Master's thesis

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

Åsmund Brekke, Fredrik Vatsendvik

3D Object Detection for Autonomous Driving Using Real and Simulated Data

Master's thesis in Computer Science
Supervisor: Frank Lindseth
June 2019

NTNU
Norwegian University of Science and Technology

Åsmund Brekke, Fredrik Vatsendvik

3D Object Detection for Autonomous Driving Using Real and Simulated Data

master project, spring 2019

NTNU Autonomous Perception Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering

Supervisor: Frank Lindseth

Abstract

Autonomous driving has gained increasing attention in the past decade, both due to the broadening public interest in artificial intelligence and astounding results achieved by research institutions and companies such as Google, Uber and Tesla. An autonomous driving system needs to perform many challenging tasks such as localizing itself precisely in the world, detecting and adapting to the behavior of other actors in the environment, and planning how it will get from point A to B.

Each of these tasks pose a number of challenges and need to be performed efficiently and concurrently in real-time. In order to generalize to a large variety of scenarios, these systems often require a large amount of labeled training data, which can be very expensive to obtain and annotate. Commonly, human expertise is required to either manually label the data or to supervise and correct automatic generation of labels.

In this thesis, we investigate how unsupervised generation of labeled data from simulation can be used to lessen the requirement for real-world data and speed up convergence for object detection architectures. In addition, we perform several experiments with LIDAR feature map representations projected to a Bird's Eye View (BEV) in order to investigate potentials for improvements in run-time or accuracy with sensor fusion between camera and LIDAR.

We first present the Carla Automated Dataset Extraction Tool (CADET) – a novel tool for generating training data from the autonomous vehicle simulator CARLA [Dosovitskiy et al., 2017]. This tool is used to generate a dataset of 10,000 samples, including 3D bounding box labels for cars and pedestrians, followed by a statistical evaluation on the distribution of classes, bounding box sizes, number of objects per sample and orientations with comparison to the KITTI dataset [Geiger et al., 2013]. The CADET dataset is used to pre-train a number of 3D object detection models using the AVOD-FPN [Ku et al., 2018] and SECOND [Yan et al., 2018] architectures, followed by fine-tuning on the KITTI dataset. Models are trained in single-class and multi-class variations and used to evaluate potential improvements to model accuracy, convergence speed and reduction in the required amount of real-world training data following simulated pre-training. Additionally, AVOD-FPN is modified and tested with multiple novel BEV configurations trained on both KITTI and CADET, comparing performance to the default configuration. Finally, a selection of multi-class models is used to perform inference on a locally collected dataset in arctic environments, curated by NTNU Autonomous Perception (NAP). The inference results from a collection of models are visually evaluated, comparing their ability to generalize to the unseen environment with and without simulated pre-training.

Our findings indicate that the use of simulated data for pre-training 3D object detectors can contribute to improving accuracy, especially when real-world data is limited, as well as offering significantly faster convergence during training on new data. We also find that simulated data can aid generalization, especially for the SECOND architecture which only uses LIDAR. Lastly, two of our introduced BEV configurations show a relative improvement in 3D detection of pedestrians of 12-19% when compared to the default configuration for AVOD-FPN. Another configuration shows similar performance on cars, with 4-5% faster inference, however with considerably worse performance on pedestrians.

Sammendrag

Selvkjørende biler har blitt et populært tema det siste tiåret, både grunnet bredere offentlig interesse i kunstig intelligens og imponerende resultater oppnådd av forskninsinstitusjoner og private selskaper som Google, Uber og Tesla. Et system for autonom kjøring må kunne utføre mange krevende oppgaver, blant annet å lokalisere seg selv nøyaktig i omverdenen, detektere og tilpasse seg oppføreselen til aktører i omgivelsene, og planlegging av hvorden det skal komme seg punkt A til punkt B.

Hver av disse oppgavene innebærer en rekke utfordringer og må utføres effektivt og i sanntid. For å kunne generalisere til en stor variasjon med scenarier krever disse systemene store mengder annotert data, som kan være dyrt å anskaffe og annotere. Typisk kreves menneskelig ekspertise for å enten manuelt merke data eller overåke og korrigere automatisk generering av merket data.

I denne oppgaven undersøker vi hvordan uovervåket generering av merket data fra simulering kan bli brukt for å redusere nødvendigheten for data fra den virkelige verden og øke hastigheten for konvergens hos systemer for 3D objektdeteksjon. I tillegg utfører vi flere eksperimenter for ulike representasjoner av LIDAR punktskyer projisert til fugleperspektiv (BEV) for å undersøke potensialet for forbedringer innen kjøretid eller presisjon med sensor fusjon mellom kamera og LIDAR.

Først presenterer vi verktøyet Carla Automated Dataset Extraction Tool (CADET) – et nytt verktøy for å generere treningsdata fra simulatoren CARLA [Dosovitskiy et al., 2017] for selvkjørende biler. Dette verktøyet benyttes for å generere et datasett på 10,000 datapunkter, inkludert 3D avgrensingsbokser for biler og fotgjengere. Videre følger en statistisk analyse av distribusjonen av klasser, størrelse på avgrensingsbokser, objekter per datapunkt og orienteringer sammenlignet med datasettet KITTI [Geiger et al., 2013]. CADET datasettet brukes for å pretrene et sett med modeller for 3D objektdeteksjon ved bruk av modellene AVOD-FPN [Ku et al., 2018] og SECOND [Yan et al., 2018], etterfulgt av finjustering på KITTI-datasettet. Modellene blir trent i konfigurasjoner for enkeltklassifisering og multiklassifisering og brukt for å evaluere potensielle forbedringer i treffsikkerhet, hastighet for konvergens og reduksjon i behovet for data fra den virkelige verdenen etter simulert pretrening. I tillegg blir AVOD-FPN modifisert og teset med flere nye BEV konfigurasjoner trent på både KITTI og CADET, hvor ytelse sammenlignes med standardkonfigurasjonen. Til slutt blir et utvalg multiklassemodeller brukt for å utføre inferens på et datasett i arktiske omgivelser samlet inn av NTNU Autonomous Perception (NAP) i Trondheim. Inferensresultatene fra modellene blir visuelt evaluert ved å sammenligne deres evne til å generalisere til usette omgivelser med og uten simulert pretrening.

Våre funn indikerer at bruken av simulert data for pretrening av 3D objektdetektorer kan bidra til økt treffsikkerhet, spesielt når data fra den virkelige verden er begrenset. Resultatene presentert i denne oppgaven viser også at pretrening kan tilby langt raskere konvergens under trening på ny data. Vi merker oss også at simulert data can hjelpe generalisering, særlig for SECOND-modellen som kun bruker LIDAR. Vi har til slutt introdusert to nye BEV konfigurasjoner som viser en relativ forbedring i 3D deteksjon av fotgjengere på 12-19% når sammenlignet med standardkonfigurasjonen for AVOD-FPN. En annen konfigurasjon viser lignende ytelse på biler med 4-5% raskere inferens, men med betydelig lavere ytelse på fotgjengere.

Preface

This project was done as a part of the NAP project, which consists of multiple Master's theses conducted at the Norwegian University of Science and Technology (NTNU), Faculty of Information Technology and Electrical Engineering. All these Master's theses share the commonality that they are researching topics within autonomous driving. Parts of this work was presented at the 2019 Symposium of the Norwegian AI Society in form of an oral presentation and an accompanying paper, the latter of which can be found in Appendix F. The paper will also be available through Springer's CCIS series, with a preprint currently available [Brekke et al., 2019]. The authors would like to thank Frank Lindseth for his continued support throughout this project. Frank played a large role in deciding the research topic for this thesis and has provided helpful advice along the way. In addition, Frank has been the main driving force behind NAP, and our experiments could not have been performed without his help.

Fredrik would like to thank his friends for numerous discussions and ideas related to the project, and his parents for continued support during the writing of this thesis.

Åsmund would like to thank his friends and family for encouragement during the writing of this thesis. In particular, he would like to thank Carl Otto Steen for giving helpful feedback and assisting with proofreading.

Åsmund Brekke, Fredrik Vatsendvik
Trondheim, June 14, 2019

Contents

List of Figures

List of Tables

List of Acronyms

ALV Autonomous Land Vehicle

AOS Average Orientation Similarity

API Application Programming Interface

BAIR Berkeley Artificial Intelligence Research

BDD100K Berkeley Deep Drive 100K

BDD Berkeley Deep Drive

BEV Bird's Eye View

BRN Box Regression Network

CADET Carla Automated Dataset Extraction Tool

CNN Convolutional Neural Network

E-RPN Euler Region Proposal Network

EFL Effective Focal Lens

EOL End Of Life

FMCW Frequency Modulated Continuous Wave

FOV Field Of View

FPN Feature Pyramid Network

FPS Frames Per Second

FV Front View

GNSS Global Navigation Satellite System

GPS Global Positioning System

HIL Hardware In the Loop

HOG Histogram of Oriented Gradients

ILSVRC ImageNet Large Scale Visual Recognition Challenge

IMU Inertial Measurement Unit

IoU Intersection over Union

KIT Karlsruhe Institute of Technology

LIDAR LIght Detection And Ranging

mAP mean Average Precision

NAP NTNU Autonomous Perception

NMS Non-Maximum Suppression

NTNU Norwegian University of Science and Technology

RADAR RAdio Detection And Ranging

RNN Recurrent Neural Network

ROS Robot Operating System

RPN Region Proposal Network

ReLU Rectified Linear Unit

RoI Region of Interest

SSD Single Shot Detector

SVM Support Vector Machine

ToF Time-of-Flight

VFE Voxel Feature Encoding

DoF Depth of Field

MLP Multilayer Perceptron

GPU Graphics Processing Unit

Chapter 1

Introduction

This chapter aims to give a brief description of the tasks this projects aims to solve, and the motivation behind it. Furthermore the main contributions of this thesis are explained shortly, followed by a description of the thesis structure.

1.1 Background and Motivation

The field of autonomous driving have achieved incredible results since its inception. With works such as Carnegie Melon's NavLab and Autonomous Land Vehicle (ALV) [Thorpe et al., 1988], the first end-to-end system in ALVINN [Pomerleau, 1989], to today's autonomous vehicles developed by companies such as Waymo and Uber. The perception systems of autonomous vehicles have developed from simple color segmentation techniques as in ALV, to complex object detection and instance segmentation using deep convolutional neural networks present in most modern autonomous vehicle systems. This increase in complexity has lead to a need for more labeled data, which can be expensive to obtain. In the past decades autonomous perception has mainly focused on a single sensor – the camera. However, since all sensors have their own strengths and weaknesses, there has been increasing attention into combining information from different sensors to utilize the strengths of multiple sensors together, often referred to as *sensor fusion*. Although sensor fusion architectures can achieve better performance in some instances, they require careful synchronization between sensors, and use multimodal datasets which are often more strenuous to label and curate.

In light of these factors, the field of autonomous driving has in recent years invested resources into developing advanced driving simulators, which can easily simulate a vast amount of sensors, environments and actors and are highly configurable. Some of these simulators are open-source projects, making them easy to use for research. Despite this, the simulators are generally not capable of exporting sensor data out of the box, and are usually meant as a way of validating existing machine learning models whether it be object detection or reinforcement learning.

1.2 Goals and Research Questions

Goal Develop a state-of-the-art autonomous vehicle at NTNU

This thesis is a part of a larger team at NTNU researching autonomous vehicles, called NTNU Autonomous Perception (NAP). Therefore, some of the work presented in this report is not directly con-

nected to the main experiments, but rather serves as a springboard for further research in the NAP team. Norway will be a large consumer of autonomous technologies in the years to come, and are investing resources into developing autonomous technologies for vehicles including ships, ferries, trucks and cars. With regards to autonomous driving, NAP believes that it can contribute to the field by developing and testing autonomous systems in the demanding and dynamic arctic environment. A long term goal of NAP is to generate its own datasets from this environment, and to offer exiting projects for students interested in the field. The authors aims to answer questions regarding the real-time performance and accuracy of multimodal 3D object detection systems utilizing RGB cameras and LIDAR laser, and how the use simulated data affects these metrics. Making use of the state-of-the-art architectures, AVOD-FPN for multimodality and SECOND for single modality as a basis for investigation, the following questions will receive focus for research and discussion:

Research question 1 Will state-of-the-art 3D object detection architectures be able to detect multiple classes of objects at different sizes in a single forward pass, maintaining real-time performance and sufficient accuracy?

Research question 2 Will a model trained solely on simulated data be able to generalize to unseen, real world environments with different weather and lighting conditions?

Research question 3 Can a model trained on simulated data achieve similar performance to full training on real sensor data after limited fine-tuning?

Research question 4 Can different feature map representations of LIDAR sensor data improve performance for sensor fusion, without the cost of slower inference?

Research question 5 Does multimodal architectures generalize better to unseen environments than solely LIDAR based architectures?

Research question 6 Will the addition of simulated pre-training improve results when tested on unseen arctic environments?

1.3 Research Method

The work presented in this thesis can be divided into several projects, each with their own goal. First, we develop a tool capable of generating datasets for autonomous driving. This tool will be analyzed based on the data it generates, the simulator configurations, and the implemented algorithms. When evaluating the implemented algorithms, we will first present the theoretic background, and then evaluate the algorithms performance analytically. The generated dataset will also be compared to another dataset for autonomous driving in order to highlight its strengths and weaknesses. As the main goal of a simulator is to mimic the real world, we will perform the majority of analysis as a comparison between the simulated data and its real world counterpart. For instance, simulated sensor data such as from LIDAR or camera are compared to sensor data from real world datasets.

With regards to the machine learning models implemented in this thesis, all experiments are conducted in a manner where each experiment is designed to answer a given hypothesis. Some of these hypotheses are in the form of research questions, whereas other experiments are intended to partially answer one or more research questions. For instance, each experiment with LIDAR feature map representations are intended to answer research question 4 - *Can different feature map representations of LIDAR sensor data improve performance for sensor fusion, without the cost of slower inference?*, but also provide a larger collection of results to support conclusions regarding pre-training on simulated data. Lastly, we perform a visual analysis of the trained machine learning models on the NAP dataset. Compared to the

previous experiments, where we deploy commonly used metrics such as mean Average Precision (mAP) to evaluate the trained models, the analysis of NAP data has to be solely visual due to the lack of labeled data.

The authors of this thesis intends for every step of all experiments to be reproducible. This is a critical part of research, which in the field of machine learning is often forgotten or disregarded. We intend to release all code, models and datasets used for this thesis in order for others to be able to validate our findings. Note that the NAP dataset can not be released at the time of this writing, due to privacy issues. All privacy related content in the images from the NAP dataset, such as license plates or recognizable faces in this thesis are therefore anonymized.

1.4 Contributions

In this thesis we present several contributions we believe to be novel and interesting, and belonging to several different fields. The most important contributions in no particular order are:

- Creating an open-source framework for easily extracting training data including 2D/3D bounding boxes, lidar point clouds, ground plane data and labels for multiple classes from the CARLA simulator
- Improving mAP scores on the KITTI validation set for the AVOD architecture with several novel lidar feature map representations
- Evaluating our trained models on a new dataset generated in an artic environment in Trondheim
- Expanding the AVOD and SECOND architecture to run in real-time as ROS nodes
- Publishing models for AVOD-FPN and SECOND trained on real and simulated data with improvements to generalization for unseen arctic environments
- Presenting parts of our work at the Norwegian AI Society Symposium 2019 and publishing the results in a peer-reviewed paper in Springer's Communications in Computer and Information Science

The significance of each of these contributions are described in detail in Chapter 7.

1.5 Thesis Structure

Chapter 1: Introduction introduces the reader to the the motivation behind this thesis, and what problems the thesis aims to answer. In addition, it gives a brief introduction to the contributions presented in this work.

Chapter 2: Background Theory and Motivation gives a brief introduction to the area of autonomous driving, including sensor types, popular datasets and state of the art object detection architectures. The chapter is intended to introduce the reader to all concepts necessary to follow the experiments and results discussed in later chapters.

Chapter 3: Related work discusses state-of-the-art 3D object detection architectures, with an emphasis on different techniques for performing 3D object detection.

Chapter 4: Architecture describes core functionality of the main object detection architecture used in this thesis and details the modifications that were applied to the model by the authors.

Chapter 5: Experiments and Results introduces the reader to the experiments that were performed in this thesis, with an emphasis on reproducibility. Furthermore, the chapter displays the results of each experiment and performs a brief analysis of the results

Chapter 6: Discussion critically evaluates the results presented in Chapter 4, and makes comments about positive and negative aspects of the results.

Chapter 7: Conclusion and Future Work relates the results presented in this thesis to the research questions, and attempts to draw conclusions where applicable. In addition, the chapter discusses possible improvements to our presented work, with regards to the data generation tool and the trained models. Lastly, the chapter gives a short statement about reproducibility and how to replicate the work presented in this thesis.

Chapter 2

Background Theory and Motivation

Autonomous driving has seen tremendous improvements in previous years, in particular with regards to perception. Advances in convolutional architectures, modalities for sensing, availability of high-quality autonomous driving datasets and increased realism in driving simulators have made autonomous vehicles perform exceptionally well. In October 2018, Google Waymo's CEO John Krafcik reported that its self-driving Car fleet had driven 10 million miles on US public roads [Krafcik, 2018]. Moreover, it has also driven *7 billion* miles in a virtual environment, highlighting the importance simulators have in the field.

The technology behind autonomous driving spans several fields and disciplines. This chapter will introduce some important concepts within the field of autonomous driving, and in particular within autonomous perception. These are categorized into sections of Sensors, Datasets, Camera projection, Software, Hardware, Transfer Learning, and Deep Learning. Some familiarity with neural networks and in particular convolutional neural networks is assumed. The authors recommend the book Deep Learning by Goodfellow et al. [2016] for a thorough introduction to neural networks.

2.1 Sensors

An autonomous vehicle perceives the environment using sensors, which are needed in order to perform some of its most important tasks, including *perception* and *localization*. The most common sensors included in autonomous vehicles are LIDAR, RADAR, Camera, GPS, and IMU. This section discusses these sensors with regards to price, the underlying principle of each sensor, its strengths and weaknesses and lastly how they are used in autonomous vehicle systems.

2.1.1 LIDAR

LIght Detection And Ranging (LIDAR) is a sensor that works by emitting laser beams and measuring the time it takes for the laser to bounce back to the sensor, often referred to as the Time-of-Flight (ToF). Therefore, a LIDAR sensor can detect the distances of points, often in 360 degrees as a result of mechanical rotation. The resulting output is thus a sparse *point-cloud*. LIDARs are excellent at detecting distances to objects and can operate with a range of more than 200m[1]. However, the semantic information contained in these point clouds are hard to extract, and extracting semantic information from point clouds is an active topic of research within computer vision. LIDAR sensors are beneficial in autonomous vehicles

[1]This is dependent on the LIDAR sensor. Notable examples such as the Velodyne Ultra Puck and the Pandora Hesai have an operational range of 200m, while the newer solid state LIDAR PandarGT 3.0 by Hesai supports an extended range of 300m.

mainly because of depth information, and since they perform well both with and without the presence of external light. However, particles such as dust, ice and snow may affect the quality of the LIDAR [Goodin et al., 2019]. LIDARs are normally quite expensive, costing up towards 100,000$. However, both Google Waymo and Velodyne are releasing their own low-price LIDAR sensors, which are stated to have a price below 10,000$ [Korosec, 2019; Velodyne Lidar Inc., 2019] as well as the development of solid state LIDARs potentially lowering prices even further in coming years [Wang, 2019].

2.1.2 RADAR

RAdio Detection And Ranging (RADAR) is a ToF-sensor using radio waves. RADARs are already used in vehicle applications, from blind spot detection in trucks to assistance in cruise-control on passenger vehicles. Currently, the most popular type of RADAR for autonomous vehicles is Frequency Modulated Continuous Wave (FMCW) RADAR, which works by emitting radio wave signals (often referred to as *carriers*) with varying frequencies. These signals are emitted continuously, so they often require a separate antenna for emitting and receiving. It is not necessary to vary the frequencies if detecting only distance. However, the variation of frequencies allows one to determine the relative velocity of the object the point belongs to, as discussed by Parker [2013]. Since RADARs utilize radio waves instead of light, they are relatively insensitive to changing weather conditions. On the other hand, RADARs perform poorer than LIDARs on small objects, since very few of the radio wave signals will reflect off the object if its target surface is small, such as in the case of a pedestrian, cyclist or motorbike. In addition, radio waves act very differently based on the reflective surface. Large metal objects such as trucks and cars reflect radio waves very well, while pedestrians and cyclists reflect radio waves poorly. This can cause these small objects to be nearly invisible from a RADAR when situated near a large vehicle.

2.1.3 Camera

Cameras maintain an important role in autonomous vehicles for several reasons. Understanding image data has been an important part of computer vision, and there exists a plethora of techniques for performing object detection, instance segmentation, depth estimation, and object tracking. In addition, cameras are widespread and mass-produced and are relatively cheap compared to other sensors such as LIDAR. Cameras are *passive sensors* that detect light and do so by letting photons pass through a lens and onto a light sensor. A simplified version of a camera model is shown in Figure 2.1. Although cameras give excellent semantic information in a limited Field Of View (FOV) with high resolution, dense images, they inherently lack depth information and give drastically reduced quality under poor lighting conditions. Therefore, additional sensors that can operate in night-time conditions are often employed together with cameras.

2.1.4 GPS

Global Positioning System (GPS) is a Global Navigation Satellite System (GNSS) developed by the United States Department of Defense. GPS is used for localization and works by measuring the transmission time between the GPS sensor and multiple in-orbit satellites. In autonomous vehicles, it is critical to have a very precise measurement of the vehicle's location relative to the road. Therefore, one often uses high-precision GNSS consisting of multiple GNSS constellations [NovAtel Inc, 2019]. These constellations are groups of satellites and include the American GPS, the Russian GLONASS, the Chinese Beidou and the European Galileo. By combining these measurements, one can achieve high precision localization with centimeters of accuracy. However, this is dependent on strong GNSS signals, which can be weakened in urban environments, or non-existent in tunnels. Therefore, an autonomous vehicle will often also contain an Inertial Measurement Unit (IMU).

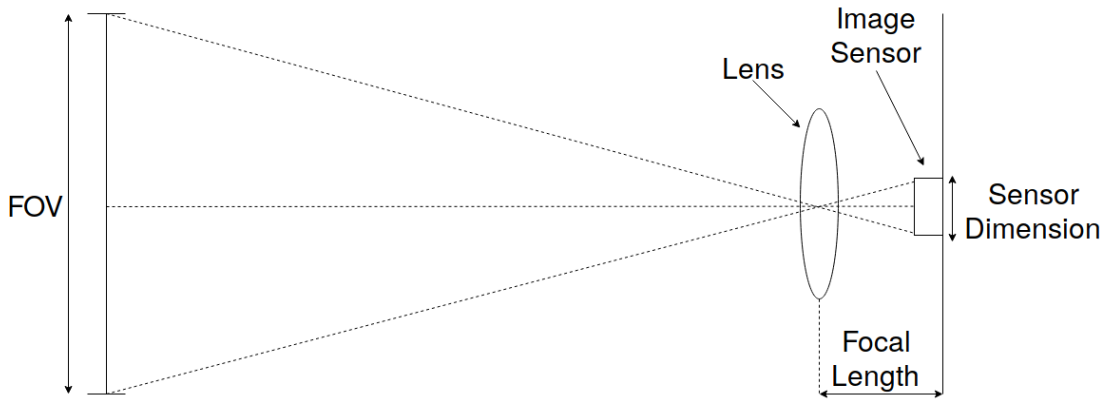


Figure 2.1: Lens and image sensor model

2.1.5 IMU

IMU is a sensor consisting of accelerometers and gyroscopes that measure the rotation and movement in the X, Y and Z axis defined internally in the sensor. IMU are often useful in autonomous vehicles since they can, given an initial starting point, determine the relative location of the vehicle. In coordination with a high precision GPS, IMU is often used when the GPS signals are deemed too weak. In addition, IMUs can be used to measure instability in the vehicle, and thus give information about the quality of the ride. For example, the IMU sensor readings may give an indication of a bumpy road which in an autonomous driving system could lead to lowering the speed of the vehicle.

2.2 Datasets for autonomous driving

Machine learning models nowadays usually employ deep neural networks with millions of parameters and are trained in a supervised fashion, which means that they learn by examples of input and the corresponding output. To gather these examples is often very expensive – especially for autonomous perception where a recording vehicle with the appropriate sensors is required. The cost associated with labeling data is also large, not only because it requires a human in the loop, but also because the models need a large amount of labeled data to tune their parameters. The most common way of performing research within autonomous perception is therefore to utilize an existing dataset. This section covers some of the most popular datasets for autonomous driving, with a particular focus on object detection. The datasets covered are KITTI, BDD100K, nuScenes, and ApolloScape.

2.2.1 KITTI

The KITTI dataset [Geiger et al., 2013] is a dataset for autonomous driving created by Karlsruhe Institute of Technology (KIT) and the Toyota Technological Institute at Chicago. The dataset is generated from driving around Karlsruhe, Germany, and consists of mono and color camera stereo images, GPS, IMU and LIDAR data. Among other things KITTI contains annotated 3D bounding boxes for cars, trucks, pedestrians, trams, and cyclists, as well as image segmentations of roads and lanes.

The KITTI dataset is a very common benchmark for numerous tasks regarding autonomous driving and perception, including object detection, instance segmentation, depth estimation, and odometry. However, since the dataset contains a limited number of training samples, all sampled from the same general area, it does not contain a large variety of driving scenarios. Most of the images in the dataset contain

sunny weather and depicts an urban environment. An example image is shown in Figure 2.2. Therefore, models trained on this dataset may fail to generalize to other driving scenarios, such as in conditions with rain or snow. The KITTI object detection dataset consists of 7481 labeled training images and 7518 test images without published labels as these are used for official submissions to the KITTI leaderboards [Geiger, 2019].

Figure 2.2: Sample image from the KITTI dataset, created by Geiger et al. [2013]. Available at `http://www.cvlibs.net/datasets/kitti/` under a Creative Commons Attribution 3.0 `https://creativecommons.org/licenses/by-nc-sa/3.0/`

2.2.2 BDD100K

The Berkeley Deep Drive 100K (BDD100K)[Yu et al., 2018] is a dataset compiled by the Berkeley Artificial Intelligence Research (BAIR) intended for autonomous driving research. With its vast amount of labeled images numbering over 100,000, containing 2D bounding box labels for classes such as Bike, Person, Traffic Light, and Traffic Sign, BDD100K is one of the largest datasets for object detection in autonomous driving research. In addition to 2D bounding boxes, the dataset offers over 10,000 pixel-level segmented images, with segmentations for drivable areas, full roads and lanes. A sample image is shown in Figure 2.3.

Compared to the KITTI dataset, BDD100K offers a variety of weather conditions such as rain, snow, sun and fog, and is sampled from multiple geographical locations including New York, Berkeley, San Francisco and the Bay Area, whereas KITTI is sampled solely from Karlsruhe. The vastness and diversity of BDD100K allow machine learning models to better generalize to unseen environments, whereas models trained only on KITTI would be expected to perform worse in other weather conditions. However, BDD100K does not contain multi-modal sensor data such as LIDAR or RADAR which makes it unusable for sensor fusion architectures. The authors of BDD100K state that the inclusion of multi-modal data is something they hope to provide ”in the near future” [Yu, 2018].

2.2.3 nuScenes

nuScenes [Caesar et al., 2019] is a large dataset for autonomous driving curated by nuTonomous, which is a part of Aptiv – a large auto parts company. The dataset is collected from both Boston and Shanghai, which means that it contains scenarios with both left and right-handed traffic. In addition, the dataset contains a vast sensor suite, including one LIDAR, six cameras, five RADARs, one IMU and GPS. The dataset is not only intended for object detection in autonomous driving but also for 3D tracking, which means that each object has attributes such as *standing* or *lying down* for pedestrians, and *moving*, *stopped* and *parked* for vehicles, as well as attributes for pose and visibility.

Figure 2.3: Sample image from the BDD100K dataset, created by Yu et al. [2018].

In total, the dataset contains annotations for 23 classes, including but not limited to Animal, Pedestrian Adult, Pedestrian Child, Pedestrian Construction worker, Bus, Truck, Motorcycle, and Bicycle. The labels are generated by Scale.ai, which is detailed in Section 2.4.2. The nuScenes dataset contains 1.4 million camera images and 390 000 LIDAR scans gathered from over 15 hours of driving in urban environments. A large problem in generating a dataset in autonomous driving is the inherent class imbalance when driving. Most of the objects in a given scene will be vehicles, and machine learning models may have problems performing well on the underrepresented classes. Therefore, nuScenes oversamples scenes where there are a lot of minority classes represented. The data gathered in nuScenes are one thousand scenes, each lasting for 20 seconds. This amounts to 1.4 million 3D bounding boxes, annotated at a frequency of 2 Hertz. Compared to KITTI, nuScenes contains over 7 times the amount of object labels and has a larger variety of driving scenarios. In addition, KITTI only contains 8 classes and only uses Car and Pedestrian in its evaluation, since the other classes are so underrepresented in the dataset that it does not warrant an evaluation. Note that the nuScenes dataset is only free to use for non-commercial purposes. A sample scene is shown in Figure 2.4.

2.2.4 ApolloScape

The ApolloScape dataset Huang et al. [2018] was released in the beginning of 2018 and contains multiple datasets for tasks such as instance segmentation, trajectory prediction and 3D object detection. The data is gathered from four regions in China and contains multiple weather types, including snow, rain and fog. For object detection, the ApolloScape data offers two datasets. The first consists of 70,000 pixel-level labeled images with 3D bounding boxes for Cars, while the second contains approximately 6,000 labelled LIDAR point clouds with 3D bounding boxes for the classes small vehicle, large vehicle, pedestrian, cyclist or motorcyclist, and traffic cones. However, evaluation is only performed for small vehicles, large vehicles, pedestrians and cyclists. The evaluation metric for the LIDAR-based dataset is identical to Geiger et al. [2013]. While the former contains data from several regions in China, the latter is only sampled from Beijing. This means that the dataset may contain a limited number of driving scenarios, but these scenarios are very rich in information, as they are captured in high-traffic areas with many labels per image. Figure 2.5 shows a roadmap of the ApolloScape progress including examples of masks and projected depth.

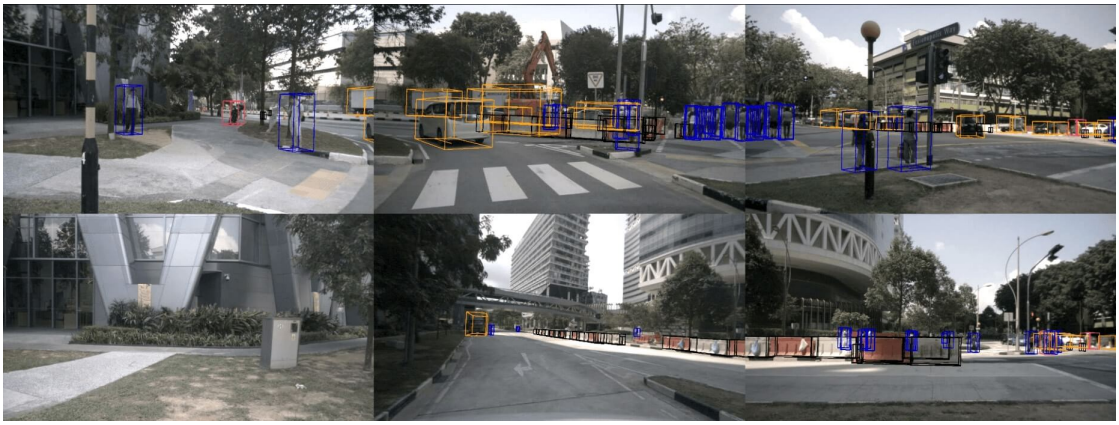


Figure 2.4: Sample image from Shanghai in the nuScenes dataset created by Caesar et al. [2019]. The image shows the six cameras - Front Left, Front, Front Right, Back Left, Back, and Back Right from top left to bottom right respectively.

2.3 3D projection

When dealing with sensors and simulators, one needs tools for transferring points in one coordinate system into another. In particular, it is necessary to project points from the world onto other coordinate systems such that they are relative to the camera, image plane or other sensors. For a point $\mathbf{X} = (X, Y, Z, 1)$ in a 3D world, we can describe the transformation from 3D coordinates to sensor coordinates as a series of matrix multiplications[2]. The transform from a 3D point to its pixel coordinates in the camera frame, (u,v) can be described by rotating and translating the point to camera coordinates via the rotation matrix R, and the translation between the camera and the world origin, t, followed by a projection from camera coordinates to the image plane, described by K. This transformation is described in Equation (2.1). α_x and α_y are the focal lengths of the camera in pixels in the x and y axis, while C_u and C_v represent the principal point in pixels coordinates. The pinhole camera model is a special version of this generalized projection model, with $\alpha_x = \alpha_y$ and $s = 0$. These conditions assume that the pixels are square, that the skew of the projection is zero, and that C_u and C_v is known [Hartley and Zisserman, 2004, p. 185]. The parameters in K are often referred to as the intrinsic parameters of the camera, while the parameters for translation and rotations are considered extrinsic parameters. For a camera with a given FOV, the focal length in pixels can be calculated. This is shown in Equation (2.2) for the horizontal focal length. Note that the horisontalAngleView is expressed in radians.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P\mathbf{X} = K[R|t]\mathbf{X} = \begin{bmatrix} \alpha_x & s & C_u \\ 0 & \alpha_y & C_v \\ 0 & 0 & 1 \end{bmatrix} [R|t]\mathbf{X} \tag{2.1}$$

$$f = \frac{width}{2 * tan(\frac{horisontalAngleView}{2})} \tag{2.2}$$

2.3.1 KITTI sensor calibration

In the KITTI dataset, sensor calibration is performed on each day of recording. In addition, each sensor uses a unique coordinate system, so a transform between coordinate systems is required to project points

[2]Disregarding the non-linear corrections for effects such as barrel distortion

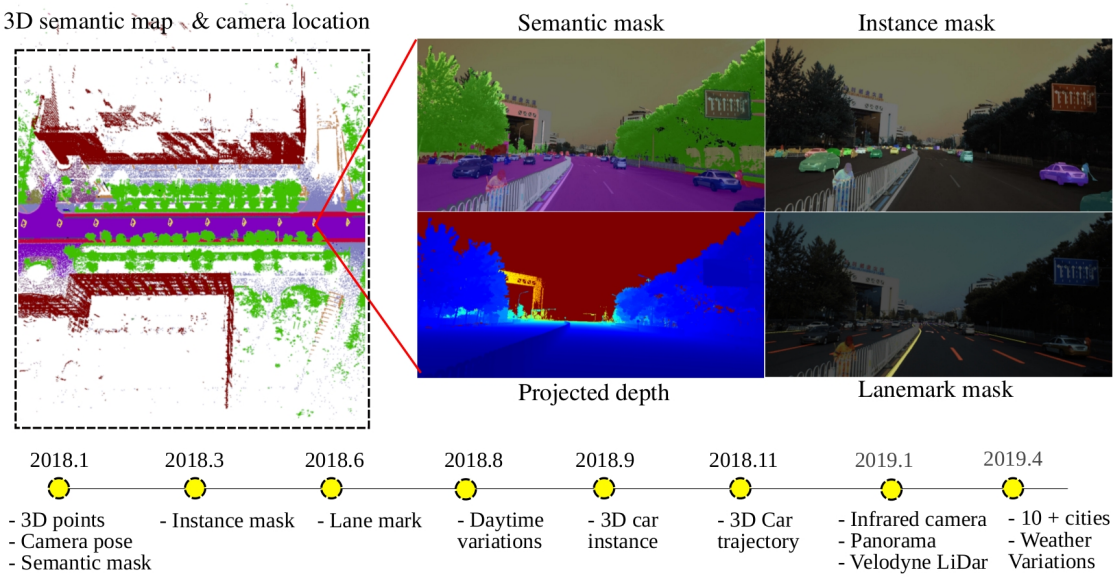


Figure 2.5: Roadmap and examples from ApolloScape dataset as presented in Huang et al. [2018] ©2018 IEEE

into the same space or plane. If we define forward as the direction of the vehicle, the coordinate systems used for the sensor are:

- Camera: **x**: right, **y**: down, **z**: forward
- IMU: **x**: forward, **y**: left, **z**: up
- LIDAR: **x**: forward, **y**: left, **z**: up

The rigid body transform from coordinate system a to coordinate system b is represented using the matrix T_a^b. Thus, any point **x** in the LIDAR coordinate frame can be projected to the camera space using Equation (2.1). In this case, this will result in the projection $y = \mathbf{P}T_{lidar}^{cam}x$. The matrix T_{lidar}^{cam} can be written as

$$T_{lidar}^{cam} = \begin{bmatrix} R_{lidar}^{cam} & t_{lidar}^{cam} \\ 0 & 1 \end{bmatrix} \tag{2.3}$$

Here, R_{lidar}^{cam} and t_{lidar}^{cam} are the rotational matrix and translation vector from LIDAR to camera space, respectively. Since the IMU and LIDAR share the same coordinate system, the transform between the two spaces will only be a translation.

2.4 Software

This section covers a large part of the frameworks and tools that are necessary components of the autonomous driving research within NAP. First, a preliminary study of simulators for autonomous driving research is conducted. This study assesses the desirability of each simulator with regards to the criterias needed for our research. Since a long-term goal of NAP is to generate its own datasets, we also investigate which annotation tools are suited for this purpose and recommend an annotation tool for future use. Lastly, the reader is briefly introduced to a crucial part of the software in many autonomous vehicle systems and robotics, namely ROS.

2.4.1 Simulators for autonomous driving

Simulators have taken on an increasingly important role in autonomous driving in the last few years. Previously simulators were mainly used for validating existing models, ensuring that the models would still perform at a satisfactory level even in difficult conditions such as at night or in demanding intersections. However, with the increase in demand of labeled data as the complexity of models rise, simulators can offer vast amounts of training data at a very low cost. This means that models can be trained on a combination of generated data and real-life data. This process reduces the amount of real-life data required substantially, and therefore decreases the cost of training the models.

When evaluating the quality of a simulator for generating a dataset, there are a number of criteria to consider. For the authors use case, the criteria below were used to assess each simulator, but should not be seen as an exhaustive list.

- **How fast is the simulator able to run?**: This partially determines how fast we can gather samples
- **Which sensor array does the simulator support?**: The vehicle in the simulator should reflect the real-world vehicle as closely as possible. Therefore, sensors that are often included in autonomous vehicles such as cameras, LIDAR, radar, IMU and GPS should be supported in the simulator.
- **How configurable are the sensors in the simulator?**: In order to mimic the sensor-setup of an actual autonomous vehicle it is important that the sensors are configurable, both with regards to its internal and external parameters. For internal parameters, cameras need to have a configurable FOV and resolution, while a LIDAR sensor might have a configurable rotational frequency, number of channels or number of points. External parameters are related to the location and rotation of the sensor relative to the vehicle.
- **How photorealistic is the simulator?**: While it is difficult to give a strict definition of photorealism in simulators, one often requires illumination effects to closely reflect the real world. These effects include how light and reflection behave under numerous weather conditions including haze, snow, fog and rain. In addition, it encompasses all notions about the photorealism of textures and geometry, and how these are affected by illumination and other rendering effects.
- **Does the simulator support a multi-agent environment?**: The simulator should allow for a configurable number of other agents in the environment. It is rarely useful to learn an autonomous vehicle, and especially perception modules, in an environment without other agents such as pedestrians and vehicles.
- **Does the simulator support a variety of driving scenarios?**: The simulator should reflect the real world as closely as possible, and this entails supporting a number of driving scenarios, both with regards to variability in the available maps and weather conditions.
- **Does the simulator offer Application Programming Interfaces (APIs) for interacting with the agents and or environment?**: In order to relate sensor information to the agents in scene (for example from a camera image), the agent's location, direction and extent is often needed.
- **How realistic is the behaviour of other agents in the simulator?**: The simulator needs to accurately reflect behavior of other agents in the scene, for instance with regards to traffic rules and interaction between vehicles and/or pedestrians.

Based on these criteria, a short evaluation of some of the most popular simulators for autonomous vehicles were performed. The discussed simulators are CARLA, NVIDIA Drive Constellation, Baidu/Apollo and AirSim.

CARLA

The CARLA simulator [Dosovitskiy et al., 2017] is an open-source driving simulator originally created by Intel Labs in cooperation with the Computer Vision Center at the Universitat Autonoma de Barcelona. The simulator, based on Unreal Engine, is intended for research within autonomous driving and includes a variety of weather conditions, vehicle models, and sensors. In 2018, the CARLA team released CARLA version 0.9, which is an entire remake over the previous 0.8.X releases. Currently, 0.9.X is under active development, and is thus missing several features from 0.8.X, such as easy transformations between sensor and world coordinate systems. One weakness with CARLA is the challenge of adding new maps. In order to do this, one has to do a lot of configuration, including pathing of agents. By default, CARLA 0.8 includes two maps[3] - *Town01* and *Town02*. Both of these are urban environments that includes building of different heights, in addition to trees and grassland. However, neither of these maps includes any road inclination, which means that the car will always be approximately level, except when braking, accelerating or making sharp turns. CARLA supports many different weather scenarios including sunny, raining and cloudy, as shown in Figure 2.7.

At the time of writing, CARLA supports four sensor types:

1. Scene Final
2. Depth Map
3. Semantic Segmentation
4. Ray-cast LIDAR

The first three are camera-based sensors, where *Scene Final* is the camera image or rendered screen, but with applied post-processing. This post-processing is intended for making the camera image more realistic than the rendering and currently includes vignette, grain jitter, bloom, auto-exposure, lens flares and Depth of Field (DoF) which blurs objects very close or very distant from the camera. It is not possible to modify these post-processing effects in the simulator itself, as they are a part of the Unreal Engine. When these post-processing effects are applied, the results are images that look quite photorealistic (Figure 2.6). However, the grain jitter effect, which adds noise to the rendered image tends to make the image more noisy than one would often like. Usually these effects are image augmentation steps taken by a machine learning model, and one would prefer to include these augmentations in the actual model instead of in the data generation tool.

The ray-cast LIDAR sensor is usually the preferred way to generate point clouds in CARLA, as it supports the commonly used **Stanford Triangle Format**, also known as **PLY**. However, the ray-cast LIDAR currently supported in CARLA gives a less than ideal representation of agents in the scene, as every agent of the same class has the same LIDAR representation, with only minor differences between certain classes of vehicles such as sedans vs. minivans. This means that if we have two vehicles in the scene - one tall and wide and one slightly shorter and lower, both will have the same surface in the LIDAR view. This is because all models of the same type, e.g. vehicles or pedestrians have the same underlying mesh which is used for generating point clouds. One way to circumvent this is to use the *Camera depth map*, which uses the actual rendered depth of each model, and then sample points in the generated depth map to simulate a LIDAR sensor. However, this is not implemented as of yet in CARLA.

NVIDIA DRIVE Constellation

The NVIDIA Drive Constellation simulator is a new driving simulator currently only available for automakers, startups and selected research institutions. The simulator is built for Hardware In the Loop

[3]Later versions of CARLA include additional maps, numbering a total of seven in CARLA 0.9.5

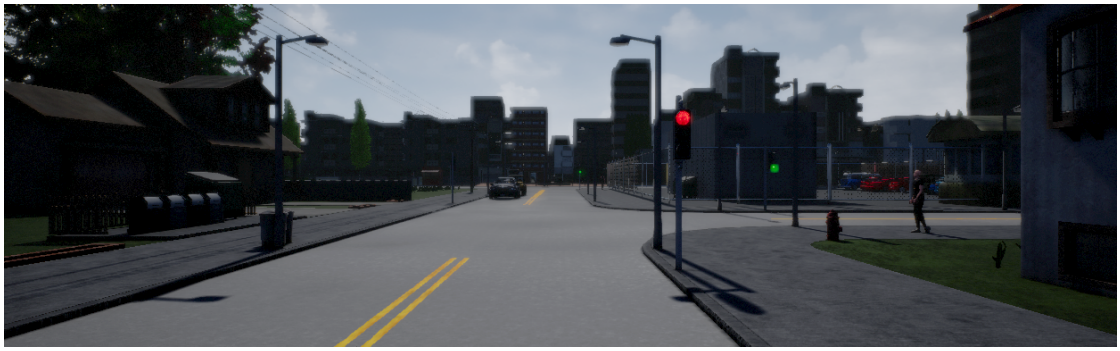


Figure 2.6: Sample image rendered by the CARLA simulator with added post-processing effects

(HIL) simulation, which creates a feedback loop between the simulator and NVIDIAs DRIVE AGX Pegasus AI car computer, allowing the computing device to be trained on data very similar to a real-world situation. The simulator contains a variety of sensors, including camera, LIDAR and RADAR, and introduces very photorealistic simulation when compared to for example CARLA. In addition, the simulator includes support for creating simulated environments from HD maps, thus allowing models to be trained on a variety of driving scenarios. Although NVIDIA Drive Constellation is currently under a closed release, it is expected that more research institutions will get access to the simulator in the future.

Baidu/Apollo

The Baidu Apollo Simulator is a simulator offered in the Apollo autonomous vehicle platform. The simulator offers two solutions, the first is based on playback of ROS-bags on an open-source platform, while the second offers simulation in a closed-source cloud platform. However, the platform functions differently from most simulators, in that it is not an open world. Instead, the simulator consists of several pre-defined scenarios which one can train models on. An example scenario is shown in Figure 2.8. In addition, the Apollo Open Platform Data offers labeled data such as multi-sensor fusion data and trajectory prediction. However, these models can not be tested in the simulator, as the simulator only supports LIDAR and RADAR. Another important aspect of the Apollo Simulator is that it does not allow modifications to the underlying simulator. Therefore, it is difficult to extract ground-truth information such as extents of vehicles, and their position in the world. However, these aspects are already covered in the pre-existing scenarios. One interesting aspect about the Baidu/Apollo simulator is its comprehension evaluation suite. The evaluation gives a thorough measure of driving ability using criteria such as speed limit adherence, object detection and collision detection. An advantage with using the Apollo Simulator is that is part of the Apollo ecosystem which offers a variety of tools for autonomous driving, including a reference drive-by-wire vehicle, hardware platform and open-source software platform. This allows developers to easily set up a fully operative autonomous vehicle without creating all necessary modules themselves. However, the lack of important sensors such as cameras make the Apollo simulator less desirable.

AirSim

AirSim [Shah et al., 2018] is a simulator for drones and vehicles, developed by Microsoft based on the Unreal Engine with an experimental version based on Unity. It allows for photorealistic simulation in a multi-agent environment, with many available assets and maps. The simulator is open-source, and can be extended to export labels and bounding boxes. However, this is not as straight forward as in CARLA,

(a) Sunny

(b) Cloudy

(c) Drizzly

(d) Downpour

Figure 2.7: Weather types in the CARLA simulator.

where vehicle locations and extents are supported out of the box. AirSim supports a variety of sensors out of the box, including Barometer, Magnetometer, Camera, GPS, Distance, LIDAR and IMU. Some of these sensors are mainly included for multicopters, and has little to no use for most machine learning models within autonomous driving. The AirSim simulator contains relatively photorealistic simulation and offers an APIs in a variety of languages, including C++, C#, Python and Java. A sample image from the simulator is shown in Figure 2.9.

	Open Source	Varied weather	Varied maps	Photorealistic	Languages
Baidu Apollo	✓[4]	✓	✓	Medium	Unknown
NVIDIA Drive	×	✓	✓	High	Unknown
AirSim	✓	✓	✓	Medium	Python, C++, C#, Java
CARLA	✓	✓	✓	Medium	Python, C++

Table 2.1: Comparison of simulators for autonomous driving.

2.4.2 Annotation tools

In order to create a object detection model with satisfactory performance, one needs a sufficiently large, varied dataset. If using real-world data, obtaining a large dataset can be very expensive. The two most common methods for labeling data is either by hand, or to use a pre-trained model to create ground

[4]Only for ROS-playback

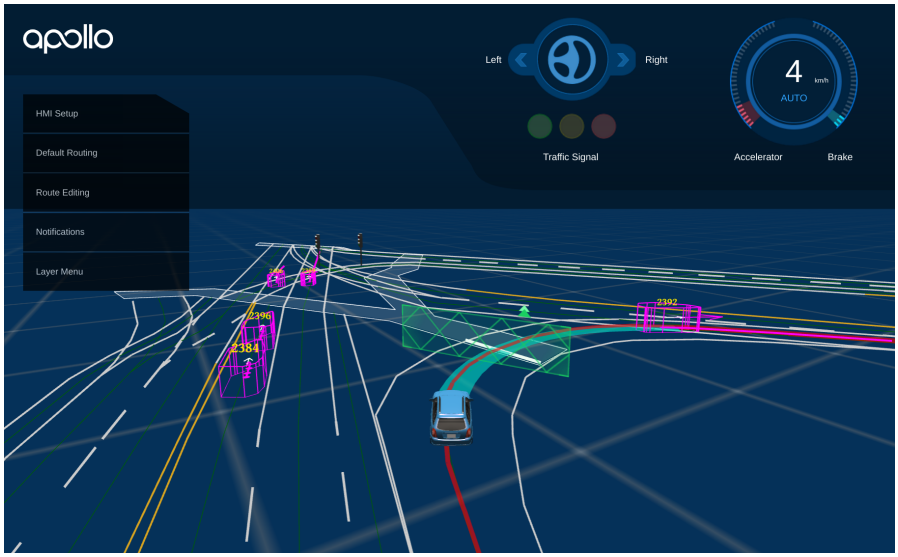


Figure 2.8: A screengrab of the Apollo Simulator in a scenario making a right turn in an intersection. Reprinted from the Apollo Simulator Github Repository [Baidu Apollo , 2019], available under the Apache-2.0 license.

truths. The former is very expensive, requiring many hours of human labour, while the latter may yield inaccurate bounding boxes and needs to be supervised and corrected.

There exists a plethora of annotation tools for object detection, ranging from simple 2D bounding box labeling tools, to proprietary collaborative tools with options for 2D/3D bounding boxes, instance segmentation and more. An important facet to consider is which criteria to evaluate the annotation tools on. Because of limited time to evaluate the tools with regards to usability, the main focus of this evaluation will be on available functionality in the tool, as well as whether or not the tool is proprietary or open-source. The reasoning for including open-source in the evaluation is that it allows users to include new functionality into the existing tool, such as exporting labels to a different format or including new sensor data. The results of this evaluation is shown in Table 2.2. The evaluation shows that the best annotation tools that covers most use-cases in autonomous driving is *Scalabel.ai*, which is a tool developed by Berkeley Deep Drive (BDD) - the same group that gathered the BDD100K dataset [Yu et al., 2018]. It should be noted that although the annotation tool supports all functionality listed in the table, the 3D bounding box labeling is under active development.

2.4.3 ROS

The Robot Operating System (ROS) is a set of open-source frameworks for software development in robotics, and is often described as a robotics framework. ROS offers many helpful abstractions for writing software for robotics, inluding abstractions for hardware, messaging and interoperability.

On a high level, ROS defines a communication peer-to-peer network that consists of one master node and many other nodes. The master node exists to enable communication between other nodes, as it contains information about the whole network. Each node is responsible for a particular function. For example, a LIDAR sensor can be a node. A node can send information to other nodes in the network via messages, which has a predefined structure. The node sends information by publishing the message onto a topic. Other nodes that are subscribed to the given topic receives a callback when there is data

Figure 2.9: A screengrab of the AirSim simulator using the precompiled City environment.

	2D bbox	3D bbox	Segmentation	Video Tracking	Open-source
LabelMe	✓				
Dataturks	✓				
LabelBox	✓				
YOLO_mark	✓				
RectLabel	✓		✓		
Imglab	✓				✓
LabelImg	✓				✓
ALT	✓				✓
Scale.ai	✓	✓	✓	✓	
Scalabel.ai	✓	✓[5]	✓	✓	✓

Table 2.2: Comparison of annotation tools for autonomous driving.

published to the topic. If there are no subscribers on a given topic, there is no actual data being transported over the network. In addition, every subscriber on the topic will be notified at the same time, since there exists separate connections for each subscriber. This is very similar to the Observer software design pattern, where all listeners of a given object is alerted and updated when the object changes its state [Gamma, 1995]. In order to be able to evaluate a ROS-system without working directly with real-time data from sensors, ROS allows storing data in ROS-bags. This allows developers to 'play back' the previously recorded communication, thereby emulating real-time behavior. This is very helpful in autonomous driving scenarios where one wants to test models or algorithms without actually having to drive the vehicle.

Although ROS works out of the box with multiple languages, including C++, Lisp and Python, it currently only supports Python 2.7. However, its successor ROS2 supports Python 3.X, but is under active development and lacks some features at the time of writing.

[5] Only for LIDAR pointclouds - in beta.

2.5 Hardware

2.5.1 Hesai Pandora sensor

The Pandora sensor is a sensor suite created by Hesai Technologies for autonomous driving applications. The Pandora contains a LIDAR sensor, one color camera and four mono cameras. The specifications of each sensor is shown in Table 2.3. Since all sensors are mounted statically in the Pandora, one does not need to perform calibration between the sensors, as it is already done by the manufacturers. In addition the Pandora offers seamless integration with the Apollo platform, making it essentially plug and play - especially if one uses a reference vehicle supported in Apollo. The Pandora sensor suite also offers pre-built ROS nodes that offers synchronization between the different sensor modules.

Color Camera	
Resolution	1280×720
Vertical FOV	$28.6°$
Horizontal FOV	$52°$
Effective Focal Lens (EFL)	5.47mm
Mono camera	
Resolution	1280×720
Vertical FOV	$81.8°$
Horizontal FOV	$129°$
EFL	1.65mm
LIDAR	
Number of channels	40
Points per second	720 000
Vertical FOV	$[-16°, 7°]$
Range	$[0.3m, 200m]$

Table 2.3: Pandora sensor specifications

In this project we utilize the Pandora sensor suite to gather real data, and autonomous vehicle simulators for collecting simulated data. Although these types of data are very closely related, they may be considered as separate domains, since the representations of objects and scenes may be quite different. Therefore, the next section introduces the concept of transfer learning, and how it relates to the gap between domains – in our case real and simulated data.

2.6 Transfer learning

Transfer learning is a technique commonly applied in training machine learning models, using a model trained on one data distribution as a starting point for training on another distribution. These distributions are often similar, but comes from different domains. A more rigorous definition is presented in Pan and Yang [2009], where transfer learning is defined as improving a target function $f_T(\cdot)$ on a task T_T in the domain D_T by utilizing information in a different domain D_S with task T_S. The term transfer learning is closely related to domain adaptation, and there is not yet a rigid definition in the literature [Li, 2012]. Broadly, one can think of transfer learning as transferring the knowledge learned by a model trained in one domain, onto a new model in a different domain, as illustrated in Figure 2.10. Transfer learning has become standard practice in Convolutional Neural Networks (CNNs), where models trained on large datasets such as ImageNet [Deng et al., 2009] are used as a starting point for a variety of tasks. For instance, the task of detecting cats may be improved by utilizing the knowledge of a model trained on

detecting dogs. In object detection networks, transfer learning is commonly done by using an existing convolutional neural network as a backbone, and then adding different output layers based on the specific task. The weights of the backbone network can either be frozen, so that just the head layers are trained, or be trained together with the initialized heads. This method works very well for CNNs since the weights learned early in the architecture are general features such as edges and blobs.

The method of transfer learning can be divided into several tasks. Usually, the procedure of training the model on the first dataset or domain D_T is called pre-training, while training the learned model on the target domain D_S is referred to as fine-tuning.

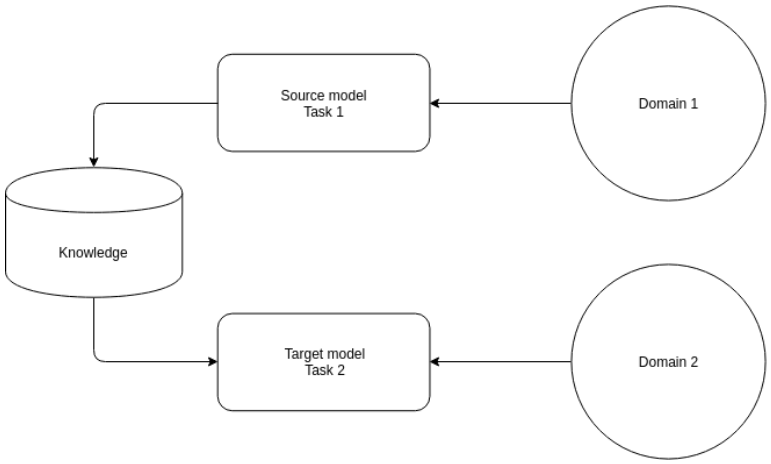


Figure 2.10: Simplified representation of transfer learning. A source model trained on Task 1 is utilized by the target model to improve performance on Task 2.

2.7 Deep Learning

Deep learning is a term commonly used in machine learning when referring to learning systems using neural networks with many computational layers. According to the universal approximation theorem, a neural network with one hidden layer and a finite number of neurons can theoretically approximate any continuous bounded function[6], practice shows that learning can be achieved faster by increasing the number of layers given the same number of parameters. The theoretical explanation behind this phenomenon is that the number of linear regions, which can be interpreted as the decision boundaries of the model, grows polynomially with the number of parameters in a network with one hidden layer, and exponentially in a deep network [Montufar et al., 2014]. The number of linear regions is thus a measure of how well the model is able to learn to separate features in the input space, and therefore a measure of how well the model can approximate a function. The use of deep neural networks has brought incredible results in the last decade, but has also introduces some problems such as vanishing gradients and longer training times due to the increase in the number of parameters. In perception, deep convolutional neural networks have gained widespread use for feature extraction. Compared to previous manual methods of feature extraction such as Histogram of Oriented Gradients (HOG) [Dalal and Triggs, 2005], convolutional neural networks have shown great improvements, but require large amounts of training data.

[6]Given that some assumptions about its activation functions are made. See Hornik [1991] for a detailed description.

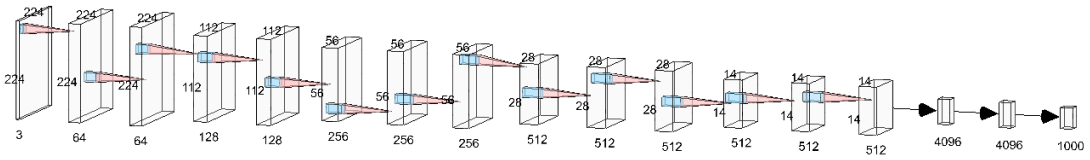


Figure 2.11: VGG-16 architecture. Activation functions and pooling layers omitted for brevity. Created with `http://alexlenail.me/NN-SVG`

2.7.1 VGG

VGG was introduced by Simonyan and Zisserman [2014] for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where it reached first and second place in the localization and classification tasks respectively. The authors demonstrated that deep convolutional networks could be vastly improved by introducing more weight layers, increasing the depth of the networks to around 16-19 layers. This was possible by using 3×3 filters with a stride of one, which is the smallest filters capable of capturing the directions up, down, left and right. By stacking three 3×3 filters on top of eachother, the network is able to capture more non-linear features, while having the same receptive field as a single 7×7 convolutional layer and fewer parameters. In order to perform localization and classification, 3 fully connected layers are added after the convolutional layers. The two first dense layers contains 4096 neurons each, while the output layer contains 1,000 - one for each class in the ImageNet dataset. The activation function on all hidden layers is the Rectified Linear Unit (ReLU), while the output layer contains a softmax nonlinearity as to transform the output to a probability distribution of all classes. A simplified version of the VGG-16 network is shown in Figure 2.11. The VGG networks has shown to generalize well when trained on a number of different datasets including ImageNet, PASCAL VOC and COCO, and achieved over 89% mAP on VOC-2007 and VOC-2012 [Simonyan and Zisserman, 2014].

2.7.2 ResNet

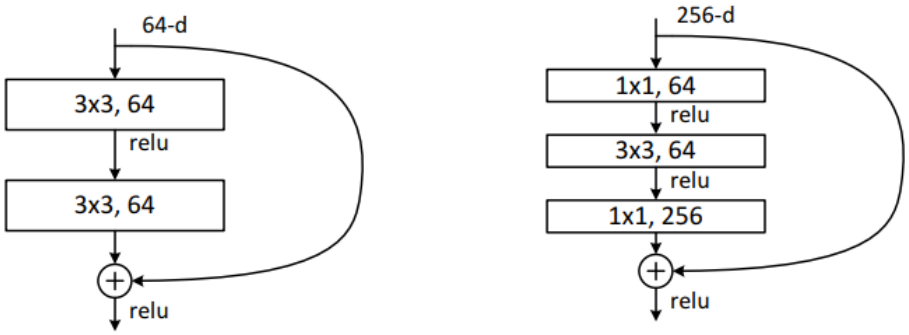
As mentioned previously, the universal approximation theorem states that a neural network with a single hidden layer can approximate any continuous bounded function. However, the modern approach is to include several computational layers, which can reduce the number of parameters and learn faster. A common problem with this approach is that the gradient, which is computed by the chain rule, will be a product of many very small numbers. Therefore, the weight updates tend to approach zero, and the network will either train very slow or not converge at all. A method of combatting vanishing gradients was introduced by He et al. [2015] through residual networks (ResNets). In regular[7] neural networks, the activation of a layer l is calculated as in Equation (2.4) , where g is the activation function, w is the weight matrix, a^{l-1} is the activation of the previous layer and b is the bias vector. In residual neural networks, the activation of a layer is computed similarly, but skip connections between layers are introduced. A skip connection between a layer $l-2$ and l, transfers the activation of layer $l-2$ to the input of layer l. Thus, the activation of layer l can be computed as $a^l = g(z^l + a^{l-2})$. The introduction of skip connections ensures that the gradient is preserved throughout the network, which allows for networks to increase the number of layers by tens or hundreds of layers without facing issues with vanishing gradients.

$$a^l = g(w^l a^{l-1} + b^l) = g(z^l) \tag{2.4}$$

The residual networks introduced in He et al. [2015] are built using stacks of building blocks, which

[7] Neural networks without residual connections are also often referred to as *plain* networks.

are shown in Figure 2.12. The most shallow ResNet with 34 layers is built using 2.12a, while ResNet 50/101/152 are built with 2.12b. The latter is referred to as a bottleneck building block, which reduces the number of trainable weights and allows the network to converge faster. Using these building blocks, He et al. [2015] used a 152 layer ResNet to outperform VGG with regards to both time complexity and performance. In addition, the authors used an ensemble of ResNets to achieve 3.56% error rate on the ImageNet test set, resulting in first place in the ILSVRC 2015 classification competition. Because of their relatively low complexity and stability, ResNets are often used in object detection architectures as a backbone or feature extractor[8].



(a) Building block used in ResNet-34

(b) Building block used in ResNet 50/101/152

Figure 2.12: ResNet building blocks (from He et al. [2015] ©2015 IEEE)

2.8 Evaluation metrics for object detection

There are many ways of measuring the performance of machine learning models for classification, segmentation or object detection. However, most of these methods utilize the same underlying measurements, namely precision and recall. This section introduces the basic concepts of precision, recall and Intersection over Union (IoU), in order to discuss a common metric in object detection – mAP.

2.8.1 Precision and Recall

Precision is a measurement of the number of how many of the positive samples that are classified correctly. This can be written as the fraction of True Positives (TP) over the total number of identified postive samples, which are the True Positives and False Positives (FP), shown in Equation (2.5). Precision does not take into account False Negatives (FN), which means that any object that was not detected by the model does not affect the precision score.

Recall, on the other hand, is fraction of true positive samples that the model detected. Comparing the equation of Precision (2.5) and Recall (2.6), we observe that the only difference is the denominator. Recall is agnostic to False Positives, so any falsely detected object will not affect recall.

Consider a search engine that delivers relevant and irrelevant items based on a search query. Let us refer to this reply of items as the result. Then, the recall would be the fraction of relevant items in the

[8]In machine learning terminology, a neural network is often thought of as to have a backbone and one or multiple *heads*. The heads are the the output layers of the network, and are often initialized randomly, while the backbone may be pre-trained on another dataset.

result (as opposed to not in the result), while the precision would be the fraction of items in the result that are relevant.

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

2.8.2 Intersection over Union

Intersection over Union (IoU) is a metric that indicates the amount of overlap between two shapes. The metric is often applied when performing Non-Maximum Suppression (NMS), which in object detection is a method for pruning multiple detections of the same object. Given two shapes A and B, the IoU is calculated as in Equation (2.7). In the KITTI evaluation benchmark, the IoU threshold for detection is set as 0.7 for Cars, and 0.5 for other classes. IoU is often colloquially referred to as overlap.

$$IoU(A, B) = \frac{A \cap B}{A \cup B} \tag{2.7}$$

2.8.3 mean Average Precision

mean Average Precision (mAP) is a common indicator of a machine learning models performance on object detection tasks. The metric is calculated after NMS is applied, and measures an algorithms precision at different recall values. The algorithm is evaluated at different values of recall since a system that has high precision may correctly detect all identified objects, but does not identify every object in the scene. For instance, an object detection network may detect all front-facing pedestrians in the scene (high precision) but fails to detect every other pedestrian (low recall).

Calculating mAP for 2D object detection is performed by discretizing the recall values in the range $[0, 1]$ into N bins and calculating the precision corresponding precision of each recall value r. The mean average precision is calculated as the average of all the precision values at the different recall steps. In the case of 3D object detection, the orientation of the bounding box is also included in the evaluation metric. The metric is then referred to as Average Orientation Similarity (AOS), which is shown in Equation (2.8). Here, $D(r)$ is the set of all detections for an object with recall r, B is the set of discretized recall values, $\Delta_\theta^{(i)}$ is the difference in predicted and ground truth orientation, and δ_i is 1 if the current object has been assigned a detection, else 0. The criteria for a ground truth annotation to be assigned a detection is that the detection has an IoU over 0.5, or 0.7 in the case of Cars. The exact implementation of AOS in KITTI can be found in Geiger et al. [2012].

$$\begin{aligned} AOS &= \frac{1}{|B|} \sum_{r \in B} s(r) \\ s(r) &= \frac{1}{|D(r)|} \sum_{i \in D(r)} \frac{1 + cos(\Delta_\theta^{(i)})}{2} \delta_i \end{aligned} \tag{2.8}$$

Chapter 3

Related Work

There are several different techniques for applying object detection networks to sensor data in autonomous driving. While companies such as Waymo and Uber use LIDAR and cameras, other companies such as Nissan and Tesla are moving away from LIDAR due to its price and limited capabilities[Tajitsu, 2019]. However, for detecting 3D bounding boxes LIDAR appears as a valuable sensor because of its accurate depth measurement capabilities. All models discussed in this chapter are therefore either solely based on LIDAR or are using both camera and LIDAR data. They are categorized as 2D proposal architectures, LIDAR-based architectures, and Sensor fusion architectures. Among these, the focus for experimentation will be on the sensor fusion architecture *AVOD-FPN*, with additional experiments performed using the LIDAR based *SECOND* architecture.

3.1 From 2D to 3D object detection

There exists several different techniques for performing 3D object detection. Many of these techniques are inspired by their 2D counterparts, and are quite similar with regards to architecture and design. Architectures such as YOLO [Redmon and Farhadi, 2018], Mask R-CNN [He et al., 2017] and NAS-FPN Ghiasi et al. [2019] have given ideas to several of the networks presented in this chapter, but only utilize image data and are therefore not covered here.

One method for performing 3D object detection is to use a 2D object detection model to find objects in the scene, and then extrude each bounding box by using a different model to predict the depth and orientation of the bounding box. This method requires two models, which might increase time complexity with regards to inference and training. Another method is to extend the 2D object detection model with additional regressors to encode a 3D bounding box. This would usually be done by including depth as a target variable. As cameras have weak depth information, Most architectures for 3D object detection prefer the latter strategy, and the next sections wil cover some of the state-of-the-art architectures for this task.

3.2 Sensor fusion architectures

Object detection in autonomous driving occurs in challenging environments and poses a number of challenges related to occlusion, lighting conditions, weather conditions, large variation in distances as well as the variation in object size and shape. Of the multitude of sensors used in the field, each pose with their strengths and weaknesses.

While the use of cameras is a common starting point offering largely unmatched semantic information, it suffers in poor lighting and when objects are either more distant or largely occluded from the point of view of the camera. Sensors such as radars or ultrasound sensors can offer more accurate depth information than stereoscopic cameras, the former also being superior at longer ranges, but can not match the resolution and semantic information of the data. Recently LIDAR sensors have become a popular field of study for use in autonomous vehicles due to their relatively strong matching of semantic information and accurate depth readings in the form of point clouds. LIDAR produces its own laser, being invariant to changes in lighting conditions, is more robust to complete or partial occlusion and also maintains object size as distance increases. However, point sparsity also increases with distance, and as LIDAR point clouds already are more sparse than the dense camera images, LIDAR still cannot quite match cameras on resolution or semantic information. There has been a lot of research into how to best combine multiple different sensors [Chen et al., 2017; Ku et al., 2018; Cho et al., 2014], mainly concerning the use of cameras combined with LIDAR for autonomous driving, in order to create more robust and accurate models. The direct combination of sensor data is generally referred to as sensor fusion, and is commonly categorized as either early, late or deep fusion.

Early fusion, as the name implies, consists of combining the multiple sources of sensor data before passing it through a network to extract features. This is commonly done simply by alignment, such as by transformations and projection of a LIDAR point cloud onto an image, and concatenation along a certain axis. The polar opposite of this approach is late fusion where the concatenation first happens after feature extraction, joining the separate feature maps. Recent research also shows promising results with a third approach, deep fusion, where the fusion of sensor features occurs over multiple layers of deep network architectures using learnable weights, such that the model itself learns how to make use of each sensor.

3.2.1 MV3D

Multi-View 3D [Chen et al., 2017] is a multimodal architecture for 3D object detection providing deep fusion of images and point clouds by the use of three separate views; camera view, Front View (FV) and Bird's Eye View (BEV or BV). MV3D is widely regarded as the first architecture employing the projection of a LIDAR point cloud to BEV for 3D object detection, marking a shift from the previously common approach of projecting the points directly onto the image as a front view or depth map, with markedly better results. At the time of its release, MV3D produced results 25% better for localization and 30% better for 3D AP than previous state-of-the-art, with additionally 10% better AP for 2D detection than other LIDAR based architectures on KITTI's hard test set. The overall MV3D architecture is depicted in Figure 3.1.

The 3D proposal network and feature extraction segment of the architecture has three separate branches relating to each of the aforementioned views. Each of these branches is based on modified versions of the VGG-16 architecture with channels reduced by half, bilinear upsampling before the 3D proposal generation using BEV, removal of the 4th pooling operation and addition of an extra fully connected layer. Despite the three branches, the total number of parameters is only 75% of the original VGG-16 network. The 3D proposal network is inspired by Region Proposal Networks (RPNs), as introduced in Faster-RCNN [Ren et al., 2015], taking the BEV as input to extrapolate from 2D proposals to 3D using a set of prior 3D boxes. This also simplifies the separation of occluded objects, as for example vehicles do not occupy the same space in BEV, as well as offering size invariance of objects depending on their distance.

For projection to BEV, MV3D discretises points in the LIDAR point cloud to a resolution of 0.1m and segments the point cloud into M horizontal slices. The value of each discrete cell of a slice is taken as the highest point within the boundaries of the cell. Additional grids of the same resolution are constructed for an intensity map and a density map as well, taken over the entire point cloud rather than the individual slices. The intensity values of each cell are taken from the highest point, while the density map contains

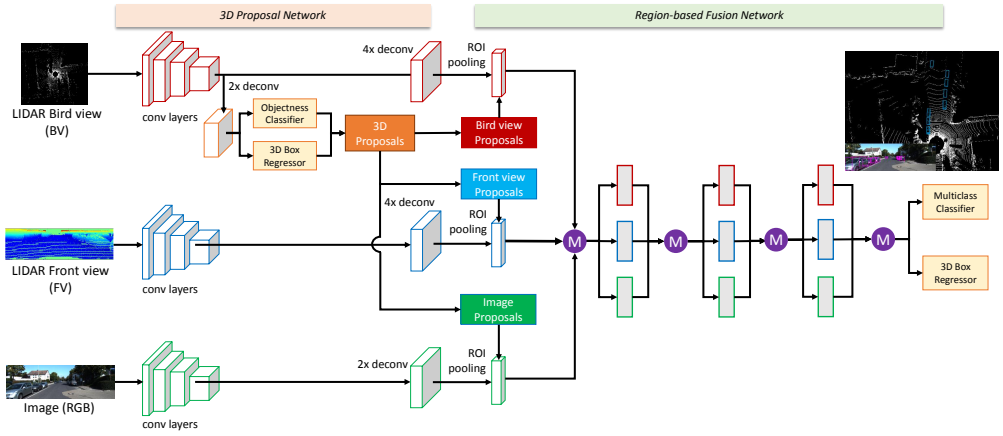


Figure 3.1: Architectural overview of the MV3D model(from Chen et al. [2017], ©2016 IEEE).

the number of points in a cell normalized by Equation (3.1).

$$min(1.0, \frac{log(N+1)}{log(64)}) \tag{3.1}$$

The Front View primarily serves as complementary information to the Bird's Eye View, providing a single height map, distance map and intensity map. Projection of the FV to a regular 2D image still provides a sparse representation of the data, so instead the FV is projected onto a cylinder plane providing a dense mapping. Points x, y, z in the point cloud are converted to cylinder coordinates r and c using $\Delta\theta$ and $\Delta\phi$ as the horizontal and vertical resolutions of the LIDAR, as in Equations (3.2) and (3.3).

$$r = \lfloor atan2(z, \sqrt{x^2 + y^2})/\Delta\phi \rfloor \tag{3.2}$$

$$c = \lfloor atan2(y, x)/\Delta\theta \rfloor \tag{3.3}$$

After the generation of feature maps and 3D proposals, the three views are fused in-between multiple layers using an element-wise mean operation in the region-based fusion network. A problem with projecting LIDAR to BEV as a discretised 2D grid and projecting the Front View as a cylinder plane with the camera image is that it produces feature maps with different resolutions. Therefore the authors employ Region of Interest (RoI) pooling before the deep fusion steps. Fusion is applied through trainable weights between each of a series of layers, with an additional probability of only selecting a single view or dropping a single view from the fusion step entirely during training. This is fairly similar to dropout for conventional regularization. To aid the learning process during this type of drop-path training, additional auxiliary paths are used when training, but removed before inference. These auxiliary paths share weights with the fusion layers, but backpropagate the gradients through separate losses for each view.

The output of MV3D consists of object class and 3D coordinates for each of the 8 box corners, illustrated in Figure 3.3. This 24D bounding box representation might seem superfluous but was reported to obtain better results than conventional height, size, width, and center while also being oriented.

3.2.2 AVOD-FPN

The Aggregate View Object Detection Feature Pyramid Network [Ku et al., 2018], or AVOD-FPN, is a joint architecture for object classification and 3D bounding box regression using RGB images and LIDAR

point clouds building upon the methods used in MV3D. The AVOD architecture consists primarily of two subnetworks; a region proposal network (RPN) and a detector network. AVOD-FPN uses a Feature Pyramid Network (FPN) in the feature extraction phase to produce high resolution upscaled feature maps for 3D object proposals from LIDAR point clouds in BEV, as well as the front-facing camera images. As with MV3D this produces accurate 3D bounding boxes with explicit orientation and provides state-of-the-art results for both classification and localization of cars, pedestrians and cyclists on the KITTI leaderboards. Novel LIDAR based methods have in recent months superseded AVOD-FPN in real-time performance, especially on the detection of cars, but AVOD-FPN still boasts competetive performance for small classes such as pedestrians due its multimodal approach making good use of high resolution camera images. A graphical overview of the architecture can be seen in Figure 3.2.

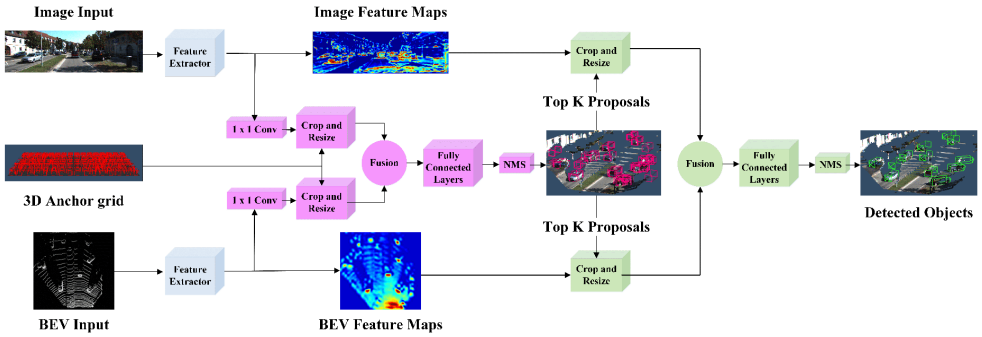


Figure 3.2: Architectural overview of the AVOD-FPN model (from Ku et al. [2018] ©2017 IEEE).

As in MV3D, AVOD-FPN uses multiple slices of max height values as well as a single density map, normalized as in Equation 3.1 but by a factor of $log(16)$ rather than $log(64)$, projected to BEV after being discretized to a resolution of 0.1m. However, AVOD-FPN manages to discard both the intensity map in BEV, as well as all the Front View maps, greatly reducing complexity on the LIDAR side. For both image and LIDAR data, AVOD-FPN uses two essentially identical VGG-16 based feature extractors, reducing channels by half from the original architecture and cutting it at the conv-4 layer. Taking in maps of size $M \times N \times D$, this produces resulting feature maps that are of size $\frac{M}{8} \times \frac{N}{8} \times D^*$ where $D*$ is the number of BEV images. These maps are of high representational power but can result in sub-pixel representation of small classes such as pedestrians in BEV. Additionally, RPNs are typically designed for dense high-resolution images, whereas the representation of LIDAR point cloud is sparse and fairly low resolution. Taking inspiration from Feature Pyramid Networks the authors therefore introduced a bottom-up decoder that learns to upsample the feature map back to its original size while still maintaining high run-time speeds. As a result this produces a feature map of high representational power and resolution, essential for detection from BEV.

AVOD-FPN also alters the targets for the bounding box regression by use of relative distance to the ground plane, which can either be estimated at run-time or by preprocessing. However, the method used for estimating these ground planes on the KITTI dataset are not included with the source code or documentation. Bounding boxes are represented by the 4 lowest corners in 2D coordinates as well as height from the ground plane and height of the box, as visualized on the right hand side of Figure 3.3. This representation is more compact than in MV3D, reduced from 24D to 10D, and the authors argue that this maintains more of the physical restrictions the bounding boxes must conform to. Box priors are determined for each class by use of K-Means clustering on all ground truth objects in the dataset, with two clusters used for cars, one for pedestrians and one for cyclists.

During training AVOD-FPN also utilizes drop-paths, but only considers one path for images and one for LIDAR data due to no FV on the LIDAR side. In addition, Auxilliary loss is removed. Multi-task loss

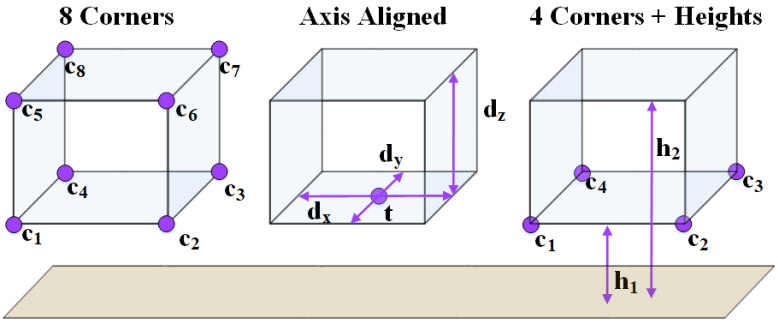


Figure 3.3: Visualization of different box representations used for bounding box regression. The far left is used for MV3D while the far right is used for AVOD-FPN (from Ku et al. [2018] ©2017 IEEE).

is employed using Smooth L1 for bounding box and orientation vector regression, while softmax cross-entropy is used for classification. For use on the KITTI dataset two models are trained; one for cars and one for combined detection of pedestrians and cyclists. Both models train for 120k iterations, amounting to about 16 hours on a NVIDIA TITAN Xp. Each of the models are able to perform inference at a rate of 10 times per second with memory consumption as low as 2GB. At the time of its submission, AVOD-FPN outperformed other state-of-the-art architectures in almost all measures for 3D object detection on the KITTI dataset.

3.3 2D proposal architectures

Another approach to multimodal 3D object detection using camera and LIDAR is to use camera-based 2D object detection in order to generate proposals and filter the point cloud based on these proposals. 2D object detection is a more mature field, potentially determining classes more accurately and more efficiently segmenting the visible area into regions of interest. Recent methods have surfaced that are able to treat raw point clouds directly, without projection and discretization to an image. Using 2D object detection methods to indicate regions of interest, these methods can potentially generate better results, especially for small classes.

3.3.1 PointNet

The aforementioned architectures treat point clouds as images from multiple viewing angles such as BEV or FV through the use of conventional CNNss The recently proposed PointNet [Qi et al., 2017] architecture shows state-of-the-art results on classification and semantic segmentation tasks by direct processing of the unstructured points of a point cloud. The authors argue that their novel approach does not feature issues such as unnecessary voluminous representation in 3D voxel grids or multiple images, while exploiting the geometric interaction between points. Although not a object detection architecture in itself, PointNet serves as a basis for several novel architectures for 3D object detection, warranting some investigation into how it works.

To perform well when treating a point cloud directly the model must be invariant to permutations of the points, consider neighbouring points and be invariant to rigid transformations such as rotation and translation of all object points. The authors argue that permutations can either be handled by sorting, sequence augmentation using a Recurrent Neural Network (RNN) or by a symmetric function aggregating information to form an order invariant feature vector. The first method is infeasible as no stable method exists bijectionally mapping a high dimensional space to a 1D line. The proposed solution is the latter,

as Qi et al. [2017] show that while RNNs are robust to permutation changes they cannot be omitted, especially for very long sequences. PointNet approximates a general function defined on the set of points by applying a symmetric function on the transformed points as in Equation (3.4). The approximation of h is performed by a Multilayer Perceptron (MLP) while g is approximated by a composition of a single variable function and max-pooling.

$$f(x_1, ..., x_n) \approx g(h(x_1), ..., h(x_n)) \tag{3.4}$$

The output vector generated holds a set of global features that is sufficient for classification using a Support Vector Machine (SVM) or a MLP, but does not provide local information required for segmentation. Towards this end the authors concatenate the global features with local point features, considering both simultaneously. This allows the network to consider geometric relations between points to perform tasks such as generation of per-point normals, shape part segmentation or scene segmentation.

In order to be invariant to rigid transformations the authors introduce a small alignment network, called a T-Net, whose task is to predict an affine transformation that aligns the input points in a canonical frame. This transformation then directly applies to all points. Similarly, such a network can be used for alignment in feature space as well, but due to increased dimensionality the authors include an additional restriction. The affine transformation is restricted, using regularization, to be near orthogonal, as not to lose information from the input. The full architecture can be seen in Figure 3.4.

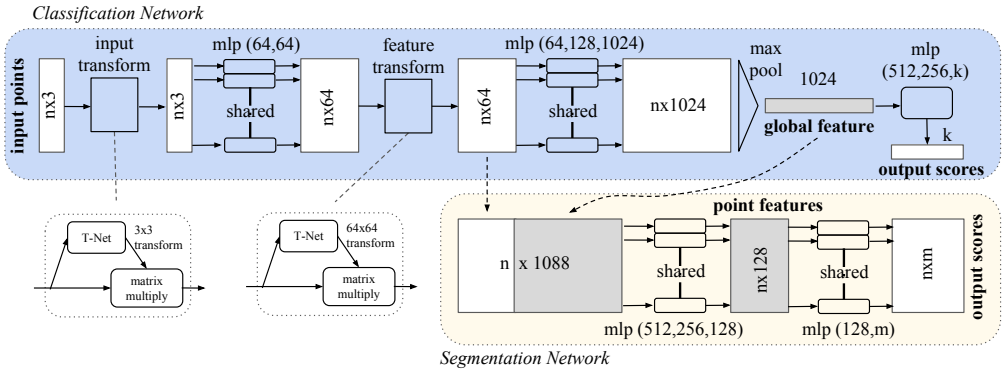


Figure 3.4: Architectural overview of the PointNet model (from Qi et al. [2017] ©2016 IEEE).

The PointNet architecture learns to summarize a shape by a sparse subset of points representing its skeleton. The computation is easily parallellized and scales linearly with the number of points, meaning that PointNet can efficiently process large point clouds, while being robust to noise and missing data. The architecture provides state-of-the-art results for part segmentation on the ShapeNet parts dataset as well as for several categories of furniture on the Stanford 3D semantic parsing dataset. The inclusion of PointNet-based arhcitectures in 3D object detectors has resulted in several new architectures competing for state-of-the-art in the field.

3.3.2 Frustum PointNet

Frustum PointNet [Qi et al., 2018], or F-PointNet, is a 3D object detection architecture from the creators of PointNet with solid performance in both outdoors and indoors environments, as well as varying object sizes. Compared to existing state-of-the-art architectures at the time of release, F-PointNet could boast with class leading results on the KITTI and SUN-RGBD datasets. F-PointNet exceeds the performance of MV3D by a significant margin for 3D object detection, with similar or better results for localisation while

also decreasing run-time. For the latter dataset, containing objects in indoor environments, F-PointNet additionally obtained significantly higher inference speeds than competing state-of-the-art, at an order of one to three magnitudes.

Contrary to previous works, F-PointNet does not treat the point cloud as projection to an image or as 3D voxels. Instead they rely on more mature 2D object detection, of which F-PointNet is agnostic to the architecture, as it only requires the bounding box dimensions. Using a 2D bounding box, F-PointNet projects a frustum onto the LIDAR point cloud and filters out any points exceeding the boundaries. Within these boundaries the points are treated in its raw format using a PointNet, keeping geometric structures intact. This also yields benefits in cases of strong occlusion and very sparse point clouds. Transformations, as discussed in 3.3.1, are applied to align point clouds into a sequence of constrained, canonical frames that factor out pose variations.

PointNets are typically used to either classify an entire point cloud or semantically segment the individual points, and expanding this to perform 3D object detection is not entirely trivial. The filtered points help reduce the complexity of the search space, which grows cubically with resolution when treated directly. These points are treated by two variants of PointNets; one performing 3D object instance segmentation and the other amodal 3D bounding box regression. The center of the object mask serves as an initial estimate for the object center before regression and is additionally treated by a lightweight T-Net in order to further align the actual center estimation. An overview of the F-PointNet architecture, showing the full pipeline, is featured in Figure 3.5.

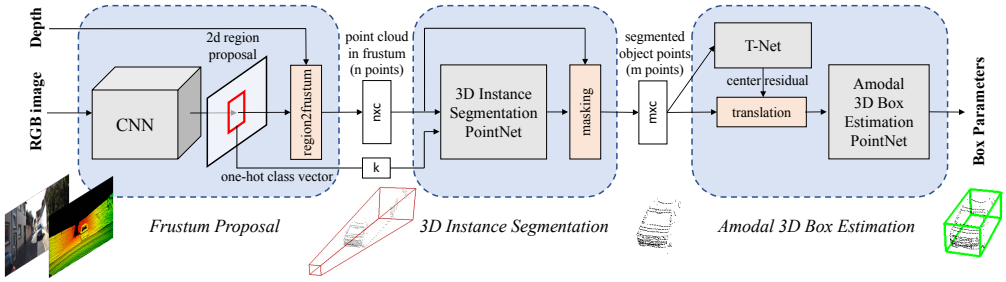


Figure 3.5: Architectural overview of the Frustum PointNet model (from Qi et al. [2018] ©2017 IEEE).

F-PointNet uses a compact bounding box representation for regression, consisting of x, y and z coordinates of the center, as well as height, width, length, pitch, roll and yaw. Qi et al. argue that this unfortunately results in a somewhat disjointed optimization task in terms of the typical losses based on size, angle and center. To better combine the regression targets for loss estimation they therefore add a corner loss, as the sum of the distance between each of the 8 estimated corners from the ground truth. For classification the authors use softmax, while a smoothed L1-loss is used for regression.

3.3.3 RoarNet

The RegiOn Approximation Refinement Network [Shin et al., 2018], named RoarNet for short, is a two stage 3D object detector based around the use of mature 2D object detection for region proposals. Like the aforementioned F-PointNet, which likely served as inspiration, RoarNet uses camera images for 2D object detection and generation of region proposals, which are then used processed by a PointNet for 3D object detection. However, RoarNet differs in several aspects such as adding pose estimation in the 2D detection phase, as well as using different methods for region proposals and regression targets. RoarNet boasts good results, being the leading architecture on cars in the KITTI dataset on the easy and medium difficulties at the time of its submission, 1-2% higher than that of AVOD-FPN while falling further behind on the hard classification by about 7%. A strong point, due to a relatively less restrictive connection

between the 2D and 3D object detection stages, is that RoarNet has a more robust behaviour than earlier multimodal approaches when the camera and LIDAR sensors are poorly synchronized.

The first stage, consisting of 2D object detection and 3D pose estimation from monocular image, is referred to as RoarNet_2D. The 3D poses are estimated by producing dimensions and orientation of detected objects and solving an over-constrained equation for the location of the object, referred to as a geometric agreement search. As the 3D pose of an object is restricted by seven degrees of freedom, namely three coordinates, three dimensions and yaw[1], this restricts the coordinates given the latter four to a finite number of possible combinations that can produce the same 2D bounding box when the projection matrix is known. These estimates are used to determine geometrically feasible object locations, producing potentially several region proposals for a single object that serve as initial seeds for 3D region proposals used in the second stage. The proposals are cylindrical in shape to accommodate potential errors in pose estimation, as well as spatially scattered along the direction vector between camera and object to account for errors in the size of the object using a scattering range. A visual example of proposals with scattering can be seen in Figure 3.6.

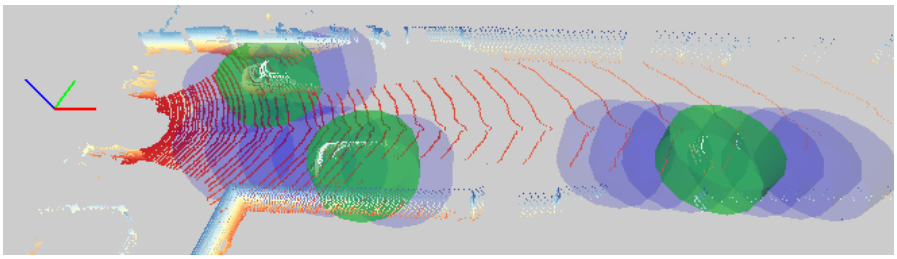


Figure 3.6: Scattered cylindrical 3D region proposals in RoarNet (reprinted with permission from Shin et al. [2018]).

The second stage, RoarNet_3D, consists of two sub-networks, a RPN and a Box Regression Network (BRN) that is used to generate more accurate bounding box predictions from the previous region proposals. These bounding boxes are modelled by taking their location as offset from the center of the region proposal, while formulating angle and dimensions as hybrids of class and raw values, such that for example object size is related to the class. The network produces $2 * N_R$ angle values and $4 * N_C$ size values by distribution into N_R bins, equally divided between 0 and π, and K-Means clustering to obtain N_C clusters. In addition an objectness score is used to estimate the probability that a region proposal contains an object. The RPN and BRN losses are taken as seemingly unweighted sums of the huber loss [Huber, 1992, p. 3] and cross-entropy loss used for objectness, location, classification, rotation and dimensions. However, binary values are used for true objectness, rotation and size when applicable, as well as for 3D IoU below 0.8 to give a form of dynamic weighting. For treating the point cloud, RoarNet uses a simplified PointNet architecture without a T-Net, maintaining the use of max-pooling layers to extract global features directly from unstructured point clouds. A summary of the architecture can be seen in Figure 3.7.

3.4 LIDAR-based architectures

The final category, under which we have placed the rest of the investigated 3D object detection architectures, is those who perform inference only using LIDAR point clouds. As mentioned before, LIDAR is probably the most balanced sensor to be used standalone to determine objects in 3D space. While

[1] simplified from full 3D rotation as objects are assumed to be on the ground plane

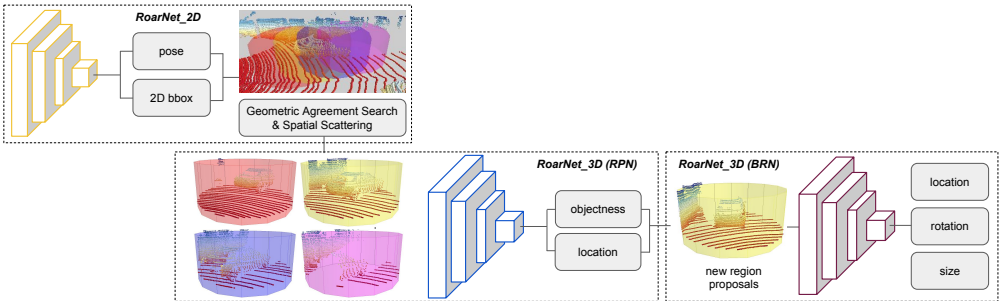


Figure 3.7: Architectural overview of the RoarNet model (reprinted with permission from Shin et al. [2018].

relatively sparse, it offers accurate distance metrics over a wide range of distances. Point clouds are characteristically unordered and sparse, leading to difficulties in processing them. Recent years have seen a lot of research into both general processing of point clouds, such as filtering, as well as for networks to be able to learn directly from the representation of points in 3D space. The introduction of new processing methods, such as in the aforementioned PointNet architecture, means that LIDAR based architectures can efficiently produce more accurate predictions and classifications without discarding crucial information by downsampling.

3.4.1 PointRCNN

PointRCNN is a LIDAR-based model by Shi et al. inspired by the PointNet architecture that outperformed architectures such as F-PointNet, AVOD-FPN and MV3D on cars and cyclists in the KITTI dataset upon its release. The architecture employs a two-stage approach for 3D object detection, similar to the RCNN architectures [Girshick, 2015; Ren et al., 2015; He et al., 2017]. However, unlike the RCNN architectures, PointRCNN operates on 3D LIDAR data, which introduces several challenges as compared to 2D images. While many operate on discretized and projected LIDAR grids or voxels, PointRCNN subsamples the LIDAR point cloud and calculates features per point. These per-point features are then used to generate 3D bounding boxes and segmentation of foreground pixels, which together forms the first stage of the model. The second stage of the PointRCNN model uses these calculated features and predicts refinements of the bounding boxes, as well as confidence for each box.

The first stage of the model can be divided into four objectives: per-pixel feature extraction, 3D box generation, foreground point segmentation and 3D proposal generation. The per-point feature generation uses PointNet++ as a backbone, which returns a feature vector for each point. These points are fed into the 3D box generation module and the foreground point segmentation module. The first module essentially discretizes the LIDAR coordinate system into a fixed number of bins, and uses the foreground points generated by the second module in order to determine the location of the object center. This bin-based method limits the search space of object locations, and discretization is only applied to the X and Z axis. More formally, the bin targets can be expressed as in Equation (3.5). In the equation, a foreground point is defined as $(x^{(p)}, y^{(p)}, z^{(p)})$, while (x^p, y^p, z^p) is the ground truth object center. For predicting orientation, the total orientation of 2π is divided into N bins, and calculated similarly to the X and Z bins.

$$bin_x^{(p)} = \left\lceil \frac{x^p - x^{(p)} + S}{\delta} \right\rceil, bin_z^{(p)} = \left\lceil \frac{z^p - z^{(p)} + S}{\delta} \right\rceil \tag{3.5}$$

After predicting target variables for bounding boxes, also called the bounding box proposals, the extent of each 3D bounding box is slightly enlarged. This is in order to preserve more information of the

points surrounding the object before being sent to the second stage of the architecture.

In the second stage of the PointRCNN architecture, the bounding box proposals from the first stage are transformed to a canonical coordinate system. This ensures that the model will learn the refinement parameters easier, but may result in poorer performance since objects that are further away will have fewer points in the 3D box, and depth information is discarded. Therefore, the euclidean distance from the LIDAR of each point inside the 3D proposal is concatenated with the existing feature vector. Under training, a NMS of 0.85 is applied, and the top 300 3D proposals are passed on to stage 2. When evaluated on the Car class of the KITTI dataset, PointRCNN outperforms the other discussed architectures on all difficulties.

3.4.2 SECOND

Sparsely Embedded Convolutional Detection [Yan et al., 2018], or SECOND, is a very fast and accurate architecture for 3D object detection using only LIDAR point clouds for inference. It is placed as the 10th best performing model for the Car class on the KITTI leaderboards at the time of this writing, being the best performing architecture with openly available documentation and source code. Being capable of inference at 20 Frames Per Second (FPS) with it's fully fledged configuration, and 40 FPS with only a minor decrease in accuracy, SECOND is on paper one of the best suited architectures today for real-time inference in autonomous vehicles.

Yan et al. argue that voxel based 3D convolutional networks remain slow for inference, and have poor orientation estimation performance. Therefore they investigated improved sparse convolution methods that significantly speed up both training and inference. These methods maintain more of the rich and geometric 3D information contained in the point cloud. Spatially sparse convolutional networks are used to extract more height information before downsampling feature maps similar to 2D images. The utilized rule generation algorithm, used for indexation of points, is modified to utilize a Graphics Processing Unit (GPU), which significantly speeds up computation. The structure of the architecture is outlined in Figure 3.8.

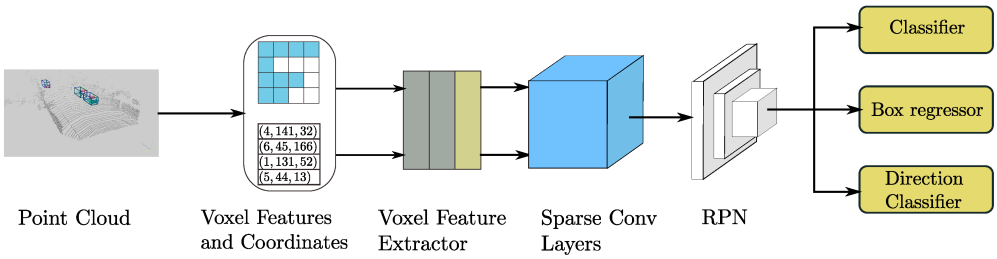


Figure 3.8: Architectural overview of the SECOND model (reprinted with permission from Yan et al. [2018], licensed under CC By 4.0 `https://creativecommons.org/licenses/by/4.0/`).

Before providing the point cloud to the sparse convolution layers, the points are converted to voxel features and coordinates, and passed through two Voxel Feature Encoding (VFE) layers and a linear layer. The output of sparse convolution is then passed on to a RPN, structured as a Single Shot Detector (SSD) that generates detection in form of classification, box dimensions and box direction. Since detected objects are of very similar size for a specific class, SECOND uses fixed sized anchors based on the mean sizes and center locations of all ground truth objects in the KITTI dataset.

The box regression features 7 targets; x, y and z of the box center, width w, height h, length l and yaw θ (often referred to as heading angle). These are taken as variations of the difference or proportion between the ground truth values, marked with g, and the anchor values, marked with a. The full collection

of box regression targets can be seen in Equation 3.6, where d is the diagonal of the base of the bounding box.

$$\begin{gathered} x_t = \frac{x_g - x_a}{d_a}, y_t = \frac{y_g - y_a}{d_a}, z_t = \frac{z_g - z_a}{d_a}, \\ w_t = log(\frac{w_g}{w_a}), l_t = log(\frac{l_g}{l_a}), h_t = log(\frac{h_g}{h_a}), \\ \theta_t = \theta_g - \theta_a \end{gathered} \tag{3.6}$$

The authors argue most previous methods of angle regression exhibit poor performance, and are often subject to the adversarial problem when boxes at relative angle offsets 0 or π fit the object equally well, but generate large losses when misidentified. In order to combat this SECOND uses a new format for angle loss which uses the periodicity of the sinus function. as seen in Equation (3.7). In order to maintain explicit orientation a separate direction classifier is added to the RPN, that is either positive or negative depending on direction. For classification, SECOND follows the method introduced in RetinaNet [Lin et al., 2017] of using a Focal Loss. The Focal Loss helps combat imbalance between foreground and background classes and is described in Equation (3.8). The total loss for the network is simply a weighted combination of the classification loss, regression loss and direction loss, with a relatively low weight on direction classification.

$$L_\theta = SmoothL1(\sin(\theta_p - \theta_t)) \tag{3.7}$$

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma log(p_t) \tag{3.8}$$

SECOND also incorporates a novel form of data augmentation that combats the sparsity of objects in point clouds as well as the potentially poor balancing of classes in the dataset. By applying affine transformations to groups of points in a point cloud, ground truth objects from another point cloud can be copied into the target point cloud. In this way it is possible to augment the scene while maintaining a logical composition through avoiding collisions, as they may not realistically occur in 3D space. This would be far more difficult using 2D images as overlaps may occur depending on the projection plane. Before training, all ground truth objects are collected into a database so that they may be randomly sampled during training, increasing convergence speed and performance of the model.

3.4.3 Complex-YOLO

Complex-YOLO [Simon et al., 2018] can be seen as an extension of the YOLO architecture for 2D object detection by Redmon et al. [2016], and predicts both bounding box dimensions and orientation in a single-shot manner, resulting in real-time performance of over 50 FPS on an NVIDIA Titan XP GPU. The novelty introduced in Complex-YOLO is the *Euler Region Proposal Network (E-RPN)* that encodes each bounding box angle into a real and imaginary part. The preprocessing of Complex-YOLO is based on the same idea as MV3D, insofar that they perform feature extraction from point clouds in BEV using maximum heights, density and intensity values. However, the height values are not taken from a set of vertical slices, but are instead extracted from the entire point cloud, which is a $80m \times 40m$ area in front of the vehicle. The Euler Region Proposal Network introduced in [Simon et al., 2018] is an augmentation to the more common method the authors refer to as *grid-rpn*. The difference can be seen in Equation (3.9) and (3.10), where differing lines are shown in bold. Note that for YOLO, the target regressors are t_x, t_y, t_w, t_h, t_o, while they are $t_x, t_y, t_w, t_l, t_\Phi, t_o$ for Complex-YOLO. The astute reader may have noticed that Complex-YOLO does not predict t_h. This is because each class in Complex-YOLO assumes a pre-defined height, which reduces the number of target regressors, but may fail to encapsulate the intra-class variability in height of each class. For example, 'pedestrians' might have varying height,

from children to adults. However, this fact might be of less importance since detections of the object is most important, not its height.

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \\ Pr(Object) * IOU(b, object) &= \sigma(t_o) \end{aligned} \tag{3.9}$$

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ \mathbf{b_l} &= \mathbf{p_l e^{t_l}} \\ \mathbf{b_\Phi} &= \mathbf{arg}(|\mathbf{z}|\mathbf{e^{ib_\Phi}}) = \mathbf{arctan_2}(\mathbf{t_{Im}}, \mathbf{t_{Re}}) \\ Pr(Object) * IOU(b, object) &= \sigma(t_o) \end{aligned} \tag{3.10}$$

3.4.4 PIXOR

PIXOR [Yang et al., 2018] is a deep architecture created at the Uber Advanced Technologies Group that applies predictions in a pixel-wise manner from a BEV-projection of the point cloud. Using this representation, the architecture predicts bounding boxes for each pixel, which makes it a single-stage detector. In order to encode the LIDAR data to a suitable input for the architecture, the LIDAR point cloud tensor of shape $L \times W \times H$ is discretized with a suitable resolution in each dimension, 0.1m in the case of KITTI. In addition, the intensity of each point in the discretized cell is stored in the last channel of the output tensor, along with the occupancy value, which is 1 if the cell contains a point, else 0. The resulting output tensor is thus a $C_l \times C_w \times (C_h + 1)$, where C_i is the number of discretized cells in dimension i.

The architecture of PIXOR consists of a backbone network and a header network, shown in Figure 3.9. The backbone network is comprised of convolutional layers with 3×3 filters, followed by residual blocks and upsampling layers. The header network consists of convolutional layers, followed by two different branches of convolutional layers that each outputs a map. One convolutional layer outputs a $200 \times 175 \times 1$ map that encodes the class probability of each cell, while the other outputs a $200 \times 175 \times 6$ tensor that represent the bounding box parameters of each pixel.

A novelty of PIXOR is that each bounding box is represented as $(cos(\theta), sin(\theta), dx, dy, w, l)$, where θ represents the object's heading angle, dx, dy represents the offset of the objects location relative to its pixel position, (xc, yc), and w, l are the objects width and length, respectively. During training, the bounding box predictions are represented as $(cos(\theta), sin(\theta), log(dx), log(dy), log(w), log(l))$. Since each pixel in the output tensor corresponds to its own prediction, there is a massive class imbalance in the training set. Therefore, Focal Loss is employed to combat this issue, by weighting difficult samples more. The loss function of PIXOR is a combination of the cross-entropy loss for classification and Smooth L1 loss for regression. Since each pixel outputs a prediction of bounding box parameters, but only a minority of the pixels actually contains an object, the L1 loss is only computed over locations with positive examples.

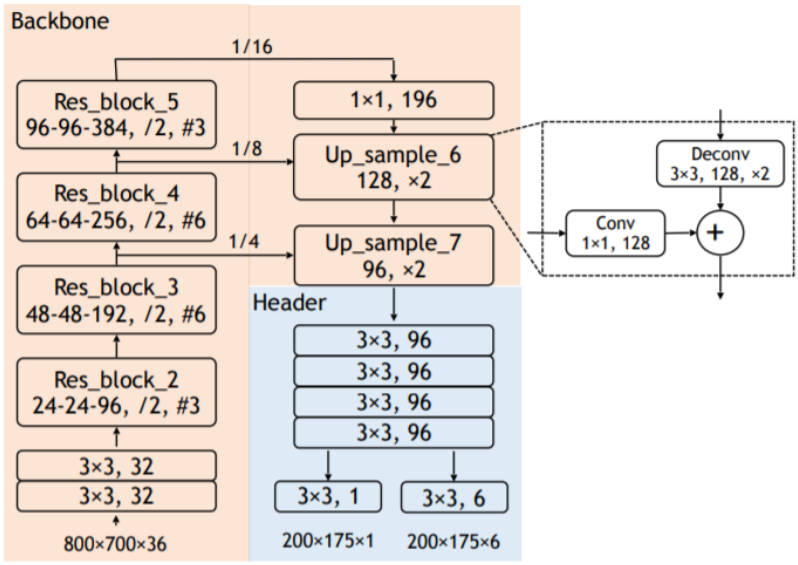


Figure 3.9: Architectural overview of the PIXOR model (from Yang et al. [2018] ©2018 IEEE).

Chapter 4

Architecture

In this chapter, the main architecture used for the experimentation presented in the report is discussed. The chapter also discusses how the architecture is modified to experiment with different LIDAR feature map representations, and the motivations behind these modifications.

4.1 Choice of Architectures

As the focus of this report of is to investigate and experiment with state-of-the-art sensor fusion, AVOD-FPN stands out as a clear candidate for experimentation. Firstly because it is the only current state-of-the-art architecture that is available open-source and configured for deep sensor fusion, as opposed to only using the camera for 2D object detection and proposal generation. Additionally, most of the 2D proposal based architectures are outperformed by AVOD-FPN by a significant margin either in terms of measured accuracy on one or more classes or in run-time speeds, or had parts of the source code not fully available before beginning the experiments. When it comes to training on simulated data and validating performance, state-of-the-art LIDAR based architectures will also be considered, in this case using SECOND due to its strong documentation, source code availability and performance. AVOD-FPN and SECOND are both used for testing detection of objects in single- and multi-class configurations for real and simulated data, while the AVOD-FPN architecture is additionally modified such that multiple different BEV feature map representations can be tested and compared.

4.2 Modifications to the AVOD architecture

As 2D object detection is a much more mature field, both in autonomous driving and the general field of artificial intelligence, the focus of experimentation with the AVOD-FPN architecture is on the processing of LIDAR point clouds. More specifically, experimentation was dedicated towards determining what features to use and how to divide the point cloud into BEV maps. In order to easily be able to switch between configurations of the BEV representation, the model was set up to take in two new configuration parameters under slices segment of the bev_generator in the config files: slice maps and cloud maps. Both these parameters are defaulted to empty lists and take in the types of features to be used from either each slice of the point cloud, or taken from the point cloud as a whole. With no parameters defined, the model defaults to the standard AVOD-FPN configuration from Ku et al. [2018]. The slices evenly segments the point cloud vertically within the boundaries supplied by the height_lo and height_hi parameters and the count num_slices. Following is a description of all the tested configurations as well as a discussion on

why the representation could potentially be better or more efficient than the default, as well as potential weaknesses of each configuration. Results from the experiments will be listed chapter 5 and discussed in chapter 6.

4.2.1 Default Configuration

The default configuration, as described in Ku et al. [2018] and used for official submissions to the KITTI leaderboards [Geiger, 2019], serves as the baseline for testing. All other configurations are constructed from making the minimum required changes to the original configuration in order to attempt to isolate parameters. Shared by all configurations is a horizontal discretization into voxels at a resolution of 0.1m within the boundaries of 40 meters left and right, 70 meters forward, 2.3 meters down and 0.2 meters up. All other points in the point cloud are filtered out before generating feature maps. The default configuration consists of a total of 6 feature maps; 5 height maps and 1 density map. The 5 height maps are generated by vertically slicing the point cloud, within the boundaries defined above, into 5 equally large slices. For each slice, the highest point within each discretized voxel cell, measured as the height from the estimated ground plane if provided, is stored and normalized to the height of the slice, producing a value between 0 and 1. For the additional density map, the whole point cloud is considered, and the number of points existing within the height boundaries of each cell is counted. The density of the LIDAR point cloud is also normalized logarithmically to values between 0 and 1, with upper values being clipped following Equation (4.1). This combined representation of accurate, but potentially noisy raw height values with the smoother, aggregated information represented by density is shown to produce strong results. The set of feature maps is visualized in Figure 4.1.

$$min(1.0, \frac{log(N+1)}{log(16)}) \tag{4.1}$$

4.2.2 3M3D

LIDAR sensors are often susceptible to noise, and this noise is particularly troublesome when utilizing only a sample of the raw point cloud values represented in the height maps. The height values in the feature maps gives an indication of object dimensions, but it may in some cases be difficult to whether a point actually belongs to an object or not when only selecting one in a cell. This value might be an outlier, or a part of an overhanging surface. Thus, a more interlaced use of height and density maps might be able to better combine the strengths and weaknesses of each representation. By segmenting the point cloud into 3 slices, taking the highest point and density of each cell in each slice, the intermediate density layers might help reduce the impact of outliers, at a cost of lower vertical resolution. The density is calculated as before, but with the number of points N now being counted separately for each slice. The configuration is visualized in Figure 4.2.

4.2.3 MMD

In an attempt to reduce computational complexity we propose a configuration consisting of only 3 feature maps. Instead of slicing the point cloud into segments and taking the highest points in each, the whole point cloud is considered when extracting the raw height values in the form of the highest and lowest point. This representation is clearly more compact, and can speed up inference on the LIDAR side of feature extraction, but may be more susceptible to noise situated both above and below objects of interest as well as offering less vertical resolution. While the minimum point might seem superfluous if all objects are on the ground plane, it does offer the potential for correcting errors in ground plane estimations. The density map is identical to the default configuration, with the full configuration visualized in Figure 4.3.

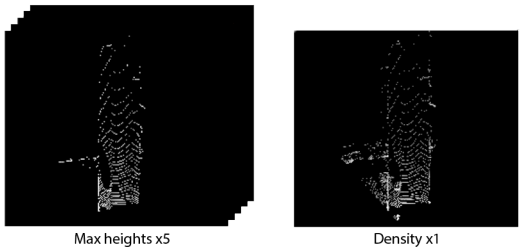


Figure 4.1: Visualization of the Default AVOD BEV configuration, taking the maximum height within 5 vertical slices as well as the density of the full point cloud.

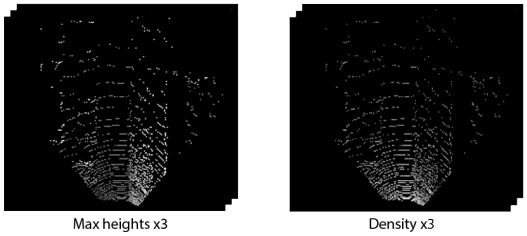


Figure 4.2: Visualization the 3M3D AVOD BEV configuration, taking the maximum height and density within 3 vertical slices.

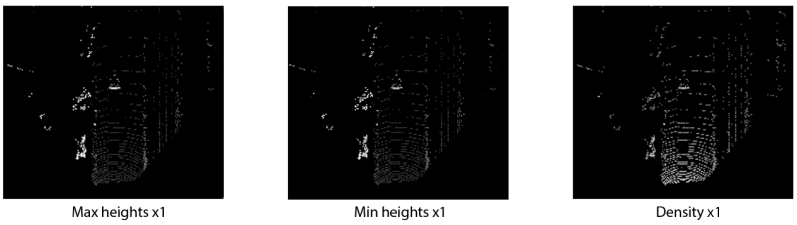


Figure 4.3: Visualization of the MMD AVOD BEV configuration, taking the maximum height, minimum height and density of the full point cloud.

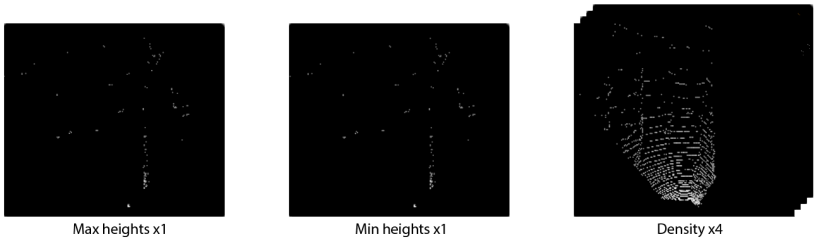


Figure 4.4: Visualization of the MM4D AVOD BEV configuration, taking the global maximum and minimum heights in addition to 4 vertical slices of density.

4.2.4 MM4D

By extension of the configurations described in sections 4.2.3 and 4.2.2, we also propose a configuration using the global maximum and minimum height values combined with multiple layers of density. By using 4 layers of density, the overall complexity of the default configuration is maintained, while providing more granular density information that can aid in accurately correcting for the potentially noisy global height values. Segmenting the density into vertical slices might also improve performance when objects are partially occluded, resulting in a visible shadow in the LIDAR point cloud. In these cases only a smaller vertical part of the point cloud might have the expected density of a certain object. The BEV feature maps are visualized in Figure 4.4.

4.2.5 3M3DND

The dispersion of the LIDAR beams results in more distant objects producing less dense collections of points. Although the LIDAR feature maps produce size invariant object representations in BEV, the relation between density and distance is an additional function a neural network would have to approximate. This could potentially instead be tackled through scaling the density with the distance D. In order to test this an extension of the previous configuration was designed that includes scaling of cell densities by the euclidean distance of the cell from the point cloud origin by its discretized x and y coordinates. The Distance Normalized Density (DND) and visualization of the configuration can be seen in Equation (4.2) and Figure 4.5 respectively.

$$min(1.0, \frac{log(N+1) * D}{log(16)}) \tag{4.2}$$

4.2.6 Cluster

The final tested configuration in essence includes the same 3 feature maps as proposed in section 4.2.3, using global maximum height, minimum height and density. However, before being passed to the network the point cloud is filtered such that only the largest vertical cluster is kept. A vertical cluster is here defined as a group of points, discretized to the same cell, whose vertical distance is less than the maximum of either the average vertical distance of all points in the cell or the voxel size (defaulted to 0.1m). The resulting point cloud, depending on how aggressively applied in terms of grouping thresholds and required amount of points in a cluster, can potentially reduce noise or even completely remove unnecessary points such as those belonging to the ground or a roof above the LIDAR sensor. Such a filtering approach although does come with a risk of removing points belonging to objects of interest as well, while also producing an additional preprocessing overhead.

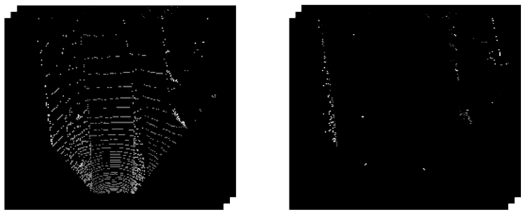


Figure 4.5: Visualization of the 3M3DND AVOD BEV configuration, taking the maximum height and distance normalized density within 3 vertical slices.

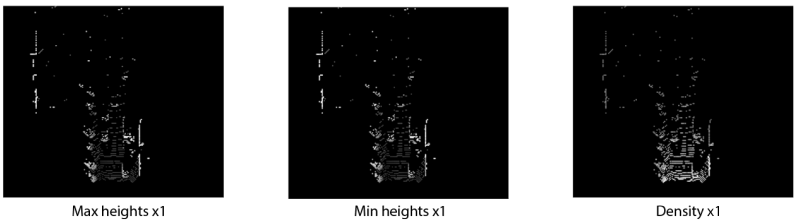


Figure 4.6: Visualization of the Cluster AVOD BEV configuration, clustering the points vertically and then taking the maximum height, minimum height and density of the largest cluster.

Chapter 5

Experiments and Results

This chapter introduces the reader to all the experiments conducted and how each experiment is related to the overall goal of the project. Firstly, the experimental plan is presented, which discusses the high-level objective of each experiment. Secondly, the experimental setup details how each experiment was conducted in such detail that it should be both understandable and reproducible. Lastly, the experimental results section presents the results of each experiment in an orderly manner and comments on them briefly. A thorough discussion of the results presented in this chapter is presented in Chapter 6.

5.1 Experimental Plan

Since the project is inherently multifaceted, the authors propose dividing the experimental plan into several groups. These groups are Data generation tool, Model training and BEV experimentation, and Inference on the NAP dataset. Each of these groups are relatively self-contained, and will thus have a separate experimental plan. The ordering of these groups is intentional since each group depends on the group before it. For instance, the **Model Training and BEV experimentation** is explained after the data generation tool, since it requires the data generated by the tool in its experiments.

With regards to the **data generation tool**, the entire experiment can be described as one process, which is to export data from an autonomous vehicle simulator. This process can be divided into several tasks such as choosing a simulator, simulator configuration, sensor calibration, occlusion detection and label generation, which together define the pipeline from simulator to generated dataset. Section 5.2.1 provides a detailed explanation of each of these steps. The data generation tool will be evaluated by inspecting the annotated data it creates, considering quality measures of each data sample such as the fit of bounding boxes, and the overall statistics of the generated dataset such as class distribution. The former is performed through a visual evaluation, while the latter is analysed by comparing the dataset with the aforementioned KITTI dataset by Geiger et al. [2013]. A tool for generating synthetic data for autonomous driving inherently valuable, as it allows for researchers to freely create data from the simulator in order to train machine learning models, without needing to worry about sensor calibration, occlusion detection, and other issues with exporting data from a simulator.

Model Training and BEV experimentation covers everything relating to how AVOD-FPN and SECOND are used in order to investigate whether or not using simulated data can benefit object detection networks for autonomous driving. This investigation will be performed in multiple stages, using models trained fully on the KITTI dataset, trained only on simulated data or pre-trained on simulated data before being fine-tuned on the KITTI dataset. By comparing performance across multiple models using these

different training regimes, the aim is to be able to evaluate how well training on simulated data works when evaluated on the KITTI dataset, as well as whether solid performance can be achieved with less training or with a smaller KITTI training set. Additionally, experiments using the discussed modifications to the AVOD architecture will serve as a basis for evaluating the potential of novel BEV feature maps, and included with training on real and simulated data.

The experimental plan for the **Inference on NAP dataset** is created with two milestones in mind; to be able to export sensor data for labeling and inference and to run real-time inference with an object detection architecture in the car. These experiments are performed in order to evaluate the generalization of our trained models on data from arctic environments, again comparing performance with and without simulated pre-training, and to create a framework to easily curate datasets in the future.

The following sections will adhere to the same template, where the data generation tool is discussed first, followed by training and evaluation of object detection models, and concluding with inference on the NAP dataset. The main goal of this chapter is to make the findings described in this thesis as detailed as possible, such that they are reproducible. The issue of reproducibility is discussed more thoroughly in Section 7.4.

5.2 Experimental Setup

This section details all necessary information to fully reproduce the experiments outlined above, as well as some of the key algorithms and processes behind some of the experiments. Some of the details regarding simulator configuration or training parameters for the architectures are omitted and can be found in the code repositories listed in Appendix A.

5.2.1 Data generation

For choosing a simulator, the authors focus on the criteria laid forth in Section 2.4.1. The main differences between each simulator is mainly regarding photorealism and whether or not the simulator is open source. NVIDIA Drive Constellation outperformed all other simulators with regards to photorealism. Unfortunately, NVIDIA Drive Constellation was at the time of this writing only limited to a handful of research institutions, and thus not available for experimentation. In addition, Baidu/Apollo was discarded as the entire simulator is only available in a closed-source online platform, which makes it impossible to perform modifications to the simulator itself, such as modifying its rendering time or photorealism. This leaves AirSim and CARLA, which are very similar with regards to implemented features and grade of photorealism. However, due to AirSims focus on both multicopters and cars, the authors decided that CARLA would be the better choice, since it has its sole focus on autonomous driving. In addition, the API of CARLA could be easily used to extract all data required for sensor calibration and projection, such as the location of vehicles and sensors.

Data generation tool

In order to generate simulated training data for 3D object detection, the LIDAR data, images and training labels has to be extracted from CARLA. To be able to achieve this, the authors introduce the Carla Automated Dataset Extraction Tool (CADET). The tool performs projection from 3D world coordinates to pixel coordinates, with training labels that correspond to the KITTI format defined by Geiger et al. [2013], in addition to calibration between camera and LIDAR, and ground plane extraction.

The CARLA simulator was set up to have a resolution of 1248×384 with an fps of 10 in CARLAs benchmark mode. By running in benchmark mode we are guaranteed that the simulator is running as fast as possible, and that the time between each rendering is fixed. The resolution above yields the intrinsic matrix seen in Equation (5.1), and mimics the resolution of the images in the KITTI dataset. Note that the

horisontal and vertical angle view are assumed to be equal, and is calculated using Equation (2.2) using a camera FOV of 90 degrees. Furthermore, the LIDAR was set to have a range of 70 meters, with vertical FOV in range $[-16, 7]^\circ$ similar to the Hesai Pandora sensor described in Section 2.5.1. The LIDAR sensor in CARLA is creating 720 000 points per second with a rotational frequency of 10 Hz. Since the rotational frequency of the LIDAR is equal to the frame rate of the simulator, we are guaranteed to have a full 360 degree scan of the surrounding environment for each generated training sample. The camera and LIDAR are synchronized by the simulator, so there exist correspondence[1] between LIDAR scans and camera images.

$$K = \begin{bmatrix} \alpha_x & 0 & C_u \\ 0 & \alpha_y & C_v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 624 & 0 & 624 \\ 0 & 624 & 192 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

In order to export training labels in the format defined by Geiger et al. [2013], the bounding box of each vehicle in the scene has to be extracted. Each agent in CARLA is described by a 3D bounding box defined by 8 vertices. First, the vertices are transformed as to be relative to the world coordinates. The 2D pixel coordinates of each vertex in world coordinates are calculated using Equation (2.1). Vertices outside the camera perimeters or with a distance too far away from the camera are removed. This distance was set to 70m, as to correspond to the LIDAR maximum range.

In order to estimate the ground plane, the pitch (θ) and roll (ϕ) of the vehicle is extracted. Furthermore, we define a unit vector in the height direction and rotate it with regards to the vehicles rotation. Note that yaw is disregarded as it does not affect the direction of the normal vector. The ground plane vector is thus a rotation around the pitch and roll axis, which corresponds to the vector defined in Equation (5.2).

$$y_{normal} = \begin{bmatrix} cos(\theta)sin(\phi) \\ -cos(\theta)cos(\phi) \\ sin(\theta) \end{bmatrix} \tag{5.2}$$

Since CARLA is a simulated environment, it is perfectly fine for multiple sensors to occupy the same position. The camera and LIDAR sensor is placed relative to the car, 1.6 meters above the car's center[2], which means that the rectification matrix between required to transformed points between the two coordinate systems will be an identity matrix. In addition, the translation vector between the two coordinate systems will be the zero-vector. In order to replicate the KITTI coordinate system, we convert the points in the CARLA coordinate system to KITTI's. Thus, for a point $[X, Y, Z]$ in CARLA, the exported location vector will be $[X, -Y, Z - LIDAR\_HEIGHT]$. This corresponds to a flip of the y-axis, and a translation in the z-axis. The former is because CARLA uses a left-handed coordinate system, while the latter is a transform from the local space of the sensor to the space defined by the vehicle. The calibration matrix TR^{cam}_{lidar}, which corresponds to the transformation between LIDAR space and camera space, transforms a LIDAR point $[X, Y, Z]$ to $[-Y, -Z, X]$. The transformation matrix as written to the calibration files is shown in Equation (5.3).

$$TR^{cam}_{lidar} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.3}$$

For debugging and illustration purposes, CADET is also able to visualize both LIDAR and 3D bounding boxes while running the simulator. As the inherent transformation of the LIDAR sensor does not correctly bring the LIDAR point cloud into the coordinate space used by the car, an additional transformation in CARLA is applied that rotates the point cloud 90 degrees around the y-axis and flips the z-axis.

[1] This is merely an approximate correspondance, as the LIDAR ray-cast uses the underlying mesh of an object as the reflective surface, while the camera sensor uses textures. This difference is detailed in Chapter 6

[2] The center, or location of the vehicle in CARLA is defined as the midpoint of the bottom plane of the vehicle.

During experimentation, the authors found that although the camera was affected by the rotation of the vehicle, the LIDAR sensor was not. This meant that any data captured while the vehicle was affected by pitch or roll had a mismatch between camera and LIDAR points. The authors opted to correct the LIDAR to follow all vehicle motions rather than making the camera static, as the former would result in more realistic and difficult scenarios with slight tilt and rotation in images and point cloud. This was achieved by shifting the point cloud origin to ground level by subtracting the sensor height, then reading the vehicle pitch and roll and performing rotation using a combination of pitch and roll matrices, seen in Equation (5.4), through matrix multiplication. Finally, the point cloud is shifted back to its original height. In order to convert to the KITTI format the last transformation required is flipping the y-axis and adding a static intensity value. In a real-world sensor, the intensity value is a measure of how much light that was reflected of the surface. This data is not currently available in CARLA, and a static intensity value of 1 was used for all points. Using the inverse of the transformation from camera to car space, the LIDAR can be visualized in real time while running the simulator. The 3D bounding boxes are drawn using the projected 2D pixel coordinates of each vertex of the given bounding box together with a graph that defines the connection between each vertex. CADET visualizes the connections between these boxes using a simple line drawing algorithm shown in Algorithm 2.

$$Rot_{pitch} = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \qquad Rot_{roll} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{5.4}$$

In order to simulate a wide variety of driving scenarios, a handful of rules regarding data generation were devised. These rules ensure that our dataset will be sampled from many different scenes, including variation in weather, vehicle models and locations. In addition, they ensure that our agent does not get stuck, which may happen since the car is driving on the autopilot that exist in CARLA. In particular, the rules are:

1. Record only each 10th frame, which corresponds to 1 sample per second at 10 FPS
2. The car must drive at least 10 meters between each generated sample
3. A ground truth label will only be generated if the 2D bounding box of the agent occupies at least 100 pixels in the image
4. A ground truth label will only be generated if at least four of the agents 8 vertices are not occluded.
5. The simulator may only generate a maximum of 20 samples per environment.
6. If there are 40 empty frames since a recording, reset the environment.

In order to detect the occlusion of vertices, the camera depth map is utilized together with the vertex depth. The vertex depth is the Euclidean distance between the camera sensor and the vertex in camera coordinates. For each vertex the depth of the four neighbouring pixels are retrieved from the depth map. If all pixels have a depth that is shorter than the depth of the vertex, the vertex is defined as occluded (see Algorithm 1 for a rigid definition). This local depth search runs reasonably fast, but may give a lot of false positives, especially when a vertex is occluded by a small object such as a light poles or chain link fences. Therefore, an occluded object is defined as an object where at least five out of the eight total vertices are occluded. The method outlined above works quite well in practice, but may fail if see-through objects such as fences are modeled by the simulator engine as walls, thus rendering a uniform depth in the depth map.

When the conditions specified above are met, CADET generates a new sample. This sample consists of a LIDAR pointcloud saved as a 3D array, a PNG image from the camera, calibration matrices, the

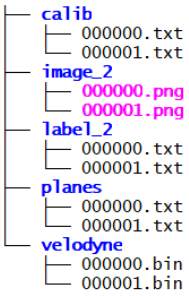


Figure 5.1: Example of dataset folder structure with two training samples. This is very similar to Geiger et al. [2013], except that ours contains an additional `planes` folder, which is the groundplane estimation of each training sample required by for example AVOD-FPN.

estimated ground plane, and finally the ground truth labels of objects in the scene. CADET creates the resulting dataset in a folder structure shown in Figure 5.1, here shown only containing two samples for brevity. For a detailed explanation of what each sample consists of, please refer to Chapter C in the Appendix. CADET automatically generates this structure, and also has the options of appending or overwriting the currently generated dataset if one would like to make some changes to the simulator or CADET settings after generation of the dataset has already been started.

5.2.2 Experimental setup for model training

As most the considered models are designed for the KITTI dataset, and the nuScenes dataset was not released before beginning the experiments in this thesis, it is a natural choice for the performed training on real-life data. Other potential datasets, such as BDD100K, for example does not feature LIDAR data and would be unsuitable for the desired experiments.

Training and evaluation on the KITTI and CADET datasets were performed using predefined splits into training and validation sets. The KITTI dataset was divided into a 50/50 training/validation set following common practice in similar work, while the slightly larger, simulated CADET dataset was divided into a 80/20 training/validation set as we care less about accurate validation performance and more for variety in the training data. All run-time estimation was performed on a machine fitted with a NVIDIA GTX 1080 GPU, an Intel i7-6700 CPU and 32GB of memory, similar to what is installed on the test car used to gather data. Each model performed inference on the first 2000 images of the validation set, of which the mean inference time was rounded to the nearest millisecond. Other than adding more classes, the KITTI evaluation code included in the original repositories was not modified. Due to the extensive testing and active use of these architectures through GitHub, the authors consider these evaluation scripts to be accurate.

AVOD setup

In order to gather qualitative results, each AVOD model was trained for a total of 120k steps on the respective datasets, with a batch size of 1. After training, the last 20 checkpoints stored at an interval of 2k steps were evaluated and the best was selected for the presented results. Models that are initially trained on the CADET dataset were modified for fine-tuning on the KITTI dataset such that training is resumed from step 90k until step 150k, meaning the models only received half as much training on real data when compared to the fully KITTI-trained models. Learning rate was not altered for the fine-tuning step, such that it was gradually reduced from its original value at 90k steps. For every test of a new

BEV configuration, preprocessing was performed again using the accompanying configuration file under mb_processing. Visualized bounding boxes are generated by first running inference using the selected model, after modifying the configuration file to point to the correct dataset, followed by running the show_predictions_2D script under demos. As AVOD includes the configuration name in every stored file when training, some manual steps are required for fine-tuning a model rather than us including a separate configuration file for fine-tuning. Therefore we list the required steps for resuming training on a CADET pre-trained model:

- Backup the folder including the original, pre-trained files.
- Create a new folder using the same configuration name as the original pre-trained model.
- Copy the configuration file and checkpoint files for the wanted checkpoint over to the new folder.
- Modify the copied file named ”checkpoint” such that the list of checkpoints and latest checkpoint point only to the checkpoint selected for resuming training.
- Modify the original configuration file (in avod/avod/configs) and update the number of training steps as well as the referenced dataset (for example altering dataset name and dataset_dir from carla to kitti).
- Make sure that the preprocessed mini-batches are correct for the specific model and dataset and delete pre-existing preprocessed files if necessary.
- Resume training in the same way as when training a new model by specifying the modified configuration file. The model folder will be recognized and training will be resumed.

SECOND setup

SECOND models were trained with a batch size of 6 for 50k steps for single-class and 70k steps for multi-class, with evaluation of the last 15 checkpoints saved at intervals of 2k steps. For fine-tuning multi-class models on the KITTI dataset after pre-training on the CADET datasetm models were restored at 50k steps and trained for an additional 30k steps. As SECOND does not save the model name with the files, the new fine-tuned model can be given a new name after simply copying the checkpoint files from the original model. A configuration file for fine-tuning is included with the modified source code. Tested configurations were based upon existing single- and multi-class configurations, such that slight architectural differences exist between single-class and multi-class. Notably the multi-class configurations feature a smaller point cloud and anchor range, less input features, less filters, simplified voxel extraction, an increased maximum number of voxels and reduced weight decay. This results in faster inference speeds for multi-class of about 30%, but is for not included with the multi-class results for brevity. Readers are referred to the source code [Brekke and Vatsendvik, 2019a] for full specification of all configurations.

5.2.3 Experimental setup for NAP dataset

Experimental Vehicle

The real world data used in this report is gathered using a KIA Niro Plug-In Hybrid with a Hesai Pandora LIDAR/camera sensor suite, GPS and a Nuvo-7160GC on-board IPC . All sensors communicate with the IPC using ROS. The vehicle also contains a DriveKit drive-by-wire system, although this was not used when gathering data.

ROS

The ROS system used by the authors to perform the experiments related to inference on the NAP dataset and described in Section 5.1 was ROS Lunar-Loggerhead, which was released May 23rd, 2017. Although the Lunar distribution has End Of Life (EOL) date in May 2019, it was chosen because of its compatibility with the authors computer systems running Ubuntu 16.04 Xenial Xerus. However, there are no dependencies used in the ROS setup that are not available in the latest ROS release, which at the time of writing is Melodic Moriena. In order to experiment with real data, the authors devised a setup that reads from a ROS-bag and publishes its topics to a LIDAR node and a camera node. Furthermore, a callback was created such that the model in question could run inference in real-time when it received messages on both the LIDAR and image node. Note that the camera and LIDAR nodes were synchronized using the *ApproximateTime Policy* found in the ROS `message_filters` package. This allows nodes with different frequencies to be synchronized approximately based on the timestamp in the message header. Figure 5.2 describes the general data flow of the ROS setup. A generic ROS topic source, which can be either playback from a ROS bag or directly a sensor such as the Hesai Pandora, publishes image and LIDAR topics. Both an image and LIDAR subscriber are set up, and using the message filter *ApproximateTime* Policy, results in a single callback. Furthermore, both SECOND and AVOD requires the input data to be in the KITTI format, which creates the need for a dataconverter in the ROS data flow that converts the ROS message data, which is ROS *PointCloud2* and *Image* messages, to KITTI's binary matrix format, and numpy image arrays.

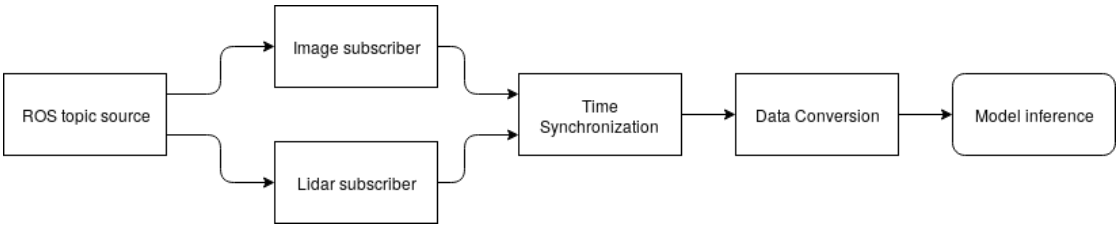


Figure 5.2: Data flow in ROS setup

Modifying models for real-time inference

Both AVOD and SECOND are models mainly intended for the research of autonomous driving object detection architectures, and do not have any easily configurable option to do online inference[3]. For instance, AVOD requires the generation of mini batches and label clusters for its region proposal network. Normally AVOD performs inference in batches of existing data, and has to load the model in its entirety for every prediction batch. Therefore, the authors introduce new functionality to AVOD such that the model and its weights are only loaded once in the entire session. The introduced functionality allows AVOD to run seamlessly in a ROS system, and can be used for online inference with, The model API makes it seamless to pass images, point clouds and calibration matrices from a ROS system to the model without writing anything to file as an intermediate step. As detailed in Chapter 3, AVOD requires an estimation of the ground plane in order to perform inference. At the time of recording the NAP vehicle was not equipped with an IMU, which meant that performing an estimation of the vehicles tilt would be infeasible. The AVOD model was therefore fed the ground plane vectors from KITTI.

SECOND also only includes code for training and evaluation on a predefined list of preprocessed files. However, SECOND includes a javascript application, called kittiviewer, for viewing and performing inference on individual LIDAR point clouds from the preprocessed files using a Python backend. The

[3]Online inference is also referred to as dynamic inference, or making predictions on demand

backend restores the weights once and can then perform inference on individual samples by request, but requires the use of a sample index and an info file generated during preprocessing. Instead, the authors have constructed a new interface based on the original backend for the kittiviewer and added functionality in the core code of the SECOND framework. These modifications allow SECOND to perform inference by directly providing the LIDAR point cloud, calibration matrices and image shape, without batched preprocessing or the need for stored files. Just as done with AVOD, this allows for easily setting up the model once and then performing inference in a ROS loop with little added overhead. As the anchor sizes are included in the configuration files, no preprocessed files from KITTI are required. We also include code that directly performs inference and draws the projected 3D bounding boxes on the provided images. This code is based on the code used in AVOD for consistency, and modified to work with the generated annotations from SECOND.

5.3 Experimental Results

In this section, the results of each experiment will be discussed. First, the generated data of CADET will be analyzed, with emphasis on the quality of the generated data. Furthermore, the same tool is used to generate the CADET dataset consisting of 10,000 training samples with labeled cars and pedestrians. This dataset is studied with regards to bounding box dimensions, class distributions and number of labels per image. In addition, the results of the machine learning models that are pre-trained on this dataset are discussed, with comparison to results from only training on the KITTI dataset. Lastly, a selection of these trained models are used to perform and visualize inference on the KITTI dataset, CADET dataset and a real world, arctic dataset, referred to as the NAP dataset, gathered by the recording vehicle outlined above.

5.3.1 Data generation

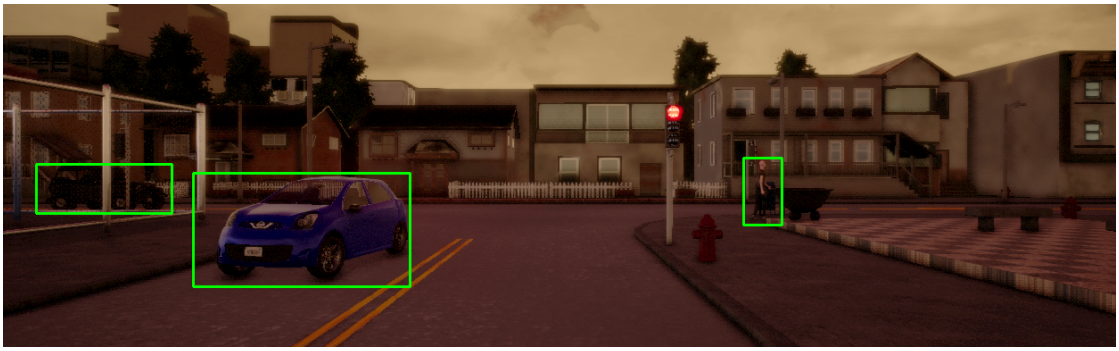
Data generation tool

When evaluating a simulation toolkit, which the CADET tool certainly is, the method of evaluation is often very dependent on how the tool is being used. For CADET, one important measure is how fast the tool is able to generate samples, which is dependent on a number of factors. In particular, regarding the rules set forth in Section 5.2. If one were to decrease the frames between each capture, the number of samples per unit of time would increase. However, this could lead to a number of fairly similar samples, for instance if the agent vehicle is tailing another car. The speed of the data export tool is estimated by generating a dataset with 10,000 samples, and measuring the expended time. By this metric, the data export tool is able to generate approximately 10 samples each minute, or one per sixth second. It should be noted that the speed of the data generator can be increased vastly if the number of environmental resets are decreased. The simulator uses up to 5 seconds to generate a new environment after a reset, which happens fairly often using the rules laid forth previously. We also note that speed of dataset generation depends on computer hardware, in particular the GPU and CPU. When generating our dataset we used a machine fitted with an NVIDIA GTX 1080ti and an Intel i7-6700 CPU.

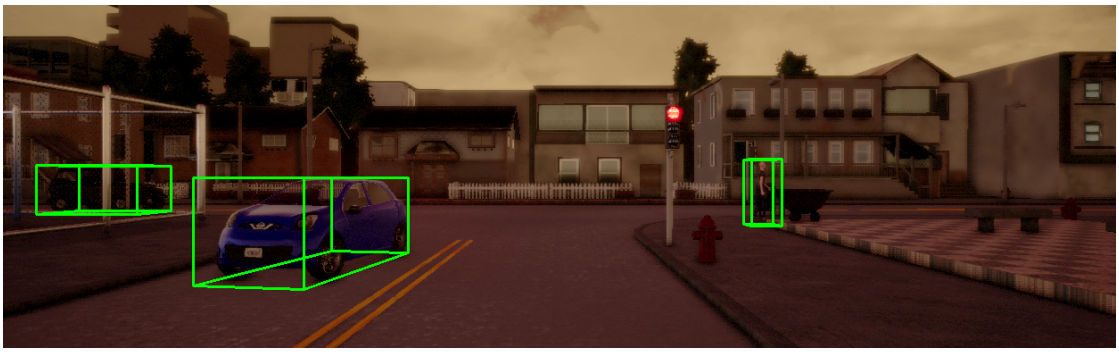
More important than speed is the *quality* of the generated samples. Here, quality is defined as how well the bounding boxes fit to the objects in the scene, as well as how the LIDAR point cloud and image corresponds. The correspondence between an object as depicted in the image, and its bounding box as labeled is shown in Figure 5.3, both for 2D and 3D bounding boxes. Figure 5.6 shows the LIDAR point cloud projected into the image plane, which illustrates the correspondence between objects in the scene and their LIDAR points. A third criterion of the data tool is how well occlusion is dealt with. As discussed in the previous section, vehicles behind see-through objects such as chain-link fences can be marked false-positives if their vertices are barely occluded. In addition, the occlusion detection is only

as good as the depth buffer provided by the simulator. In Figure 5.4, the vertices and bounding boxes are drawn on screen for visibility. All occluded vertices in this example are correctly identified. As is the case for see-through objects, such as the chain-link fence shown in Figure 5.5. The vertices of the grey car are correctly identified as occluded when they appear behind the pole of the fence, while they are marked as visible even when viewed behind two layers of fence. Figure 5.7 shows a fully occluded object behind a solid wall. Lastly, Figure 5.8 illustrates that vehicles outside the maximum range of 70 meters are discarded, while nearby vehicles are labeled.

In addition to the criteria discussed above, a crucial part of the quality of the data generation tool is how well the simulated sensor data approximates that of a real world sensor. Sensors in the real world are often affected by noise, inaccuracies and environmental aspects, which are often not accounted for in simulators. With regards to the camera sensor we observe that the simulation is quite photorealistic and is affected by weather, yielding effects such as lens flare and bloom. However, the LIDAR sensor included in CARLA does not offer the same realistic approximation as the camera. Comparing the LIDAR scan of a vehicle in CARLA with an actual LIDAR scan from the Hesai Pandora. which the LIDAR simulator is set to approximate, we get the results shown in Figure 5.9. The same comparison is displayed for pedestrians in Figure 5.10. All images in Figure 5.9 and Figure 5.10 are displayed using the kittiviewer from SECOND, which automatically includes ground truth annotations. Therefore, the pedestrian in Figure 5.10a is displayed with a green bounding box, which is not a part of the original LIDAR scan.



(a)



(b)

Figure 5.3: Sample images of 2D (a) and 3D (b) bounding boxes generated by CADET

Figure 5.4: Vertex occlusion detection. Occluded vertices are shown in red, while visible vertices are shown in green.

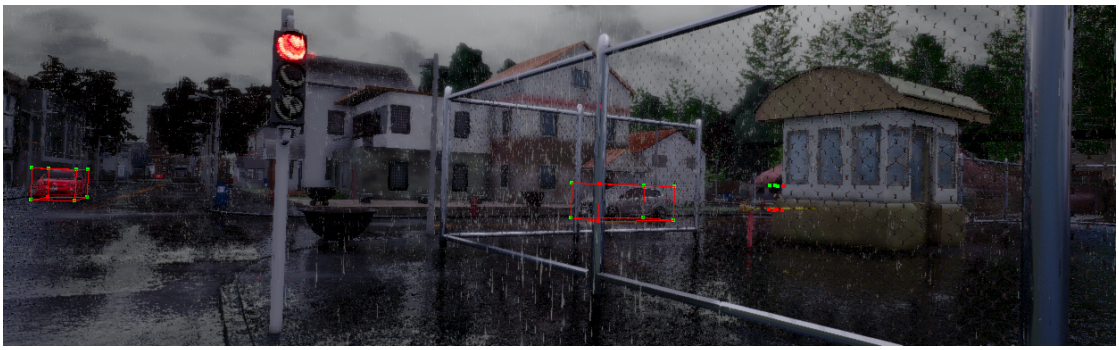


Figure 5.5: Vertex occlusion detection on see through objects. Occluded vertices are shown in red, while visible vertices are shown in green.

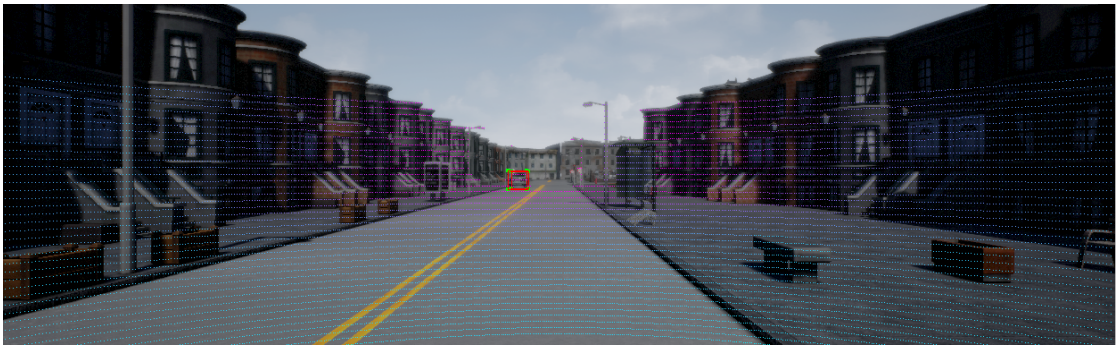


Figure 5.6: Lidar to camera projection. The color of each point in the point cloud is based on its depth value

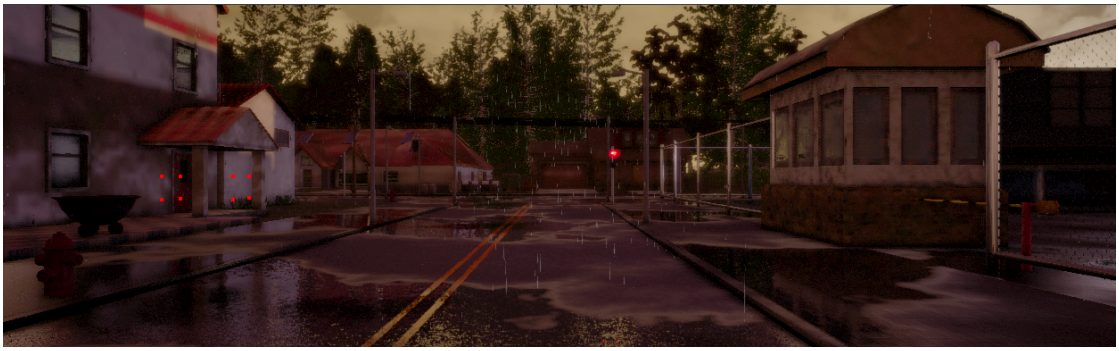
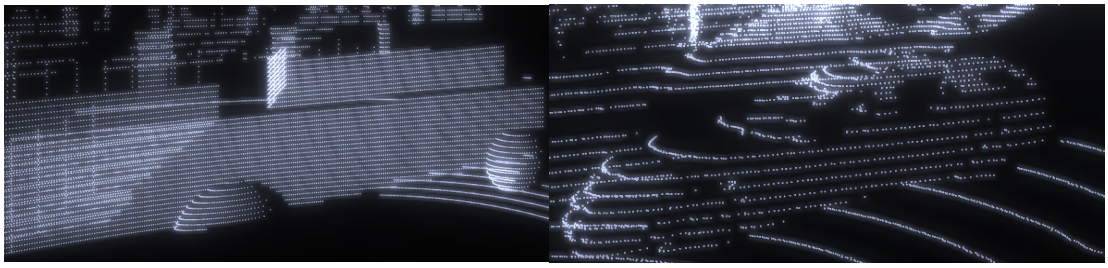


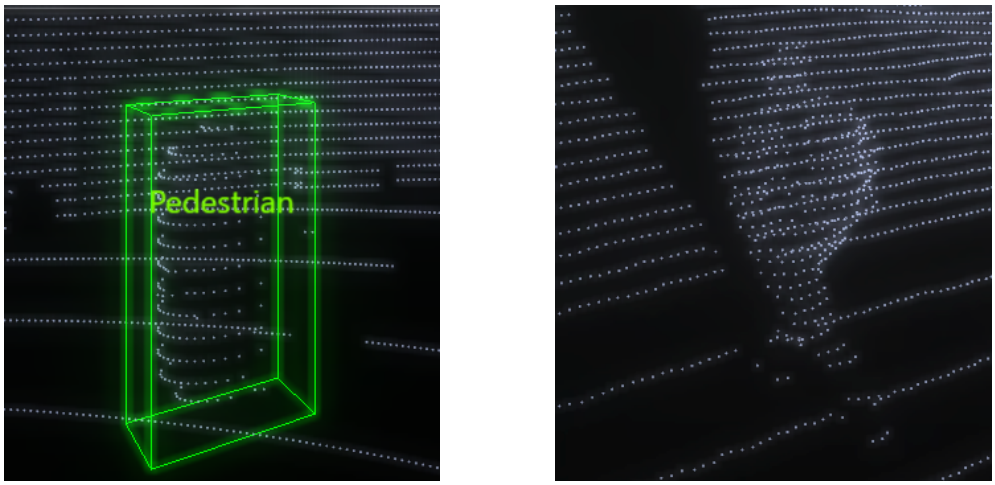
Figure 5.7: Vehicle fully occluded behind solid wall.

Figure 5.8: Vehicle inside and outside maximum label range.



(a) (b)

Figure 5.9: LIDAR scan of a Car from (a) CARLA and (b) real-world LIDAR sensor.



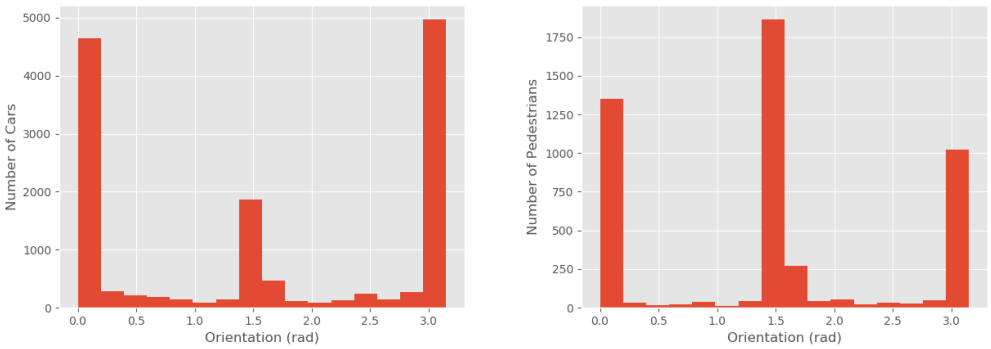
(a) (b)

Figure 5.10: LIDAR scan of a Pedestrian from (a) CARLA and (b) real-world LIDAR sensor.

Generated dataset

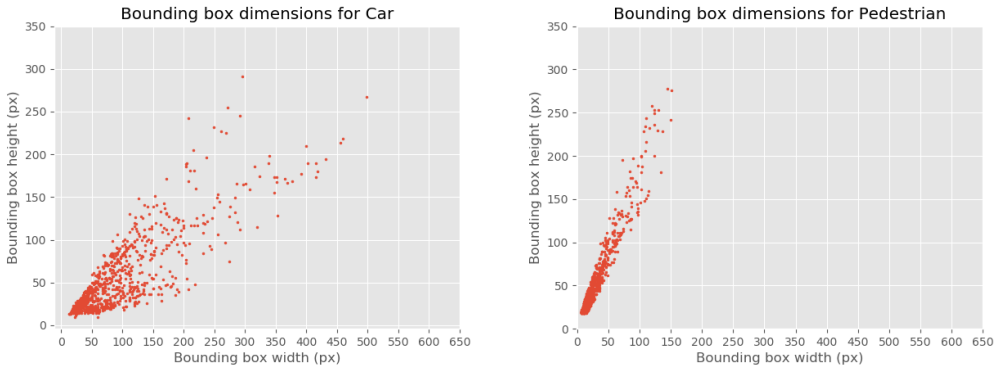
Using CADET, we generate the CADET dataset, consisting of 10,000 samples. In total, the dataset contains 13,989 cars and 4,895 pedestrians, with an average of 1.9 labeled objects per image. The dataset contain images with one or more labeled objects in the scene of either class, and the distribution of number of annotated objects per image is shown in Figure 5.14. An important aspect of the labeled objects in the dataset is their bounding box dimensions. This metric may indicate the difficulty of the object detection task, since smaller objects are generally harder to detect. In addition, if the distribution of bounding boxes is dense, then one would assume that a model will have an easier time learning that distribution. The bounding box dimensions are highlighted in Figure 5.12, and clearly shows that the pedestrians in the dataset generally have smaller labeled bounding boxes in pixels. When performing 3D object detection, one is often interested in not only predicting a 3D bounding box, but also the direction or heading of the bounding box. Without this heading the 3D bounding box is ambiguously defined. Therefore, it is valuable to investigate the distribution of orientations in the dataset, shown in Figure 5.11.

Another important feature of the generated dataset is how it compares to other datasets for autonomous driving with regards to class distribution and the number of labels per image. Therefore, we have also included the same statistics for the KITTI dataset, which is displayed in Figure 5.13 and Figure 5.15.



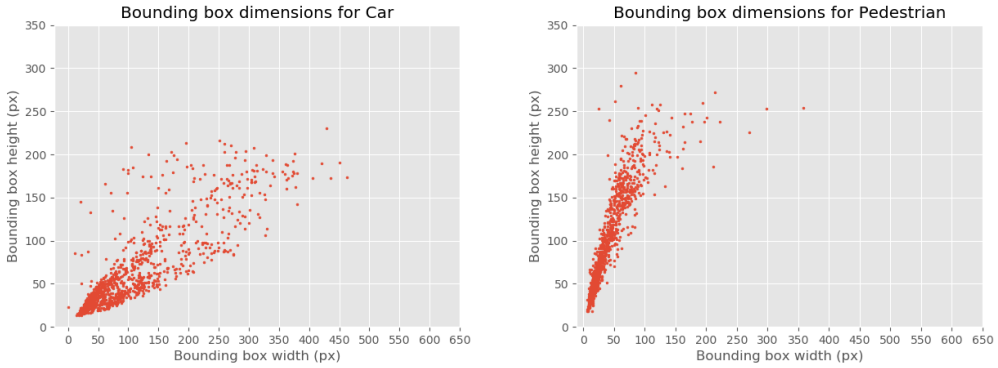
(a) (b)

Figure 5.11: Orientation distribution for (a) Cars and (b) Pedestrians in the CADET dataset.



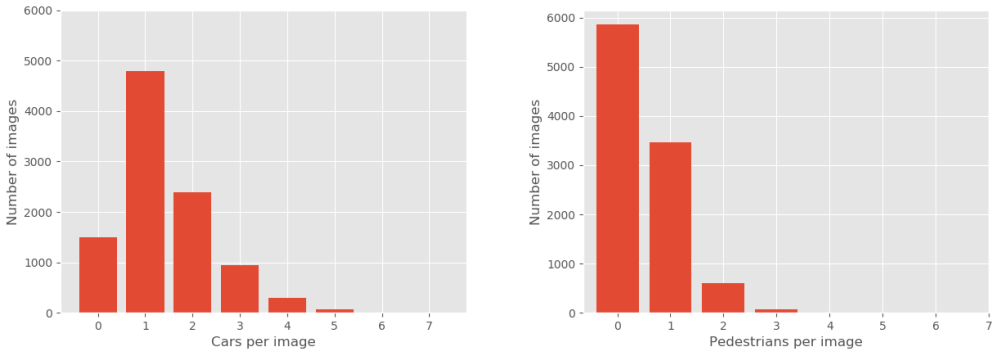
(a) (b)

Figure 5.12: Bounding box dimensions in pixels for (a) Cars and (b) Pedestrians in the CADET dataset.



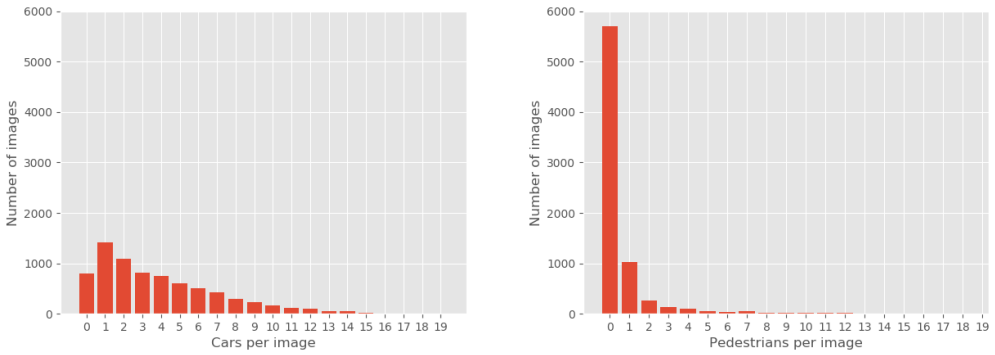
(a) (b)

Figure 5.13: Bounding box dimensions in pixels for (a) Cars and (b) Pedestrians in the KITTI dataset.



(a) (b)

Figure 5.14: Number of annotations per image for (a) Cars and (b) Pedestrians in the CADET dataset.



(a) (b)

Figure 5.15: Number of annotations per image for (a) Cars and (b) Pedestrians in the KITTI dataset.

5.3.2 Model training and BEV experiments

This section presents the results from all experiments using the BEV feature map configurations discussed in Section 4.2. In addition, comparative results using the largely unmodified SECOND architecture are presented in most tables. All evaluations shown are presented as 3D average precision and BEV average precision using a minimum IoU between prediction and ground truth of 0.7 for cars and 0.5 for pedestrians as the official KITTI leaderboard demands. As modifications to architecture overall are relatively minor, results on the KITTI dataset are gathered using the validation set and not submitted to the KITTI evaluation server for evaluation on the KITTI test set. This is due to KITTI policies and means baseline results might differ from those presented by Ku et al. [2018] and Yan et al. [2018]. We further note that whenever a model is referred to as pre-trained or fine-tuned, this always means pre-trained on CADET and fine-tuned on KITTI. To make it easier to understand the tables and compare results between configurations, we highlight both the best AVOD model and the best overall model in bold. In cases where SECOND performs best this results in two highlights per column. The full set of experiments serve as a collection to both evaluate the performance of the different BEV configurations as well as to assess the usability of the CADET dataset, in terms of generalization, increase in convergence speed and overall performance, for pre-training models to be fine-tuned and used in a real environment. In order to keep performance fluctuations when evaluating multiple checkpoints at a minimum, most experiments are done using single-class configurations. However, as it is of interest for real world applications due to better combined inference speeds, additional testing was performed using a selection of the model configurations as multi-class variants at the end of this section. Due to the large number of presented results, we refrain from commenting on individual results in this chapter and instead simply present them collectively. We advise the reader to first get an overall look at the results and then refer back to the tables and figures in this section when they are discussed in Chapter 6.

Single-class results

Tables 5.1 and 5.2 show single-class model results for models trained and evaluated on the KITTI dataset. Models are trained and evaluated separately for the Car and Pedestrian class, with measured inference run-time in milliseconds per sample. Results from training and validation on the CADET dataset are shown in Tables 5.3 and 5.4 for a selection of the AVOD BEV configurations, as well as for SECOND. Note that these tables do not segment into easy, moderate and hard categories, but instead simply uses the definitions large and small. Per the KITTI definition occlusion and maximum truncation, which are not currently included in the CADET labels, increases from easy to moderate to hard. In addition, KITTI limits easy objects to a minimum bounding box height of 40 pixels, while moderate and hard reduces this requirement to 25 pixels. The large and small definitions follow the same respective height requirements. Tables 5.5 and 5.6 show results from the same set of models, trained on the CADET dataset and evaluated on the KITTI dataset. Results after further fine-tuning the selected AVOD models on the KITTI training set and evaluating on the KITTI validation set are presented in Tables 5.7 and 5.8. Also note that these tables do not include fine-tuning on the SECOND models as these were omitted in light of multi-class results and due to time constraints. The multi-class section includes simulated pre-training and fine-tuning of both AVOD and SECOND models. We also include results using a limited KITTI training set to evaluate whether our simulated dataset can reduce the required amount of real data in Table 5.9. This table includes results from AVOD models of the default configuration after training on the first 500 samples of the full KITTI training set, with and without pre-training on CADET. As an additional experiment, the default AVOD configuration for cars was trained for 60k steps, with checkpoints for the entire training period stored and evaluated at every 2k steps. This was compared against the results from fine-tuning the CADET pre-trained model and compiled into the graphs in Figures 5.16-5.18 to evaluate convergence speed.

Table 5.1: KITTI-trained models, evaluated on cars in the KITTI dataset

		$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	Run-time (ms)	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	119	**83.46**	73.94	67.81	89.37	86.44	78.64
3M3D	120	83.16	**73.97**	**67.98**	**89.84**	**86.62**	**79.85**
MMD	**114**	82.98	73.92	67.84	89.62	86.61	79.68
MM4D	119	82.40	73.07	67.42	89.38	86.16	79.44
3M3DND	120	83.45	73.65	67.88	89.30	86.20	79.47
Cluster	**114**[4]	82.94	73.49	67.19	89.72	86.46	79.36
SECOND	50	**88.11**	**77.92**	**76.01**	**89.91**	**87.55**	79.38

Table 5.2: KITTI-trained models, evaluated on pedestrians in the KITTI dataset

		$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	Run-time (ms)	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	122	41.05	37.00	32.00	44.12	39.54	38.11
3M3D	122	45.61	42.66	38.06	49.16	45.99	**44.53**
MMD	**117**	27.85	27.17	24.54	33.39	33.10	29.78
MM4D	123	40.43	37.81	36.04	44.89	44.43	39.45
3M3DND	122	**46.46**	**43.70**	**38.10**	**52.27**	**46.28**	40.59
Cluster	**117**	33.74	28.95	27.40	40.43	35.43	30.57
SECOND	**49**	34.35	28.23	25.47	38.71	33.55	29.37

Table 5.3: CADET-trained models, evaluated on cars in the CADET dataset

	$AP_{3D}(\%)$		$AP_{BEV}(\%)$	
Method	**Large**	**Small**	**Large**	**Small**
Default	70.86	69.37	**80.13**	**71.32**
3M3D	**70.96**	**69.59**	79.81	71.28
MMD	68.79	60.87	78.75	70.72
3M3DND	70.68	68.69	79.49	71.20
SECOND	**77.59**	68.06	**87.23**	**78.44**

[4]Current preprocessing implementation not multi-threaded, feed dict run-time higher than inference at 132ms

Table 5.4: CADET-trained models, evaluated on pedestrians in the CADET dataset

	$AP_{3D}(\%)$		$AP_{BEV}(\%)$	
Method	**Large**	**Small**	**Large**	**Small**
Default	75.43	**73.89**	75.43	**73.91**
3M3D	**76.13**	72.99	**80.24**	73.41
MMD	75.49	71.82	79.73	72.37
3M3DND	75.87	73.71	75.87	73.77
SECOND	59.65	34.42	59.77	34.50

Table 5.5: CADET-trained models, evaluated on cars in the KITTI dataset

	$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	29.85	20.29	18.40	50.58	37.81	30.77
3M3D	35.85	**29.40**	**24.99**	**57.32**	**49.63**	**43.25**
MMD	30.34	24.28	20.25	45.22	37.56	31.17
3M3DND	**36.22**	28.45	24.26	56.78	42.93	36.62
SECOND	10.97	6.93	5.91	26.72	19.71	16.64

Table 5.6: CADET-trained models, evaluated on pedestrians in the KITTI dataset

	$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	**9.09**	**9.09**	**9.09**	9.38	**9.33**	**9.38**
3M3D	2.27	2.27	2.27	2.27	2.27	2.27
MMD	**9.09**	**9.09**	**9.09**	**9.78**	9.09	9.09
3M3DND	1.82	0.83	0.83	1.82	0.83	0.83
SECOND	2.27	2.27	2.27	2.27	2.27	2.27

Table 5.7: CADET-trained models, fine-tuned on KITTI and evaluated on cars in the KITTI dataset

	$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	**82.97**	**73.63**	**67.31**	**89.18**	79.36	78.71
3M3D	81.09	72.37	66.17	88.47	**85.38**	78.88
MMD	81.00	66.95	65.88	88.76	79.36	78.41
3M3DND	81.04	72.33	67.00	88.96	85.11	**79.11**

Table 5.8: CADET-trained models, fine-tuned on KITTI and evaluated on pedestrians in the KITTI dataset

	$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	40.26	38.55	33.93	46.96	44.80	40.73
3M3D	39.19	38.02	34.15	46.14	43.54	40.44
MMD	37.32	34.34	32.60	45.71	42.43	37.62
3M3DND	**46.93**	**43.00**	**40.10**	**50.90**	**48.48**	**43.61**

Table 5.9: AVOD performance comparison on the KITTI validation set after training using a KITTI training set of 500 samples

	$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Car	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Without pre-training	72.12	62.68	56.10	87.06	**77.72**	69.75
With pre-training	**73.73**	**64.47**	**63.09**	87.34	77.39	**76.15**
Pedestrian	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Without pre-training	19.41	15.06	15.02	20.67	15.60	15.64
With pre-training	**30.92**	**31.40**	**28.71**	**38.54**	**36.81**	**35.82**

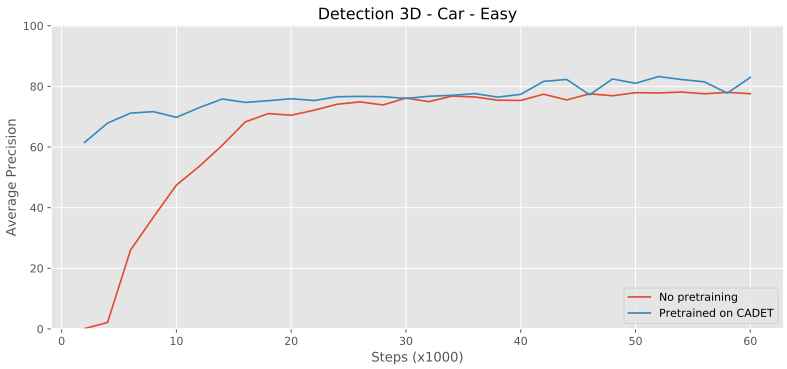


Figure 5.16: Convergence of default AVOD configuration on cars in the KITTI validation set, with and without pre-training. Easy difficulty.

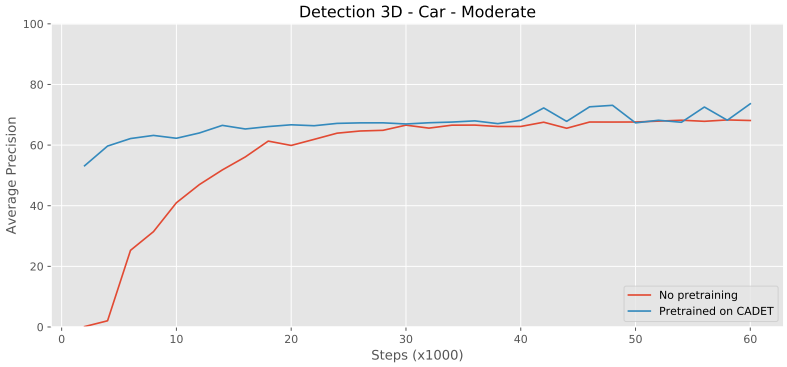


Figure 5.17: Convergence of default AVOD configuration on cars in the KITTI validation set, with and without pre-training. Moderate difficulty.

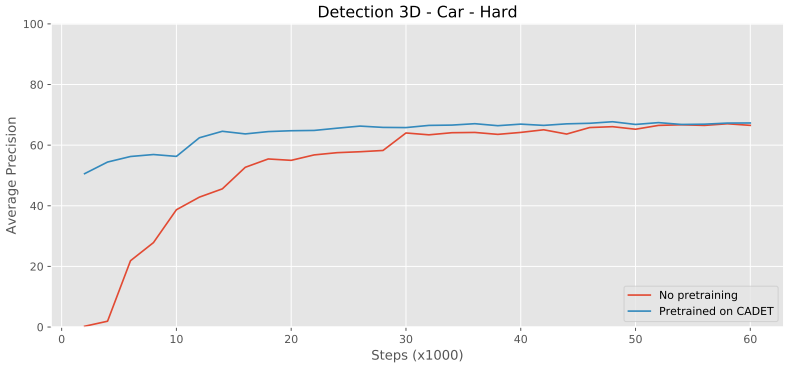


Figure 5.18: Convergence of default AVOD configuration on cars in the KITTI validation set, with and without pre-training. Hard difficulty.

Multi-class results

Single-class object detection serves well to augment differences in architectures, with less susceptibility to irregular behaviour during training. However, when used in an autonomous system with real-time requirements, there is considerable overhead to running multiple single-class models simultaneously. Therefore we also include results from experiments using multi-class configurations, tested for the prediction of cars and pedestrians. As with single-class configurations, models are either trained directly on KITTI or pre-trained first on CADET. For brevity, results such as training and evaluating on CADET, training using a limited KITTI dataset and measuring of convergence speed are omitted in multi-class. Table 5.10 shows results on the Car and Pedestrian class in the KITTI validation set for multi-class models trained on the KITTI training set. As AVOD-FPN tends to produce fluctuating results per class, the best performing model checkpoint is included for each class, noted with how many steps the model was trained to produce those results. If the best performing model in each class happens to be the same, or if the differences are negligible, only one model checkpoint is selected. Table 5.11 show results after pre-training on the CADET dataset and fine-tuning on the KITTI dataset, as specified in Sections 5.2.2 and 5.2.2. We note that as SECOND does not appear to maintain a decreased learning-rate going into fine-tuning, an additional configuration was tested that reduces the learning rate by a factor of ten before resuming training. The total number of training steps, including pre-training, is also specified with model checkpoint selection following the same criteria as before.

Table 5.10: Multi-class results on the KITTI validation set after training on the KITTI dataset

		$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Car	**Steps**	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	106k	**82.29**	67.74	66.81	89.09	85.99	79.18
Default	120k	76.22	66.79	65.95	89.03	79.50	79.07
3M3D	104k	81.52	**72.18**	66.63	88.59	85.68	79.08
3M3D	118k	77.20	67.34	66.57	88.88	79.52	78.99
MMD	106k	82.21	67.95	66.66	**89.33**	**86.34**	**79.44**
MMD	120k	76.39	66.85	65.86	88.24	79.55	79.08
3M3DND	116k	76.36	67.68	**66.96**	89.27	86.04	79.36
SECOND	62k	**87.28**	**77.22**	**75.40**	89.26	**86.53**	79.19
Pedestrian	**Steps**	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	106k	45.30	43.79	38.49	53.76	48.42	46.37
Default	120k	51.35	45.83	40.27	55.53	52.95	46.52
3M3D	104k	43.40	43.75	38.26	52.83	48.98	46.92
3M3D	118k	**51.44**	**47.98**	**42.36**	**55.91**	**55.48**	**49.38**
MMD	106k	30.33	30.80	27.21	40.04	39.56	35.39
MMD	120k	39.58	39.27	35.06	48.33	47.80	42.78
3M3DND	116k	47.12	43.60	41.63	53.46	49.54	44.26
SECOND	62k	**57.23**	**51.02**	**48.20**	**60.47**	**57.11**	**51.25**

Table 5.11: Multi-class results on the KITTI validation set after pre-training on the CADET dataset and fine-tuning on the KITTI dataset

		$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Car	**Steps**	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	146k	78.31	68.13	66.45	89.59	85.99	79.15
Default	148k	**82.97**	**72.79**	**67.22**	**89.63**	85.43	**79.34**
3M3D	146k	76.99	66.74	65.42	88.52	79.38	78.82
MMD	136k	74.49	66.02	64.34	88.63	85.00	78.66
3M3DND	142k	77.37	67.61	66.25	89.07	85.45	79.08
3M3DND	146k	81.94	72.55	**67.22**	89.17	**85.51**	79.29
SECOND (lr=3e-3)	62k	**87.64**	**76.45**	**68.79**	**90.08**	**86.48**	**79.22**
SECOND (lr=3e-4)	72k	84.75	72.94	66.31	89.77	85.59	78.47
Pedestrian	**Steps**	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	146k	**51.86**	**46.94**	**41.41**	**56.26**	50.99	**48.80**
Default	148k	46.23	43.85	38.35	53.39	**52.77**	46.87
3M3D	146k	48.38	45.99	40.64	55.80	49.79	47.82
MMD	136k	30.80	27.43	26.49	42.60	37.40	32.31
3M3DND	142k	40.88	36.01	31.45	48.51	43.56	38.38
3M3DND	146k	36.91	32.97	29.03	47.89	42.52	37.52
SECOND (lr=3e-3)	62k	**53.57**	**47.61**	**41.90**	**59.10**	**54.74**	**48.97**
SECOND (lr=3e-4)	72k	51.15	44.06	40.49	57.40	49.33	44.01

Table 5.12: Multi-class results on the CADET validation set after training on the KITTI dataset

		$AP_{3D}(\%)$		$AP_{BEV}(\%)$	
Car	**Steps**	**Large**	**Small**	**Large**	**Small**
Default	106k	**12.30**	**11.69**	25.98	24.16
3M3D	104k	9.14	8.06	23.54	21.36
MMD	106k	11.72	11.02	**31.58**	**25.81**
3M3DND	116k	9.34	6.95	26.03	24.68
SECOND	62k	0.43	0.06	0.45	9.09
Pedestrian	**Steps**	**Large**	**Small**	**Large**	**Small**
Default	106k	**9.09**	**9.09**	**15.04**	**11.75**
3M3D	104k	**9.09**	**9.09**	14.30	11.23
MMD	106k	**9.09**	**9.09**	11.02	9.09
3M3DND	116k	2.045	1.63	6.75	3.77
SECOND	62k	0.00	0.00	0.00	0.00

Table 5.13: Multi-class results on the CADET validation set after pre-training on the CADET dataset and fine-tuning on the KITTI dataset

		$AP_{3D}(\%)$		$AP_{BEV}(\%)$	
Car	**Steps**	**Large**	**Small**	**Large**	**Small**
Default	146k	23.28	21.23	**51.66**	**43.28**
3M3D	146k	**25.49**	**22.10**	49.93	42.46
MMD	136k	17.40	17.37	50.03	42.70
3M3DND	142k	20.39	17.26	50.08	41.77
SECOND (lr=3e-3)	62k	9.76	6.46	13.29	11.84
SECOND (lr=3e-4)	72k	**26.30**	19.27	38.59	32.07
Pedestrian	**Steps**	**Large**	**Small**	**Large**	**Small**
Default	146k	15.53	8.65	38.46	24.99
3M3D	146k	**35.16**	**22.16**	**47.45**	**31.98**
MMD	136k	12.42	11.95	18.21	13.14
3M3DND	142k	20.78	14.32	31.78	23.57
SECOND (lr=3e-3)	62k	4.55	4.55	14.39	12.11
SECOND (lr=3e-4)	72k	5.61	4.55	26.42	22.75

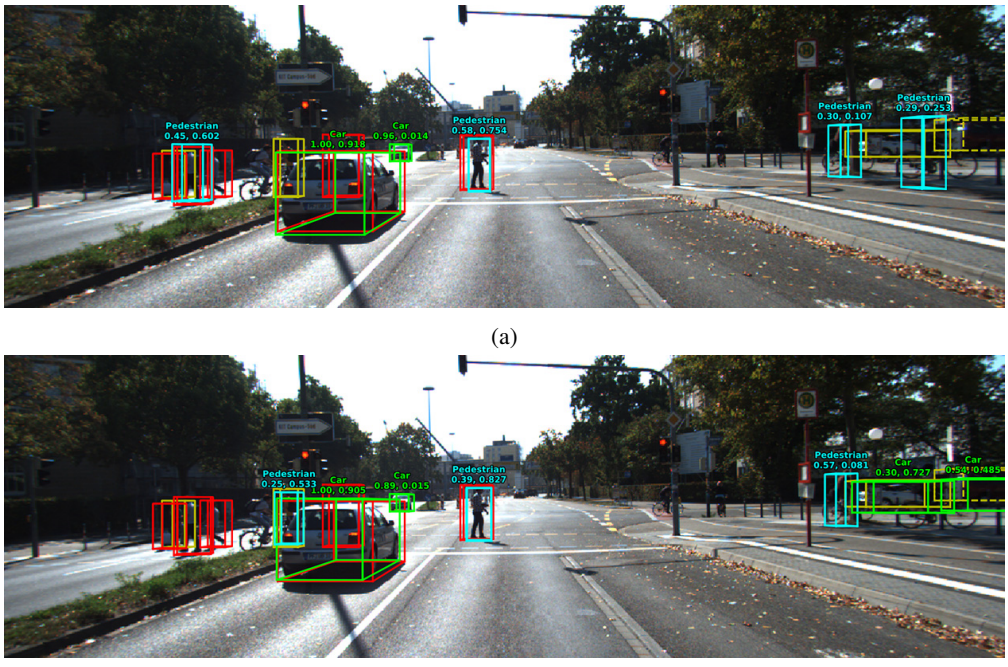
5.3.3 Inference examples on KITTI and CADET

This section presents a small collection of images resulting from inference on the KITTI and CADET dataset in order to visually evaluate model performance, rather than just considering average performance. This potentially helps highlight wider generalization after pre-training on the simulated CADET dataset and fine-tuning on the KITTI dataset. Additionally it helps illustrate the scenarios that may favor performance for models trained using either method. Two variants of the AVOD-FPN multi-class default configuration, one with and without pre-training, was selected and used to perform inference on a subset of the validation set in both KITTI and CADET. Figures 5.19-5.20 and 5.21-5.22 show a selection of images resulting from inference on the KITTI and CADET datasets respectively, with a few additional results in Appendix D. All images are produced using a minimum confidence threshold of 0.25. Ground truth bounding boxes are drawn in red, yellow or stippled yellow lines depending on difficulty, while predictions are drawn in green for cars and blue for pedestrians. For brevity, and as we intend for this section to focus on the usability for simulated pre-training of multimodal systems, only inference results from AVOD-FPN are included.

(a)

(b)

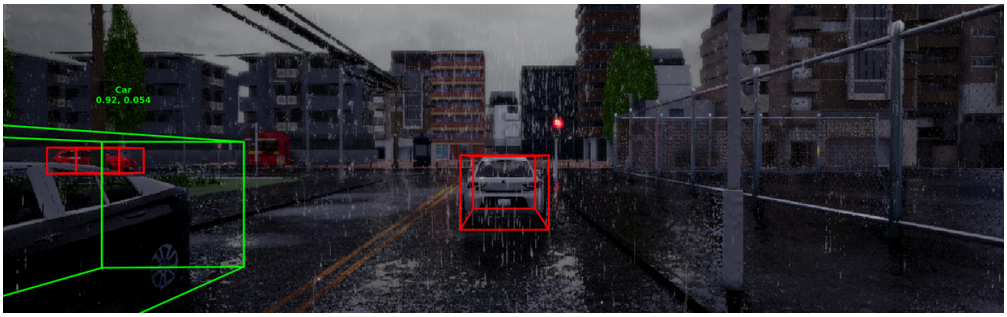
Figure 5.19: AVOD inference on KITTI sample 000015 without (a) and with (b) pre-training on CADET.



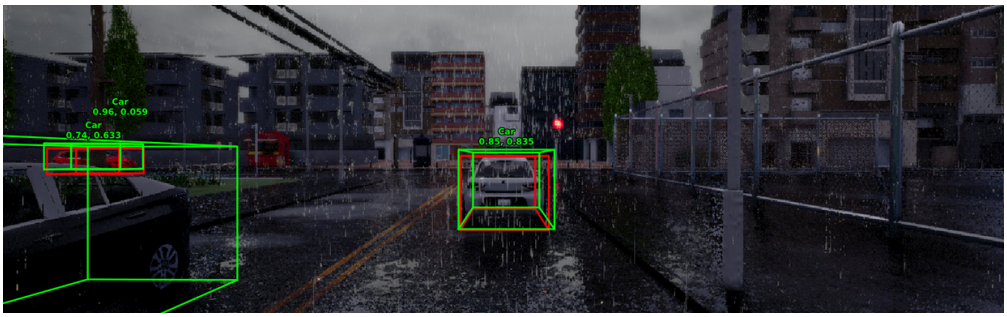
(a)

(b)

Figure 5.20: AVOD inference on KITTI sample 134 without (a) and with (b) pre-training on CADET.



(a)



(b)

Figure 5.21: AVOD inference on CADET sample 8025 without (a) and with (b) pre-training on CADET.

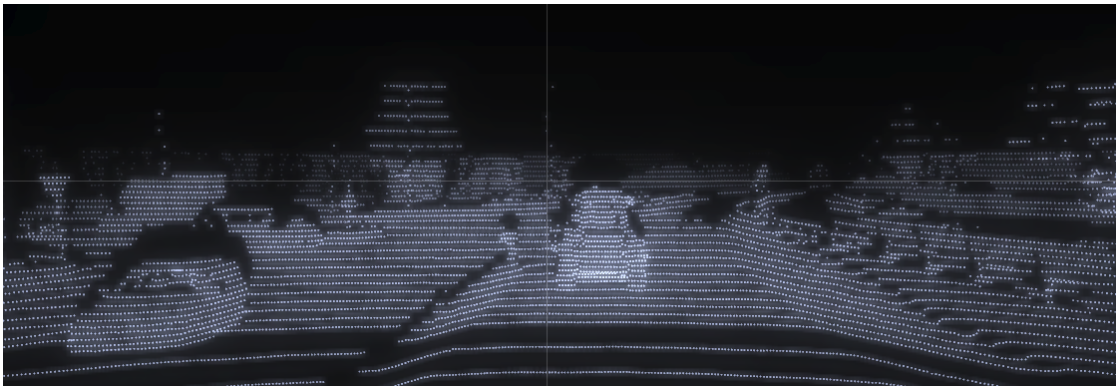
(a)

(b)

Figure 5.22: AVOD inference on CADET sample 8064 without (a) and with (b) pre-training on CADET

5.3.4 NAP dataset

The data gathered from the Kia recording vehicle is stored in ROS bags, which are then looped through and converted to the KITTI format to ensure compatibility with the object detection architectures. Since the dataset does not contain any labels at the time of this writing, it is only used for inference for previously trained models. An example of a scenario from the dataset is displayed in Figure 5.23, which displays the LIDAR scan and the corresponding camera image of the given scene. Note that the LIDAR displayed here is cropped to approximately match the camera view, since the full LIDAR scan is 360 degrees. The data in the NAP dataset is gathered from a single recording while driving in Trondheim, and only contains recordings from sunny conditions. The dataset contains images with and without lens flare, and includes roads ranging from uncovered to fully covered in snow. The images in the dataset contains unlabeled scenes featuring pedestrians, cars, trucks, cyclists and construction machines to name a few. Note that each sample image displayed from the NAP dataset in this project contains a sample ID, which is the sequence number from the dataset. Even though the NAP dataset is yet not released due to privacy concerns, we hope that it will be in the near future. We therefore choose to include this sequence number when referring to samples from the dataset.



(a)

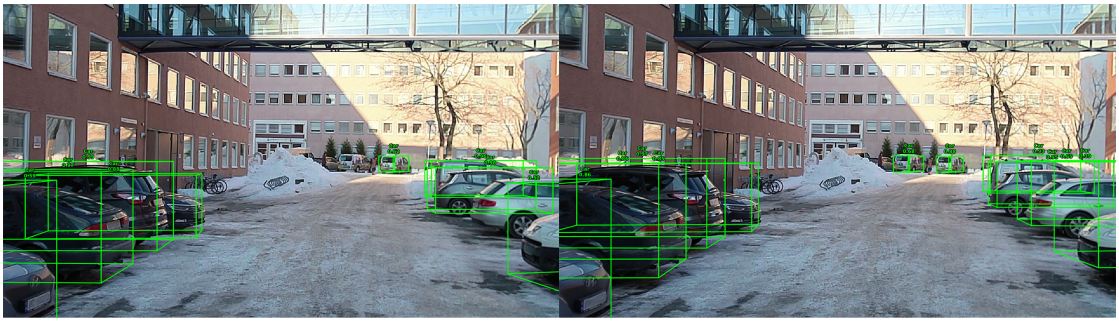
(b)

Figure 5.23: Visualization of (a) LIDAR and (b) Camera data of sample 1887 from the NAP dataset

5.3.5 Inference examples on NAP dataset

In addition to evaluating our models on CADET and KITTI in section 5.3.3, a visual evaluation was performed on the real-world data gathered by NAP. As the results on the unseen dataset gives a stronger indication of actual generalization, we have chosen to include results from both AVOD-FPN and SECOND models featuring full training on KITTI, pre-training on CADET with fine-tuning on KITTI and full training on CADET. Figures 5.24-5.27 show inference examples using AVOD-FPN models of the default configuration trained to achieve similar accuracy on cars and pedestrians when evaluated on the KITTI dataset. We additionally display inference examples using an AVOD-FPN model of the same configuration, but trained solely on CADET in Figures 5.28-5.29. Figures 5.30-5.33 show inference from SECOND on the same samples using the best performing model trained only on KITTI, as well as the model pre-trained on CADET and then fine-tuned on KITTI with a manually reduced learning rate. Lastly, we include inference results using a SECOND model trained only on CADET in Figures 5.34-5.35.

The minimum confidence threshold for detection using AVOD-FPN was set to 0.25 as before, while a setting of 0.3 was used for SECOND. The images displayed here are shown to illustrate certain aspects about the models performance, and will be discussed in more detail in the next chapter. For brevity, only a handful carefully selected images are displayed, with more inference images shown in Appendix E. Note that since the NAP dataset contains sensitive information such as people or license plates, the images have been processed to remove these. This process was performed after model inference in order to preserve image features for detection. Green bounding boxes denotes detection of cars, while blue boxes denotes pedestrians.



(a) (b)

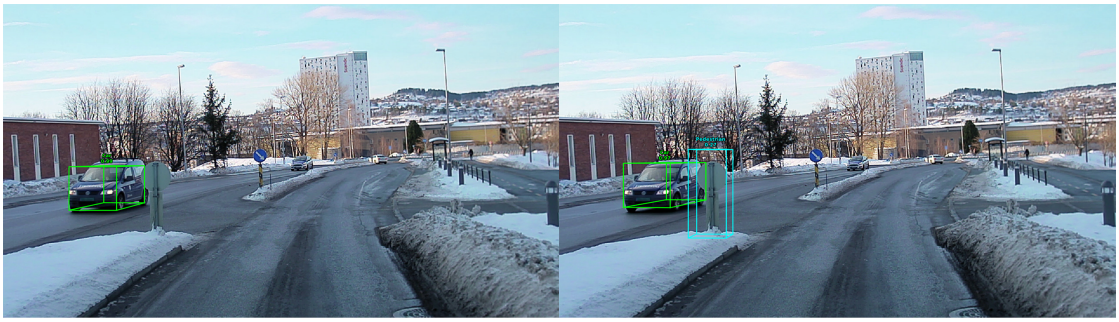
Figure 5.24: AVOD inference on NAP sample 140 without (a) and with (b) pre-training on CADET.

(a) (b)

Figure 5.25: AVOD inference on NAP sample 1020 without (a) and with (b) pre-training on CADET.

(a) (b)

Figure 5.26: AVOD inference on NAP sample 3080 without (a) and with (b) pre-training on CADET.

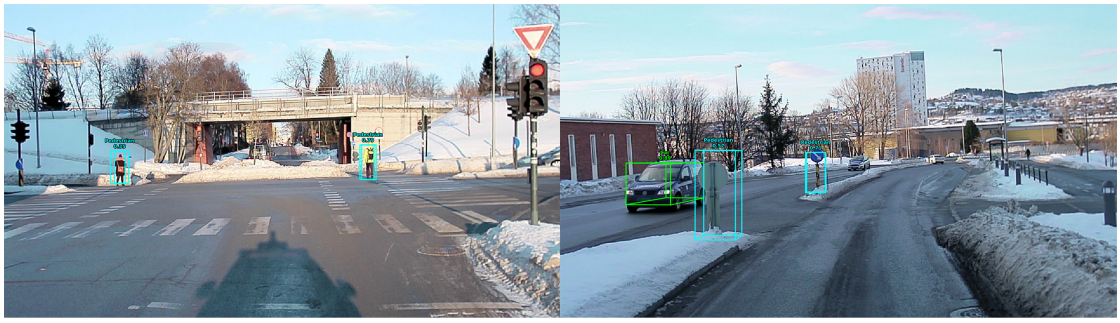


(a) (b)

Figure 5.27: AVOD inference on NAP sample 4180 without (a) and with (b) pre-training on CADET.

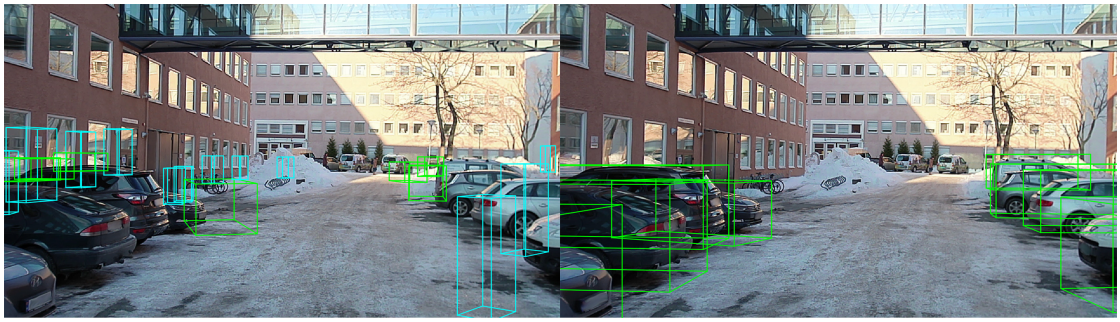
(a) (b)

Figure 5.28: AVOD inference on NAP sample (a) 140 and (b) 1020 trained solely on CADET.



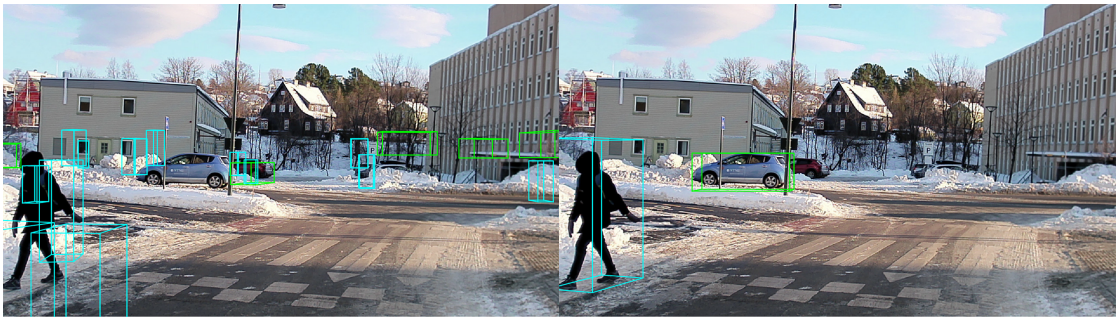
(a) (b)

Figure 5.29: AVOD inference on NAP sample (a) 3080 and (b) 4180 trained solely on CADET.



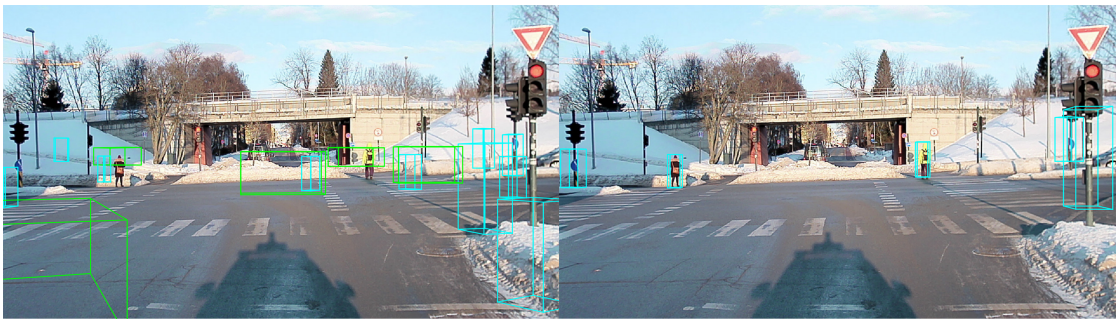
(a) (b)

Figure 5.30: SECOND inference on NAP sample 140 without (a) and with (b) pre-training on CADET.



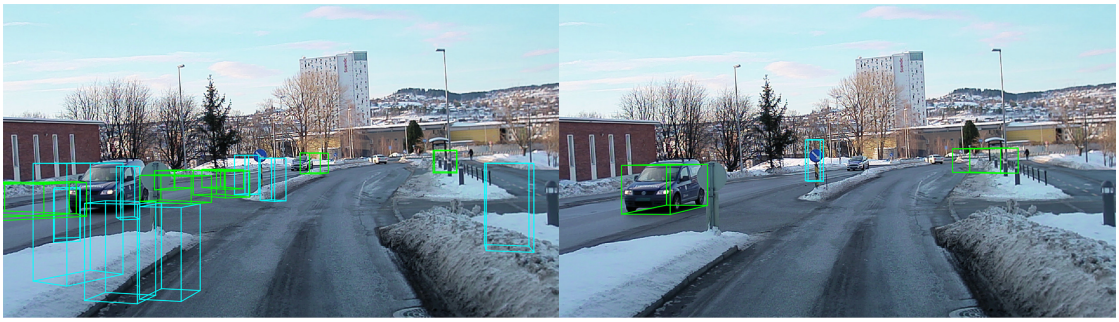
(a) (b)

Figure 5.31: SECOND inference on NAP sample 1020 without (a) and with (b) pre-training on CADET.



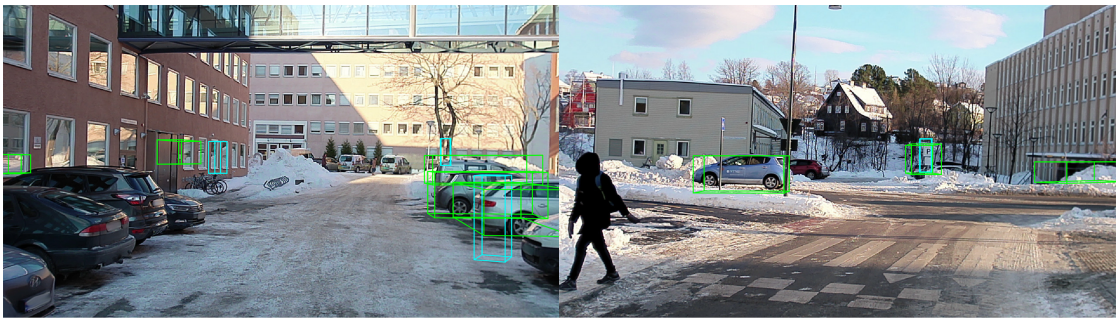
(a) (b)

Figure 5.32: SECOND inference on NAP sample 3080 without (a) and with (b) pre-training on CADET.



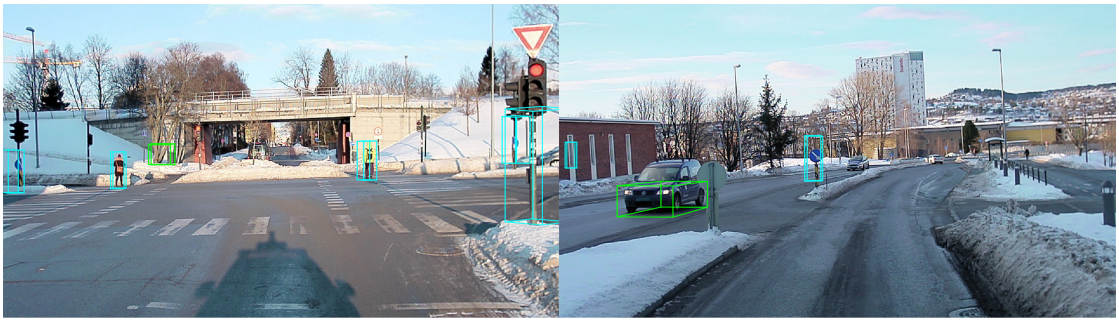
(a) (b)

Figure 5.33: SECOND inference on NAP sample 4180 without (a) and with (b) pre-training on CADET.



(a) (b)

Figure 5.34: SECOND inference on NAP sample (a) 140 and (b) 1020 trained solely on CADET.



(a) (b)

Figure 5.35: SECOND inference on NAP sample (a) 3080 and (b) 4180 trained solely on CADET.

Chapter 6

Discussion

An important part of any research is to critically evaluate the results, and present not only the positive aspects of one's results, but also its weaknesses and possible improvements. This chapter discusses the results presented in the previous chapter and comments on their merits. The discussion will follow the same structure as Chapter 5, where we in each section first discuss the CADET tool, then the CADET dataset, the evaluation of the trained models and BEV experiments, and lastly the model inference on the NAP dataset.

6.1 CADET tool

Evaluating the data generated by CADET, we observe that the tool is able to generate sensor data that produces a consistent alignment between sensors. In other words, there is a good correspondence between points generated by the LIDAR sensor and by the camera, as depicted in Figure 5.6. This correspondence is independent of the pitch and roll of the vehicle, due to the normalization step performed in Section 5.2.1. By close inspection, we observe that the lampposts in Figure 5.6 contains projected LIDAR points. These lampposts are only a couple of pixels wide in the image, which means that the projection is fairly accurate. If not, these LIDAR points would lie outside the bounds of the object. This correspondence extends to the projected bounding boxes of each object, as seen in both 2D and 3D in Figure 5.3. However, the fit of the bounding box of the frontmost vehicle in the scene and pedestrian seems a bit too loose. In particular, the front facing vertices of the blue vehicle in Figure 5.3b appears to extend a little too wide. The same can be said for the pedestrian in the same figure, where the height of the bounding box seems correct, but the horizontal extent of the bounding box is too loose. Since these bounding boxes are calculated based on each objects vertices, which in turn is pre-built into CARLA, this issue has to be fixed by modifying the vehicle models in CARLA. However, the fit of bounding boxes on most vehicle models are tighter than the one depicted here, which serves as an illustration that some vehicle models have poor fit. It is unclear whether or not this will affect the quality of a machine learning system trained on this data. Since the bounding box fully contains the object, one would assume that the system would be able to select useful features contained in the bounding box, and thus be able to detect the object despite the loose bounding box. However, the dimensions of this bounding box might be inaccurate, as there exist an imprecise relation between an objects extent and its labeled bounding box.

As mentioned in the previous chapter, the CADET tool has to perform occlusion detection in order to determine which objects to label or exclude from labeling. This algorithm is quite important, since it will confuse machine learning models if it generates labels for vehicles that are fully occluded, such as vehicles

behind a wall. Similarly, the algorithm also needs to label objects that are visible in the scene, otherwise model training may produce unnecessary many false positives. The CADET occlusion detection applies a two-step approach to detecting occluded objects. First, it detects whether or not individual vertices are occluded, and then decides whether or not the object is occluded based on the number of occluded vertices. Since a vertex can be occluded by the object itself, there is always at least one vertex occluded for each object. Figure 5.4 illustrates the result of the algorithm in a regular driving scenario. Here, all vehicles are detected as visible, and thereby labeled. This is indicated by drawing the 3D bounding box in red. We observe that the algorithm has correctly identified all visible (green) and occluded vertices (red). For example, the white front-most vehicle in the scene is correctly identified as visible, and its back-left lower vertex is determined to be occluded. As is the case with the two red vehicles. Here we can see that the algorithm takes into account the occlusion by other objects, not just by the surrounding scene, as the vehicle furthest to the back has one of its front vertices occluded by the car in front of it. However, equally important to being able to decide occlusion by other objects is being able to determine occlusion with regards to static objects in the scene. Solid objects such as walls are handled elegantly by the algorithm, illustrated in Figure 5.7. As mentioned in the previous chapter, see-through objects such as fences or glass may be handled differently depending on how they are physically modelled. However, judging by Figure 5.5 it is clear that the occlusion detection works well, even for these types of objects. In particular, nearly all vertices of the gray car behind the fence are deemed visible, except for the two that are covered by the metal post of the fence. Notably, the pedestrian in the rightmost part of the image is deemed too occluded to be labeled, which is consistent with what we would expect if this image were to be labeled manually.

Another key aspect for determining visibility of objects in the scene is the maximum distance from the object to the camera, and the dimension of its bounding box projected in image space. As illustrated in Figure 5.8 the nearby vehicle has its bounding box vertices correctly drawn, while the small object in the distance does not have any of its vertices drawn (occluded or not), indicating that it is outside range. Distant objects are usually not critical to detect, as it is usually nearby objects that affect the decisions made by human drivers and autonomous systems.

A key aspect of evaluating a simulator is how well it reflects real world scenarios. With regards to sensors, this measurement can be thought of as ”how similar is the data generated by the sensor as compared to an actual sensor?”. For a camera, we can easily judge how ”real” the generated images look. However, this is significantly harder to do for other sensors such as LIDAR where we do not have a strict notion of realism. Figure 5.9 shows a comparison of the LIDAR reflection of a car in the CARLA simulator and one generated by the Hesai Pandora sensor. It is abundantly clear that the CARLA mesh representation of vehicles is very simplified, and does not accurately reflect how vehicles appear in a genuine LIDAR scan. The vehicle model in CARLA consists of two boxes for the main body and four ellipsoids for wheels, which is identical regardless of the car’s make. Therefore, a model trained solely on these LIDAR point clouds will likely generalize poorly to a genuine LIDAR scan, where the surfaces of vehicles are more streamlined than flat boxes. This is perhaps even more critical for pedestrians, as shown in 5.10, where the CARLA simulated LIDAR scan gives pedestrians a near barrel like shape without visible limbs. This might still work for pedestrians standing still, but likely also leads more difficulty in distinguishing other objects from pedestrians as well as for detecting pedestrians walking or running. The resulting performance degradation of machine learning models training on the simplistically simulated LIDAR mesh will however depend on how the point clouds are processed.

6.2 CADET dataset

When evaluating how challenging it is for a model to perform well on an object detection dataset with multiple classes there are some measurements one can use as an indicator for difficulty. When the models

are tasked with prediction both bounding box dimensions and orientation, the distribution of these parameters in the dataset play an important role, including factors such as smoothness and non-degeneracy[1] of the distribution [Shamir, 2018]. In addition, if the dataset has a large class imbalance, the models may perform poorly on the minority class. Compared to KITTI, the CADET dataset has a slightly higher class imbalance of cars and pedestrians. KITTI has approximately a 2:1 ratio between car and pedestrian class labels, compared to CADET's nearly 3:1. However, it should be noted that the cars and pedestrians are the two largest classes in KITTI, and that several other minority classes such as truck or trams occupy a very small proportion of the dataset. We also note that for generating a simulated dataset using tools like CADET, it is a fairly simple task to spawn more pedestrians when generating a future dataset in order to even out the class imbalance.

With regards to bounding box dimensions, the CADET dataset contains both large and small bounding boxes as shown in Figure 5.12. The bounding box dimensions for cars are generally larger than pedestrians, which is to be expected. However, this indicates that it will generally be more difficult to detect pedestrians than cars. Since objects that occupy a small portion of the screen may be very hard to detect, both for machine learning models and for humans at first glance, the minimum area of any bounding box is set to 100 pixels. The figures also indicate that most of the bounding boxes for both classes are quite clustered with just a few outliers, making it an easier prediciton task. Relative to the KITTI dataset, the CADET dataset is arguably a more challenging dataset for object detection. As shown in Figure 5.12 and Figure 5.13, bounding boxes are generally smaller in the CADET dataset, which means that the objects occupy a smaller region on the screen and are therefore more difficult to detect. In fact, the average 2D bounding box area in KITTI is 6,220 pixels for the Pedestrian class and 11,283 pixels for the Car class, whereas it is 2,219 and 6,899 pixels respectively in CADET. Thus, we would expect that a model trained on CADET would potentially perform better on smaller objects than a model trained on KITTI, if evaluated on the same dataset.

In addition to bounding box dimensions, the object detection architectures presented in this thesis also predict the orientation of each bounding box. The distribution of this target variable is shown for both Cars and Pedestrians in Figure 5.11. We observe that the distribution of orientations for both classes follow a multimodal distribution with peaks around 0, $\frac{\pi}{2}$, and π, which corresponds to objects facing towards, left or right, and away from the camera, respectively. As expected, most vehicles are either facing towards or away from the camera, which indicates that they are driving in the same or opposite direction. There is also a peak for objects facing right and left, which are usually from scenarios involving intersections. Interestingly, the orientation distribution of pedestrians (Figure 5.11b) show a larger amount of pedestrians with an orientation of approximately $\frac{\pi}{2}$, which corresponds to facing right or left. This indicates that most pedestrians are in the process of crossing the road or walking perpendicular to the vehicle direction when recorded. A reason for this might be because Pedestrians walking parallel to the direction of the camera are disregarded when far away, since the minimum bounding box area requirement is 100 pixels.

Another factor of vast significance is under which conditions each scenario is captured from. Here, conditions are meant to cover weather conditions, starting location and which town the scenario is sampled from. Each of these environmental factors are decided randomly for each reset of the environment, and therefore it is expected that they are approximately uniformly distributed in the dataset.

Summarising the evaluation of the CADET dataset, we observe that the dataset seems to be similar to KITTI in many respect, such as with regards to bounding box distributions and labels. However, the dataset generally contains smaller bounding boxes, which means that the detection task may be more difficult than for the KITTI dataset. The LIDAR representation of classes in CADET remains one of its greatest weaknesses, and should be one of the first targets for improvement along with a larger variation in environments.

[1] A multidimensional distribution is said to be degenerate if the support of the distribution is fully contained in a lower dimensional space. This essentially means that at least one of the variables is a function of some of the other variables.

6.3 Model performance

In this section the results gathered from training of single-class and multi-class models in Section 5.3.2 are evaluated. Most importantly this section includes discussion on apparent differences in AVOD BEV configurations, as well as their compared performance to the LIDAR based SECOND architecture, and highlights apparent strengths and weaknesses of the CADET dataset that arise when used to train 3D object detection models. The focus of evaluation will be on the 3D average precision as this produces more specific localiazation than in BEV, not assuming an object is on the ground plane, and is more directly correlated with the overall goal of 3D object detection. BEV results are however included for reference in all previous tables as BEV detections can be used to achieve similar behaviour in autonomous driving for all object classes that are always at ground level. Unusual results, both strong and weak, are given extra attention in order to attempt to explain the behaviour such that it may either be accredited it to reflect model performance or limitations of simulation, or determine whether the results are inconclusive.

6.3.1 Single-class evaluation

Studying the results in Tables 5.1 and 5.2 we see that all BEV configurations perform similarly on the Car class, although not measuring up to the SECOND architecture in either AP or run-time. We note however that the MMD configuration, using only half as many BEV feature maps consisting of a single layer of maximum heights, minimum heights and density, achieves about a 4-5% inference speedup at minimal loss of accuracy. Turning over to the results on the Pedestrian class however and differences become more notable. While the default AVOD configuration is a reasonably strong performer, outperforming SECOND likely due to the use of high resolution camera images, albeit at a much higher run-time, it falls considerably short of the 3M3D and 3MDND configurations. These configurations, each having 3 layers of maximum heights and densities (with and without distance normalization), seem to have a strong edge for the detection of smaller objects, likely due to mitigating the higher susceptibility to noise as objects take up less space in the point cloud. While sufficient for the detection of cars, the MMD configuration sees a severe reduction in accuracy on smaller classes, to the point where it should not be considered worthwhile for the reduction in run-time.

Results on the CADET dataset in Tables 5.3 and 5.4 are more mixed. The different AVOD configurations do for the most part perform within a reasonable margin of each other on the Car class, while SECOND again shows noticeably stronger performance for cars with large bounding boxes. Different from before however, is that the MMD configuration sees a marked reduction in performance on more distant cars given a smaller 3D bounding box. Likely, this might indicate that the CADET dataset includes labeling of distant cars that would be omitted when labeling the KITTI dataset due to the size, often resulting in more difficult predictions. For the Pedestrian class, AVOD configurations are surprisingly all performing similarly, with no clearly superior model. In this case even the MMD model significantly outperforms SECOND, although the previous KITTI results favored SECOND over MMD. One explanation of the difference in results may be accredited to the simplified physical models of dynamic objects visible in the generated point clouds. While also valid for cars, this is especially apparent when inspecting pedestrians, as they appear as blocky, cylindrical shapes without visible limbs.

When evaluating the results of the CADET-trained models on the KITTI dataset in Tables 5.5 and 5.6, the limitations of the simulated LIDAR becomes immediately apparent, as SECOND generalizes considerably worse to the unseen real data. For cars, the 3M3D configuration attains the best performance across all metrics, while the MMD configuration surprisingly also generalizes better than the default configuration. This could potentially mean that the default configuration is overfitted towards recognition of a simulated car representation that differs more from the real data. With regards to pedestrian detection, neither of the models show any strong signs of generalizing to the real data. The default and MMD configuration show strongest performance, hovering around 9% on all metrics, while the 3M3D config-

uration and SECOND barely exceed 2% AP. Interestingly the 3M3DND model, performing best on the KITTI dataset, generalizes the worst, with less than 2% AP on easy and less than 1% on moderate and hard. The poor results again does seem to indicate that the representation of pedestrians in the CARLA LIDAR relatively poorly resemble real data. The reason for identical AP among the easy, moderate and hard categories of multiple models is more unclear, but may in part be a result of the models favoring more difficult pedestrian detections, likely based on distance, suggesting the CADET dataset may be over-saturated with pedestrian labels a fair distance away from the origin. We note, although not presented in the tables as orientation was not a focus of evaluation, that the heading angle average precision for cars in BEV and 3D for the tested models was in the range 25-40% on the CADET dataset as opposed to 80-90% on the KITTI dataset. Whether this is a results of it being much more difficult to differentiate between the front and the back of a vehicle in the CADET dataset, or a potential error in labeling the rotation is not entirely certain, as either situations could cause tight-fitting bounding boxes with flipped orientation. Pedestrian heading AP on the CADET dataset is more in line with the results on the KITTI dataset, although still significantly lower at 25-32% AP as opposed to 35-40%.

After fine-tuning the AVOD-FPN models on the KITTI dataset, we see that the models pre-trained on CADET achieve largely similar results in Tables 5.7 and 5.8 when compared to the fully KITTI-trained models in Tables 5.1 and 5.2. Interestingly, the MMD configuration sees a noticeable drop in performance for detections of cars in the moderate category when compared to the other configurations and the fully KITTI-trained counterparts. The same behaviour was previously seen in the default and 3M3D configuration, but was improved after re-training from step 90,000. The default, 3M3D and 3M3DND configurations achieve close to regular results on all degrees of difficulty, however we note that also these exhibit some instability when examining multiple checkpoints with only one or two checkpoints having balanced performance. It is difficult to determine the results as favoring the individual strengths of the configurations with the random factors of the gradient instability in mind.

Results on the Pedestrian class, as without pre-training, favors the 3M3DND configuration which outperforms the other models by about 4-7% AP across the board. Results are again very similar to full KITTI training and actually slightly better on the hard category. The 3M3D configuration surprisingly performs significantly worse, with results very similar to the default configuration after fine-tuning. More interesting is the performance of the MMD configuration, actually exceeding the performance of the fully KITTI-trained variation by a considerable margin, and managing to do so after just 18,000 steps of training on real data as opposed to the 116,000 steps previously. Perhaps the simplified BEV representation helps prevent overfitting on the simple physical representation of pedestrians in the simulated LIDAR, as the common features in BEV fall more in line with the real data. The other configurations feature higher fidelity feature maps that more faithfully represent the actual objects, but in return might be overfitting to this specific representation, which in CARLA is neither especially accurate or varied. We expect that improved sensor modelling can greatly improve upon the gathered results, and would show an even stronger preference for the more detailed feature maps.

Table 5.9, showing performance on a limited training set with and without simulated pre-training, produces quite interesting results. On cars, models performs fairly similarly to each other, especially for the easy and moderate categories. Differences arise for hard detections where the pre-trained model receives a relative performance boost of approximately 12.5% for 3D detection. Performance is even more similar in BEV, but again with slightly better results in the hard category after simulated pre-training, perhaps as the corresponding features are better captured and more easily generalized from in BEV. On the Pedestrian class differences are much more apparent, as the pre-trained model sees a relative increase in performance of approximately 60-110% in 3D and 85-135% in BEV. Likely, this is a result of both class imbalance and the smaller object size of pedestrians, but both results clearly favor simulated pre-training when real data is scarce. As models are more likely to overfit with limited data, more testing is however encouraged.

Figures 5.16-5.18, charting the average precision of models with and without pre-training on cars in

the KITTI dataset, clearly demonstrate the benefit of simulated pre-training with regards to speeding up convergence. After only 2,000 steps the pre-trained model achieves more than 50% AP on all categories, for which similar results without pre-training takes eight times as long. As the pre-trained model very quickly converges near optimal performance and is being fine-tuned with a lower learning-rate, the performance gap between the models steadily decreases as training proceeds. It takes around 30,000 steps for the two models to compare in performance, however the pre-trained model is already at a later training stage and able to reach performance peaks significantly higher than without pre-training for the next half of the plotted training.

6.3.2 Multi-class evaluation

Upon inspecting the multi-class results on the KITTI validation set in Table 5.10 no single BEV configuration for AVOD-FPN comes out as clearly superior to the others. Most configurations show significant peaks in performance at only one class at a time, meaning compromises must be made with regards to optimal performance on the detection of cars or pedestrians if used in real time. The default and 3M3D configurations perform best overall, however, the 3M3D takes a slight edge per class for the best performing checkpoints where moderate performance on cars and moderate and hard performance on pedestrians is noticeably higher. All configurations reach a higher peak in performance on the Pedestrian class when compared to single-class models, with some models exceeding 50% 3D AP on the easy category. SECOND boasts quite impressive results in this regard, achieving a boost in performance on the Pedestrian of more than 20% 3D AP for all difficulties, even without any concerning reduction in performance on the Car class. Furthermore, the multi-class configuration of SECOND (having limited the LIDAR range and using a simplified architecture with fewer input features, filters, and voxels) is able to perform faster inference at about 35 milliseconds compared to the previous 50. This makes SECOND an especially compelling alternative for multi-class inference for autonomous driving, leaving more resources for other tasks in the pipeline.

Turning over to the results of pre-training on the CADET dataset and fine-tuning on the KITTI dataset in Table 5.11, we again see results generally comparable to full KITTI-training. The MMD and 3M3DND configurations are not able to hit the same peaks in performance on the Pedestrian class, but 3M3DND in return produces a checkpoint with stronger performance on cars. The default configuration achieves stronger optimal performance on cars after fine-tuning, with less adherent reduction in performance on pedestrians, resulting in a compelling choice for real-world use as the overall best AVOD multi-class model. SECOND sees a significant reduction in performance on the hard category for cars as well as general performance for pedestrians. It however still manages to outperform all the AVOD configurations. Reducing the learning rate on SECOND does produce slightly lower accuracy on either class, but in return produces a much stronger result when evaluated back on the CADET dataset in Table 5.13.

As evident in Table 5.12, AVOD-FPN does generalize much better to the CADET dataset than SECOND. SECOND generally performs inadequately to even reach the IoU requirements for a true positive when having only seen data from KITTI. The difference in performance is somewhat unsurprising as the LIDAR produces a less accurate representation of real-world objects. When comparing to the results after pre-training and fine-tuning in Table 5.13, we see that the latter models show much stronger results on the CADET dataset, meaning the fine-tuning has not fully eradicated the knowledge stored in the pre-trained weights. Reducing the learning rate for SECOND significantly boosts the performance on the CADET dataset for the Car class, with a minor increase as well for the Pedestrian class, suggesting better potential generalization as a wider object representation is captured by the final weights.

6.4 Inference on KITTI

By inspecting the images generated from inference on the KITTI validation set in Figures 5.19-5.20 some differences in detection patterns become apparent, especially for the detection of pedestrians, but with overall very similar performance. While the fully KITTI-trained model catches the pedestrian furthest back in Figure 5.19a, which the pre-trained model misses, it also falsely detects a pedestrian down in the bottom right. While the pre-trained model, in Figure 5.19b, does not produce a false positive pedestrian detection, it instead produces a false positive detection of a car in the far left. Comparing this with Figure 5.20, and other examples in Appendix D, the overall impression is that the model without simulated pre-training detects more pedestrians, but potentially also produces more false positives, suggesting a generally higher confidence score. The CADET pre-trained model on the other side seems to be better at detecting more distant cars.

6.5 Inference on CADET

When comparing the models back on the CADET dataset in Figures 5.21-5.22 larger differences emerge, as suggested by the measured 3D average precision in Tables 5.12 and 5.13. The model pre-trained on CADET produces far more detections of both pedestrians and cars, especially when cars are more distant, directly in front or visibly occluded. Likely this in part due to the model without simulated pre-training having difficulty connecting the LIDAR points with the image as they are not as strongly correlated as in KITTI. The fully KITTI-trained model does not produce any false positives in the sampled images, likely as overall confidence is very low on the simulated data, whereas the CADET trained model produces a false detection of a pedestrian in Figure 5.22b. The KITTI-trained model does not produce any detections of pedestrians in the selected images, however does generalize to some extent as can be seen in Figure D.4 in Appendix D where a pedestrian detection with fairly good localization is produced. The same figure also shows an example where the CADET pre-trained model can produce bounding boxes with poor localization. To summarize, we argue that missing as many detections as the fully KITTI-trained model is far worse performance-wise than the false positives produced by the CADET pre-trained model, as it can more easily lead to dangerous situations.

6.6 Inference on NAP dataset

Figures 5.24-5.27, showing inference results on the NAP dataset from AVOD-FPN with and without simulated pre-training, are much more closely aligned with what seen on the KITTI dataset. This further suggests the AVOD-FPN architectures manages well to generalize from the multimodal sensor data, though a few noticeable differences emerge. In Figure 5.24 the pre-trained model detects an additional car far back in the picture which the other model misses, as well as the middle car on the right, although not discerning much in difficulty from the other cars around it. In Figure 5.26 the pre-trained model detects the pedestrian on the right, while the model trained only on KITTI misses both pedestrians. The pre-trained model however also produces two false positives, whereas the other model produces none in the selected examples. One of a car next to a snow pile in Figure 5.25b and one of a pedestrian in place of a street-sign in Figure 5.27b. It is likely the street sign looks quite similar to the simplified physical representation of pedestrians in the LIDAR scans used in the CADET dataset.

Although the inference results from only training on the CADET dataset in Figures 5.28-5.29 as a whole are clearly inferior to the two previous models, some interesting exceptions occur. For instance, in Figure 5.29a, the model manages to detect both pedestrians, with fairly good localization, despite having never seen real data. The fully KITTI-trained model on the other hand managed neither. Additionally, the model trained only on simulated data manages to detect a pedestrian far back in Figure 5.28a that neither

of the preceding models detected. Apart from this a good number of detection are missed, especially on truncated cars in 5.28a. The model also produces more false positives detections on cars and pedestrians from snow heaps and road signs than when trained on real data.

Finally, and more perhaps interestingly, the LIDAR-based SECOND shows a significant difference in inference performance with and without pre-training in Figures 5.30-5.33. The model solely trained on KITTI, even at a confidence threshold of 0.3, generates a discerning amount of false positive detections, many of which show little to no resemblance to either a car or a pedestrian seen with human eyes. Equally important, practically none of the bounding boxes, even when classified correctly, provide anywhere close to a good overlap of the object. Instead, boxes are generally too small to fit the objects in question or are placed entirely outside the boundaries of the object. Generalization dramatically improves after pre-training on the CADET dataset and fine-tuning on the KITTI dataset. The pre-trained model manages to produce a similar amount of true positive detections on cars and pedestrians when compared the best performing AVOD models, occasionally even showing better performance, although with more false positive pedestrian detections. A good example is Figure 5.32b, where the pre-trained SECOND model manages to detect both pedestrians, but also produces false positive detections in place of two of the traffic lights. Considering how poorly the model generalized from having only seen KITTI LIDAR, it is surprising to see SECOND achieving competitive performance with the multimodal AVOD-FPN architecture only by adding training on simulated data that does not even emulate the arctic environments of the NAP dataset.

Even more surprising is the results from inference using SECOND trained solely on CADET in Figures 5.34-5.35 greatly surpassing those of the fully KITTI-trained model. As with AVOD, results are considerably worse than when pre-trained on CADET and fine-tuned on KITTI, but manages to produce detections of a small number of cars and pedestrians with reasonably good localization in figures 5.34a, 5.34b and 5.35a. In addition to missing more detections when compared to the fine-tuned model, we also see a larger number of false positives in most images, as well as generally poorer localization such as with the car on the left in Figure 5.35b.

Chapter 7

Conclusion and Future Work

In the previous chapter the main results of our conducted experiments were discussed. This chapter aims to place these results into the larger context of autonomous perception by answering the research questions put forth in Chapter 1. Following this, Section 7.2 contains a short summary of the overall achievements of the thesis in the form of the contributions made available for other researches to make use of. Finally, Section 7.3 highlights some of the weaknesses of the contributions and ways in which they could be improved, as well as potential fields for further research in simulation for autonomous driving and multimodal 3D object detection.

7.1 Conclusion

This section aims to discuss the results presented in this thesis in relation to the research questions proposed in Chapter 1. Each research question is first restated and then discussed briefly based on the results and evaluation in the previous chapter.

> **Research question 1:** *Will state-of-the-art 3D object detection architectures be able to detect multiple classes of objects at different sizes in a single forward pass, maintaining real-time performance and sufficient accuracy?*

Judging by the multi-class results, we can conclude that this is indeed possible. While AVOD-FPN did not produce single models that performed its very best at cars and pedestrians simultaneously, it still managed to produce models that achieved near equal or occasionally better performance for both cars and pedestrians than the single-class alternatives. SECOND, having a clever approach to redistributing classes in the LIDAR point clouds when training, manages to achieve strong performance on cars and superior performance on pedestrians to the single-class models, and even manages to do so with faster inference, greatly exceeding any requirement we might have for real-time performance. While there is certainly a limit to how many classes can accurately be detected by a single model, both architectures tested handled two very different classes sufficiently well, without the need to reduce inference speeds.

> **Research question 2:** *Will a model trained solely on simulated data be able to generalize to unseen, real-world environments with different weather and lighting conditions?*

As we can tell from the evaluation on the KITTI dataset and inference on the NAP dataset, a model trained solely on simulated data is able to perform some detections in unseen real-world environments.

Performance pales in comparison to models trained on real data, especially for pedestrians, resulting in systems that are not adequate for real-world use. However we were surprised to see the LIDAR based SECOND architecture generalize better to the unseen environments after training on CADET as opposed to training on KITTI alone. The CADET dataset, based on simulation in CARLA, features a too simplistic LIDAR emulation to achieve competetive performance without any training on real data, as physical models do not properly match the image or real-world objects and weather conditions such as rain are not properly reflected by noise. Additionally, the limited set of maps and persistent environmental impact of certain weather conditions, such as snow on the ground and vehicles, means that there is still a limited number of scenarios that can currently be easily constructed. We expect that as simulators get closer to true photorealism and feature more variety, they should be able to produce sufficiently functional systems for real-world use.

> **Research question 3:** *Can a model trained on simulated data achieve similar performance to full training on real sensor data after limited fine-tuning?*

This question has mainly been investigated in relation to whether less time could be spent training on real data, and additionally using a smaller dataset from the real world, in order to achieve similarly high accuracy on validation or testing data. Considering results gathered on performance after simulated pre-training and fine-tuning on real data, as well as inspecting convergence rate, we conclude that the pre-trained models achieve sufficiently high levels of accuracy much quicker, and with less required real data than when training from scratch. Most single-class models achieve similar performance, and occasionally better, on cars and pedestrians after simulated pre-training within a range of half as much training on real-data. As we only present the best performing checkpoints of each model, rather than list all over multiple test runs, this does not mean we claim top performance can consistently be achieved twice as fast. Rather, our results strongly indicate considerably faster convergence during training and that the pre-trained weights approximates a similar function to what is needed for use on real data. We also include results on a limited dataset of 500 samples as these favors the statement that simulated data can be used to improve model performance when limited by the amount of real data available.

> **Research question 4:** *Can different feature map representations of LIDAR sensor data improve performance for sensor fusion, without the cost of slower inference?*

By evaluation of the single-class results from full KITTI-training, we have introduced novel feature map representations of LIDAR in BEV that either demonstrate significantly better accuracy on the detection of pedestrians, without reduction in inference speeds, or faster inference on the detection of cars, without significant loss in accuracy. The 3M3D and 3M3DND configurations, being slight variations of the same concept, perform significantly better on pedestrians with similar performance on cars compared to the default AVOD-FPN configuration. The MMD configuration achieves a speed up of 4-5% with negligible losses in accuracy on the Car class, although lacking fidelity for accurate detection of pedestrians. Results after simulated pre-training alters the evaluation slightly, as MMD sees a slight reduction in accuracy on cars in the moderate difficulty category, and 3M3D performs much more similarly to default configuration on pedestrians. The 3M3DND configuration however still outperforms the default configuration by a similar margin to the previous results. Multi-class evaluations does not produce as clear results, but largely favors the default and 3M3D configuration, trading blows in performance on cars and pedestrians over various models, although 3M3D achieves significantly better performance to all other models tested when evaluated on the pedestrian class of the CADET dataset after fine-tuning. Further testing, perhaps using larger batch sizes, batch normalization and more frequent checkpoint evaluation might provide more conclusive results for multi-class configurations.

> **Research question 5:** *Does multimodal architectures generalize better to unseen environments than solely LIDAR based architectures?*

As investigation into this question is limited to research gathered from two architectures, we cannot give an entirely conclusive answer, but rather use the results for an informed assumption. Inference results on the NAP dataset clearly favors AVOD-FPN when solely trained on KITTI, as SECOND does not manage to generalize to the new environment and produces a large amount of bounding boxes with seemingly random localization and classification seen from a human perspective. Results on SECOND, however greatly improve after simulated training or simulated pre-training with fine-tuning on KITTI. Based on generalization from training on only one dataset, AVOD-FPN produces clearly better results, but when pre-trained on CADET and fine-tuned on KITTI the performance is more comparable. This might be suggesting that the SECOND architecture is expressive enough to handle a more generalized representation of the objects in question. However, when evaluated back on the CADET dataset after fine-tuning, SECOND does not achieve comparable performance to AVOD-FPN on the Pedestrian class. We consider the gathered results to suggest that solely LIDAR based systems are more prone to overfit, whereas the addition of more sensors such as cameras seem to help the model accept larger deviations from trained data. As such, we consider multimodal systems to have more promise for generalization to unseen environments, especially as new and improved technologies and architectures emerge.

> **Research question 6:** *Will the addition of simulated pre-training improve results when tested on unseen arctic environments?*

AVOD-FPN generalizes well from training on KITTI alone and does not obtain any considerable performance increase from pre-training on the simulated CADET dataset when evaluated on the NAP dataset, as either model makes a similar amount of detections the other omits under the same confidence threshold. SECOND however, sees a massive improvement in the quality of predictions after simulated pre-training, taking it from practically unusable in arctic environments to comparable with the AVOD-FPN and even occasionally superior in the detection of pedestrians, although with generally more false positives at the tested threshold. To conclude, simulated pre-training in its current state does not seem to hurt generalization, and shows the potential for greatly improving generalization on certain architectures, even despite the simulation not directly emulating the target environment. Better simulation and more varied environments should only serve to improve these results.

7.2 Contributions

In this work, we have presented a tool capable of generating large amounts of synthetic data from the CARLA simulator. As far as the authors are aware, this is the first of its kind for the CARLA simulator, or an open source simulator in general, and is completely open source and available under the MIT license. We believe that this tool can be of use for everyone interested in training object detection models for vehicle or pedestrian detection, since it is easy to use and does not require any manual labeling or supervision. As demonstrated the tool can potentially improve generalization of models for object detection and is based on an actively developed, open-source driving simulator with strong potential for better quality simulations, more classes, more environments and more weather conditions as development proceeds.

Furthermore, we have introduced novel feature map configurations in BEV, with altered source code in the forked AVOD-FPN repository such that further experimentation is simple to perform and our experiments are simple to reproduce. These use of these feature map configurations indicate better performance either with regards to accuracy or run-time speeds for several of the tested models. As projection to BEV is still at the state-of-the-art for sensor fusion in 3D object detection, the authors hope that the experiments can support further improvements to state-of-the-art multimodal architectures and motivate for further research in feature representations for sensor fusion.

Parts of our work have been peer-reviewed and published [Brekke et al., 2019], and was presented at the Norwegian AI Society Symposium in May 2019, presenting CADET and prelimenary results on simulated pre-training and BEV experiments. The authors also include added code in the forked repositories for AVOD-FPN and SECOND that allow for online inference in real time without the need for batch preprocessing or indexing. As a result, anyone can set up these models and test them in real-world scenarios with little modification. The SECOND repository also includes added code based on the AVOD-FPN source code in order to directly perform inference and visualize the results.

7.3 Future Work

In this section, we discuss possible improvements to the work presented in this thesis. Many of these are ideas that could not be implemented due to time constraints, but would provide improvements to the systems currently implemented. First, the improvements to the CARLA simulator are discussed, ranging from how to improve the currently available sensors, to adding more maps and classes. Following this, some thoughts on possible improvements to multimodal 3D object detection architectures are discussed. Lastly, we discuss how the inference results on the NAP dataset can be improved by labeling data, using the supervised or semi-supervised tools discussed in Chapter 2.

7.3.1 Improvements to the CARLA simulator

LIDAR sensor

When generating synthetic data, it is important that the data mimics real-world data as closely as possible. For simulators, this means to create sensors such as cameras to generate photorealistic images, and for LIDAR sensors to generate realistic point clouds. Sensors in the real world often generate noisy data – cameras often contain salt and pepper noise or motion blur, while LIDAR sensors can be negatively affected by aspects such as rain or snow. Currently, the CARLA simulator only offers a small variety of post-processing effects for cameras, including motion blur, bloom and lens flare, but does not contain any post-processing for LIDAR. In fact, the LIDAR sensor uses the same underlying mesh for almost all vehicles, which means that even though two vehicles may have completely different shapes and sizes, they will look identical in the LIDAR point cloud. In addition, the LIDAR does not take into account the weather conditions, and will thus generate an identical point cloud with and without rain or snow. A promising solution can be to simulate the effects of noise on LIDAR sensors under rainy or snowy conditions [Goodin et al., 2019]. The issue with handling of meshes in the LIDAR can be tackled by utilizing the depth map sensor instead of ray-cast LIDAR. However, this depth map sensor does not cover 360 degrees around the vehicle and acts more similarly to a camera, not capturing the physical traits of a rotating LIDAR. This is not a problem for most object detection architectures utilizing LIDAR since they only use the front-facing section of the LIDAR scan.

Adding more weather conditions

At the time of writing, CARLA only contains the weather conditions presented in Figure 2.7, which is drizzly, sunny, downpour and cloudy. Cameras perform relatively well for all these weather conditions, but might be affected slightly by lens flare and other artifacts. Increasing the number of weather types in CARLA would make models generalize to a larger variety of driving scenarios, assuming that the simulation includes realistic effects on sensors as discussed above. For foggy weather, the results presented by Li et al. [2017] seems to yield realistic simulations of fog and haze by utilizing a depth map of the scene. It is important that these weather conditions do not only appear as visual effects in the air, but also makes

an impact on the ground and objects as well. For instance, snow and rain should affect the ground, in the same way that it does in the real world.

Increasing number of driving scenarios

A major drawback of utilizing CARLA as a simulator for generating synthetic datasets is the lack of varied maps. I version 0.8.5 of CARLA there are only two maps, both of which are from an urban environment with relatively similar scenery. Thus, generating data from these environments will not cover the vast amounts of possible driving scenarios, such as driving in rural areas with varied terrain. In addition, both maps are completely flat, which means that all vehicles will have very similar pitch. Therefore, models that are trained solely on this synthetic data will most likely perform worse when faced with vehicles on an incline plane. Version 0.9.5 increases the number of maps to seven, including one rural map, but carries with it a different API and some lacking features, meaning CADET can not be directly transferred to the new versions of CARLA.

It is possible to create new maps for the CARLA simulator using Unreal Engine, but this was not done in during the work on this thesis since it is a time consuming process which includes creating the actual map, creating *pathing*[1] for all agents, and selecting spawn points. New maps yield a larger variety in driving scenarios, which can make machine learning models generalize better to unseen data.

Adding more classes

Compared to the KITTI dataset, the CADET tool offers a smaller variety of classes that can be labeled. Currently, this limitation is in part due to how CARLA organizes its different classes. Even though the CARLA simulator contains both bikes, trucks and busses, all fall under the same category, namely Vehicle, which makes it challenging to correctly label each object. In addition, the underlying object meshes of bikes are very simplified, which makes them less suitable for labeling. However, it is possible to differentiate between the different classes using a combination of their dimensions, bounding boxes and agent ID. If implemented, the generated datasets will have increased value, as several object detection architectures are capable of detecting a large variety of classes.

7.3.2 Improvements to multimodal architectures

Although some BEV configurations, especially in single-class, produced considerably stronger results than the original AVOD-FPN configuration, they still lack in performance compared to the multi-class SECOND configurations when sufficient training data is available. We are fairly convinced that there are considerable limitations with the use of LIDAR in BEV or otherwise projecting the 3D information to a set of image planes. Future state-of-the-art architectures will likely require combinations of image processing, potentially still through the use of CNNs, with full LIDAR processing maintaining geometric relations in 3D similarly to architectures such as PointNet or SECOND. We also imagine that inclusion of other measurements such as movement speed, for example mapped to either regions or points in the point cloud, estimated from either the camera, LIDAR or inclusion of other sensors, could augment the data to better be able to detect moving objects. Temporal models based on RNN architectures can likely also improve stability in detection of moving objects, or when the system is in motion, eliminating most of the jitter visible in modern object detectors when used in real-time.

[1] Pathing: allocation or planning of a path

7.3.3 The future of NAP

Judging by the results presented in this thesis, there are still a lot of challenges to solve before an autonomous system can be put into traffic. Arctic environments pose a multitude of unique challenges, ranging from the reflectance of ice in RADAR and LIDAR sensors to interference from snow or low temperatures. We believe that NAP can play a central role in conducting research within this field.

In the field of autonomous perception, most of the available datasets contain training samples from a very limited set of environments. BDD100K contains samples from several large cities in the United States, while KITTI is sampled entirely from Karlsruhe, Germany. If autonomous vehicle systems are to be deployed in arctic environments it is crucial that they have been trained, at least in part, in such environments[2]. Sensors such as LIDAR and cameras might behave differently under snow or rain, which is common in Nordic environments. A major milestone for NAP would be to generate its own labeled dataset, which contains a wide variety of driving scenarios with conditions difficult to capture many other places. The authors believe that this dataset would benefit the autonomous perception community, as no large dataset of its kind exists today.

On the basis of the evaluation of annotation tools discussed in Section 2.4.2, the authors recommend Scalabel.ai as the main annotation tool for use with projects such as NAP. By utilizing the labeled data gathered by NAP, it may be possible for models to generalize to an even larger variety of environments, bringing autonomous vehicle systems closer to being available in larger parts of the world. The dataset may also serve as a benchmark for object detection in autonomous perception, much like KITTI has for several years. However, labeling data is still very costly. We therefore propose that NAP uses a form of bootstrapping for labeling data. This process would start by first labeling a small amount of data from arctic environments, and then training machine learning models on a combination of simulated data and the labeled data. This trained model would then be used to suggest bounding boxes on unlabeled data, with or without the use for human supervision. Then, this newly labeled data would be used to train new models, which would presumably outperform the old due to the increase in data. Furthermore, suggested by the inference results on the NAP dataset in this report, we propose that the CADET pre-trained AVOD and SECOND models would serve as a good starting point for assisted labeling.

Although we have focused on supervised learning in this thesis, several fields of machine learning has shifted towards unsupervised learning in the last few years. As Yann LeCun put it, *"The revolution will not be supervised"*[NYU Tandon School of Engineering, 2019]. By utilizing neural networks to cluster unlabeled data, one can gain insight into the distribution of the underlying data, and possibly alleviate the issues of the annotation bottleneck. In June 9th, 2019, the 1st workshop on Unsupervised Learning for Automated Driving was conducted at the IEEE Intelligent Vehicles Symposium[IEEE Intelligent Vehicles Symposium, 2019], which illustrates that the field of autonomous driving is also paying attention to unsupervised learning. Although these methods are on the rise, we expect that the future autonomous driving systems will be semi-supervised, using a combination of labeled and unlabeled data. An important factor for the continued research in supervised learning however is the explainability of inference in such systems, as errors in the real world must be accounted for and remedied. For autonomous vehicles to be accepted by society and working in conjunction with human beings, it is crucial that their behaviour is explainable [Langley et al., 2017].

7.4 Reproducibility

The authors consider reproducibility a topic of considerable importance. Therefore, we have published the CADET dataset and the weights for the best performing AVOD and SECOND models, shown in the list below. The tool to generate a similar dataset [Brekke and Vatsendvik, 2019b], as well as modified

[2]Datasets such as BDD100K contain some instances of snow and rain from scenes in New York.

source code for AVOD-FPN [Brekke and Vatsendvik, 2019a] and SECOND [Brekke and Vatsendvik, 2019c] are also available and open-source, including configuration files for all tested models. This should ensure that any reader can reproduce the results presented in this thesis.

CADET: `https://studntnu.sharepoint.com/:u:/s/teamsite2/10199/EfudfwtiCqFOqUcFWTKcUkEBCJeIQIYHGT83_dRQA4w2fQ?e=IfzAqE`

AVOD model weights: `https://studntnu.sharepoint.com/:f:/s/teamsite2/10199/Ek3vSVACVppIuxvJa7MaAE0B2OK-7UVeJA9Cp7prHwdvgA`

SECOND model weights: `https://studntnu.sharepoint.com/:f:/s/teamsite2/10199/Ev0NWJBBh7ROjwru_WAPjvkBTIG0hU7P55rI-tWaBA0u_g?e=gE1fa5`

Bibliography

Baidu Apollo (2019). Apollo auto. https://github.com/ApolloAuto/apollo/. [Online; accessed 10-June-2019]. (document), 2.8

Brekke, Å. and Vatsendvik, F. (2019a). Avod. https://github.com/Fredrik00/avod. [Online; accessed 10-June-2019]. 5.2.2, 7.4

Brekke, Å. and Vatsendvik, F. (2019b). carla-data-export. https://github.com/Ozzyz/carla-data-export. [Online; accessed 6-June-2019]. 7.4

Brekke, Å. and Vatsendvik, F. (2019c). Second. https://github.com/Fredrik00/second.pytorch. [Online; accessed 10-June-2019]. 7.4

Brekke, Å., Vatsendvik, F., and Lindseth, F. (2019). Multimodal 3d object detection from simulated pretraining. *arXiv preprint arXiv:1905.07754*. (document), 7.2

Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. (2019). nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*. (document), 2.2.3, 2.4

Chen, X., Ma, H., Wan, J., Li, B., and Xia, T. (2017). Multi-view 3d object detection network for autonomous driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6526–6534. IEEE. (document), 3.2, 3.2.1, 3.1

Cho, H., Seo, Y.-W., Kumar, B. V., and Rajkumar, R. R. (2014). A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1836–1843. IEEE. 3.2

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE Computer Society. 2.7

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE. 2.6

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). Carla: An open urban driving simulator. In *Conference on Robot Learning*, pages 1–16. (document), 2.4.1

Gamma, E. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Education India. 2.4.3

Geiger, A. (2019). The kitti vision benchmark suite - 3d object detection evaluation 2017. `http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d`. [Online; accessed 10-June-2019]. 2.2.1, 4.2.1

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237. (document), 2.2.1, 2.2, 2.2.4, 5.1, 5.2.1, 5.2.1, 5.1

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE. 2.8.3

Ghiasi, G., Lin, T.-Y., Pang, R., and Le, Q. V. (2019). Nas-fpn: Learning scalable feature pyramid architecture for object detection. *arXiv preprint arXiv:1904.07392*. 3.1

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448. 3.4.1

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. 2

Goodin, C., Carruth, D., Doude, M., and Hudson, C. (2019). Predicting the influence of rain on lidar in adas. *Electronics*, 8(1):89. 2.1.1, 7.3.1

Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition. 2.3

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969. 3.1, 3.4.1

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385. (document), 2.7.2, 2.7.2, 2.12

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257. 6

Huang, X., Cheng, X., Geng, Q., Cao, B., Zhou, D., Wang, P., Lin, Y., and Yang, R. (2018). The apolloscape dataset for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 954–960. (document), 2.2.4, 2.5

Huber, P. J. (1992). Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer. 3.3.3

IEEE Intelligent Vehicles Symposium (2019). 1st workshop on unsupervised learning for automated driving. `http://intelligent-vehicles.org/ulad-2019/`. [Online; accessed 10-June-2019]. 7.3.3

Korosec, K. (2019). Waymo to start selling standalone lidar sensors. `https://techcrunch.com/2019/03/06/waymo-to-start-selling-standalone-lidar-sensors/`. [Online; accessed 5-May-2019]. 2.1.1

Krafcik, J. (2018). Where the next 10 million miles will take us. https://medium.com/waymo/where-the-next-10-million-miles-will-take-us-de51bebb67d3. [Online; accessed 10-June-2019]. 2

Ku, J., Mozifian, M., Lee, J., Harakeh, A., and Waslander, S. L. (2018). Joint 3d proposal generation and object detection from view aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE. (document), 3.2, 3.2.2, 3.2, 3.3, 4.2, 4.2.1, 5.3.2

Langley, P., Meadows, B., Sridharan, M., and Choi, D. (2017). Explainable agency for intelligent autonomous systems. In *Twenty-Ninth IAAI Conference*. 7.3.3

Li, K., Li, Y., You, S., and Barnes, N. (2017). Photo-realistic simulation of road scene for data-driven methods in bad weather. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 491–500. 7.3.1

Li, Q. (2012). Literature survey: domain adaptation algorithms for natural language processing. *Department of Computer Science The Graduate Center, The City University of New York*, pages 8–10. 2.6

Lin, T., Goyal, P., Girshick, R. B., He, K., and Dollár, P. (2017). Focal loss for dense object detection. *CoRR*, abs/1708.02002. 3.4.2

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932. 2.7

NovAtel Inc (2019). High-precision gps for autonomous vehicles. https://www.novatel.com/industries/autonomous-vehicles/. [Online; accessed 6-June-2019]. 2.1.4

NYU Tandon School of Engineering (2019). "the revolution will not be supervised" promises facebook's yann lecun in kickoff ai seminar. https://engineering.nyu.edu/news/revolution-will-not-be-supervised-promises-facebooks-yann-lecun-kickoff-ai-seminar. [Online; accessed 10-June-2019]. 7.3.3

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359. 2.6

Parker, M. (2013). Implementing digital processing forautomotive radar using socs (white paper). https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01183-automotive-radar-socfpga.pdf. [Online; accessed 10-June-2019]. 2.1.2

Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 1*, pages 305–313. Morgan-Kaufmann. 1.1

Qi, C. R., Liu, W., Wu, C., Su, H., and Guibas, L. J. (2018). Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 918–927. (document), 3.3.2, 3.5, 3.3.2

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660. (document), 3.3.1, 3.4

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788. 3.4.3

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*. 3.1

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99. 3.2.1, 3.4.1

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2018). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer. 2.4.1

Shamir, O. (2018). Distribution-specific hardness of learning neural networks. *The Journal of Machine Learning Research*, 19(1):1135–1163. 6.2

Shi, S., Wang, X., and Li, H. (2018). Pointrcnn: 3d object proposal generation and detection from point cloud. *arXiv preprint arXiv:1812.04244*. 3.4.1

Shin, K., Kwon, Y. P., and Tomizuka, M. (2018). Roarnet: A robust 3d object detection based on region approximation refinement. *CoRR*, abs/1811.03818. (document), 3.3.3, 3.6, 3.7

Simon, M., Milz, S., Amende, K., and Gross, H.-M. (2018). Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds. In *European Conference on Computer Vision*, pages 197–209. Springer. 3.4.3

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556. 2.7.1

Tajitsu, N. (2019). On the radar: Nissan stays cool on lidar tech, siding with tesla. `https://finance.yahoo.com/news/radar-nissan-stays-cool-lidar-092439416.html`. [Online; accessed 6-June-2019]. 3

Thorpe, C., Hebert, M. H., Kanade, T., and Shafer, S. A. (1988). Vision and navigation for the carnegie-mellon navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):362–373. 1.1

Velodyne Lidar Inc. (2019). Velodyne slashes the price in half of its most popular lidar sensor - velodyne press release. `https://velodynelidar.com/newsroom/velodyne-slashes-the-price-in-half-of-its-most-popular-lidar-sensor/`. [Online; accessed 5-May-2019]. 2.1.1

Wang, M. (2019). The state of solid-state 3d flash lidar. `https://medium.com/@miccowang/the-state-of-solid-state-3d-flash-lidar-f0107dcc4c84`. [Online; accessed 10-June-2019]. 2.1.1

Yan, Y., Mao, Y., and Li, B. (2018). Second: Sparsely embedded convolutional detection. *Sensors*, 18(10). (document), 3.4.2, 3.8, 5.3.2

Yang, B., Luo, W., and Urtasun, R. (2018). Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7652–7660. (document), 3.4.4, 3.9

Yu, F. (2018). Bdd100k: A large-scale diverse driving video database. `https://bair.berkeley.edu/blog/2018/05/30/bdd/`. [Online; accessed 10-June-2019]. 2.2.2

Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., and Darrell, T. (2018). Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*. (document), 2.2.2, 2.3, 2.4.2

Appendix A

Code

In order to ensure that our findings can be reproduced and applied easily, we have released all the code described in this thesis.

The tool developed for dataset generation, named CADET is available under the MIT license at `https://github.com/Ozzyz/carla-data-export`.

The modifications made to the AVOD architecture, and all training configurations are available at `https://github.com/Fredrik00/avod`, also under the MIT license. Note that this project is originally forked from `https://github.com/kujason/avod`.

The configurations for the SECOND architecture is found at `https://github.com/Fredrik00/second.pytorch`, which is originally forked from `https://github.com/traveller59/second.pytorch`. Our version contains the aforementioned training configurations and modifications to run in real-time, in addition to the original architecture.6a

The ROS setup described in Section 5.2.3 can be found at `https://github.com/Ozzyz/pandora-kitti-ros`. Note that this solely includes the communication setup and no actual data. The dataset created by NAP contains confidential data, and can therefore not be released as-is.

Appendix B

CADET algorithms

Algorithm 1 Vertex occlusion detection. A vertex is defined as occluded when all neighbouring pixel depths are closer than the vertex.

```
1: procedure IsOccluded(vertex, depthBuffer)
2:     for x,y in vertex.neighbours do
3:         if depthBuffer[x, y] >= vertex.depth then
4:             return False
5:         end if
6:     end for
7:     return True
8: end procedure
```

Algorithm 2 Line drawing algorithm implemented in CADET

```
1: procedure drawLine(x1, y1, x2, y2)
2:     steep ← abs(y2 − y1) > abs(x2 − x1)
3:     if steep then
4:         Swap(x1, y1)
5:         Swap(x2, y2)
6:     end if
7:     dx ← x2 − x1
8:     dy ← abs(y2 − y1)
9:     err ← int(dx/2)
10:    y ← y1
11:    y_step ← sign(y2 − y1)
12:    for x:=x1 to x2 do
13:        if steep then
14:            draw(y, x)
15:        else
16:            draw(x, y)
17:        end if
18:        err ← err − dy
19:        if err < 0 then
20:            y ← y + y_step
21:            err ← err + dx
22:        end if
23:    end for
24: end procedure
```

Appendix C

CADET sample structure

The CADET tool generates training samples in a structure defined in 5.1. In this section, we illustrate all data that belongs to a single sample generated by CADET. This includes an image, a LIDAR pointcloud, a file for camera calibration, a file for groundplane estimation and finally the labels of the objects in the image. Normally, the filenames of these files would be the number of the generated sample, left padded with zeroes. For instance, the first sample would contain data with filenames like `000000.txt` for calib, label and groundplane files, while the LIDAR would have the name `000000.bin`, and the image `000000.png`. For clarity, we refer to these files by their usage, so for instance the calibration file is named `calib.txt`.

`calib.txt`

```
P0: 624.0 0.0 624.0 0.0 0.0 624.0 192.0 0.0 0.0 0.0 1.0 0.0
P1: 624.0 0.0 624.0 0.0 0.0 624.0 192.0 0.0 0.0 0.0 1.0 0.0
P2: 624.0 0.0 624.0 0.0 0.0 624.0 192.0 0.0 0.0 0.0 1.0 0.0
P3: 624.0 0.0 624.0 0.0 0.0 624.0 192.0 0.0 0.0 0.0 1.0 0.0
R0_rect: 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0
Tr_velo_to_cam: 0 -1 0 0 0 0 -1 0 1 0 0 0
TR_imu_to_velo: 1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0 0.0
```

`label.txt`

```
Car 0 0 -10 804 192 902 220 1.5797 4.9932 1.9417 13.2613 1.61324
    36.2599 1.5708
Car 0 0 -10 497 193 553 235 1.5232 3.6688 1.6640 -3.84996 1.6072
    24.8819 1.06532e-06
Car 0 0 -10 448 192 542 250 1.5797 4.9932 1.9417 -3.8498 1.6127
    19.6589 6.3916e-06
Pedestrian 0 0 -10 707 183 729 226 2.0 0.8999 0.6999 4.4821 1.5890
    29.6993 1.570
```

`groundplane.txt`

```
# Plane
Width 4
Height 1
-1.7576863191307934e-05 -0.9999999997141049 1.62124624880309e-05 1.6
```

image.png

Figure C.1: The corresponding image to the label.txt file. Note that there are three visible vehicles and one pedestrian, which coincides with the label file.

lidar.bin

Since the .bin file is written in a binary format, it is not easy to visualize. Therefore, we illustrate the 3D visualization of the data in the .bin file instead.

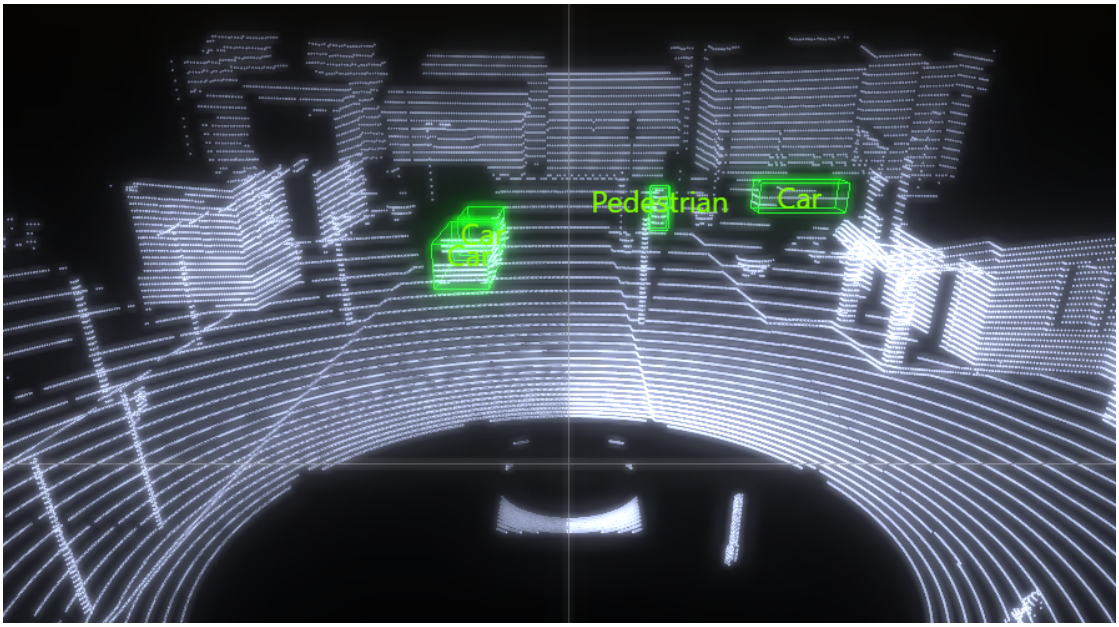


Figure C.2: The corresponding LIDAR scan of the scene. Note that there are three visible vehicles and one pedestrian, which coincides with the label file.

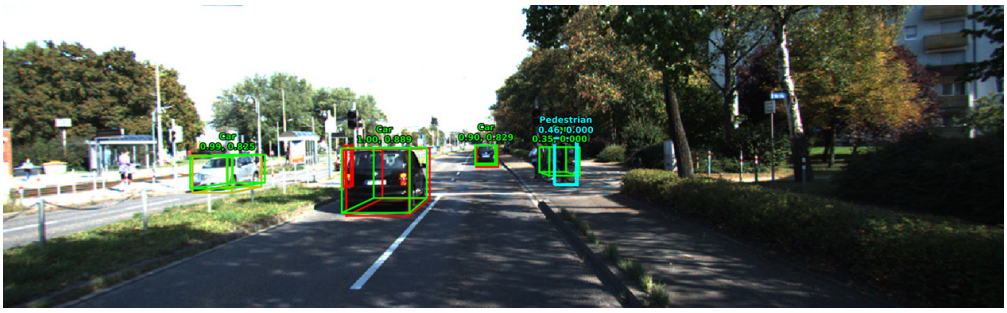
Appendix D

KITTI and CADET inference results

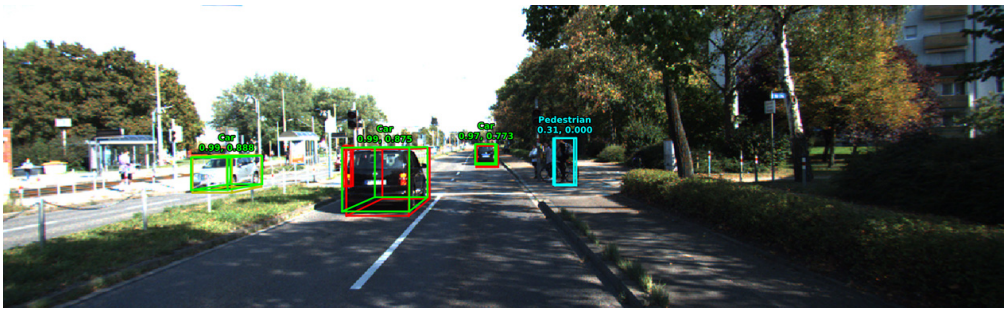
(a)

(b)

Figure D.1: AVOD inference on KITTI sample 33 without (a) and with (b) pre-training on CADET.

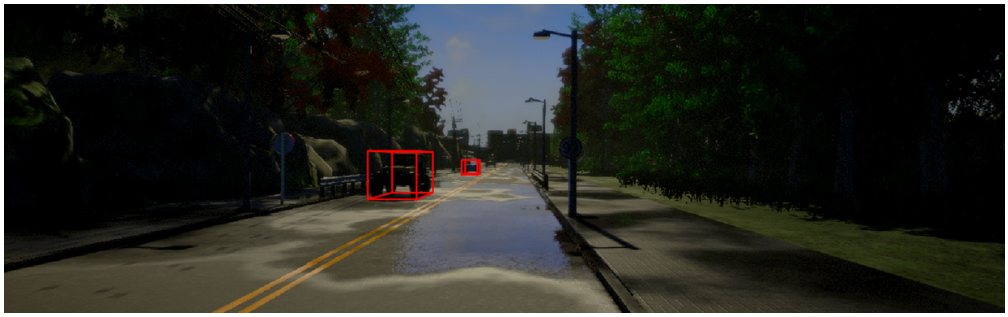


(a)

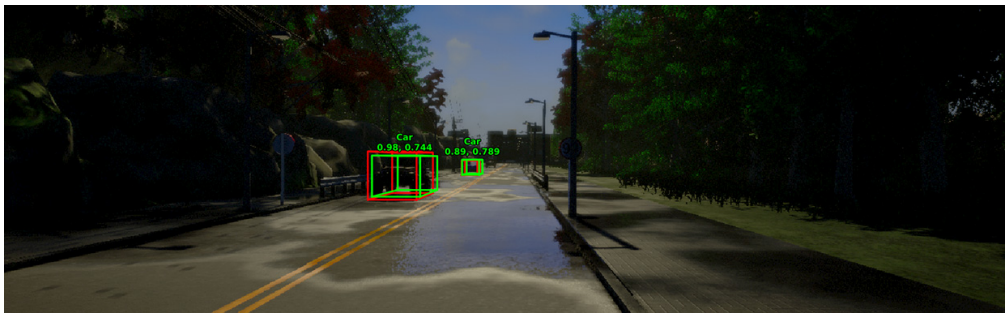


(b)

Figure D.2: AVOD inference on KITTI sample 35 without (a) and with (b) pre-training on CADET.



(a)



(b)

Figure D.3: AVOD inference on CADET sample 8047 without (a) and with (b) pre-training on CADET

(a)

(b)

Figure D.4: AVOD inference on CADET sample 8065 without (a) and with (b) pre-training on CADET

Appendix E

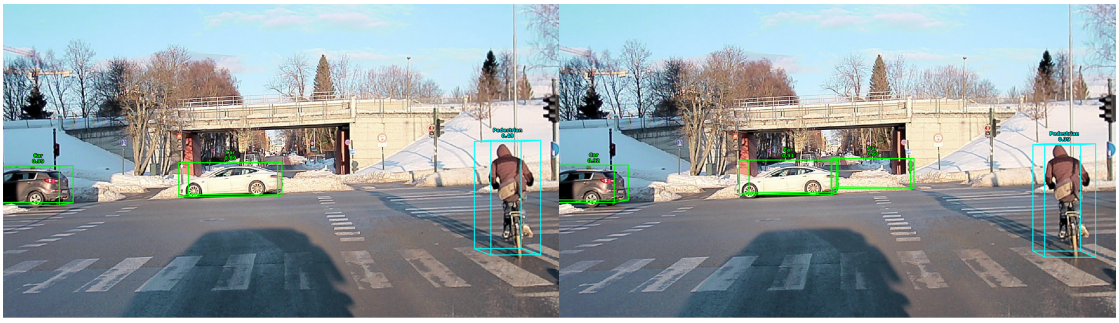
NAP dataset inference results

(a) (b)

Figure E.1: AVOD inference on NAP sample 240 without (a) and with (b) pre-training on CADET.

(a) (b)

Figure E.2: AVOD inference on NAP sample 2960 without (a) and with (b) pre-training on CADET.

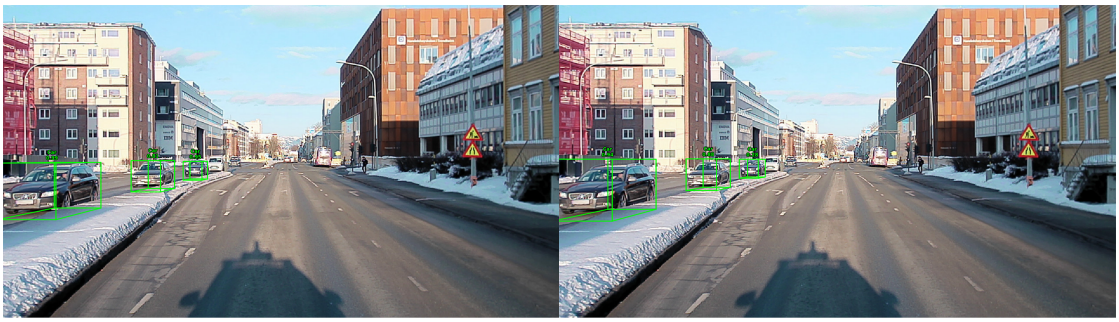


(a) (b)

Figure E.3: AVOD inference on NAP sample 3280 without (a) and with (b) pre-training on CADET.

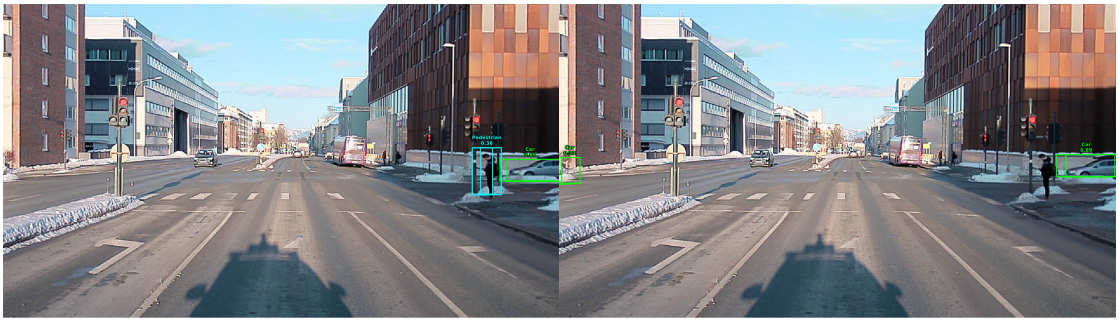
(a) (b)

Figure E.4: AVOD inference on NAP sample 3680 without (a) and with (b) pre-training on CADET.



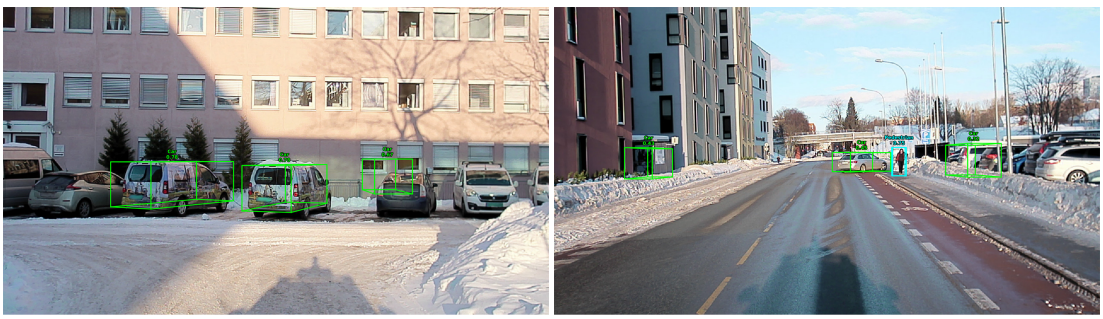
(a) (b)

Figure E.5: AVOD inference on NAP sample 5620 without (a) and with (b) pre-training on CADET.



(a) (b)

Figure E.6: AVOD inference on NAP sample 5660 without (a) and with (b) pre-training on CADET.



(a) (b)

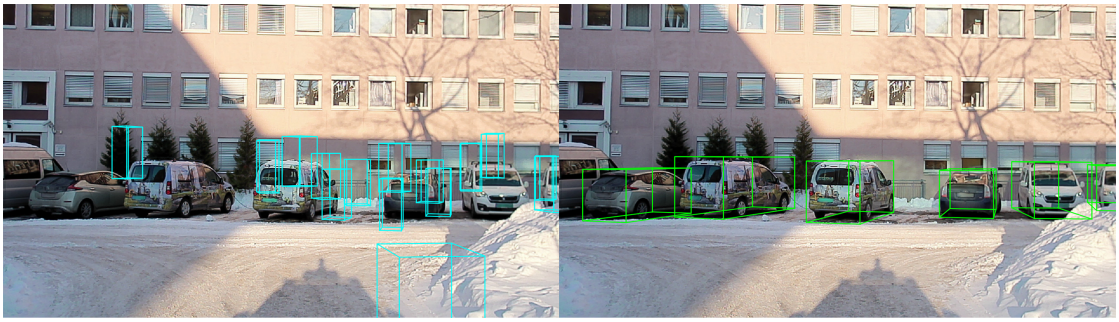
Figure E.7: AVOD inference on NAP sample (a) 240 and (b) 2960 trained solely on CADET

(a) (b)

Figure E.8: AVOD inference on NAP sample (a) 3680 and (b) 3280 trained solely on CADET

(a) (b)

Figure E.9: AVOD inference on NAP sample (a) 5620 and (b) 5660 trained solely on CADET

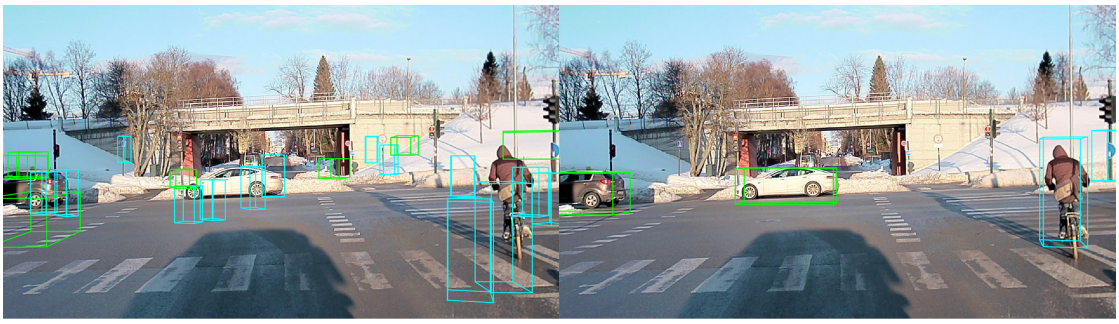


(a) (b)

Figure E.10: SECOND inference on NAP sample 240 without (a) and with (b) pre-training on CADET

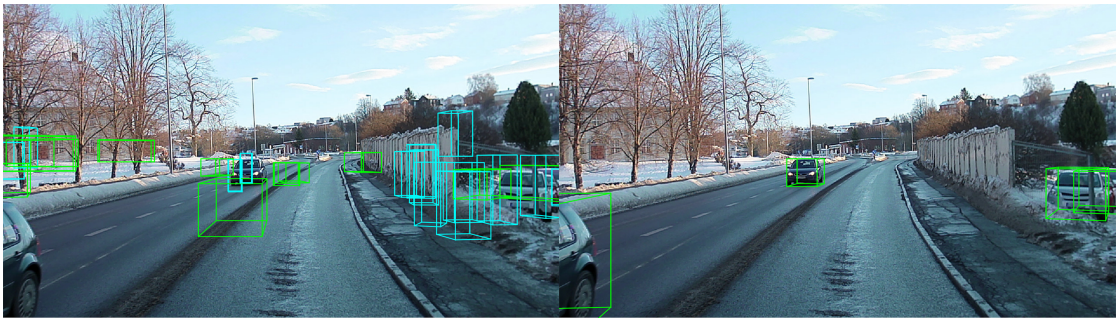
(a) (b)

Figure E.11: SECOND inference on NAP sample 2960 without (a) and with (b) pre-training on CADET



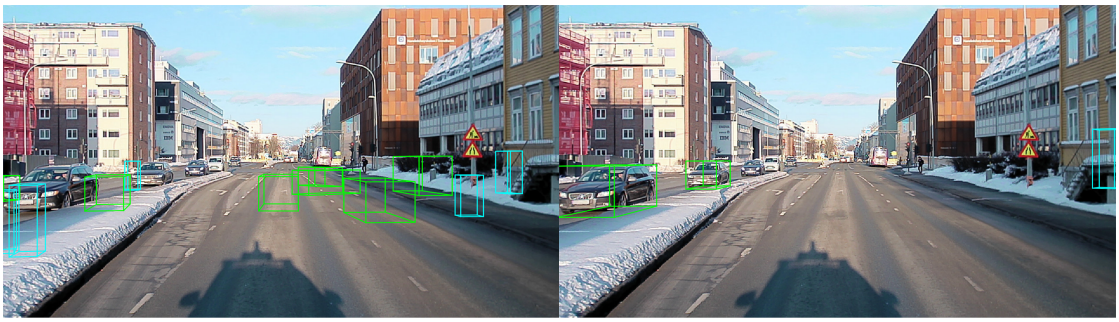
(a) (b)

Figure E.12: SECOND inference on NAP sample 3280 without (a) and with (b) pre-training on CADET



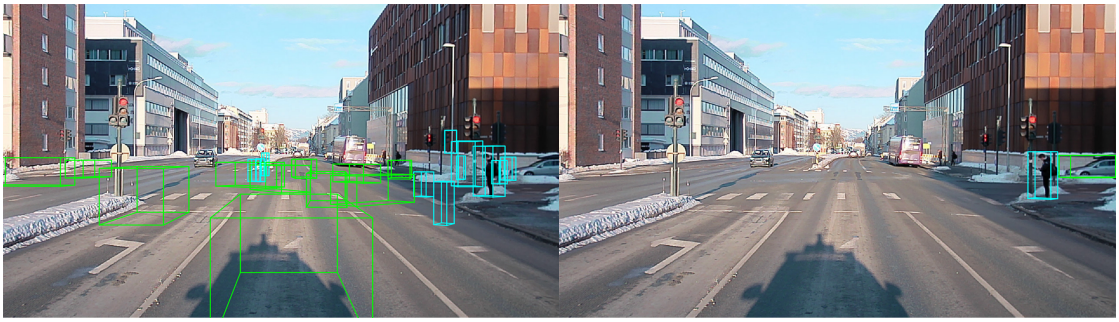
(a) (b)

Figure E.13: SECOND inference on NAP sample 3680 without (a) and with (b) pre-training on CADET.



(a) (b)

Figure E.14: SECOND inference on NAP sample 5620 without (a) and with (b) pre-training on CADET.



(a) (b)

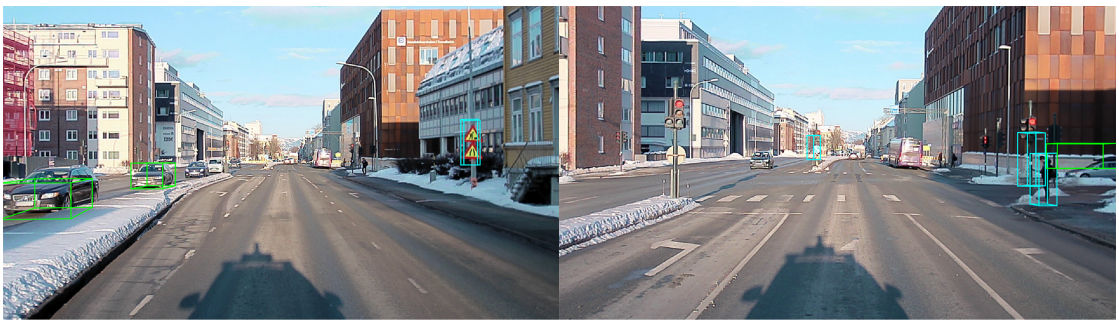
Figure E.15: SECOND inference on NAP sample 5660 without (a) and with (b) pre-training on CADET.

(a) (b)

Figure E.16: SECOND inference on NAP sample (a) 240 and (b) 2960 trained solely on CADET

(a) (b)

Figure E.17: SECOND inference on NAP sample (a) 3680 and (b) 3280 trained solely on CADET



(a) (b)

Figure E.18: SECOND inference on NAP sample (a) 5620 and (b) 5660 trained solely on CADET

Appendix F

NAIS2019 Paper

This section contains the paper we presented at the symposium for the Norwegian AI Society the 28th of May 2019. The paper is also available at `https://arxiv.org/abs/1905.07754`.

Multimodal 3D Object Detection from Simulated Pretraining

Åsmund Brekke[0000−0003−1114−5320], Fredrik Vatsendvik[0000−0002−5300−0202], and Frank Lindseth[0000−0002−4979−9218]

Norwegian University of Science and Technology, Trondheim, Norway
{aasmunhb,fredrva}@stud.ntnu.no, frankl@ntnu.no

Abstract. The need for simulated data in autonomous driving applications has become increasingly important, both for validation of pretrained models and for training new models. In order for these models to generalize to real-world applications, it is critical that the underlying dataset contains a variety of driving scenarios and that simulated sensor readings closely mimics real-world sensors. We present the Carla Automated Dataset Extraction Tool (CADET), a novel tool for generating training data from the CARLA simulator to be used in autonomous driving research. The tool is able to export high-quality, synchronized LIDAR and camera data with object annotations, and offers configuration to accurately reflect a real-life sensor array. Furthermore, we use this tool to generate a dataset consisting of 10 000 samples and use this dataset in order to train the 3D object detection network AVOD-FPN, with finetuning on the KITTI dataset in order to evaluate the potential for effective pretraining. We also present two novel LIDAR feature map configurations in Bird's Eye View for use with AVOD-FPN that can be easily modified. These configurations are tested on the KITTI and CADET datasets in order to evaluate their performance as well as the usability of the simulated dataset for pretraining. Although insufficient to fully replace the use of real world data, and generally not able to exceed the performance of systems fully trained on real data, our results indicate that simulated data can considerably reduce the amount of training on real data required to achieve satisfactory levels of accuracy.

Keywords: Autonomous driving · Simulated data · 3D object detection · CARLA · KITTI · AVOD-FPN · LIDAR · Sensor fusion

1 Introduction

Machine learning models are becoming increasingly complex, with deeper architectures and a rapid increase in the number of parameters. The expressive power of such models allow for more possibilities than ever before, but require large amounts of labeled data to properly train. Labeling of data in the autonomous driving domain requires extensive amounts of manual labour, either in the form of actively producing annotations such as class labels, bounding boxes and semantic segmentation by hand, or by supervising and adjusting automated

generation of these using a pretrained ensemble of models from previously labeled data. For the use of modern sensors such as LIDAR, not many sizable labeled datasets exists, and those that do generally offer little variation in terms of environments or weather conditions to properly allow for generalization to real world conditions. Popular datasets such as KITTI [2] offers a large array of sensors, but with largely unchanging weather conditions and lighting, while the larger and more diverse BDD100K [3] dataset does not include multimodal sensor data, only offering camera and GPS/IMU. The possibility of new sensors being introduced that greatly impact autonomous driving also carry the risk of invalidating the use of existing datasets for training state-of-the-art solutions.

1.1 Simulated Data for Autonomous Driving

With the advances in recent years in the field of computer graphics, both in terms of photorealism and accelerated computation, simulation has been a vital method of validating autonomous models in unseen environments due to the efficiency of generating different scenarios [13]. More recently there has been added interest in the use of modern simulators for also generating the data used to train models for autonomous vehicles, both for perception and end-to-end reinforcement learning [10,11,12]. There are several advantages in generating training data through simulation. Large datasets, with diverse conditions can be quickly generated provided enough computational resources, while labeling can be fully automated with little need for supervision. Specific, difficult scenarios can be more easily constructed and advanced sensors can be added provided they have been accurately modeled. Systems such as the NVIDIA Drive Constellation [5] are pushing the boundaries for photorealistic simulation for autonomous driving using clusters of powerful NVIDIA GPUs, but is currently only available to automakers, startups and selected research institutions using NVIDIAs Drive Pegasus AI car computer, and only offers validation of models, not data generation for training. However, open source solutions based on state-of-the-art game engines such as Unreal Engine 4 and Unity, are currently in active development and offer a range of features enabling anyone to generate high quality simulations for autonomous driving. Notable examples include CARLA [1] and AirSim [4], the former of which was used for this research.

2 Simulation Toolkit

In order to facilitate the training and validation of machine learning models for autonomous driving using simulated data, the authors introduce the Carla Automated Dataset Extraction Tool (CADET), an open-source tool for generating labeled data for autonomous driving models, compatible with Carla 0.8. The tool supports various functionality including LIDAR to camera projection (Figure 1), generation of 2D and 3D bounding box labels for cars and pedestrians (Figure 2), detection of partially occluded objects (Figure 3), and generation of sensor data including LIDAR, camera and groundplane estimation, as well as sensor

calibration matrices. All labels and calibration matrices are stored in the data format defined by Geiger et al. [2], which makes it compatible with a number of existing models for object detection and segmentation. As a varied dataset is crucial for a machine learning model to generalize from a simulated environment to real-life scenarios, the data generation tool includes a number of measures to ensure variety. Most importantly, the tool resets the environment after a fixed number of samples generated. Here, a sample is defined as the tuple containing a reading from each sensor, corresponding ground truth labels and calibration data. Resetting the environment entails randomization of vehicle models, spawn positions, weather conditions and maps, and ensures a uniform distribution of weather types, agent models for both pedestrians and cars and starting positions of all vehicles. The LIDAR and camera sensors are positioned identically and synchronized such that a full LIDAR rotation exists for each image[1]. The raw sensor data is projected to a unified coordinate system used in Unreal Engine 4 before determining visible objects in the scene, and projecting to the relative coordinate spaces used in KITTI. As the initial LIDAR configuration in CARLA ignores the pitch and roll of the vehicle it is attached to, additional transformations are applied after projection such that the sensor data is properly aligned. One challenge when generating object labels is determining the visible objects in the current scene. In order to detect occluded objects, the CARLA *depth map* is utilized. A vertex is defined as occluded if the value of one of its neighbouring pixels in the depth map is closer than the vertex distance to the camera. An object is defined as occluded if at least four out of its eight bounding box vertices are occluded. This occlusion detection performs satisfactory and much faster than tracing the whole object, even when objects are localized behind see-through objects such as chain-link fences, shown in Figure 3. A more robust occlusion detection can be performed by using the semantic segmentation of the scene, but this is not implemented as of yet.

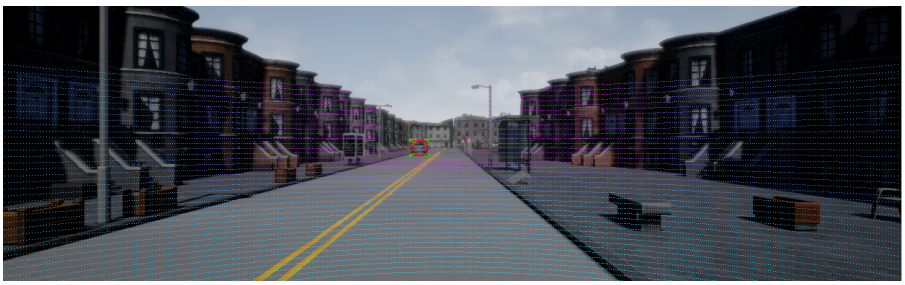


Fig. 1: LIDAR point cloud projected to image space. The color of each LIDAR point is determined by its depth value.

[1] Note that this only implies an approximate correspondance between points from the camera and LIDAR sensors.

Fig. 2: 2D (top) and 3D (bottom) bounding boxes as generated by CADET. Class labels are omitted.

Fig. 3: Occluded vertices behind a chain fence. Note that both cars are visible, and thus have a bounding box drawn around them. Occluded and visible vertices are drawn with red and green, respectively.

3 Generated Dataset

Using CADET we generate the CADET dataset, consisting of 10 000 samples. In total, there are 13989 cars and 4895 pedestrians in the dataset, averaging about 1.9 labeled objects per image. The dataset contains 2D and 3D bounding box annotations of the classes Car and Pedestrian, and contains both LIDAR and camera sensor data, as well as ground plane estimation and generation of sensor calibration matrices. The environment is generated from two maps, namely *Town01* and *Town02* in the CARLA simulator, which are both suburban environments. The distribution of objects in each image is shown in Figure 8. In comparison with the KITTI dataset, the CADET dataset has less cars and pedestrians per image, which is mostly due to city-environment of KITTI, where cars are frequently parked at the side of the road and pedestrians are present in a higher degree. The orientation of each labeled object is shown in Figure 9. We observe that the distribution of orientation have a sharp multimodal distribution with three peaks, namely for objects seen from the front, behind or sideways. Note that the pedestrians in the dataset generally have a smaller bounding box than cars, shown in Figure 7, making them harder to detect.

4 Training on Simulated Data

In order to evaluate the use of the simulated CADET dataset, as well experiment with LIDAR feature map representations, several configurations were used of the AVOD-FPN [6] architecture for 3D object detection using camera and LIDAR point cloud. The AVOD-FPN source code has been altered to allow for customized configurations by specifying the features wanted for two groups, slice maps and cloud maps. Slice maps refer to feature maps taken from each vertical slice the point cloud is split into, as specified in the configuration files, while the cloud maps consider the whole point cloud. Following the approach described in [6], two networks were used for detecting cars and pedestrians separately, repeating the process for each configuration. As multiclass detection might also produce more unstable results when evaluating per class, this was considered the better option. All models used a feature pyramid network to extract features from images and LIDAR, with early fusion of the extracted camera and LIDAR features. Training data is augmented using flipping and jitter, with the only differences between models of the same class being the respective representations of LIDAR feature maps in Bird's Eye View (BEV) as described in Section 4.1. All configurations used are available in the source code [9].

4.1 Model Configurations

AVOD-FPN uses a simplified feature extractor based on the VGG-16 architecture [14] to produce feature maps from camera view as well as LIDAR projected to BEV, allowing the LIDAR to be processed by a Convolutional Neural Network (CNN) designed for 2D images. These separate feature maps are fused together

using trainable weights, allowing the model to learn how to best combine multimodal information. In addition to what will be referred to as the default BEV configuration, as proposed in [6], two additional novel configurations are proposed for which experimental results either show faster inference with similar accuracy, or better accuracy with similar inference speed. In all cases the BEV is discretized horizontally into cells at a resolution of 0.1m. The default configuration creates 5 equally sized vertical slices within a specified height range, taking the highest point in each cell normalized by the slice height. A separate image for the density of the entire point cloud is generated from the number of points N in each cell following Equation 1, as used in [6] and [7], though normalized by $log(64)$ in the latter. We propose a simplified structure, taking the global maximum height, minimum height and density of each cell over the entire point cloud, avoiding the use of slices and halving the amount of BEV maps. We argue that this is sufficient to determine which points belong to large objects and which are outliers, and that it sufficiently defines box dimensions. For classes that occupy less space, we argue that taking three slices vertically using the maximum height and density for each slice can perform better with less susceptibility to noise, as the network could potentially learn to distinguish whether the maximum height value of a slice belongs to the object or not depending on the slice density. All configurations are visualized in Figures 4-6.

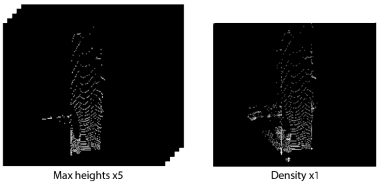


Fig. 4: Visualization of default BEV configuration, taking the maximum height within 5 vertical slices as well as the density of the full point cloud.

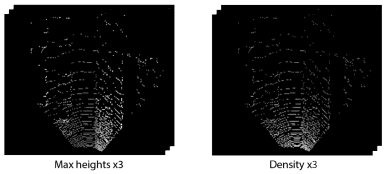


Fig. 5: Visualization of first custom BEV configuration, taking the maximum height and density within 3 vertical slices.

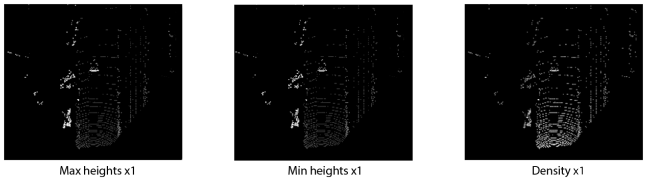


Fig. 6: Visualization of second custom BEV configuration, taking the maximum height, minimum height and density of the full point cloud.

$$min(1.0, \frac{log(N+1)}{log(16)}) \tag{1}$$

4.2 Results

In order to gather qualitative results each model trained for a total of 120k steps on the respective datasets, with a batch size of 1, as described in [6]. Checkpoints were stored at every 2k steps, of which the last 20 were selected for evaluation. Tables 1 and 2 show generated results on the KITTI dataset, for the Car and Pedestrian classes respectively, selecting the best performing checkpoint for each of the 3 BEV configurations. To measure inference speed, each model performs inference on the first 2000 images of the validation set, with learning deactivated, using a NVIDIA GTX 1080 graphics card. The mean inference time is rounded up to the nearest millisecond and presented in the tables.

Table 1: KITTI-trained model evaluated on the KITTI dataset for the Car class

		$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	Runtime (ms)	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	119	**83.46**	73.94	67.81	89.37	86.44	78.64
Max*3, Density*3	120	83.16	**73.97**	**67.98**	**89.84**	**86.62**	**79.85**
Max, Min, Density	**114**	82.98	73.92	67.84	89.62	86.61	79.68

Table 2: KITTI-trained model evaluated on the KITTI dataset for the Pedestrian class

		$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	Runtime (ms)	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	122	41.05	37.00	32.00	44.12	39.54	38.11
Max*3, Density*3	122	**45.61**	**42.66**	**38.06**	**49.16**	**45.99**	**44.53**
Max, Min, Density	**117**	27.85	27.17	24.54	33.39	33.10	29.78

Following evaluation on the KITTI dataset, all configurations were trained from scratch on the generated CADET dataset following the exact same process. Results from evaluation on the validation set of the CADET dataset can be seen in Tables 3 and 4. Note that as dynamic occlusion and truncation measurements are not included in the dataset (these are only used for post training evaluation in KITTI), evaluation does not follow the regular easy, moderate, hard categories used in KITTI. Instead objects are categorized as large or small, following the minimum height requirements for the bounding boxes of 40 pixels for easy and 25

Table 3: CADET-trained model evaluated on the CADET dataset for the Car class

	$AP_{3D}(\%)$		$AP_{BEV}(\%)$	
Method	**Large**	**Small**	**Large**	**Small**
Default	70.86	69.37	**80.13**	**71.32**
Max*3, Density*3	**70.96**	**69.59**	79.81	71.28
Max, Min, Density	68.79	60.87	78.75	70.72

Table 4: CADET-trained model evaluated on the CADET dataset for the Pedestrian class

	$AP_{3D}(\%)$		$AP_{BEV}(\%)$	
Method	**Large**	**Small**	**Large**	**Small**
Default	75.43	**73.89**	75.43	**73.91**
Max*3, Density*3	**76.13**	72.99	**80.24**	73.41
Max, Min, Density	75.49	71.82	79.73	72.37

Table 5: CADET-trained model evaluated on the KITTI dataset for the Car class

	$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	29.85	20.29	18.40	50.58	37.81	30.77
Max*3, Density*3	**35.85**	**29.40**	**24.99**	**57.32**	**49.63**	**43.25**
Max, Min, Density	30.34	24.28	20.25	45.22	37.56	31.17

Table 6: CADET-trained model evaluated on the KITTI dataset for the Pedestrian class

	$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	**9.09**	**9.09**	**9.09**	9.38	**9.33**	**9.38**
Max*3, Density*3	2.27	2.27	2.27	2.27	2.27	2.27
Max, Min, Density	**9.09**	**9.09**	**9.09**	**9.78**	9.09	9.09

pixels for moderate and hard. These models were additionally evaluated directly on the KITTI validation set, with results summarized in Tables 5 and 6.

The CADET-trained models were subsequently restored from their checkpoints at step 90k and modified for further training on the KITTI dataset. Training was resumed until step 150k, meaning the models received 60k steps of training on the KITTI training set as opposed to 120k originally. Other than increasing the amount of steps and switching the target datasets, the configura-

tion files were not altered from when training on the CADET dataset. Tables 7 and 8 show results from the top performing checkpoint of each model.

Table 7: CADET-trained model, fine-tuned and evaluated on the KITTI dataset for the Car class

	$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	**83.84**	68.67	**67.40**	**89.41**	79.77	**78.86**
Max*3, Density*3	76.85	**72.44**	66.55	88.20	**85.18**	78.71
Max, Min, Density	81.00	66.95	65.88	88.76	79.36	78.41

Table 8: CADET-trained model, fine-tuned and evaluated on the KITTI dataset for the Pedestrian class

	$AP_{3D}(\%)$			$AP_{BEV}(\%)$		
Method	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
Default	**40.26**	**38.55**	33.93	**46.96**	**44.80**	**40.73**
Max*3, Density*3	39.19	38.02	**34.15**	46.14	43.54	40.44
Max, Min, Density	37.32	34.34	32.60	45.71	42.43	37.62

5 Discussion

For the fully KITTI-trained models, results on the Car class are very similar for all configurations, where the largest loss in amounts to only 0.5% 3D AP on the easy category from the default configuration to our configuration using half as many layers in the BEV map. Larger differences are apparent for the Pedestrian class, where 3 layers is not sufficient to compete with the default configuration. However, the use of 3 slices of maximum heights and density, totalling 6 layers as with the default configuration, shows noticeably better results across the board suggesting a more robust behaviour.

Evaluation of the CADET-trained models on the CADET validation set shows similar relative performance between the models, however with the simpler custom configuration showing a dip in accuracy for the moderate category on the car class. With regards to pedestrians, differences are much smaller than what could have been expected. Also considering the unremarkable and rather inconsistent performance on the Pedestrian class of the KITTI dataset, we can likely accredit this to overly simplified representation of the physical collision of pedestrians visible in the simulated LIDAR point cloud. The CADET-trained models does perform better on the car class however, suggesting better and more consistent generalization this task.

The fine-tuned models show performance on the Car class mostly similar to the fully KITTI-trained models, however with each model showing a noticeable drop in performance on either the easy or moderate category. Results on the Pedestrian class are a bit more interesting. The default configuration sees a slight increase in accuracy on the moderate and hard category, with a slight decrease for easy. The max/density configuration sees a significant decrease in performance on all categories, where as the less complex max/min/density configuration, although still being the weakest performer, sees a significant increase in performance compared to when only trained on the KITTI dataset. The reason for the rather inconsistent results when compared to the KITTI-trained models are not thoroughly investigated, but can in part be due to somewhat unstable gradients not producing fully reliable results. The CARLA generated LIDAR point cloud does not feature accurate geometry due to simplified collision of all dynamic objects. As such the different capabilities of the configurations may not be exploited during the pretraining on the CADET dataset, impacting overall results. The simplest configuration significantly closing the gap on the Pedestrian class may be a testament to this, as the simplified pedestrian representation is more easily recognizable.

While results from simulated and partly simulated training do not generally exceed the performance of direct training on the dataset, there is a clear indication that the use of simulated data can achieve closely matched performance with less training on actual data. The ease of generation and expandabililty in terms of sensors, scenarios, environments and conditions makes tools such as CADET very useful for training and evaluating models for autonomous driving, although improvements are needed before they are sufficient for training real world solutions.

6 Conclusion

In recent years, the use of synthetic data for training machine learning models has gained in popularity due to the costs associated with gathering real-life data. This is especially true with regards to autonomous driving because of the strict demand of generalizability to a diverse number of driving scenarios. In this study, we have described CADET - a tool for generating large amounts of training data for perception in autonomous driving, and the resulting dataset. We have demonstrated that this dataset, while not sufficient to directly train systems for use in the real world, is useful in lowering the amount of real-life data required to train machine learning models to reasonably high levels of accuracy. We have also suggested and evaluated two novel BEV representations, easily configurable before training, with potential for better detection of smaller objects and reduced complexity for detection of larger objects respectively. The CADET toolkit, while still requiring improved physical models in LIDAR modelling, is currently able to generate datasets for training and validation of virtually any model designed for the KITTI object detection task.

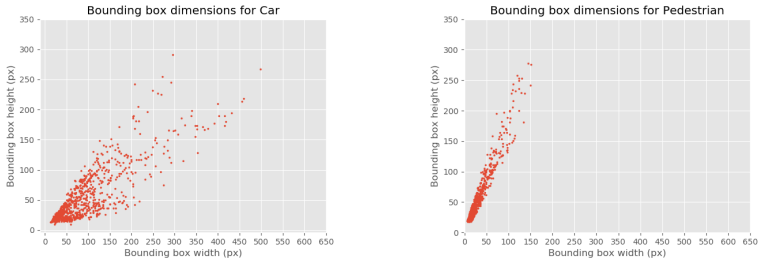


Fig. 7: Dimension of 2D bounding boxes for the classes in the CADET dataset.

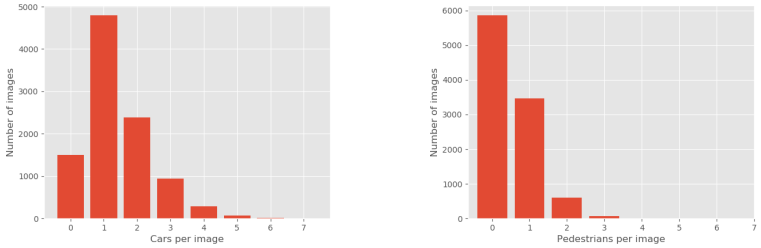


Fig. 8: Number of annotations for each class per image in the CADET dataset.

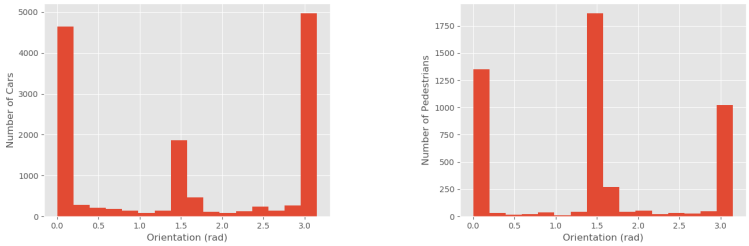


Fig. 9: Distribution of orientation per class in the CADET dataset.

References

1. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: An Open Urban Driving Simulator. In Conference on Robot Learning pp. 1–16 (2017).
2. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research, vol. 32, pp. 1231–1237, Sage Publications Sage UK: London, England (2013).
3. Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., Darrell, T.: BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling. arXiv preprint arXiv:1805.04687 (2018).
4. Shah, S., Dey, D., Lovett, C., Kapoor, A.: Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Field and service robotics pp. 621–635. Springer, Cham (2018).
5. NVIDIA Drive Constellation Homepage, `https://www.nvidia.com/en-us/self-driving-cars/drive-constellation`. Last accessed 8 April 2019
6. Ku, J., Mozifian, M., Lee, J., Harakeh, A. and Waslander, S.L.: Joint 3d proposal generation and object detection from view aggregation. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 1–8). IEEE (2018).
7. Chen, X., Ma, H., Wan, J., Li, B. and Xia, T.: Multi-view 3d object detection network for autonomous driving. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1907-1915) (2017).
8. Brekke, Å., Vatsendvik, F.: CARLA Data Export Tool (2019), `https://github.com/0zzyz/carla-data-export/`
9. Modified AVOD architecture, `https://github.com/Fredrik00/avod`. Last accessed 8 Apr 2019
10. Ravi Kiran, B., et al.: "Real-time Dynamic Object Detection for Autonomous Driving using Prior 3D-Maps." Proceedings of the European Conference on Computer Vision (ECCV). (2018).
11. Codevilla, Felipe, et al.: "End-to-end driving via conditional imitation learning." 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE (2018).
12. Gaidon, A., Lopez, A., Perronnin, F.: The reasonable effectiveness of synthetic visual data. International Journal of Computer Vision, 126(9), 899-901 (2018).
13. Schöner, H.P.: Simulation in development and testing of autonomous vehicles. In 18. Internationales Stuttgarter Symposium (pp. 1083-1095). Springer Vieweg, Wiesbaden (2018).
14. Simonyan, K. and Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

Asmund Brekke, Fredrik Vatsendvik

3D Object Detection for Autonomous Driving Using Real and Simulated Data

NTNU
Norwegian University of
Science and Technology