type and recognition is the process of detection and classification the road sign.

The decision to split the detection and classification tasks is based on following reasons. First, the availability of a comprehensive dataset plays a key role in training robust neural network models. However, in this context, acquiring a dataset that adequately covers both detection and classification is complicated. Despite effort, the dataset available for road signs detection and classification falls short in terms of diversity and volume, making it insufficient for training a unified model to handle both tasks effectively. For example, German Traffic Sign Detection Benchmark (GTSDB) [13] has 43 classes and only 600 training images, which means about 20 images per class. This amount of training images is insufficient. Neural networks such as YOLOv3 need about 1000 images in the worst case per class to correctly learn, but it is recommended to use about 1500 images to train the YOLOv3 neural network [32]. The number of images per class is also possible to increase by image transformation. However, from 20 images, it is not feasible to make 1000 unique images. This problem is described, for example, in the paper [33].

Creating a dataset for classification tasks poses significant challenges, particularly in terms of time and resource constraints. In our case, we aim to classify approximately 10 different classes, requiring a dataset of about 15 000 images. Based on our experiments, which involve creating a small dataset with road signs, we estimate that we would need a total of 25 000 images, requiring approximately 7 hours to capture, not including setup time. Additionally, the process of separating the images is time-consuming, taking about 9 hours. Each image should be manually labeled by a human, which, even with the use of specialized software, takes an average of 20 seconds per

image. Therefore, labeling 15 000 images would require approximately 84 hours. Considering all these factors, the total time required to create the dataset amounts to 100 hours. Given these constraints, it becomes evident that the task of creating a proper dataset within our timeframe is not feasible.

Moreover, the inherent limitations of the YOLOv3 model further necessitate the separation of detection and classification. While YOLOv3 excels in real-time object detection tasks, it may not deliver optimal performance in class estimation. Class estimation achieves greater precision with a specialized neural network.

In summary, the decision to split the detection and classification tasks arises from the unavailability of a suitable dataset, time constraints, and limitations in the precision of the YOLOv3 model, collectively highlighting the importance of approaching these tasks separately to achieve satisfactory outcomes.

4. Implementation of autonomous vehicle

In this chapter, we introduce the developed model of the autonomous vehicle and all components including the power supply system, computation logic, and sensors. The hardware preparation of the autonomous vehicle was conducted as part of the project that directly preceded this thesis.

4.1. Assembly of and architecture autonomous vehicle

In the thesis, we work with a model of Mercedes-Benz G63 with chassis TRX-6 [34], see Figure 5. The chassis has a gearbox that changes the speed of the vehicle, and two gears are available. One gear is faster, but with low torque. The second gear is for a low speed, but with a higher torque.

Figure 5: Photo of the vehicle model

The autonomous vehicle's communication and control architecture with all components is depicted in Figure 6. In this figure, blue rectangles depict physical components, red lines represent power connections, and blue lines are data connections. Physical implementation of the components is then shown in Figure 7.

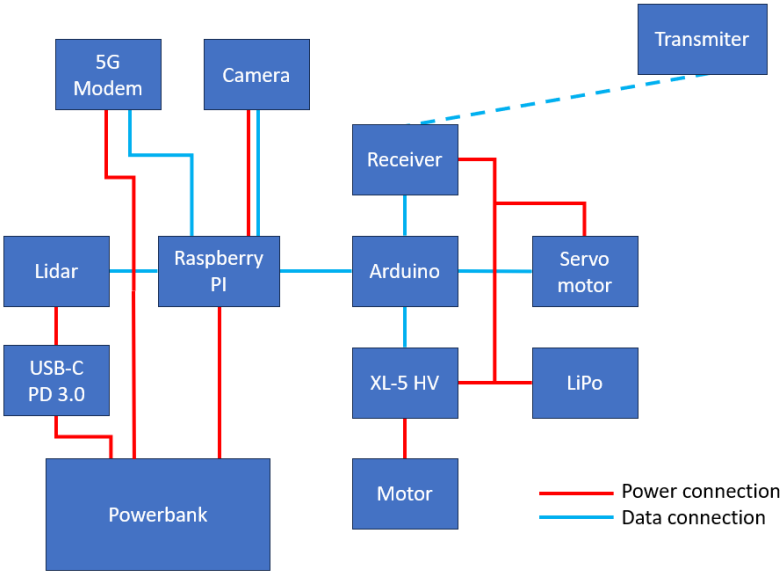

Figure 6: Scheme of autonomous vehicle

The Raspberry PI 4 Model B (Raspberry Pi) serves as the primary computational unit for autonomous tasks and sensor data collection. The Raspberry Pi handles tasks such as Simultaneous Localization and Mapping (SLAM), which involves vehicle localization in space, path planning between the start and end points, and subsequent navigation along the planned path. The power bank supplies power to computational logic, sensors, and 5G modem. The Light Detection And Ranging (lidar) captures information about the surrounding environment for navigation purposes, while the USB-C PD 3.0 delivers power to the lidar's motor from the power bank with 9V output. The camera captures images of the environment for purposes of road sign detection and navigation. Furthermore, Arduino is deployed to provide interface between the Raspberry Pi and motors for the purposes of the motor control by the Raspberry Pi. The Transmitter and Receiver facilitate a manual control of the vehicle. A servo motor steers the front axle of the vehicle and the XL-5 HV regulator adjusts the vehicle speed based on commands from the Raspberry Pi or transmitter. The motor propels the vehicle and a LiPo battery powers the motor, servo motor, and receiver. We deploy the 5G modem RM 500 Q, that is directly connected to the Raspberry Pi, to ensure 5G connection to the 5G network. This modem has a total power consumption of 15 W.

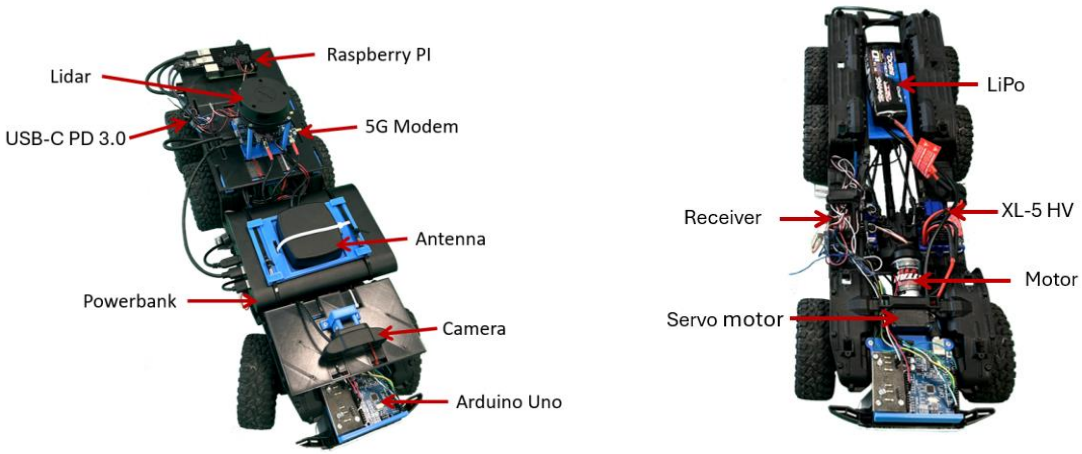

Figure 7: Description of autonomous vehicle a) computation power b) motors

4.2. Power supply system

The vehicle is powered by two separate batteries. The first battery is used for computation logic. This battery is a laptop powerbank Viking smartech II with a capacity of 40 000 mAh [35]. This powerbank has several outputs like two USB-A, one USB-C with power delivery 3.0, and one output with adjustable voltage from 5V up to 20V. This power bans supplies Raspberry Pi with 5V and at least 3A. We select this power bank due to its ample capacity to ensure enough time for experiments. Estimated power supply is at maximum 39 W. Raspberry Pi consumes 15 W as well as 5G modem RM 500 Q. The lidar motor consumes at maximum 9 W to ensure the rotation of lidar.

The powerbank lifetime t for our setup is determined as follows:

$$E_p = C_p \times U_p = 40 \times 3.7 = 148\ [Wh], \tag{15}$$

$$t = \frac{E_p}{E_C} = \frac{148}{39} = 3.8\ [h]. \tag{16}$$

where E_p stands for the energy of the powerbank in watt-hours, C_p denotes the capacity of the powerbank in amper-hours, and U_p is the voltage at which the capacitance is specified, E_C represents the energy required for the setup to operate for one hour.

The second battery in the vehicle powers the Titan 550 21T DC electric motor, which takes care of the vehicle's movement. This battery is a 2-cell LiPo battery with a nominal voltage of 7.4 V and a capacity of 7600 mAh [2]. The battery for the motor is connected directly to the speed controller XL-5 HV.

4.3. Computation logic

The main computing power of the vehicle is concentrated in Raspberry Pi 4 Model B - 8GB RAM [36]. This computer is the main control computer on the vehicle. The current version of autonomous vehicles uses A^* algorithm [37] with the several modifications for pathfinding and classical carrot-following [38] algorithm is used for path following. This means that A^* finds the path between the start and end point. The carrot-following algorithm then guides the vehicle along the found path. Modification in A^* are there because we want a path, which can represent a vehicle dynamic and does not have a sharp curve. We penalized this sharp change of direction.

Raspberry Pi is equipped with a 1.5 GHz quad-core ARM Cortex-A72 processor. The ARM architecture imposes challenges related to the missing official build of some libraries in Python, for example library TensorFlow. The main advantage of this computer is 8 GB RAM which is necessary for SLAM. SLAM is a complex algorithm that makes a map of surrounding space based on lidar data and the localized position of the vehicle in the map which corresponds with the position in the real world. 8 GB RAM is required for storing the map and its associated information, enabling optimization to accurately describe the real environment. A microSD with a capacity of 256 GB is used to store files. Raspberry Pi can generate two independent hardware pulse-width modulations (PWMs). This is useful for motor and lidar control.

4.4. Sensors deployed at the vehicle

The vehicle learns about the surrounding environment via various sensors connected to Raspberry Pi. Two main sensors are currently installed on the vehicle: lidar and camera. In this section, we describe the general information about these sensors and how these sensors are integrated into the vehicle for autonomous driving.

4.4.1. Lidar

In this subsection, we describe the lidar in detail. We look at the principle of lidar function, its parameters, and how we utilize the lidar in autonomous vehicle. The autonomous vehicle uses lidar RPLIDAR A1M8 [39]. Lidar contains one laser diode for sending a light beam and a photodiode that subsequently captures the reflected beam of light sent from the diode. In the case of the RPLIDAR A1M8 lidar, the beam is emitted in the infrared spectrum. The lidar operates on a time-of-flight basis. An illustration of this process is in Figure 8.

The distance d of the obstacle from the lidar is determined based on the time it takes for the beam to reach the obstacle as:

$$d = \frac{t}{2c}, \tag{17}$$

where t is the time of flight between the LED and the photodiode, and c is the speed of light in air and is equal to 299 792 458 $^{km}/_{s}$.

The lidar rotates on its axis and therefore the diodes rotate and scan the space in whole 360 degrees. Each scan makes the 2D point of obstacle in the created map. This gives us a 2D image in the form of a map of the surrounding space.

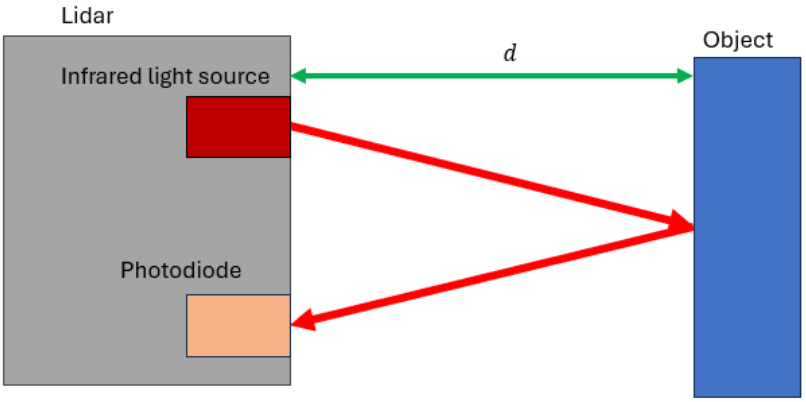

Figure 8: Illustration of how lidar works

The lidar rotates on its axis and therefore the diodes rotate and scan the space in whole 360 degrees. Each scan makes the 2D point of obstacle in the created map. This gives us a 2D image in the form of a map of the surrounding space. Each round involves approximately 1450 scans of the surrounding environment leading to the angle resolution of 0.25 degrees. The range of the laser is 12 meters, and the minimal measurable distance is 0.15 meters. Distance precision is 1% of the measured distance and angular precision is 1 degree [39].

The lidar RPLIDAR A1M8 rotates in the basic setting only at a frequency of 5 Hz. This frequency might be insufficient for some localization algorithms. However, the slow rotation of the lidar, inherent to the vehicle's design and its computing power, is leveraged as an advantage. The limited amount of data is compensated by utilizing the SLAM setting of the SLAM Toolbox [5]. Improving the localization process involves increasing the frequency of lidar rotation.

The frequency of lidar rotation is controlled by pin CTRL_MOTOR on lidar. In basic configuration is lidar motor powered by 5 V and the maximum frequency of rotation is only 5 Hz. This frequency is controlled by a pin CTRL_MOTOR and is set in the range from 0 Hz up to 5 Hz. According to [39], it is possible to power up the lidar motor up to 10 V to increase the maximum frequency of lidar rotation up to 17 Hz. We power the lidar motor with 9 V using a power bank and a USB-C PD/QC device, which enables direct extraction of 9 V from the power bank output. The power bank only allows the device to be powered with specific voltages. That is why we chose to power the lidar motor with 9V instead of 10V.

The input pin CTRL_MOTOR is controlled by PWMs, where the rotation speed is adjusted by varying the duty cycle of this PWMs signal. The generated PWMs has a frequency of 192 000 Hz and a duty

cycle ranging from 0% to 100%. This setup allows for fine adjustment of the lidar's rotation speed, ranging from 0 Hz to 15 Hz in an almost linear function.

The main challenge arises from the absence of the original connector between the lidar and the computer. This connector powers up the lidar as well as the lidar's motor by the 5 V and there is a special pin connected to the pin CTRL_MOTOR, which allows to change of the frequency of lidar rotation. The lidar communicates with the computer via a simple UART protocol, facilitating the use of a UART-USB converter TTL CP2102 [40]. However, this converter presents a limitation, since the direct control of the input pin CTRL_MOTOR, as with the original connector, is not feasible.

To address this limitation, a simple bridge is established between the Raspberry Pi and lidar in the form of two separate data cables, see Figure 9. In the figure G denotes ground, the number in blue circle denotes the data output of Raspberry Pi and red circle denotes power output. The bridge connects the input pin CTRL_MOTOR and pin GPIO12 in Raspberry PI, while the lidar pin ground power is linked to Raspberry Pi pin GPIO14. GPIO14 serves as the ground pin, while GPIO12 can generate hardware PWMs. This PWMs can simulate the behavior of the original connector and can control the lidar speed. The lidar rotation control script, written in C, adjusts the duty cycle of PWMs using hardware PWMs. This approach offers the advantage of reducing CPU load, thereby enabling independent operation from other tasks on the Raspberry Pi.

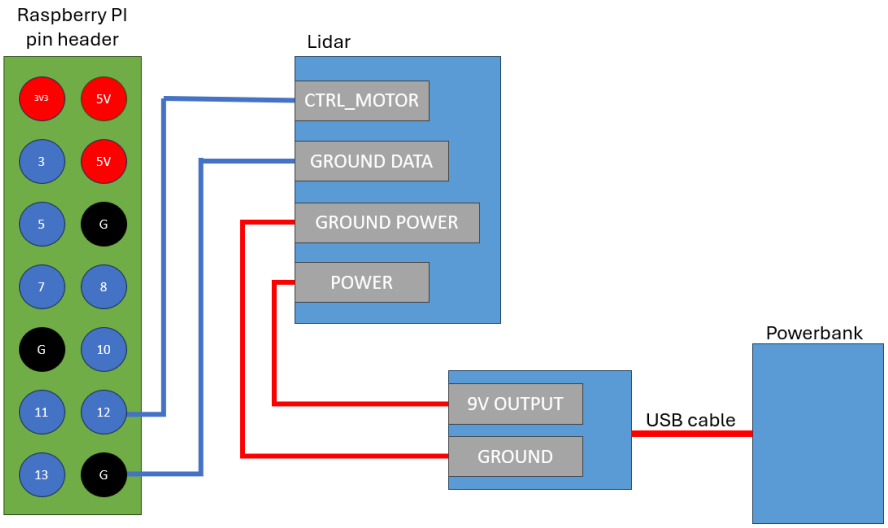

Figure 9: Bridge between lidar and Raspberry PI

The lidar is integrated with Robot Operating System (ROS) in order to enable easy control of the vehicle. ROS is an open-source framework that facilitates the development of robotic applications by providing essential libraries, tools, and packages [41]. It enables seamless integration and synchronization of various functionalities, including sensor communication, planning, control, and autonomy, through the concept of packages.

Integration of the lidar A1M8 to ROS is facilitated by the ROS package RP Lidar [42]. This package translates UART-formatted information received from the USB port of the Raspberry Pi into a ROS message, which is then sent to another package.

4.4.2. Camera

The camera is primarily utilized for capturing video of the surrounding environment to detect road signs and determine the type of sign. We installed a Niceboy STREAM webcam at the front part of the vehicle. This camera features an aperture of f/2.2 and can capture video up to a resolution of 1280x720 with a field of view of 90 degrees. In this thesis, we work with images with resolution of 832x832, 416x416, 224x224, and 128x128. The first image resolution must be interpolated using the OpenCV library to ensure the correct resolution of the image in the second dimension. The camera is connected to Raspberry Pi via a USB cable.

4.5. Control system

In this section, we introduce the basics of the control system and the interface between the Raspberry Pi and Traxxas control systems with the custom-made protocol for controlling this interface.

To enable autonomous driving of the vehicle, an interface is developed. Raspberry Pi sends commands to this interface, which then relays commands to the servo for rotating the front axle and the regulator. To facilitate autonomous or manual control of the vehicle, different modes are introduced. In the autonomous mode, indicated by a green light at the front of the car, see Figure 10, the vehicle is controlled by the Raspberry Pi. In the manual mode, signaled by no light at the front, the vehicle is controlled only by a human via a remote transceiver.

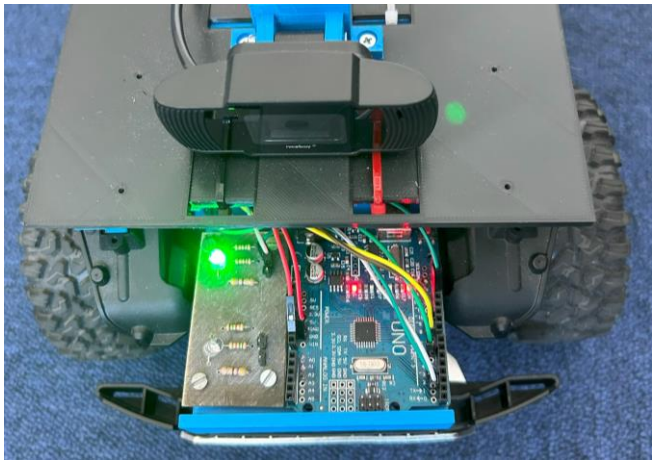

Figure 10: Light that signalized the autonomous mode

The mode change from autonomous to manual is initiated by turning the knob on the transmitter. Conversely, to switch from the manual to the autonomous mode, a special command is sent to the interface from the Raspberry Pi. To accommodate these modes, a microcontroller ATmega328P onboard Arduino Uno is placed between the receiver and the XL-5 HV controller.

Incorporating a microcontroller alongside the Raspberry Pi is essential due to several factors. Although the Raspberry Pi offers 2 hardware PWMs, a minimum of 3 separate PWMs is required to control various functions such as vehicle speed, front axle rotation, and lidar rotation speed. Additionally, this approach allows for the separation of control logic. While the Raspberry Pi generates control commands, the Arduino Uno handles the processing. Moreover, the Arduino enables remote stopping of the vehicle via the transmitter during autonomous operation, providing an additional layer of safety to mitigate any potential software bugs that could compromise the model's ability to stop safely.

4.5.1. Arduino Uno

The addition of the Arduino Uno adds another level of safety to the autonomous vehicle by separating the control and computational logic. This section explains its integration into an autonomous vehicle. The Arduino Uno is a widely used development board featuring the ATmega328P microcontroller, offering a straightforward and cost-effective method to incorporate microcontroller functionality into autonomous vehicles. With its array of input-output pins, USB interface for programming, and a user-friendly coding environment, the Arduino Uno facilitates seamless code development.

The Arduino Uno is linked to the Raspberry Pi through a USB connection, establishing communication via UART, and utilizing a dedicated text protocol. This custom text protocol is detailed in the Section 4.5.2. Additionally, the Arduino Uno is powered by the same USB cable used for communication with the Raspberry Pi, as the Arduino Uno's power consumption is approximately 50mA, allowing direct power supply from the Raspberry Pi.

To Arduino's GPIO are connected the outputs from the receiver and inputs are connected to the steering servo and speed controller. The Arduino Uno behavior is different in every mode. In manual mode, it simply takes signals on input pins and mirrors them on output pins. The vehicle is so fully controlled by the transceiver and operator. In autonomous mode, Arduino takes command from the Raspberry Pi. Arduino Uno generates two PWM, one for speed control and the second for steering control.

4.5.2. Communication protocol

This subsection introduces a simple text communication protocol between Raspberry Pi and Arduino Uno. This protocol is only one way in the direction of Raspberry Pi to Arduino Uno. The communication protocol is described below.

> "m\n" – Change from manual to autonomous mode.
>
> "n\n" - Change from autonomous to manual mode.
>
> "s\n" – Stop the vehicle.
>
> "s xxx\n" – Command to steer front axle. The xxx is the percentage of steering (-100, 100) -> (max right, max left)
>
> "v xxx\n" – Command to set speed. The xxx is the percentage of speed (-100, 100) -> (max speed backward, max speed frontward)

The crucial aspect to note is that each message must end with "\n" to ensure proper transaction termination, as the Arduino firmware requires this symbol to correctly finalize transactions. Without it, new commands will not be properly processed or evaluated. Between two commands from Raspberry Pi, there must be a space of about 50ms, to correctly evaluate command and change the duty cycle of PWMs. When this space is shorter Arduino Uno starts to act unpredictably and there is a risk of damage to the vehicle. After the transition from autonomous mode, the vehicle stops moving and is imminently controlled by the transceiver. This feature prevents damage to the vehicle or surrounding objects during testing.

5. Dataset in this work

This chapter explores the datasets used in the development of neural networks for road sign detection and classification. As discussed in Section 3.2, road sign recognition is divided into road sign detection and sign classification. For this reason, using two completely different datasets is also necessary. Additionally, in this chapter is a discussion about the sources, characteristics of each dataset and a discussion of the usefulness of these datasets for our purposes. Due to unsatisfactory results achieved with commonly used datasets for the detection part, we opted to create our own dataset. The problem of unsatisfactory result is more concreted in Section 5.2. The process of creating this dataset is also detailed in this chapter.

5.1. Detection part of datasets

This subsection delves into the application of neural networks for detection tasks, primarily focusing on road sign detection. Here, we provide a comprehensive overview of the datasets utilized throughout this analysis. The examined datasets are Open Images Dataset, and German Road sign Detection Benchmark.

5.1.1. Open Images Dataset V7

The Open Images Dataset V7 [11], managed by Google, stands out as one of the largest datasets available and offers different versions for different research needs. With 600 classes, it covers a wide array of objects, it provides a complex resource for training and evaluation purposes. The range of classes is wide from people to various insect species to vehicles. People can easily choose which classes they want and only these classes download. This is the biggest advantage over other datasets, which are usually downloaded with all classes. Within this study, the focus is directed only toward the road sign class from this dataset. In this class are about 1100 images for training and 18 images for validation. This class has biggest disadvantage is that most of the images are photos of road signs coming from Asian countries and therefore their applicability in Europe is very limited, see Figure 11 and Figure 12.

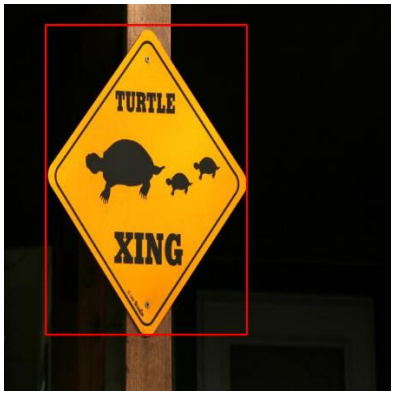

Figure 11: Road sign from Open Images Dataset v7 I.

Figure 12: Road sign from Open Images Dataset v7 II.

5.1.2. German Road sign Detection Benchmark (GTSDB)

The German Road sign Detection Benchmark [13] (GTSDB) was developed in 2012 by Ruhr-Universität Bochum Institut Für Neuroinformatik and includes a total of 600 training images and 300 validation images, each annotated with bounding boxes and class names. This dataset encompasses 43 different classes of road signs commonly found on German roads, such as stop

signs, yield signs, and no-passing signs. This dataset is commonly used in various benchmarks focused on road signs in Europe.

Notably, the road signs in this dataset tend to be smaller in size compared to those in the Open Images Dataset V7. This poses a challenge, as the road signs in our case occupy a larger area in the camera picture than those in this dataset. This imbalance may lead to the neural network erroneously learning to detect them.

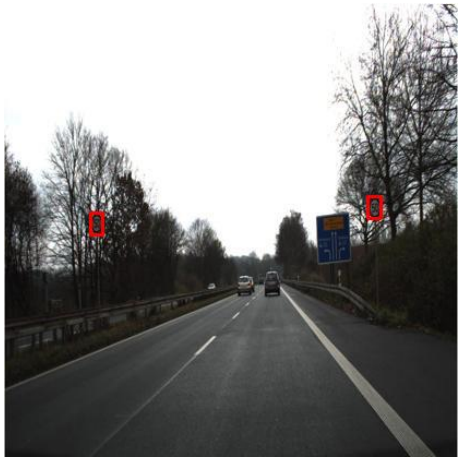

Figure 13: Picture from GTSDB I.

Figure 14: Picture from GTSDB II.

5.2. Custom dataset for detection of road signs

Open Images Dataset V7 has many different road signs mainly from Asian countries and the USA, not identical (in some cases even not similar) to European road sings. Additionally, there are also billboards and directional boards in this road sign class in this dataset. The second issue is the insufficient quality of labels. Many labels are missing or do not correspond to real road signs. This is because the Open Images Dataset V7 is open, and every registered user can add new images and labels to this dataset.

GTSDB has a problem with an insufficient number of images for training. Even if training and validation data is connected into a big training set, the number of pictures is only 900 and YOLOv3 needs about 1500 labeled images per class. The second problem is bigger because data augmentation addresses the size of the road signs on images. Images are photos from a car perspective and the road signs occupy a small place in these photos and do not fully represent the data obtainable from the autonomous vehicle's camera.

Therefore, we create a small dataset comprising 40 labeled images from validation captured from a vehicle camera. Using the previously mentioned datasets, we trained YOLOv3 to detect road signs. Subsequently, we applied YOLOv3 to the validation dataset to detect the road signs. The results, presented in Table 1, are evaluated in terms of mean average precision (mAP) by IoU between the predicted label and ground true label at 0.45. The network is trained in two different image resolution: 416 pixels and 832 pixels, with the training process sustained at 200 epochs.

Table 1: Validation results of neural network accuracy learned on different datasets

Dataset	Image resolution 416 pixels	Image resolution 832 pixels
Open Images Dataset V7	60.8	65.4
GTSDB	20.1	47.9

For autonomous vehicle properties, we need precision as high as possible and mAP 65% is not sufficient for utilization in autonomous driving. There is a danger when the road signs are recogized incorrectly to damage the surrounding environment or cause loss of human life. Considering these factors outlined above, the decision is made to develop a new dataset that accurately reflects the vehicle's environment.

5.2.1. Selection of road signs for dataset creation

In this chapter, we discuss the selection process of road signs for our autonomous vehicle system. We selected 7 types of road signs. This selection includes:

- Speed limitations 20 km/h, 50 km/h, and 80 km/h
- End speed + passing limits
- No vehicles
- Priority road
- Stop

We have selected these road signs to represent frequently used road signs. At the same time, we wanted these road signs to be represented in the dataset used for road sign classification. Several speed limits are chosen because they are similar and therefore it is interesting to compare how accurately they are classified. Finally, it is considered that the individual signs can interact with the autonomous vehicle. They could limit their maximum speed or modify the rules to avoid obstacles. The frequency of use of road signs in the real environment play an additional role. This frequency of use is related to the occurrence of signs in the GTSRB dataset that is used to learn the neural network for road sign classification, as we wanted to test whether even road signs with lower number of training images would be reliably recognized.

We draw the individual road signs in Canva and then we create cardboard stands for them, see Figure 15. The stands are created to be as stable as possible and therefore do not fully represent traditional road sign stands, which is anchored into the ground and therefore do not need such a large base.

Figure 15: Cardboard stand for road signs

5.2.2. Process of creation of the new dataset

Photos for this dataset are created using an autonomous vehicle and a webcam mounted on it. The vehicle is driving across the rooms at the Faculty of Electrical Engineering at the Czech Technical University., and every second camera took a picture. We have captured over 2600 images representing the environment where the vehicle operates, including road signs. Afterward, a selection process is undertaken, resulting in the selection of 1303 images. These excluded images typically lacked road signs or are too blurred to recognize any features. For this task, we created a Python script. This script loads and show in window every image in certain folder. By pressing key "s" is image deleted and by pressing "a" or "d" we can move in the images.

After this procedure, we need to create a bounding box for every image. For this labeling process, we use the software Roboflow for making bounding boxes in these images [43]. After creating dataset, we divide the dataset into training and validation images using a ratio of 95:5. Images from the created dataset are shown in Figure 16 and Figure 17.

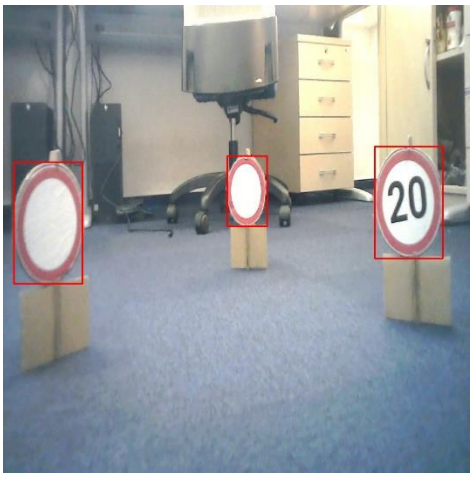

Figure 16: Images from own dataset I.

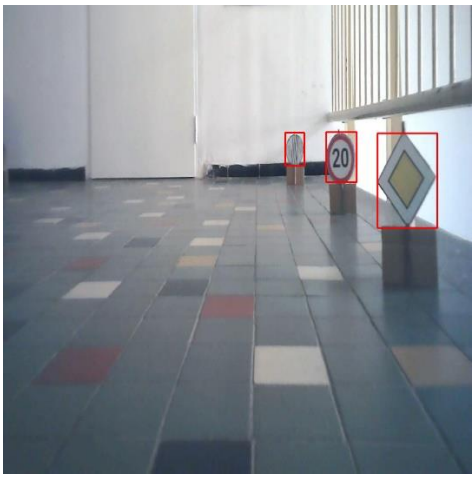

Figure 17: Images from own dataset II.

5.3. Classification part of dataset

This chapter explores dataset for classification tasks to classification of road signs. The dataset used in this thesis is German Traffic sign Recognition Benchmark (GTSRB). GTSRB [44] is a comprehensive dataset comprising 39 209 images of road signs, categorized into 43 classes. These images are sourced from German roads and serve as a benchmark for training and validating algorithms for road sign classification. This dataset was developed by Ruhr-Universität Bochum Institut Für Neuroinformatik in 2012. Each image in this dataset is manually annotated, providing information about the sign class. This dataset is used for training neural networks for road sign classification because is the biggest and the most coveted among the named datasets. The main advantage of this dataset is that it contains a large number of road signs images under different lighting conditions. The disadvantage, on the other hand, is the uneven representation of individual road signs [12]. In every image, only one road sign is included and nothing more, hence, there is not enough context for YOLOv3 to detect position of the road sign on more complex pictures. Therefore, we need to employ two neural networks how is described in Section 3.2.

6. Implementation of neural network

In this chapter, we discussed the selection of neural networks for detection and classification. We also show implantation details of neural networks for road signs detection and classification as well as the pipeline for the road sign detection and classification. The structure of implemented codes is in Appendix B.

6.1. Neutral networks for detection task

In this section, we select the neural network for road sign detection that best fits our requirements for use on an autonomous vehicle. To benchmark neural networks, we utilized two metrics mAP, which indicates the ratio of correctly detected objects to all detectable objects, and Frames Per Second (FPS), which measures the rate at which images are evaluated per second using the neural network on the reference graphics card. The Nvidia Titan X is used as the reference graphics card in this case. These metrics for Faster R-CNN, SSD and YOLOv3 are in Table 2. Three neural networks mentioned above represent commonly used networks for object detection in images.

Table 2: Neural network for detection comparison [14]

	mAP [%]	FPS [Hz]
Faster R-CNN	83.8	0.4
SSD	76.8	19
YOLOv3	78.6	40

By comparing networks, we choose the YOLOv3 to implement in this thesis for road signs detection, because it has higher accuracy than SSD and is much faster than Faster R-CNN. It makes a compromise between network speed and precision.

6.2. Neutral networks for classification task

In this section, we select the neural network for road sign classification that best fits our requirements for use on an autonomous vehicle. Commonly used networks are ResNet-152 [17], VGG-16 [18], GoogleNet [19], and AlexNet [18]. We compare These neural networks at the base of accuracy of corrected classified image and number of layers. These parameters are in Table 3.

Table 3: Neural network for classification comparison

	accuracy [%]	number of layers [-]
ResNet-152	87	153
VGG-16	71.5	16
GoogleNet	78.2	22
AlexNet	57	8

Visual Geometry Group (VGG) and AlexNet are not feasible for our work, because these do not achieve high accuracy in image classification. In contrast, GoogLeNet and ResNet are very deep and computationally heavy. For these reasons we create a custom neural network for road signs classification. This neural network is shallower than the neural networks described above and achieves better accuracy by tuning the network to classify specifically road signs.

6.3. Implementation of YOLOv3

This chapter delves into the implementation of YOLOv3 neural networks, covering support functions, metrics, network structure, learning processes, and hyperparameter settings. Our focus on object detection in images led us to select YOLOv3 for its speed and accuracy, especially in detecting multiple objects simultaneously. It achieves 40 fps and 78.2 % mAP. While there are numerous existing implementations of YOLOv3, we opted to customize one for our specific needs in optimizing road sign detection for our autonomous vehicle system.

Given the abundance of available implementations optimized for speed and performance, it seemed impractical to develop our own from implementation. Instead, we chose to leverage an existing implementation and modify it to our requirements. We selected an implementation referenced in [45] for its thorough documentation. However, upon testing, we encountered several bugs resulting mainly from compatibility issues with updated Python and supporting libraries. Despite the difficulties, we decided to fix the existing implementation rather than invest time in familiarizing ourselves with a new one. The original implementation does not consist of an evaluation script. So, we need to create a new one.

6.3.1. Description of supported function

In this subsection, we describe the support function used in the process of YOLOv3 training and evaluation, that we take from original implementation.

6.3.1.1. Intersection over union

The provided Python function *intersection_over_union()* calculates the IoU metric, which measures the overlap between predicted bounding boxes and ground truth bounding boxes. The function takes two sets of bounding boxes as input: *boxes_preds* containing predicted bounding boxes and *boxes_labels* containing ground truth bounding boxes. These bounding boxes are represented in the midpoint format with center position and width and height (x_c, y_c, w, h).

The IoU computation is performed by first extracting the coordinates of the bounding box from the input tensors. Then, the function calculates the corners of the bounding boxes using this formula:

$$\mathrm{box1}_{\mathrm{x1}} = x_{c1} - w_1/2, \tag{18}$$

$$box1_{y1} = y_{c1} - h_1/2, \tag{19}$$

$$\mathrm{box1}_{\mathrm{x2}} = x_{c1} + w_1/2, \tag{20}$$

$$box1_{y2} = y_{c1} + h_1/2, \tag{21}$$

$$\mathrm{box2}_{\mathrm{x1}} = x_{c2} - w_2/2, \tag{22}$$

$$box2_{y1} = y_{c2} - h_2/2, \tag{23}$$

$$box2_{x2} = x_{c2} + w_2/2, \tag{24}$$

$$box2_{y2} = y_{c2} + h_2/2, \tag{25}$$

where x_{c1}, y_{c1}, x_{c2} and y_{c2} are coordination of bounding boxes centers and h_1, w_1, h_2 and w_2 denotes the height and width of bounding box.

Then, the function calculates the coordinates of the intersection rectangle between the predicted and ground truth bounding boxes using the formula:

$$\mathrm{x1} = \max(box1_{x1}, box2_{x1}), \tag{26}$$

$$\mathrm{y1} = \max(box1_{y1}, box2_{y1}), \tag{27}$$

$$x2 = \min(box1_{x2}, box2_{x2}), \tag{28}$$

$$y2 = \min(box1_{y2}, box2_{y2}), \tag{29}$$

where x_1, y_1, x_2 and y_2 are coordinates of corners of intersection rectangle. Subsequently, the function computes the intersection area i by:

$$i = (x_2 - x_1)(y_2 - y_1), \tag{30}$$

and the area of the union u by:

$$u_1 = (box1_{x2} - box1_{x1})(box1_{y2} - box1_{y1}), \tag{31}$$

$$u_2 = (box2_{x2} - box2_{x1})(box2_{y2} - box2_{y1}), \tag{32}$$

$$u = u_1 + u_2 - i. \tag{33}$$

Finally, we can compute the IoU:

$$\mathrm{IoU} = \frac{i}{u + 10^{-6}}. \tag{34}$$

Note that a small epsilon value 10^{-6} is added to the denominator to avoid division by zero errors.

6.3.1.2. No max suppression

The non-maximum suppression (NMS) algorithm is a post-processing step commonly used in object detection tasks to filter out redundant bounding boxes. YOLOv3 found many bounding boxes of same objects, the NMS algorithm removes overlapping bounding boxes and retains only those with high confidence scores.

The NMS function begins by filtering out bounding boxes with confidence scores below a specified threshold, called *CONF_THRESHOLD*. Next, the remaining bounding boxes are sorted based on their confidence scores in descending order. The algorithm iterates through the sorted list of bounding boxes, selecting the box with the highest confidence score called the chosen box at each iteration, then removes all other bounding boxes with the same class that have a higher IoU with the chosen box than threshold *NMS_IOU_THRESH*. This prevents significant overlap of bounding boxes with the same class. After iterating through all bounding boxes, the function returns a list of bounding boxes that have passed the NMS process, ensuring that redundant detections are eliminated.

6.3.2. Code structure

In this subsection, we describe the whole process of training or evaluating the neural network. For clarity, the code is divided into several modules, and we describe each of them individually. The main modules are model, dataset, loss, training, and evaluation. Support function play config file and utils where are implemented function from Section 6.3.1. In config file, every constant for the YOLOv3 neural network is specified. For code structure see Figure 18.

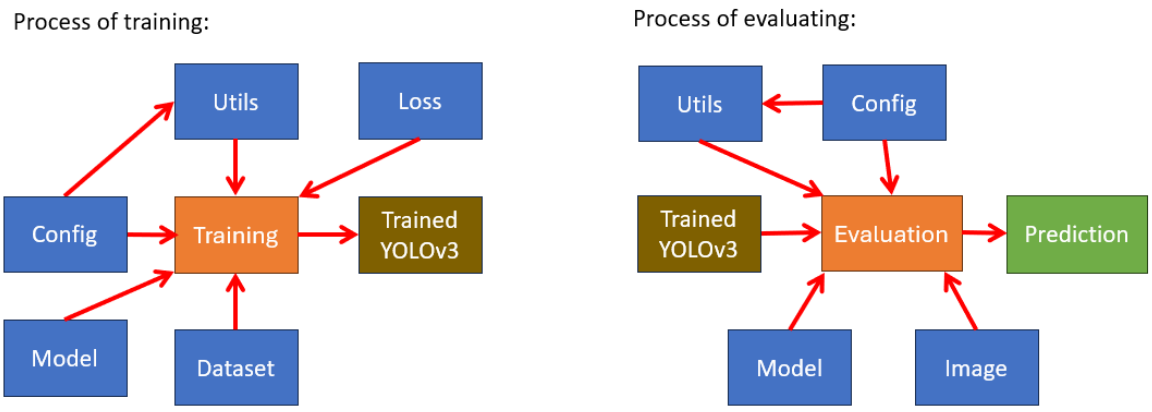

Figure 18: Structure of YOLOv3 imp

6.3.2.1. Model

The model is taken over from original paper [15] without modification. At the beginning of this module, the neural network in text format, is described in Figure 1. The main code intenerates through this text description and adds blocks or layers to the neural network model. The blocks are described in Section 2.2. The forward function is straightforward and is implemented with the skip connections.

6.3.2.2. Dataset

This module defines the class *YOLODataset*. This class loads the dataset for YOLOv3 training in the correct format. Class *YOLODataset* is based on a class dataset from the library PyTorch and inherits the methods *__len__()* and *__getitem(index)__*. These methods are modified to use with new datasets. The dataset format is described in Appendix A, together with the adaptation of Open Image V7 and GTSDB datasets. First, we describe the methods and then we describe the transformations in this dataset which is the main part we must correct because the original transformation did not work.

6.3.2.2.1. __init__()

Method initializes the YOLODataset class. It accepts several arguments, including the paths to images and labels, as well as the path to a CSV file, image resolution, number of classes, anchor boxes, scale of anchor boxes, and an argument to allow or block transformation. Previous arguments are stored as class variables for future reference. Image names and corresponding labels are loaded from CSV files using the Pandas library and *the read_csv()* function.

6.3.2.2.2. __len__()

Method is included here for comprehensive understanding. It calculates the total number of items in the dataset based on the length of the class variable annotations. The variable annotations is an array containing pairs of image filenames and their respective labels, loaded during the initialization process using the *__init__()* method.

6.3.2.2.3. __getitem(index)__()

Method *__getitem(index)__()* must be rewritten because the original implementation does not work and has several deficiencies. The index says which item should be loaded. First, the method loads the image and corresponding label. The image is loaded with the help of the Python library PIL. The image is resized to the corresponding resolution set by the class variable initialized during class initialization. Block of code that transforms the image to provide greater dataset diversity is used, after resizing. Transformations are described in the Section 6.3.2.2.4. Then is image transformed to Numpy array. After the image transformation, we need to re-arrange the image dimension from image resolution x image resolution x 3 to a resolution 3 x image resolution x image resolution. We need to re-arrange the original image shape because of the convolutional layer, that accepts only images in this resolution. Re-arrangement is done in three for cycles. After for cycles is image ready to be put into the neural network and we need to load the labels.

Target values are crucial for model training as they specify the expected outputs for each input image. The algorithm iterates through each bounding box in the input data, determining which grid cell the box's center belongs to. Algorithm then selects the corresponding anchor box based on the overlap with predefined anchor boxes. If the selected anchor box is free and does not already contain another box, information about the box is stored. If the overlap with the anchor box exceeds a specified threshold *ignore_iou_thresh*, the anchor box is marked as ignored. This process of image and label preprocessing is repeated for each input image, resulting in a set of target values used during training the YOLOv3 detection model.

6.3.2.2.4. Transformations

Transformations are a crucial aspect of preprocessing data in computer vision. Transformations help increase the diversity of training data, thereby enhancing the model's ability to generalize new data. We used several common image transformations, including horizontal and vertical flipping, adding noise, changing brightness, and changing contrast. Each transformation is applied randomly with a certain probability, ensuring that each training image is biased in multiple ways, which increases data diversity and helps the model adapt better to various situations.

The code initiates by randomly selecting whether to perform horizontal, vertical, or both flipping of the image, or leave it unchanged, with equal probabilities assigned to each option. If transformation is chosen, the image is modified by using functions *ImageOps.mirror* or *ImageOps.flip* from the Python library PIL. Subsequently, the code randomly decides whether to introduce noise, adjust brightness, or modify contrast. Each transformation is applied independently, contributing to the diversity of the training data, with a 50% probability for each transformation. To adjust brightness and contrast, the images are loaded using the function *ImageEnhance.Brightness()* and *ImageEnhance.Contrast()*. Recent functions are then followed by a *call .enhance()* with a random argument determining the transformation's effect on the image. Adding noise involves generating an array of the same size as the original image, filled with random values, which are then added to the original image.

After applying horizontal or vertical flipping to the image, the bounding box coordinates are adjusted to ensure consistency between the image and its description. However, the bounding

boxes remain unchanged for contrast, brightness, and noise transformations. When a horizontal flip is applied, the change in the center coordinates of the bounding box is determined by the following equations:

$$x_1 = 1 - x_2, \tag{35}$$

$$y_1 = y_2, \tag{36}$$

where x_1 and y_1 denote the original coordinates of the center of the bounding box, and x_2 and y_2 denote the new coordinates of the bounding box. The change in the coordinates of the center of the bounding box, to which the vertical flip is applied, is determined by the following equations:

$$x_1 = x_2, \tag{37}$$

$$y_1 = 1 - y_2. \tag{38}$$

6.3.2.3. Loss

The loss function is straightforward implantation of equitations described in Section 2.2. We mainly change the constant λ_{class} in loss function because we only classify one class, we put λ_{class} to zero and do not consider a class loss. Setting λ_{class} to zero helps us to put more effort into other losses, especially to box loss. The loss function is implemented as a class with two methods __init__() and forward(). Method __init__() prepares a function of loss by (11). Method *forward()* takes as arguments predicted and true bounding boxes as well as anchor boxes and computes a numerical value of loss by function initialized in the previous method.

6.3.2.4. Training

Module training starts with loading the model of the YOLOv3 network and initializing the optimizer. The task of the optimizer is to find the optimal values of the model parameters to achieve the highest results when predicting unknown data. After optimizer initialization, the loss function and scaler are initialized. Scaler [46] is a special method from the library PyTorch that helps with numerical stability in the learning process. After initialization are loads datasets for training and validation as well as the initialization weights. Initialization weights are loaded from a file or randomly initialized. The last initialization function is preparing the anchor boxes by scaling anchor boxes to the appropriate size for every of three grid sizes.

Training is done in for cycle. The number of repeats of for cycle is typically in the lower hundreds. The training procedure is looped over the images. The images and labels are converted to float format and are moved to the graphic card (GPU) if Cuda is enabled, if not they are moved to the processor (CPU). Then image is sent to the network and the predicted output with labels is sent to the loss function separately for every grid size. After this, loss is backpropagate through the network.

The process of backpropagation looks as follows. Firstly, *optimizer.zero_grad()* is called to reset the gradients of all model parameters to zero before computing gradients for the next batch of training data. Then, *scaler.scale(loss).backward()* computes the gradients of the loss concerning the model parameters, scales the loss value using the gradient scaler, and performs the backward pass to calculate the gradients. The *scaler.step(optimizer)* method updates the model's parameters using the gradients and the optimization algorithm specified by the optimizer. Finally,

scaler.update() adjusts the scale factor used for the next iteration based on whether gradient values overflowed or underflowed during the computation, ensuring numerical stability.

After the process of training, every 10 epochs is the process of validation. Before evaluating we need to change the mode to evaluation mode by call *model.eval()* and after evaluating we need to change the mode to training mode by call *model.train()*. The rest of the validation process is described in Section 6.3.2.5, but we used the weights from learning and did not load the new one. After validation, we calculate the mAP, based on predicted and true bounding boxes. We save the checkpoint every 10 epochs and note the weights with the largest mAP.

6.3.2.5. Evaluation script

The correctness of the predicted classes should also be checked. However, since we have only one class to predict, we do not perform classes checked. In the process of validation are loaded the validation images with their labels as well as checkpoint specified in the config file. The images as well as labels are moved to GPU. Then, the model generates predictions for the input data. Predictions are processed to extract bounding boxes for each image. Bounding boxes are generated for each grid size of the model's output and are consolidated to form a single list for each image. Additionally, the true bounding boxes are obtained from the ground truth labels. NMS is applied to the predicted bounding boxes to filter out redundant detections, based on an IOU threshold and confidence threshold. Filtering by NMS is done because the YOLOv3 network finds several bounding boxes from different grind sizes, and we try to find the most suitable one for each search object. After getting true and predicted bounding boxes we compute the mAP. The process of computing the mAP is mathematically described in Section 7.3.1.1. To this point is the validation the same as in training.

Now we want to draw the true and predicted bounding boxes in the images. That is possible to make it by library OpenCV and its function *rectangle()*, which draws rectangles into images. Function *rectangle()* takes as the argument the corners of the bounding boxes in pixels and the array that represents the image. The process of computing the coordinate of corners x_1, y_1, x_2 and y_2 is follows:

$$x_1 = \left(x_c - \frac{h}{2}\right) \times s, \tag{39}$$

$$y_1 = \left(y_c - \frac{w}{2}\right) \times s, \tag{40}$$

$$x_2 = \left(x_c + \frac{h}{2}\right) \times s, \tag{41}$$

$$y_2 = \left(y_c + \frac{w}{2}\right) \times s, \tag{42}$$

where x_c and y_c are relative coordinates of center of the bounding box, variable h and w are height and width of the bounding box and s is the resolution of the picture in pixels. The program first loads the image and computes the corners of bounding boxes in pixels, draws the rectangles in the image, and then stores the result, see Figure 19, where red rectangles are ground true bounding boxes and yellow rectangles are predicted bounding boxes.

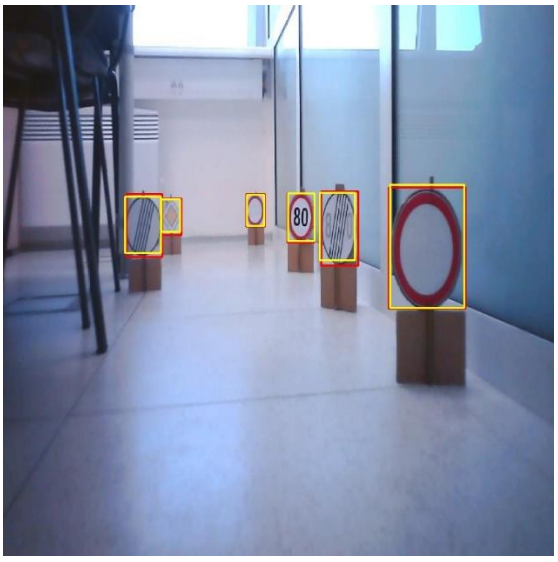

Figure 19: Example of road sign detection script output

6.3.3. Process of training and setting the hyperparameters

The process of training is essential for the performance of neural networks. The neural network learning process runs on an Intel NUC with an Nvidia RTX 2060 graphics card. This process is influenced by selecting so-called hyperparameters. We describe hyperparameters and their function in this subsection. First, we need to select a suitable optimizer for our task. The most used methods are Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam). When utilizing both optimizers, we observed a similar reduction in loss throughout the training epochs, without experiencing the common occurrence of loss oscillating around a certain value.

SGD has two tunable parameters, the learning rate and momentum. We achieved the highest result by using the learning rate 5e-5 and the momentum is set to value 0.9. Adam also has two tunable parameters learning rate and weight decay. We set these parameters to 5e-5 and 5e-4. The chosen learning rates are relatively small, prioritizing precision in learning. However, this precision comes with the trade-off of longer training times. We also set the confidence threshold to 30%, which means that YOLOv3 must be at least 30% confident with the bounding box found. Confidence is one of the outputs of YOLOv3 and NMS threshold is set to 0.15.

The next parameter we adjust is the image resolution. Given the architecture of the network and the dimensions of the camera images, we have several options: the original resolution of 416 pixels or the newly created resolution of 832 pixels. We compare the accuracy of road sign detection for both resolution, as shown in Table 1. We also train weights for image resolution 224 and 128 pixels for experiments with payload size on 5G network. The achieve mAP are about 95% with IoU 0.5 for both images resolutions. The choice of image resolution is closely linked to the batch size, which determines how many images are processed by the YOLOv3 simultaneously. With an image resloution of 416 pixels, we can use a batch size of up to 8 images, but with a resolution of 832 pixels, only a batch size of 2 is feasible. Batch size limitation is due to the size of the VRAM in the GPU and the caching of images. A larger batch size helps prevent overfitting and improves the efficiency of gradient backpropagation [47]. The last parameter connected to the image is the number of workers which say how many threads load the image and we use 4. The images are loaded in random order.

The last thing that we need to tune are the dimensions of anchor boxes. For this design, we based it on the original anchor box setup [29]. Unfortunately, the anchor box sizes are not mentioned in the original paper. This original anchor is edited to be mainly square because the bounding boxes of road signs are mainly square. This may not be directly visible from the size of the anchor boxes as they are square at the original resolution. However, when preprocessing, the image size is resized asymmetrically in both dimensions and thus the size of the anchor boxes is also resized asymmetrically. This resizing of the bounding box helps with the learning rate, and the neural network learns within 100 epochs to a value of 80% IoU per mAP of 99% instead of the original 0.98 at 200 epochs on image resolution of 832 pixels. Trained weights for YOLO with created dataset are available on OneDrive[2].

6.4. Implementation of neural network to road sign classification

In this chapter, we introduce the neural network for road sign classification, we also describe neural network architecture, neural network implementation, process of training and evaluation as well as discussion about the results. Our goal is to create a lightweight neural network with a high level of confidence in the predicted class. Neural networks described in Chapter 6.2 are very deep or have a small level of corrected predicted class. The insufficient accuracy is caused because the mentioned neural networks are very generative. From the newly created neural network, we are able to better train it on the specific features contained in road signs.

Neural networks accurately classify various road signs, including speed limits or stop signs from visual input captured by onboard cameras. Capability of recognition road signs enables autonomous vehicles to interpret and respond to traffic regulations and road conditions effectively, contributing to safer and more reliable autonomous driving experiences.

6.4.1. Architecture of neural network

In this chapter, we discussed the model of a neural network and its layers. We also slightly describe the process of modeling the neural network. The architecture of the neural network is in Figure 20.

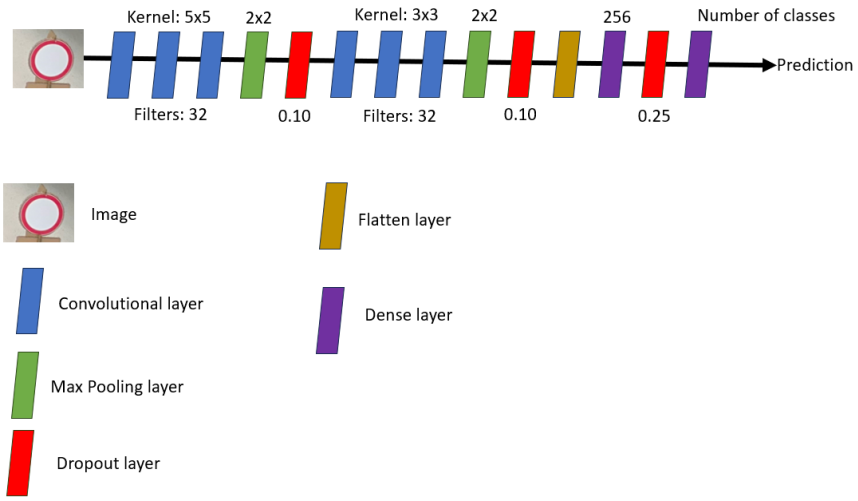

Figure 20: Model of YOLOv3

[2] https://campuscvut-my.sharepoint.com/:f:/g/personal/danekja5_cvut_cz/EuRQ8S9fe5tItubM9KMra9QBv0EVYAgu3X3989riBVQe4g?e=xQ2Rzd

The output of neural network is a one-dimensional array of the size of the number of classified classes. Each entry in the array contains the probability of how closely a given index matches a classified road sign with the same ID as the element in the index array.

In Figure 20 number above max pooling layer is kernel size, number above dropout layer is dropout ratio and number above dens layer is output size. Max pooling is a technique used to reduce the spatial dimensions of an input feature map by selecting the maximum value from each local region. This process helps to extract the most important features while discarding irrelevant information. Dropout is a regularization technique used to prevent overfitting by randomly setting a fraction of input units to zero during training. This process helps to reduce the reliance of the model on specific features. As a result, dropout improves the generalization ability of the network and increases its performance on unknown data. The flatten function reshapes a multi-dimensional tensor into a one-dimensional tensor, typically used to convert the output of a convolutional layer into a format suitable for feeding into a fully connected layer. In the dense layer, also known as the fully connected layer, is each neuron connected to every neuron in the previous layer, propagating information from all input neurons to all output neurons. Dense layer is used for learning complex patterns in data.

When designing the network, we started from the back. First, we specified the format of the inputs as an array of probabilities. This is to be taken care of by the flatten function. However, since the flatten function is not sufficient and would only extract the probabilities of the features from the previous layers, it is necessary to link these features to specific road signs, and two fully connected layers took care of this since one proved insufficient. Two classic convolutional blocks are used to extract the features, each containing 3 convolutional layers and one max pool layer. These blocks are standard in image classification using a convolutional network. Since overfitting is occurring during learning, it is necessary to add dropout layers to help reduce overfitting. These layers are added at the end of each convolutional block.

Activation function ReLU is chosen for each convolutional layer. ReLU function is used because it can highlight individual features in a computationally inexpensive way. The last dense layer has an activation function softmax. Softmax function normalizes the output of the layer so that the sum of its output adds up to 1 to produce the probability values for each class. The rate values for the dropout layer are determined empirically based on the observed outputs and received accuracy.

Given the size of the bounding boxes found in the road sign detection, we decide that the input image resolution for this neural network would be 30x30 pixels. Since the road signs occupy the entire image area, we decide to use a 5x5 kernel for the first convolutional block to extract the large features and then we use a 3x3 kernel in the second convolutional block to find the smaller features in more detail. The number of filters for each block is then determined as a compromise between model complexity and classification accuracy. Based on empirical findings, the values are determined as 32 filters for the first convolutional block and 64 filters for the second convolutional block. The remaining values in the convolutional network are computed so that the layers are dimensionally related to each other.

6.4.2. Implementation

In this section, we describe the implementation of a neural network for road sign classification. Neural network is implemented in Keras. Keras is more suitable for small neural network than PyTorch, where we need to make complex structures for dataset loader or even to create a neural network model. Unlike in YOLOv3, implementation of neural network for road sign classification is not divided into modules and the network is sustained from two files, each representing the Python script. The first script loads the dataset and creates the model, then the script trains the model, and there is an evaluation process to validate the neural network on validation data from the dataset. In the second script load the images and then go through the process of evaluation. The second script is made for evaluating the images without known class.

6.4.2.1. Training script: Load the dataset

The GTSRB dataset is used to train the neural network. The format of the dataset is folded-based. All the labels of a given class are placed in one folder which is named according to the class ID. This dataset contains a total of 43 classes.

Training script first loads images from the dataset directory, which contains folders labeled with numbers from 0 to 42, each representing a different class. Images are loaded using library PIL and resized to a dimension of 30x30 pixels. Images are stored in the data list, and the corresponding labels are stored in the labels list.

Subsequently, the data and labels are converted into Numpy arrays and then split into training and validation datasets using the *train_test_split()* function from the Python library Sklearn. Split is made in a ratio of 80% for training data and 20% for validation data. Both split data is used in the process of training. By splitting data we see if the probability of successful prediction increases during training of this network. Then the labels are converted into the format suitable for learning by using the function *to_categorical()* from the library Keras.

6.4.2.2. Training script: Initialize and training of neural network

To initialize the neural network, we use the function *sequential()*, and then by using the function *.add()* we add to network the layers from Figure 20. The process of training the neural network is simpler than in the case of the YOLOv3 network. After initializing the network, we compile neural network model by using the function *.compile()*. We specified the optimizer and loss function in *.compile()*. We selected Adam as the optimizer and the cross entropy as a loss function. Cross entropy loss function is commonly used in image classification and cross entropy loss function is mathematical formulated in similar way as class loss from YOLOv3 described in (11). We don't need to adjust the learning hyperparameters because the default setting is sufficient for our purposes. We start the process of learning by calling function *.fit()* on the model. In function *.fit()* we specify the batch size as 32 and the number of epochs as 15. We also pass to *.fit()* function on to the load data including the labels. This is enough to train the neural network. We train neural network on CPU i5 13th generation. Each epoch takes approximately 1 minute to process. After the successful training of the neural network, the whole network is stored in the file for further use.

6.4.2.3. Training script: Evaluation of neural network

We used the different images, in this part of the script, for the validation part and to test the final accuracy. The GTSRB dataset provides a folder with images for validation. The process of validation begins by loading the test dataset from a CSV file by using the library Pandas. This CSV file contains information about the images and image's corresponding labels. Then, script iterates through each image in the test dataset and makes the same preprocessing of images as in the

training part, the code feeds these images into the trained neural network model to make predictions about their labels.

Following the predictions, the script determines the predicted label for each image by selecting the class with the highest probability from the model's output. Once all predictions are obtained, the code compares them with the true labels from the test dataset to calculate the accuracy of the model's predictions by using the function *accuracy_score()* from Sklern. Finally, the achieved accuracy score is printed to the console, providing an evaluation of how well the model performs on unknown data. Trained model of neural network for road sign classification is available on OneDrive[3].

6.4.2.4. Evaluation script

The evaluation script for evaluating unknown data is almost the same as the part for testing the images in the previous part. To difference is in loading the model from the file by using the function *load()*. Images with unknown labels are loaded and fed into the load model. Subsequently, the code determines the predicted label for each image by selecting the class with the highest probability from the model's output and writing the predicted class to the console.

6.5. Pipeline of neural networks and implementations on the autonomous vehicle

In this section, we discuss the implementation of the pipeline of the neural network for road sign recognition. We start by obtaining images of the environment with road signs from the camera and end by sending information about localized road signs to control algorithms. The scope of this thesis does not include control algorithms. Control algorithms are for path planning and path following or avoiding dynamic obstacles. The Raspberry Pi, utilized within an autonomous vehicle, lacks sufficient computational power to execute real-time neural network-based detection and classification of road signs.

For completeness, the road sign detection neural network runs on a Raspberry Pi for an average of 26 seconds per image, and on a computer with an RTX Nvidia 2060 graphics card it runs for an average of 1.1 seconds. A response time of 26 seconds is insufficient for autonomous driving. Time for detection do not counting the time to classify road signs, which on the Raspberry Pi is about 400 ms per sign. During the evaluation of the captured image, it is not possible to utilize the Raspberry Pi processor, as the neural network computation fully occupies it. Consequently, there are no computational resources available for the control algorithms. For this reason, it is decided that the captured images are sent to the MEC server using the 5G network. In this scenario, partial offloading is not feasible, and all data for the neural network must be consistently offloaded to the server. This ensures that the complete evaluation of the neural network occurs on the server. But for test purposes we also implemented the possibility to perform road sign detection and recognition on autonomous vehicle. The Camera node (describe for MEC processing in Section 6.5.1) is corrected for testing purposes. According to the message from the manger, which controls the offloading process, and is out of scope this thesis, camara node decide whether to process the pipeline of neural network (describe in Section 6.5.2) locally in the vehicle or send the image to the MEC server for evaluation.

[3] https://campuscvut-my.sharepoint.com/:f:/g/personal/danekja5_cvut_cz/EuRQ8S9fe5tItubM9KMra9QBv0EVYAgu3X3989riBVQe4g?e=xQ2Rzd

A pipeline for road sign detection and classification on MEC is made of this part:

- Capture an image of the surrounding environment
- Send the image to the MEC server
- Road sign detection
- Road sign classification
- Estimation of the position of the road sign
- Send obtained data to control algorithms on the vehicle control algorithms

We discussed the individual parts of this and how we modify them to run on the MEC server in the fastest possible time given by the implementation pipeline in the rest of this chapter.

6.5.1. Capture the images with a camera mounted on an autonomous vehicle

This part of the pipeline implements the camera functionality for an autonomous vehicle using Raspberry Pi 4, camera Niceboy stream, and the ROS framework and is implanted as a stand-alone node. The Camera class is initialized to handle image acquisition from the camera connected to the Raspberry Pi 4 device. It utilizes the OpenCV library for camera access and processing. Captured images are resized to a predefined resolution based on input resolutin of YOLOv3 to decrease the size of the payload sent through the 5G network. The cv_bridge package is utilized to convert the OpenCV image format to ROS image messages, allowing seamless integration with other ROS nodes and display in RVIZ. The message containing the image is sent through the ROS environment to the ROS node called gateway which encodes the image and sends it to the MEC server through the path described in Section 3.1. Note that Gatway does not send every image to the server, but only sends an image when it receives a response to the last image sent. We encode the 3D array values captured by the camera into .jpg format and experiment with various compression ratios. The selection of image resolution and compression ratio is the subject of the experiments in the 8 chapter.

6.5.2. Road signs detection and classification

Apache Kafka receives the message from the core network of the 5G network. This message is encoded in JSON format and contains information about the image and the image itself. This message is then forwarded to a Docker container [30] dedicated to the neural networks. To Docker container is also forwarded the message with the encoded position of the autonomous vehicle in JSON format. Based on a key header in JSON we decide if the message is a position or image. If it is position, we save the actual position of the vehicle for further computation of the road sign position. If it is the image we decode the image from .jpg format to a 3D array with RGB values. The rest of road sign detection and classification is almost the same, like in Sections 6.3.2.5 and 6.4.2.4. First, we found the bounding boxes of road signs by employing YOLOv3, after this we crop the image to get only the images of road signs. Small images of road signs are then put to costume make neural network to obtain road sign class. We need to enlarge the original bounding box to 1.4 times the original resolution, for visual approximation to the GTSRB.

6.5.3. Estimating the road sign position

Measuring the position of road signs is key to identifying road signs validity areas, allowing the autonomous vehicle to adapt its behavior and react according to the applicable traffic laws and road conditions. A road sign visible on the camera is not always valid for an autonomous vehicle. We chose a method described in Section 2.3 in which we work with bounding boxes dimensions.

Since the determination of bounding boxes proved to be sufficiently accurate in most cases. Unfortunately, this method is not accurate, as the experiments performed in the Section 7.5 prove. Unfortunately, for more accurate measurements we would need 2 cameras that work on the principle of the human eye called synchronous stereo cameras, which are more expensive.

We know the a what is the height of the bounding box surrounding the road sign in (14). We chose height because it is not influenced by the rotation of the road sign with respect to the camera. Because the road signs are standardized, we know road sign size in the real world. In our case, these road signs have the size of 12 centimeters in diameter. The only unknown value in (14) is f. But previous parameter is constant for a given camera and it is possible to compute focal length value based on reference measurements with known a, h and d based on the following equation:

$$f = \frac{a}{h \times d} = 1\,380. \tag{43}$$

Only the distance between the camera and road signs is not sufficient to get the road sign position. We also need to obtain the angle between the camera and road signs. To obtain the angle, we use a proportion based on the vertical position of the center of the bounding box of the road sign, as follows:

$$\alpha = \frac{90 \times \left(x_c - \frac{s}{2}\right)}{s}, \tag{44}$$

α is the angle between the camera and the road sign, value s denotes the resolution of the image in pixels and x_c means the position of the bounding box center in pixels on the x-axis, the range of α is from -45° to 45° where 0° corresponds to the center of the image.

Based on computed data and data about vehicle position x, y and yow obtained from the vehicle, we compute the position of the road sign x_m and y_m with respect to the created map by Slam. Equations are as follows:

$$x_m = x + d \times \cos(\mathrm{yow} - \alpha), \tag{45}$$

$$y_m = y + d \times \sin(\mathrm{yow} - \alpha). \tag{46}$$

Computed data about road sign ID and computed position in the map of the road sign is then sent to the vehicle in JSON format through Apache Kafka and the 5G network.

7. Performance evaluation

In this chapter, we introduce the scenario of experiments, we discuss performance metrics for evaluating the neural network and achieved result for both neural network. We also introduce metrics for evaluating the capability of MEC server or local processing in autonomous vehicle. We estimated the accuracy of road sign recognition. We also focused on experiments with MEC server as processing time. During the entire measurement, we are use 5G networks, and we want to avoid causing heavy loads. Therefore, we limit the size of the transmitted data by image resolution and compression. We measured the energy consumption of autonomous vehicle at several values of bitrate, and we also estimate deadlines for recognition of road sign from time image is capture to time autonomous vehicle knows about road sign based on vehicle speed.

7.1. Setup of experiments

In our experiments, we use a Raspberry Pi 4 Model B equipped with a quad-core ARM Cortex-A72 processor running at a core frequency of 1.5 GHz for local computation. The Raspberry Pi has 8GB of RAM. The processor in the Raspberry Pi has a performance of 9.69 GFLOPS [48]. The Intel NUC11PHKi7CAA2 serves as the MEC server, which is equipped with an i7-1165G7 processor, 16GB of RAM, and an Nvidia RTX 2060 graphics card. The processor in the Intel NUC has 4 cores on frequency of 4.7 GHz and achieves 20.4 GFLOPS [49]. The graphics card in the Intel NUC, on the other hand, achieves 52 TFLOPS [50]. For local computations, only the Raspberry Pi's processor is used. In the case of computations on the MEC server, the graphics card also participates and handles the neural networks processing.

7.2. Scenario for experiments

In this section, we look at the scenarios and execution of the experiments. All experiments, including the capture of images for the datasets, is provided using a real model of an autonomous vehicle, described in Section 4, hereafter referred to as an autonomous vehicle or vehicle, and a real MEC server connected to an emulated 5G network. The only change to the autonomous vehicle is the connection of the second antenna to the modem. The lab, where we make experiments, features an emulated network using OAI, with the gNB located in the ceiling above the lab. The autonomous vehicle moves in an area no more than 7 m away from the gNB. The bitrate measurements in the uplink and downlink directions are obtained with minimum values of 5 Mb/s and 40 Mb/s, when the vehicle is moving. To ensure the stability and reproducibility of the experiment, we limited the bitrate values on the autonomous vehicle to 4 Mb/s in uplink and 40 Mb/s in downlink.

Next, we need to determine the speed of the vehicle to perform the experiments. The average speed of a car in the city is 30-40 km/h [51], when we convert this speed to our model, which is in 1:10 scale, we get a speed of 3-4 km/h which corresponds to 1 m/s. For this reason, we conducted the experiments at vehicle speed 1 m/s. Higher speeds, which reflect driving outside the city or on the highway, are not possible due to the design of the model and the layout of the laboratory in which the experiments are performed.

To ensure the regularity of the experiments, the images of the road signs are taken before the experiments and then processed on the vehicle or sent to the MEC server for processing. They are processed in the order in which they are taken, because in the experiments we are not interested in the exact timing of the images, but rather in evaluating what the road signs are in the images, possibly determining the processing time on the Raspberry Pi or on the MEC server, along with the

energy consumed. Due to the lack of a camera, it is necessary to take images when the vehicle is not moving. In the case of a moving vehicle, the captured images are often too blurry for the neural networks to accurately detect and classify road signs. This issue is not attributed to the limitations of the neural networks or the chosen solution but rather to the inadequacy of the camera system. Consequently, the camera's limitations significantly impact our results in case of moving vehicle. In total, two datasets are taken, for a stationary vehicle. One to determine the distances of each road sign between 30 and 600 cm, with values increasing from 60 cm by 60 cm. For each distance, 27 photos of different road signs were taken, so that all 7 classified road signs are equally represented in the whole dataset. The second dataset contains 7 road signs with 27 images of each. The road signs are positioned 60-180 cm in front of the vehicle, so that road signs are evenly distributed in the area captured by the vehicle's camera.

The distance dataset is evaluated, both on the Raspberry Pi on the vehicle and on the MEC server, to determine the accuracy of road sign recognition at each distance. The same dataset is used to evaluate the accuracy of the distance determination.

The road sign classification dataset is then compressed using several JPG quality parameter values in the openCV library. Based on this, the average size of the image at a given compression ratio is then determined, as well as the average compression ratio for given JPG quality. Based on the compression size, we then search for the most appropriate file size to transfer to the MEC server by creating a classification accuracy for each JPG quality value. Subsequently, confusion matrices are created to discuss the effect of compression. All evaluations are made locally on the vehicle on the Raspberry Pi and on the MEC server. After finding a suitable compression size, only images with suitable compression are further processed for the rest of the experiments. In these experiments, the autonomous vehicle is not driven because we are not interested in the processing time of the image, but rather what is in the image, which does not affect the network and changing networks parameters while driving.

Next, we evaluate the energy consumed to process one image. We measure only the energy consumed by vehicle in both cases, local image processing in vehicle and image processing on the MEC server. In these experiments, the vehicle moves at a constant speed of 1 m/s. To investigate the effect of the network on the consumed energy, the bitrate of the vehicle is limited to several values and the measurement of the consumed energy is also performed during local processing. The measurement of the consumed energy is done by using special USB meters [52] that measure the consumed energy from the power bank that is used to power the Raspberry Pi and the 5G modem.

We then process the collected data and determine the deadline threshold for road sign recognition to avoid missing a road sign without reacting to that road sign. We perform this evaluation based on the stop distance of the real car, converted to the size of our vehicle.

7.3. Performance metrics

In this section, we first define metrics for detection and classification tasks. Next, in the metrics, we focus on the accuracy of determining the distance of a road sign from an autonomous vehicle. Next, metrics that tell us about the usability of computation at the network edge, such as power consumption and deadlines or process times.

7.3.1. Detection

In this subsection, we introduce a detailed exploration of testing methodologies specifically created for evaluating the performance of detection models. Detection is the process of searching objects in images.

7.3.1.1. Mean Average Precision

The mAP [53] is a widely used metric for evaluating object detection algorithms. It is calculated by first determining the Average Precision (AP) for each class, which is the area under the precision-recall curve (PR curve) for that class. Precision and recall are defined as:

$$P = \frac{TP}{TP + FP}, \tag{47}$$

$$R = \frac{TP}{TP + FN}, \tag{48}$$

where TP is true positive, denotes the number of correctly predicted positive instances, FP is false positives means the number of incorrectly predicted positive instances and FN is false negative denotes the number of positive instances that were not correctly identified. The mAP is then computed as the mean of the AP values across all classes:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i, \tag{49}$$

where N is the total number of classes and AP_i is the Average Precision for a specific class.

7.3.1.2. Intersection over union

The second metric considered in the thesis is IoU [53]. His metric measures the overlap between two bounding boxes and is calculated as the ratio of the intersection area to the union area:

$$IOU = \frac{Area\ of\ Intersection}{Area\ of\ Union}. \tag{50}$$

The IoU values range from 0 to 1, where 1 indicates full overlap and 0 indicates no overlap. The IoU is often used as a criterion for evaluating the accuracy of object detection, where a suitably chosen IOU threshold influences the number of success detection between the ground true bounding box and predicted bounding box. Thus, as the IoU threshold increases, the mAP value tends to decrease.

7.3.2. Classification

In this subsection, we delve into the realm of classification testing methodologies, essential for evaluating the efficiency and performance of classification models. Classification is the process of assigning the class to examined image.

7.3.2.1. Classification Accuracy

Classification accuracy CA [54] is mathematically described as follow:

$$CA = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}, \quad (51)$$

Top-1 classification accuracy identifies whether the model correctly predicts the most probable class for a given input, while top-5 classification accuracy considers whether the correct class is among the top five predictions. However, since our focus is on accurately classifying road signs, we only consider top-1 accuracy in the whole thesis.

7.3.2.2. Confusion Matrix

The confusion matrix [53], [55] provides a detailed overview of a classification model's performance across various classes. Confusion matrix compares the model's predictions with the actual labels in the test dataset, highlighting correct predictions along the diagonal and misclassifications off-diagonal in matrix. By analyzing confusion matrix, we can pinpoint the model's strengths and weaknesses points. The confusion matrix shows that, for example, the 80-speed limit sign is often misclassified as a no vehicles sign. By leveraging this information, we can further adapt the neural network to enhance neural network classification accuracy.

7.3.3. Distance estimation error

The distance, in centimeters (cm), is the distance between the vehicle camera and the road sign using Euclidean distance. The autonomous vehicle calculates distance based on the image data taken by the camera and the bounding boxes found, see more in Section 6.5.3. Our goal is to analyze and determine the accuracy of this calculation by comparing the actual distance with the distance calculated by the vehicle. The distance estimation error is defined as the difference between the actual distance of road sign and the calculated distance of road sign.

7.3.4. Energy consumption

The energy consumption is measured in watt-hours (mWh) and provides an overview of the amount of energy consumed during the processing of the camera data. In thesis, we are not directly interested in energy consumption as a function of time, but we are primarily concerned with the energy consumption to evaluate a single image, and thus consumed energy is not affected by the runtime of different algorithms at different points in the network, such as when offloading data to an MEC server or local processing. Our primary focus is on saving energy on the vehicle side rather than on the MEC server side, as the server is connected to the electrical network. Specifically, for the local measurements, we only consider the energy consumed by the Raspberry Pi and add together the energy consumed by the Raspberry Pi and the 5G modem when processing data on the MEC server.

7.3.5. Maximum distance of successful recognition

The maximum distance of successful recognition refers to the longest distance, in centimeters (cm), over which neural networks accurately recognize and classify a road sign, with a certain level of confidence or classification accuracy. The measurements help in evaluating the system's ability to detect and classify road signs from different distances and provide insights into the effectiveness of road sign recognition.

7.3.6. File size

The file size metric indicates the amount of digital storage space required to store a single image after processing or compressing them. This metric is measured in kilobytes (kB). In the context of our experiments, file size is a key metric because it directly affects the amount of data transferred over the network. Decreasing the file size usually leads to a reduction in the transmission time. By analyzing the file size, we can evaluate the tradeoffs between image quality and road sign recognition efficiency, which leads to reduced network traffic load.

7.3.7. Deadlines

Deadline in the context of autonomous vehicles is the critical time, in milliseconds (ms), for a vehicle to recognize and start to react to a road sign. Vehicles have a certain stop distance, and the road sign should be safely recognized before the vehicle is closer to the road sign than the vehicle's stop distance. Thus, as seen the deadline depends on the speed of the vehicle.

7.3.8. Processing time

Processing time, in milliseconds (ms), encompasses the duration needed for road sign recognition on MEC server or locally in autonomous vehicle. In local processing, processing time reflects the autonomous vehicle's processing duration, calculated from image capture to end of calculation of the recognition result. In MEC server processing, processing time reflects the time between receiving an image to recognize from vehicle and sending back to the vehicle a message about a detected road sign. Communication time on MEC server is time in ms and means the time between the vehicle sending the image to MEC server and vehicle receive the message with information about road sign. Latency, in this scenario, signifies the time data requires to traverse the network distance between the autonomous vehicle and the MEC server with the image, and vice versa, between the MEC server and the autonomous vehicle with the recognition message. The integration of latency into MEC processing time is pivotal for precise assessment and scheduling of performance within MEC systems.

7.3.9. Bitrate

Bitrate, expressed in kilobits per second (kb/s), is a key parameter in mobile networks because it determines the speed at which data is transferred over the network. A higher bitrate means faster data transfer, which is important for real-time applications. Bitrate is also divided into uplink, the speed of uploading data to the MEC server from the vehicle, and downlink, which is the speed of downloading data from the MEC server to the autonomous vehicle.

7.3.10. Image resolution

Image resolution refers to the amount of detail an image holds, measured in pixels (px). Higher resolution means more pixels, resulting in greater image detail and clarity, which is crucial for tasks processing image in neural networks. Conversely, lower resolution reduces image detail, making the image appear more pixelated or blurry.

7.3.11. JPG quality

JPEG quality refers to the compression level applied to a JPEG image, which affects both the file size and image quality. Higher JPEG quality results in less compression, maintaining more detail and clarity but producing larger file sizes. Lower JPEG quality increases compression, reducing file size at the cost of image detail, potentially introducing visible artifacts, and reducing overall image quality.

7.4. Achieve mAP of YOLOv3 results

The presentation of YOLOv3 results is here, as YOLOv3 forms a significant part of the road sign recognition in the in our image processing pipeline. Without proper road sign detection, it is not possible to correctly determine the type of road sign or its distance from the camera. For road sign classification, it is important to find road sign's bounding box correctly. Figure 21 presents mAP calculated by different IoU between predicted bounding box and ground true bounding boxes on own validation images. The YOLOv3 is train on own dataset described in Section 5.2, This is taken in consultation with the thesis supervisor.

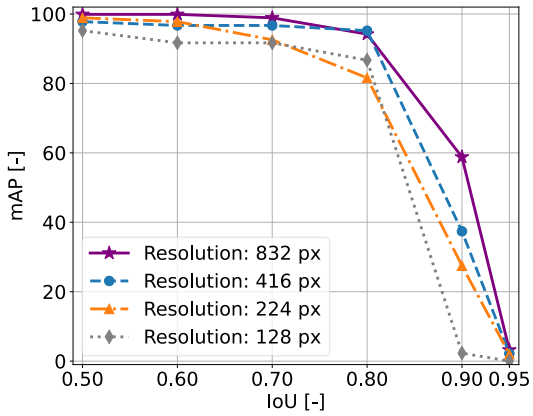

Figure 21: Dependency of mAP on IoU between ground true bounding box and predicted bounding box

7.5. Classification accuracy based on distance and image resolution

Now, we investigate how distance between the camera and road sign affect classification accuracy. The results of this experiment are depicted in Figure 22. For each distance, results are based on 27 captured and recognized images.

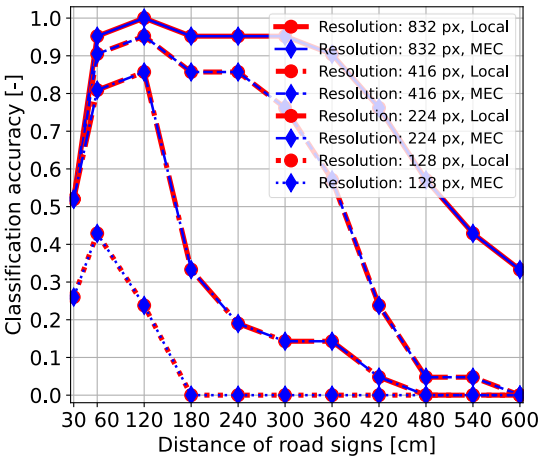

Figure 22: Impact of distance of the vehicle from the road sign and of image resolution on classification accuracy.

Since the MEC server and the Raspberry Pi run the same neural network pipeline with the same weights, and the evaluated images are identical to maintain consistent conditions, the classification accuracy remains the same, as depicted in Figure 22. In Figure 22 Local mean the processing on Raspberry Pi in vehicle and MEC means processing on MEC server.

Figure 22 reveals that the images with a resolution of 832x832 pixels reach the highest accuracy (above 0.9 up to a distance of 360 cm). The classification accuracy in general decreases with the resolution, as expected. The classification accuracy of at least 0.9 is reached also for the resolution 416x416 pixels, but only for a lower distance (up to 120 cm). The impact of the distance on classification accuracy for all resolutions is non-monotonic. When the distance is 30 or 60 cm, the road signa small too close to the vehicle's camera and the neural networks are not able to recognize the road sign accurately, since such a small distance is not included in training. The highest accuracy is generally achieved when the road sign is between 60 and 180 cm from the camera.

7.6. Classification accuracy based on file size

Since transferring a high resolution images can impose a load on mobile network and can prolong the overall image processing, we investigate also possibility to reduce resolution of the image, but this has not proved robust enough as demonstrated in Figure 22. Alternative option to reduce volume of data for offloading is to use image compression. We used the JPEG quality compression from openCV2 for compression original images. It is not feasible to determine the file size of a new file directly from the JPEG quality, because there is stochastic equation for compression. We compressed 27 images at different JPEG quality levels using the OpenCV2 library, offering valuable insights for this purpose. We also calculate the ratio to determine the ratio of compressed image resolution to the original image resolution, see Table 4. The dependence of the size of an image of 832x832 pixels on the JPEQ Quality is shown in Figure 23 and Figure 24. Note that we focus on the images with resolution of 832x832 pixels, because based on the results from Figure 22, lower resolutions lead to a significant degradation in classification accuracy and further compression would not allow to recognize road signs accurately enough. The images from the Figure 22 are not compressed and images have an average file size 66 kB. Classification accuracy in Figure 24 is based on the 27 images for every of 7 road signs.

Table 4: JPEG Quality setting to file size

JPEG Quality setting [-]	Approximate Compression Ratio [-]
90	0.90
80	0.80
70	0.70
60	0.60
50	0.40
40	0.18
30	0.15
20	0.13
10	0.10

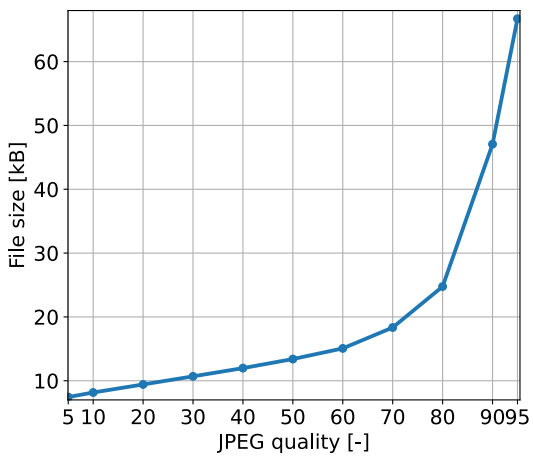

Figure 23: JPEG quality vs. file size

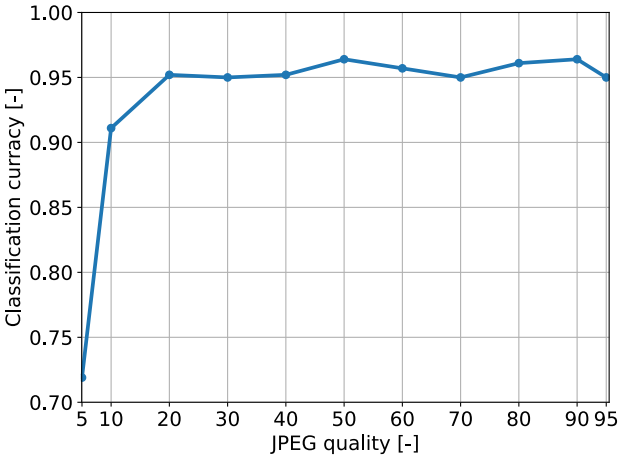

Figure 24: Dependency of correct prediction probability on JPEG image compression

From experiment with file size, we see that the compression has almost no effect on the classification accuracy up to JPG quality 20, where the classification accuracy is at least 0.95. Subsequently, for compressions with JPG quality 20 and lower, the achieved classification accuracy drops significantly down to a value of 0.72, see in Figure 24. It can also be noticed that the largest reduction in file size takes place between JPG quality values of 95 and 60, see Figure 23. Between JPG quality 20 to 5, the file size does not decrease as much, but the classification accuracy decreases significantly.

Subsequently, we create 27 testing images for each class of classified road sign, and based on how accurately they are classified, we generate a confusion matrix for image sizes of 832x832 pixels and compression ratios of 95, 20, and 5. Results are in Table 5, Table 6

and Table 7 for the respective compression ratios. The generated confusion matrixes are the same for the evaluating on the MEC server as well as on the vehicle. Tables compare the impact of the compression ratio on recognition of the road signs. Based on experiments with file size, we concluded to transmit on MEC server images with JPG quality 30 for data processing on the server, because the achieve a classification accuracy 0.95 and the file size is significantly smaller compared to the JPG quality 95. Reduction in file size is 54 kB.

Analysis of Table 5 revealed that the most common errors in road sign recognition are due to misclassification as not classified or not found. That is, the sign is not classified or not found at all. No vehicle and priority road signs are confused with each other in 2 cases and stop and no vehicle road signs are confused with each other in 1 case. The end of speed limit sign shows a higher rate of misclassification, particularly as not found, indicating problems with its visibility in the chosen environment.

Analysis of the Table 6 revealed similar trends to the Table 5, with a few differences. For the end of speed limit sign, there are repeated classification errors, mostly as not found. An improvement is seen for the 50 km/h road sign, which shows only one error compared to the Table 6. The no vehicle road sign shows the same level of accuracy in both matrices. However, the stop marker has improved and now shows 100% classification accuracy. And classification accuracy is lower in case of priority road.

Analysis of Table 7 revealed several differences and similarities compared to Table 5. The 20 km/h, 50 km/h and 80 km/h road signs showed a reduction in the number of correctly recognized signs, with a larger proportion of errors are classified as not classified and not found. The expected result was that the road signs 20 km/h, 50 km/h and 80 km/h should be misclassified among themselves, which did not occur. The end of speed limit sign showed a similar trend to Table 5, with a higher number of errors classified as not found. The stop and priority road signs show similar errors to the Table 5, suggesting that the road signs are unique enough in shape to be correctly classified, even if they are not completely sharp. Overall, road sign classification errors are largest in Table 7, while the errors were comparable for the Table 5 and Table 6.

Table 5: Confusion matrix for resolutions 832x832 and JEPG Quality of 95, where each class consists of a total of 27 images

20 km/h	26	0	0	0	0	0	0	1	0
50 km/h	0	25	0	0	0	0	0	2	0
80 km/h	0	0	25	0	0	0	0	1	1
end of speed limit	0	0	0	23	0	0	0	1	3
no vehicle	0	0	0	0	25	2	0	0	0
priority road	0	0	0	0	0	27	0	0	0
stop	0	0	0	0	1	0	26	0	0
	20 km/h	50 km/h	80 km/h	end of speed limit	no vehicle	priority road	stop	not classified	not found

Table 6: Confusion matrix for resolution 832x832 and JEPG Quality of 20, where each class consists of a total of 27 images

20 km/h	27	0	0	0	0	0	0	0	0
50 km/h	0	26	0	0	0	0	0	1	0
80 km/h	0	0	25	0	0	0	0	1	1
end of speed limit	0	0	0	21	0	1	0	1	4
no vehicle	0	0	0	0	25	2	0	0	0
priority road	0	0	0	0	1	26	0	0	0
stop	0	0	0	0	0	0	27	0	0
	20 km/h	50 km/h	80 km/h	end of speed limit	no vehicle	priority road	stop	not classified	not found

Table 7: Confusion matrix for resolution 832x832 and compression ratio of 5, where each class consists of a total of 27 images

20 km/h	16	0	1	0	0	0	1	2	7
50 km/h	0	15	0	1	0	0	0	5	6
80 km/h	0	0	20	0	0	0	1	3	3
end of speed limit	0	0	0	13	0	1	0	6	7
no vehicle	0	0	0	0	22	2	0	2	1
priority road	0	1	0	0	0	21	0	1	4
stop	0	0	0	1	1	0	25	0	0
	20 km/h	50 km/h	80 km/h	end of speed limit	no vehicle	priority road	stop	not classified	not found

7.7. Time of processing the image

In Figure 25, we see the average processing time of one image depending on the resolution of images. The processing is done using a pipeline of neural network for road sign detection and classification, both on the MEC server and locally on the Raspberry Pi in the vehicle. This measurement is evaluated as the average classification time of 1 image when 27 images are recognized.

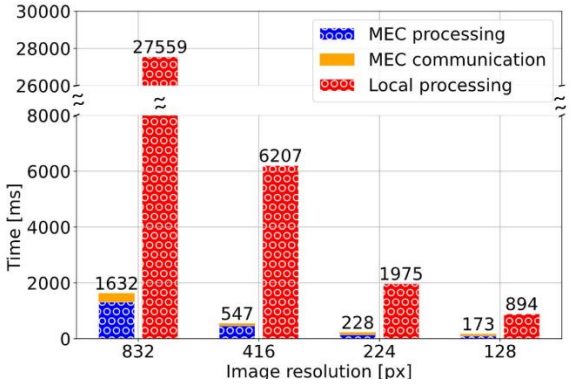

Figure 25: Dependency of processing time on image resolution

Figure 25 also illustrates the time between the start of image transmission by vehicle and subsequent reception of message with recognized road sign by vehicle (orange bar) with the JPEG quality of 30.

7.8. Energy consumption

We evaluate the energy required to process camera data, whether locally on the vehicle by Raspberry Pi or offloaded to the MEC server. The average energy consumption for processing one images is shown in Figure 26. For the second case, we considered both the energy consumption of the Raspberry Pi and the 5G modem. However, in the case of local processing, we only consider the energy consumed by the Raspberry Pi, as communication is not necessary for local computation. We try several bitrates for transfer the image to MEC server and to transfer message with information about road sign back to vehicle. We set the same maximum bitrates for uplink and downlink. The examined values of bitrate are 0, 25, 50, 100, 200, 300 kb/s. 0 kb/s refers to the fact that the image is not transferred to the MEC server, and the image is processed locally in the vehicle. As shown in Figure 26, the energy consumption during offloading is significantly affected by bitrates. The lower the bitrate is the more power is consumed on the vehicle as it takes longer to transfer images to the MEC server. The average energy consumption by processing one image decreases 15 times when we compare the local processing in vehicle to transfer the image to MEC server with bitrate 300 kb/s. Even if we compare the local processing with the slowest bitrate of 25 kb/s we decrease the energy consumption by 2-times. When we compare the fastest bitrate 300 kb/s and slowest bitrate 25 kb/s we see that decrease the energy consumption by 7-times. It is worth noting that the energy reduction between bitrates of 200 kb/s and 300 kb/s is not significant because the computation time on the server outweighs the time saved in image and message transmission. This experiment is based on processing 27 images.

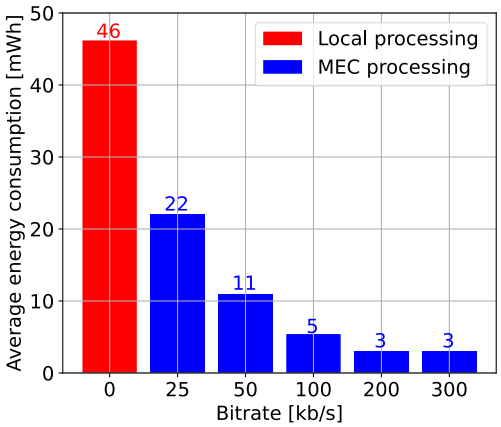

Figure 26: Bitrates vs. average energy consumption of processing one image

7.9. Error in distance estimation

In this experiment, we look at the accuracy of determining the distance of the road sign from the camera. Distance from a road sign is an important metric because we can use distance to determine position in space as shown in Section 6.5.3. This determination of the position in space, and thus the distance, allows us to define the boundary of the validity of the road sign, from which point in space the road sign is valid. Moreover, as seen from Figure 22 and Figure 27 the determination of the distance of a road sign proved to be more accurate than its classification and, for this reason, it is possible to filter out signs that are too far away to be reliably classified. The accuracy of this distance determination is shown in Figure 27. The error in the distance estimation typically ranges around 10 cm in absolute values, with a maximum of 55 cm, while the average error of 14.32 cm. For the chosen distance measurement method, this result is accurate above expectations. The determination of the distance with an average error of 14.32 is accurate enough for our scale. For a maximum successful detection distance of 360 cm, the relative error is 4%. Error in distance evaluation is based on the 27 images for every real distance. This dataset is same as in Classification accuracy based on distance and image resolution in Section 7.5. Data computed on vehicle and on MEC server are same, therefore we plot only one of them for better clarity.

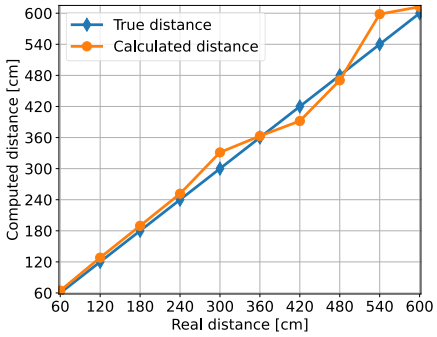

Figure 27: Dependency of calculated distance on real distance

7.10. Time to react to road sign

In this section we focus on evaluating the results from the previous sections, but first we need to determine the stop distance of the vehicle. This provides the vehicle with ample distance to respond to the road sign, enabling it to stop precisely at the road sign's location. The average vehicle speed in cities is around 30-40 km/h [51]. Considering the size of our model to the real car that is 1:10. We interpret this to average speed 3-4 km/s, which corresponds to 0.83-1.11 m/s. For this reason, we decided to investigate situations where the autonomous vehicle travels 0.5, 1 and 1.5 m/s, corresponding to average speeds in cities and speeds slightly higher and lower. A table of stop distances converted to the size of our vehicle is in Table 8. In our case, the stop distances are smaller, but we use the braking distances of real cars for reference.

Table 8: Stop distances at a given vehicle speed [56], [57]

Vehicle speed [m/s]	Stop distance [cm]
0.5	40
1	80
1.5	140

Maximum distance of successful recognition is in Table 9 is based on results from Figure 22. Confidence or classification accuracy for maximum distance of successful recognition is set to 90% because this percentage provides a sufficient level of safety and reliability for the autonomous vehicle moving in the lab. The higher result was achieved with a resolution of 832 pixels, while the worst were images with a resolution of 224 and 128 pixels, as they did not surpass the required threshold in classification accuracy of 90%.

Table 9: JPEG image resolution to maximum distance of successful recognition

Image resolution [px]	Maximum distance of successful recognition [cm]
832	360
416	120
224	NaN
128	NaN

We determine the maximum time t_m for image processing image, before it is too late to react to road sing as follows:

$$t_m = \frac{d_m - d_s}{v}, \tag{52}$$

where d_s denotes stopping distance, d_m means maximum distance of successful recognition, $d_m - d_s$ denotes the maximal distance, that vehicle can travel before vehicle must have information about road sign. Time t_m is also call deadline. Results are in Figure 28 and t_m is show as text above the computed points. Only results for image resolution 832 and 416 px are shown, because for lower image resolution we do not recognize the road sign accurately enough, see Table 9, as well as the result for image resolution 416 px and vehicle speed 1.5 m/s are unfeasible because stop distance is bigger than maximum distance of successful recognition.

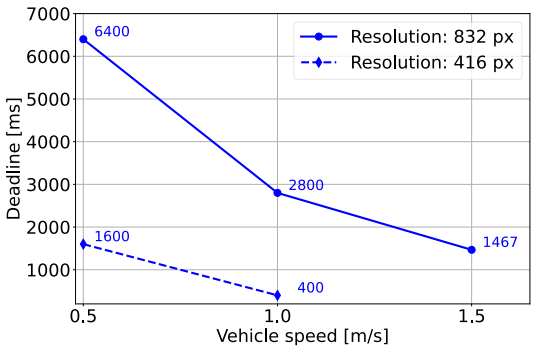

Figure 28: Vehicle speed vs. deadline vs. image resolution

Base on Figure 25 we determined the feasible vehicle speed and images resolution. Processing time on autonomous vehicle is 27559 ms for image with resolution of 832 px and 6207 ms for image with resolution 416 px, base on this fact local processing is not feasible for this vehicle speed and image resolution. Consider an image resolution of 416 px and the fact that a vehicle at very low speeds has a stop distance of 10 cm [56], so an autonomous vehicle would have to travel at 0.18 m/s to be able to recognize a road sign. Similarly, for 832 px the vehicle speed is 0.01 m/s.

The complete time it takes to recognize road sign on the MEC server for image with a resolution of 832 px is 1632 ms, or for images with a resolution of 416 px is this time 547 ms. Considering comunication time in MEC server is possible to classify the images with resolution 832 px and vehicle speed 0.5 and 1 m/s, and the images with resolution 416 px only for vehicle speed 0.5 m/s. Unfortunately, the vehicle speed of 1.5 m/s is too high to react reliably to road signs in our setup.

Based on our results from this chapter, images with a resolution of 832 px and a JPG quality of 30 appear to be the most suitable for transfer to the MEC server. These images achieve a high level of classification accuracy, see Figure 24, have the largest maximum distance of successful recognition at set classification accuracy 0.9, see Figure 22, and there is a time safety margin [58] when processing images at MEC server at vehicle speeds of 0.5 and 1 m/s.

7.11. Demonstration of the road sign recognition

The demonstration[4] of the thesis result showing the entire autonomous vehicle system in action. The autonomous system is activated, including path planning, and driving along the computed path. The algorithm in the MEC server dynamically decides which tasks to compute locally and which to offload on MEC server, choosing to perform image processing on the MEC server for whole video. We instruct the vehicle to drive to a specific point on the other side of the lab. As the

[4] https://www.youtube.com/watch?v=aPKcAR9Qli4

vehicle moves, its camera captures the surroundings, and upon detecting a road sign, a corresponding message is displayed on the terminal, as shown in Figure 29. In Figure 30, it is possible to see the autonomous vehicle.

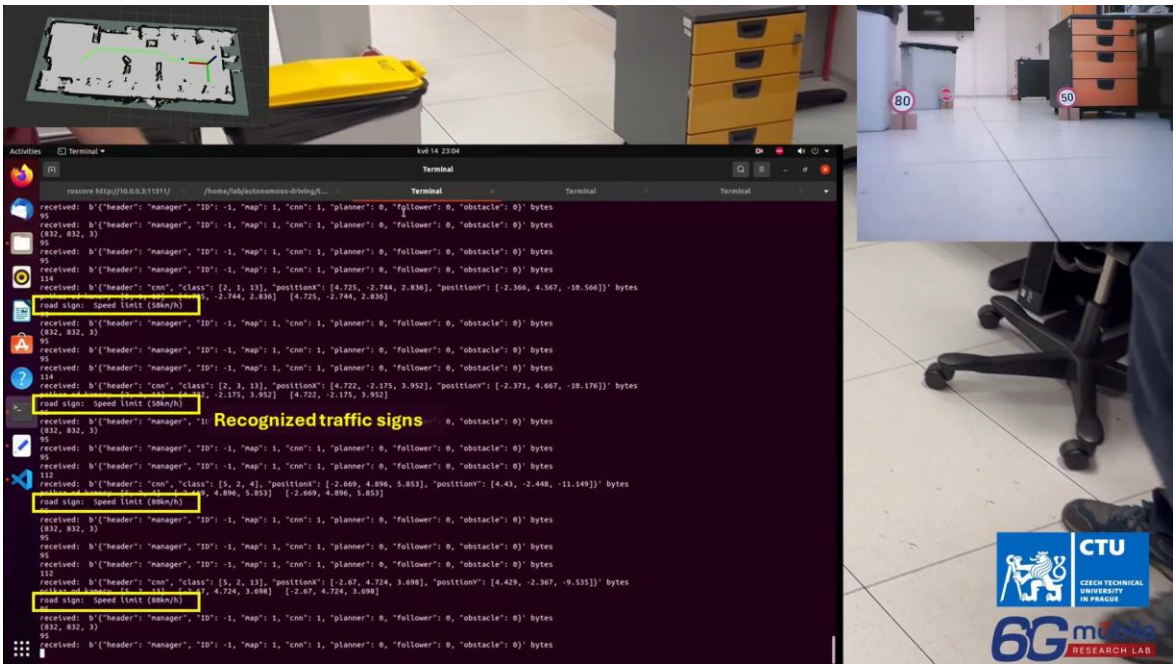

Figure 29: Demonstration of road sign recognition I.

Figure 30: Demonstration of road sign recognition II.

8. Conclusion

The diploma thesis has addressed the road sign detection and classification in the context of autonomous driving systems. In this thesis, we have developed functionalities enabling to offload the road sing recognition at the edge of the mobile network represented by MEC server

For performance evaluation, a custom dataset covering seven road signs is developed. YOLOv3, chosen for detection due to its 40 FPS and mAP of 78.6%, running locally in the vehicle equipped with limited computing resources (Rapsberry Pi 4) can not handle the road sign detection and classification in required time, as the autonomous vehicle model would have to travel at a maximum speed of 0.08 m/s to obtain the information about the road sign in the sufficient time in advance to be able to react to the road sign. The processing time of YOLOv3 on the vehicle with Raspberry Pi is 26 seconds for 832 px images. However, offloading to MEC reduces processing time to 1.7 seconds, allowing the vehicle to travel at 1 m/s, while the autonomous vehicle has time to react to a recognized road sign. In the case of the local processing, no road sign is recognized early enough for the model of the vehicle moving with the speed of 1 m/s. In the same scenario, MEC processing achieves up to 95% classification accuracy. Additionally, the experiments show that the energy spent for offloading is up to seven-times lower than if the processing takes place locally in the vehicle, depending on network bitrate.

The developed concept for autonomous driving with road sign recognition is presented in the form of video, where autonomous vehicle drives in laboratory and detects and classifies the road signs.

As future work, the dataset should be expanded to include more road signs. Another extension is the implementation of the acquired information on the road sign into control algorithms enabling autonomous vehicles to react to the road signs. Furthermore, testing an advanced camera providing more clear images of the road signs even at high speed and at wider range of distances of the vehicle to the road sign together should be tested in order to improve accuracy of the road sign detection and classification. Investigation of possible energy saving and lower road sign detection and recognition time via multi-exit architecture and split of the neural networks should be explored as well. Further extension should consist in management of communication and computing resources for scenario with multiple vehicles and MEC servers.

Bibliography

[1] E. Yurtsever, J. Lambert, A. Carballo and K. Takead, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," *IEEE Access,* vol. 8, pp. 58443-58469, 2020.

[2] T. Litman, "Autonomous Vehicle Implementation Predictions: Implications for Transport Planning," 2020.

[3] K. Muhammad, A. Ullah, J. Lloret, J. D. Ser and V. H. C. de Albuquerque, "Deep Learning for Safe Autonomous Driving: Current Challenges and Future Directions," *IEEE Transactions on Intelligent Transportation Systems,* vol. 22, pp. 4316-4336, 2021.

[4] Z. Bečvář and P. Mach, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys & Tutorials,* vol. 19, no. 3, 2017.

[5] L. Qian, Z. Luo, Y. Du and L. Guo, "Cloud computing: An overview," *In Cloud Computing: First International Conference,* pp. 626-631, 2009.

[6] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo and J. Zhang, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," *Proceedings of the IEEE,* vol. 107, pp. 1738-1762, 2019.

[7] T.-D. Do, M.-T. Duong, Q.-V. Dang and M.-H. Le, "Real-Time Self-Driving Car Navigation Using Deep Neural Network," *2018 4th International Conference on Green Technology and Sustainable Development (GTSD),* pp. 7-12, 2018.

[8] M. Babiker, M. A. O. Elawad and A. H. Ahmed, "Convolutional Neural Network for a Self-Driving Car in a Virtual Environment," *2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE),* pp. 1-6, 2019.

[9] Á. Arcos-García, J. Alvarez-Garcia and L. Soria Morillo, "Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods," *Neural Networks,* vol. 99, 2018.

[10] T. Verschoor, V. Charpentier, N. Slamnik-Kriještorac and J. Marquez-Barja, "The testing framework for Vehicular Edge Computing and Communications on the Smart Highway," *IEEE Consumer Communications & Networking Conference (CCNC),,* 2023.

[11] "open-datasets/open-image-v6," V7Labs, [Online]. Available: https://www.v7labs.com/open-datasets/open-image-v6. [Accessed 29 03 2024].

[12] J. Stallkamp, M. Schlipsing, J. Salmen and C. Igel, "The German Traffic Sign Recognition Benchmark: A multi-class classification competition," *IEEE International Joint Conference on Neural Networks (IJCNN) ,* pp. 1453-1460, 2012.

[13] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing and C. Igel, "gtsrd," Institut für neuroinformatik RUB, 05 11 2013. [Online]. Available: https://benchmark.ini.rub.de/gtsdb_news.html. [Accessed 29 03 2024].

[14] Z.-Q. Zhao, P. Zheng, S.-t. Xu and X. Wu, "Object Detection With Deep Learning: A Review," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 30, no. 11, pp. 3212-3232, 2019.

[15] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 21 09 2018. [Online]. Available: https://arxiv.org/abs/1804.02767. [Accessed 29 03 2024].

[16] L. Yang, G. Chen and W. Ci, "Multiclass objects detection algorithm using DarkNet-53 and DenseNet for intelligent vehicles," *EURASIP J. Adv. Signal Process,* no. 85, 2023.

[17] S. Nazmul and A. Maida, "Enhancing ResNet Image Classification Performance by using Parameterized Hypercomplex Multiplication," 2023.

[18] M. A. Wani, F. A. Bhat, S. Afzal and A. I. Khan, "Basics of Supervised Deep Learning," *Advances in Deep Learning,* pp. 13-29, 2020.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 1-9, 2015.

[20] A. Gupta and R. K. Jha, "A Survey of 5G Network: Architecture and Emerging Technologies," *IEEE Access,* vol. 3, pp. 1206-1232, 2015.

[21] 3GPP, "5G Mobile System Architecture, TS 23.501, version 17.5.0," *3GPP Technical Specification,* 2022.

[22] J. Sachs, G. Wikstrom, T. Dudda, R. Baldemair and K. Kittichokechai, "5G Radio Network Design for Ultra-Reliable Low-Latency Communication," *IEEE Network,* vol. 2, pp. 24-31, 2018.

[23] H. Saqib, R. G. Thippa , P. K. R. Maddikunta, S. P. Ramu, P. M, C. D. Alwis and M. Liyanage, "Autonomous vehicles in 5G and beyond: A survey," *Vehicular Communications,* vol. 39, 2023.

[24] P. K. Agyapong, M. Iwamura, D. Staehle, W. Kiess and A. Benjebbour, "Design considerations for a 5G network architecture," *IEEE Communications Magazine,* vol. 52, pp. 65-75, 2014.

[25] Y. Gao, C. Sun, X. Zhang and X. Hong, "Study on MCS Selection and Spectrum Allocation for URLLC Traffic under Delay and Reliability Constraint in 5G Network," *Entropy,* vol. 24, 2021.

[26] Y. Zhang, W. Wang, J. Ren, J. Huang, S. He and Y. Zhang, "Efficient Revenue-Based MEC Server Deployment and Management in Mobile Edge-Cloud Computing," *IEEE/ACM Transactions on Networking,* vol. 31, pp. 1449-1462, 2023.

[27] S. Basodi, C. Ji, H. Zhang and Y. Pan, "Gradient amplification: An efficient way to train deep neural networks," *Big Data Mining and Analytics,* vol. 3, pp. 196-207, 2020.

[28] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," 2018.

[29] S. Persson, "/yolov3-implementation-with-training-setup-from-scratch-30ecb9751cb0," Medium, 21 03 2021. [Online]. Available: https://sannaperzon.medium.com/yolov3-implementation-with-training-setup-from-scratch-30ecb9751cb0. [Accessed 29 03 2024].

[30] Docker Inc., Docker Inc., 2024. [Online]. Available: https://docs.docker.com. [Accessed 24 05 2024].

[31] OpenAirInterface.org, OpenAirInterface.org, 2024. [Online]. Available: https://openairinterface.org. [Accessed 02 05 2024].

[32] T.-Y. Lin, . M. Maire, S. Belongie, L. D. Bourdev, R. B. Girshick, H. James , . P. Pietro, D. Ramanan and . Z. C. Lawrence, "Microsoft COCO: Common Objects in Context," *Conference on computer vision,* pp. 740-755, 2014.

[33] C. Shorten and T. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Big Data,* vol. 6, 2019.

[34] Traxxas, "/sites/default/files/88096-4-OM-EN-R01.pdf," Traxxas, [Online]. Available: https://traxxas.com/sites/default/files/88096-4-OM-EN-R01.pdf. [Accessed 29 03 2024].

[35] "https://www.best-power.cz," BONA SPES s.r.o.,, 2024. [Online]. Available: https://drive.google.com/file/d/19plxAEmIeRSCj8AFL9aGLZHDIoW9UjbC/view?pli=1. [Accessed 12 03 2024].

[36] "https://datasheets.raspberrypi.com," Raspberry Pi Ltd., 03 2024. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf. [Accessed 03 12 2024].

[37] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah and E. Gunawan, "A Systematic Literature Review of A* Pathfinding," *Procedia Computer Science,* vol. 179, pp. 507-514, 2021.

[38] C. R. Conlter, "pub_files," 1 1992. [Online]. Available: https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf. [Accessed 27 4 2024].

[39] Shangai Slamtec Co., Ltd., "RPLIDAR A1," 15 10 2020. [Online]. Available: https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf. [Accessed 13 12 2023].

[40] "https://pdf1.alldatasheet.com," Silicon Labs, 03 2007. [Online]. Available: https://pdf1.alldatasheet.com/datasheet-pdf/view/201067/SILABS/CP2102.html. [Accessed 12 03 2024].

[41] Open Robotics, "ros.org," Open Robotics, [Online]. Available: https://www.ros.org. [Accessed 19 12 2023].

[42] deyouslamtec, "github," Slam Tec, 20 11 2023. [Online]. Available: https://github.com/Slamtec/rplidar_ros. [Accessed 22 12 2023].

[43] Roboflow, Inc., Roboflow, Inc., 2024. [Online]. Available: https://roboflow.com. [Accessed 02 04 2024].

[44] J. Stallkamp, M. Schlipsing, J. Salmen and C. Igel, "/gtsrb," Insitut für neuroinfromatic RUB, 10 05 2019. [Online]. Available: https://benchmark.ini.rub.de/gtsrb_news.html. [Accessed 29 03 2024].

[45] aladdinpersson, "YOLOv3," 26 03 2021. [Online]. Available: https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/object_detection/YOLOv3. [Accessed 05 04 2024].

[46] Sphinx, "docs/stable/notes/amp_examples.html," 2023. [Online]. Available: https://pytorch.org/docs/stable/notes/amp_examples.html. [Accessed 19 05 2024].

[47] P. M. Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets.," *Information Technology and Management Science ,* pp. 20-24, 2017.

[48] deater, "~vweaver/group/green_machines.html," [Online]. Available: https://web.eece.maine.edu/~vweaver/group/green_machines.html. [Accessed 18 05 2024].

[49] "en/cpu/intel-core-i7-1165g7," NanoReview.net, 2024. [Online]. Available: https://nanoreview.net/en/cpu/intel-core-i7-1165g7. [Accessed 2024 05 18].

[50] "en-gb/geforce/news/gfecnt/nvidia-geforce-rtx-2060/," [Online]. Available: https://www.nvidia.com/en-gb/geforce/news/gfecnt/nvidia-geforce-rtx-2060/. [Accessed 18 05 2024].

[51] V. Verbavatz and M. Barthelemy, "Critical factors for mitigating car traffic in cities," *PLoS one,* vol. 14, no. 7, 2019.

[52] GME Electronics, "v/1514936/kws-mx17-usb-merici-pristroj-oled," GME Electronics, 2024. [Online]. Available: https://www.gme.cz/v/1514936/kws-mx17-usb-merici-pristroj-oled. [Accessed 20 05 2024].

[53] J. Solawetz, "mean-average-precision," Roboflow, Inc., 12 07 2020. [Online]. Available: https://blog.roboflow.com/mean-average-precision/. [Accessed 27 04 2024].

[54] B. Dwyer and J. Solawetz, "glossary," Roboflow, Inc., 05 10 2020. [Online]. Available: https://blog.roboflow.com/glossary/#:~:text=accuracy. [Accessed 27 04 2024].

[55] yozawiratama, "questions," Stack Exchange Inc, 13 06 2023. [Online]. Available: https://datascience.stackexchange.com/questions/122123/how-to-read-confusion-matrix-from-multiclass-dataset. [Accessed 18 04 2024].

[56] "cs/articles/brzdna-draha/," autolexicon.net., [Online]. Available: https://www.autolexicon.net/cs/articles/brzdna-draha/. [Accessed 18 05 2024].

[57] Policie ČR, "/clanek/uzemni-odbor-trutnov-dopravni-inspektorat-rychlost.aspx," 2024. [Online]. Available: https://www.policie.cz/clanek/uzemni-odbor-trutnov-dopravni-inspektorat-rychlost.aspx. [Accessed 21 05 2024].

[58] W. van Winsum, D. de Waard and K. A. Brookhuis, "Lane change manoeuvres and safety margins," *ransportation Research Part F: Traffic Psychology and Behaviour,* vol. 2, no. 3, pp. 139-149, 1999.

[59] "wiki.ros.org," Open Robotis, 2 1 2019. [Online]. Available: http://wiki.ros.org/laser_scan_matcher. [Accessed 19 12 2023].

[60] "/sites/default/files/6507-6508-6509-TQi-Manual-120130.pdf," Traxxas, 04 02 2013. [Online]. Available: https://traxxas.com/sites/default/files/6507-6508-6509-TQi-Manual-120130.pdf. [Accessed 29 03 2024].

[61] userbenchmark.com, "Compare/," 24 02 2024. [Online]. Available: https://gpu.userbenchmark.com/Compare/Nvidia-RTX-3090-Ti-vs-Nvidia-GTX-Titan-X/m1818101vs3282. [Accessed 29 03 2024].

[62] theAIGuysCode, "theAIGuysCode/OIDv4_ToolKit," 1 03 2020. [Online]. Available: https://github.com/theAIGuysCode/OIDv4_ToolKit. [Accessed 29 03 2024].

[63] SteveMacenski, "ros.org," Open Robotics, 16 12 2021. [Online]. Available: http://wiki.ros.org/slam_toolbox. [Accessed 20 12 2023].

[64] S. Ananth, "/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46," Medium, 9 10 2019. [Online]. Available: https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46. [Accessed 31 03 2024].

[65] H. Jonathan, "ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06," Medium, 14 03 2018. [Online]. Available: https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06. [Accessed 31 03 2024].

[66] J. Klapálek, "VYHÝBÁNÍ SE DYNAMICKÝM PŘEKÁŽKÁM S AUTONOMNÍM AUTEM F1/10," ČVUT, 24 5 2019. [Online]. Available: https://dspace.cvut.cz/handle/10467/83424. [Accessed 14 03 2024].

[67] R. Potter, "anolytics," Medium, 23 08 2022. [Online]. Available: https://medium.com/anolytics/how-bounding-box-object-detection-is-taking-over-and-what-to-do-about-it-9130b6d6c1f1. [Accessed 18 04 2024].

[68] "/html/5G," 09 2018. [Online]. Available: https://www.sharetechnote.com/html/5G/5G_MCS_TBS_CodeRate.html. [Accessed 15 05 2024].

Appendix A: Re-arrange and modify the labels in the dataset for detection part

This part of appendix discusses the scripts written in Python that we write to reannotate or re-arrange the original datasets to datasets used in this thesis. First, we learn about the dataset format used by our YOLOv3 implementation and then we discuss the modification process of Open Image Dataset v7 and GTSDB.

A.1. Dataset format

The neural network for detection utilized in this thesis is the You Only Look Once version 3 (YOLOv3). It takes an image of resolution $32 \times x$, where x is a natural number as input and generates a list of labels indicating the location and size of the road signs. These bounding boxes are defined using relative coordinates to the image resolution, comprising four coordinate numbers. The first two represent the relative position of the bounding box's center, while the next two denote its relative width and height. Each label consists of five numbers, including the four coordinates and a class identifier. Although in this thesis is only one class, YOLOv3 retains the ability to classify image classes for compatibility reasons across datasets. The relative coordinates are described in Figure 31.

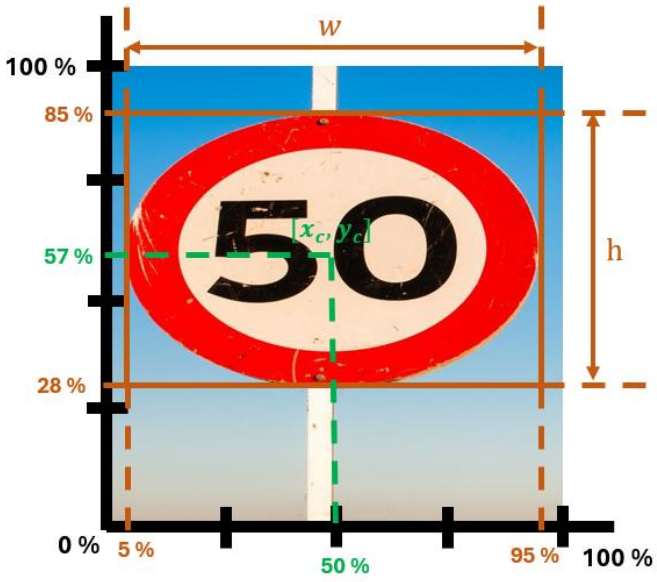

Figure 31: Relative coordinates

The images are stored in JPEG format. The labels are stored in a conventional .txt file format. In cases where there is more than one road sign in the image, a corresponding label is attached to each road sign. Each training and validation image typically has a corresponding text file, with the filename often matching that of the image, albeit with a different file extension. However, this naming convention may not always be strictly followed. Within each text file, every row represents a single label, formatted as follows: Class ID, the relative position of the bounding box center for the X and Y coordinates, and the relative height and width of the bounding box.

The PyTorch dataloader used for training YOLOv3 requires a specific CSV file format. Each row in this CSV file contains the name of an image file along with the corresponding name of the text file

containing the labels. These two names are separated by coma. Two separate CSV files are used, one for training the neural network and the other for validation purposes. All images for both training and validation are stored in a single large folder, and the same applies to the labels. Each dataset includes two folders and two corresponding CSV files.

A.2. Create datasets from Open Images Dataset V7

The dataset is divided into two separate folders, one containing images for validation purposes and the other for training purposes. The labels come in two CSV files, first with labels for training and second with labels for validation. Each row of these files contains the name of the image and the label in this image. Labels consist of class ID, position of the top left, and the top right corners of the bounding box. These positions are in relative coordinates to the resolution of the picture. Some additional information about the relationship between bounding boxes is also presented on this label. This information is not used for creating datasets for YOLOv3. The biggest challenge in this dataset is the number of bounding boxes because images and their annotations are downloaded also with other labels than road signs. For the new dataset, we only chose labels that represent the road sign class, and we assigned the class ID 0 to these bounding boxes. On every picture with road signs are also other labeled objects like cars or others and these things are also written in CSV file.

Conversion between the notation used in YOLOv3 and Images Dataset V7 is described below:

$$x_c = \frac{x_1 + x_2}{2}, \tag{53}$$

$$y_c = \frac{y_1 + y_2}{2}, \tag{54}$$

$$w = x_2 - x_1, \tag{55}$$

$$h = y_2 - y_1, \tag{56}$$

where the x_c and y_c are relative positions of the center of the bounding box, x_1, y_1, x_2 and y_2 are relative positions of the top-left and bottom-right corners, the variable w denotes the width of the bounding box and h means the height of the bounding box. By doing the dataset the images are also resized to 832 pixels. This step is done because we want to save time in the loading process in a neural network, but the data loader can resize the images too.

This process is facilitated by a simple Python script. Initially, the script parses through the CSV file line by line. Each row is then split by commas to extract individual items. Subsequently, the script filters out rows containing the class code "/m/01mqdt", indicative of a road sign class and it works only with these filtered labels. For each of these rows, the first item denotes the image name to which the label belongs. Initially, an attempt is made to locate in the image folder a corresponding .txt file with the same name as the image. In cases where the .txt file is not found; a new one is created. Following this, information from the fifth to eighth item, representing the coordinates (x_1, y_1, x_2, y_2) in sequence, is extracted. These bounding box coordinates are then recalculated to confrom to the YOLOv3 format using (53)-(56). The class ID is added to these coordinates. In our case, it is 0 for every label. Finally, this updated information is appended to the corresponding .txt file associated with the image labels. Script is run separately for the training and the validation folder with the corresponding CSV file.

Now we need to create a CSV file with the names of the images and corresponding labels and then we need to put the images and labels into separate folders common for training and validating. The Python script assumed that both images and labels are stored in a single folder. Since the data for training and validation are divided into two separate folders, the script first loaded the names of all files in a specified folder. These names are then divided into two groups: one for images and the other for text files containing labels. Next, the script iterated over the image names, checking for corresponding text files. If a matching text file is found, the script wrote the file names into the CSV in the format: "image.jpg, label.txt". Files with different extensions are skipped, and if a corresponding text file is not found for an image file, a warning is logged to the console.

The second Python script initially created two folders named "labels" and "images". Subsequently, it loaded all file names from the training and validation folders. Using the load names, the script then copied the files and stored them in the respective folders. JPEG images are copied to the "images" folder, while text files containing labels are stored in the "labels" folder. Any other files present in the original folders are not copied by the script. Open Image Dataset v7 consists of two separate folders, one for validation and one for training purposes. This helps us to create a two separate CSV files.

A.3. Create datasets from GTSDB

This dataset is structured similarly to the Images Dataset V7, with the primary distinction lying in the label notation. It utilizes bounding box descriptions, specifying the absolute positions of the top-left and bottom-right corners in pixels within the image. Additionally, it contains only classes relating to road sign types. During the dataset preprocessing for YOLOv3 compatibility, all classes are unified under a single class labeled "road signs" with a class ID of 0.

Recalculation of bounding boxes to the relative distance used in YOLOv3 from an absolute position is described as follows:

$$x_c = \frac{x_1 + x_2}{2 \cdot w_p}, \tag{57}$$

$$y_c = \frac{y_1 + y_2}{2 \cdot h_p}, \tag{58}$$

$$w = x_2 - x_1, \tag{59}$$

$$h = y_2 - y_1, \tag{60}$$

where the x_c and y_c relative positions of the center of the bounding box, x_1, y_1, x_2 and y_2 are absolute positions of the top-left and bottom-right corners. w is the width of the bounding box and h is its height. w_p and h_p are the width and height of the image in pixels.

The challenge lies in the image format, as the pictures are in .ppm format, which is unconventional. However, Python's OpenCV library can readily convert them to the JPEG format used in this project. During the image loading process in Python scripts, we also resize them from their variable resolution to 832 pixels by 832 pixels. This resizing is done because YOLOv3 accepts input images of this resolution, allowing us to save time in the YOLOv3 pipeline in the training process.

We need to create the CSV files and re-arrange images and labels into separate folders called “images” and “labels”. These scripts are the same as the scripts used at the end of Appendix A.2. for the same purpose.

A.4. Adapting own dataset

The own dataset downloaded from Robotflow is in the YOLO darknet format as it closely aligns with our desired format. In this format, labels are correctly structured and stored in text files corresponding to the image names. However, there is no CSV file for training and validation of the YOLOv3 network. Moreover, all images are stored in two separate folders. The first folder includes validation images and the second covers training images. In both folders, there are images together with their labels. We developed a Python script to generate CSV files, and another script is created to organize the labels and images into two separate folders, one for validation and one for training. Scripts are the same as the scripts used at the end at the end of Appendix A.2. for the same purpose.

Appendix B: Git repository structure

In this appendix we describe the structure of the work at GIT[5], mainly focused on debugging neural network with is in branch NeuralNetwork and implantation to MEC server and autonomous vehicle is in branch Main.

B.1. NeuralNetwork branch:

Main branch for development of neural networks.

B.1.1. YOLO directory:

Implantation of YOLOv3 for the road sing detection. Partly taken from [45].
config.py - Information about YOLOv3 configuration
dataste.py - Load dataset from directory
eval.py - Main file for evaluating - input is file with trained YOLOv3
loss.py - Implement YOLOv3 loss function
train.py - Main file to train YOLOv3 - output is file with trained weights
utils.py - Implementation of support function

B.1.2. NeuralNetworkForClasification/new directory:

Implementation of neural network for the road sign classification.
eval.py - Main file for evaluating - input is file with trained model
train.py - Main file to train neural network- output is file with trained model

B.1.3. helpUtilities directory:

Implementation of help scripts to work with datasets.
compressonJpg.py - Compress all images in directory to desire JPG quality
csvCreate.py - Create a CSV file to ensure YOLOv3 format
drawBoundigBox.py - Open image and show bounding box
kamera.py - Open camera port and capture the images
numberOfImages.py - Calculate the number of images and plot them in graph
porvnej.py - Compare two images in image1_path and image2_path
redirect.py - Redirect the .txt and .jpg to two separate folders
rename.py - Rename the name of .jpg to more suitable format
renotate.py – Re-annotate dataset to YOLOv3 format
resizePicture.py - Resize picture in input directory and save in output directory
showAndDiscard.py – Load image and show it with bounding boxes

B.1.4. pipeline directory:

Implementation of pipeline of neural networks for the road sign detection and classification.
config.py - Information about YOLOv3 configuration
eval.py - Test file for evaluating YOLOv3 with Nural network for image classification
model.py - Implement YOLOv3 architecture
utils.py - Implementation of support function

[5] https://gitlab.fel.cvut.cz/mobile-and-wireless/autonomous-driving/-/tree/main?ref_type=heads

B.2. Main branch:

Main branch of project autonomous vehicle.

B.2.1. MEC/CNN-Kafka directory:

Implementation of pipeline of neural networks for the road sign detection and classification on MEC server in Docker composer.
config.py - Information about YOLOv3 configuration
eval.py - Test file for evaluating YOLOv3 with Nural network for image classification, developed together with Anastas Nikolov
model.py - Implement YOLOv3 architecture
utils.py - Implementation of support function

B.2.2. trax_repo/src/camera/src

Implementation of local processing of neural networks for autonomous vehicle.
camera.py – Take image by camera and send them to ROS environment.
camera2.py-Load images from folder and send them to ROS environment, designed for test purposes
camera3.py-Rake images from camera, based on a message from the manager, it decides whether to send the image to the ROS environment for processing on the MEC server or to process the image locally in the autonomous vehicle.
config.py - Information about YOLOv3 configuration
model.py - Implement YOLOv3 architecture
utils.py - Implementation of support function