Master's thesis

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

Vebjørn Johansen Nilsen

Autonomous Vehicles: 3D Object Detection Using Dual Sensor Input of LiDAR Point Clouds and 3-channel LiDAR Images

Master's thesis in Informatics - Artificial Intelligence
Supervisor: Frank Lindseth
Co-supervisor: Gabriel Kiss and Durga Bavirisetti
June 2023

NTNU
Norwegian University of
Science and Technology

Vebjørn Johansen Nilsen

Autonomous Vehicles: 3D Object Detection Using Dual Sensor Input of LiDAR Point Clouds and 3-channel LiDAR Images

Master's thesis in Informatics - Artificial Intelligence
Supervisor: Frank Lindseth
Co-supervisor: Gabriel Kiss and Durga Bavirisetti
June 2023

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

Abstract

The idea of autonomous vehicles has been a futuristic vision people have dreamed about for decades. In recent years huge steps towards autonomy has been made with the introduction of CNNs and transformers in combination with exponential-like improvement in compute power. This rapid expansion within computer vision has sparked interest for many researchers, and is the inspiration for this thesis.

The work in this thesis revolves around autonomous vehicles, and more specifically 3D object detection using images and LiDAR data. The model Frustum-PointPillars was chosen based on performance, real time inference and dual sensor input. The goal of the project was to reproduce the stated state of the art results and create a real-time inference pipeline adjusted for Ouster LiDAR sensors. In the first chapter, an introduction to the topic and the research goals is thoroughly explained. Then the necessary knowledge for background is described in detail and related work within state of the art 3D object detection is explored with a deep-dive into different fundamental methods.

The results of this thesis show that the reproducibility of the model through training a new model from scratch is possible using a single RTX 8000 GPU. The model works well on the KITTI data used for training, but it struggles somewhat on unseen data. The results on cars and cyclists exceed the authors results with 37.5% less training epochs. The inference time of the model is more than 2 times faster than average human reaction time, making it viable for deployment in autonomous vehicles.

The conclusion as to why the model performs subpar on Ouster data is lack of annotated data. The process of annotating 2D and 3D data of the same scene is time consuming when there is limited access to free annotation tools. The results would probably be better with fine-tuning on similar data.

Sammendrag

Autonome kjøretøy har vært en futuristisk visjon som folk har drømt om i flere tiår. De siste årene har det blitt gjort enorme fremskritt mot autonomi med introduksjonen av konvolusjonsnettverk og transformere i kombinasjon med eksponentiell forbedring av beregningskraft i maskiner. Den raske utviklingen innen datasyn har vekket interesse hos mange forskere, og er inspirasjonen bak denne oppgaven.

Arbeidet i denne oppgaven dreier seg om autonome kjøretøy, og da mer spesifikt 3D-objektdeteksjon ved hjelp av bilder og LiDAR-data. Modellen Frustum-PointPillars ble valgt basert på ytelse, sanntidsinferens og mulighet for dobbel sensordatainput. Målet med prosjektet var å gjenskape state-of-the-art resultatene til skaperne av modellen og skape en modell som kan operere i sanntid, samt er tilpasset Ouster LiDAR-sensorer. I det første kapittelet blir temaet og forskningsmålene grundig forklart. Deretter blir den nødvendige bakgrunnskunnskapen beskrevet i detalj, og relatert arbeid innen state-of-the-art 3D-objektdeteksjon blir utforsket med en grundig gjennomgang av ulike grunnleggende metoder.

Resultatene av denne avhandlingen viser at reproduksjonen av modellen ved å trene opp en ny modell fra bunnen av er mulig ved hjelp av en enkelt RTX 8000 GPU. Modellen fungerer bra på KITTI-dataene som ble brukt til trening, men den sliter noe med data den ikke har blitt trent på. Resultatene for biler og syklister overgår forfatterens resultater med 37,5% færre treningsrunder. Inferenstiden til modellen er mer enn to ganger raskere enn gjennomsnittlig menneskelig reaksjonstid, noe som gjør den egnet for implementering i autonome kjøretøy.

Konklusjonen om hvorfor modellen ikke presterer optimalt på Ouster-data er mangel på annoterte data. Prosessen med å annotere 2D- og 3D-data fra samme scene er tidkrevende når det er begrenset tilgang til gratis annoteringsverktøy. Resultatene ville trolig vært bedre med finjustering på mer lignende data.

Preface

I am delighted to show my final work at the Department of Computer Science at NTNU in Trondheim through this masters thesis. The beginning of the semester felt overwhelming because of the amount of work needed to be done, but after 5 months of hard work, I can proudly say that I am finished and feel grateful for the experience.

I would like to thank my supervisor Frank Lindseth and co-supervisors Gabriel Kiss and Durga Bavirisetti for guiding me through this project. I also want to thank friends, family and fellow students for taking interest in my work and exchanging ideas.

Contents

Figures

Tables

Acronyms

AOS average orientation similarity. 37

AV Autonomous Vehicle. 1, 5, 16

BEV Birds-eye view. xii, xiii, 16, 37, 39, 40, 42, 44–48, 51

CNN Convolutional neural network. 13, 16

FoV field of view. 39, 51, 52, 54

IoU intersection of union. xii, 8, 9, 27, 33–41, 49–51

mAP mean average precision. 9, 16, 19, 33–38, 49, 50

NLP natrual language processing. 9

ViT Vision transformer. 13, 14

Chapter 1

Introduction

1.1 Motivation

The evolution of Autonomous Vehicle (AV) technology is developing at a staggering pace, and is moving closer and closer to becoming a reality for every day that goes by. There are multiple sub-problems that are required to be solved in order to achieve autonomous behaviour. Firstly, the car must have the ability to perceive the environment in a correct manner to operate safely and not crash into objects. Secondly, the car must have appropriate planning-logic to properly plan its path around other agents in the environment. Lastly, there is the control-logic that needs to execute the planning appropriately in order to avoid collisions. There are also companies that are pursuing an end-to-end approach, where the system works like a black box, taking environmental signals through sensors and outputting control commands.

The implications of solving the problem of autonomous vehicles are tremendous both socially and economically. If vehicles are able to drive people to their destination with no human driver involved, it significantly reduces the cost of transportation and will revolutionize the taxi industry. No human driver implies lower expense per mile travelled, meaning that the competitive cost per mile will be reduced significantly. This will have an extraordinary effect on demand for the taxi services today as it is simply too expensive for most people. In addition, people with disabilities will have easier access to more convenient and affordable alternatives to public transport. Lastly, most of the accidents on roads today happen as a result of human error Treat *et al.* [1], and thus by replacing human drivers with computers that exhibit superhuman attention ability, less people will hopefully be involved in accidents.

There are many companies pursuing solving this problem, and it hard to determine who is leading the AV race as there are fundamentally different approaches to solving the problem of perception. GM's Cruise project[1] has successfully deployed autonomous vehicles in geo-fenced areas in San Francisco, however they

[1] `https://getcruise.com`

are scaling very slowly as the sensors and chips inside each self-driving car are expensive and the usable areas are thoroughly tested. Cruise is relying on LiDAR HD-maps to pinpoint the position of the vehicle, which also makes the roll-out slower. Other companies like Tesla and MobileEye has taken another approach where they rely solely on cameras to perceive the environment. Even though this method is more scalable in theory because it requires less hardware, it is yet to be proven reliable enough to be able to cut out the driver and achieve complete autonomous status. There are still many hurdles to overcome for both approaches in order to achieve scalability with appropriate safety levels.

In 2017 a paper called "Attention is all you need" was released by Vaswani *et al.* [2] which took the nlp community by storm. It was a new neural network architecture that was scalable and more precise than previous methods. In 2020 Dosovitskiy *et al.* [3] released a paper where they had successfully adapted the original transformer architecture to work with images. This new architecture in combination with CNNs is the fundamental cornerstone of 3D object detection as of early 2023.

1.2 Goal and Research Questions

In this research project, the main goal is to train a state of the art 3D object detection model to identify objects and locate their position in 3D space using LiDAR data and images. Ideally, the model should have low enough inference time to be used in an autonomous driving setting. The project should result in a pipeline that is able to detect a set of objects and reconstruct their position in 3D space in real-time. This pipeline is also going to be adapted to Ouster 360 degree data.

Research question 0: Are state of the art frustum 3D detection methods reproducible?

Research question 1: How well does a dual input 3D object detection model perform with 3-channel LiDAR images containing signal, reflectivity and near-IR combined with LiDAR point cloud data?

Research question 2: Is it feasible to use a multi-stage model with dual 2D and 3D input in an autonomous setting?

1.3 Research method

This project mainly revolves around experimenting with LiDAR data combined with object detection algorithms to get a working prototype that performs 2D to 3D reconstruction and object localization. The main method of evaluating the results is qualitative and will be based on visualizations of 2D and 3D spaces.

1.4 Contributions

The main contribution of this project is research on object detection models on point clouds to 3D space with different types of data. Different methods of detecting and localizing objects in 3D space will be examined and evaluated in terms of efficiency, accuracy and visual inspection. A new method for 3D object detection is proposed by combining multiple LiDAR sensor signals into a multi-channel image. The method will be tested and results will be documented. The objects in focus are mainly relevant for autonomous vehicle applications, but can also be of value for other tasks. Another contribution from this thesis is the research on reproducing and modifying state of the art 3D detection models to fit other sensory input methods.

Chapter 2

Background and Related Work

In this chapter, necessary background material and related work will be presented to give a clear understanding of the existing work in the field, and which areas need more research. We will give a short intro to LiDAR and a recap of deep learning, computer vision, transformers and current state of the art 3D object detection methods using LiDAR data.

2.1 LiDAR

LiDAR is a sensor that utilizes optical measurements to determine the distance to objects. It operates by emitting light and measuring the time delay and frequency shift of the light that returns after striking an object. The measurements obtained from the LiDAR provide information about range, intensity, reflectivity, and ambient near-infrared. The spherical projection of the LiDAR data is represented by assigning the value of each measurement as a pixel value in a two-dimensional image. The different channels of the LiDAR image capture various information, such as the range image that represents the distance to the objects, the signal image that measures the number of photons detected upon hitting an object, the near-infrared image that measures ambient light, and the reflectivity image that adjusts the photon measurement according to the range to produce consistent reflectivity measurements. The LiDAR sensor fits well in AV technology as it is versatile and can capture detailed distance measurements in a 3D-grid. In this thesis we will explore different state of the art 3D detection methods using the LiDAR range measurement data fused with other channel information.

2.2 Deep learning

Deep learning is a type of machine learning that involves using neural networks to learn from data. It is called "deep" because the neural networks that are used have many layers, allowing them to learn complex patterns in the data. In deep learning, the neural network is trained on a large dataset, which it uses to learn

the underlying patterns and structures in the data. This is done by adjusting the weights and biases of the network's individual neurons in order to minimize the error between the predicted output and the actual output. Once the neural network has been trained, it can be used to make predictions on new data. For example, a deep learning model trained on images of animals could be used to classify new images as belonging to one of several different animal classes.

One of the key advantages of deep learning is that it can automatically learn features from the data, rather than having them handcrafted by the user. This makes it well-suited to complex problems where the relationships between different variables may be difficult to define. This makes deep learning is a powerful tool for making sense of complex data and making predictions based on that data. It has been applied to a wide range of problems, including image recognition, natural language processing, decision making and much more.

2.2.1 Perceptron

A perceptron is a type of artificial neuron that is often used in the building blocks of neural networks. A perceptron takes in multiple inputs, performs a linear combination of those inputs using weights, and then gives some output. In other words, a perceptron can be thought of as a simple linear classifier. It takes in a set of inputs, multiplies each input by a corresponding weight, and then outputs a value based on the weighted sum, as expressed in the formula below.

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

Where z is the output, w_i is the weight, x_i is the input and b is the bias

2.2.2 Neural network

Neural networks are connected layers consisting of perceptrons. It takes some input, then performs operations sequentially through hidden layers and finally get a output through the output layer. Both inputs and outputs can have many different sizes and shapes, which makes the neural network architecture adaptable for many problems. The strength of the connections between the perceptrons are called weights, and determines how the inputs are transformed as they move through the network. During the training process, the weights of the connections between the perceptrons are adjusted in order to minimize the error between the predicted output and the actual target. This allows the neural network to learn from the data and improve its predictions over time.

2.2.3 Forward Pass, Backpropagation and Activation Functions

The forward pass is the process of using a neural network to make predictions on a set of inputs. It involves passing the inputs through the various layers of the

network, using the weights of the connections between the neurons to transform the inputs as they move through the network. This results in an output prediction from the network.

Backpropagation is the process of adjusting the weights of the connections in a neural network in order to minimize the error between the predicted output and the actual output. This process is performed after the forward pass. It involves calculating the error at the output layer and then propagating that error by its gradients back through the network, then adjusting the weights in each layer in a way that reduces the overall error.

Activation functions are used to transform aggregated and dotted weights, inputs and biases between layers of a neural network. They are typically non-linear functions that take in the weighted sum of the layer output, then output a value within a certain range based on the activation used. Some of the most known activation functions are Sigmoid, the rectified linear unit (ReLU) and the Softmax.

$$Sigmoid(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{2.2}$$

$$Relu(x) = max(0, x) \tag{2.3}$$

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.4}$$

In general, the forward pass, backpropagation, and activation functions are all important components of a neural network that combined creates a robust learning algorithm.

2.3 Computer Vision

The study of computer vision aims to enable computers to comprehend and interpret visual information from their environment. It is achieved by designing algorithms and systems that can automatically analyze and comprehend images and videos to extract relevant information. There are many sub-fields within computer vision, called tasks. These tasks include object detection, semantic segmentation, image classification, image generation and many more. The task relevant for this project is object detection in 2D and 3D spaces.

2.3.1 Object detection

Object detection is the task of identifying objects in images or videos and localizing them with a bounding box. This bounding box is typically rectangular in 2D-object detection, and cubical in 3D-object detection. Usually the bounding box is represented as coordinates of for example top-left corner and bottom-right corner of a rectangle.

2.3.2 Evaluation metrics

When evaluating the performance of an object detection method, evaluation metrics are needed. In image classification, each image has its own label or labels, however when dealing with objects, there can be multiple within a single image and the location needs to be specified. This makes evaluation a little bit harder, because we need to know where each object is localized and what type of object it is. The main evaluation metrics of object detection will be explained in detail in this section.

TP, FP, TN, FN When a model makes a prediction, we assign a true or false label to the prediction based on what the actual label of the class is. For example, if the model predicts a positive sample and the ground truth for the same sample is positive, then we say the model predicted a true positive. However if the model predicted negative on that very same sample, we would call the prediction a false negative. The table below shows the naming scheme in more detail.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

intersection of union (IoU) The IoU is a metric commonly used to evaluate the performance of object detection algorithms. It measures the overlap between the predicted bounding box and the ground-truth bounding box for an object, and is calculated as the ratio of the area of overlap to the area of union.

The IoU is typically used in object detection as a threshold for deciding whether a predicted bounding box is a true positive or a false positive. A predicted bounding box is considered a true positive if its IoU with the ground-truth bounding box is above a certain threshold, typically around 0.5. If the IoU is below this threshold, the predicted bounding box is considered a false positive.

In addition to being used as a threshold for true and false positives, the IoU can also be used as a measure of the overall performance of an object detection algorithm. In this case, the average IoU across all predicted bounding boxes is computed, and higher values indicate better performance. It is worth noting that it does not measure the classes within each image, but rather how well the bounding boxes fit the objects.

Precision and Recall Precision and recall are two commonly used metrics for evaluating the performance of a classification model that is predicting discrete classes. The model can have advantages and disadvantages based on which metric it is optimized for, and thus having multiple metrics that measure different performance result in a more robust overview of the predictive strengths and weaknesses of the model.

Precision refers to the proportion of true positive predictions out of all positive predictions made by the model, meaning true positives divided by true and false

positives. Precision gives us a ratio that describes how well the model performs on all positive samples.

Recall, also known as sensitivity or true positive rate, refers to the proportion of true positive predictions out of all actual positive instances. This is found by taking true positives and divide by true positives and false negatives, meaning we find how many positive samples were found out of all positive samples.

$$Precision = \frac{TruePositives}{TruePositives+FalsePositives}$$
$$Recall = \frac{TruePositives}{TruePositives+FalseNegatives}$$

mean average precision (mAP) mAP is a metric used to evaluate the performance of object detection algorithms. It is a measure of the overall precision of the algorithm, and is calculated as the average of the precision at different IoU thresholds. Higher values of mAP indicate better performance

To calculate mAP, the object detection algorithm is first run on a set of images, and the predicted bounding boxes are compared to the ground-truth bounding boxes for the objects in the images. For each predicted bounding box, the IoU with the ground-truth bounding box is computed.

Next, the mAP is calculated by computing the precision at a range of IoU thresholds, from 0 to 1. The precision at each threshold is defined as the number of true positives divided by the total number of predicted bounding boxes at that threshold. The mAP is then calculated as the average of the precision at all of the different IoU thresholds.

2.4 Transformers

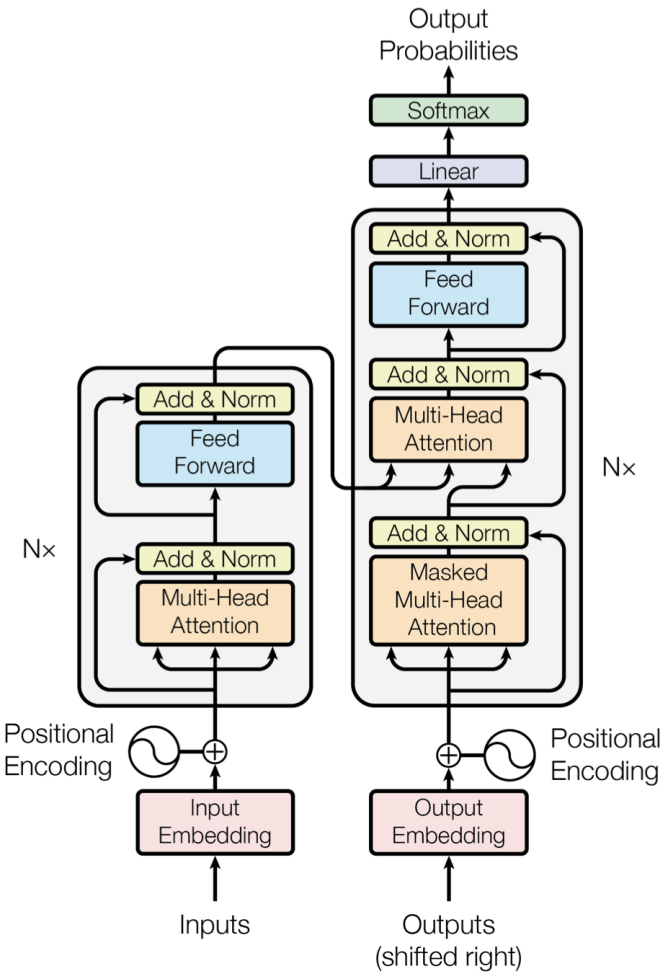
A transformer is a type of neural network architecture that mainly has been used in natural language processing tasks up until recently. It was introduced in a paper by researchers at Google in 2017 Vaswani *et al.* [2] and has since become one of the most widely used models in the field. One of the most well known transformers is the GPT-3 Brown *et al.* [4] model by Open AI which has around 175 billion parameters, which is absolutely massive and requires huge GPU clusters in order to train within reasonable time-frames. These large models are capable of performing well on a range of NLP tasks. One of the most controversial transformer models as of recent is GitHubs Copilot Chen *et al.* [5], which is designed to assist software engineers in writing code, but has gotten some backlash because of licenses and ethical concerns.

2.4.1 Architecture

At a high level, the transformer architecture consists of an encoder and a decoder, where the encoder takes an input and creates an embedding, then sends it to the decoder which decodes the embedding and outputs the transformed input. The input and output can be altered based on which specific task the network is being trained for, by changing the shape of the layers. Both the encoder and

decoder consists of multi-head attention blocks, feed-forward layers and batch normalization layers.

Figure 2.1: Architecture of the original transformer model Vaswani *et al.* [2]



Tokenization and Positional Encoding

The input requires tokenization before being passed to the encoder or decoder, as the transformer model cannot interpret words represented as letters in sequences. Tokenization is the process of converting words to transformer-readable tokens in the form of integers. After creating token representations for the words, positional embeddings are created to store the position of the words in the text in order to preserve contextual information. This step is crucial to make the model understand language, because permutations of word positions of the same sentence can result in different meanings. The positional encoding is added as a sinusoidal

vector to the input token with equal dimensionality to make sure that the length of the positional encoding does not exceed the length of the tokenized input vector

Encoder

In the transformer, the encoder is responsible for converting the input words into a set of learned representations, called embeddings. This is done by passing the pre-processed input tokens through a series of blocks containing multi-head attention and feed-forward layers. The encoder is crucial for transforming text into dense vector embeddings with different sets of meanings in a concise and separable representation.

Decoder

The decoder is used to generate the output sequence of words. It does so by using the learned embeddings from the encoder as inputs, and passing them through a series of layers that use self-attention to consider the relationships between the words in the output sequence. By passing the information of the relationships to a feed-forward layer with a Softmax activation, probability distributions are generated and can be used for text generation or classification tasks.

2.4.2 Attention

In the context of transformers, attention refers to the mechanism that allows the model to directly consider the relationships between all tokens of words in a sentence, rather than just considering each token in isolation. This allows the model to capture long-range dependencies in the data and make more accurate predictions.

The attention mechanism in the transformer architecture works by calculating the "attention scores" between all pairs of tokens in the input and output sequences. These scores reflect the degree to which each output token should be influenced by each input token.

The attention scores are then used to compute a weighted sum of the input tokens, with each input token receiving a weight proportional to its attention score. This weighted sum is then used as input to the next layer of the network, allowing the model to consider the relationships between the tokens in the input sequence when making predictions. attention is an important component of the transformer architecture and is a key factor in its ability to capture long-range dependencies in natural language data.

It turns out this attention mechanism can be implemented in parallel, which then allows us to create "multi-head attention" that looks at multiple relationships between tokens at once, and thus can extract more information.

Scaled dot product

As mentioned, attention works by calculating the "attention scores" between all pairs of tokens in the input and output sequences. This operation is called the scaled dot product

It is performed by doing a dot product between the embeddings of each pair of input and output tokens. The dot product can be viewed as a measure of the similarity between the two vectors, as it is the sum of the element-wise product of the vectors.

However, the dot product can sometimes have very large values, which can lead to numerical instability in the model. To prevent this, the transformer scales the dot product by dividing it by the square root of the dimensionality of the embedding vectors. This has the effect of limiting the range of the attention scores and making the model more stable.

Once the attention scores have been calculated, they are used to compute a weighted sum of the input words, known as the "context vector". This is done by first splitting the input sequence into three separate vectors: the "keys", the "values", and the "queries". The keys and values are both fixed and are determined by the encoder, while the queries are determined by the decoder. The queries can be viewed as the translation part, where a query can be mapped to a specific key-value pair based on the task to be solved.

The attention scores are then calculated by taking the dot product of the queries and the keys, and then passing the result through a softmax function to normalize the scores. The context vector is then computed by taking the element-wise product of the values and the normalized scores, and then summing the resulting vectors. The scaled dot product is expressed as

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.5}$$

where Q is the matrix of query vectors, K is the matrix of key vectors, V is the matrix of value vectors, and d_k is the dimensionality of the key vectors.

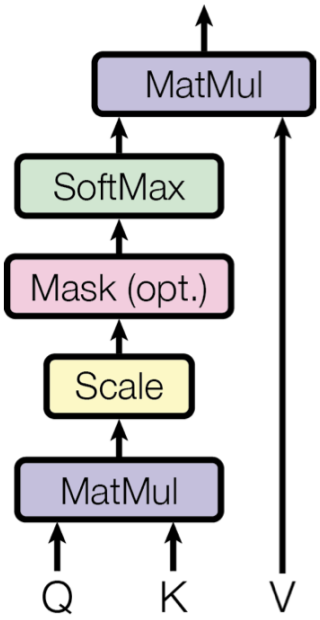
Multi-head attention

As mentioned multi-head attention is a mechanism that allows the model to attend to different parts of the input simultaneously and in parallel. This allows the model to learn multiple different relationships between the input tokens, which can help improve its performance.

To calculate multi-head attention, the transformer first splits the input sequence into three separate vectors: the "keys", the "values", and the "queries", just like explained in the previous paragraph.

Next, the transformer computes multiple dot products between the queries and the keys, one for each of the different attention "heads". This results in a set of attention scores for each head, where each score reflects the degree to which each

Figure 2.2: Scaled dot product flow diagram Vaswani *et al.* [2]



output token should be influenced by each input token. The scores are then scaled and softmaxed the same way as mentioned in the "Scaled dot product paragraph".

Finally, the transformer computes a weighted sum of the values, using the normalized attention scores from each head as the weights. This results in a set of context vectors, one for each attention head. These context vectors are then used as inputs to the next layer of the network, allowing the model to consider multiple of the relationships between the tokens in the input sequence when making predictions.

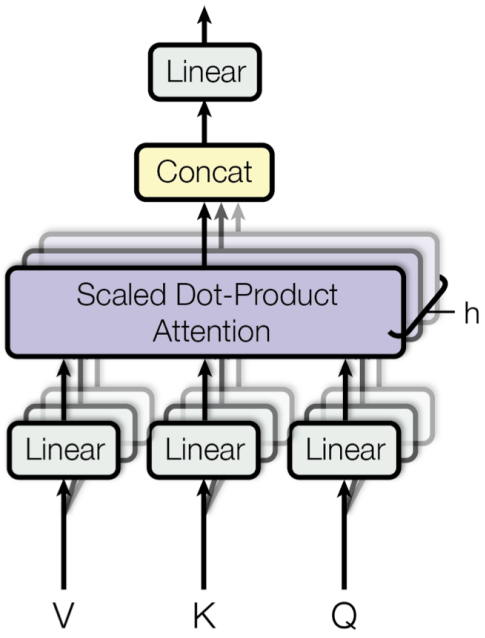
2.4.3 Vision Transformers

A Vision Transformer is a type of deep learning architecture that uses the transformer architecture to process visual data such as images. It combines the advantages of CNNs for image processing and the transformer architecture for handling sequential data. Consequently, vision transformers are highly effective at tasks such as image classification, object detection, and semantic segmentation.

Vanilla transformer vs ViT

ViTs and regular transformers are similar in that they both use the transformer architecture as showed in section 2.1. In spite of that, there are several differences in implementation details of the architecture between the two that are crucial to what task is being solved.

Figure 2.3: Multi-head self-attention Vaswani *et al.* [2]



Input Representation Vanilla transformers are typically used for processing sequential data such as text, whereas ViTs are designed to process visual data. Thus, the input representation for ViTs is often large as the spatial resolution of images is sizeable compared to text. As seen in figure 2.4, the input image is divided into patches, flattened and vector encoded with positional embeddings before being fed into the transformer encoder block.

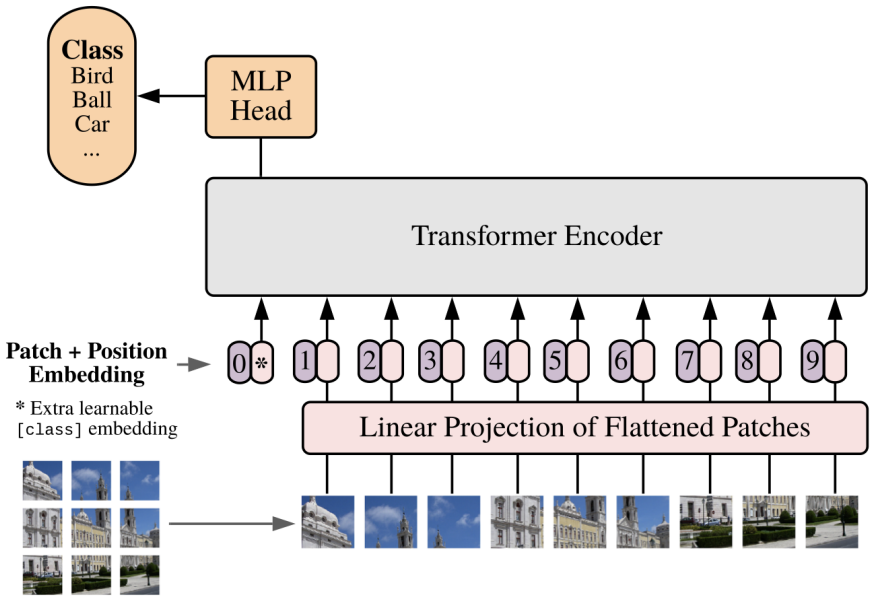
Feature Extraction Vanilla Transformers typically operate on a vector representation of the input data, whereas ViTs can use convolutional layers or other methods to extract image features before feeding them into the transformer. Text input on the other hand can have other forms of pre-processing like stemming, lemmatization, rephrasing or lower-casing to mention a few.

Architecture While both vanilla transformers and ViTs use the core components of the transformer architecture such as the attention mechanism and the feed-forward network, the exact architecture of the two types of models can differ. For example, ViTs may use a more complex encoder to extract image features, and a more complex output layer to handle the specific task (e.g. image classification, object detection, semantic segmentation).

Training Data Vanilla Transformers are typically trained on large amounts of text data, whereas Vision Transformers are trained on image data. This can have

an impact on the choice of loss function and the type of data augmentation used during training.

Figure 2.4: Illustration of the vanilla vision transformer architecture. Image from [3]



2.5 Recent advances in 3D Object Detection

3D object detection is a crucial task in a variety of applications operating in the real world, such as robotics, augmented reality, and autonomous vehicles. In recent years, significant progress has been made in this field thanks to the development of powerful deep learning algorithms and the availability of large-scale 3D datasets. In this section, we provide an overview of different data representation methodologies and state-of-the-art methods for 3D object detection.

2.5.1 Different approaches to 3D object detection

One of the main differentiating factors when it comes to methods for 3D object detection is what kind of sensor is used. Fernandes *et al.* [6] did a survey where they reviewed the different approaches in state of the art 3D object detection and found that the most commonly used sensors are LiDAR and cameras as a result of their compelling information gathering properties. In this section we will review different methods within camera-based, LiDAR-based and fusion between the two.

3D object detection from 2D images

One of the most novel methods and maybe the holy grail of AV is 3D object detection through cameras only. As cameras are relatively cheap compared to other sensors like LiDAR, a bigger opportunity for mass marked adoption is possible because of economic efficiencies. In 2017 Mousavian *et al.* [7] used RGB images to estimate 2D bounding boxes by using a CNN architecture, then by adding geometric constraints on the given 2D prediction, they were able to predict the 3D bounding box of given objects. This method served as a proof of concept and led to further research on how to improve upon the results. Later Li *et al.* [8] showed that single image 3D detection can be more reliable by using the 2D-detection for search pruning, then further analyze surfaces within each given 2D bounding box with CNNs. By pruning the search, more compute can be used for analyzing surfaces, and thus more information about angles, direction and size can be incorporated into the 3D bounding box prediction.

Mottaghi *et al.* [9] and Zhu *et al.* [10] tried using 3D models with clustering of similar 3D models in order to predict the 3D bounding box with corresponding direction and size of objects. This method relies heavily on using shape and geometric cues for comparison with the given object. A drawback of this method is the lack of 3D models compared to the rich complexity of real-world data.

More recently, Li *et al.* [11] and Wang *et al.* [12] have improved image only 3D object detection by using stereo imagery. Wang *et al.* [12] trained a CNN to reproduce LiDAR signals, effectively creating a pseudo LiDAR that improved upon state of the art 3D object detection results by 22% on the KITTI benchmark within ranges of 30 meters. There is currently an explosion in the amount of papers researching this subtopic and additional improvements to the capabilities of 3D bounding box detection by only using 2D images from cameras will probably happen in rapid succession.

3D object detection from point clouds

LiDAR sensors are expensive, but with the rapid technological development in light detection and ranging sensors in recent years, the cost is clearly trending down. As a result, research conducted regarding point cloud 3D detection is surging and thus the advancements in the state of the art escalate.

One of the earliest methods for 3D detection using point cloud data is transforming the scene to BEV and using a 2D CNN detection backbone. Chen *et al.* [13], Ku *et al.* [14], Yang *et al.* [15] and Yin *et al.* [16] used 2D BEV for 3D detection and orientation refinement. It is worth noting that transforming point clouds to BEV and using 2D detection alone is not enough to compete with state of the art methods in mAP. However, computational load is significantly decreased by reducing dimensional complexity in the point cloud from 3D to 2D. Therefore adding BEV to the bounding box detection and orientation refinement pipeline seems like a good trade-off when comparing inference and training time to output in mAP.

Qi *et al.* [17] introduced PointNet in 2016, which takes raw point cloud data

as input and gives 3D bounding box predictions as output. It was one of the first methods that did operations on point cloud data and was able to discard some pre-processing techniques earlier introduced. One of the advantages with processing 3D point cloud data directly is that more information about the 3D structures can be extracted and thus the model becomes more accurate. Even though PointNet achieved state of the art results at the time, it had some flaws regarding capturing local structures. Qi *et al.* [18] later improved upon the PointNet architecture with hierarchical feature learning in PointNet++.

One downside to operating on each point in the point cloud individually is that it creates large computational complexity and makes real-time application harder to accomplish. Zhou and Tuzel [19] proposed a voxel-based method that divides the point cloud into a 3D-grid of voxels, where each point belongs to a voxel potentially containing multiple points instead of existing individually. This allows the feature extraction backbone to work on grouped data and consequently reduces computational complexity while achieving state of the art results at the time.

2.5.2 Data representation and extraction of point clouds

One of the key challenges in 3D object detection is the efficient representation of 3D point clouds. Early works in this field typically used hand-crafted features such as depth and geometric shapes to represent 3D objects. However, these methods were not able to capture the rich and complex structure of 3D objects and limited their performance.

Point-based

As discussed in 2.5.1, point-based methods are using raw point clouds as input to the feature learning backbone. They then produce a sparse representation of the scene that can be parsed by a neural network. The points are usually represented by a vector containing its local neighbor points. As seen in figure 2.5, the global information flow happens after multiple of these points are pooled in multiple layers, creating a larger receptive field.

Voxel-based

A voxel is a three-dimensional grid element that represents a specific value in a 3D space. Voxel-based methods divide the point clouds into regularly spaced voxels in the Cartesian coordinate system. This division enables the application of feature learning to extract features from groups of points within each voxel, resulting in dimensionality reduction and memory savings. As seen in figure 2.6, Zhou and Tuzel [19] uses random sampling after dividing points into voxels in order to generate a more compact representation of the points while maintaining decent accuracy.

Figure 2.5: Illustration of the hierarchical feature extraction from single points. Image from [17]

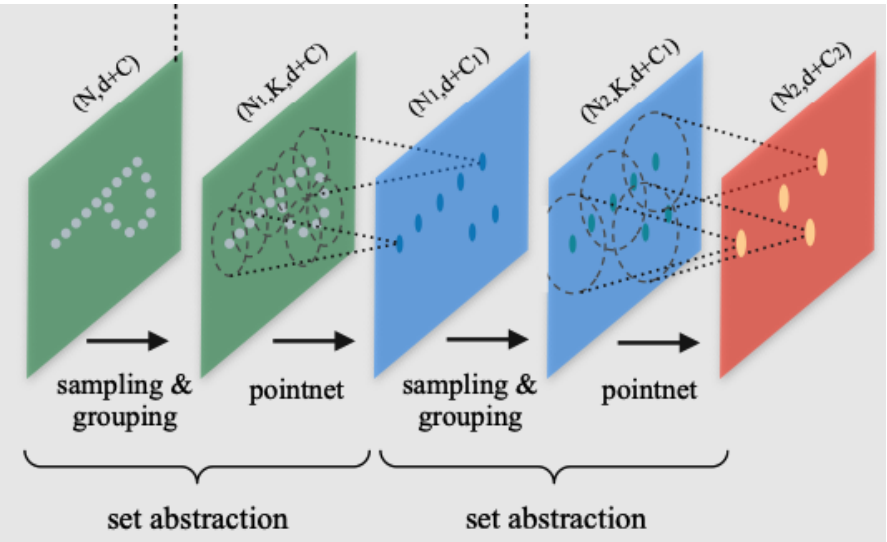
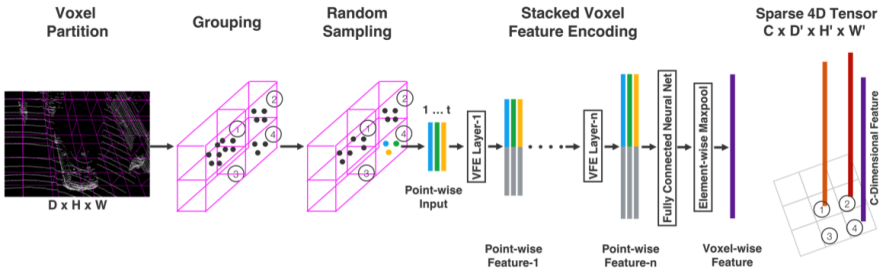


Figure 2.6: Illustration of the voxelization of points for feature extraction. Image from [19]



Frustum-based

Frustums are parts of a cone or pyramidical structure that is cut off by a plane parallel to its bottom plane. The core idea of frustum-based representation and feature extraction is to limit object proposal detection to two dimensions in order to save memory and compute. After a object is detected in 2D space, a frustum is generated in the point cloud based on the coordinates of the 2D bounding box, as seen in figure 2.7. This frustum is then used to separate the relevant and irrelevant points regarding the specific 3D object by instance segmentation. A 3D feature extraction backbone does further unravelling of the 3D structure of points in order to encode important geometric cues for the detection module to learn. Figure 2.8 shows the explained 2D proposal to 3D instance segmentation pipeline in more detail.

Figure 2.7: Visualisation of the frustum-based search pruning for feature extraction. Image from [20]

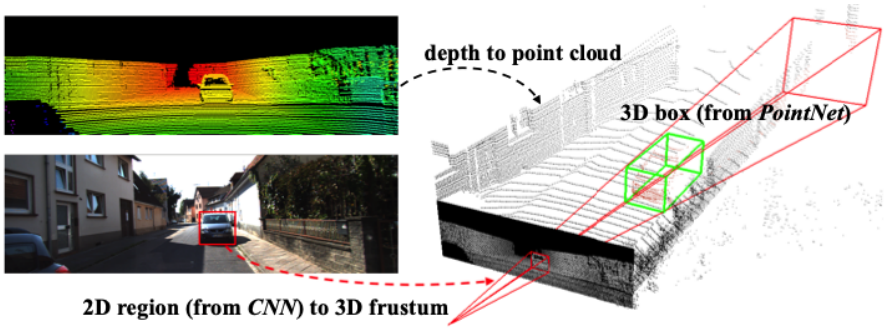
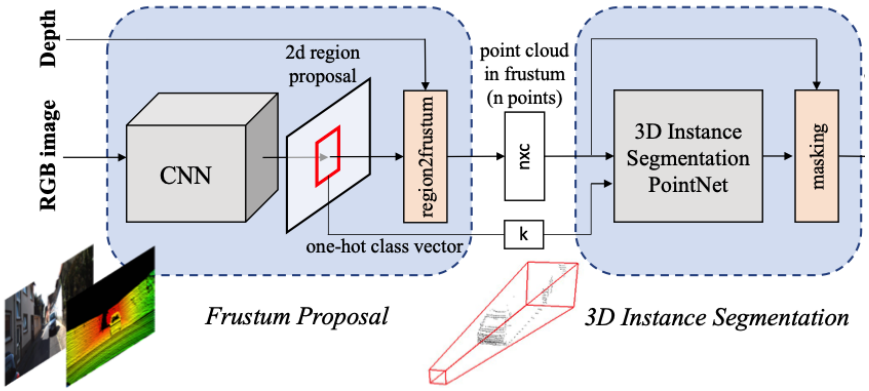


Figure 2.8: Visualisation of the 2D detection to frustum-based 3D detection for feature extraction. Image from [20]

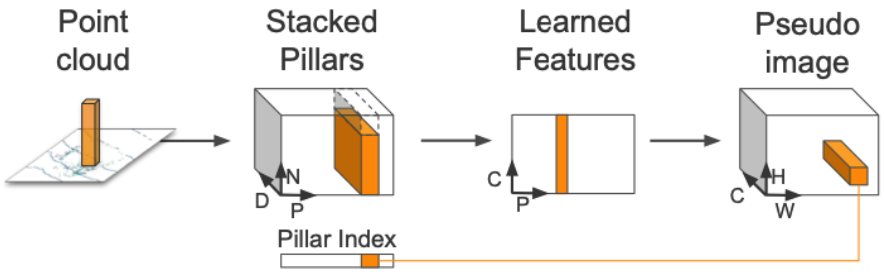


Pillar-based

The pillar-based method is similar to the voxel-based method as both designates points to voxels in 3D-space. The key differentiating factor is that pillar-based methods disregard the limitation on the z-axis of each voxel, resulting in long vertical pillars with fixed length x-y plane. Lang *et al.* [21] proposed PointPillars in 2019, which matched or exceeded state of the art mAP on different objects like cars, pedestrians and cyclists while simultaneously achieving 40Hz faster run-time. Stanisz *et al.* [22] further optimized PointPillars to run on a smaller FPGA device, decreasing mAP 5-9% while reducing the model size almost 16-fold, making real-world application possible.

In figure 2.9 we can see an illustration of pillar generation, concatenation and 2D-pseudo image creation in the feature extraction process.

Figure 2.9: Illustration of the pillar-based pipeline for feature extraction in PointPillars. Image from [21]



Chapter 3

Method

3.1 Hardware

The hardware used in this project were a Macbook Pro 2018 with Intel processor and a Ubuntu virtual machine with 6 CPUs, 32GB RAM and RTX8000 with 8GB VRAM. The VM was running Ubuntu version 20.04 and was virtually connected through VMware Horizon Client. Access to NTNUs IDUN-cluster was also available, however, the model did not support multi-GPU training or inference, so it was not utilized.

3.2 Datasets

3.2.1 KITTI-360

The KITTI-360 dataset for 3D object detection contains 7481 training images and 7518 test images and their corresponding velodyne point clouds. This makes a total of 80 256 labeled objects. The dataset is commonly referenced in state of the art papers regarding 3D object detection benchmarks since it was released in 2017.

Download and Preparation

The KITTI-360 dataset was downloaded from the official KITTI Vision Benchmark Suite[1] under the 3D objects tag. RGB images with their respective velodyne point clouds, labels and calibration matrices were downloaded. The size of the files can be seen in table 3.1. For preparation, the calibration file was used in order to match the RGB scene with the point clouds. The calibration file contains matrices that describe the needed translation, cut-off, yaw and rotation of each scene in order to synchronize the images with the point clouds.

[1] https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d

Table 3.1: Memory usage for different data types

File name	File size
RGB	12GB
RGB preceding frames	36GB
Velodyne point clouds	29GB
Camera calibration matrices	15MB
Training labels	5MB
TOTAL	**∼78GB**

Objects

There are 8 different types of objects within the KITTI-360 dataset for 3D object detection, which includes car, van, truck, standing pedestrian, sitting pedestrian, cyclist, tram and miscellaneous. In figure 3.1 we can see that there is a major skew towards car instances within the dataset which is to be expected as the data is captured on roads. We also notice that ∼ 38% of the cars in the dataset are occluded and ∼ 10% are truncated.

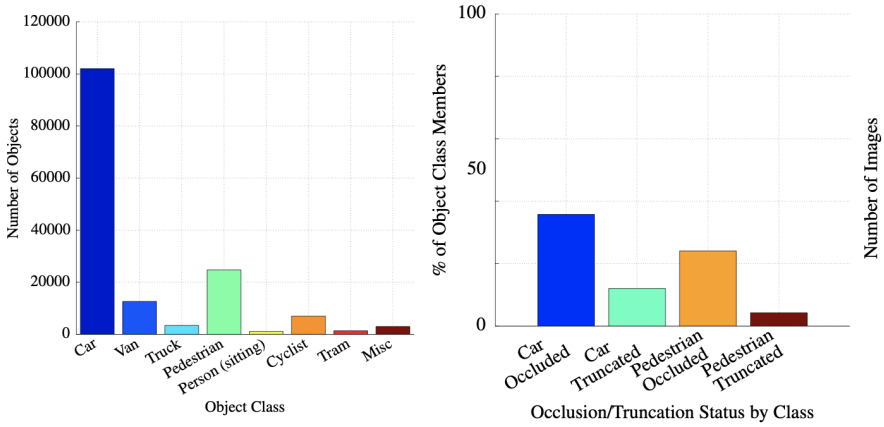


Figure 3.1: Number of object instances in KITTI dataset. Source [23]

In figure 3.2 we can see that there is most commonly 1 object per class in each individual image. For pedestrians there are rarely more than 1 instance within an image. That is predominantly the case for car instances as well, though to a lesser degree as the 2+ cars within an image cumulatively make up more than 50% of the total amount of images containing cars.

Size and orientation

Information about the height, width and length of objects is important for anchor selection in neural networks and our model which we will get back to in section 3.3. The KITTI dataset contains what resembles normal distributions as seen in

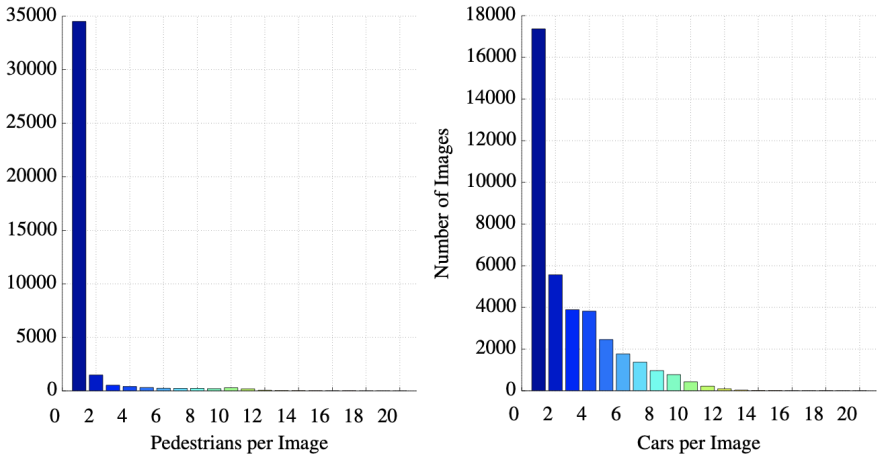


Figure 3.2: Distribution of number of object-instances per image for car and pedestrian in KITTI dataset. Source [23]

figure 3.3 which shows the distribution for height, width and length for cars and pedestrians.

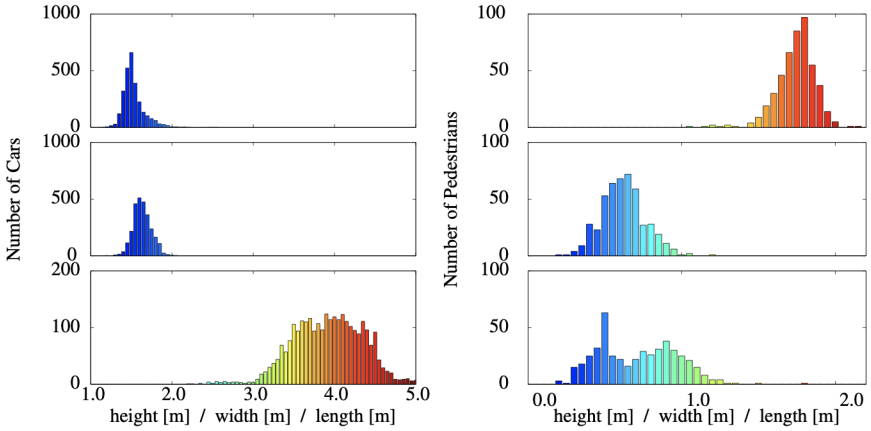


Figure 3.3: Distribution of height, width and length for cars and pedestrians in the KITTI dataset. Source [23]

As for the orientation degree of different objects, the distribution skews more towards 0° and 180° which can be seen in figure 3.4 where we can see the orientation distribution of cars and pedestrians. This is to be expected as the data is captured from a driving vehicle where the road system is designed for parallel traffic flow for vehicles and pedestrians.

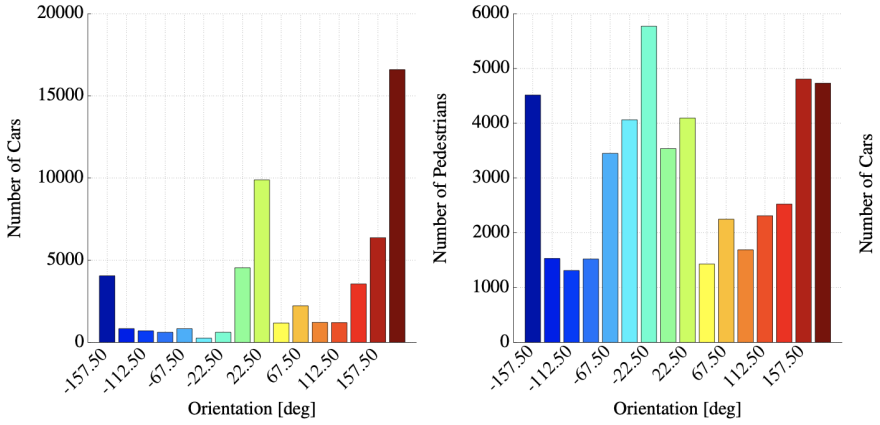


Figure 3.4: Distribution of orientations for objects in KITTI dataset. Source [23]

3.2.2 Ouster data

Ouster data was used to see if the model has learned general point cloud features for cars, cyclists and pedestrians. Ouster has some sample data on their SDK tutorial page that was downloaded [2]. Since the Ouster sensor does not capture RGB images, a synthetic 3-channel LiDAR image containing reflection, signal and near-infrared was created as substitution.

Preparation

As the Ouster data is using PCAP format, we had to unpack the point cloud and reflection data and convert it into binary format. Each file has a corresponding info-file that contains labels, bounding boxes, truncation, occlusion, location etc as seen in figure 3.5. Since the objects in the Ouster data was not labeled in either 2D or 3D, we had to manually label each object. This was conducted through changing the height, length and width of each object and its coordinate location with x, y and z in the label text file and visually inspecting the changes with the KITTI-VIS tool.

After creating the label-file, a corresponding pickle info-file which contains metadata, calibration data and object data needs to be created. All items in the pickle file is described in table **??**. The file is created as a dictionary

The entire labeling pipeline is shown in figure 3.6.

Prediction

In order to predict the prepared Ouster data, some modifications on the model pipeline needed to be made. Since the Ouster data is captured with 1:1 pixel correspondence between LiDAR and other measurement types, no direct calibration

[2] `https://static.ouster.dev/sensor-docs/#sample-data`

Table 3.2: Table showing the items in the pickle file with the respective descriptions.

Key	**Description**
image_idx	Index of image
pointcloud_num_features	Number of features in point cloud
velodyne_path	The path to the binary file containing the point cloud
img_path	The path to the image file
img_shape	The shape of the 2D image
calib/P0	Calibration matrix for camera 0 (reference cam)
calib/P1	Calibration matrix for camera 1 (not used)
calib/P2	Calibration matrix for camera 2 (not used)
calib/P3	Calibration matrix for camera 3 (not used)
calib/R0_rect	Rectification matrix for transforming P1-3 into the same image plane
calib/Tr_velo_to_cam	Matrix for transforming points to reference image coordinates
calib/Tr_imu_to_velo	Matrix for height angle adjustments
annos	**Annotation description**
name	Name of objects (array)
truncated	Truncation of objects (array)
occluded	Occlusion of objects (array)
alpha	Observation angle of objects (array)
bbox	2D bounding box of objects: x-min, y-min, x-max, y-max (2D array)
dimensions	Height, width and length of objects (2D array)
location	X, y and z coordinates of objects (2D array)
rotation_y	Objects rotation about y-axis (array)
score	Confidence of prediction (array)
index	Index of objects in current image (array)
group_ids	Group index of objects (array)
difficulty	Easy, moderate or hard difficulty of object (array)
num_points_in_gt	Number of points inside ground truth bounding box (array)

Figure 3.5: Table showing the format of the label file with corresponding values

Number of Values	Attribute	Description
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram','Misc' or 'DontCare'
1	truncation	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries
1	occlusion	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of the object, ranging [-pi..pi]
4	2D bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x,y,z in camera coordinates (in meters)
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

was necessary. Thus, frustums and were made through masking the ground truth 2D bounding boxes directly and applied to filter irrelevant points. Gaussian probability features were made by using the pixel image coordinates of the Ouster scan. After modifying the pipeline, the LiDAR and reflection measurements needed to be destaggered. For default staggered representations, each column represent a single timestamp. We use the Ouster SDK to destagger the representation, meaning now each column corresponds to a single azimuth angle, compensating for the azimuth offset of each beam which is captured by the sensor.

3.2.3 Setup

The model was cloned from the Frustum-PointPillar repository on GitHub [3]. The code was tested on Ubuntu 18.04 and only supports Python version 3.6, PyTorch 1.4+ and CUDA 11+. As a result, there were lots of bugs that needed to be sorted out, and lots of testing on different builds with different combinations of versions.

After fixing bugs and setting up the model, the KITTI dataset could be prepared for training the model. The dataset was setup in a file-structure as shown in figure 3.10. KITTI info files, a ground truth database and reduced point clouds were created for training. The reduced point clouds were created by filtering out all LiDAR points that are located outside the field of view of the 4 RGB-cameras.

3.2.4 Calibration

The KITTI data was captured by a couple separate sensors, namely 4 RGB cameras and a velodyne LiDAR sensor. Thus, there are multiple coordinate systems that need to be synchronized, as shown in figure 3.9. Camera 0, P0, was used as

[3]https://github.com/anshulpaigwar/Frustum-Pointpillars

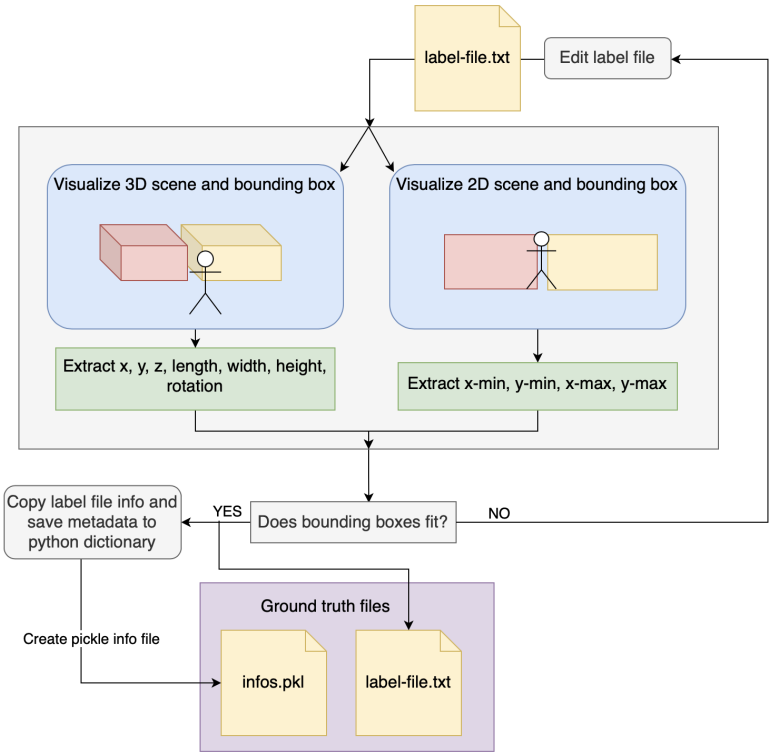


Figure 3.6: Visualization of manual labeling pipeline.

reference coordinate system. In order to locate the X-Y position in the reference coordinate system for each point in the velodyne point cloud; rectification, rotation and translation had to be performed. Each of the cameras were first rectified with a 3x4 rectification matrix, then rotated and translated with matrix P1-P3 with shape 3x4.

3.3 Frustum-PointPillar

3.3.1 Tuning

After configuring the file structure and pre-processing the KITTI data, tuning of the configuration files for the model was performed. The main parameters that were tuned were epochs, learning rate, learning rate decay, steps, batch size and anchor points. These parameters were tuned for 3 objects; car, pedestrian and cyclist. In order to track the training process, WandB was used to receive live updates for important metrics like precision and recall at IoU 10, 50, 95. Other metrics like total loss, bounding box loss and angle of orientation loss were also tracked and stored.

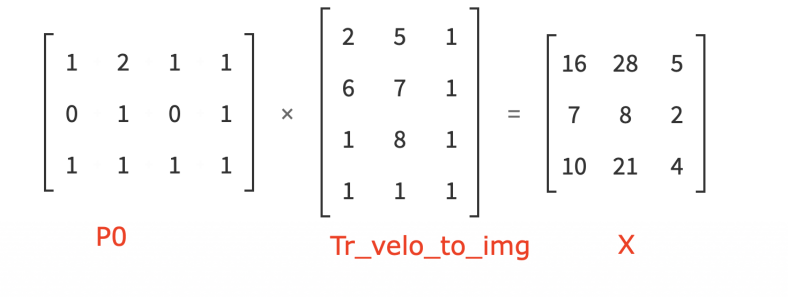


Figure 3.7: Visualisation of calibration to get matrix X from rotation/translation matrix P0 with shape 3X4 and translation from velodyne point cloud matrix with shape 4x3 to get X of shape 3x3

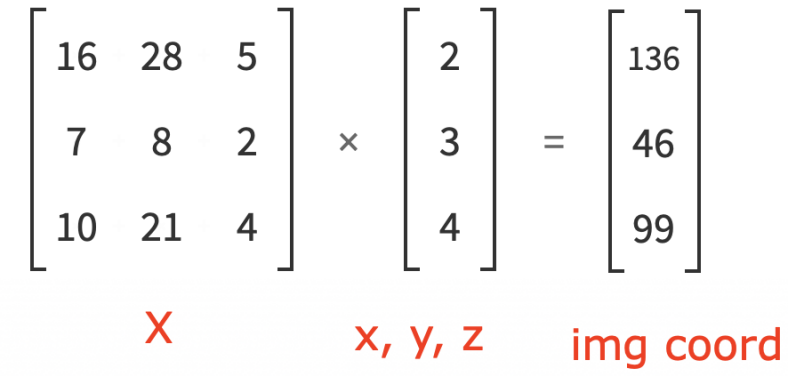


Figure 3.8: Visualisation of calibration matrix X from figure 3.7 and the x-y-z-coordinate in the point cloud to get image coordinate in reference coordinate system

3.3.2 Pipeline

The model pipeline is shown in figure 3.11. First, the point cloud is masked based on the field of view of the image. Secondly, 2D object detection is performed on a three-channel image to get bounding boxes of objects in the scene. Then the points located outside the frustums created by the bounding boxes are masked and removed. The reduced point cloud is then voxelized and 3D-point detection is conducted to get object label, location in x-y-z coordinates and height, width and depth with corresponding angle of orientation around the y-axis of the velodyne coordinate space.

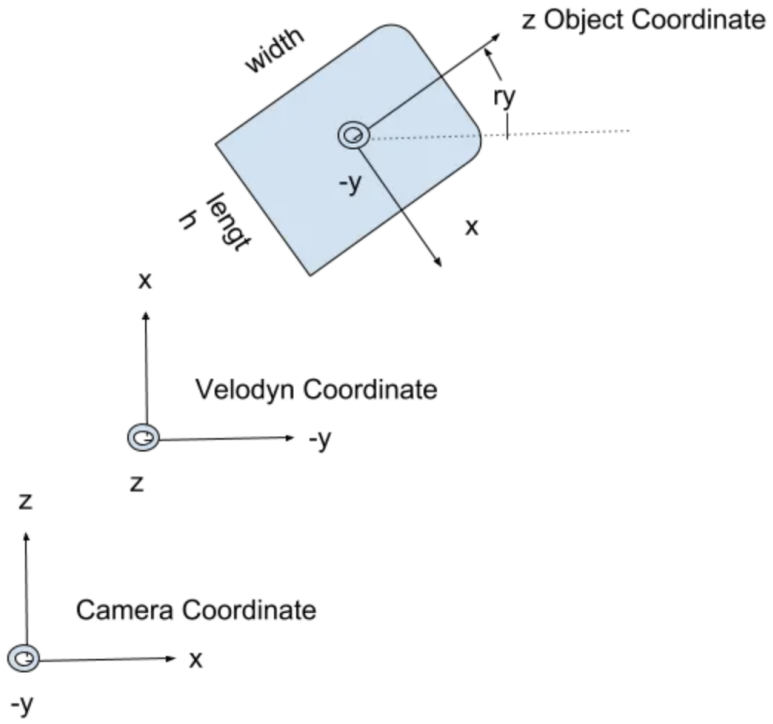


Figure 3.9: The different coordinate systems between RGB-cameras and LiDAR sensor and objects within both. Image source

3.4 KITTI visualization

The visualisation tool was cloned from the KITTI-object-visualisation repository on GitHub [4]. The repository only supports Python version 3.7 and was tested on Ubuntu version 18.04. My VM uses Ubuntu 20.04 and thus, implementation was not as smooth as hoped for. Some bugfixes were sorted out with the Mayavi visualisation tool, one of which required an older version for Ubuntu 20.04, while some smaller visual bugs are still not fixed. The tool shuts down sometimes and the reason why is unknown even after contacting the maintainer of the repository. Despite visual bugs, we managed to get some visualisations working in order to show the models performance more explicitly.

In order to visualize the results, the visualization tool needed a specific KITTI file structure with a symbolic link to the predictions made by the model, as shown in figure 3.12

[4]https://github.com/kuixu/kitti$_o bject_v is$

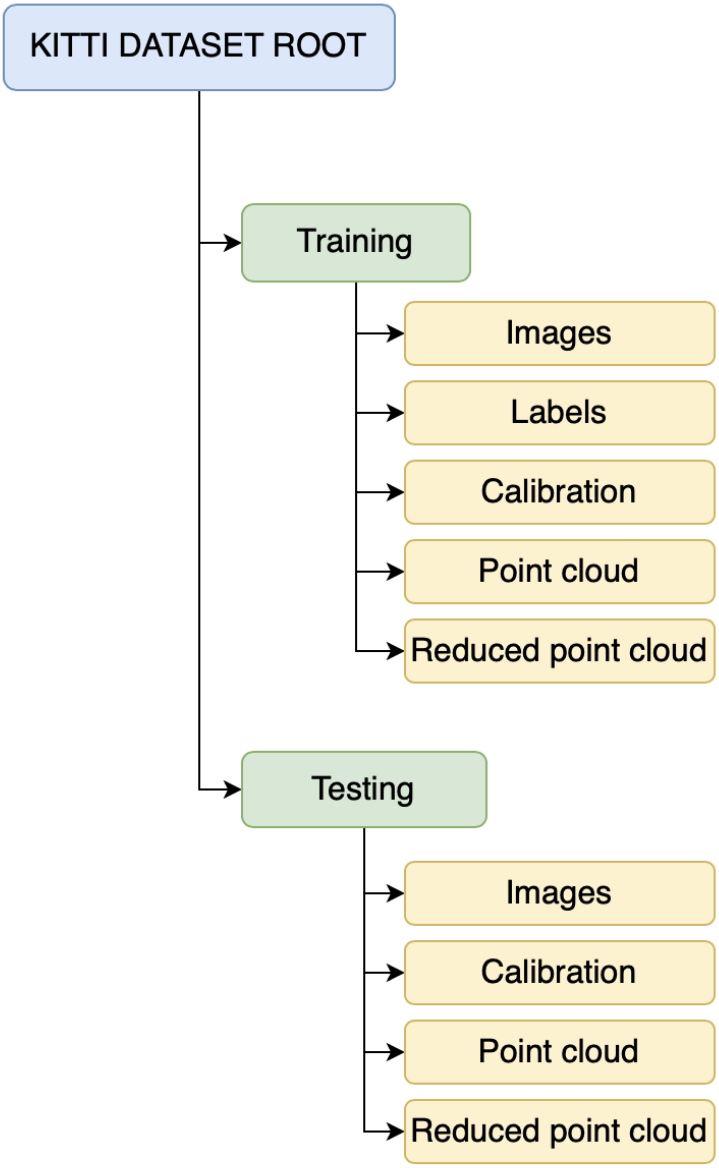


Figure 3.10: KITTI file structure for training and testing with images, point clouds, calibration and labels

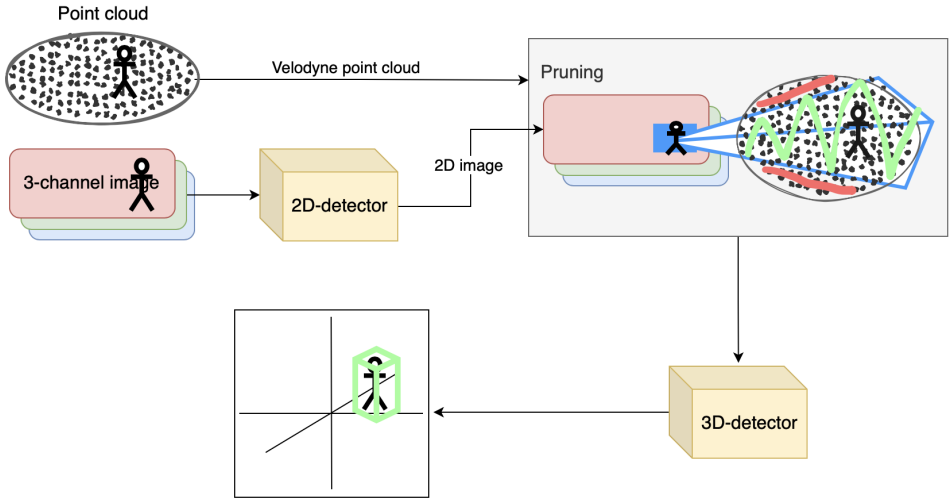


Figure 3.11: Frustum-PointPillars model inference pipeline

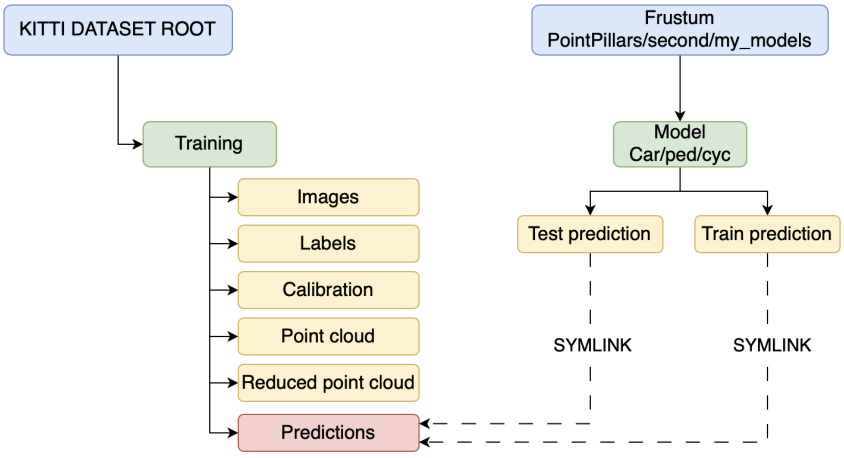


Figure 3.12: KITTI visualisation file structure with images, point clouds, calibration and labels with symbolic link to the predictions made by the model

Chapter 4

Experiments and Results

4.1 Frustum-PointPillar results

4.1.1 Precision and recall vs validation

In figure 4.1 we can see that the model achieves around 0.9, 0.75 and 0.7 on easy, moderate and hard respectively on the official KITTI validation dataset for cyclist mAP at IoU70. For pedestrians, it performs worse at 0.4, 0.35 and 0.3. We also see that precision at IoU70 is rapidly increasing while recall IoU70 is slowly increasing for both cyclist and pedestrian.

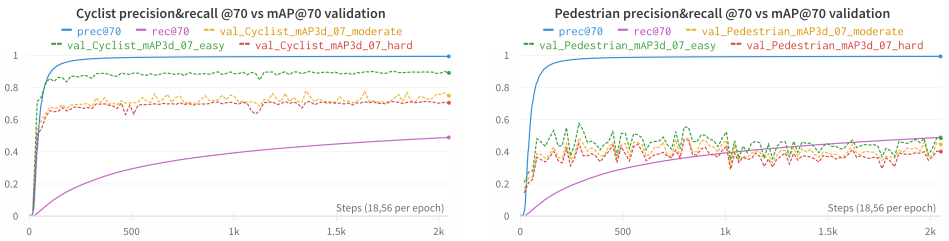


Figure 4.1: Precision and recall @ 70 vs validation mAP@70 easy, moderate and hard for pedestrian and cyclist

For the car object, the model achieves a higher validation mAP at IoU70 than both cyclist and pedestrian at 0.9, 0.8 and 0.8 for easy, moderate and hard, as seen in figure 4.2. In the figure, we can also see that precision is increasing faster for car than for cyclist and pedestrian. On the other hand, recall is climbing slower and stabilizes at below 0.6, while both pedestrian and cyclist recall at IoU70 stabilizes above 0.6.

In figure 4.3 we can see that the model achieves around 0.9, 0.8 and 0.8 on easy, moderate and hard respectively on the official KITTI validation dataset for cyclist mAP at IoU50. For pedestrians, it performs slightly worse at 0.85, 0.8 and 0.8 accordingly. Like the results at IoU70, we also see that precision at IoU50 is also increasing rapidly while recall IoU70 is increasing slower for both cyclist and

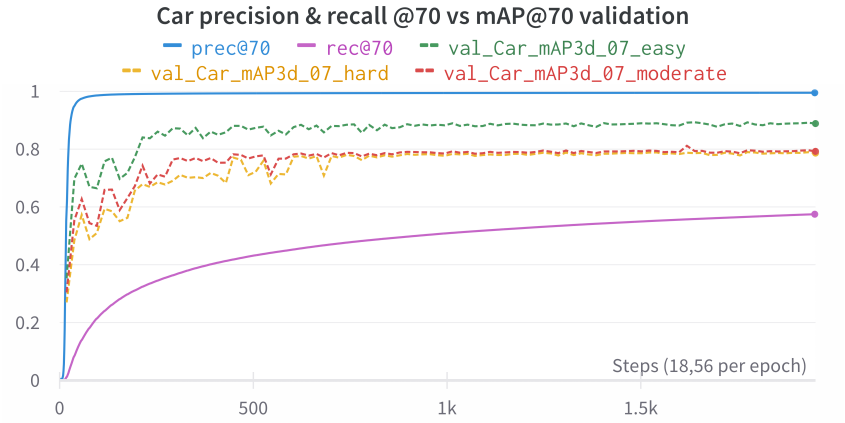


Figure 4.2: Precision and recall @ 70 vs validation mAP@70 easy, moderate and hard for car

pedestrian.

The results for cyclist did not improve for the easy validation dataset when decreasing IoU threshold from 50 to 70. However, for moderate and hard validation there was a slight improvement of around 0.05 and 0.1 in accuracy.

Decreasing IoU threshold from 70 to 50 significantly improved the validation results for pedestrians in all difficulties. Easy and moderate increased accuracy by 0.45, which is a very big change. For hard, the change in accuracy was 0.5.

These results indicate that lower IoU threshold is most beneficiary for pedestrian object detection, as it drastically improves the mAP.

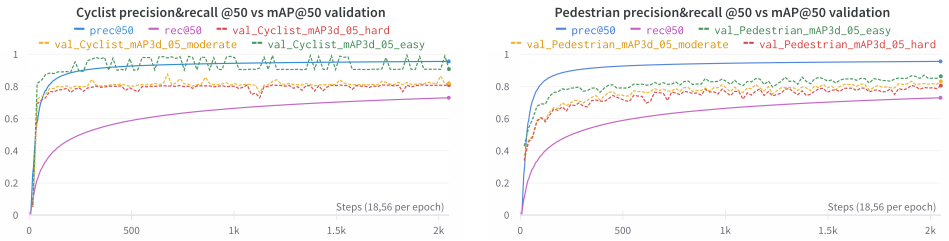


Figure 4.3: Precision and recall @ 50 vs validation mAP@50 easy, moderate and hard for pedestrian and cyclist

For car at IoU50, the model achieves a higher validation mAP than both cyclist and pedestrian at above 0.9 for all validation difficulties, as seen in figure 4.4. In contrast to recall at IoU70, recall at IoU50 is increasing faster and stabilizes above 0.7, which is higher than for both pedestrian and cyclist which stabilizes slightly below 0.7.

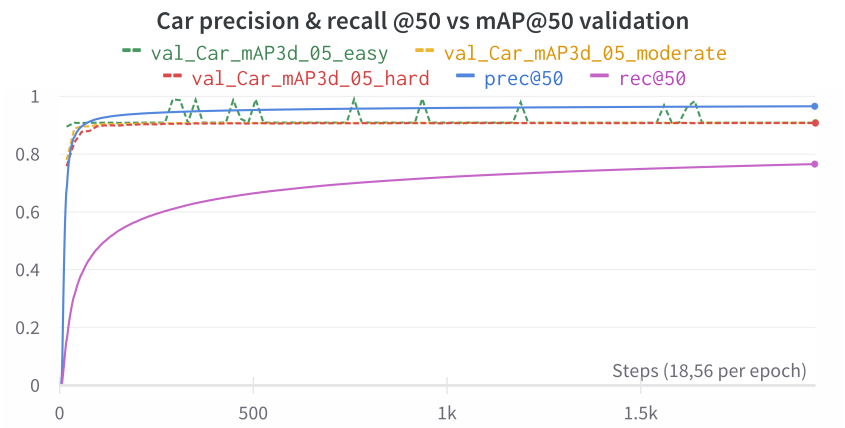


Figure 4.4: Precision and recall @ 50 vs validation mAP@50 easy, moderate and hard for car

4.1.2 Train loss vs validation accuracy

In figure 4.5 we can see that location loss for car, which measures error in distance of bounding box predictions is decreasing rapidly, then flattening after around 500 steps. Classification loss has the same trend, just stabilizing around 0.2 below location loss. Moderate validation at both IoU50 and IoU70 seems to stabilize around 500 steps as well.

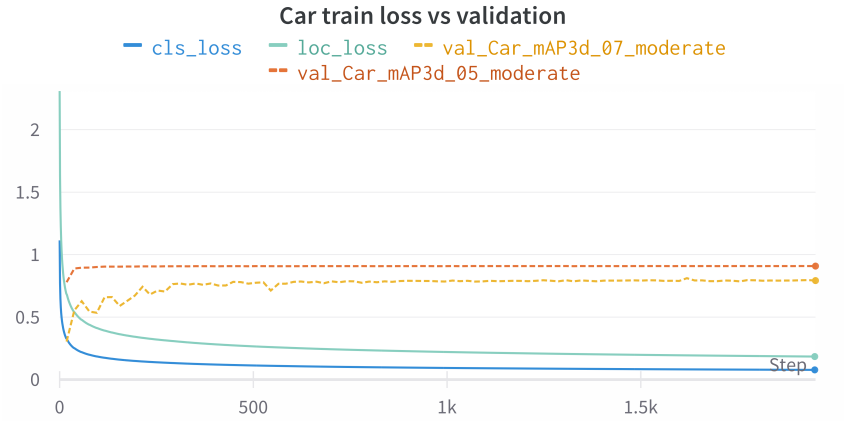


Figure 4.5: Train location and classification loss vs moderate 3D validation mAP@50 and mAP@70 for car

Figure 4.6 shows location and classification loss compared to moderate mAP validation at IoU50 and IoU70. For pedestrian, the location loss does not stabilize before 1500 to 2000 steps, while classification loss stabilizes around 500 steps. We can also see that validation mAP is slowly increasing even after 2000 steps for

IoU50, while IoU70 seems to have peaked around 1000 steps.

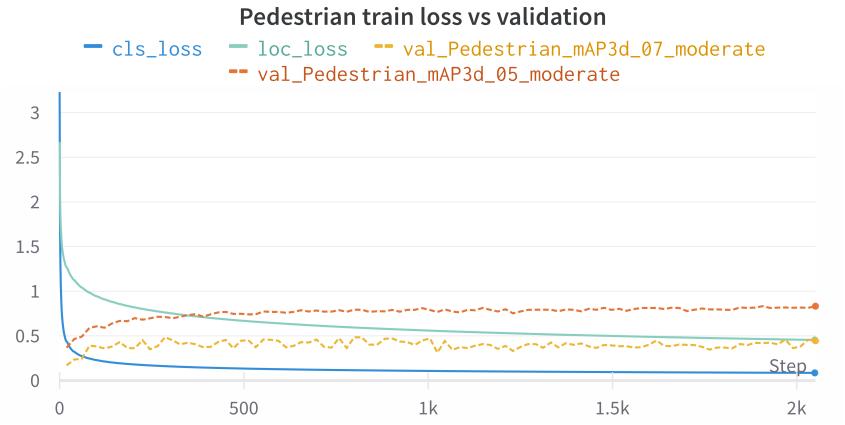


Figure 4.6: Train location and classification loss vs moderate 3D validation mAP@50 and mAP@70 for pedestrian

Figure 4.7 shows location and classification loss compared to moderate validation mAP at IoU50 and IoU70 for cyclist. Location and classification loss has the same trend as with pedestrian, where location loss is slowly decreasing even after 2000 steps while classification loss stabilizes around 500 steps. Both moderate mAP validation metrics are rapidly increasing the first few steps, then stabilizing before 500 steps.

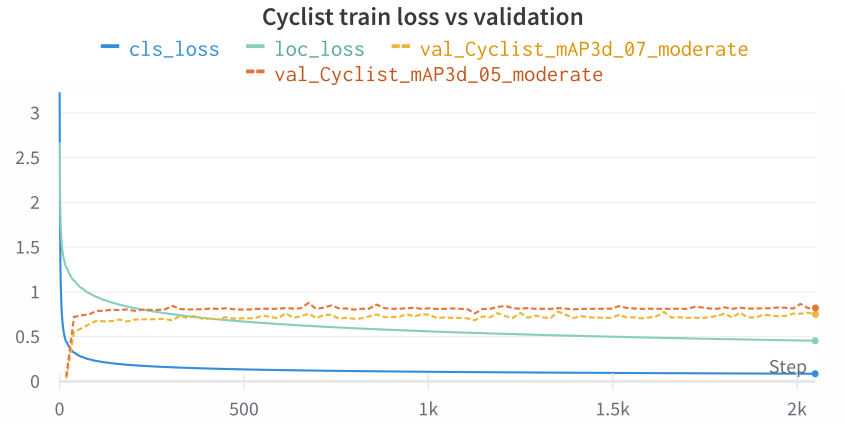


Figure 4.7: Train location and classification loss vs moderate 3D validation mAP@50 and mAP@70 for cyclist

4.1.3 3D, BEV and AOS moderate validation

Figure 4.9 and 4.8 shows BEV, 3D mAP and AOS at IoU70. AOS measures the error of the angle between ground truth and predictions for car, pedestrian and cyclist. For car and cyclist, all three metrics stabilize around 0.7 to 0.9 before 500 steps, while still slowly increasing for pedestrian after 2000 steps or 100 epochs.

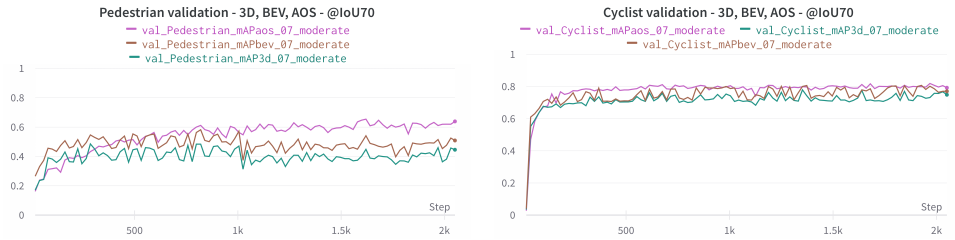


Figure 4.8: mAP@70 3D, BEV and AOS validation at moderate difficulty for pedestrian and cyclist

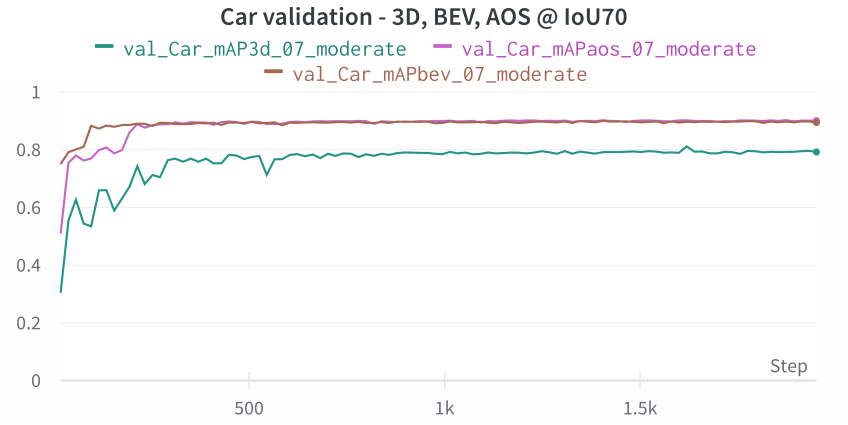


Figure 4.9: mAP@70 3D, BEV and AOS validation at moderate difficulty for car

Figure 4.11 and 4.10 shows BEV, 3D mAP and AOS at IoU50. The trend seems to be similar to IoU70, however car stabilizes at 0.9 and cyclist at 0.8 for all metrics. For pedestrian we see that the metrics are still in an increasing trend after 2000 steps or 100 epochs, indicating that further training could improve results.

4.1.4 Precision and recall @ 10, 50, 95

Precision at IoU 10, 50 and 95 is shown for car in figure 4.12 and pedestrian/cyclist in figure 4.13. We can see that IoU10 has lower precision while IoU95 has the highest precision for all objects. We can also see that IoU95 takes more steps to start increasing as the curve is horizontal for around 150 steps while it increases significantly in the first couple of steps for IoU 10 and 50.

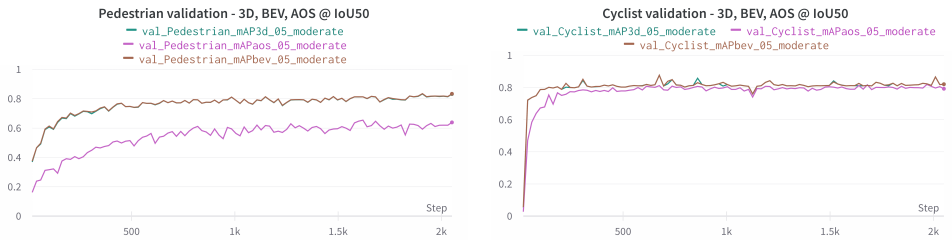


Figure 4.10: mAP@50 3D, BEV and AOS validation at moderate difficulty for pedestrian and cyclist

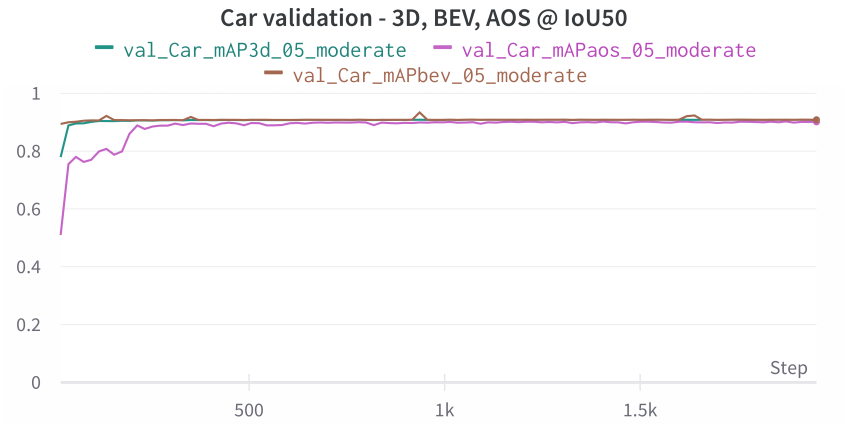


Figure 4.11: mAP@50 3D, BEV and AOS validation at moderate difficulty for car

Figure 4.14 shows recall at IoU 10, 50 and 95 for car, while figure 4.15 shows recall curves for cyclist and pedestrian. The IoU thresholds achieve results in opposite order as with precision, as IoU 10 achieves highest recall, 50 second best and 95 lowest. IoU10 seems to have stabilized somewhat, while 50 and 95 are still in an increasing trend.

4.2 Author results vs my results

In table 4.1 we can see the results of the author of the Frustum-PointPillars model compared to my results with some small modifications. For car, the original (author) model performs slightly better for easy and moderate difficulty. However for hard, my model scores 55 basis points better. Overall between the 3 difficulties for car, my model achieves on average 43 basis points better mAP score. For pedestrian, my model performs significantly worse on all three difficulties with around 10-15% worse mAP score. My model achieves better results for cyclist on all difficulties, beating the original model by around 2-4%.

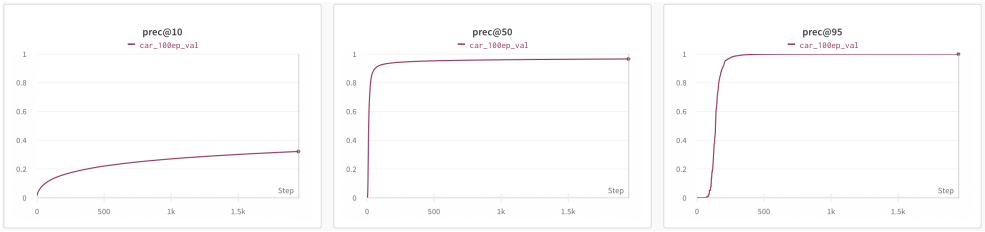


Figure 4.12: Precision at IoU 10, 50, 95 for car

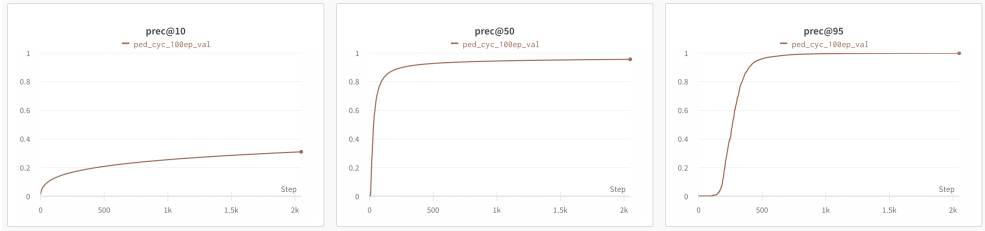


Figure 4.13: Precision at IoU 10, 50, 95 for pedestrian and cyclist

4.3 Visualisation of predictions

In this section the visual results of car-, pedestrian- and cyclist predictions will be presented

4.3.1 Car visualisation

Figure 4.16 shows the 2D image of a scene with cars where the cars are labeled with green bounding boxes as ground truth. The equivalent scene is shown in 3D-space as a point cloud in figure 4.17 where the green bounding boxes are the ground truths and the red bounding boxes represent the model predictions. We can see that the predictions are pretty good in this example as the red bounding boxes are mostly overlapping with the green bounding boxes.

Figure 4.18 shows the same point cloud scene from BEV. The red lines represent the predicted directional angle for the cars. We can see that the predicted angles of the cars match the direction shown in figure 4.16 for all cars. It is worth noting that the bounding box prediction with the highest error is the one for the car that is truncated in the FoV of the RGB image, where the predicted bounding

Table 4.1: Comparison between author results and my own results on car, cyclist and pedestrian for easy, moderate and hard validation

	Car			Pedestrian			Cyclist		
Difficulty	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Author model	**88.90**	**79.28**	78.07	**66.11**	**61.89**	**56.91**	87.54	72.78	66.07
My model	88.84	79.22	**78.62**	48.53	44.57	40.16	**89.09**	**74.90**	**70.52**

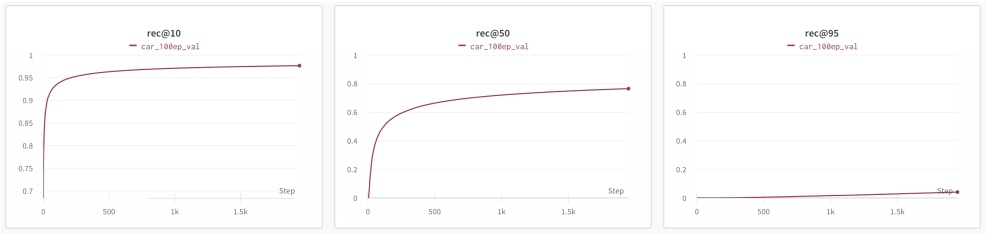


Figure 4.14: Recall at IoU 10, 50, 95 for car

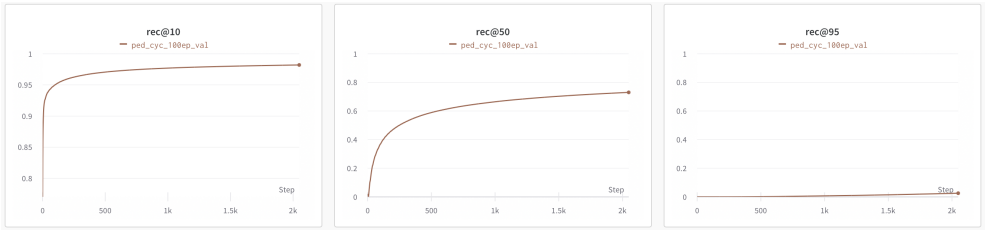


Figure 4.15: Recall at IoU 10, 50, 95 for pedestrian and cyclist

box stretches further than the ground truth bounding box.

Figure 4.16: RGB-visualization of the image to be predicted where green boxes indicate ground truth

4.3.2 Pedestrian and cyclist visualisation

Figure 4.19 shows the 2D image of a scene with pedestrians and a cyclist where the pedestrians are labeled with blue bounding boxes and the cyclists with yellow. The equivalent scene is shown in 3D-space as a point cloud in figure 4.20 and 4.21 where the blue and yellow bounding boxes are the ground truths and the red bounding boxes represent the model predictions. We can see that the predictions are decent in this example, but the bounding box predictions are not as good as with the car predictions in the previous example. We can also see that one of the pedestrians were not detected at all in this example.

Figure 4.22 and 4.23 shows the same point cloud scene from BEV. The red lines represent the predicted directional angle for the pedestrians and cyclists. We

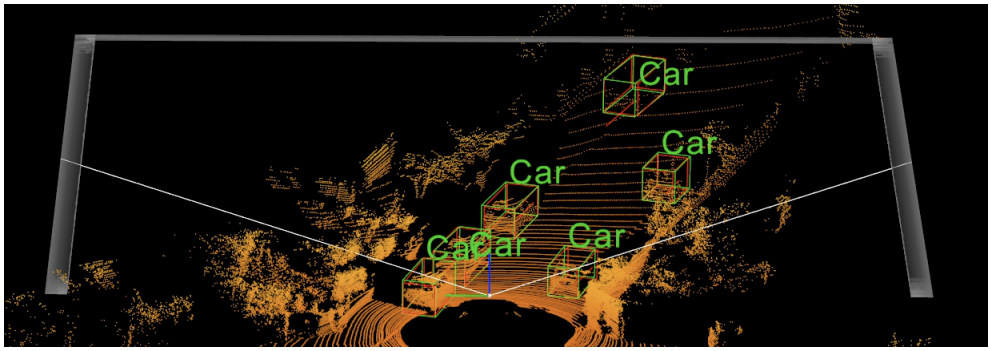


Figure 4.17: Velodyne-visualization of point cloud with green boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

can see that the predicted angles of the pedestrians mostly match the direction shown in figure 4.19, however there are some that are pointing in completely wrong direction or a direction that is not properly aligned with the suspects. For cyclist, the bounding box angle is pretty accurate, and the bounding box size is matching the ground truth well.

4.3.3 Prediction on Ouster data

Scene 1

In the first scene in figure 4.25 the predicted bounding boxes lay on a line on the y-axis. Only one of the predictions is close to a ground truth, which is better illustrated in figure 4.26. None of the predicted bounding boxes have overlapping IoU. Some of the predictions with low confidence were removed to make illustration clearer.

Scene 2

Figure 4.27 and 4.28 shows predictions and ground truths for scene 2, where we can see that the prediction closest to the middle actually has a slight overlap with the ground truth, however the IoU is small.

Scene 3

In scene 3, which is shown in figure 4.29 and 4.30, we can see that the prediction matches orientation, but is too small and too far along the y-axis. Also, some of the predictions with low confidence were removed to make illustration clearer

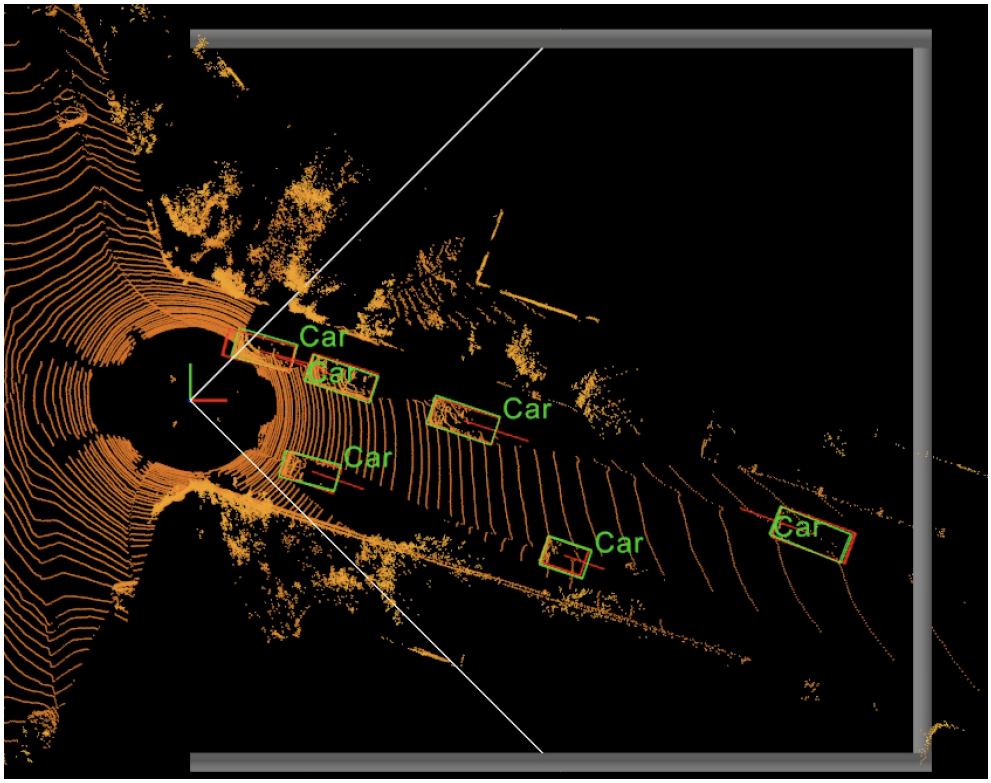


Figure 4.18: BEV-visualization of point cloud with green boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

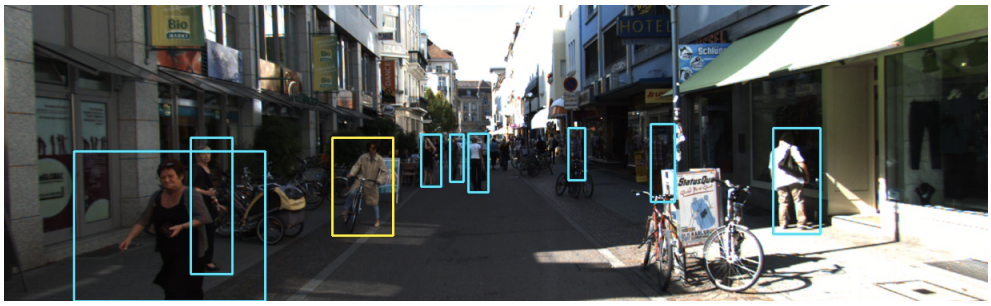


Figure 4.19: RGB-visualization of the image to be predicted where blue boxes indicate ground truth for pedestrian and yellow boxes for cyclists

Figure 4.20: Far velodyne-visualization of point cloud with blue and yellow boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

Figure 4.21: Close velodyne-visualization of point cloud with blue and yellow boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

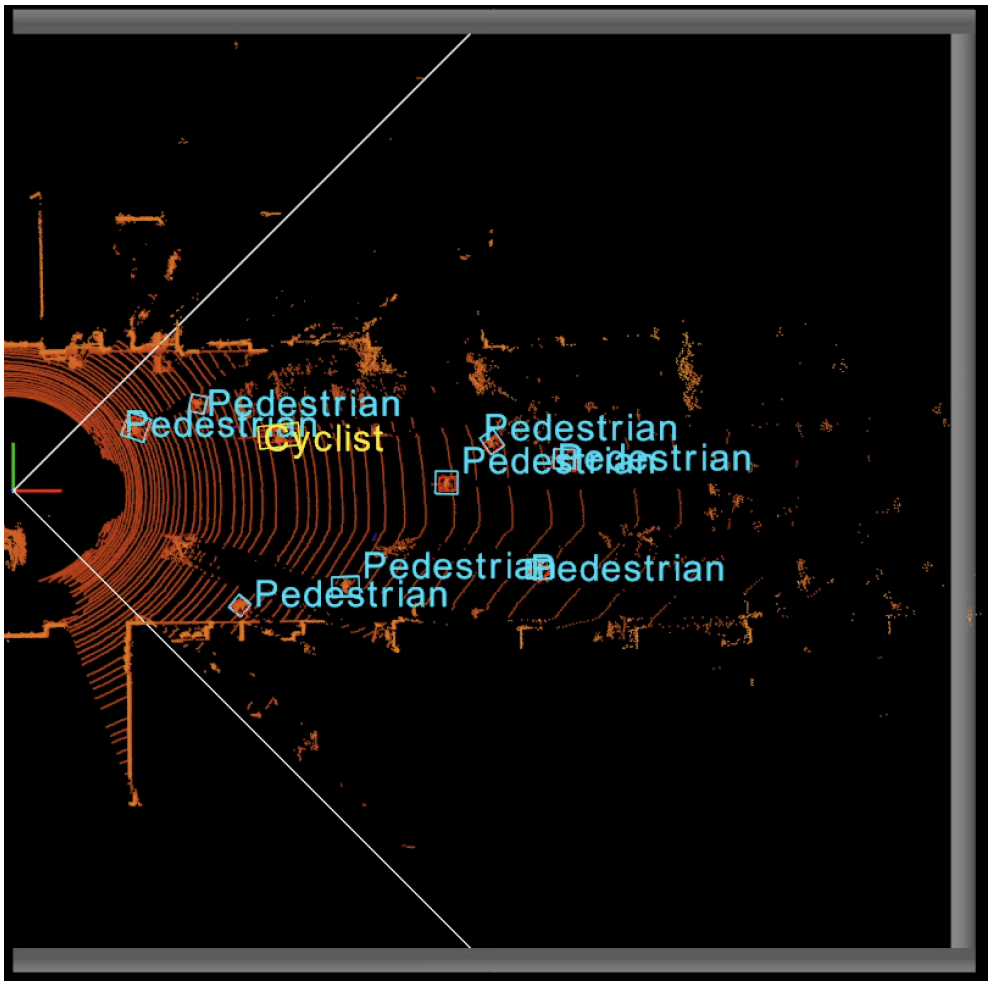


Figure 4.22: BEV-visualization of point cloud with blue and yellow boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

Figure 4.23: BEV-visualization of point cloud with blue and yellow boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

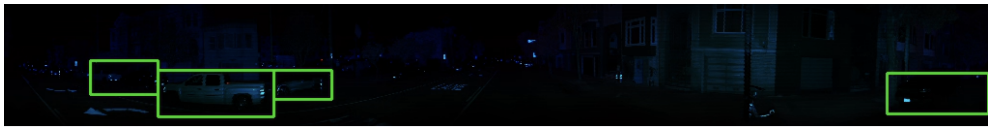


Figure 4.24: SCENE 1: 3 channel LiDAR-image consisting of Near-infrared, signal and reflection with green ground truth boxes for cars

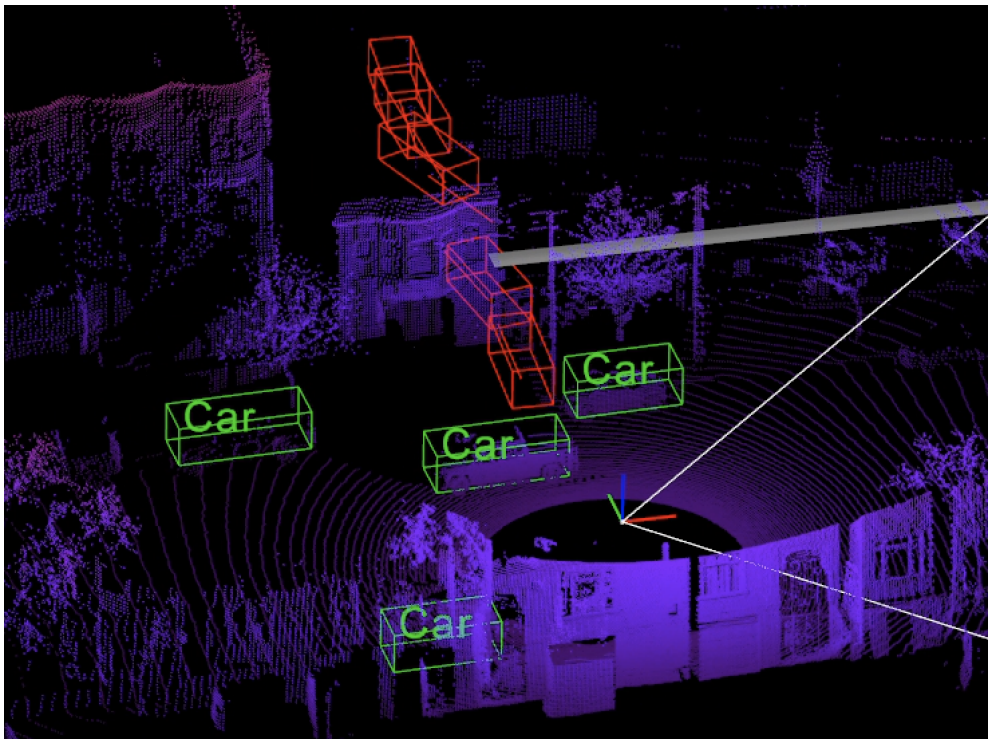


Figure 4.25: SCENE 1: Velodyne-visualization of point cloud with green boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

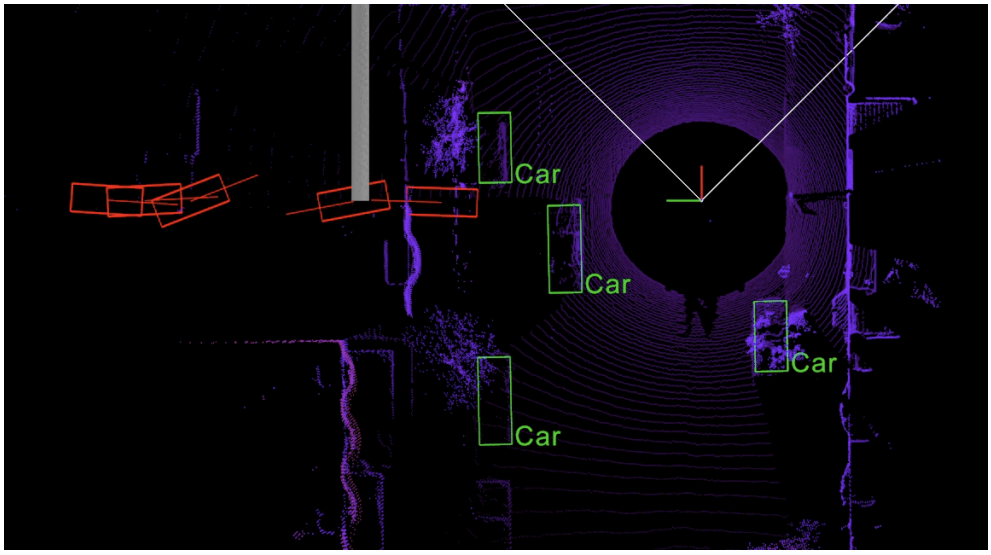


Figure 4.26: SCENE 1: BEV-visualization of point cloud with green boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

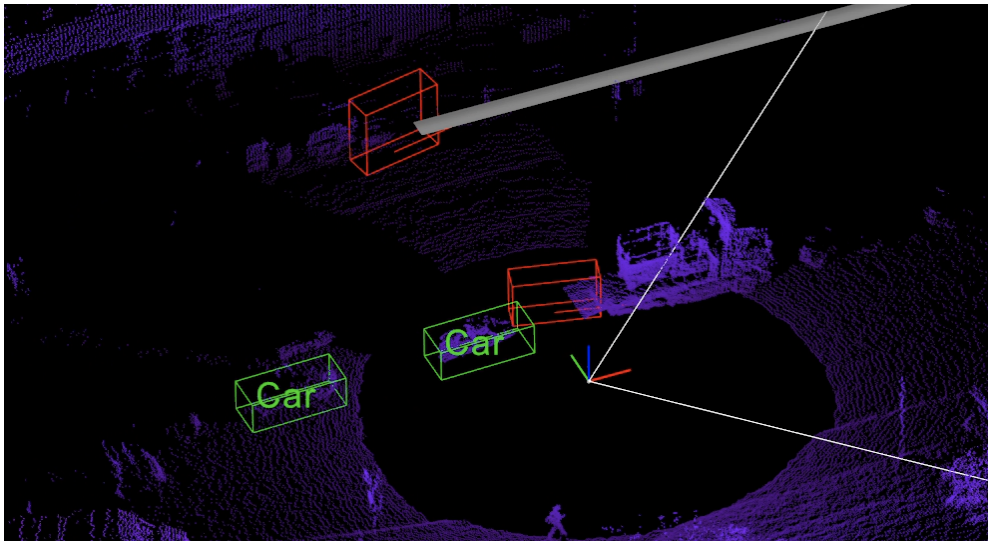


Figure 4.27: SCENE 2: BEV-visualization of point cloud with green boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

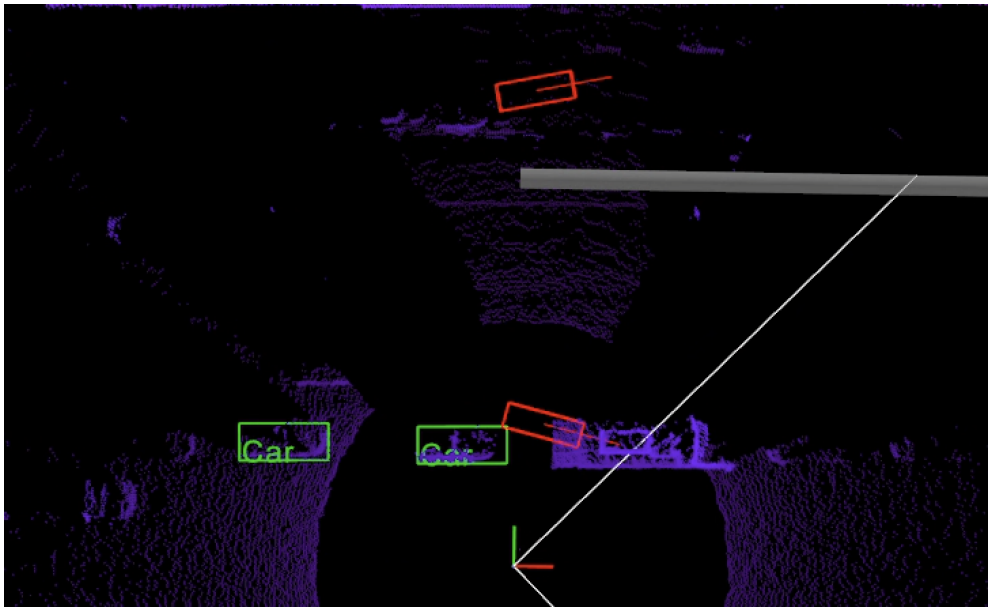


Figure 4.28: SCENE 2: BEV-visualization of point cloud with green boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

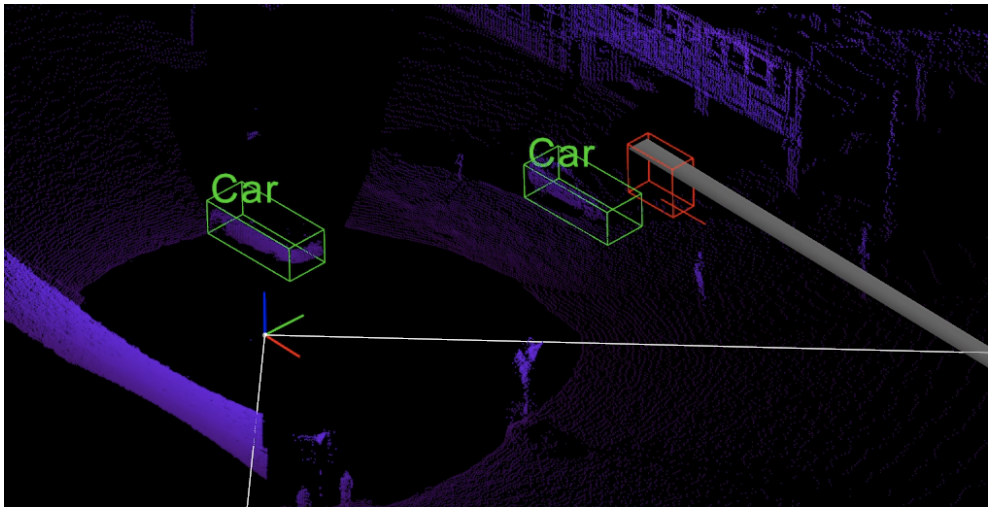


Figure 4.29: SCENE 3: BEV-visualization of point cloud with green boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

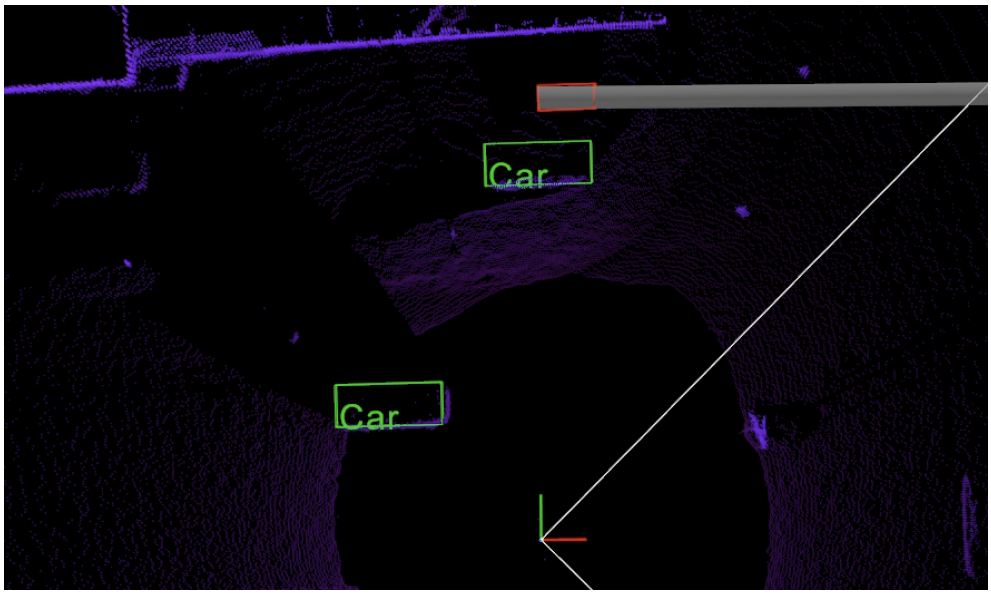


Figure 4.30: SCENE 3: BEV-visualization of point cloud with green boxes as ground truth and red boxes as the model predictions. Red lines indicate orientation

Chapter 5

Discussion

5.1 Model performance

This section discusses the performance of the Frustum-PointPillars model on KITTI data, where two main parts are examined; Metric results and visual inspection.

5.1.1 Metric performance on KITTI data

My results on the KITTI data were in line with the authors results in the paper listed in the GitHub repository[1] as mentioned in section 4.2. Overall, my model performs slightly better for the objects car and cyclist, while significantly underperforming for pedestrian. The authors model achieves a higher score for 5/9 difficulty and object metrics.

It is interesting that my model scores higher for both car and cyclist even though the amount of training epochs used to train the model was reduced to 37.5% less than the author model. Because most of the hyperparameters were unchanged, one could assume that it is a possibility that the author model was slightly overfitted for these objects. One theory as to why their model overfitted for car and cyclist could be that there is a huge skew towards car objects in the KITTI dataset as mentioned in section 3.2.1. As bicycles share some of the characteristics of car, there is a higher possibility that the same learned features can be applied for both objects. Since there is significantly less pedestrians in the dataset and the similarity to car for the object is smaller, more epochs could be needed to get the model to learn appropriate representations in feature space. Furthermore, by looking at figure 4.6 and 4.8 we can see that the validation score for pedestrian has not stabilized yet and location/classification loss is still declining, which indicates that more training epochs probably could further improve the results for pedestrian and thus match or exceed the author results.

In figure 4.3 and 4.1 we can see that the pedestrian validation mAP increased significantly when decreasing IoU from 70 to 50. We know that the pedestrian object on average is pretty small from figure 3.3, where the average width and

[1] `https://github.com/anshulpaigwar/Frustum-Pointpillars`

length is around 0.5 meters and height at around 1.8 meters. This shape property makes it hard for the velodyne sensor to be accurate as the 360 scan has low angular resolution, meaning that the depth measurements are not very precise at centimeter resolution perpendicular to capturing angle at range. Thus, by reducing the IoU requirement significantly increases the models ability to correctly detect pedestrians.

In figure 4.2 and 4.4 we can see that decreasing IoU from 70 to 50 boosts recall and validation **??** for car slightly, while for pedestrian and cyclist the improvements are significant as seen in figure 4.1 4.3. Again, part of the reason why decreasing IoU requirement from 70 to 50 works better for smaller objects like pedestrians and cyclists is because of the angular resolution of the LiDAR sensor. This indicates that in order to rely on point cloud data in autonomous settings, higher resolution LiDAR sensors are preferred to make sure smaller objects are identified correctly with high confidence.

For the precision and recall curves at the different confidence values 10, 50 and 95 in figure 4.12 and 4.14, we can see that the precision increases with higher confidence levels, while recall decreases with higher confidence levels. This is to be expected as the model is more precise at higher confidence levels, while creating lots of lower quality guesses at lower confidence. Finding the correct confidence level depends on the application context. In our case, if the model were to be deployed in a real autonomous setting, we would rather like the model to be overly cautious to prevent accidents, and thus setting a lower confidence level would make the model recall more of the surrounding objects. On the other side, we do not want the model to be too cautious either, because then the car would simply brake unnecessarily all the time, also referred to as ghost braking. The model is not actually deployed in this project, thus this problem of confidence level tuning was not investigated further and remains as a future work task.

5.1.2 Visual results

The model predictions on the KITTI data are best for the car object when performing a visual inspection. Figure 4.17 and 4.21 shows some of the model predictions, and we can see that the bounding boxes for pedestrian and cyclist have less IoU overlap than for car. This is not the case for all the predictions, but after inspection of more than 100 examples it seems to be the trend. This is expected after looking at the precision, recall and validation mAP in the previous section. One could assume that part of the reason why cyclists and pedestrians are harder to predict good bounding boxes for, is because there are more degrees of freedom. Cars are mostly static shapes where the wheels are the only part that can rotate in a different direction to the chassis. For humans and cyclists there are arms, legs and heads that have large range of motion in different directions, making it difficult to encapsulate the bounding box edges and the orientation of the object. The other explanation and perhaps the larger reason why cars perform so much better is because of the large skew in number of objects as mentioned in section

5.1.1. One interesting detail in figure 4.18, which shows the BEV view of the car scene, is that the car that had the largest IoU miss is the one car furthest to the left, where part of the car is cut of by the RGB image FoV. We can see that the model misses on truncation, making the car prediction bounding box extend further than the ground truth. This is an indication that 2D frustum masking might have a flaw, as some parts of the objects get cut out of the frustum if the object elongates out of frame. This flaw is mostly occurring in the KITTI dataset because the RGB image does not stretch a full 360 degrees like the LiDAR sensor does. This is an advantage with the Ouster sensor, which will be discussed later.

5.2 Thesis research questions

This section discusses the main findings related to the research questions of this thesis.

5.2.1 Research question 0: Reproducibility

Setup and equipment

The authors used Ubuntu version 18.04 with Python version 3.6, PyTorch version 1.4 and unspecified CUDA version. The documentation of working versions is very inadequate and led to lots of time used for debugging and version controlling. As the visualisation tool in the repository did not work, KITTI-VIS was downloaded and used to visualize point clouds with objects. This visualisation tool was also poorly documented and contained some bugs, however it worked at last. Better documentation and testing would make the reproducibility significantly easier even though it was manageable to make it work nonetheless.

Results and visualisation

After finally getting the model to train properly, Weights and Biases was used to track the training parameters of the model to ensure that everything was running smooth. By implementing Weights and Biases, live tracking of model training runs was possible in a structured way. After successfully training a model, the KITTI-VIS tool was crucial for visualizing the point cloud with the predictions and the corresponding ground truth labels.

Conclusion

As discussed in section 5.1.1, my results are matching or exceeding the results stated by the authors of the Frustum-PointPillars model, and thus the reproducibility of the results are approved. The model is also correctly predicting the bounding boxes for multiple object classes as seen in the visual inspection performed in section 5.1.2 utilizing the "KITTI-VIS" tool. Overall, even though the repository contained some bugs and needed implementation of tools like WandB and

KITTI-VIS to inspect the results, the reproducibility is acceptable if one navigates through bugs and has a compatible setup.

5.2.2 Research question 1: Performance on Ouster data

Modification

The modification of the pipeline is running without any errors and produces output in the correct format. The modification allows the model to do predictions on Ouster data with 360 degree view of the surrounding scene. The advantage of using this Ouster sensor data is that the entire scene is mapped with one to one correspondence between range signals and other signals, meaning that a pre-processing step is removed from the pipeline, probably allowing for faster runtime. The pre-processing time savings is unfortunately neglected by the increased number of points to be predicted, as a 360 degree FoV contains more objects and thus more data to compute. The end result is computation of a larger scene with slightly slower inference time. If the scene were to be adjusted to the same FoV as the KITTI data, the pre-processing time would decrease and thus speed up the overall pipeline.

Performance

The model does correctly predict the height (z-axis) of the objects in the Ouster example, as seen in figure 4.25. It misses on x, y and orientation of the objects, which is shown in figure 4.26. The model has not been trained on Ouster sensor data, which probably is the reason why it performs subpar in this example. The confidence of the predictions are 15% and below, indicating that the model is uncertain of the structure in the different objects point clusters. Another factor as to why the model has low confidence might be because the reflection intensity of the points in the point clouds are not matching properly to the KITTI point clouds. This is because different sensors have different settings and hardware, resulting in variance in reflectance between point clouds of the same scene. Also, different angular resolution could be a factor as the clustering of points will be different.

5.2.3 Conclusion

The modified pipeline for predicting Ouster sensor data is working, and the model does correctly predict the height of the objects in the scene. However, due to difference in sensor hardware, the model needs fine-tuning on annotated Ouster data in order to correctly predict x, y and orientation of the objects in the scene

5.2.4 Research question 2: Feasibility of multi stage model

Real-time compute

The pipeline was described in detail in section 3.3.2. Because the pipeline takes both 2D and 3D input, it creates a lot of potential processing bottlenecks. The authors of the Frustum-PointPillars model claimed a total of 14.6Hz or 70ms in computation time per example. Those 70ms consists of around 10ms pre-processing, 18ms masking with frustums, 12.5ms forward pass and 29ms post-processing time. My model does not achieve quite the same run-time performance as the authors claim, with around 100ms or 10Hz total compute time. It can be many reasons as to why my model runs a little bit slower, including different CPU, GPU, package versions, OS version etc. 100ms is still considered real-time, and seems even faster taking into account that human reaction time to visual input is around 250ms. It is also worth noting that if such a model were to be deployed, further optimizations for compute time could be implemented. For example by performing pre-processing and masking on GPU instead of CPU, the run-time could be further improved. Also, increasing compute power by upgrading the components or even creating a dedicated setup would drastically improve the pipeline efficiency.

Part of a larger architecture

Even though the results are decent and the computational time is sufficient for real-time deployment, it would require a lot of work to deploy such a model into a car to make it safe enough for autonomous driving. Many of the largest automakers in the world, including Volkswagen Group, Tesla, BYD, Nissan and BMW, are working on such solution as the total addressable market is enormous. The fact that none of these car manufacturers have made a working solution that is ready for real world deployment without supervision indicates that the task of achieving autonomous driving is rather large. Object detection is just a small part of a larger architecture that is needed to sense, interpret and act in a real world environment with human agents.

Conclusion

Assuming that the supportive components mentioned in the previous section are working, a multi-stage frustum based network has the potential of being feasible for autonomous driving in the future. Frustum PointPillars offers the crucial real-time object detection capability that autonomous technology relies upon, and by further refurbishing the deployment pipeline as discussed, such a system can exceed human ability when it comes to driving.

5.3 Sources of error

In this section, sources of error related to the model, the pipeline and other categories will be discussed, and suggestions for improvement will be mentioned.

5.3.1 Model and pipeline errors

In section 5.1.1 we discussed that pedestrian results did not match the authors pedestrian results, and would probably improve by increasing the number of epochs during training. To combat this issue, learning rate could possibly be increased while doing the same number of epochs to reduce training time and maybe increase model performance. However, it is not given that increasing learning rate will give better results, as larger gradients can often make the training process unstable and produce errors that are hard to predict.

One underlying issue is that the model does not predict all object classes at the same time, meaning that separate models need to be trained for each group of object classes, which obviously is not computationally efficient for both training, storage and inference. This is because when multiple models are trained, it takes more compute given that the neural net architecture remains the same, and duplicate neural net architectures are stored, taking up more storage space. This also effects pipeline decisions, as the models need to concatenate the predictions for each scene. The model performs poorly when trained for predicting more than two objects at the same time, more specifically, when the object classes are dissimilar in size distributions. For example, when the model is trained for cars and pedestrians, it performs worse than when trained for pedestrians and cyclists. The point at issue is the way the model does anchoring to reduce the bounding box search space. The anchoring size range for cars is larger than the anchoring size range for pedestrians, meaning that the total anchoring range when combining the two objects are increased. This may make the model struggle to find good bounding box predictions early on in the training phase, which makes the process take longer time. It may also decrease the total performance on each object as a result. If the anchoring ranges were made dynamically based on the size of the object to be predicted, this problem might be solved or at least reduced.

5.3.2 Sensor errors

One of the main drawbacks of the KITTI dataset is that the FoV of the velodyne LiDAR sensor and the camera sensor does not overlap in the majority of the captured scenes. This leads to problems like truncation and unseen objects based on frustum masking. By implementing the Ouster LiDAR sensor into the pipeline, a 360 degree common FoV is available, and thus the entire scene around the car is captured and predicted. The problem is that for this project, a lack of annotated Ouster data causes the model to do bad predictions. If there was more time and higher quality annotation tools available, this task would be more reasonable to complete for a project of this size.

Another inherent flaw with the sensors is the discretization of continuous real-world data, which leads to simplifications and reduction in information captured. For the camera sensor, the resolution is below 1 megapixel. This causes objects at distance to be significantly noisy with low pixel resolution. By feeding the model higher resolution images from a higher resolution camera, object detection at farther ranges would be possible. For the LiDAR sensor, the resolution is mainly held back by the angle between each LiDAR beam, which significantly degrades the granularity at range. By using a LiDAR sensor which has a smaller angle between capture beams, the angular resolution would improve and perhaps better performance could be achieved. It is also worth discussing that improving resolution of sensors does have its drawbacks also. Higher resolution means more data and thus more computation needed to extract relevant information. This is a balancing problem where one needs to find the sufficient resolution for the given task to optimize for compute efficiency and data quality.

5.3.3 Other errors

The sources of error mentioned in the subsections above talks about known problems. In such a technical project with a large code-base it is possible to have undetected bugs which cause poorer performance without explicitly deteriorating the results. Especially with the Ouster data, there can be bugs that cause the model to do poor prediction. This is one of the problems with neural networks, as the debugging process can be laborious because of the black box properties of AI.

Another possible weakness of the project is the usage of road data captured from the road and not from different perspectives. By not including data from multiple perspectives, some bias towards the view of the car can be introduced in the model. For example using data captured from security cameras would create a higher diversity of angles and object rotations. Another possibility to combat uniform data would be to create augmentations of the dataset by stretching, flipping, cropping and copy-pasting objects and scenes. Simulations could also improve upon this.

As the dataset was created externally, we do not have the capacity to validate the labeling process and inspect all ground truth bounding boxes. There is a possibility that there are errors in the ground truths or small variance in labeling technique. For example, some of the people labeling the data are not labeling objects at certain distances, while others do. Some bounding boxes might be larger, and others might be smaller. This causes a element of uncertainty which is hard to account for with limited time resources.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis research has been conducted on object detection in 3D space using 3-channel images and LiDAR data. There were 3 main goals of the work; reproduce a state of the art 3D detection model, make it run in real-time and adapt the model for 3-channel LiDAR images using Ouster LiDAR sensor data. A literature review was conducted early in the project in order to understand the current state of the art. The decision of selecting a model was tough as there are hundreds of different models with their own advantages and disadvantages, but in the end Frustum-PointPillars was chosen because of the combination of good performance, fast run-time and dual sensor input option. The model was modified to fit a 360 degree Ouster LiDAR scanner which captures one to one image pixel coordinates between the 3-channel LiDAR images and LiDAR point cloud images.

The model was trained using the KITTI dataset for 3D object detection with a RTX 8000 GPU. With the aim of validating the reproducibility and performance of the model and adapting it for Ouster data, different tools were adjusted and used. Weights and Biases was used to visualize the benchmarks of the training sessions real-time. The Ouster Python SDK was used to extract and pre-process the Ouster point clouds and the corresponding signal, reflectance and near-infrared data. Then the KITTI-VIS tool was adapted to show the point clouds with the corresponding model predictions for both KITTI data and Ouster data. No free tools for labeling 2D and 3D data was found, so the Ouster testing data was manually annotated through editing text file metadata and visualizing in KITTI-VIS.

Overall, the results in this thesis matched the results stated by the authors of the Frustum-PointPillars model. My model performed better on car and cyclist, but worse on pedestrians. For the Ouster data, the results were poor due to lack of annotated data for fine-tuning. The learning outcome of the thesis is invaluable, greatly enhancing knowledge in conducting state of the art research, 3D object detection, LiDAR data processing and a deeper understanding of the state of autonomous vehicles.

6.2 Future work

As mentioned in chapter 3, there is a significant skew in amount of objects in the dataset. To make the model perform more equally on all objects, a possible solution would be to increase the amount of pedestrian and cyclist objects in the dataset. However, since it is hard to capture and annotate 3D data at large scale, the feasibility of this proposal is low. Not to mention that when capturing road data while driving, it is reasonable to assume that most of the objects encountered will be cars. Creating such a large dataset could improve the model performance, but the feasibility is not great.

The model does not perform well on Ouster data as a result of no available labeled training data of this sort. In order to make the model perform better on the Ouster data, a dataset needs to be annotated and the training pipeline needs to be adjusted for the sensor calibration from the Ouster LiDAR sensor. This dataset should contain 3-channel LiDAR images with channels consisting of signal, reflectivity and near-infrared, combined with a corresponding LiDAR point cloud.

The current implementation of the model does not support prediction of more than 2 object classes at the same time, like discussed in section 5.3. This is because of the inherent difference in size of the objects. By creating a dynamic way of anchoring and specifying object size, more object classes could be predicted with the same model and thus reduce compute.

The confidence level of the model was set to 0.8 after some visual inspection, however this confidence threshold should be further analyzed in order to find an optimal threshold for an autonomous setting. A more optimal confidence threshold could be found by analyzing precision and recall curves at a continuous confidence spectrum, and consequently improve the results in light of autonomous driving in a real world setting.

Bibliography

[1] J. R. Treat, N. S. Tumbas, S. T. McDonald, D. Shinar, R. D. Hume, R. E. Mayer, R. L. Stansifer and N. J. Castellan, 'Tri-level study of the causes of traffic accidents: Final report. Executive summary.,' English, Indiana University, Bloomington, Institute for Research in Public Safety, Technical Report, May 1979, Accepted: 2010-02-10T14:14:22Z. [Online]. Available: http://deepblue.lib.umich.edu/handle/2027.42/64993 (visited on 16/01/2023).

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, *Attention Is All You Need*, arXiv:1706.03762 [cs], Dec. 2017. DOI: 10.48550/arXiv.1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762 (visited on 02/12/2022).

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, arXiv:2010.11929 [cs], Jun. 2021. [Online]. Available: http://arxiv.org/abs/2010.11929 (visited on 30/01/2023).

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, *Language Models are Few-Shot Learners*, arXiv:2005.14165 [cs], Jul. 2020. DOI: 10.48550/arXiv.2005.14165. [Online]. Available: http://arxiv.org/abs/2005.14165 (visited on 03/12/2022).

[5] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever and W. Zaremba, *Evalu-*

ating Large Language Models Trained on Code, arXiv:2107.03374 [cs], Jul. 2021. DOI: 10.48550/arXiv.2107.03374. [Online]. Available: http://arxiv.org/abs/2107.03374 (visited on 03/12/2022).

[6] D. Fernandes, A. Silva, R. Nevoa, C. Simoes, D. Gonzalez, M. Guevara, P. Novais, J. L. Monteiro and P. Melo-Pinto, 'Point-cloud based 3D object detection and classification methods for self-driving applications: A survey and taxonomy,' eng, Apr. 2021, Accepted: 2022-05-30T11:29:06Z Publisher: Elsevier, ISSN: 1566-2535. DOI: 10.1016/j.inffus.2020.11.002. [Online]. Available: http://repositorium.sdum.uminho.pt/ (visited on 24/01/2023).

[7] A. Mousavian, D. Anguelov, J. Flynn and J. Kosecka, *3D Bounding Box Estimation Using Deep Learning and Geometry*, arXiv:1612.00496 [cs], Apr. 2017. DOI: 10.48550/arXiv.1612.00496. [Online]. Available: http://arxiv.org/abs/1612.00496 (visited on 23/01/2023).

[8] B. Li, W. Ouyang, L. Sheng, X. Zeng and X. Wang, *GS3D: An Efficient 3D Object Detection Framework for Autonomous Driving*, arXiv:1903.10955 [cs], Mar. 2019. DOI: 10.48550/arXiv.1903.10955. [Online]. Available: http://arxiv.org/abs/1903.10955 (visited on 23/01/2023).

[9] R. Mottaghi, Y. Xiang and S. Savarese, *A Coarse-to-Fine Model for 3D Pose Estimation and Sub-category Recognition*, arXiv:1504.02764 [cs], Apr. 2015. DOI: 10.48550/arXiv.1504.02764. [Online]. Available: http://arxiv.org/abs/1504.02764 (visited on 23/01/2023).

[10] M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce and K. Daniilidis, 'Single image 3D object detection and pose estimation for grasping,' in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, ISSN: 1050-4729, May 2014, pp. 3936–3943. DOI: 10.1109/ICRA.2014.6907430.

[11] P. Li, X. Chen and S. Shen, *Stereo R-CNN based 3D Object Detection for Autonomous Driving*, arXiv:1902.09738 [cs], Apr. 2019. DOI: 10.48550/arXiv.1902.09738. [Online]. Available: http://arxiv.org/abs/1902.09738 (visited on 23/01/2023).

[12] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell and K. Q. Weinberger, *Pseudo-LiDAR from Visual Depth Estimation: Bridging the Gap in 3D Object Detection for Autonomous Driving*, arXiv:1812.07179 [cs], Feb. 2020. DOI: 10.48550/arXiv.1812.07179. [Online]. Available: http://arxiv.org/abs/1812.07179 (visited on 23/01/2023).

[13] X. Chen, H. Ma, J. Wan, B. Li and T. Xia, *Multi-View 3D Object Detection Network for Autonomous Driving*, arXiv:1611.07759 [cs], Jun. 2017. DOI: 10.48550/arXiv.1611.07759. [Online]. Available: http://arxiv.org/abs/1611.07759 (visited on 27/01/2023).

[14] J. Ku, M. Mozifian, J. Lee, A. Harakeh and S. Waslander, *Joint 3D Proposal Generation and Object Detection from View Aggregation*, arXiv:1712.02294 [cs], Jul. 2018. DOI: 10.48550/arXiv.1712.02294. [Online]. Available: http://arxiv.org/abs/1712.02294 (visited on 27/01/2023).

[15] B. Yang, M. Liang and R. Urtasun, *HDNET: Exploiting HD Maps for 3D Object Detection*, arXiv:2012.11704 [cs], Dec. 2020. DOI: 10.48550/arXiv.2012.11704. [Online]. Available: http://arxiv.org/abs/2012.11704 (visited on 27/01/2023).

[16] T. Yin, X. Zhou and P. Krähenbühl, *Center-based 3D Object Detection and Tracking*, arXiv:2006.11275 [cs], Jan. 2021. DOI: 10.48550/arXiv.2006.11275. [Online]. Available: http://arxiv.org/abs/2006.11275 (visited on 19/01/2023).

[17] C. R. Qi, H. Su, K. Mo and L. J. Guibas, *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*, arXiv:1612.00593 [cs], Apr. 2017. DOI: 10.48550/arXiv.1612.00593. [Online]. Available: http://arxiv.org/abs/1612.00593 (visited on 27/01/2023).

[18] C. R. Qi, L. Yi, H. Su and L. J. Guibas, *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*, arXiv:1706.02413 [cs], Jun. 2017. DOI: 10.48550/arXiv.1706.02413. [Online]. Available: http://arxiv.org/abs/1706.02413 (visited on 27/01/2023).

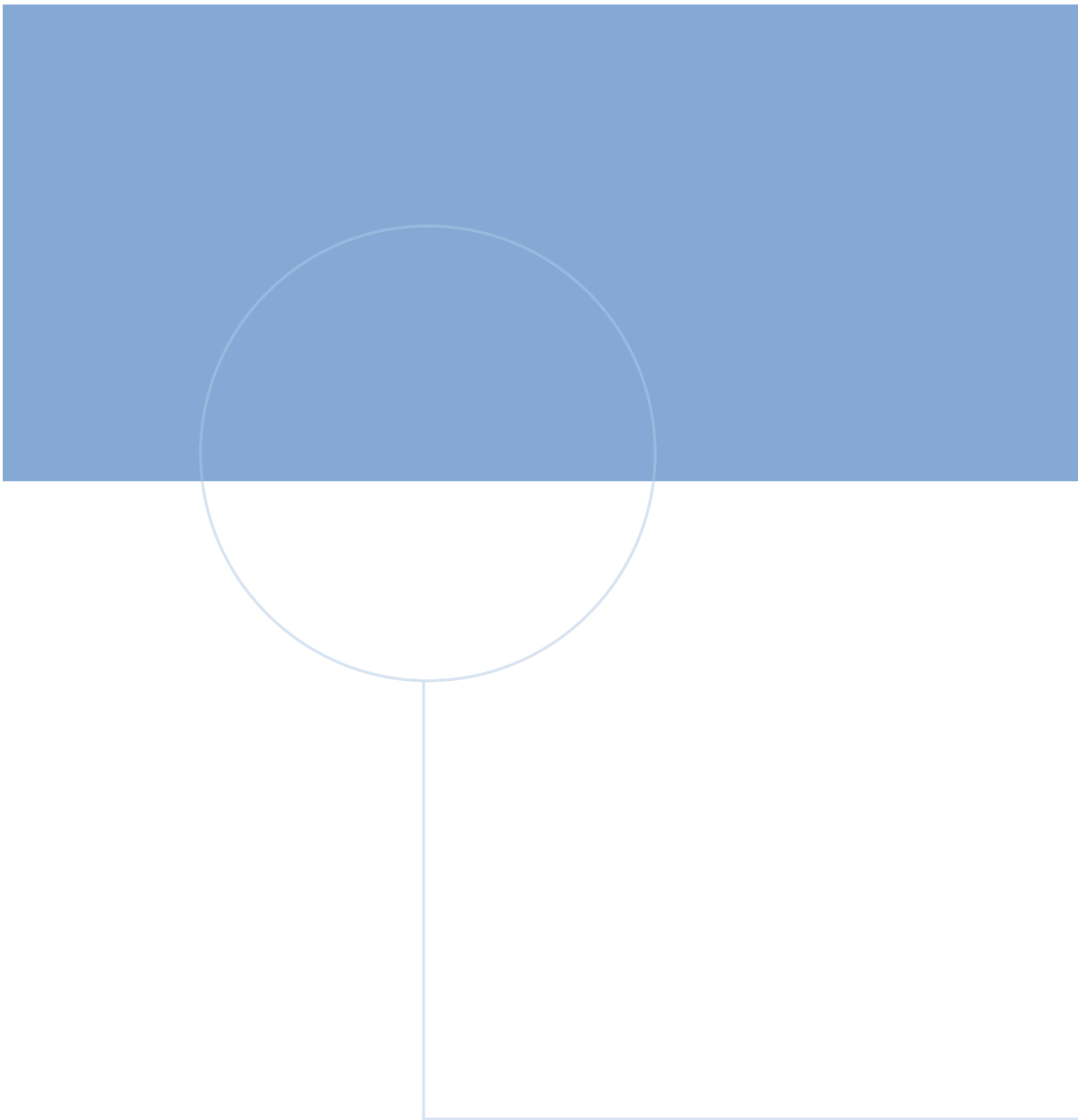
[19] Y. Zhou and O. Tuzel, *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*, arXiv:1711.06396 [cs], Nov. 2017. DOI: 10.48550/arXiv.1711.06396. [Online]. Available: http://arxiv.org/abs/1711.06396 (visited on 06/12/2022).

[20] C. R. Qi, W. Liu, C. Wu, H. Su and L. J. Guibas, *Frustum PointNets for 3D Object Detection from RGB-D Data*, arXiv:1711.08488 [cs], Apr. 2018. DOI: 10.48550/arXiv.1711.08488. [Online]. Available: http://arxiv.org/abs/1711.08488 (visited on 30/01/2023).

[21] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang and O. Beijbom, *PointPillars: Fast Encoders for Object Detection from Point Clouds*, arXiv:1812.05784 [cs, stat], May 2019. [Online]. Available: http://arxiv.org/abs/1812.05784 (visited on 30/01/2023).

[22] J. Stanisz, K. Lis, T. Kryjak and M. Gorgon, 'Optimisation of the PointPillars network for 3D object detection in point clouds,' in *2020 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, arXiv:2007.00493 [cs, eess], Sep. 2020, pp. 122–127. DOI: 10.23919/SPA50552.2020.9241265. [Online]. Available: http://arxiv.org/abs/2007.00493 (visited on 30/01/2023).

[23] A. Geiger, P. Lenz and R. Urtasun, 'Are we ready for autonomous driving? The KITTI vision benchmark suite,' en, in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI: IEEE, Jun. 2012, pp. 3354–3361, ISBN: 978-1-4673-1228-8 978-1-4673-1226-4 978-1-4673-1227-1. DOI: 10.1109/CVPR.2012.6248074. [Online]. Available: http://ieeexplore.ieee.org/document/6248074/ (visited on 14/03/2023).



NTNU

Norwegian University of
Science and Technology