Henrik Latsch Haugberg
Øyvind Henriksen
Magnus Nordahl

Bachelor's thesis

A Proposed Solution for Detection, Classification and Geopositioning of Traffic Signs, Utilizing Machine Learning, Images and LIDAR Data

May 2022

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Bachelor's thesis

2022

NTNU
Norwegian University of
Science and Technology

Henrik Latsch Haugberg
Øyvind Henriksen
Magnus Nordahl

A Proposed Solution for Detection, Classification and Geopositioning of Traffic Signs, Utilizing Machine Learning, Images and LIDAR Data

Bachelor's thesis
May 2022

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Modern machine learning methods provides the possibility to have machines perform tasks that can be tedious and time consuming for humans, or to analyze large amounts of data that would not be feasible for humans to process. This thesis presents a prototype system that utilizes machine learning to detect traffic signs in image data, classify the detected traffic signs, and approximate their geolocation using LIDAR data. The YOLOv5 machine learning model for object detection is chosen from several candidates, based on both correct detection performance and possibility for integration in the larger system. A model for classification is constructed using the Tensorflow library in Python, and is based on literature about previous models used for similar tasks. A pinhole camera projection method for determining the geolocation of the detected objects using LIDAR data is proposed and integrated into the prototype. Alternative methods for approximating the geolocation of objects are also discussed. The prototype is a containerized system, which provides a simple web application where an operator can initialize the process of detection, classification and geopositioning on a set of data corresponding to a stretch of road. The system returns a JSON object containing all detected objects, which can be further inspected by the operator and uploaded to a database, along with original images and cropped traffic sign images. The system is integrated with a parallel student project based in Oslo that has developed a prototype for a tool that retrieves this data and makes it possible for an operator to control and make necessary edits.

Sammendrag

Moderne maskinlæringsmetoder gjør det mulig å la maskiner utføre oppgaver som kan være monotone og tidkrevende for mennesker, eller å analysere store mengder data som vil være umulig for mennesker å prosessere. Denne oppgaven presenterer en prototype til et system som bruker maskinlæring til å detektere trafikkskilt i bildedata, klassifisere de detektere skiltene, og finne en tilnærming til skiltenes geolokasjon ved hjelp av LIDAR data. Maskinlæringsmodellen YOLOv5 ble valgt blant flere kandidater basert på både prestasjon og muligheten for integrasjon med resten av systemet. En modell til klassifisering ble satt sammen med Tensorflow-biblioteket i Python, og er basert på litteratur om tidligere modeller som har vært brukt til lignende formål. En pinhole-kamera-metode for å bestemme geolokasjonen til de detekterte objektene ved hjelp av LIDAR-data blir foreslått og integrert i prototypen. Alternative metoder for å finne en tilnærming til objektenes geolokasjon blir også diskutert. Prototypen er et modulært system, som inkluderer et enkelt web-basert brukergrensesnitt hvor en operatør kan initiere prosessen som gjennomfører deteksjon, klassifisering og geoposisjonering på et datasett som korresponderer til en veistrekning. Sytemet returnerer et JSON-objekt som inneholder alle de detekterte objektene, som så kan inspiseres av operatøren og lastes opp en database sammen med opprinnelige bilder og utklipte objektbilder. Systemet er integrert med et parallelt studentprosjekt med base i Oslo som har utviklet en prototype til et verktøy som henter opplastet data, og gjør det mulig for en operatør å kontrollere resultatet og utføre nødvendig redigering.

Preface

This bachelor's thesis concludes our three-year bachelor's degree program in Computer Science at NTNU. This last semester we have been working on this thesis, where we have been exploring the field of machine learning, focusing on detection and classification of objects in images and geopositioning of objects using LIDAR data.

Working on the thesis has been rewarding and challenging, and has sparked interest in further work with these subjects in the future.

Trondheim, May 20, 2022

Henrik Latsch Haugberg
Øyvind Henriksen
Magnus Nordahl

Problem Description

The problem to be solved is to propose methods for automatic detection and classification of roadside objects in image data using machine learning, and a method for establishing the geolocation of these objects from corresponding LIDAR data. The methods should then be integrated into a prototype administration tool. The system should have a web based user interface where an operator can administer the process of running the machine learning models and geopositioning on a set of data, and upload the results to a central database along with images of the objects.

Acknowledgments

First we would like to thank the team from Triona for giving us the opportunity to work on this thesis, and for providing helpful feedback and advice throughout the project. Triona employees involved include Shohaib Muhammad, Magnus Ulstein, Per Ola Roald, Rune Dragsnes, Menno Kemp, Hanne Jeremic and Edvard Tollefsrud.

We want to thank the OsloMet team, Julie Kvarme, Kristian Sørum and Kristine Løken for the collaboration.

We also wish to thank our supervisor from NTNU, Ole Christian Eidheim for valuable feedback and technical discussion.

Contents

List of Figures

List of Tables

Terminology and Abbreviations

AMQP – Advanced Message Queuing Protocol

BNN – Bayesian Neural Network

CD – Continuous Deployment

CI – Continuous Integration

CNN – Convolutional Neural Network

CPOS – Norwegian RTK-service

CSS – Cascading Style Sheets

DL – Deep Learning

DOM – Document Object Model

GTSDB – German Traffic Sign Detection Benchmark

GTSRB – German Traffic Sign Detection Benchmark

IDE – Integrated Development Environment

JSON – JavaScript Object Notation

JVM – Java Virtual Machine

LOB – Line of bearing

mAP – Mean Average Precision

MMS – Mobile Mapping System

NVDB – National Roads Database

OS – Operating System

ROI – Region Of Interest

RTK – Real-time kinematic positioning

STOMP – Simple Text Oriented Messaging Protocol

TS – TypeScript

Chapter 1

Introduction

1.1 Pre-project

This thesis is a direct continuation, albeit with a larger scope, of a project the team worked on in the course IDATT2501 Fordypningsprosjekt in the fall of 2021, which was structured as a pre-project to the bachelors thesis. There has been a natural evolution to some of the objectives and scope throughout the projects, and it is important to see the two projects in unison to understand how the results were reached. The background and motivation presented in the following two sections are copied from the pre-project report, as they are still relevant for this report. The pre-project report is referenced several times throughout this report, and is included as Appendix B to make it possible to see the whole context.

1.2 Background

"Machine learning technology has been in rapid development over the past several decades, and advancements in hardware technology makes it possible to train deep convolutional neural networks more efficiently. A large field in machine learning is object detection and classification, where models are trained to detect objects in images, and correctly classify the type of object. A good example of technologies that are driving development in this field is self-driving cars, where it is necessary for this classification to happen quickly and with a high level of accuracy." (Appendix B)

"Object detection and classification can also be useful in monitoring and maintaining infrastructure. The National Roads Database (NVDB) is the Norwegian Public Roads Administration's central register, and contains information about the Norwegian road network, and objects such as traffic signs, noise barriers and a wide array of other objects of interest are registered here. These objects are often registered manually in the database, with operators using image data collected with a mobile mapping system. Machine learning could be utilized to automatically detect and classify objects and, if not fully automate the process of registering objects, at least make it significantly more efficient by giving suggestions that only need to be checked and edited by the operator." (Appendix B)

1.3 Motivation

"Developing skills in a large and complicated field like machine learning requires time, and it is important to get familiar with previous work and development in technology. Having

an opportunity to work on the same project over two semesters means that the students can spend time gaining a deeper understanding of theory before starting the product development, and this gives more confidence when starting the development process later." (Appendix B)

"When working on this project the students have to utilize all of the different skills developed throughout their studies. It gives an opportunity to take programming skills and knowledge of specific frameworks and development methods, and combine them in a larger context than before." (Appendix B)

"Gaining familiarity with a field like machine learning in particular is also a motivating factor, and it will be rewarding to work with cutting edge technology. Machine learning technology is being utilized in an increasing number of businesses, and many potential future employers will be looking for candidates who have expertise in the field." (Appendix B)

1.4 Objectives and Scope

The problem as defined by Triona was to explore what information was possible to extract from a given type of data sources, recorded from a mobile mapping system (MMS). As the problem was largely exploratory in nature, the team defined a few constraints to what was the desired output from the project. The team would develop a proof of concept-system following Scrum principles. The system should take known inputs from a MMS, and output digitally modelled objects, found within the provided data sources.

The project can be viewed as having two parts that are closely connected. The first part is to explore machine learning models and methods to find solutions suitable for the purpose of this thesis. The result of this is a proposed method for each of the three steps in the machine learning and geolocation module. One model for detecting traffic signs in images, one model for classifying the detected traffic signs, and a method for approximating the geolocation of each traffic sign, are proposed.

The goal of the second part is to implement the proposed methods into a proof of concept administration tool. The tool will allow an operator to run inference on a set of data, and upload the results to a central database.

In parallel with this project, another group of students from OsloMet have been writing their bachelor's thesis, directly related to this project's output. Their thesis involves creating a user interface for manual correction and verification of the data produced in this project.

1.5 Research and Innovation

This thesis does not claim to introduce groundbreaking discoveries to the field of machine learning, but it proposes a solution for efficient registration of roadside objects. For Norwegian companies to stay competitive internationally, and also to reduce the need for buying services from abroad, it is important to participate in the early development of these modern technologies.

Machine learning is a field that has been in rapid development the last few decades, and coupled with improvements in hardware capacity, the areas where it can be utilized are expanding. Tasks that would be time consuming for humans to perform manually, such as analyzing vast amounts of data, can be done in a efficient and cost effective manner using

machine learning. Even if the starting cost of developing these systems can be significant, it will in many cases be economically beneficial in the long run, while at the same time eliminating the need for human labor to perform Sisyphean tasks.

Triona wishes to explore the possibility of developing a system that decreases the need for manual registration of roadside objects, which would both make this process more efficient, as well as making it possible to focus the human resources at other tasks. The system proposed in this thesis is a step towards achieving these objectives.

Chapter 2

Methodology and Technology

This chapter gives an overview of the various research and development methods that have been used when working on this thesis. Since the thesis is a combination of research and software development, these methods include both regular research methods such as literature studies, as well as traditional software development techniques. It also gives an overview of and short introduction to many of the technologies that have been central in the system development part of the project.

2.1 Literature Studies

Literature studies are used to acquire a deeper general understanding of the field of machine learning, and has also been important to get an overview of previous related work in the specific fields that are explored in the thesis. As an example, the architectures of LeNet and AlexNet has been used as inspiration when constructing the CNN used for classifying traffic signs. When attempting to utilize LIDAR data for determining the geolocation of objects, several different methods could be considered. Literature studies have been important in deciding the correct approach for utilizing the data available, and also to look at alternative methods for the case where it is discovered that using the LIDAR data is not viable, or as an alternative for when LIDAR data is not available.

2.2 Pre-project

As previously mentioned, a pre-project for this bachelors thesis was completed as part of another course in the fall of 2021. An advantage of doing a pre-project is to get the opportunity to thoroughly plan the project, and to do more extensive research ahead of starting development. In the case of this project much of the focus in the pre-project was to study literature, and this helped in having a deeper understanding of the topics to be explored in the thesis. The basis for the system to be developed was also set up as part of the pre-project, and this meant it was possible to start implementing the machine learning modules at an earlier stage. Had one not had the opportunity to work on the project for a longer period of time, it is likely that the scope of the project would have to be narrowed down in the end.

2.3 Scrum

Since a significant part of the project is developing a proof of concept, the team have been following Scrum methodology through the entire project. Sprints have had a duration of between 6-8 working days, with sprint plannings and sprint reviews at the beginning and end of each sprint. A Scrum board has been used where user stories have been made for each component of the system. Each user story was split into smaller tasks with an agreed upon time estimate, and at the beginning of each sprint a goal was set for which tasks to complete during the sprint. Daily stand-ups have been held, where supervisors from Triona have participated and given advice on both technical issues and project management.

2.4 Technology

2.4.1 Docker

The open-source platform for containerization, Docker, uses isolated, lightweight, containerized applications to run simultaneously on a host environment. Containers offer all the functionality of VMs, which include isolation, scalability and disposability, as well as extra functionality. A workflow using continuous integration (CI) and continuous delivery (CD) is well integrated with the use of Docker containers. Docker makes it easier and safer to develop distributed applications, and also deliver these applications to potential cloud environments. (Docker 2021; Education 2021)

2.4.2 React

React is a JavaScript library developed and maintained by Meta for building user interfaces. It uses a component based system, where each component manages its own state. The rendering logic for the components is coupled with the user interface logic, e.g. how events, state changes and data preparation is handled. Each component can be thought of as a reusable piece, with a state and a lifecycle. React also has its own syntax, called JSX, which is a syntax extension to JavaScript, which allows the production of React 'elements' that can be displayed in the Document Object Model (DOM). (Meta Platforms 2022)

2.4.3 TypeScript

TypeScript is a programing language developed and maintained by Microsoft. It is a superset of JavaScript and is an optional static type checker, meaning it checks a program for error before execution. TypeScript offers all of JavaScript's features, as well as the functionality to add types by inference, as well as defining custom types. Types can be composited to create complex types, by using unions and generics. (Microsoft 2022)

2.4.4 Gradle

Gradle is an open-source build automation tool. It focuses on being flexible, with high customization. Gradle is fast, by reusing outputs from previous task executions, processing inputs only if they have been altered, and running tasks in parallel. It runs on the Java Virtual Machine (JVM), and has support from several major integrated development environments (IDEs). (G. Inc. 2021)

2.4.5 Spring Boot

Spring Boot is an open-source Java framework developed by Pivotal Team. It is used to build standalone and production ready applications. Spring Boot has several features, including flexible configuration of Java Beans, XML, and Database Transactions, batch processing and REST endpoints, auto configuration, annotations, dependency management, and also has an Embedded Servlet Container included. (Tutorialspoint 2022)

2.4.6 Flask

Flask is a Python micro-framework, mainly used for creating web applications. It is 'micro' by default, meaning that the framework does not require any tools or libraries. It does not have any database abstraction layer, form validation or other functionality typically found in third-party libraries. This can be added through extensions which acts as if they were already implemented in Flask. Some of Flask's features entails a development server, unit testing support and RESTful request dispatching. (Pallets 2010)

2.4.7 Celery

Celery is an open-source asynchronous task queue or job queue. Task queues are used to distribute work across threads, where dedicated worker processes constantly monitor the task queues for changes. It communicates via messages, which is usually done by a message broker, to mediate between clients and workers. Celery supports scheduling of tasks, but mainly focuses on running tasks in real-time. (Solem 2021)

2.4.8 RabbitMQ

RabbitMQ is an open-source message broker. It originally implemented AMQP, but has later been extended with the support of STOMP and other protocols. RabbitMQ supports asynchronous messaging, with features such as message queuing, delivery acknowledgement, and flexible routing. It also supports cross-language messaging, with programming languages such as Java, .NET, PHP, Python, JavaScript, and several others. RabbitMQ is lightweight and can be deployed to public or private cloud environments, as well as supporting authentication and authorization. (V. Inc. 2022)

2.4.9 STOMP

STOMP is a protocol for asynchronous communication between clients and servers via text based messages. It provides higher semantics, and is often derived on top of WebSockets. STOMP is inspired by the HTTP protocol, in its structure as a frame based protocol, where the default encoding for STOMP is UTF-8. A STOMP server is has a set of destinations where messages can be sent to. A STOMP client is a user-agent that can act as a producer, by sending messages to a destination on the server via a SEND frame, or as a consumer, by receiving messages from the server via a SUBSCRIBE frame. It is possible for a client to act in both modes simultaneously. (STOMP 2012)

Chapter 3

Theory

This chapter gives an overview of theory that is relevant for this thesis. It includes article reviews and introductions to machine learning methods that has been used in the project, as well as some pinhole camera model theory.

3.1 Article Reviews

To gain a greater understanding of the problem domain, it is beneficial to study related work that has already been done in the field. Reviews of earlier research articles are performed, both to summarize information from lengthy technical reports that directly relate to the solution proposed in this thesis. It also aids in gaining personal understanding of the literature, and to give some insight into alternative methods that have been researched or considered. Some article reviews done during the pre-project can also be found in Appendix B.

3.1.1 (Balado et al. 2020) Novel Approach to Automatic Traffic Sign Inventory Based on Mobile Mapping System Data and Deep Learning

During the second round of searching for previous work relevant to the project, the focus shifted from object detection and classification, and started lookings towards geolocating of said objects. A rough idea was already outlined, of how the LIDAR data could be interpreted, but none of the necessary techniques were known to the team at the time. A number of articles describe similar sets of data, but with different objectives, constraints and solutions, that were often not applicable to this project. This paper describes a viable solution along what was already imagined, and this review summarizes their findings, and the major talking points of the paper.

The solution to traffic sign inventory described in this paper, involves using state of the art object detection and classification models, to first detect, then classify traffic sign objects. The objects are then geolocated using a pinhole camera model, and finally duplicate detections are filtered out.

They give four main reasons to why using point clouds for detection and classification may be sub optimal:

- "DL (Deep Learning) point-cloud processing techniques are computationally more expensive than their equivalent in image processing" (Balado et al. 2020)
- "The addition of point-cloud data to images increases processing times" (Balado et al. 2020)
- "TSs (Traffic Signs) are not always in good condition to be detected by their reflectivity" (Balado et al. 2020)
- "The low point density does not provide useful information for TSR (Traffic Sign Recognition/Classification)" (Balado et al. 2020)

It is argued that using separate machine learning models for detection and classification allow for greater modularity. As better machine learning become available, they can be interchanged, without great disruptions to other stages of the workflow. This is in contrast to some other projects utilizing object detection models to both perform detection and classification in one single model.

The paper describes using a RetinaNet model for object detection, with an unnamed dataset. The dataset separates detections into five classes, distinguishable by colour and shape. The RetinaNet model is chosen for it's good performance with unbalanced datasets and at several scales of object sizes.

The proposed system utilizes three different classification models. Of the five detected classes in the detection stage, both 'stop' and 'yield' signs are directly inferred to their class after detection, while the last three classes of signs, 'Danger', 'Prohibitory' and 'Mandatory' are sent to their respective model. The models used for classification is 3 separate Inception V3 models, which take inputs of $299 \times 299 \times 3$ pixels. The models are trained on a combination of Spanish, German and Belgian traffic signs.

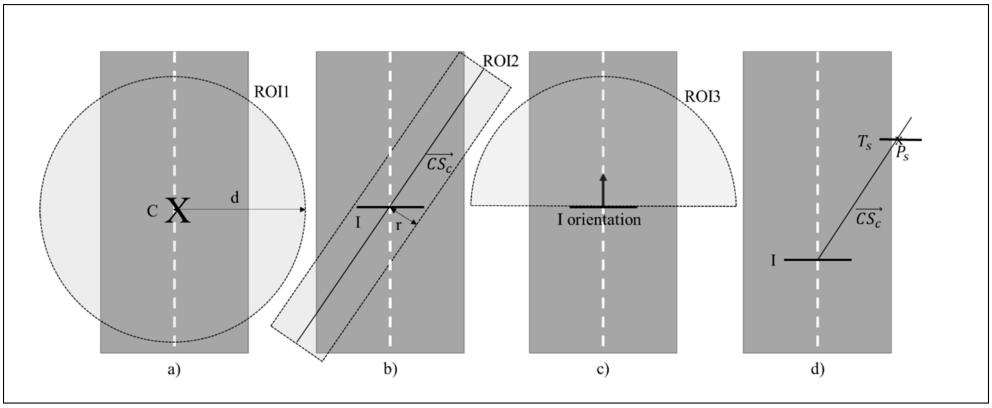


Figure 3.1: Top view of ROI delimitation in a point cloud road environment: **a**) delimitation by distance d from camera location, (**b**) delimitation by distance from projection TS line, (**c**) delimitation by image from camera location, (**d**)location of P_s in the first S of points forming a plane and crossed by $\overrightarrow{CS_s}$. (Balado et al. 2020)

The geolocation stage starts out by filtering out points into three different regions of interest (ROI), which are illustrated in figure 3.1. They include **ROI1**, which filters points further than distance d away from the MMS vehicle at time of recording. **ROI2** defines a

projection line $\overline{CS_s}$, and discards all points that are a distance r or further from the projection line. **ROI3** discards all points opposite to the direction of interest. The points present in all three ROIs are then combined, to form the final ROI. At this stage the remaining points in the ROI point cloud, is filtered by a plane detection method, that discards points that do not lie in the plane T_S, assuming that the traffic sign is mounted near perfectly perpendicular to the road. The last illustration in the figure, shows the resulting points of point cloud P_S, which all reside in the plane T_S which filters out the remaining noise present in the point cloud.

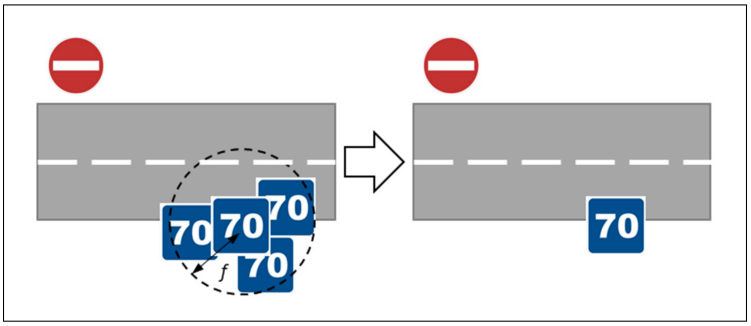


Figure 3.2: Illustration of how duplicate detections are handled. (Balado et al. 2020)

Filtering out the duplicate detections are done based on the rule that a sign post can only contain one of each sign type, so a distance f is determined, and all equal detections within f distance of each other, are discarded, illustrated in figure 3.2.

	EP9701		EP9703		TOTAL	
Total detections	113		137		250	
TS total	98		116		214	
TS detected	86	87.8%	106	91.4%	192	89.7%
TS undetected	12	12.2%	10	8.6%	22	10.3%
False detections	22	19.5%	27	19.7%	49	19.6%
TS duplicated	5	5.1%	4	3.4%	9	4.2%
TS correctly classified	84	92.3%	102	92.7%	186	92.5%
TS uncorrectly classified	7	7.7%	8	7.3%	15	7.5%

Table 3.1: Results from the two case studies performed in the paper, **EP9701** and **EP9703** (Balado et al. 2020)

Finally, the results from two different case studies of the system (Table 3.1) is presented, in which a total of 250 traffic signs are detected, among 214 existing. The two studies involve two different stretches of rural roads in Spain, covering 9.2km and 5.5km of road, in a total of approximately 12.000 images, and point clouds of 350 million and 180 million points.

Summarized, the detection rate was 89.7%, while 10.3% of signs were not detected. False detections were at 19.6%, of which a significant amount were traffic mirrors. Duplicated traffic signs accounted for 4.2% of detections. 97.5% of geolocations were actual points from the original signs, and only 5 total traffic signs were placed with an error of 0.5m to 8m distance from the original post. The classification rate was 92.5%, not counting any signs that the models were not trained for.

3.1.2 (Li et al. 2022) Automatic Positioning of Street Objects Based on Self-Adaptive Constrained Line of Bearing from Street-View Images

Perhaps the biggest challenge of this project has been to find a good method for establishing the geoposition of detected objects. LIDAR data was available for the area where the data used in this project was collected, and the main focus has been to utilize this data in some way, but it was also desirable to do research on possible alternative methods for geopositioning. The article reviewed in this section proposes a method for automatically positioning of objects in street-view images, based on results from object detection and line of bearing (LOB).

After object detection is done and a bounding box has been marked in the image, the image space coordinates of the center pixel of the bounding box is obtained, and combined with the absolute location and attitude of the vehicle, the mapping relationship between the bounding and the observation orientation is calculated, and the LOB is constructed. When the same object is observed in several images, and a LOB is constructed for each, it is possible to find the point where the LOBs intersect. As shown in figure 3.3, all of the intersections does not necessarily overlap completely, but are aggregated in a cluster within a certain range, where the center of the cluster represents the geolocation of the object. A problem that arises is ghost nodes, where LOBs intersect in locations where no object is observed, as shown in figure 3.4. This study introduces two constraint rules for elimination of ghost nodes. (Li et al. 2022)

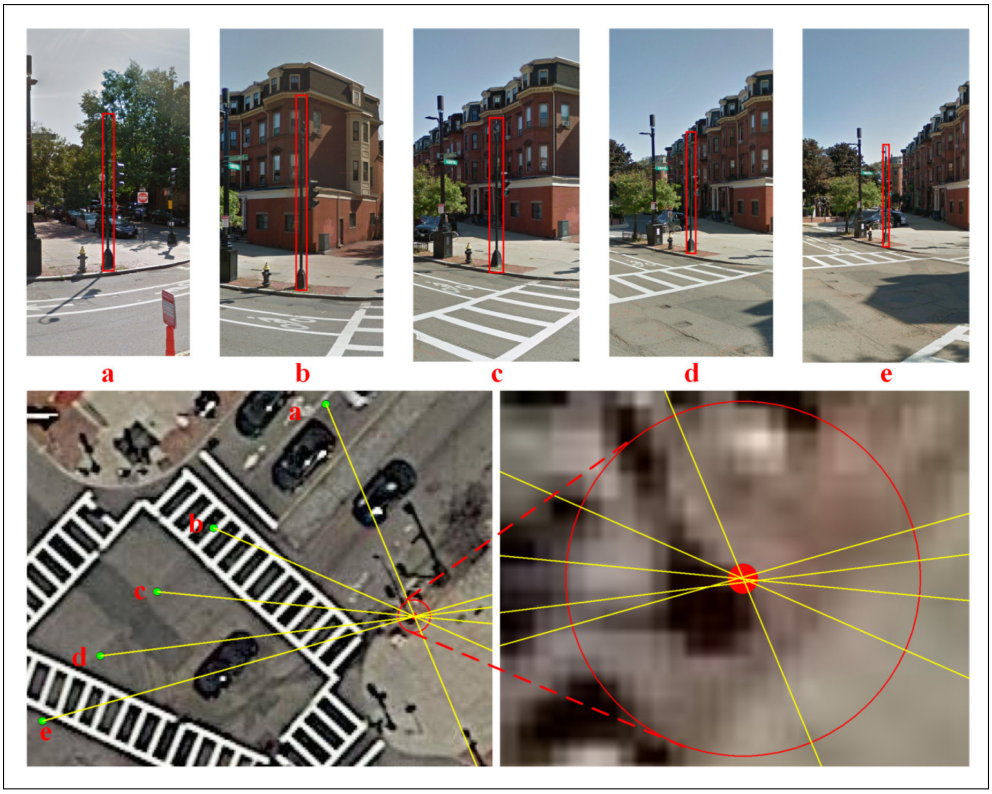


Figure 3.3: Schematic diagram of bounding box, LOB, and cluster: letters (a) to (e) represent different views. (Li et al. 2022)

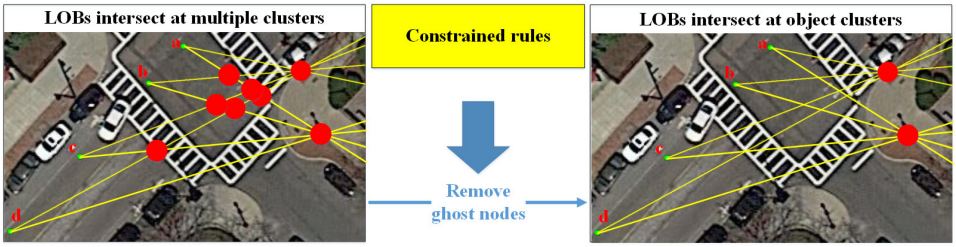


Figure 3.4: Schematic diagram showing reduction of the influence of ghost nodes on the positioning results based on constrained rules: letters a to d represent different views. (Li et al. 2022)

Constraint rule 1 is based on the number of intersections i the cluster. The number of intersections in each cluster is counted, and if there is only one intersection the cluster is determined to be a ghost node. In figure figure 3.5 all the nodes marked with 'x' are considered ghost nodes by this constraint, and will be deleted, while the nodes marked with '?' are candidates that will be further judged in the next step. (Li et al. 2022)

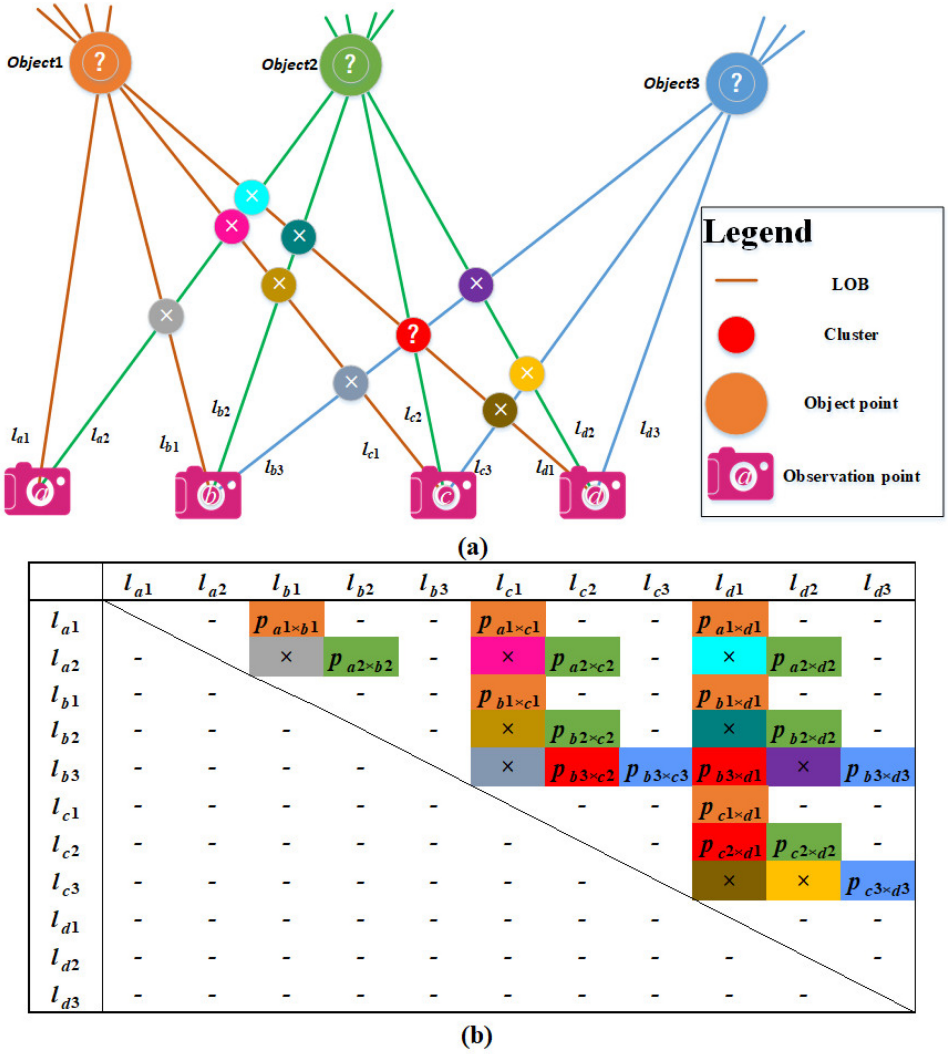


Figure 3.5: Elimination of ghost nodes based on the rule of minimum number of intersections in the cluster: (a) deletion of clusters that do not satisfy this rule; (b) disassociation of relationships that do not satisfy this rule. (Li et al. 2022)

Constraint rule 2 is based on the uniqueness of LOB association. If a LOB is only associated with one cluster of intersection, this cluster must be the location of an object. In figure 3.6 we see that the LOB l_{a1} is only associated with the Object1 cluster, which means this must be an object location. The LOB l_{d1} is also associated with this cluster, but in addition has an association with the red cluster. Since the Object1 cluster has to be an object, l_{d1} can be disassociated from the red cluster, which then only has one intersection and can be eliminated by constraint rule 1. (Li et al. 2022)

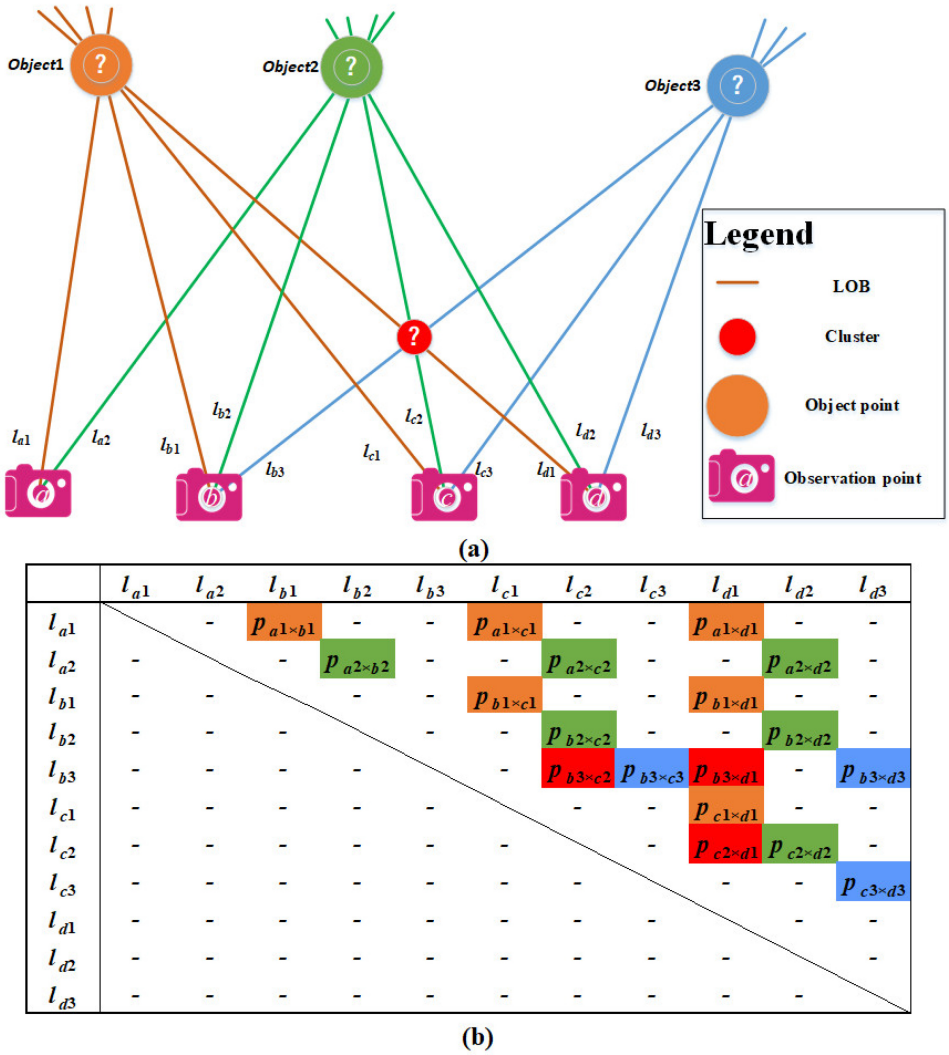


Figure 3.6: Elimination of ghost nodes based on the uniqueness of LOB association: (a) clusters filtered by this rule; (b) associations filtered by this rule. (Li et al. 2022)

The study concludes that the proposed method is effective for positioning of large-scale pole-like objects, and that the positioning has high accuracy. (Li et al. 2022)

3.2 Machine Learning

Machine learning or the greater field of artificial intelligence, is the act of mimicking human intelligence within a computer. Machines generally perform certain specific tasks very well, but often struggle in more open and general areas of intelligence, where for example humans excel. This need for specific and well defined tasks, is a defining character of artificial intelligence at the current stage. Machine learning is one of the

major driving forces of change within the computer science field in the twenty first century. Surveillance, research and fabrication are typical examples of tasks where humans are replaced, in favour of machines.

3.2.1 Object Detection

Object detection is a part of the greater field of 'computer vision'. It encompasses all tasks involving detecting certain objects within an image, a video or a three dimensional point cloud. This detection can be used to spot defects in manufacturing, or performing surveillance etc. Complex machine learning models are trained on images with existing 'ground truth' annotations, and get constant feedback on performance. Object detection models often either focus on high accuracy, or high speed inference. Depending on the domain the model is to be used in, one is usually more important than the other, and models are chosen based on this evaluation.

3.2.2 Classification

Classification is a typical machine learning task, where algorithms are trained to take an input of a given format, and output which class the input belongs in. The number of outputs are determined at the time of training the model, and the models have certain known constraints. The model will always tell what it thinks the input object in fact is, but can often struggle with estimating own certainty in it's predictions. Sometimes this can be confusing, as humans are used to having the ability to also tell when an object is in fact 'none of the above'.

3.3 LIDAR

A LIDAR sensor, is a sensor technology which uses a laser beam, to measure distances to objects in the real world. Often deployed as a spinning device on top of for example a moving car, the sensor performs thousands of measurements per second, and records a depth map to all objects in its vicinity. This depth map can be combined with geolocation sensors, to create a 3D map of the world with global coordinates. Data from LIDAR sensors are often stored in files referred to as 'point clouds', with each point representing one measurement each. (McManamon 2019)

3.4 Pinhole Camera Projection

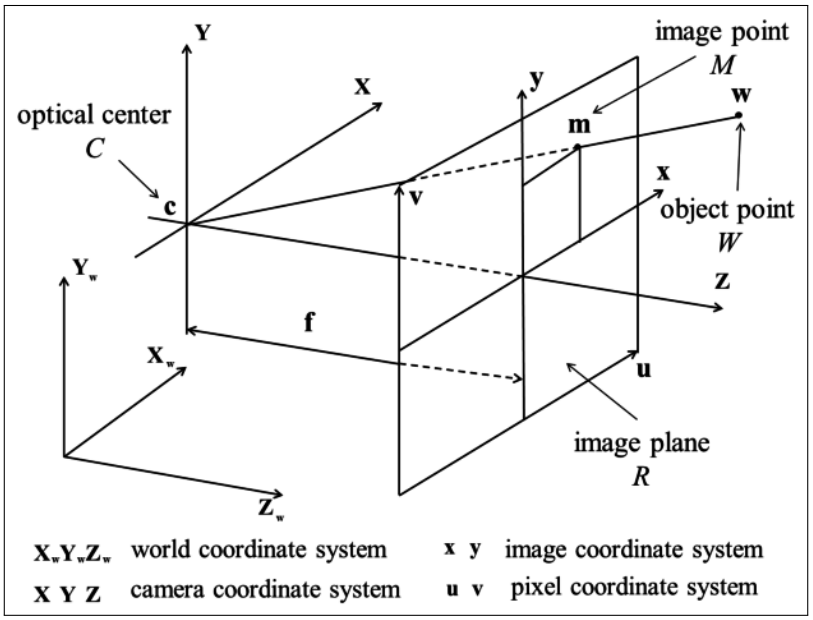


Figure 3.7: Illustration of a pinhole camera model (Ko et al. 2016)

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$

Where **P** is the camera projection matrix, **K** is the intrinsic matrix, also known as the calibration matrix, of 3×3 in the form

$$\mathbf{K} = \begin{bmatrix} \alpha_u & \gamma & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

where $\alpha_u = s_u f$ and $\alpha_v = s_v f$ are the focal length expressed in the u and v axes. f denotes the physical focal length in millimetres, and s_u and s_v are scaling factors. (u_0, v_0) are the coordinates of the principal point, where the optical center intersects the image plane. γ is the skew factor, but for simplicity, we can ignore it, since horizontal and vertical focal length are assumed to be the same. **K** can then be simplified to

$$\mathbf{K} = \begin{bmatrix} \alpha & 0 & \frac{w}{2} \\ 0 & \alpha & \frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

With (w, h) being the width and height of the image.

[R | t] is the extrinsic matrix of format 3×4, where **R** is the rotation matrix at dimension 3×3, which is concerned with the rotations of the camera. **t** is the displacement vector of 3×1, which places the camera within a coordinate system. (Ko et al. 2016)

Chapter 4

Machine Learning Models and Geopositioning – Implementation and Results

This section explains in detail the system implemented for this thesis, including the datasets used for model training, and for testing the system's performance. Results for each individual component is presented, as well as a presentation of the prototype application.

4.1 Data Sources

The primary data sources used for testing the system proposed in this project, are images, LIDAR data and geographical data. They are all recorded simultaneously from a Mobile Mapping System (MMS) mounted on a car, as shown in figure 4.1. The data set used in this project was recorded in Oslo, in the summer of 2021, as part of a project to create digital models of the road network in the county. An estimated 1365km of roads and bike paths were recorded for this dataset. The data is all unannotated, meaning it can't be used for training machine learning models currently. Creating annotations for the dataset, is described in subsection 5.1.3.

The data is organized in a directory system where each stretch of road or street has a unique identifier and matching sub folder, where both images and LIDAR data are stored. Along with this, a separate tsv-file contains all geographical data.

Figure 4.1: The MMS vehicle used to record data (Triona, 2022)

4.1.1 Images

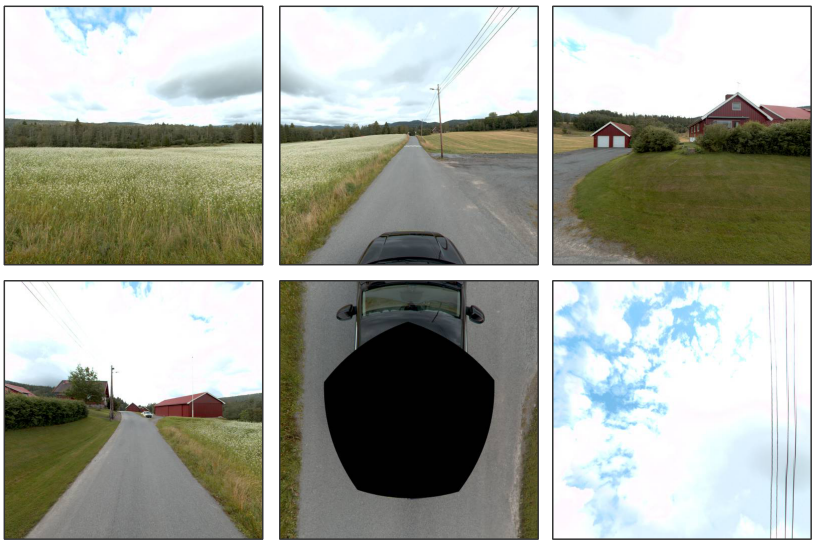


Figure 4.2: Example images from the six different cameras of the MMS (Triona, 2021)

Images from the MMS are recorded in 360 degrees, by six independent lenses approximately every 5 meters. The resulting images are both available as individual images, or stitched together as panorama images. Images are available in both 2048 $\times$ 2048 pixel resolution, and 512 $\times$ 512 pixel resolution.

4.1.2 LIDAR Data

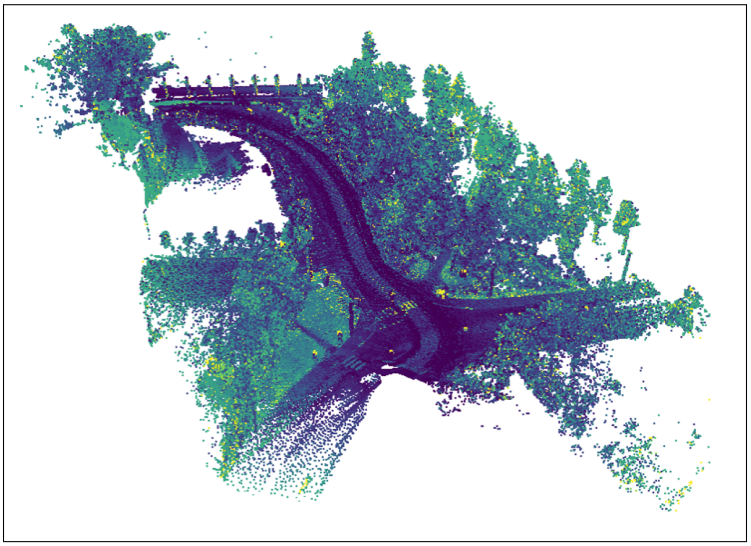


Figure 4.3: Example of a point cloud recording from Oslo, 2021 (Triona, 2021)

A Velodyne LIDAR sensor records point cloud data continuously, while the vehicle is moving. The recorded data is cut into individual point cloud files, in the .laz format. The resulting files are approximately 100MB in size and cover roughly 200-300 meters of road each. Each point cloud has around 1,5 million individual points. Each point holds information about it's coordinates and the intensity of the laser beam's returned strength.

The intensity is a value between 0 and 255, where 0 is a low intensity/reflectivity, and 255 is perfect reflection of the laser beam. All of the coordinates are recorded in the UTM zone 33N.

4.1.3 Geographical Data

In addition to the cameras and LIDAR sensor, the MMS carries a GNSS sensor, with RTK correction by CPOS, promising precision within centimeters of actual coordinates. The GNSS sensor records a geolocation snapshot of the vehicle for each position where an image is taken. This snapshot contains coordinates, altitude, heading, roll and pitch of the vehicle. All of the resulting information is stored in a .tsv-file, which connects each location to their respective images.

4.2 Object Detection Model

In the pre-project, for which the report is included in Appendix B, several object detection models were tested and trained on a few different open source datasets. The object detection model used in the final system, is a product of the results from the pre-project. A YOLO-type model in combination with the Open Images traffic sign subset produced the most promising results.

4.2.1 YOLO v5

In the pre-project a YOLO v4 model showed the most promising results. YOLO models generally prioritize speed over precision, but can still be powerful in certain scenarios where precision is the highest priority. (Redmon et al. 2016)

The original YOLO v4 model trained for the pre-project was a fairly large model trained in the Darknet framework. The Darknet framework relies heavily on utilizing GPU-enhancement to speed up both training and inference, thus inference was fairly slow on consumer computers without GPU-acceleration options. Inference time for this model would be around 5 seconds per image, and it made testing larger amounts of data tedious. (Redmon 2013–2016)

A lighter weight model that could perform faster inference without sacrificing too much accuracy was therefore needed. The solution was a YOLO v5 model trained in the Ultralytics/YOLOv5 framework. This interpretation of the YOLO architecture runs in PyTorch, and was able to decrease inference times drastically, compared to performance from the earlier Darknet model. (Jocher et al. 2022)

Ultralytics's YOLO framework offers 5 different models, each increasing in weight and performance. After testing, it was deemed satisfactory to use the second largest of the models, 'YOLOv5L'. This model takes an input of colour images at 640x640 pixel resolution, and performs inference at subsecond speeds, without GPU acceleration.

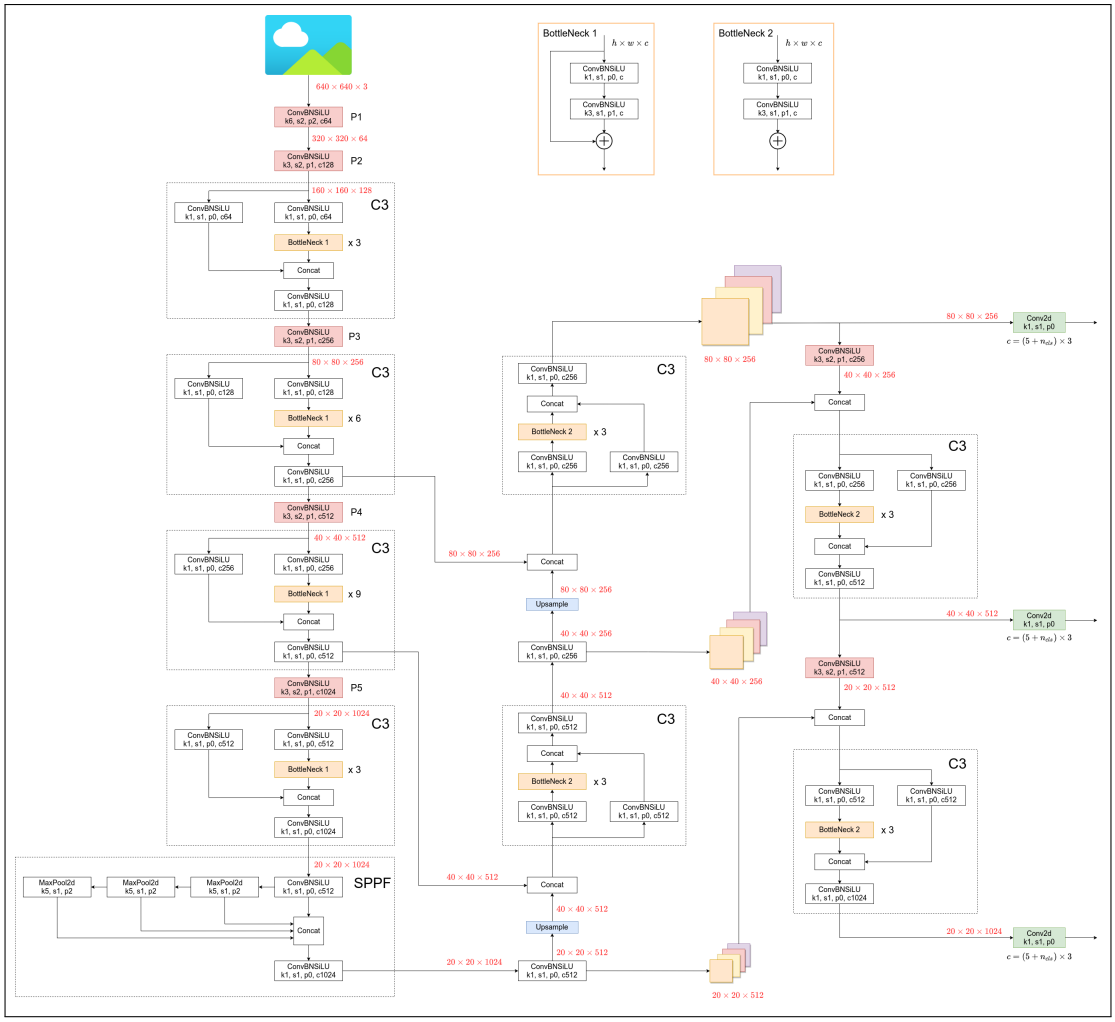


Figure 4.4: Architecture of the YOLOv5L model. (Jocher et al. 2022)

Training the model to convergence is estimated to take 6 days, on a V100 GPU. Although significant training can improve performance, the model will likely reach a sufficient performance quite early, when training on a dataset with only one target class. In figure 4.5, the model performance is shown during 20 epochs of training on the 2800 images of traffic signs from Open Images. This took approximately 8 hours of training on Google's Colaboratory free tier service. The main focus in early training is the **mAP_0.5** score, which quickly reaches a satisfactory level. Mean average precision (mAP) measures how accurately detected bounding boxes intersect ground truth bounding boxes. **Precision** is calculated by dividing True Positive observations, by all detected observations (True Positive + False Positive). **Recall** is found by dividing True Positive detections by (True Positive + False Negative). In short, the precision tells how precise the detected objects are (regardless of missed detections), and the recall tells how many of all available objects are detected. The **mAP_0.5:0.95** score is a more precise measurement than mAP_0.5, and essentially just measures the same score with a lower tolerance for error. mAP_0.5:0.95 becomes more essential towards the end of model training. With a recall of almost 100%, and a precision of near 90%, it means that the model is detecting false positives at this stage. This is supported by early testing, when undertrained models had a tendency to eagerly overdetect objects. Further training remedies this effect. (Research 2021)

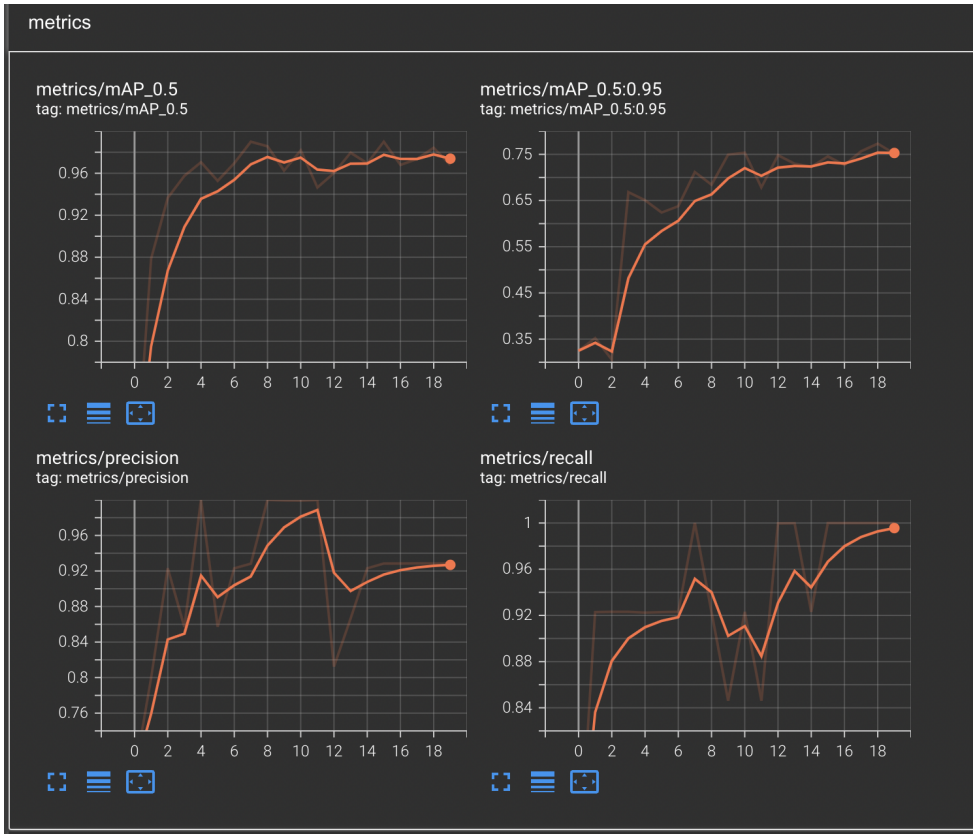


Figure 4.5: Training the YOLOv5L model for 20 epochs on Open Images traffic signs

4.2.2 Open Images Dataset

The Open Images dataset is a database of more than 9 million open source images, with 600 classes. It features rich annotation types, and on average more than 8 different objects per image, making the dataset adaptable for a wide range of detection tasks. (Kuznetsova et al. 2020)

The reason for choosing this specific dataset, was the overall score from testing during the pre-project. Another potential alternative was the German Traffic Sign Detection Benchmark (GTSDB) dataset, which has much more rigid boundaries to what should be detected. Since many Norwegian traffic signs do not resemble their counterparts from the GTSDB dataset, the resulting performance was suboptimal. The results from the Open Images dataset does produce some false positives, but it has been generally agreed upon that these are less costly than false negatives. Ideally, one would have a dataset of Norwegian traffic signs for detection, but to date, no open source datasets of this kind have been found.

For the specific task of training traffic sign detection, Open Images contains approximately 2800 images of traffic signs annotated with bounding boxes. Image resolution is generally high, with a natural variance since they are crowd sourced. Images have a wide range of variety, and include signs from different continents and countries. Some of the images have a rather liberal interpretation of what constitutes a traffic sign i.e. not actual traffic signs, but rather pedestrian signs or similar looking signs.

The great variation in the traffic sign images of this dataset is also possibly the source to its greatest strengths and weaknesses. Models trained on this dataset will have a very generalized idea of what a traffic sign looks like, and could likely be used with similar

accuracy in different continents. This effect is likely to produce a number of false positives, with this being its greatest weakness. Specifically it has been found that for example traffic lights, are detected as traffic signs. This example highlights the 'generality' of what is annotated as traffic signs in this dataset.(Appendix B)

Figure 4.6: Example of traffic sign images from Open Images (Kuznetsova et al. 2020)

4.2.3 Object Detection Results

Model 1: YOLO v4

Threshold	True Positive	False Positive	False Negative	Bonus	Precision	Recall
10%	72	23	8	83	**75.78%**	**90%**
50%	66	1	14	36	**98.50%**	**82.50%**

Model 2: Faster R-CNN Inception Resnet V2

Threshold	True Positive	False Positive	False Negative	Bonus	Precision	Recall
10%	42	0	38	4	**100%**	**52.50%**
50%	39	0	41	2	**100%**	**48.75%**

Model 3: EfficientDet D0

Threshold	True Positive	False Positive	False Negative	Bonus	Precision	Recall
10%	37	63	43	2	**37%**	**46.25%**
25%	23	9	57	0	**71.87%**	**28.75%**
50%	16	1	64	0	**94.11%**	**20%**

Table 4.1: Object detection results from 143 test images containing 80 annotated signs

The initial development of object detection models was performed during the pre-project stage. In the pre-project, a study was performed to determine the accuracy of different object detection models with different datasets. The YOLO v5 model used in the final implementation is a result from this study, and even though the architecture is now a v5 model rather than a v4, their performance should nearly be identical, when trained on the same dataset. The results from the study performed in the pre-project is presented in Table 4.1, which highlights especially the superior recall score of the YOLO model. The 'bonus' column were detections of traffic signs that were deemed too far away or occluded during annotation. To summarize, a bonus detection means a higher sensitivity. This

study can not be treated as a direct comparison of the models, as each model was trained on different datasets. Despite this, the YOLO model showed the best results in the given context.

4.3 Classification Model

To perform classification of objects, a convolutional neural network (CNN) model was developed. The following section describes the model in detail, and the dataset used to train the model.

4.3.1 Convolutional Neural Network with Tensorflow

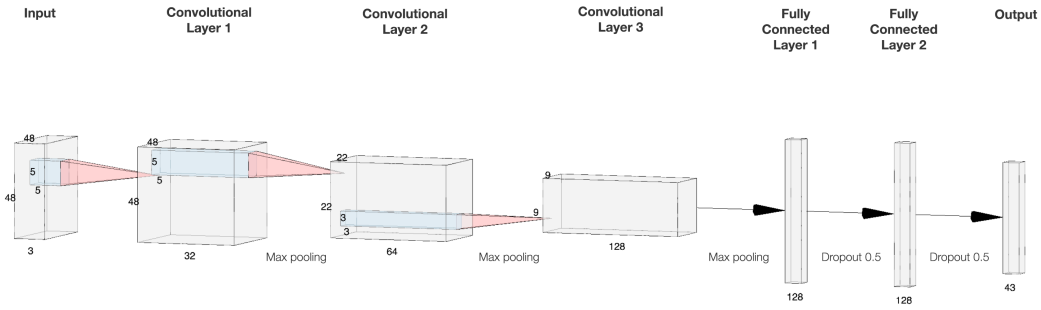


Figure 4.7: Illustration of the CNN model developed for the system

The deciding factors to designing the CNN model for this project, came down to the size of the input data source during inference, and the size of inputs from the training data. Inputs from inference can be as large as 400 $\times$ 400 pixels , down to 50 $\times$ 50 pixels at the smallest acceptable size. The majority of images in the GTSRB dataset, used for training, are between 30 $\times$ 30 and 50 $\times$ 50 pixels. This is also likely why the best performing models on the GTSRB dataset usually don't need an input size of more than 48 $\times$ 48 pixels.

With the constraint of training data-size in mind, the resulting model takes an input size of 48 $\times$ 48 $\times$ 3 pixels. The model takes RGB input colours, but could possibly also have used greyscale images, without much detriment to performance.

Table 4.2 shows a detailed account of each layer in the CNN model. This general architecture takes inspiration from the famous 2012 paper on 'Alexnet' (Krizhevsky et al. 2012). Three ReLu activated convolutional layers, with a doubling of filters in each layer, from 32 up to 128. The kernel size on the first two layers are 5x5, with the last layer being 3x3. Each convolutional layer is followed by a max pooling layer, to reduce the overall size. The max pooling is followed by two fully connected layers with ReLu activation. Each fully connected layer is followed by a dropout layer to prevent overfitting, set to a sensitivity of 0.5. The final output layer uses a softmax activation function, to output a probability-score, but these kinds of models have a tendency to be overconfident. These output values should generally be disregarded as confidence scores, even though it is tempting to treat them as such. It is treated as a 'better than nothing'-value currently, given the model is too confident in its predictions. A potential future work-item could include looking into Bayesian Neural Networks (BNN), if outputting

model confidences are important to the system as a whole. (Kwon et al. 2020)

Layer	Input	Output	Activation	Parameters	Kernel	Filters
Image		48x48x3				
Conv1	48x48x3	44x44x32	ReLu	2432	5x5	32
Max Pooling	44x44x32	22x22x32				
Conv2	22x22x32	18x18x64	ReLu	51264	5x5	64
Max Pooling	18x18x64	9x9x64				
Conv3	9x9x64	7x7x128	ReLu	73856	3x3	128
Max Pooling	7x7x128	3x3x128				
FC1	1152	128	ReLu	147584		
Dropout(0.5)	128	128				
FC2	128	128	ReLu	16512		
Dropout(0.5)	128	128				
Output	128	43	Softmax	5547		

Table 4.2: CNN model architecture

Before training the dataset is augmented in a few different ways, to increase robustness of the model. A few of the classes in the GTSRB dataset are severely underrepresented, which can be detrimental to model performance. This kind of dataset imbalance is a common pitfall of model accuracy, as it will potentially favour the larger classes of data. In data augmentation, a portion of all classes are duplicated and slightly altered, to seemingly increase their number of appearances in the dataset. The classes that are underrepresented are duplicated at a higher ratio, to try to balance out the dataset. The benefit of this augmentation is also that the individual traffic signs are seen in a number of slightly distorted and occluded ways, which usually increases model robustness.

Training the model to peak performance is reached in a matter 3-4 of hours on a consumer laptop. It is performed with an Adam optimizer starting with approximately 100 epochs at a learning rate of 0.0001, and steadily decreasing the learning rate for around 100 more epochs, until the lower learning rate starts decreasing accuracy. A bit of trial and error is needed to achieve the last few percentages of accuracy during training. Often reloading the model from an earlier stage, and trying another set of parameters, to compare outcomes.

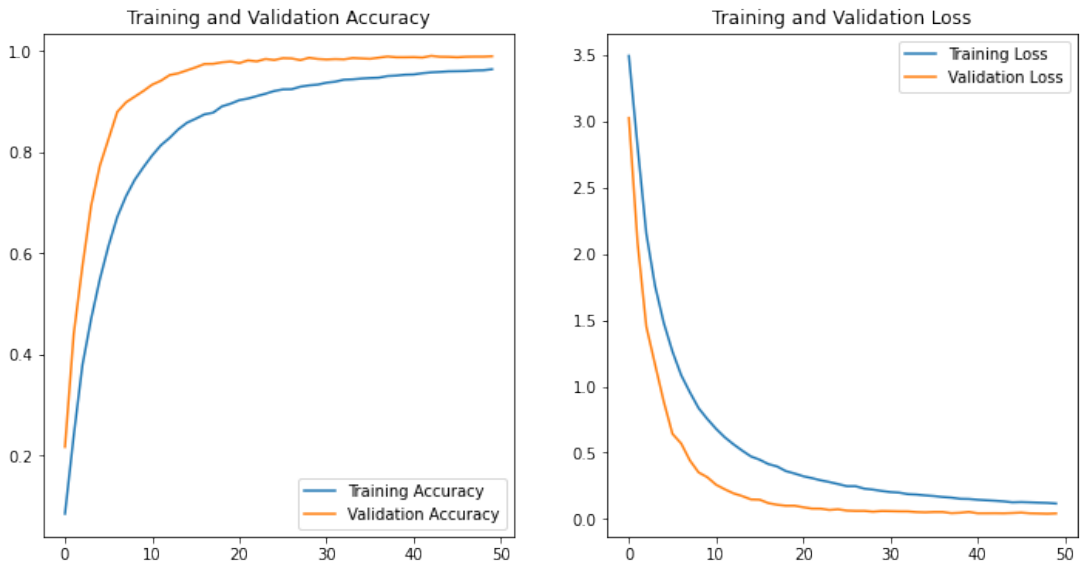


Figure 4.8: Accuracy and loss values of the CNN model for the 50 first epochs

This particular model achieves an accuracy of approximately 96% on the test set of images from GTSRB. This score is perfectly adequate, although this metric has little bearing on it's use in the final system. Given better quality training data, the model could potentially be scaled up, to take advantage of the higher resolution images in the data source images.

4.3.2 GTSRB Dataset

Figure 4.9: Overview of the classes represented in the GTSRB dataset (Stallkamp et al. 2011)

The German Traffic Sign Recognition Benchmark (GTSRB) dataset was originally created for a machine learning competition in 2011. The dataset contains 43 classes of common traffic signs, in a total of more than 50 000 images. The images are extracted from around 10 hours of video, recorded along German roads in 2010. The original video frames had a resolution of 1360 $\times$ 1024 pixels. The cropped out traffic signs end up with sizes ranging from approximately 200 $\times$ 200 pixels, to 15 $\times$ 15 pixels at the lowest. The majority of the images are however estimated to be in the 50 $\times$ 50 to 30 $\times$ 30 range. The dataset is split into a training set of around 39 000 images, and a test set of around 12 000 images. (Stallkamp et al. 2011)

Although very similar to Norwegian road signs, the signs in the GTSRB dataset have slight differences to some of their Norwegian counterparts. This means that in effect, the GTSRB dataset cannot effectively be used to recognize all 43 classes of Norwegian road signs. In spite of this, the dataset is still one of the most comprehensive in this field, and was deemed satisfactory for demonstrating the capabilities of the CNN model. In subsection 5.1.3 the possibility of producing dedicated training data from available data sources is discussed, and an alternative dataset is presented.

4.3.3 Object Classification Results

The classification model has not been tested on a large Norwegian dataset, due to the lack of annotated images of Norwegian traffic signs being available. The model performs sufficiently on traffic signs present in the training data set, however, due to the training dataset excluding numerous traffic signs found in Norway, several traffic signs are classified incorrectly. Examples of traffic signs the model classifies correctly with relative high precision are 'Yield', 'Priority road' and several speed limit signs, which are prevalent in the GTSRB training dataset. Traffic signs that are not present in the training set classes, have to be ignored when measuring the accuracy of the model.

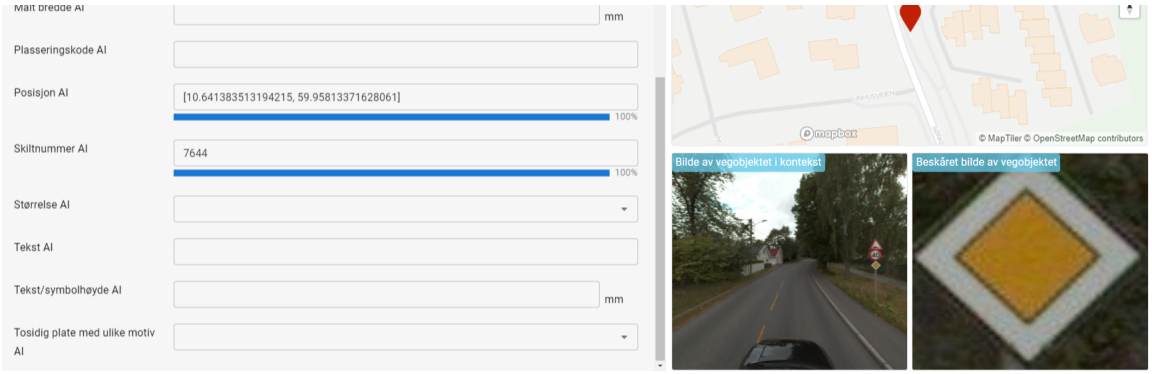


Figure 4.10: Example of correct traffic sign classification. Code 7644 corresponds to 'Priority Road' in the NVDB database

As mentioned previously, the model achieves an accuracy of approximately 96% on the GTSRB test set, which consists of about 12 000 images. This score gives the model some perceived credibility, but it is not an accurate description of the model's overall performance in the system. A balance needed to be struck between the training data available, and the data source to be used in the project. subsection 5.1.3 discusses what could be done to further improve the solution.

4.4 Geopositioning

The final step in producing a valuable digital twin of an object, is knowing it's exact location. This section explains how the location of each traffic sign is found through use of LIDAR generated point clouds.

4.4.1 Geopositioning Implementation

The pinhole camera projection method, involves creating a mathematical model of a camera, in relation to the real world. This camera model has intrinsic and extrinsic matrices, that express the camera's field-of-view, orientation, heading and location at the time of recording. A common use of a pinhole projection is in 3D graphics, where it is used to map 3D points to a 2D screen. The following section gives a view into how pinhole projections are used in this project. In subsection 5.1.4, further improvements on the solution is discussed. Theory on pinhole camera projection is included in section 3.4

The implementation of geolocation through point clouds, is done partially with contributions from the open source library Pylot. Pylot is a platform for testing self driving cars, self driving simulators, and for real world applications, like data collection through camera and LIDAR. (Gog et al. 2021)

The workflow of the geolocation service developed for this project, starts out by running a script, which creates a separate file containing the geographical center of each point cloud file. This is a crucial step, because the data has no innate connection between specific images and point cloud files. It is known which images and point clouds belong to a specific stretch of road, but this road can be several kilometers long. The aforementioned script, calculates the (x_{min}, x_{max}) and (y_{min}, y_{max}) to then find the middle x and y values of a point cloud, to determine which point cloud most likely contains information about a certain image. In edge cases this will likely run into quirks, but was deemed satisfactory for the time being. The output of this script is saved to a separate .csv-file dubbed 'midpoints.csv', which is used as a future reference at runtime.

This script could potentially be executed at runtime, but given the size of the point clouds, it is much more efficient to execute it once beforehand, only reading the data once into memory during a normal workflow.

After running the script once, the program has a record of all available point clouds, and their centre points. Detected traffic signs can then be sent to the geolocation service. The service starts by assigning a point cloud to each detected traffic sign object, by determining the shortest distance from the object's camera coordinates, and the point cloud centres.

It is now known which point clouds belong to each traffic sign, and which point clouds need to be accessed. To minimize read times, each point cloud is only read once, and each traffic sign object is processed while the given point cloud is loaded into memory. When a point cloud is loaded into memory, the points are filtered by intensity, both to filter out noise, but also to take advantage of the high reflectivity of traffic signs. For example, filtering a point cloud of 1.3 million points to only contain points of an intensity above an intensity value of 150, would return approximately 5000 points, which is a considerably smaller amount of data to handle.

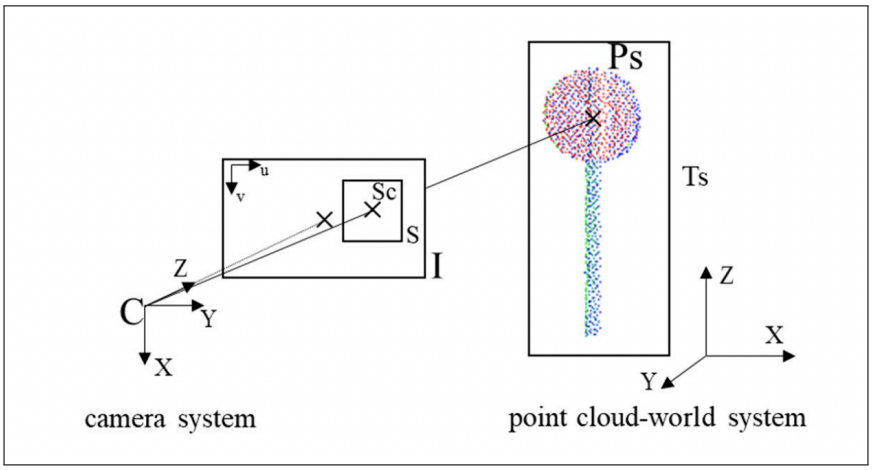


Figure 4.11: Illustration of pinhole projection towards traffic sign (Balado et al. 2020)

Now each single traffic sign-object is processed to determine the (x, y) coordinates of the detected object in an image. The coordinates are transformed from pixel coordinate space (u, v), into camera coordinate space, by using the intrinsic camera matrix. In the same way, the world coordinates are transformed to camera coordinate space using the extrinsic camera matrix. The next step is to process all points from the augmented world coordinates (now in camera coordinates) against the augmented pixel coordinates (also in camera coordinates) to determine the shortest euclidean distance. The point with the

shortest distance is deemed the best candidate, and is returned as the correct location of the traffic sign object.

Theoretically this process is quite straight forward, but there are some quirks to this method. Ideally, point clouds should be filtered beforehand, to manage noise and outliers, by for example clustering points. This is to avoid detecting reflective details on buildings or other objects in traffic, with similar high reflectivity as the desired objects. There is still great potential value in the dataset, that can be leveraged further, which will be discussed in subsection 5.1.4.

4.4.2 Handling Duplicate Detections

After all objects have been detected, classified and geolocated, all duplicate detections are removed. Since the MMS captures images every 5 meters, it is bound to record and detect the same traffic signs around 3 times or more. Every traffic sign object is compared to other detected objects, and if both classifications are the same, and the distance between their positions are below 5 meters, they are grouped together as suspected duplicates. When a group of two or more detections are marked as duplicates, the object that was recorded at the closest range to the car, is kept, while all others are discarded. This follows the assumption that geolocation of this object is likely the most accurate, and has the best resolution image, for the human operator doing quality assurance.

There are some caveats to this solution, such as identical traffic signs occurring close to each other, triggering a false duplicate detection. The threshold value is set so that detected objects at either side of the road still register as different road signs, like speed limit signs etc. In subsection 5.1.4 some more solutions that will benefit duplicate detection accuracy is discussed. Generally, as geolocating accuracy increases, the duplicate deletion stage becomes increasingly accurate.

4.4.3 Geopositioning Results

The figures below are taken from the parallel student project in Oslo, and show their visualization of the traffic sign geolocation determined by the AI, with the use of LIDAR data and pinhole projection, as described in subsection 4.4.1.

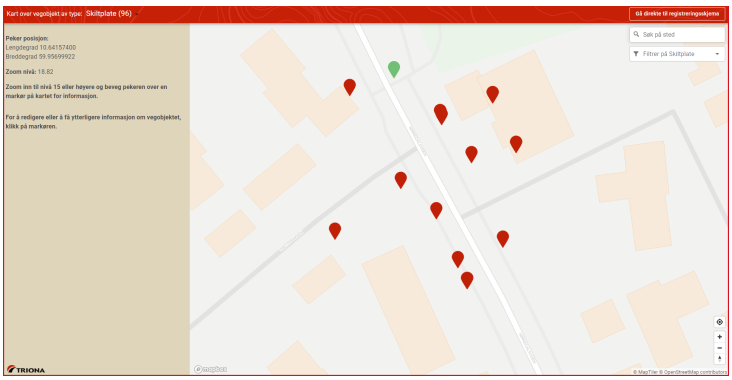


Figure 4.12: Example of geopositioning with varying degree of success (OsloMet student project, 2022)

Figure 4.13: Example of accurate geopositioning (OsloMet student project, 2022)

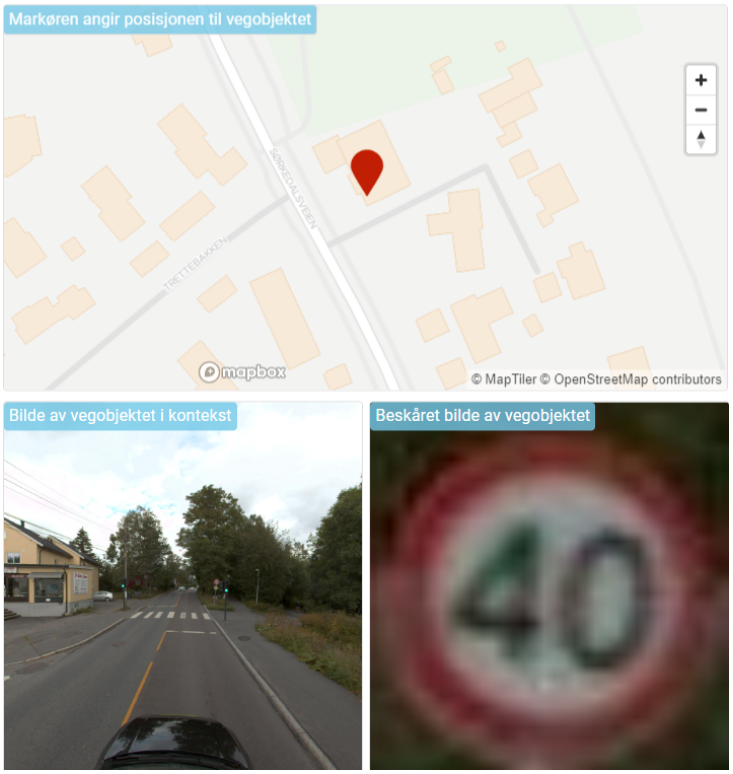


Figure 4.14: Example of inaccurate geopositioning (OsloMet student project, 2022)

The results show varying degrees of success in geopositioning of detected traffic signs. Some of the detected traffic signs have fairly accurate geolocations, located closely to the road, pavement or intersection, while others are located in more obscure locations, e.g. inside houses and forests. Overall, the traffic signs are located in the general area of the corresponding stretch of road, along with some edge cases.

4.5 Prototype Implementation

This chapter describes the system that has been developed, with an overview of the program flow. The architecture and the various components it consists of, together with the reasoning for the technologies that were chosen, is explained in Appendix A.

4.5.1 Program Flow

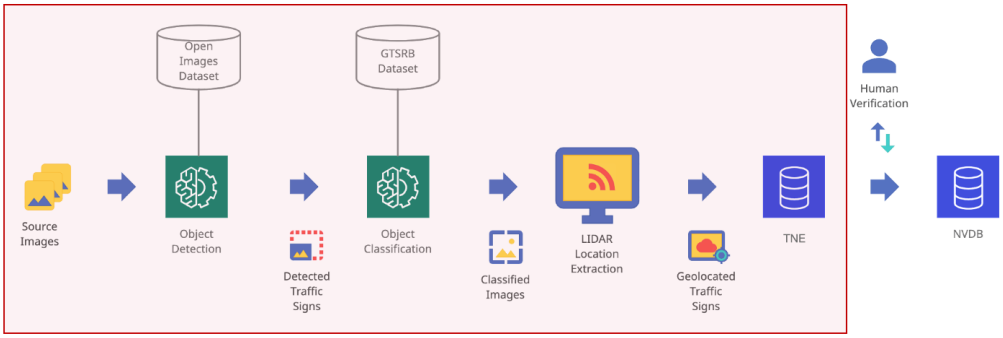


Figure 4.15: Illustration of the program flow

The system consists of a web application where the user can initiate inference on a set of data. This process will be automatically completed in the background, while giving status updates to the user for each stage. The list below shows the different stages during the inference process.

1. Detect objects of interest in the input data
2. Crop images of detected traffic signs
3. Classify the detected traffic signs
4. Locate the classified traffic signs in world coordinates
5. Remove duplicate classified traffic signs
6. Upload original and cropped traffic sign images to storage

Finally, a JSON-object containing data about all the detected traffic signs will be generated and returned to the user for inspection. The user can then store the objects in the TNE database, along with the original and cropped images associated with each object.

The figures below show the web application in different stages. The layout and CSS for the web application is minimalistic, following the Triona color scheme and integrating Triona's red topological lines in the background and header, to ensure cohesiveness with Triona's branding.

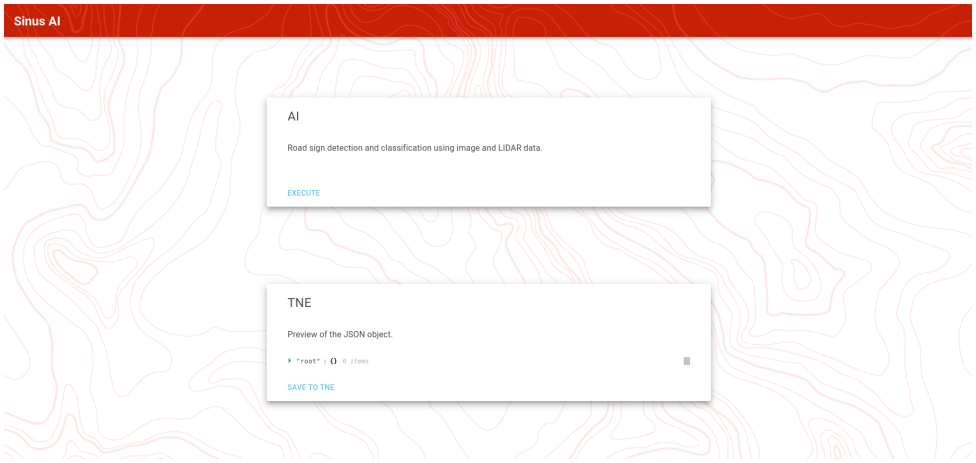


Figure 4.16: The web application

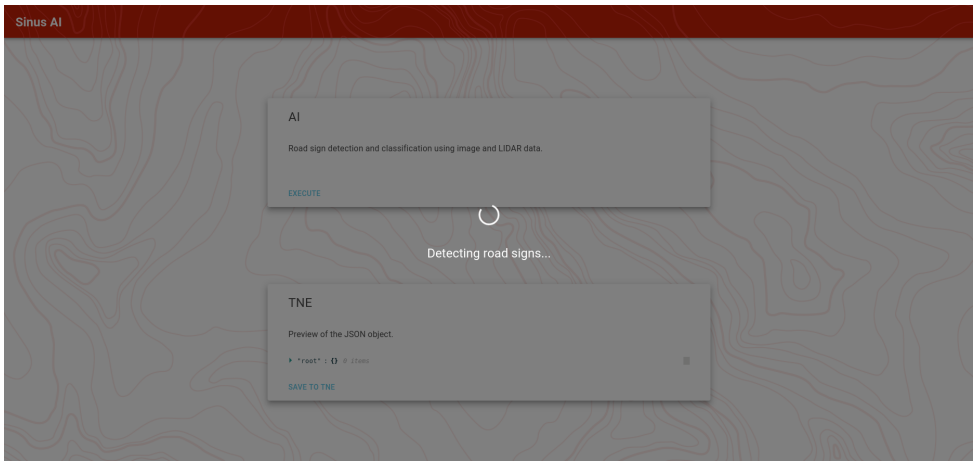


Figure 4.17: The web application with status messages in real-time

Figure 4.18: The web application with the AI generated JSON object

4.5.2 Integration with Parallel Student Project

The system the team has developed, has had a parallel focus on integration with the system developed by the student group in Oslo. The integration is done through the use of Triona's Transport Network Engine, where the AI generated JSON object is stored, and Azure Blob Storage, where both the original and cropped images are stored. A significant amount of time was spent on the configuration and setup of the connections between the system and these services.

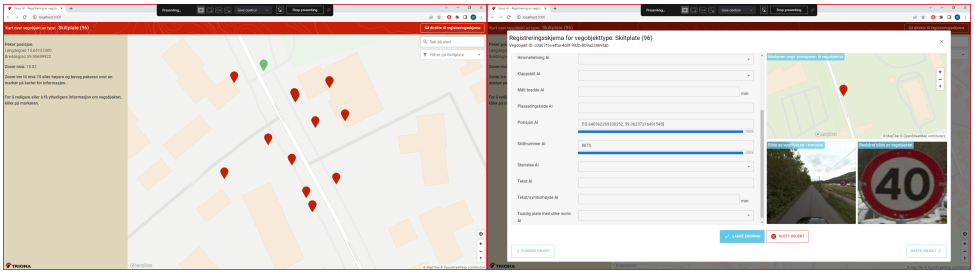


Figure 4.19: Example images of the solution created by the Oslo team (OsloMet student project, 2022)

Chapter 5

Discussion

In this chapter the results of the methods used for object detection and the development of a prototype are discussed. There will also be a discussion around limitations of the systems, as well as suggestions for possible future work and improvements.

5.1 Alternative Methods

Throughout the project, choices for which models, data sets and methods to use had to be made. Some reasoning for the choices made are given in chapter 4, but this section will provide further discussion around the alternatives that were looked at or considered.

5.1.1 Object Detection Models

Multiple machine learning models for object detection were considered, and several factors led to the decision of which one was the best option. One option was a YOLO implementation using the Darknet framework, which is a well known framework for object detection. A Faster R-CNN model was also tested and considered. In addition to having some of the same challenges with Darknet as mentioned above, running inference on images was considerably more time consuming that the YOLO model that was finally chosen.

Another option that was looked at was the possibility of finding a model that was completely pre-trained to detect traffic signs, but a good alternative for this was not found. Some experimentation was done with a YOLOv4 model that was pre-trained on the more general COCO dataset, but it was found that training a model from scratch on the more traffic sign specific Open Images dataset yielded better results.

During the pre-project, a lot of literature around the best performing state of the art one-stage and two-stage object detection models was studied, and while the best performers certainly were impressive, some of them require considerably more computing power during training and inference.

The final model was trained on Google's Colaboratory service, where it is possible to get limited access to GPU computing power. It was early on considered important to get access to GPU time on one of NTNU's HPC services, like Idun, but the decision was that the results from Google Colab was satisfactory for a proof of concept, and that time would be better spent focusing on other aspects of the thesis.

5.1.2 Classification Models

The model used for classification of the traffic signs was developed based on the teams previous experience with working with the GTSRB dataset. As shown in section 4.3, the model performs well on classification on the test data associated with the training set. Since the scope of the prototype only included traffic signs, it was decided that this style of CNN model was a decent alternative. The idea behind this is that no model would perform better with the given differences between data used for training and data from the project data source. A lot more effort can go into fine tuning a classification model, once a better suited training set is produced.

For future work it would be desirable to expand the solution to include detection of other types of roadside objects. Examples of objects of interest are noise barriers, traffic lights and fire hydrants, but also natural objects such as bushes or large stones that could affect accessibility and safety. Classification of these types of objects would likely require more complex machine learning models for both detection and classification, where the possibility of utilizing machine learning methods on LIDAR data for both detection and classification should be looked at.

5.1.3 Training Data

One of the most important issues to handle in the beginning of the project was how to acquire training data for the machine learning models.

For the classification model, unsuccessful efforts were made to find already annotated open source datasets of Norwegian traffic signs. Although the possibility of this existing cannot be completely excluded, it was decided that another solution was needed. The option of producing the data needed was also discussed, and Triona does have access to the necessary images for this. The images would still have to be manually cropped and annotated, and the amount of time and work this would require was considered not to be viable within the time frame of this thesis. The choice then fell on the open source GTSRB data set from Germany, where the design of most of the traffic signs closely resembles Norwegian traffic signs. The obvious weaknesses are that it does not include all types of traffic signs that are of interest, but tests showed that it was sufficient for training a model to correctly classify the ones that are represented int the training data.

An exciting piece of future work includes looking into meta-datasets such as 'The European Dataset' proposed by Citlalli Gámez Serna and Yassine Ruichek. It contains 164 different classes of traffic signs, recorded in six different European countries, containing around 80.000 total images. The dataset is composed of several smaller datasets, including GTSRB. The creators acknowledge the varying degrees of differences between traffic signs in these six countries, but rather work on the hypothesis that a classification model can become even sturdier from these differences. Their results are certainly promising, and unfortunately their research paper was discovered too late in the development stage of this project, to implement it. There is no guarantee that The European Dataset would drastically increase classification accuracy, but should still be considered in the future. (Serna and Ruichek 2018)

Acquiring training data for the object detection model runs into many of the same issues as for the classification model. Producing the data from scratch would involve manually marking out bounding boxes around objects of interest in images and annotating each bounding box, and this option was quickly discarded because of the amount of work it would require. After researching possible open source datasets, two that were found

stood out as the best options. A subset of the Google Open Images dataset was chosen for this thesis, but another option that was considered was the German Traffic Sign Detection Benchmark (GTSDB) dataset. This consists of 900 images containing 1206 traffic signs that are annotated within three categories. Even though it was only desirable to have a single category when detecting signs for this project, this is something that could easily be handled in code, and the decision was down to which of the two would be deemed most suited for correctly detecting as many signs as possible. An important factor here is that it is more desirable to have false positives in the detection stage than false negatives. GTSDB has a narrower definition of what a traffic sign is than Open Images, and mainly consists of some of the most common German traffic signs, while Open Images consists of a wide array of signs from a number of different countries. It is therefore believed that the latter option provides a better chance of training the model to detect a larger percentage of objects of interest. (Houben et al. 2013)

5.1.4 Utilization of LIDAR Data and Alternatives for Geopositioning

Making use of the available LIDAR data in some fashion when developing the system was part of the original project description given by Triona. LIDAR data has at the time of writing only been gathered in the Oslo area, in connection with an ongoing project updating information about the road network in this part of the country. It is desirable to explore the possibilities that lies in having access to this type of data, and the results can be helpful when deciding whether or not to expand the data gathering to other parts of the country.

An early thought was that if it was possible to use machine learning to detect objects with LIDAR, the geolocation of the detected objects would automatically be known, which is an argument for this being the best solution. To train a machine learning model to detect traffic signs using LIDAR, it would again be necessary to have access to suitable training data. Some effort was made to find data that could be used, but this was not successful, and this idea unfortunately falls into the possible future work category.

The other option was to use the LIDAR data to find the geolocation of the objects detected by the other machine learning models, and this has been the single part of the project where the most time and effort has been spent. The implemented solution is explained in section 4.4, and some theory about the topic is presented in section 3.4. There is still some exciting future work on the current method used to geolocate objects. This includes performing clustering on the point clouds, to identify which points (statistically) belong together, filtering the point clouds using ROI's, identifying which direction traffic signs are facing, and lastly identifying the height between the traffic sign and the ground. Some of these points have been proven to be possible in papers such as 'Novel Approach to Automatic Traffic Sign Inventory Based on Mobile Mapping System Data and Deep Learning' by Balado et al. 2020. Clustering of the point clouds could possibly be done both by traditional statistical methods, or by utilizing machine learning. The latter being potentially most accurate, but computationally expensive, and the former being dependable, but likely limited in accuracy. If accurate clustering is achieved, one can also extract even more information about the objects, such as size and certain types of damage.

Because of the uncertainty around the results geopositioning using the LIDAR data would yield, it was desirable to explore other solutions that could potentially serve as an alternative. The most realistic option here was to perform triangulation using the available image data. In short terms, if the position of the camera is known, and an object is

present in two or more images, it could be possible to approximate the position of the object. Triona already uses this technique in manual registration of objects, so the challenge would be to develop an automation of the process. This is also a solution that would be desirable to have available in parallel to the LIDAR solution, at least until LIDAR data is available for all areas of interest. An article review of some previous work in this field is included in section 3.1.

5.1.5 Proof of Concept System

The application developed in conjunction with this thesis was always intended to serve as as prototype, or proof of concept. Because of this, some aspects of system development methodology has not been followed thoroughly as they would had the intention been to deliver a finished product, or a system that was intended for direct further development. The system shows that the connection with Trionas existing services, and the parallel student project in Oslo has demonstrated that the data can be used. A system has not been set up for a regular operator to continue training the existing models, but this was not to considered to be within the scope of the thesis, and is likely something that also in the future will have to be done by engineers.

There are some limitations to the system that has been developed. Most notable is the inability of an operator to choose the stretch of road or set of images to run inference on. In the current version of the system there is a presupposition that the images are already located on the server where the system is being hosted, but for further development it would be desirable to let the operator upload data, or choose what data to run inference on.

During the entire development period, weekly update meetings have been held with the Oslo team, to discuss solutions, solve issues and update each other on progress. The meetings have been mutually beneficial to both teams and the two projects working towards a common goal has been an overall success. Without their user interface, there would not have been as good demonstrations of the data output from this project. See figure 4.19 for screen captures of their solution in practice.

Chapter 6

Conclusion

The main research goal for this project was to propose methods for utilizing machine learning to detect and classify roadside object in images captured by a MMS, and a method for approximating the geolocation of these objects using corresponding LIDAR data. These methods were then to be implemented into a proof of concept administration tool. In this chapter we give our conclusion on our findings regarding the proposed methods for object detection, classification and geopositioning and the viability of the product.

6.1 Machine Learning Models

The precision of the machine learning models presented in this thesis it not high enough for a product intended for commercial use, but the results shows that this in large part comes down to the available training data. We also know that access to more powerful hardware for model training would help increase the precision. Our conclusion is that the methods implemented in the prototype, or some of the alternative methods presented, would be possible to implement on a larger scale. We want to stress that this would have to involve directing resources to some key areas, these being acquiring specialized training data for training both object detection and classification models, and investing in GPU power to perform model training.

6.2 Method for Geolocation

The results from geopositioning presented in this thesis shows that this is a method that is possible to use, but we have not been able to reach the desired precision. This could likely be improved by adjustments to the interval of light intensity that is searched for, and more work on the method for deleting duplicates. Improvements in calculating a line of bearing to the bounding box marking a detected object could also yield more precise results. We are wary of making a definitive conclusion on what what the best solution for geopositioning is, and all alternatives should be further explored before making a final decision. It is possible that performing both object detection and geopositioning using LIDAR data could yield more precise results, but a method similar to the one proposed by Li et al. 2022, which is summarised in subsection 3.1.2, should also be considered.

6.3 Administration Tool Prototype

The system developed is meant to function as a proof of concept, and a system developed for commercial use would need significant further development, but the prototype presented in this thesis could be a good basis to work from. The question then becomes if we have been able to show that further development of the proposed solution could be viable commercially. We believe we have shown that it is possible to automate the detection of objects in a way that has great potential to increase efficiency, and that the process can be administered by the same human operators who does manual registration today, and who does not have a technical or engineering background.

6.4 Viability and Threats to Validity

We do not have access to large enough annotated relevant datasets to give machine learning methods and services precise accuracy measurements for the environment the system is meant to function in. There has also not been performed extensive enough testing to reveal potential edge cases and flaws in the system, that would reveal themselves in production. We believe that the system shows promise, but it is naturally not ready for production at this stage.

Another threat to the validity of the results is potential faults in the datasets used for training the machine learning models. The datasets consists of a large amount of images, and performing quality assurance to ensure they only contain objects that are of interest has not been possible. We have for instance seen that the data used for training the object detection model contains images of traffic lights, which has led to some false positives when running object detection inference. At the same time it has been heavily emphasised that the training data used is sufficient for a proof of concept, but specialized data would have to be obtained for a production-ready system.

Broader Impact

In this project, a potential for new innovation is presented, along with instructions to further improve the solution. This innovation can potentially save public spending, and relieve humans from tedious and repetitive tasks. This relief must however not come in the form of unemployment. Technological innovation in first world countries must strive to uplift and include, rather than undermine and exclude. In particular when proposing a system that replaces humans in the workforce, one should be reflective on what these people can instead perform.

The dataset central for this thesis, could potentially be enriched by adding annotations. This could in turn open it for further research and aiding in automation tasks, if made public. This is also a potential avenue of innovation for partners Triona. Public spending on infrastructure maintenance can likely be optimized by automation, and this work aims to aid in this work.

It is vital to consider the potential consequences and impact when working in a field such as artificial intelligence. Any and all works undertaken by engineers should be thoughtfully reviewed to see one's role in a larger context. In the twenty first century, the field of artificial intelligence is booming, and the ethical considerations can't always keep up with innovation. Engineers are equally as responsible in considering the ethical side of their work, as their employers.

Bibliography

Balado, Jesús et al. (2020). "Novel Approach to Automatic Traffic Sign Inventory Based on Mobile Mapping System Data and Deep Learning". In: *Remote Sensing* 12.3. ISSN: 2072-4292. DOI: 10.3390/rs12030442. URL: https://www.mdpi.com/2072-4292/12/3/442.

Docker (2021). *Docker overview*. Last accessed 09.05.2022. URL: https://docs.docker.com/get-started/overview/.

Education, IBM Cloud (2021). *Docker*. Last accessed 09.05.2022. URL: https://www.ibm.com/cloud/learn/docker.

Gog, Ionel et al. (Apr. 2021). *Pylot: A Modular Platform for Exploring Latency-Accuracy Tradeoffs in Autonomous Vehicles*.

Houben, Sebastian et al. (2013). "Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark". In: *International Joint Conference on Neural Networks*. 1288.

Inc., Gradle (2021). *What is Gradle?* Last accessed 10.05.2022. URL: https://docs.gradle.org/current/userguide/what_is_gradle.html.

Inc., VMware (2022). *RabbitMQ Features*. Last accessed 10.05.2022. URL: https://www.rabbitmq.com/.

Jocher, Glenn et al. (Feb. 2022). *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*. Version v6.1. DOI: 10.5281/zenodo.6222936. URL: https://doi.org/10.5281/zenodo.6222936.

Ko, Hyunsuk et al. (Mar. 2016). "Robust Uncalibrated Stereo Rectification with Constrained Geometric Distortions (USR-CGD)". In: *Image and Vision Computing*. DOI: 10.1016/j.imavis.2017.01.001.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Kuznetsova, Alina et al. (Mar. 2020). "The Open Images Dataset V4". In: *International Journal of Computer Vision* 128.7, pp. 1956–1981. ISSN: 1573-1405. DOI: 10.1007/s11263-020-01316-z. URL: http://dx.doi.org/10.1007/s11263-020-01316-z.

Kwon, Yongchan et al. (2020). "Uncertainty quantification using Bayesian neural networks in classification: Application to biomedical image segmentation". In: *Computational Statistics Data Analysis* 142, p. 106816. ISSN: 0167-9473. DOI: https://doi.org/10.1016/j.csda.2019.106816. URL: https://www.sciencedirect.com/science/article/pii/S016794731930163X.

Li, Guannan et al. (2022). "Automatic Positioning of Street Objects Based on Self-Adaptive Constrained Line of Bearing from Street-View Images". eng. In: *ISPRS international journal of geo-information* 11.4, p. 253. ISSN: 2220-9964.

McManamon, P.F. (2019). *LiDAR Technologies and Systems*. Press Monographs. SPIE Press. ISBN: 9781510625396. URL: https://books.google.no/books?id=cIIIwAEACAAJ.

Meta Platforms, Inc. (2022). *React Docs: Getting Started*. Last accessed 10.05.2022. URL: https://reactjs.org/docs/getting-started.html.

Microsoft (2022). *TypeScript for JavaScript Programmers*. Last accessed 09.05.2022. URL: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.

Pallets (2010). *Flask*. Last accessed 10.05.2022. URL: https://flask.palletsprojects.com/en/2.1.x/.

Redmon, Joseph (2013–2016). *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/.

Redmon, Joseph et al. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. arXiv: 1506.02640 [cs.CV].

Research, Google (2021). *Colaboratory*. URL: https://research.google.com/colaboratory/faq.html (visited on Dec. 10, 2021).

Serna, Citlalli Gámez and Yassine Ruichek (2018). "Classification of Traffic Signs: The European Dataset". In: *IEEE Access*.

Solem, Ask (2021). *Introduction to Celery*. Last accessed 10.05.2022. URL: https://docs.celeryq.dev/en/stable/getting-started/introduction.html.

Stallkamp, Johannes et al. (2011). "The German Traffic Sign Recognition Benchmark: A multi-class classification competition". In: *The 2011 International Joint Conference on Neural Networks*, pp. 1453–1460. DOI: 10.1109/IJCNN.2011.6033395.

STOMP (2012). *STOMP Protocol Spesification, Version 1.2*. Last accessed 09.05.2022. URL: https://stomp.github.io/stomp-specification-1.2.html.

Tutorialspoint (2022). *Spring Boot Introduction*. Last accessed 10.05.2022. URL: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm.

Appendix A

System Documentation

Contents

List of Figures

1 Introduction

This document is a documentation of the overall system that has been developed for the bachelor's thesis, and acts as a supplement to the main report. It describes in detail the system architecture, project structure, diagrams, server endpoints, installation and testing of the system.

2 Architecture

This chapter describes the overall system architecture, as well as the team's reasoning for the different technologies used in the system. The technologies are further described in the main report. The architecture is built on the foundation of multiple Docker containers, where each container is a separate service to the overall system. The figure below shows a diagram of the system architecture, and how the different containers communicate.

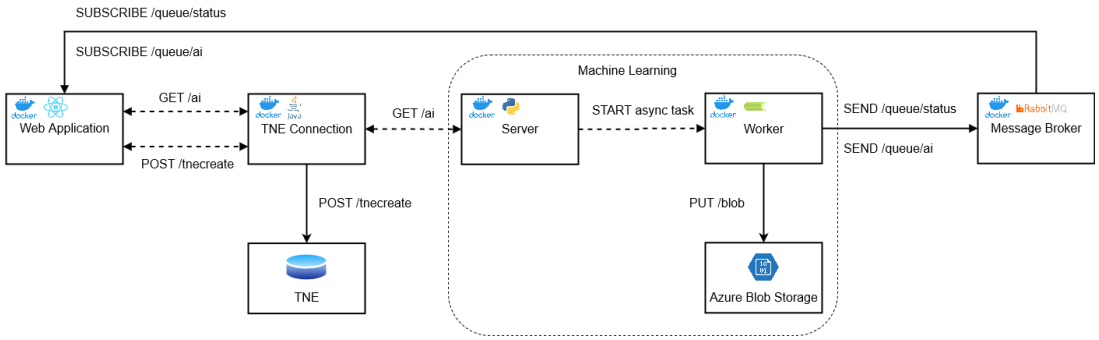


Figure 1: The system architecture

The architecture is structured this way to separate different logic components in separate environments, ensuring increased isolation and security. This also allows each environment to run independently from one another. During development of the system, this architecture structure allows the team to both run and debug the system, independently from the OS used on the host machine. The use of Docker containers allows for future automated deployment, scaling and management in orchestration software, e.g. Kubernetes.

2.1 Communication

The Docker containers communicate via an internal Docker network. The communication is done through the HTTP and STOMP protocols.

2.1.1 HTTP

To start the inference process, the following requests are sent.

1. GET request from the Web Application to the TNE Connection
2. GET request from the TNE Connection to the Machine Learning

To store the AI generated JSON object in TNE, the following requests are sent.

1. POST request from the Web Application to the TNE Connection
2. POST request from the TNE Connection to TNE

To upload images to Azure Blob Storage, the following request is sent for each image.

- PUT request from the Worker to Azure Blob Storage

2.1.2 STOMP

To send status messages and the AI generated JSON object from the Machine Learning to the Message Broker, the following requests are sent.

- SEND frame from the Worker to the queue: */queue/status*
- SEND frame from the Worker to the queue: */queue/ai*

To get the status messages and AI generated JSON object from the Message Broker to the Web Application, the following requests are sent.

- SUBSCRIBE frame from the Web Application to the queue: */queue/status*
- SUBSCRIBE frame from the Web Application to the queue: */queue/ai*

2.2 Web Application

The web application serves as the administration page, where an operator can run inference on a set of images. The application is a Single Page Application (SPA) written in React and TypeScript (TS). React allows for rewriting and updating content on the web page without refreshing the page, which makes it optimal for a SPA. The addition of a Virtual DOM, unidirectional data flow, and extensive library, are also important features. TypeScript was recommended by Triona, as it allows for static type checking, minimizing potential bugs. Features like type hinting, code completion, auto import, and overall higher code quality and confidence are also important factors. However, there are some drawbacks to using TS. Some library packages do not support TS, and lack of prior TS knowledge lead to slightly inefficient functional and component style coding while developing the application. The majority of these drawbacks were fixed during the Quality Assurance (QA) stage in the development pipeline, with help from Triona supervisors.

2.3 TNE Connection

Triona has a relational SQL database, called Transport Engine Network (TNE), where roadside objects are stored. The TNE Connection component of the system is the bridge between the web application and TNE, as well as starting the inference process in the machine learning component. The final JSON object created by the machine learning component is stored in TNE, which is why this connection is needed. The system component is written in Java Spring Boot, with Gradle as a build tool. Gradle was given as a system specification by Triona, as this is part of the technology stack that is used in their future projects. Spring Boot was chosen due to the need for a RESTful API for communication between the web application, TNE and the machine learning component of the system. The team also has prior development experience and knowledge with Spring Boot.

2.4 Machine Learning

The Machine Learning component consists of two sub-components, a server and a worker. The server is a Python Flask server, that starts the process of running inference on a set of images corresponding to a stretch of road. This inference task is processed by a Celery worker, running asynchronously in the background. After each stage in the task, the worker will send a status message to the message broker component, where it will be stored in a queue. The queue will be updated as new status messages are sent. The different stages in the inference task are described in the main report. Once the task is

finished, the machine learning component will construct a JSON object with the detected traffic signs, including the relevant meta-data from classification and geolocation. The JSON object is then sent to another queue in the message broker.

Flask was chosen as a foundation for this component because it is lightweight, and supports unit testing and RESTful APIs. The Flask server is minimalistic, only consisting of a single endpoint, which starts the inference task inside the Celery worker. The Celery worker is a separate Docker container from the Python/Flask container. Celery was chosen for its ability to create task queues, and run tasks asynchronously in the background, as well as being more scalable and having support from different message brokers. A more primitive alternative to Celery was tested which involved using a custom thread and the Flask socket extension, SocketIO, to both run the task in the background and send status messages in real-time. However, this proved to be difficult to integrate with either a message broker or directly with the web application, due to networking issues between Docker containers and SocketIO sockets, resulting in the favor of Celery in combination with a message broker.

2.5 Message Broker

The message broker is a RabbitMQ message broker, which instantiates two queues, one for the status messages, and one for the AI generated JSON object. Once the queues are updated by the Celery worker, the message broker publishes the messages to the web application in real-time, which subscribes to the queues. RabbitMQ was chosen for its support for asynchronous messaging across applications, integration with Celery, and sending messages through the STOMP protocol. The benefit of using a message broker, to a standard socket, is that the different applications or services that use the message broker does not need to know who they are communicating with. The applications only need to know information about the queue(s). A message broker also ensures that the messages are not lost, as it is based on the principal of ”guaranteed delivery”. (Education 2020)

3 Project Structure

The project structure consists of three main folders, of for each of the core components of the system, e.g. Web Application (frontend), TNE Connection (java), and Machine Learning (python). There are two additional folders, ’data’ and ’scripts’. The ’data’ folder is used as a Docker volume, for persisting and sharing data generated and used by the Docker containers. The ’scripts’ folder includes useful scripts, mainly used during development, which includes running tests and generating midpoints in a point cloud as a .csv file. The figure below shows the project structure from the root folder in Azure Devops.

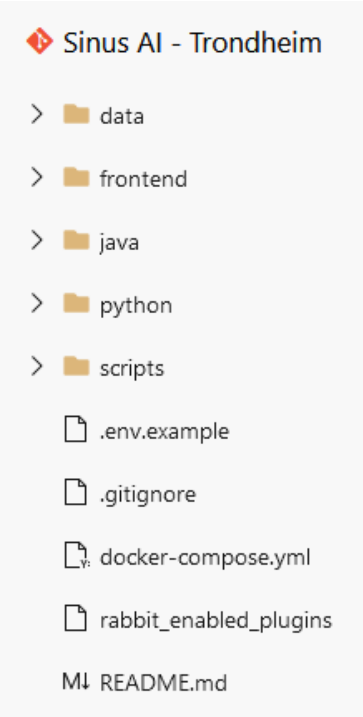


Figure 2: The project structure from the root folder

4 Diagrams

This chapter gives an overview of the different types of diagrams created to give more technical insight to the reader. The diagrams consists of class diagrams and a sequence diagram.

4.1 Class Diagram

4.1.1 TNE Connection

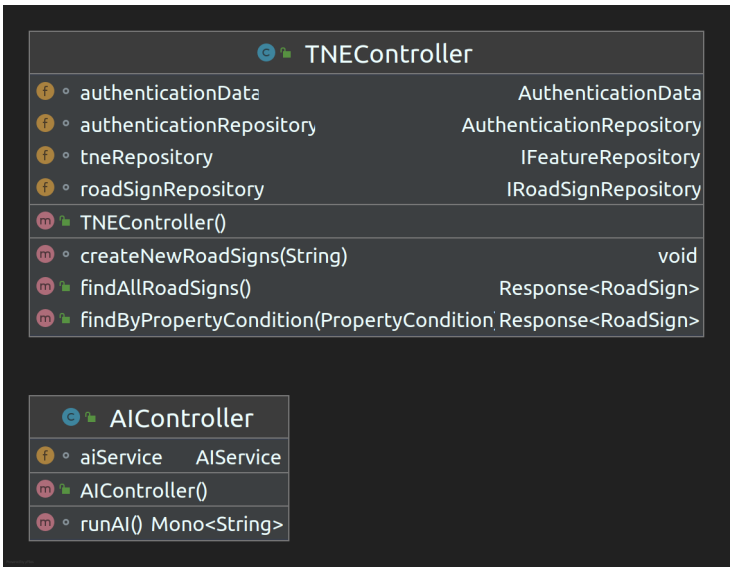


Figure 3: Class diagram of controller classes

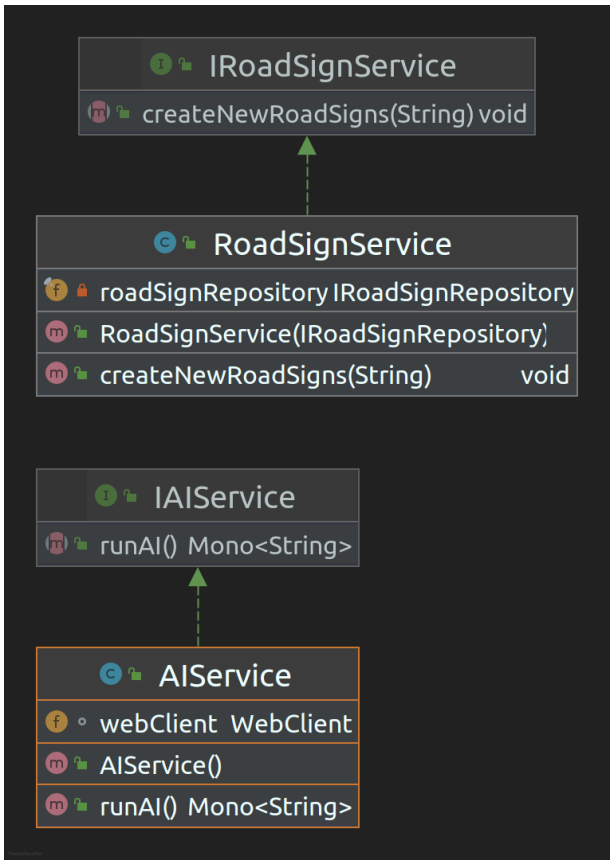


Figure 4: Class diagram of service classes

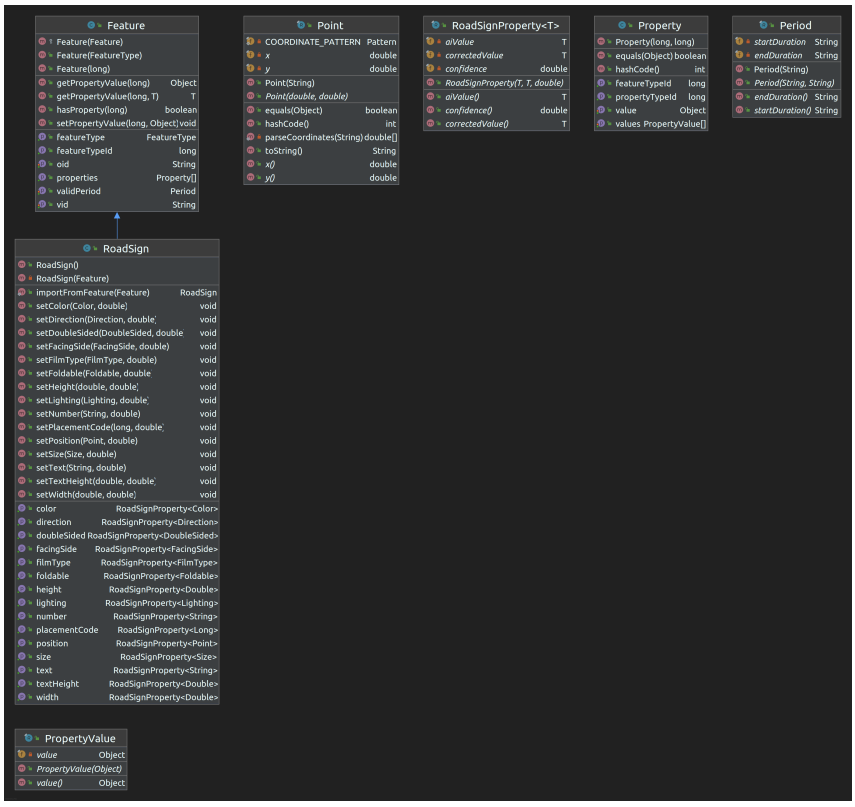


Figure 5: Class diagram of domain classes

Figure 6: Class diagram of enum classes

Figure 7: Class diagram of persistence classes

4.1.2 Machine Learning

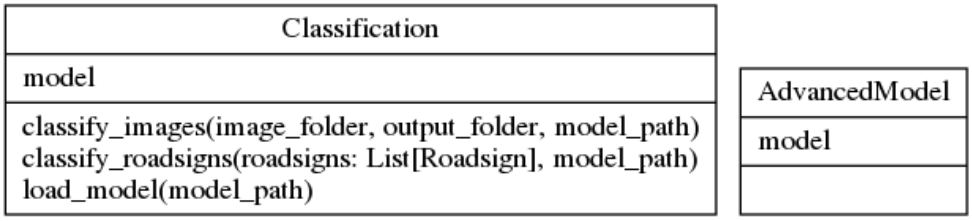


Figure 8: Class diagram of classification utility classes

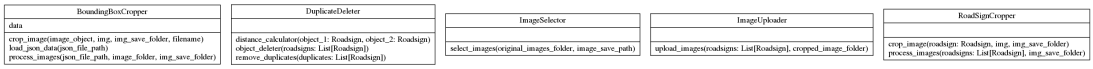


Figure 9: Class diagram of image processing classes

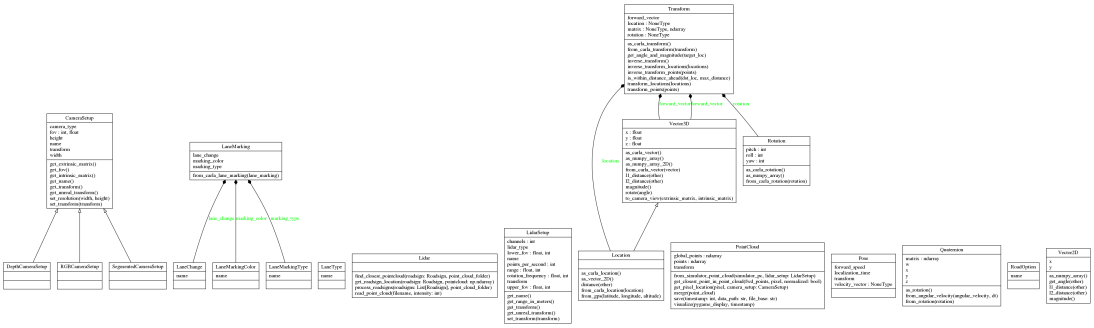


Figure 10: Class diagram of LIDAR classes from Pylot

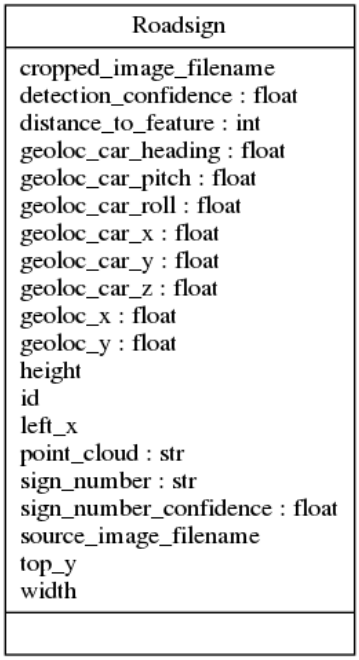


Figure 11: Class diagram of the road sign class

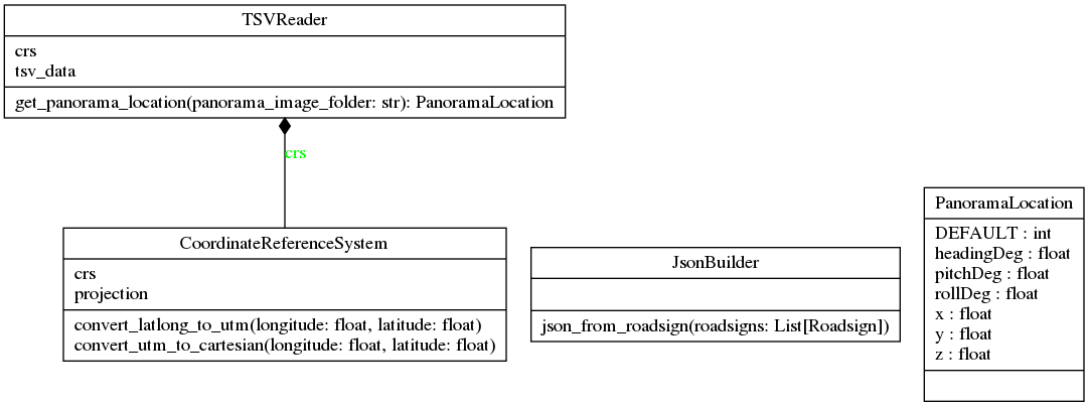


Figure 12: Class diagram of utility classes

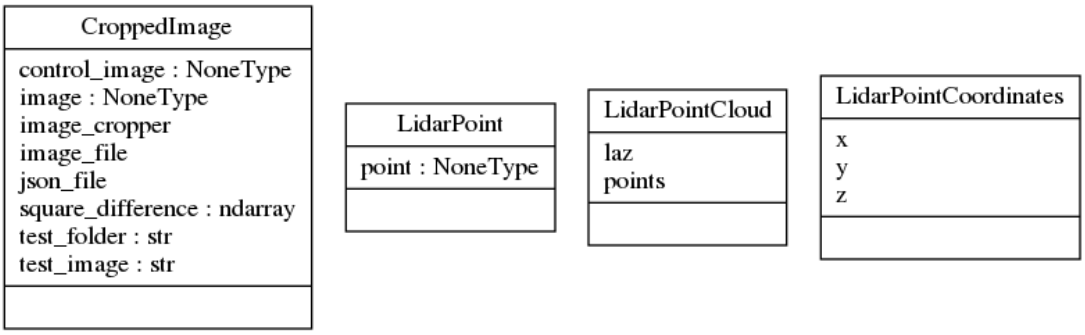


Figure 13: Class diagram of test classes

4.2 Sequence Diagram

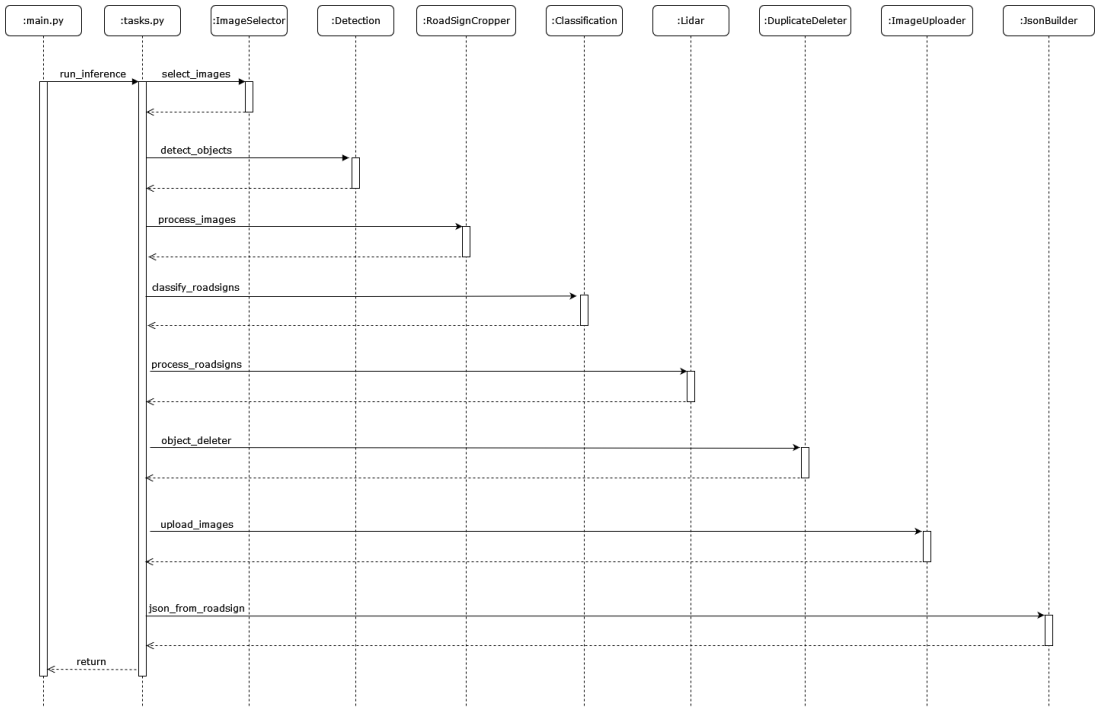


Figure 14: Sequence diagram of the program flow inside the Machine Learning component

5 Server endpoints

This chapter describes the endpoints in the different server services in the system. The system consists of a REST-server, a WebSocket in the web application, and a Socket in the RabbitMQ message broker. An overview of the endpoints in the system will be described in this chapter. Due to formatting reasons for this document, some of the properties in the JSON objects that are used as examples, have been omitted.

5.1 REST-server

get **/ai**
Start the AI inference task

Parameter

No parameter

Response application/json

200 OK

get **/tnefindall**
Get all entries in the TNE catalogue

Parameter

No parameter

Response application/json

200 OK

```
{
"success": true,
"message": "Retrieved 100 features.",
"features": [
        {
        featureTypeId: 96,
        featureType: "ROAD_SIGN",
        oid: "29efa913-6687-4b0f-a6b8-9bfbf33f95b2",
        ...
        },
        ...
    ]
}
```

post **/tnecreate**

Create a new entry in the TNE catalogue

Parameter

No parameter

Body application/json


```
{
    "Features": [
    {
      "FeatureTypeId": 96,
      "Oid": "2b58abc9-c126-45da-a2aa-860d3709935a",
      "Properties": [
        {
          "AttributeTypeId": 1,
          "Values": [
            {
              "Value": "8877"
            }
          ]
        },
        ...
        {
          "AttributeTypeId": 43,
          "Values": [
            {
              "Value": "POINT (255799.597 6657643.43)"
            }
          ]
        },
        ...
      ]
    },
    ...
    ]
}
```


Response application/json

200 OK

5.2 WebSocket

sub-scribe **/queue/ai**

Get the AI object from the queue

Parameter

No parameter

Response application/json

200 OK


```
{
  "Features": [
    {
      "FeatureTypeId": 96,
      "Oid": "2b58abc9-c126-45da-a2aa-860d3709935a",
      "Properties": [
        {
          "AttributeTypeId": 1,
          "Values": [
            {
              "Value": "8877"
            }
          ]
        },
        ...
        {
          "AttributeTypeId": 43,
          "Values": [
            {
              "Value": "POINT (255799.597 6657643.43)"
            }
          ]
        },
        ...
      ]
    },
    ...
  ]
}
```


sub-scribe **/queue/status**

Get the status message from the queue

Parameter

No parameter

Response text/

200 OK

5.3 Socket

send **/queue/ai**

Send the AI object to the queue

Parameter

No parameter

Body application/json

```
{
  "Features": [
    {
      "FeatureTypeId": 96,
      "Oid": "2b58abc9-c126-45da-a2aa-860d3709935a",
      "Properties": [
        {
          "AttributeTypeId": 1,
          "Values": [
            {
              "Value": "8877"
            }
          ]
        },
        ...
        {
          "AttributeTypeId": 43,
          "Values": [
            {
              "Value": "POINT (255799.597 6657643.43)"
            }
          ]
        },
        ...
      ]
    },
    ...
  ]
}
```

Response application/json

200 OK

send	**/queue/status** *Send the status message to the queue*
Parameter	
No parameter	
Response	text/
200 OK	

6 Installation

This chapter serves as an overview of how to get a local instance of the system installed and how to execute the system.

6.1 Requirements

The listing below shows the requirements needed for the installation and development of the system.

- Docker
- Docker Compose
- Gradle 7.3
- JDK 17
- Node.js
- npm
- Python >= 3.6

6.2 Installation Guide

Create a .env file following the .env.example file, in the root folder of the project structure.

Inside of the root folder, change directory to the 'frontend' folder:

```
$ cd frontend
```

Install all npm packages:

```
$ npm install
```

6.3 Execution

Inside of the root folder in the project structure, execute the following command to run the system:

```
$ docker-compose up --build
```

7 Testing

Testing of the system was heavily emphasised early on in the development process. However, later in the process, this focus on testing had to be altered, as the team had to prioritize completing the implementation of critical functionality to the system, as well as integrating all of the separate components together into a cohesive unit. The tests that were made are sufficient for a proof of concept, given that they cover the most critical aspects and functionality of the system. However, if this system were to be further developed into a commercially viable system, additional and more extensive testing is required. This chapter will describe the tests that were made, and how to run them.

The tests that were written can be divided into two components, one for the TNE Connection component and one for the Machine Learning component.

7.1 TNE Connection

The tests for the TNE Connection component were almost exclusively written by one of the Triona supervisors, and focuses on testing the connection and sending HTTP requests to TNE. There are a total of 41 tests, both positive testing and negative testing, ranging from unit tests to integration tests. The tests covers authentication to TNE, finding and creating entries in TNE based on different properties, and finding all entries in TNE.

7.2 Machine Learning

The tests for the Machine Learning component were written by the team. There are 24 tests in total, with both positive and negative testing, covering image cropping, reading and manipulating LIDAR data, object detection and classification.

7.3 Execution

7.3.1 TNE Connection

To run the tests for the TNE Connection component, execute the test Gradle task. This can be done either from an IDE, e.g. IntelliJ IDEA, or from the command line.

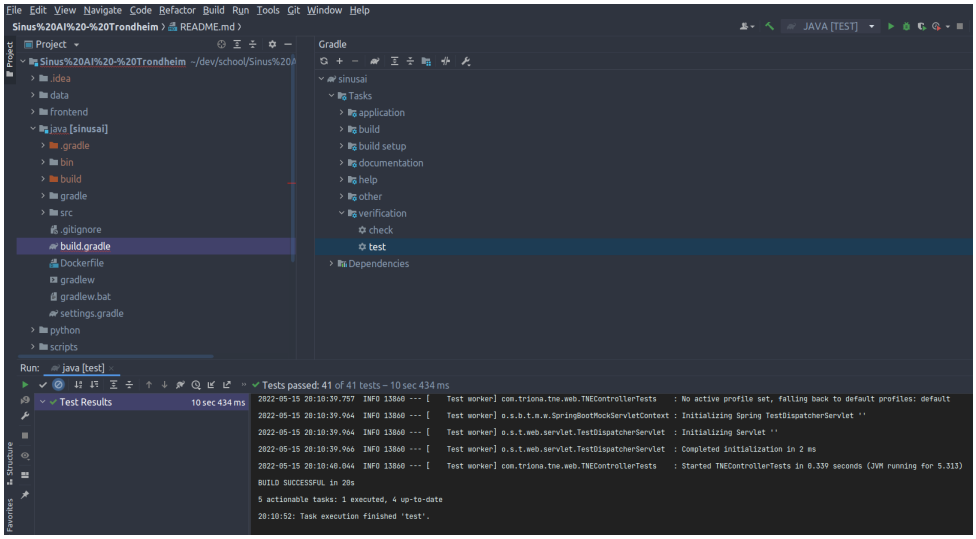


Figure 15: Test summary of the TNE Connection component, from IntelliJ IDEA

Inside the 'java' folder of the project structure, execute the following command:

```
$ ./gradlew test
```

To run the test coverage for the TNE Connection component, execute the jacocoTestReport task, either from an IDE, or from the command line inside the 'java' folder:

```
$ ./gradlew jacocoTestReport
```

The test coverage result will be located from the root folder of the project structure at **/java/build/reports/jacoco/test/html/index.html**

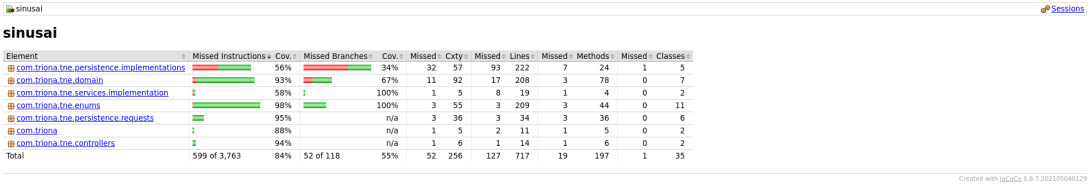


Figure 16: Test coverage summary of the TNE Connection component

7.3.2 Machine Learning

To run the tests for the Machine Learning component, execute the following command from the root folder of the project structure:

```
$ ./scripts/test.sh
```

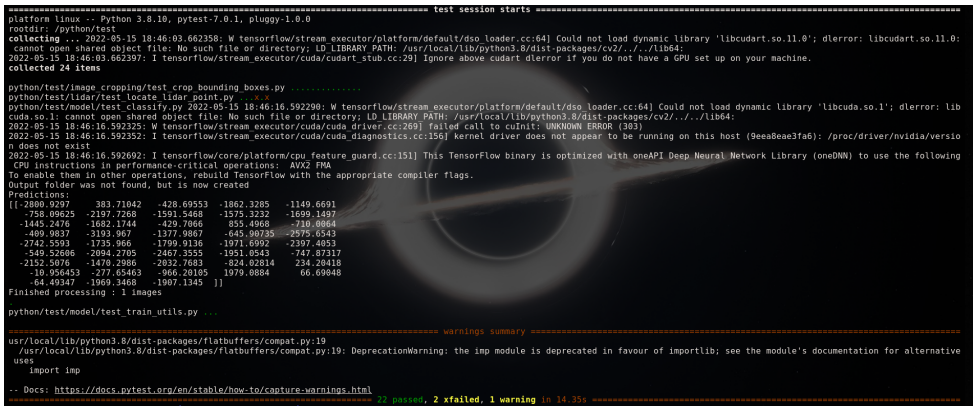
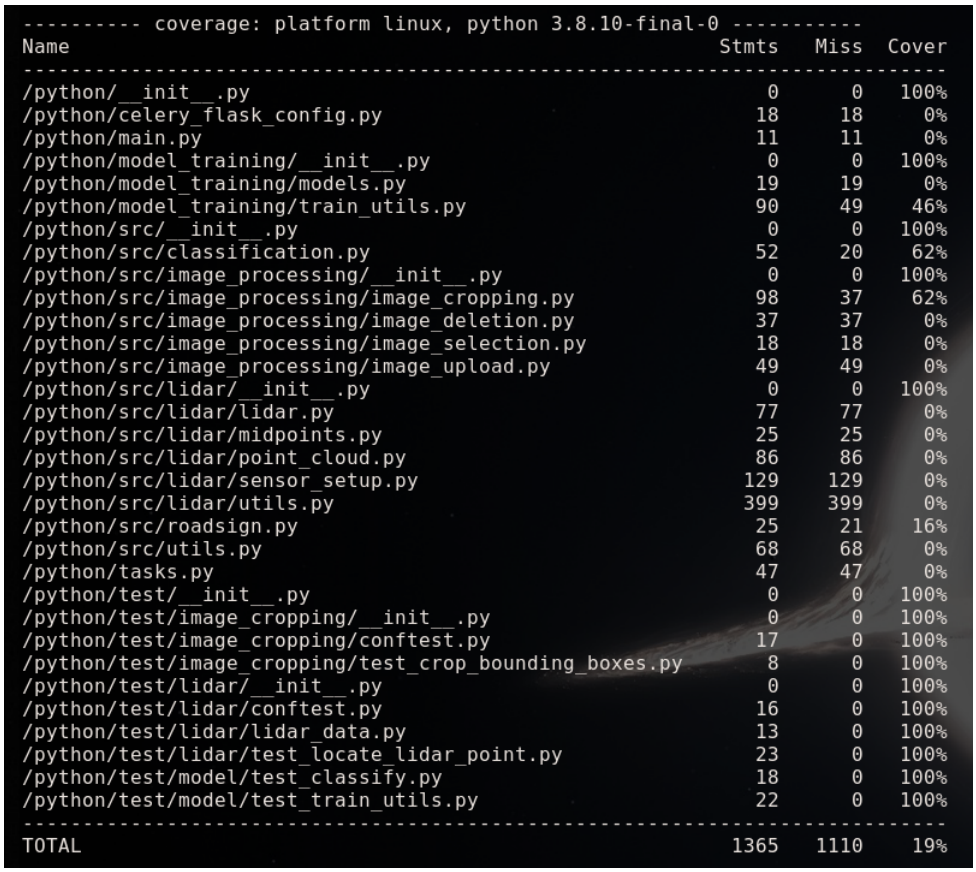


Figure 17: Test summary of the Machine Learning component

To run the test coverage for the Machine Learning component, execute the following command from the root folder of the project structure:

```
$ ./scripts/coverage.sh
```



```
---------- coverage: platform linux, python 3.8.10-final-0 -----------
Name                                                                 Stmts   Miss  Cover
----------------------------------------------------------------------------------------
/python/__init__.py                                                      0      0   100%
/python/celery_flask_config.py                                          18     18     0%
/python/main.py                                                         11     11     0%
/python/model_training/__init__.py                                       0      0   100%
/python/model_training/models.py                                        19     19     0%
/python/model_training/train_utils.py                                   90     49    46%
/python/src/__init__.py                                                  0      0   100%
/python/src/classification.py                                           52     20    62%
/python/src/image_processing/__init__.py                                 0      0   100%
/python/src/image_processing/image_cropping.py                          98     37    62%
/python/src/image_processing/image_deletion.py                          37     37     0%
/python/src/image_processing/image_selection.py                         18     18     0%
/python/src/image_processing/image_upload.py                            49     49     0%
/python/src/lidar/__init__.py                                            0      0   100%
/python/src/lidar/lidar.py                                              77     77     0%
/python/src/lidar/midpoints.py                                          25     25     0%
/python/src/lidar/point_cloud.py                                        86     86     0%
/python/src/lidar/sensor_setup.py                                      129    129     0%
/python/src/lidar/utils.py                                             399    399     0%
/python/src/roadsign.py                                                 25     21    16%
/python/src/utils.py                                                    68     68     0%
/python/tasks.py                                                        47     47     0%
/python/test/__init__.py                                                 0      0   100%
/python/test/image_cropping/__init__.py                                  0      0   100%
/python/test/image_cropping/conftest.py                                 17      0   100%
/python/test/image_cropping/test_crop_bounding_boxes.py                  8      0   100%
/python/test/lidar/__init__.py                                           0      0   100%
/python/test/lidar/conftest.py                                          16      0   100%
/python/test/lidar/lidar_data.py                                        13      0   100%
/python/test/lidar/test_locate_lidar_point.py                           23      0   100%
/python/test/model/test_classify.py                                     18      0   100%
/python/test/model/test_train_utils.py                                  22      0   100%
----------------------------------------------------------------------------------------
TOTAL                                                                 1365   1110    19%
```

Figure 18: Test coverage summary of the Machine Learning component

As shown in Figure 18, the total coverage percentage for the Machine Learning module is 19%. For a complete system, this would not be sufficient. For this proof of concept however, it is important to focus on certain parts of the system, e.g. *train-utils.py*, *classification.py*, *image-cropping.py*, and *lidar-data.py*, which have a test coverage respectively of 46%, 62%, 62% and 100%, which the team concludes is sufficient. Note that the *lidar-data.py* only covers reading and manipulating LIDAR data, e.g. filtering points in a point cloud based on intensity value, whereas the files inside the **/python/src/lidar** folder is the LIDAR component from Pylot, which is used in this system.

Bibliography

Education, IBM Cloud (2020). *Message Brokers*. Last accessed 16.05.2022. url: https://www.ibm.com/cloud/learn/message-brokers.

Appendix B

Pre-project Report

Department of Computer Science

IDATT2501 - Fordypningsprosjekt

Object detection and classification using high resolution images and LIDAR - Pre-project

Authors:
Øyvind Henriksen, Henrik Latsch Haugberg and Magnus Nordahl

January, 2022

Abstract

The field of machine learning has experienced further advancements over the years, with increasingly more complex deep convolutional neural networks, and better hardware for training these networks. A large field in machine learning is object detection and classification, where models are trained to detect and correctly classify objects in images. This project seeks to propose a solution for utilizing machine learning to detect and classify objects from images and LIDAR data collected along the Norwegian public road network. This will help to make the process of registering objects in a central database more efficient. The project is conducted in two parts, with a pre-project in the fall of 2021, and the main part of the project will be the students bachelor's thesis in the spring of 2022.

This report is the result of the pre-project, and describes the work that has been done in this part of the project. After giving an introduction to some of the methods and technology that is central in the project, a presentation is given of three relevant articles that was read in the literature study part of the project. It continues with a chapter about the project process, where different working techniques utilized are presented with a description of how they were practiced throughout the project. Here an account of the startup phase of the project is given, as well as short summaries of meetings with the NTNU supervisor, and how SCRUM techniques such as sprint plannings and daily meetings were used. Next a description of the development environment setup is given, where it is shown how a Java environment with Gradle and Spring Boot is set up to work with a Python environment using TensorFlow and Flask, when both these environments are set up inside separate Docker containers. Finally three different models for traffic sign detection are proposed, where the objective is to get an indication to which model would be best suited for the continuation of the project. One of the models, the Faster R-CNN model is completely pre-trained on the GTSDB dataset, while the other two models are trained 2800 traffic sign images from the Google Open Images dataset. Results from the Faster R-CNN model is difficult to compare to the others because of the difference in training, but it does demonstrate the possibility to do detection and classification in a single step. The YOLOv4 model performs the best out of the three by some margin, but it is expected that the EfficientDet model could reach the same precision, although it seems to require significantly more training.

Contents

Terminology and Abbreviations

POC - Proof of Concept

YOLO - You Only Look Once

R-CNN - Region Based Convolutional Neural Networks

NVDB - Nasjonal vegdatabank or National Roads Database

mAP - Mean Average Precision

NAS - Network Architecture Search

SaaS - Software as a Service

MVP - Minimum Viable Product

AI - Artificial Intelligence

OS - Operating System

ROI - Region of Interest

VM - Virtual Machine

1 Introduction

1.1 Background

Machine learning technology has been in rapid development over the past several decades, and advancements in hardware technology makes it possible to train deep convolutional neural networks more efficiently. A large field in machine learning is object detection and classification, where models are trained to detect objects in images, and correctly classify the type of object. A good example of technologies that are driving development in this field is self-driving cars, where it is necessary for this classification to happen quickly and with a high level of accuracy.

Object detection and classification can also be useful in monitoring and maintaining infrastructure. The National Roads Database (NVDB) is the Norwegian Public Roads Administration's central register, and contains information about the Norwegian road network, and objects such as traffic signs, noise barriers and a wide array of other objects of interest are registered here. These objects are often registered manually in the database, with operators using image data collected with a mobile mapping system. Machine learning could be utilized to automatically detect and classify objects and, if not fully automate the process of registering objects, at least make it significantly more efficient by giving suggestions that only need to be checked and edited by the operator.

1.2 Motivation

Developing skills in a large and complicated field like machine learning requires time, and it is important to get familiar with previous work and development in technology. Having an opportunity to work on the same project over two semesters means that the students can spend time gaining a deeper understanding of theory before starting the product development, and this gives more confidence when starting the development process later.

When working on this project the students have to utilize all of the different skills developed throughout their studies. It gives an opportunity to take programming skills and knowledge of specific frameworks and development methods, and combine them in a larger context than before.

Gaining familiarity with a field like machine learning in particular is also a motivating factor, and it will be rewarding to work with cutting edge technology. Machine learning technology is being utilized in an increasing number of businesses, and many potential future employers will be looking for candidates who have expertise in the field.

1.3 Project description

The goal of the project is to increase efficiency in registering objects of interest along road networks. The finished product should be able use machine learning to detect objects and classify of interest from image and LIDAR data, and generate a list of features for each object.

A simple web-based frontend should be developed to administer the tool, where it should be possible to feed images into the application to perform object detection and classification. A separate student group located in Oslo will be working in parallel on developing an application for administering the AI. If possible, the two projects will be merged in the end, but this is not in the main scope of the project.

The project will be split into two main parts, where the first part will be a pre-project that will be the teams deliverable for the course IDATT2501 Fordypningsprosjekt, and the

second part will be the teams bachelors thesis. The objectives and scope of the pre-project are detailed in the next section.

1.4 Objectives and scope

The first part of the project is a literature study, where the goal is to get familiar with what has previously been done within this field. This will give an overview of common technologies and methods in object detection and classification, and should help in making more informed choices about what direction to take later in the project. A summary of some of the literature reviewed is featured in chapter 3.

The second part is to construct a base for the application that will be developed in the bachelor project. Some system requirements for a Java backend is given by Triona, and a solution is needed for how to integrate this with a machine learning environment in Python. The backend also needs to be prepared for supporting a web-based frontend. The solution for this will be discussed in chapter 5.

The final objective is to create a proof of concept of object detection. The performance of two types of models trained to detect traffic signs in images will be compared in detecting Norwegian traffic signs. The training of the models will not be specialized for detecting Norwegian signs, but it will give an indication of what is possible. This will also give more information for making a choice about which approach to take when starting development in the bachelor project. The results of this is shown in chapter 7.

2 Methodology and Technology

This chapter gives brief introductions to technologies and development methodology that have been utilized in the project.

2.1 Literature studies

Literature studies are used to gain a deeper general understanding of the broader field of machine learning, and also to more specifically study object detection and classification. For this project it has been important to get an overview of different models and techniques, in order to be able to make informed choices on what approach to take when starting development. A good example of this is the difference between one-stage and two-stage object detectors, and the benefits or disadvantages of each for different use-cases.

2.2 Azure Devops

Azure Devops is a Software as a Service from Microsoft, providing team collaboration services, allowing access to sprint planning tools, repositories, pipelines and more. For this project the tool has been used for setting up user stories that has then been split into smaller tasks, and each sprint has been planned with these tasks as a basis. All code in the project is also kept in the Azure DevOps Git repository.

2.3 Java

2.3.1 Gradle

Gradle is an open-source build-automation tool designed to be flexible and build different types of software, and has support for multiple languages. It controls the development process of compilation and packaging, to testing, deployment and publishing. (Gradle 2022)

2.3.2 Spring Boot

Spring Boot is an open-source Java based framework that provides a stand-alone and production-grade Spring based application. It provides an application context and embedded web server for easy microservice development. Spring Boot allows easy configuration and development of a RESTful service. (Spring 2021)

2.4 Python

2.4.1 Flask

Flask is a light-weight Python microframework for creating a web application. It is classified as a microframework due to it not requiring any tools or libraries. Flask has no pre-existing third-party libraries, database abstraction layer, or form validation. However, this functionality and other application features can be added through extensions, as if they were implemented in Flask itself. (Flask 2022)

2.4.2 TensorFlow

TensorFlow is a machine learning platform, that offers multiple levels of abstraction to build and train models using the Keras API. TensorFlow is well integrated with Numpy, a stable Python API. Various extensions and libraries can be added to extend and advance the different models and methods used in machine learning. (TensorFlow 2022)

2.5 PyTorch

PyTorch is an open source machine learning framework, primarily developed by Facebook’s AI Research lab (FAIR). PyTorch’s interface is available both in Python and C++, although Python is the most polished one. PyTorch also offers GPU accelerated operation through CUDA. (Paszke et al. 2019)

2.6 Docker

Docker uses OS-level virtualization to containerize software, and is a set of Platform as a Service (PaaS) framework. Docker provides the ability for software to be bundled and run in a loosely isolated environment called a Docker container. These containers are lightweight, and contain everything the application needs in order to run. Docker uses a client-server architecture, and containers can communicate with each other using a REST API, Unix sockets, or a network interface. (Docker 2022)

2.7 Colaboratory

Colaboratory, also known as ”Colab”, is a product from Google Research. Colaboratory allows easy and free access to execute Python code through Jupyter Notebooks in the browser, with allocated CPU, RAM and GPU resources. The free version of Colab has been used for training the object detection models presented in this report. (Research 2021)

2.8 Darknet

Darknet is an open source neural network framework written in C and CUDA, with support for CPU and GPU execution. Darknet offers automatic data augmentation and a range of other features to aid in model training. (Redmon 2013–2016)

2.9 CUDA

CUDA, originally known as Compute Unified Device Architecture, is a parallel computing platform and API, that allows software to execute parallel instructions on a GPU. It works with programming languages such as C, C++ and Fortran. CUDA was developed by Nvidia, under a proprietary license.

2.10 Training data

There are numerous datasets publicly available both for the task of classifying and detecting road signs. The datasets often suffer from common challenges, such as small sample size, lack of diversity in lighting conditions, imbalanced class distributions or narrow scope of classes. This makes it important to choose the right dataset for the specific task at hand.

This section presents the different datasets used for training the different object detection models of the POC. Possible strengths and weaknesses of each dataset are presented.

2.10.1 Open Images Dataset

The open images dataset is a database of more than 9 million open source images, with 600 classes. It features rich annotation types, and on average more than 8 different objects per image, making the dataset adaptable for a wide range of detection tasks. (**2020**)

For the specific task of training traffic sign detection, the dataset contains approximately 2800 images of traffic signs annotated with bounding boxes. Image resolution is generally high, with a natural variance since they are crowd sourced. Images have a wide range of variety, and include signs from different continents and countries. Some of the images have a rather liberal interpretation of what constitutes a traffic signs i.e. not actual road signs, but rather pedestrian signs or similar looking signs.

The great variation in the traffic sign images of this dataset is also possibly the source to it’s greatest strengths and weaknesses. Models trained on this dataset will have a very generalized idea of what a traffic sign looks like, and could likely be used with similar accuracy in different continents. This effect is likely to produce a number of false positives, with this being it’s greatest weakness.

Figure 1: Example of traffic sign images from Open Images (**2020**)

2.10.2 GTSDB - German Traffic Sign Detection Benchmark

The German Traffic Sign Detection Benchmark dataset was introduced in 2013 at the IEEE International Joint Conference on Neural Networks 2013. It consists of 900 images divided into three classes; ”prohibitive”, ”mandatory” and ”danger”, all recorded in spring and autumn of 2010, near Bochum, Germany, and are annotated by bounding boxes. Images are 1360x800 pixels and signs within the images are between 16 and 128 pixels wide/high. (Houben et al. 2013)

As the name suggests, this dataset consists of traffic signs in Germany. As such, they follow the Vienna Convention on Road Signs and Signals, making them similar to signs from large parts of Europe, Russia and Asia.

In contrast to the traffic sign images from the Open Images Dataset, the scope of images in this dataset is fairly strict, and does not invite much in the way of generalization. There is potential value in the three distinct annotation labels, if this can be leveraged during model training. Any sign that falls outside the three categories are likely to be ignored during detection. This is also perhaps the greatest weakness when considering this dataset for training an object detection model. If there is a need for detecting signs outside the scope of the annotation labels, these signs might become false positives during training.

Figure 2: Example of images from GTSDB (Houben et al. 2013)

3 Theory

This chapter gives a summary of some of the theory that is relevant for the project. It contains article reviews that are a result of the initial literature studies, as well as an introduction to other relevant theory.

3.1 Previous work

3.1.1 Article review: LIDAR and Vision-Based Real-Time Traffic Sign Detection and Recognition Algorithm for Intelligent Veichle

The article proposes a new traffic sign detection and recognition algorithm based on the fusion of onboard camera and LIDAR on a veichle. Previous work focuses mainly on detecting and recognizing traffic signs captured by an onboard camera alone, where the visual features of a traffic sign is sensitive to illumination, angle of view, occlusion, etc. The detection of the traffic sign in 3D space is thought to help this, as well as using combined colour spaces (CCS), to treat traffic sign colours as one class. Region of interests (ROI) are used for traffic sign recognition, and usually suffer from perspective deformation. This is rectified by the fusion of LIDAR and onboard camera data. For classification of traffic signs, histogram of oriented gradient (HOG), and support vector machines (SVM) are used.(Zhou and Deng 2014)

Detection of traffic signs is done in 3D space. The ROIs that are obtained from 2D vision-based detection algorithms are generally too broad and rough to eliminate the background where the traffic sign will not appear. The data gained from LIDAR creates a much tighter ROI, increasing efficiency. Laser points in the ROI are projected into the

image plane from the onboard camera data, in order to get the corresponding colours of the traffic sign.

Colour-based methods of traffic sign detection algorithms choose a colour space to identify separate traffic sign colours, creating multiple classifiers, one for each colour. Instead, one can use combined colour spaces, to treat all traffic sign colours as one classifier. The CCS used in the proposed algorithm consists of RGB, HSV, and CIE L*a*b (LAB). The colour is further combined with laser reflectivity, aspect ratio, and size, to detect the traffic sign in 3D space.

Each laser point has a feature vector of 10 components, one for laser reflectivity, and 9 for the different colour components in the CCS. A linear SVM is trained and used to classify the colourised laser points. There are different circumstances where colour from the camera data is more useful than the laser reflectivity from LIDAR, and vice versa.

Figure 3: An example of a situation where the colour is very distorted due to exposure, and where laser reflectivity can be a useful tool to detect traffic signs. (Zhou and Deng 2014)

a) Original camera colour image.

b) Result of blue and red pixel detection in normalized RGB colour space.

c) Laser point cloud rendered by laser reflectivity.

Figure 4: An example of a situation where the laser reflectivity is not very discriminative, but colour is an effective tool to detect traffic signs. (Zhou and Deng 2014)

a) Original camera colour image.

b) Result of blue and red pixel detection in normalized RGB colour space.

c) Laser point cloud rendered by laser reflectivity.

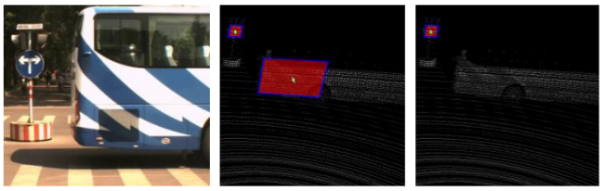


Figure 5: An example of a situation where there are multiple planar objects with similar colour and laser reflectivity. Verification of size and aspect ratio is used to solve this. (Zhou and Deng 2014)

a) Original camera colour image.

b) The bounding boxes of the ROIs.

c) The ROI after size and aspect ratio verification.

The article concludes with a proposition of a new traffic sign detection and recognition algorithm based on the fusion of LIDAR and camera data. The traffic signs are detected in 3D space, by employing the traffic sign position, color, laser reflectivity, and 3D geometric features. CCS is used to treat different traffic sign colors as one class. The laser reflectivity combined with the CCS, is used to train a linear SVM for identifying laser points belonging to the traffic sign. The 3D geometric features are used to verify the ROI's. For traffic signs within 100 meters, the detection rate for the algorithm is 95.87%, and the recognition rate is 95.07%.

3.1.2 Article review: Road signs detection and recognition utilizing images and 3D point cloud acquired by mobile mapping system

This article presents an approach for automatically detecting and recognising road signs using both images and 3D point clouds. Being able to detect and classify road signs is very important in creating highly accurate road maps which are necessary fora automated driving, and this is the starting point of the article.(Li et al. 2016)

The approach consists of three stages: Detection, filtering and classification. In the detection stage, image analysis is used to detect candidates from images based on color and shape. Multi scale image segmentation is used, where both color and shape information are considered. After the segmentation is done, objects that have specific colors are considered target objects, and are extracted. Then target object that are close to one another, and also have the same color, are merged together. The last step is to perform shape recognition on the merged target objects to select road sign candidates based on color and shape.

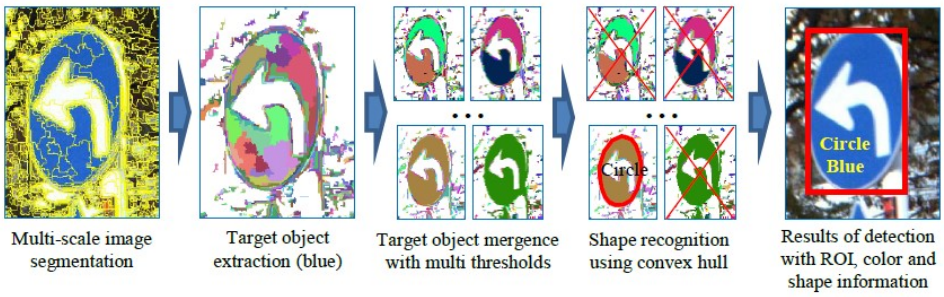


Figure 6: Detection of road sign candidates from MMS image (Li et al. 2016)

The next step is filtering, where things like size and installation height can be used to filter out false positives. In this example the fact that the size of road signs in Japan is in the range of 40cm-180cm, and the installation height is between 1m-6m is used. Size and installation height is estimated using point cloud, as well as focal length of the lens. It is pointed out in the article that it is better to have some false positives than too many false negatives. A false positive does take some manual work to correct after the automatic detection and classification is done, but a false negative is likely to not be discovered at all.

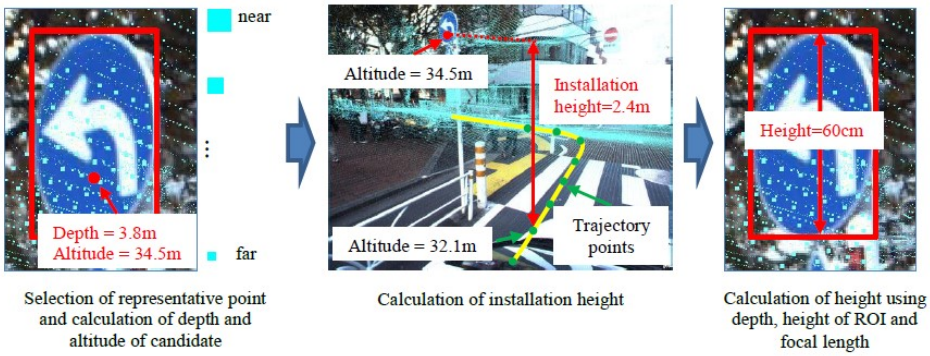


Figure 7: Filtering of detected road sign candidates using point cloud (Li et al. 2016)

The final step is classification of the remaining candidates. This approach is not based on machine learning, but instead uses template matching, which dos not require the same amount of sample data. The drawback is that it is sensitive to deformation from the perspective in the images, so a shape normalisation has to be done. For example, if the outline of the object from the image is oval, the outlines are extracted and RANSAC based ellipse fitting performed. The image is then transformed for shape normalization, as shown below.

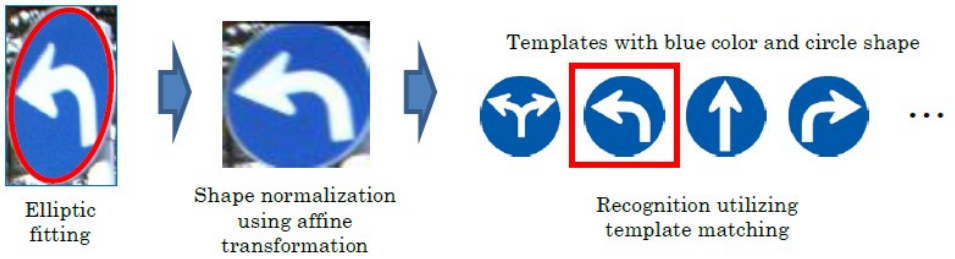


Figure 8: Classification with template matching after shape normalization (Li et al. 2016)

The article concludes that the average recall of sign detection is 98,4%, which i slightly higher than with manual extraction, which means it would be possible to improve efficiency using automatic sign detection. The amount of over detection and false classification is however still too high, and machine learning techniques such as deep learning should be considered.

3.1.3 Article review: A Survey of Modern Deep Learning based Object Detection Models

This article gives an extensive overview of the current progress in the field of deep neural network object detection, along with discussions of future trends. It introduces a discussion of two stage versus single stage detecting models, and the current standing of each branch. The authors make the point of how many of today's state of the art machine learning algorithms are tuned towards both speed and being lightweight, to be used in

realtime applications, and also be able to run on low power devices. Many models employ separate backbone networks for the task of classification after RoI's are detected. Often times other well known image classification networks are simply used as a backbone, while others employ custom networks for this task.(Zaidi et al. 2021)

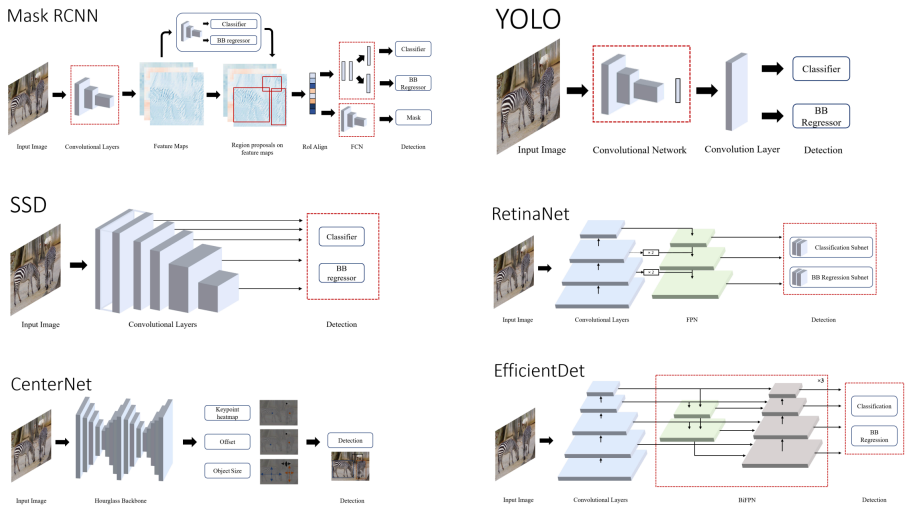


Figure 9: Illustration of the internal architecture of different two and single stage object detectors (Zaidi et al. 2021)

Traditionally two stage detectors leveraged high accuracy at the cost of computational power and speed while single stage detectors were mainly aimed at realtime applications. This often meant single stage detectors were measured in computation time (FPS), and thus suffering a loss of accuracy was accepted. At time of writing (early 2021), single stage detectors are delivering performance close to that of the strongest two stage detectors . The authors predict a future trend of using NAS for building object detector networks and backbone networks, along with an exponential increase in use of lightweight detection models for edge devices and realtime applications.

3.2 Machine learning

Machine learning is the field of creating algorithms to imitate intelligence through learning. Computer scientist and machine learning pioneer Tom M. Mitchell explains it as: "*Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience.*" (Mitchell 1997). It is a branch of the larger field of Artificial Intelligence, and in everyday speech the two words are used interchangeably. Machine learning affects many areas of society in the twenty first century, both visibly and invisibly. Commonly, it is said that a task that a human finds simple, the computer may struggle with, and vice versa.

3.3 Computer vision

Computer Vision is the branch of Machine Learning that processes and interprets the world through vision. Computer vision algorithms are trained in fields such as image classification, object detection, scene recognition and object tracking to name a few. Early efforts into computer vision have been centered around still images, but in recent times, more and more interest is put into real-time processing of images/video, to aid in tasks such as manufacturing and driving.

4 Process

This section gives a summary of how the team has worked throughout the project. It gives insight to how decisions were made, how project management tools and techniques were used, and a general view into the process.

4.1 Startup phase

The project was initially submitted by Triona to NTNU, and was assigned to the team on September 9th. The team had a short meeting with the course coordinator to talk about how to proceed, and then made contact with Triona. A meeting was held with Triona on September 23rd, where the goal was to get to know each other a little bit, and the students were given a introduction to the company. During this meeting an agreement to collaborate on the project was made. And introductory meeting was scheduled for the week after, where a plan for how to proceed was made.

Since the deadline for delivering the final project report was set to january 3rd, the team decided that the best approach would be to set aside a relatively small amount of time to work on the project in the beginning of the semester, and instead intensify in the weeks towards the deadline for delivery, since the workload in other courses would be lower in this period. In September and October the students therefore mostly worked individually on reading up on theory and previous work, but with regular checkup-meetings internally in the team, as well as some status-update meetings with Triona. The team also scheduled a couple of meetings with their NTNU supervisor, to get some input and advice on the approach for the POC.

4.2 Meetings with NTNU supervisor

Once it was clear that the project would involve machine learning the team made a request for Ole Christian Eidheim to be the supervisor on behalf of NTNU. All three students were taking the course 'IDATT2502 Anvendt maskinlæring med prosjekt', where Ole Christian was a lecturer. Ole Christian agreed to supervise the project, and this was also cleared with course coordinator Alexander Holt.

The team had three meetings with their supervisor through the fall. The first meeting was mainly spent giving an introduction to the objectives of the project, and getting advice on what kind of previous work it could be a good idea to start reading up on.

The team was also working on a project in the course 'IDATT2502 Anvendt maskinlæring med prosjekt' at the time that also involved machine learning, and specifically classification of traffic signs. The second meeting was mainly a discussion around how to separate these projects from each other. Since the other project focused on classification, the team felt it was important that this project had a different direction, and after some discussion decided that the POC could focus more specifically on object detection.

The final meeting was held around the time the team as about to start work with the POC. There was some uncertainty about what direction to take, and the students wanted some advice on what would be best for the project from NTNU's perspective. the meeting helped clear up some of the issues the team had been thinking about, and helped the team take the direction for the POC that is described in this report.

4.3 Sprint planning

After the period of reading up on theory and previous work, it was time to start the actual system development. In collaboration with the supervisors from Triona, the team planned a SCRUM style sprint for the next six workdays of the project. Azure DevOps was used to set up user stories, divide them into into smaller tasks, and time estimates was set for each task. The result was that the goal for the first sprint would be to set up the base functionality for the Java/Python backend.

The second sprint would also be the final sprint of the pre-project. The goal here would be to implement two different object detection models to serve as a proof of concept, and also to write the final report for the project. Time estimation for each task was not as much in focus here as in the first sprint, but a goal was set for when the object detection models should be finished in order to leave enough time for writing the project report.

Although the team have some previous experience from other school projects in working in sprints, it was a good experience to collaborate on sprint plannings with the supervisors from Triona, as they helped to gain more insight into how to think during the planning. A good example is time estimation, an area where the students almost always suggested lower estimates than the Triona employees. During the meetings everyone gave their reasoning for their estimate, until everyone agreed on the estimate for each task. The agreed upon estimates proved to be precise in most cases, and also gave some much needed wiggle room for resolving unexpected issues along the way.

4.4 Dailies

During both of the sprints, daily meetings were held with the team and supervisors from Triona. Dailies were used for giving short updates on what each team member had been working on since the last meeting, what they were planning to work on going forward, and for seeking help from the rest of the team with any problems or difficulties. On several occasions during this project the daily meeting were used for the team to get advice on technical challenges, or seek help with or discuss issues they were facing. Sometimes an issue was resolved by having a short discussion during the meeting, and other times a separate meeting was planned in order to have more time to discuss solutions.

5 Development environment setup

The plan for the finished product is to have a simple web-based front-end to administer the AI. In its simplest form, a user should be able to upload images using the web application, and get back a list of detected objects with classification and features. As part of the pre-project, the base for the application was set up so that it is ready for when work with the project resumes in the spring of 2022, but it will not be used for the POC shown in this report. This chapter will explain the initial setup for this system, and give reasons for the choices that were made regarding architecture and technology.

5.1 Java environment

5.1.1 Gradle

Gradle was used as the build tool for the Java application. Maven is an alternative to Gradle, and was initially meant to be used as the build tool for the project. This was later changed due to this beeing a technical requirement from Triona, as Gradle was the preferred build tool to be used in their future projects.

5.1.2 Spring Boot

Spring Boot was used to develop a Spring framework based Java RESTful service, for easy communication between a server and client. The Spring Boot backend needed to act as both a server and a client, for enabling communication with a Python Flask server. This was achieved by using the supported Spring Boot WebClient dependency.

5.2 Python environment

5.2.1 TensorFlow

The machine learning part of the application is to be developed in Python, due to limited machine learning support in Java. LIDAR point-cloud data will be incorporated with the AI, making TensorFlow the most suited platform to use, due to the release of TensorFlow 3D, which supports LIDAR data. TensorFlow 3D is currently only supported in Python 3.6 (GitHub 2021)

5.2.2 Flask

The Python service is a Flask server, which is further developed into a RESTful application. Endpoints are set up in the Flask server, which the Java application sends a HTTP request to. The Flask server then sends response data containing a list of detected objects, along with corresponding metadata. Future plans include implementation of a Python job queue together with the Flask server, in order to handle asynchronous calls and have the processes work in the background.

5.3 Docker

Docker is ideal when having multiple different software environments that need to communicate with one another. Docker containers communicate via a network, making RESTful services an ideal way of communicating between the different containers through HTTP requests. One Docker container was created for each software environment, i.e. Java and Python. Docker Compose is used to run the containers simultaneously during development.

6 Proof of concept implementation

As a POC of object detection, it was decided to implement three different types of object detection models, and compare their performance in detecting road signs in images. The result of this will help in making an informed decision about what direction to take when continuing development in the spring of 2022.

6.1 Test data

To test the different object detection models, 143 real world images were picked from a large set of images. These images were captured in Oslo during the summer of 2021, as a part of a larger project to digitize the road network in the county. Images are captured from a 360 degree camera mounted on top of a car. 6 individual images are recorded in 6 different directions every 5 metres, in a resolution of 2048x2048 pixels each. The

handpicked images stem from the front facing exposures. An effort was made to collect a test set than was varied in all areas such as type of sign, distance between camera and signs and lighting conditions. It was also made sure to include some images that did not contain traffic signs, or images where the signs were partially covered or otherwise obfuscated.

Figure 10: Sample images from the test set, with annotations in red bounding boxes.

Test images were manually annotated to a single class, to set a ground truth for each image. Traffic signs below 50x50 pixels in size were not annotated. This was done because each sign is usually seen 3-4 times each, and choosing the image with the largest or best visibility of the sign will usually produce the best classification in the next step. Detection of these smaller non-annotated signs are counted towards the ”bonus” column in the results chapter, to still give an idea of sensitivity towards smaller objects. The bonus values are however not counted towards neither recall nor precision. In total 80 signs are annotated from the 143 total test images, and these 80 signs are regarded as the ground truth of the test set.

6.2 Implementation of a YOLOv4 model

You Only Look Once (YOLO) is an object detection model first proposed in the article *You Only Look Once: Unified, Real-Time Object Detection* in 2015. YOLO burst onto the object detection scene in 2015, proposing an entirely different and more efficient approach to inference in object detection. The mAP of the model was barely on par with other non-realtime object detectors of the time, but more than doubled the precision of other realtime detectors. It has since come in a number of slight improvements with a v2, v3, v4 and a v5. (Redmon et al. 2016)

A YOLOv4 model was chosen for running performance tests, as YOLOv5 is somewhat contested as performance was not necessarily increased from YOLOv4. The model was trained on 2800 traffic sign images from the Open Images Dataset. Training was

performed through Colaboratory and Darknet, as experiments running on computers without GPU leverage yielded training time estimates of 700 hours or worse. With Colaboratory's free tier GPU-VM's, training could be done in under 24 hours. As is common practice with object detection models, the training was initialized with weights that are pre-trained on ImageNet. (Zhuang et al. 2020)

6.3 Implementation of a pre-trained Faster R-CNN model

The article *Evaluation of deep neural networks for traffic sign detection systems* compare several different models that was pre-trained on the Microsoft COCO dataset, and fine tuned on the GTSDB dataset. Their pre-trained Faster R-CNN Inception Resnet V2 model was the one that achieved the highest accuracy, and was chosen as an example of a Faster R-CNN model for this report. (Arcos-García et al. 2018)

Faster R-CNN uses a Region Proposal Network (RPN) that shares convolutional feature maps with the detection network. This means it gives a proposed region for an object, and creates a bounding box or gives coordinates for this region. It then has a classification layer to predict the class of the object. (Arcos-García et al. 2018)

The model has been implemented with the Tensorflow Object Detection API, and inference has been run on the same test dataset used for the YOLO model.

6.4 Implementation of an EfficientDet-D0

EfficientDet was proposed by engineers at Google Research in 2019, as a result of an extensive review of current and former state of the art object detection models. The same research team proposed the backbone network EfficientNet earlier in the same year, which EfficientDet is built upon. The focus of EfficientDet is as the name implies, to improve computational efficiency while maintaining accuracy. EfficientDet is released as a family of 9 different sized models, with each model taking slightly larger inputs and being more performant than the previous. (Tan et al. 2020)

EfficientDet-D0 is the smallest of the EfficientDet models, which takes inputs of 512x512 and should perform nearly identical to YOLOv3 on the COCO Dataset. This model was chosen for running performance tests, both due to it's similar performance to the YOLO counterpart, and due to constraints of resources available for training. The model was trained on the same 2800 traffic sign images as the YOLOv4 model. Training was performed in Colaboratory through Pytorch, to leverage GPU computing power.

7 Proof of Concept Results

Results are measured with the following metrics:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} = \frac{\text{True Positive}}{\text{All Observations}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} = \frac{\text{True Positive}}{\text{All Ground Truth}}$$

These numbers do not account for any traffic signs outside the range of the 80 annotations, and are simply counted towards the ”bonus” column of each table. More than 150 traffic signs are present in the test set, but traffic signs smaller than 50x50 pixels are not counted towards the model accuracy.

7.1 YOLOv4

Threshold	True Positive	False Positive	False Negative	Bonus	**Precision**	**Recall**
10%	72	23	8	83	**75.78%**	**90%**
50%	66	1	14	36	**98.50%**	**82.50%**

Table 1: YOLO object detection results from 143 test images containing 80 annotated signs

The YOLOv4 model trained on traffic sign images from the Open Images Dataset deliver a decent result. At 10% threshold the model detects approximately 14% false positives among all the detected signs, and at 50% threshold the value is less than 0.01%. A number of the detected signs are below the size threshold of 50x50 pixels, but can easily be discarded at a later stage if need be.

7.2 Faster R-CNN Inception Resnet V2

Threshold	True Positive	False Positive	False Negative	Bonus	**Precision**	**Recall**
10%	42	0	38	4	**100%**	**52.50%**
50%	39	0	41	2	**100%**	**48.75%**

Table 2: Faster R-CNN object detection results from 143 test images containing 80 annotated signs

The Faster R-CNN model trained on the GTSDB dataset is fairly accurate, when keeping in mind that the dataset does not contain some of the traffic signs in the test set. In particular, right of way, yield and a few others are simply ignored, by no fault of the model, but the dataset not containing sample images of these. The signs that are detected are usually predicted with an accuracy of 95% or above. Still a few of the false negatives are signs the model should accurately detect, which are not seen at either 10% or 50% threshold values. No false positives are detected, regardless of sensitivity. Some signs that does not belong to any of the three classes are detected and classified, as the example in figure 11 shows.

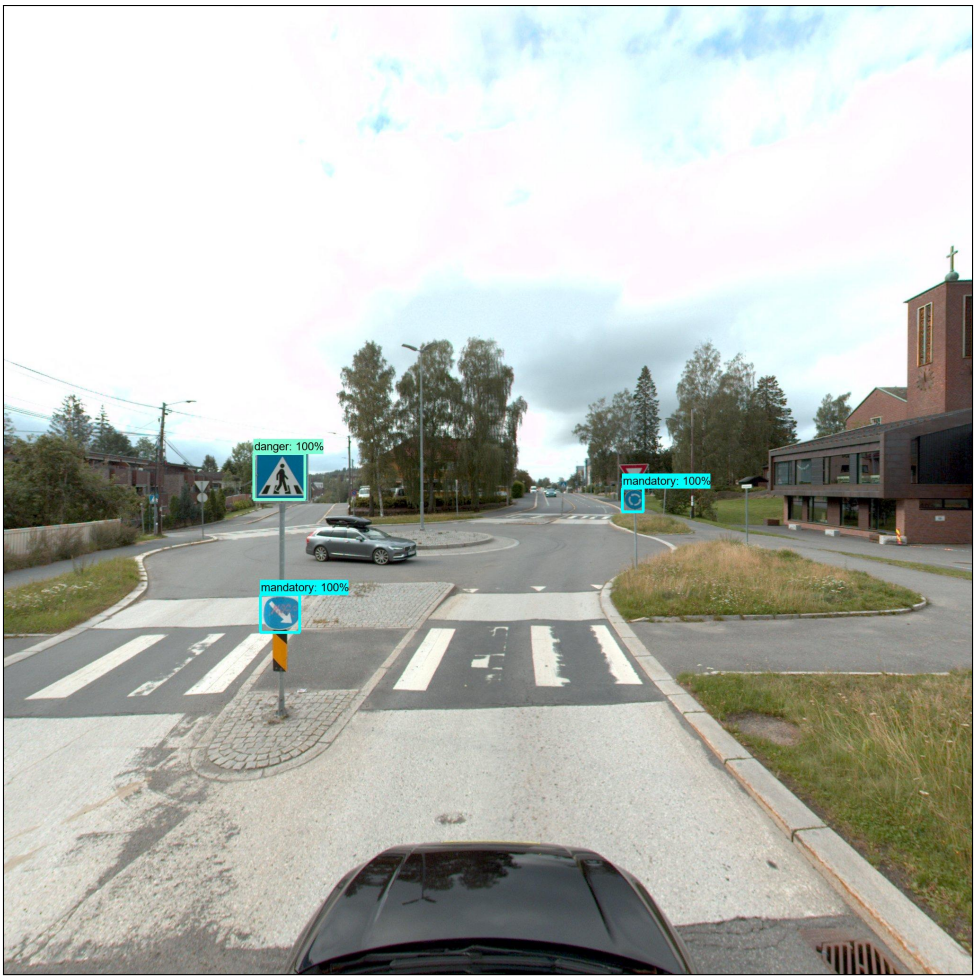


Figure 11: Example of pedestrian sign classified as danger sign

7.3 EfficientDet-D0

Threshold	True Positive	False Positive	False Negative	Bonus	**Precision**	**Recall**
10%	37	63	43	2	**37%**	**46.25%**
25%	23	9	57	0	**71.87%**	**28.75%**
50%	16	1	64	0	**94.11%**	**20%**

Table 3: EfficientDet object detection results from 143 test images containing 80 annotated signs

The EfficientDet model did not perform as expected, as this model was theoretically supposed to be fairly on par with YOLOv4 performance. The model struggles to gain a high confidence in even the most clear cut cases, with only 16 of 80 signs showing a confidence score of more than 50%. Trying even lower confidence thresholds simply yielded too much noise, which is clearly seen at the 10% threshold.

8 Discussion

8.1 Object detection models

Comparing the models trained on Open Images traffic signs to the Faster R-CNN model trained on the GTSDB dataset is not quite fair, and no real conclusions can come from directly comparing models trained on different datasets. In spite of this, the possibility to leverage pre-trained models had to be explored, and is potentially a viable option given the right model/dataset.

There were some surprises among the results from the POC implementations, and YOLO outperforming EfficientDet was one of these surprises. The EfficientDet model was chosen for it's theoretically equal performance to YOLO, but performed poorly compared to YOLO in tests. Both models were trained to convergence on the same set of training images. A potential explanation to this discrepancy could lie in the automatic image augmentation implemented in Darknet, the framework used to train the YOLOv4 model. The augmentations used are exposure, saturation and hue. These few augmentations should seemingly not account for this great of a difference alone, but no other explanations have been found thus far.

Despite YOLO looking far superior to EfficientDet in these tests, there still needs to be made further attempts with state of the art object detectors that prioritize accuracy over speed. Given more GPU processing power, there will be performed tests on higher resolution versions of models like EfficientDet, to hopefully surpass or match human performance in object detection.

These POC implementations have focused almost solely on the ROI-aspects of the models. The Faster R-CNN model classifies in three different classes, but the YOLO and EfficientDet-models both only classify into the single 'traffic sign' class. The models are originally designed to both detect and classify, but these tests have focused more on the detection of the objects, rather than classifying. Going forward, the models either need to perform classification in the same model, or pass the detected objects to a separate classification model. Splitting them into two entirely separate models could prove beneficial in tweaking and general modularity of the system.

8.2 Datasets

The GTSDB dataset is annotated with three different classes, and a model therefore gives a prediction of which of these three classes an object belongs to when running inference. This is a useful demonstration of the possibility to do this classification at the same time as detection, but for the purpose of this project it is clear that this dataset can not be used when training the final model for detecting traffic signs. The main reason for this is that it only contains a smaller number of classes of signs, and tests on the Norwegian sings showed that the model trained only on this dataset often did not detect signs that are not in these specific classes.

The models trained on the Open Images dataset show an ability to detect a wider range of traffic signs, and also detects signs that are not traffic signs specifically but rather signs with company names etc. This indicates that it could be possible to use this dataset for training the detection part of the model, and use a different training set for only training the classification models. In the continuation of the project this could eliminate the need to construct a training set for sign detection from scratch, and therefore potentially save a lot of time in the development process.

8.3 Process

A conscious decision was made to not use strict SCRUM principles in the initial phase of the project, but rather to have longer time intervals between meetings in the beginning. This may be viewed as the wrong approach by some, and it is important to reflect on how this arrangement suited the project in the end. The reasoning for this was clear form the beginning, where the different schedules of each team member could make it difficult to coordinate regular meetings, and the workload was intended to increase towards the end of the project. Each team member also had a clear understanding of what their task was, which mainly consisted of literature studies at this stage of the project, and all in all the team feels this approach worked well. A possible improvement could have been to make a decision at an earlier stage about an exact date to start the development process.

When work on setting up the development environment and developing models started, the team used SCRUM techniques as described earlier in this report. There is no doubt that this is the correct approach for this kind of development work, and the team will continue to utilize SCRUM when continuing the project in the spring of 2022.

9 Conclusion

Overall, this pre-project has given us a good foundation for the next stages of the project. We have gotten introductions to Triona's other software offerings, and gotten to know some of the employees we will be working closely with for the next six months.

Due to the different models being trained on different datasets, no direct conclusion can be drawn on which model is better suited for the task. However, the comparison of the models can give a pointer to further exploration and research in the usage of pre-trained models for object detection.

The results from the comparison of the models show that the YOLOv4 model had the best performance for a POC, with a high precision and recall score. In theory, the EfficientDet model should have equal performance as the YOLOv4 model, although this was not the case based on the results. The most limiting factor currently is the training of the models. Because of the complexity of the models, the training has to be computed on GPU's. With even further training, the EfficientDet model could potentially outperform the YOLOv4 model.

Going forward with this project, it is evident that more computing power is needed. Training of small scale image classification models can be performed on modern laptop computers, but object detection models work on much greater image sizes. Training the models used in the POC tests have been done through Googles Colaboratory, which is a free service that can be unreliable. Due to this, a new solution for training models will need to be found before continuing model development.

While the object detection models still leave something to be desired in terms of accuracy, we feel confident that the best performing model is good enough to base an MVP on. So long as the system is built in a modular fashion, a better object detection model can be implemented further down the line.

The team believes that the solution created for the the development environment will prove to be a good one, and this setup is also ready to be implemented with the planned web-based fronted in the next part of the project. The actual training of the models will likely have to be done outside of this system. The intention of the frontend is to be able run inference on new images using the trained models, where an end user who is not familiar with machine learning should be able to perform this task.

Bibliography

Arcos-García, Álvaro, Juan A Álvarez-García, and Luis M Soria-Morillo (2018). "Evaluation of deep neural networks for traffic sign detection systems". eng. In: *Neurocomputing (Amsterdam)* 316, pp. 332–344. issn: 0925-2312.

Docker (2022). *Docker overview*. url: https://docs.docker.com/get-started/overview/ (visited on Jan. 2, 2022).

Flask (2022). *Foreword*. url: https://flask.palletsprojects.com/en/2.0.x/foreword/ (visited on Jan. 2, 2022).

GitHub (2021). *TensorFlow 3D*. url: https://github.com/google-research/google-research/tree/master/tf3d (visited on Dec. 10, 2021).

Gradle (2022). *What is Gradle?* url: https://docs.gradle.org/current/userguide/what_is_gradle.html (visited on Jan. 2, 2022).

Houben, Sebastian et al. (2013). "Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark". In: *International Joint Conference on Neural Networks*. 1288.

Li, Y. H. et al. (2016). "Road signs detection and recognition utilizing images and 3D point cloud acquired by mobile mapping system". In: *International archives of the photogrammetry, remote sensing and spatial information sciences* XLI-B1, pp. 669–673.

Mitchell, Tom M. (1997). *Machine Learning*. New York: McGraw-Hill. isbn: 978-0-07-042807-2.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. url: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Redmon, Joseph (2013–2016). *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/.

Redmon, Joseph et al. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. arXiv: 1506.02640 [cs.CV].

Research, Google (2021). *Colaboratory*. url: https://research.google.com/colaboratory/faq.html (visited on Dec. 10, 2021).

Spring (2021). *Spring | Why Spring?* url: https://spring.io/why-spring (visited on Dec. 10, 2021).

Tan, Mingxing, Ruoming Pang, and Quoc V. Le (2020). *EfficientDet: Scalable and Efficient Object Detection*. arXiv: 1911.09070 [cs.CV].

TensorFlow (2022). *Why TensorFlow*. url: https://www.tensorflow.org/about (visited on Jan. 2, 2022).

Zaidi, Syed Sahil Abbas et al. (2021). *A Survey of Modern Deep Learning based Object Detection Models*. arXiv: 2104.11892 [cs.CV].

Zhou, Lipu and Zhidong Deng (2014). "LIDAR and Vision-Based Real-Time Traffic Sign Detection and Recognition Algorithm for Intelligent Veichle". In.

Zhuang, Fuzhen et al. (2020). *A Comprehensive Survey on Transfer Learning*. arXiv: 1911.02685 [cs.LG].