An Analysis and Benchmarking in Autoware.AI and OpenPCDet LiDAR-based 3D Object Detection Models

by

Samuel Yigzaw

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Light Detection And Ranging (LiDAR) sensors are widely used in applications related to autonomous driving. The ability to scan and visualize the 3D surroundings of the vehicle as a point cloud is a particular strength of this sensor. Various different object detection models have been proposed to provide bounding box predictions given a point cloud. This thesis looks at two popular, open-source frameworks which provide solutions to this problem, Autoware.AI and OpenPCDet.

The Autoware.AI framework provides models which use hand-crafted, non-neural network based methods to solve LiDAR-based object detection, while the OpenPCDet framework provides models based on neural networks. In this thesis, these models are compared with each other on a custom labeled dataset. As expected, the results of this comparison show that the non-neural network based Autoware.AI models perform significantly worse than the neural network based OpenPCDet models. Additionally, it is shown that amongst the OpenPCDet models, PV-RCNN performs better for detecting vehicles, SECOND and PV-RCNN perform better for detecting pedestrians, and SECOND and Part-A^2 Free perform better for detecting cyclists.

Acknowledgements

First and foremost, I would like to thank my supervisor, Sebastian Fischmeister, for taking me on as a master's student. My life is forever changed because of this, and for this I am eternally grateful. Second, I would like to thank Nathan Liu, who was a good friend during our undergraduate years together, and a great guide through our master's years together. Third, I would like to thank Takin Tadayon, who worked with me as part of a larger project, and who never failed to help me with anything I needed.

Dedication

This is dedicated to my family. To my father, who I would always look to for advice and counselling. To my mother, who would always give up everything to look after me. And to my younger sister, who I pray would be able to reach great heights.

Table of Contents

List of Figures

List of Tables

Abbreviations

ADAS Advanced Driver-Assistance Systems 1

ASE Average Scale Error 22

ATE Average Translation Error 22

BatchNorm batch normalization 12

BEV Bird's Eye View 5

DBL Driver Behaviour Learning 2, 26, 37

FCN Fully Connected Network 11

GT ground truth 20

IoU Intersection over Union 14

LiDAR Light Detection And Ranging 1

NMS Non-Maximum Suppression 15

ReLU Rectified Linear Units 12

RoI Regions of Interest 14

ROS Robot Operating System 6, 26

RPN Region Proposal Network 12

VFE voxel feature encoding 11

VSA Voxel Set Abstraction 18

Nomenclature

activation function A function which is used as a layer in deep learning to introduce non-linearities into the output. This is necessary for neural networks to be universal function approximators, as most functions are not linear. xiv

batch normalization A process which improves the speed and stability of deep learning, by normalizing all of the inputs to a layer within a batch. 11

centroid The central point made by averaging the positions of a set of points. 7

classification A type of machine learning task where an input is assigned one of two or more class labels. 15

downsample To decrease the resolution of data with respect to its spatial or temporal dimensions. xiv, 5

flattened Transformed into a single-dimensional array. 17

Fully Connected Network A type of neural network where in every layer, all inputs are connected to all outputs. xii, 11

ground truth The true labels for a dataset. xii, 20

Intersection over Union A metric to measure overlap between bounding boxes. It is the area or volume of the intersection of the boxes divided by the area or volume, respectively, of the union of the boxes. xii, 14

linear layer A neural network layer that outputs the product of the input with a learnable matrix. 11

Non-Maximum Suppression A technique for removing redundant bounding boxes which overlap each other. Often used at the end of an object detection network which produces too many predictions. xii, 15

normal The line perpendicular to a plane. 7

pool An operation, often done in neural networks, where a downsample is performed using a specific operator, most commonly max and average. 17

Rectified Linear Units An activation function of the form:

$$ReLU(x) = \begin{cases} 0 & x \leq 0 \\ x & 0 < x \end{cases}$$

11

Regression A type of machine learning task where an input is assigned one or more real-valued output values. 14

upsample To increase the resolution of data with respect to its spatial or temporal dimensions. 12

voxel A volumetric pixel. A 3D equivalent of a 2D pixel. An element of a voxel grid. 7

Chapter 1

Introduction

Current Advanced Driver-Assistance Systems (ADAS) are systems implemented in vehicles which help with various driving tasks, such as cruise control, lane-switching, collision warning, and parking, using sensor data from cameras, radars, Light Detection And Ranging (LiDAR), and other sensors. They are designed to improve safety and convenience for the driver. There are many different tools used in ADAS, some of which are [40]:

- Blind Spot Detection: Monitoring the blind spots in the side view mirrors.
- Rear Cross Traffic Alert: Monitoring the rear of the vehicle while it is reversing.
- Traffic Sign Assist: Monitoring the traffic signs and processing the information they contain.
- Lane Departure Warning: Monitoring the lane markings for lane departure situations, and warning the driver if the lane change seems unintentional.
- Emergency Brake Assist: Automatically breaking in dangerous situations, such as potential rear-end collisions.
- Adaptive Cruise Control with Stop & Go: Monitoring and controlling the distance to the vehicle in front, even in stop-and-go situations.

Different people have different driving styles. They may drive aggressively or defensively. They may give more or less distance between themselves and preceding vehicles. They may or may not shy away from large vehicles such as trucks. These ADAS are not personalized to

the people they assist. They are one-size-fits-all. We would like to develop a personalizable ADAS which could complete driving tasks in a manner similar to the person it is driving, which would be more comfortable for that person. The task of characterizing a driver's behaviours while driving is called Driver Behaviour Learning (DBL).

This work focuses on interpreting LiDAR sensor data for use in DBL. LiDAR data is a very important source of information for DBL. It provides a complete worldview of the surroundings of the vehicle, showing all objects as clusters of points in a point cloud. This provides the contextual information for what is happening in the environment when the driver performs certain manoeuvres.

1.1 Goal of Object Detection Frameworks

The goal of 3D LiDAR-based object detection frameworks is to produce bounding boxes which match objects which were present in the pointcloud. A pointcloud is a set of points in 3D space which corresponds to a scene. Figure 1.1 shows a point cloud representing a scene of the surroundings of a vehicle on the road. A bounding box is a 3D rectangular prism which is positioned and aligned to accurately bound, or cover, an object in a scene. Figure 1.2 shows the same scene as before, but with bounding boxes in blue covering objects that are predicted to be in the scene. The KITTI dataset [15] and others parameterize this bounding box as $(x, y, z, l, w, h, \theta)$, where x, y, z represent the location of the center of the bounding box, l, w, h represent the scale of the bounding box, and θ represents the orientation on the horizontal plane.

The problem of assigning bounding boxes to objects in a scene described by a point cloud is non-trivial. There is ambiguity over which points should be considered together as a group, especially for the clustering sub-problem that is used in the non-neural network based approaches. For neural network based approaches, learning the specific exact shape of objects will not help, since objects of various classes come in various shapes and sizes within those classes. The model should be able to handle these differences.

1.2 What is LiDAR?

LiDAR is a system which uses lasers to measure the distances of objects. Typically, multiple laser beams are swept around the surroundings, which enables creating a map of the 3D surroundings. LiDAR assumes light to travel at a constant speed, so measuring the time

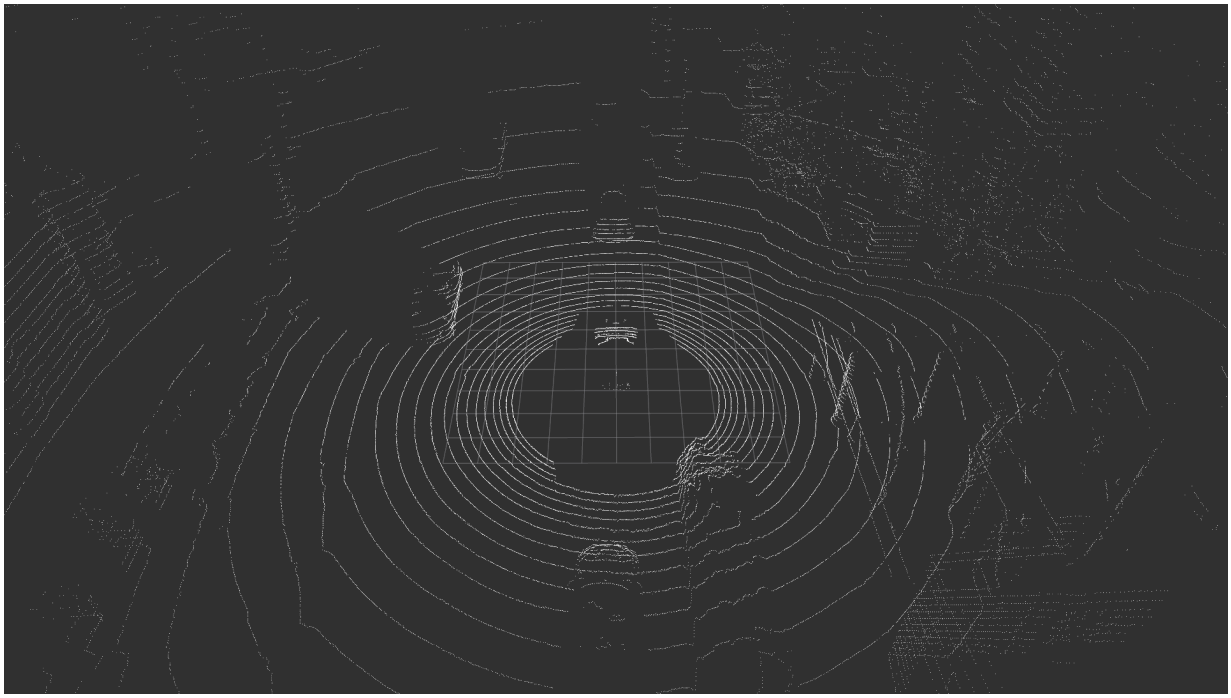

Figure 1.1: An example of a pointcloud

it takes for a beam that was just fired to reflect off of an object and return gives you the information of how far away that object is.

The LiDAR system used for this project was a Velodyne HDL-32E [12]. Its datasheet lists it as having a range of 80m-100m, with a 360° Horizontal FOV (Field of View), and a $+10.67°$ to $-30.67°$ Vertical FOV. This means that it is able to see the complete horizontal surroundings of the vehicle, while being able to see up to 10.67° above the horizontal plane and down to 30.67° below the horizontal plane. It has 32 LiDAR channels, which allows it to capture 32 data points vertically as it sweeps around horizontally.

1.3 Data Representation

The data coming from the LiDAR sensor could be represented in various different ways, depending on the part of the pipeline, from sensor to object detection model, which the data is being used in.

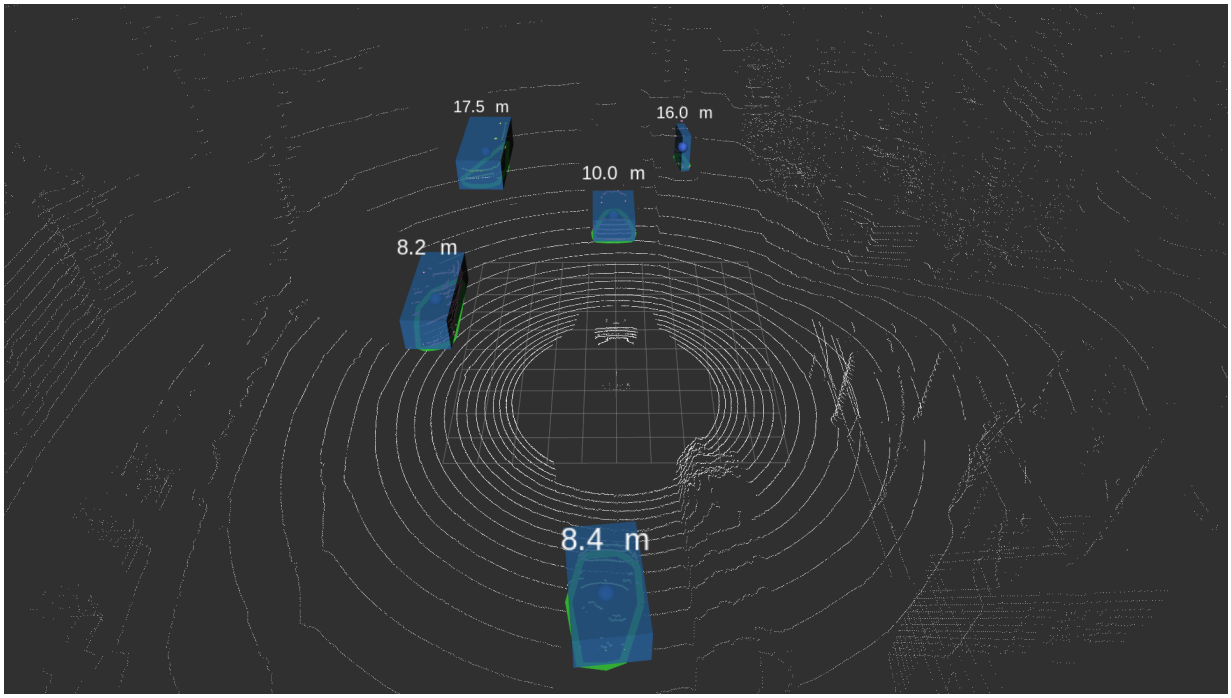

Figure 1.2: The same pointcloud, with bounding boxes placed over the detected objects

1.3.1 2D original data

The Velodyne HDL-32E, sweeps 360° with 32 lasers, which are aligned between a range of vertical angles. As the device completes a single turn, each data point has a corresponding vertical value and horizontal angle value. Additionally, the returning light gives the distance value and return intensity value. These distance values can be placed in a 2D array, with the height of the array representing the vertical location of the data point (e.g., its elevation angle), and the width of the array representing the horizontal location of the data point (e.g., its azimuth angle). In this way, each frame of data is densely represented with a 2D array. This representation is called a range image.

1.3.2 3D projection

The dense 2D representation described above has a clear bias. It has an origin at the center of the LiDAR, and objects closer to the LiDAR will appear bigger than objects further away. Furthermore, the Vertical FOV being a range of angles, rather than a range

of heights, in addition to the Horizontal FOV also being a range of angles, causes the projection to 2D to be a perspective projection. Perspective projections are distorted when the objects being imaged move tangentially to the sensor.

In order to perform object detection,it can be advantageous to normalize and remove distortions. This can be accomplished by inverse projecting the data back into 3D. This inverse projection can be done because all of the information needed is available. The position in the array of each data point corresponds to the elevation angle and azimuth angle, while the value of the data point corresponds to the distance from the LiDAR sensor. This allows for the transformation of the coordinate system from the spherical coordinate system of the perspective projection to the Cartesian coordinate system, which is free from distortions and size changes.

1.3.3 2D Bird's Eye View

The 3D Cartesian coordinate system solves the issue of distortions and size changes, but comes with another issue. The data is no longer densely represented. With the 2D representation, the size of the data per frame was $array\_length * array\_height$. In the 3D representation, each point is stored as its x, y, and z coordinates, so this turns into $3 * number\_of\_points$. Since the number of points is the same as with the 2D representation, this turns into $3 * array\_length * array\_height$, or 3 times the amount of data needed to represent the 2D representation.

Some object detection models may choose to compress this data in order to perform computations more quickly. One such compression is by using a Bird's Eye View (BEV) projection. This entails taking a downsample of the z-axis (the height dimension) from the data. The way that this downsample is performed may vary, from taking some sort of average to trying to encode each pillar of data points in some way. In the BEV projection, in contrast to a range image, not only is distortion removed, but also relevant objects (such as vehicles) typically do not overlap.

1.4 Thesis Outline

In chapter 2, a breakdown of the models that will be compared is given. The section is divided between the Autoware.AI models and the OpenPCDet models. In chapter 3, the experiment is outlined and the results of the experiment are given. In chapter 4, the thesis is concluded.

Chapter 2

Object Detection Frameworks

In this chapter, the Autoware.AI and OpenPCDet frameworks are discussed. This includes the architectures of the various object-detection models that will be tested and compared in the next chapter.

2.1 Autoware.AI

Autoware.AI [1] is an open source all-in-one autonomous vehicle framework. It has tools for perception, tracking, prediction, localization, planning, and control. The software is built on Robot Operating System (ROS), an open source middleware framework developed for use in robotics and autonomous vehicles. For the purposes of the DBL project, the focus was put on the perception capabilities. Most of the algorithms for perception are hand-crafted, meaning that they do not use neural networks. This is in contrast with OpenPCDet, which exclusively uses neural network based algorithms. For Autoware.AI, the hand-crafted perception algorithms are all based off of lidar_euclidean_cluster_detect, a hand-crafted algorithm which clusters points together based on their Euclidean distance.

2.1.1 lidar_euclidean_cluster_detect

The function lidar_euclidean_cluster_detect [7] is designed to produce clusters of points based on their Euclidean distance proximity. This is achieved in four steps, the first three of which are optional preprocessing steps, and the last of which performs the actual detection and publishing of clusters. The four steps are as follows:

Voxel-based Downsampling

Voxel-based Downsampling [3] is a downsampling method which divides the scene into a 3D grid. Each cell in the grid is a voxel. Any points inside a voxel will be averaged into a central point, called a centroid. Then all of those points are replaced with this centroid. Depending on how fine the grid is made, there is a trade off between the number of points removed and how accurate the final result will be.

A RANSAC-based ground-plane estimation algorithm

This algorithm, based on RANSAC [14] to estimate which points belong to the ground plane so that they can be removed. The algorithm estimates a plane and thresholds points by their distance to this plane. This is for the purpose of focusing the cluster generation on relevant objects.

Difference-of-Normals

Difference-of-Normals [4] [2] [18] is a final filtering method before clustering starts. First, a plane tangent to each point is estimated and its normal is taken. This estimation takes into account the neighbouring points. This procedure is done twice, once with a small neighbourhood and once with a large neighbourhood. The small neighbourhood represents the fine detail of the surface, which is susceptible to noise. The large neighbourhood represents the large-scale structure of the surface, which can overgeneralize the structure. This is shown in Figure 2.1.

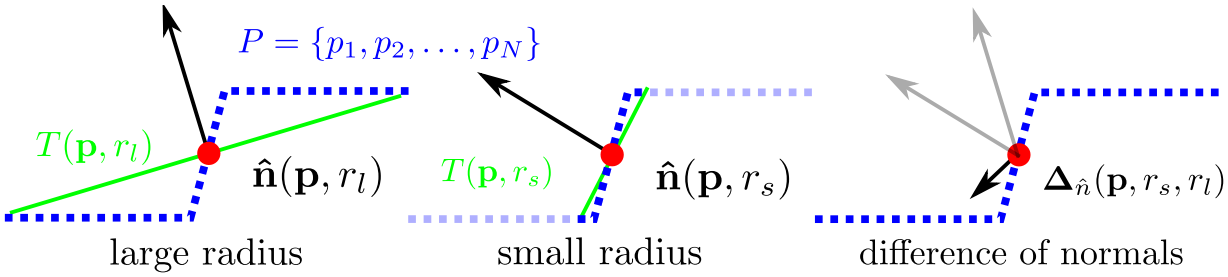

Figure 2.1: Normal estimation using small and large radii [2]

Once the normals of the tangent estimates is taken, their difference is then taken. If this difference is large, it means that the normals, and thus the estimated tangent planes,

were very different for the different scales, and that therefore, that point sits in a rough surface. If the difference is small, it means that the estimated tangent planes are very similar for the different scales, and therefore, the given point sits in a smooth surface. If it sits in a smooth surface, the point does not contribute any new information and can be removed.

Euclidean Cluster Extraction

Euclidean Cluster Extraction [5] [28] is the main clustering algorithm, after all the pre-processing is done to filter out the excess points. In this algorithm, the point cloud is converted into a kd-tree. This is a type of binary tree where at each level, the nodes specify a value along a dimension, and it's children are split across that value. This provides a data structure to perform an approximate but efficient nearest neighbours search with. The algorithm used is [28]:

1. create a kd-tree representation for the input point cloud dataset P;
2. set up an empty list of clusters C, and a queue of the points that need to be checked Q;
3. then for every point $p_i \in P$, perform the following steps:
   - add p_i to the current queue Q;
   - for every point $p_i \in Q$ do:
     - search for the set P_i^k of point neighbors of p_i in a sphere with radius $r < d_{th}$;
     - for every neighbor $p_i^k \in P_i^k$, check if the point has already been processed, and if not add it to Q;
   - when the list of all points in Q has been processed, add Q to the list of clusters C, and reset Q to an empty list
4. the algorithm terminates when all points $p_i \in P$ have been processed and are now part of the list of point clusters C.

This list of clusters is then returned as a list of point clouds. This algorithm serves as the foundation upon which the later algorithms are built.

2.1.2 lidar_shape_estimation

Function lidar_shape_estimation [9] implements this paper [38]. The algorithm described in this paper takes in point clusters, generated from lidar_euclidean_cluster_detect, and fits rectangles around each cluster. This is done while ignoring the height dimension, so only two dimensions are taken into account. The algorithm works by trying to fit rectangles oriented at a sequence of angles to the cluster, and then evaluating the fit using a criterion. At each angle orientation, the points are projected onto two perpendicular lines which represent the rectangle edges. One criterion that may be used is area. In this criterion, the length of the spread of points on each projection are multiplied together. The minimal value using this metric corresponds to the rectangle orientation that was the tightest fitting on the point cluster. Once this optimal value is chosen, the side lengths of the rectangle are chosen to fit the cluster, and together with the orientation, are returned.

2.1.3 lidar_naive_l_shape_detect

Function lidar_naive_l_shape_detect [8] implements this paper [27]. This is an alternate algorithm which does the same task as the previous lidar_shape_estimation. It takes in point clusters generated from lidar_euclidean_cluster_detect and fits rectangles around each cluster. This also ignores the height dimension. The process is shown in Figure 2.2. The way this algorithm works is by first selecting the two furthest points in a cluster. These are then assumed to be the two opposing corners of the object that the cluster is representing. Next, a line is drawn to connect these two points, and every point in the cluster has its minimum distance from this line measured. The point which is furthest from this line is assumed to be the third corner, the one nearest to the LiDAR sensor. Rotating this point around the center of the line will then give the final fourth corner.

2.2 OpenPCDet

OpenPCDet [35] is an open source 3D point cloud object detection framework. It hosts code bases for several models, and allows for training and testing. The code for each model is heavily refactored in order for them to fit within a modular framework. This structure is shown in Figure 2.3. This is possible because many of the different models are based off of each other. In particular, PointPillars [21] takes from SECOND [36], and PV-RCNN [30] takes from Part-A^2 [33], which itself takes from PointRCNN [32] and SECOND. In addition to this, Shaoshuai Shi was the lead author of the papers for PointRCNN, Part-A^2,

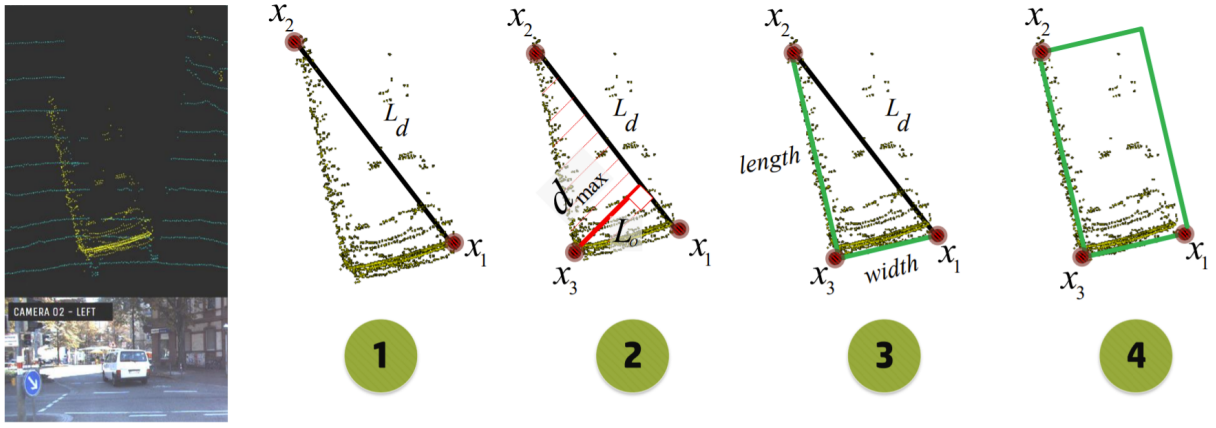

Figure 2.2: The step-by-step l-shape fitting process [27]

and PV-RCNN, as well as the lead contributor to OpenPCDet. OpenPCDet is the official repository for the codebases of those papers authored by Shaoshuai Shi. All of these models were trained on the KITTI dataset [15]. KITTI's LiDAR dataset only has points in front of the vehicle, so these models were all trained to look for objects in the front half of the vehicle.

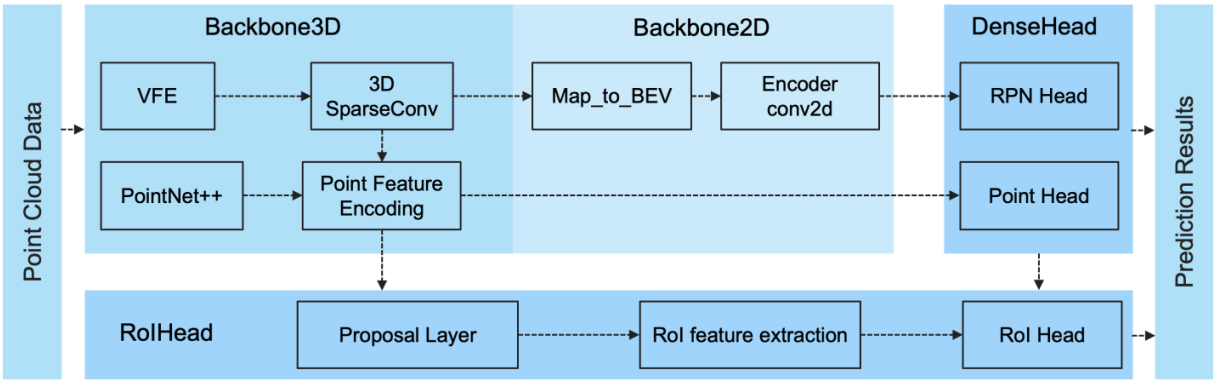

Figure 2.3: The generalized, modular structure of the OpenPCDet framework

2.2.1 SECOND

SECOND [36] stands for Sparsely Embedded CONvolutional Detection. It divides the scene into voxels, encodes the voxel features, performs sparse convolutions, and then uses a

region proposal network to provide the final bounding box. It's refactored implementation in OpenPCDet is shown in Figure 2.4. The original paper divided SECOND into the following four sections:

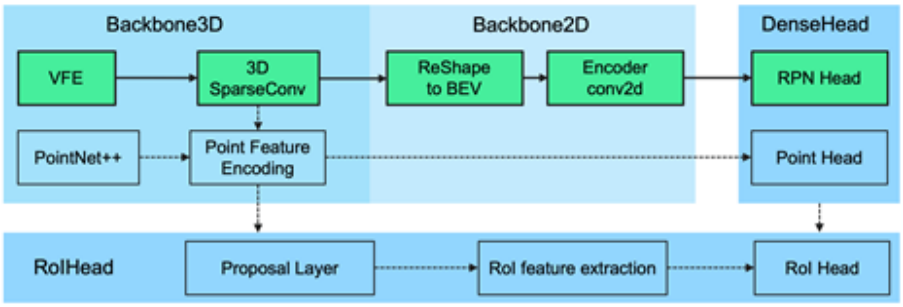

Figure 2.4: Implementation of SECOND in the OpenPCDet framework

Point Cloud Grouping

The point cloud is divided into a 3D voxel grid. Voxels with points in them are saved with their location in space, and the number of points in them is stored. A limit is placed on the total number of voxels.

Voxelwise Feature Extractor

This is taken from VoxelNet [39]. A voxel feature encoding (VFE) layer is used to extract voxelwise features. A VFE is implemented by a Fully Connected Network (FCN). For every point within a voxel, the point is passed through a linear layer, a batch normalization [19] layer and a Rectified Linear Units [23] layer, outputting point-wise features. These point-wise features are max-pooled together within their voxels to create the voxel-wise features. Two of these VFE layers are used in series, followed by an FCN layer.

Sparse Convolutional Middle Extractor

This part is based off of Spatially-sparse convolutional neural networks [16] and Submanifold Sparse Convolutional Networks [17]. The idea here is that a 3D CNN will be used to produce features from the Voxelwise Feature Extractor, but a regular 3D CNN would run very slowly, because it would scale to the cube of the size of the scene. However, most

of the space in a scene is empty, as it is essentially an inverse projection of the originally 2D LiDAR sensor data. To take advantage of this emptiness, a new kind of 3D convolutional network, a Sparse Convolutional Network, is used. This network uses several sparse convolutional layers, which work as follows:

1. The input sparse voxels are indexed, collected together, and along with all the output voxels which could be reached with the convolutional kernel, are put in a matrix table called Rule.

2. These output voxels are not unique, since two nearby input voxels could reach the same output voxel, depending on the size of the kernel. The output voxels are uniquely indexed, meaning that each output voxel located in space is assigned a unique index value.

3. The entries in Rule are then assigned these output indices. Rule is now a dense representation of the data, and parallelizable operations on a GPU can be performed with it.

In the Sparse Convolutional Middle Extractor, several layers of normal and submanifold sparse convolutional layers are used to downsample the z-axis and reduce the dimensions of the data to 2D. The difference between normal and submanifold sparse convolutional layers is in which output voxels are affected by an input voxel. In the normal case, any output voxel reachable by the convolutional kernel is affected, whereas in the submanifold case, only output voxels which are active (i.e. have input data in their location) are affected. Submanifold layers are used to not spread the data out and increase the number of active voxels. Finally, the sparse data is converted into dense 2D feature maps ready for the Region Proposal Network.

Region Proposal Network (RPN) Detection Head

This part is based on SSD [22]. Several stages of multiple blocks of convolution, batch normalization (BatchNorm), and Rectified Linear Units (ReLU) layers are applied. In each block, the first convolutional layer downsamples the input using an appropriate stride. Each stage does this downsampling to a different degree. At the end of each stage, a block containing a deconvolutional layer is used instead, to upsample the data back up to what it was originally. This has the effect of making the different stages find features of various sizes in the data. The final data from each stage is concatenated together to produce the output bounding boxes.

2.2.2 PointPillars

PointPillars [21] is an algorithm which divides the scene into columns of data points (pillars) in order to encode each pillar, turning the 3D scene into a BEV image, which can then be passed through a regular image object detection network to find bounding boxes. It's refactored implementation in OpenPCDet is shown in Figure 2.5. The original paper divided PointPillars into the following three sections:

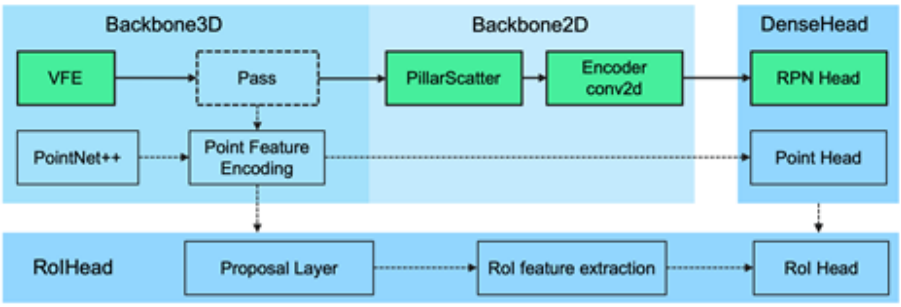

Figure 2.5: Implementation of PointPillars in the OpenPCDet framework

Pillar Feature Net

This divides the scene into pillars which are then encoded, creating a BEV pseudo-image. First, each point in the pillar is augmented with additional information. It starts with x, y, z, and reflectance r. These values come from the LiDAR device. Then, x_c, y_c, z_c, where the subscript c denotes the distance to the centroid of the points in the pillar, are added. Lastly, x_p and y_p are added, where the subscript p denotes distance to the central line of the pillar itself. There are $D = 9$ data points per LiDAR point. In the overall scene, there is a limit P placed on the number of pillars, and a limit N placed on the number of points per pillar. If the number of points in a pillar exceeds N, a random sample is taken. If it falls short, it is padded with zeros. The overall scene is described in an array of size (D, P, N).

The following part is similar to the VFE described in the SECOND section, except that it is performed on pillar voxels instead of cubic voxels. For each point of size D, a simplified version of PointNet [24] is used, followed by a layer of BatchNorm and ReLU, to encode the data into an array of size C. Finally, a max operation is done over the N points in a pillar to arrive at the final data for a scene being of size (C, P). Each pillar, which corresponds to a location in 2D space, now has a data point of size C associated

with it. The pseudo-image is now of size (C, H, W), where H and W are the height and width of the pseudo-image.

Backbone (2D CNN)

The backbone processes the pseudo-image into a high-level representation. It is similar to what was used in VoxelNet [39]. It consists of two parts, a feature extractor network which produces features at lower and lower resolutions with respect to the image space, and a second network which upsamples and concatenates the features from the feature extractor. The feature extractor consists of a series of blocks which perform convolution with a number of $3 * 3$ 2D convolution layers. After the convolutions, BatchNorm and ReLU are applied. The whole block takes the input with a stride of S. This results in each block lowering the resolution of its features by a factor of S, allowing it to pay attention to larger-scale features.

The second network takes the outputs of each of these blocks and upsamples them all to the size of the original pseudo-image. BatchNorm and ReLU are again applied to these outputs. All of these are then concatenated together for the final detection head to use.

Detection Head

This uses the high-level representation to find bounding boxes. Similarly to SECOND, this uses SSD [22] to produce the 3D bounding boxes. The boxes are matched to the ground truth in the horizontal plane using 2D Intersection over Union (IoU). Regression is then performed for the height and vertical displacement.

2.2.3 PointRCNN

PointRCNN [32] is a 2-stage algorithm for generating 3D bounding boxes. The first stage is a bottom-up 3D proposal generator which segments the point cloud into Regions of Interest (RoI). The second stage refines the proposals to generate the final bounding boxes. Its implementation in OpenPCDet is shown in Figure 2.6.

Bottom-up 3D proposal generation

Pointwise features are learned using PointNet++ [25]. From these features, the foreground points are segmented out. Foreground points are defined as points which would belong

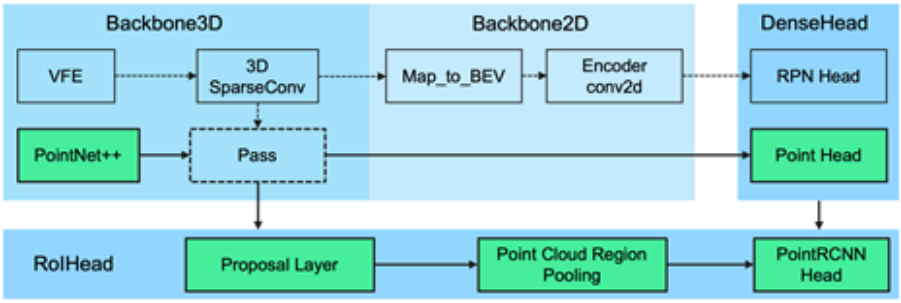

Figure 2.6: Implementation of PointRCNN in the OpenPCDet framework

to a bounding box, no matter the label. Thus, to train this part of the network, the ground-truth bounding boxes are all that is required.

Next, bin-based 3D bounding box generation is performed. In this step, for each point in a segment, a bounding box is calculated. First, the centroid of the bounding box is calculated using a combination of classification and regression. Classification is done first by binning the points along the horizontal plane, and the regression is done within those bins. Along the vertical axis, only regression is used. Orientation is also calculated with classification and regression, just as with the horizontal center. Regression is used on the object's dimensions. Finally, Non-Maximum Suppression (NMS) is used to select the best bounding box for each segmented object.

Point cloud region pooling

This stage is meant to refine the bounding boxes from the previous stage. First, for each bounding box proposal, an enlarged version is created, and every point within this enlarged bounding box is taken, whether the point is foreground or background. These points contain their location, intensity, foreground/background status, and the learned features from PointNet++.

Next, for each bounding box, the points are transformed into a coordinate system aligned with the bounding box. This means that the orientations and centroid locations for each bounding box are zero within their own coordinate systems. This is called a canonical transformation. These points are then fed through several fully-connected layers before they are concatenated with the bounding box proposal data and fed into PointNet++ again.

From here, bin-based 3D bounding box generation is performed in almost the same way

as described in the Bottom-up 3D proposal generation section above. One change is that coordinates of the ground-truth bounding boxes are transformed to match the coordinate system for each bounding box. This then produces the final 3D bounding boxes.

2.2.4 Part-A^2

Part-A^2 [33] builds upon the work from PointRCNN. It consists of two main stages, a part-aware stage and a part-aggregation stage. The part-aware stage learns to predict segmentation proposals, as well as intra-object part locations, which are the locations of each point relative to the overall object. The training information for this comes freely with the orientation of the ground-truth bounding boxes. The part-aggregation stage learns to aggregate the parts learned in the previous stage to fine-tune the segmented proposals. It's implementation in OpenPCDet is shown in Figure 2.7.

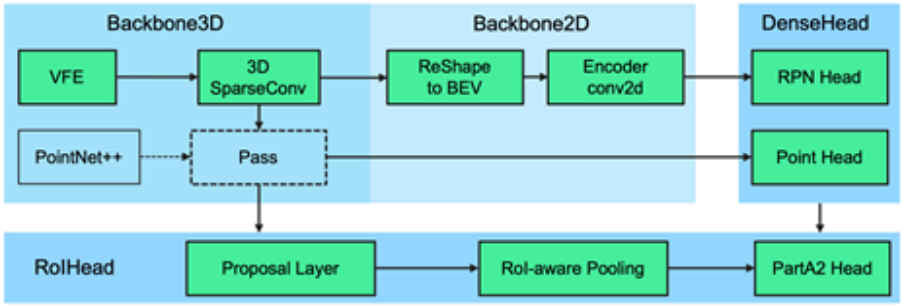

Figure 2.7: Implementation of Part-A^2 in the OpenPCDet framework

Part-aware stage

This part starts with 3D sparse convolution like SECOND. This is used as the initial encoder-decoder network, instead of PointNet++ as used in PointRCNN. As with PointRCNN, foreground points are then segmented out. Next, intra-object part locations are calculated. For each point, it's relative position within the ground-truth bounding box is learned. It is normalized across the size and orientation of the bounding box, so that all points per bounding box are within the range of $[0, 1]$, with the center being at $[0.5, 0.5, 0.5]$.

Lastly, two different approaches are tested to perform proposal generation, which segments out proposals for each bounding box. An anchor-free approach is essentially the approach used in PointRCNN to perform proposal generation. Additionally, there is an

anchor-based approach. In this approach, the downsampled output of the initial encoder is taken. This output is in a 2D BEV format. Similar to SECOND, an RPN then takes this input in. This RPN has two predefined 3D anchors per class per location on the horizontal plane, one aligned with the x-axis and one aligned with the y-axis. An anchor is a predetermined bounding box which serves as a starting point for regression. They are associated with the ground truth bounding boxes using IoU. The anchor-free method is more memory efficient, since it does not require filling the space with predefined anchors, but the anchor-based method has a slightly higher recall rate.

Part-aggregation stage

Canonical transformations, as described in the PointRCNN section are applied to the 3D bounding box proposals generated by the anchor-free or anchor-based approaches, the encoder-decoder output, as well as the intra-object part locations and semantic segmentations. Following this, the 3D bounding box proposals are segmented into voxels. An average-pool is then performed on the intra-object part location points located within each voxel, as well as a max-pool on the encoder-decoder output within each voxel. Average pooling is done on the intra-object part location points so that the average location data within a voxel can be retained, whereas max-pooling for the encoder-decoder output retains the most prominent features within a voxel. These two outputs are then brought together and passed through a sparse convolutional network with several sparse convolutional layers. The outputs are then flattened and passed through an FCN, outputting the final bounding boxes ready for scoring and refinement by regression using IoU with the ground truth bounding boxes.

2.2.5 PV-RCNN

PV-RCNN [30] builds on top of Part-A^2. It aims to bring together the more accurate contextual information of point-based methods with the greater efficiency and higher-quality 3D object proposals of voxel-based methods. There are three steps to this approach. First, a 3D voxel CNN is used to extract features and generate proposals, which is similar to previous methods. Next, a voxel-to-keypoint step summarizes the voxels into a few feature keypoints. Lastly, a point-to-grid step aggregates the feature keypoints to RoI grids for scoring and refinement. It's implementation in OpenPCDet is shown in Figure 2.8.

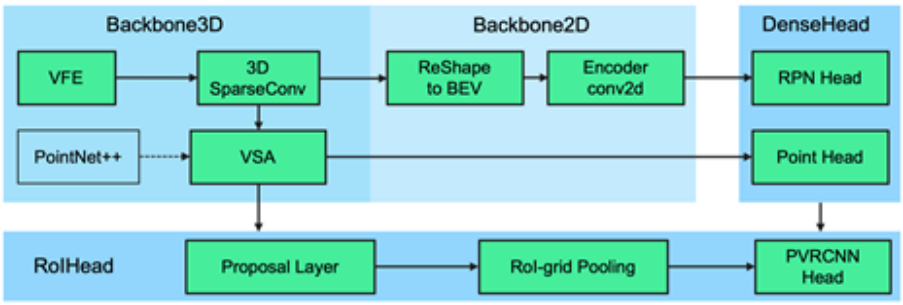

Figure 2.8: Implementation of PV-RCNN in the OpenPCDet framework

3D Voxel CNN

The scene is divided into voxels and the coordinates and reflectance values of the points within each voxel are average-pooled. Several sparse convolutional layers then downsample the data by factors of 2. The output is then turned into a BEV, and the anchor-based proposal generator from Part-A^2 is used to produce proposals.

Voxel-to-keypoint Scene Encoding via Voxel Set Abstraction

A number n of keypoints is selected from the scene using the Farthest Point Sampling algorithm, which is an algorithm that selects the n points which are furthest from each other in a sample. A process called Voxel Set Abstraction (VSA) is then used to aggregate the voxel features for each keypoint. For each keypoint, the voxel outputs of each separate downsampling level of the 3D Voxel CNN are taken. For each level, all of the non-empty voxels within a specified distance to the keypoint are selected, and their feature data and coordinates are taken. PointNet is then used on this to transform this data into a feature vector for the keypoint. This occurs for multiple downsample layers, as well as for multiple distance values, which gathers data on multiple scales and resolutions.

This data is enriched by adding to each set of keypoint features the data from the original point from the raw point cloud as well as the bilinearly-interpolated data from the nearest surrounding voxel values from the final 3D Voxel CNN downsampled output. Finally, these keypoints correspond to either foreground or background points, as defined in the PointRCNN section. For the next step, which conducts proposal refinement, the foreground points are more important. Because of this, the points which are predicted to be foreground points are weighed more heavily than the ones predicted to be background

points. This is trained, as in the PointRCNN section, using the unambiguous and well separated ground-truth bounding boxes.

Keypoint-to-grid RoI Feature Abstraction for Proposal Refinement

The 3D bounding box proposals generated by the 3D Voxel CNN are taken here. For each proposal, a set of uniformly sampled voxels is taken, with dimensions $6 \times 6 \times 6$. A set abstraction process similar to the VSA is used in order to aggregate the keypoint features for each sampled voxel. For each voxel, keypoints within a specified distance to the voxel have their features and relative coordinates taken and passed into PointNet to be transformed into a single feature vector per voxel. This also occurs using the keypoints generated using the multiple downsample layers, as well as for multiple distance values. Finally, the features for all the sampled voxels within the proposal are aggregated, and then, as with Part-A^2, are flattened and passed through an FCN, outputting the final bounding boxes ready for scoring and refinement by regression using IoU with the ground truth bounding boxes.

Chapter 3

Experiments

In this chapter, the entirety of the experiment is discussed. This includes the metrics used to quantify and compare the performances of the models, the variables associated with this experiment, the experimental setup, the integrity assessment, the comparison method, and the final results.

3.1 Metrics

When doing object detection, the model produces predictions, which are compared against the ground truth (GT).

	GT: Yes	GT: No
Prediction: Yes	True Positive (TP)	False Positive (FP)
Prediction: No	False Negative (FN)	True Negative (TN)

Precision is the proportion of predicted objects that were correct. It is calculated as:

$$\frac{TP}{TP + FP}$$

Recall is the proportion of GT objects that were predicted. It is calculated as:

$$\frac{TP}{TP + FN}$$

3.1.1 Average Precision (AP)

In object detection, predicted bounding boxes are almost always off from the GT, since they are supposed to predict location. In order to determine whether a prediction matches with the GT, a metric may be used, such as IoU or distance between centroids. If they meet a specific threshold, then the prediction is counted as a TP, and if not, then it is counted as a FP. The nuScenes [11] detection benchmark uses the following metric:

> "mean Average Precision (mAP): We use the well-known Average Precision metric, but define a match by considering the 2D center distance on the ground plane rather than intersection over union based affinities. Specifically, we match predictions with the GT objects that have the smallest center-distance up to a certain threshold. For a given match threshold we calculate average precision (AP) by integrating the recall vs precision curve for recalls and precisions > 0.1. We finally average over match thresholds of $\{0.5, 1, 2, 4\}$ meters and compute the mean across classes." [11]

The detection benchmark used for the experiments here, called 3D LiDAR BBox Detection Evaluation [20], works the same way as with nuScenes, except that it uses a single threshold, and it keeps the class results separate instead of taking their mean. Thus, it uses Average Precision instead of mean Average Precision.

Neural Network based classification predictions come with confidence scores that show the model's confidence that it's prediction was correct. As higher confidence scores strongly correlate with higher accuracy, only those predictions with a confidence score above a certain threshold are used. This threshold is a parameter which can be adjusted. If the threshold is raised, the precision will be raised, since there will be less FP, but the recall will be lowered since there will be more FN. If the threshold is lowered, the recall will be raised, since there will be less FN, but the precision will lowered since there will be more FP. The better the model, the less FP and FN. This means that as the threshold is decreased, a better model will be able recall more while retaining as much precision as possible. Graphing the parametric function of precision vs recall, with the threshold being the parameter, the larger the area under the curve, the better the model. This area under the Precision-Recall curve is the Average Precision (AP). Higher AP means that the model performs better.

For models that do not use neural networks, such as for the Autoware.AI models, a confidence score is not provided. This undermines the ability to calculate a precision-recall curve, and thus AP. To remedy this, a proxy to confidence score was used. This proxy was

the Euclidean distance from the origin. The Euclidean distance from the origin, just like confidence score, is strongly correlated with higher accuracy, as closer objects are more likely to get correctly labelled than farther objects. Using this metric, the precision-recall curve and AP were both calculated.

3.1.2 F_β Score

This metric is similar to AP in that they both use precision and recall for their calculation. With a given threshold, the precision and recall can be calculated, and from there, the F_β score is given as:

$$F_\beta = (1 + \beta^2)\frac{precision * recall}{\beta^2 * precision + recall}$$

The β parameter controls which factor will contribute more to the overall score. As $\beta \longrightarrow \infty, F_\beta \longrightarrow recall$, and as $\beta \longrightarrow 0, F_\beta \longrightarrow precision$. Higher F_β score means that the model performs better. In this experiment, F_1 score is used, treating precision and recall as both equally important to the score. The maximum F_1 score across all threshold values is used.

3.1.3 True Positive metrics

nuScenes uses a set of additional metrics for TP which are matched with a threshold distance of 2m from the center of the prediction to the center of the GT. 3D LiDAR BBox Detection Evaluation also computes those metrics which specifically measure translation and scale errors:

- Average Translation Error (ATE): Euclidean center distance in 2D in meters. In 3D LiDAR BBox Detection Evaluation, a 3D version is also used.
- Average Scale Error (ASE): Calculated as $1 - IOU$ after aligning centers and orientation.

These metrics separate the errors into those of translation and scale, which provides deeper insights into how models perform with these different parts of bounding box prediction. Lower errors mean that the model performs better.

3.2 Parameters, Factors, Levels

Parameters are elements in the experiment that are constant. These were necessary parts of setting up the experiment and processing the data. Factors are elements of the experiment that are to be changed, with the levels being the various different settings the factors can be changed to.

3.2.1 Parameters

Table 3.1 shows the software parameters involved in this experiment. Autoware.AI and OpenPCDet are the object detection frameworks that will be examined. These are listed here as parameters because although they contain multiple models each which are considered as factors, the frameworks themselves are constant, unchanging codebases that will be used within the experiment. SpConv, short for Spatially Sparse Convolution Library, is a dependency for OpenPCDet, allowing the models that do sparse convolutions, such as SECOND, to work. MATLAB Lidar Labeler is the software used to label the ground truth data. Table 3.2 shows the hardware parameters involved in this experiment.

Table 3.1: Software parameters

Software Name	Version	Source
Autoware.AI	1.13.0	https://github.com/Autoware-AI/autoware.ai/releases/tag/1.13.0
Forked OpenPCDet	0.3.0+829d99c	https://github.com/syigzaw/OpenPCDet
SpConv	1.2.1	https://github.com/traveller59/spconv/tree/v1.2.1
MATLAB Lidar Labeler	R2021b	https://www.mathworks.com/help/lidar/ref/lidarlabeler-app.html
ROS	Melodic	http://wiki.ros.org/melodic

Table 3.2: Hardware parameters

Type	Model
LiDAR Sensor	Velodyne HDL-32e [12]
Participant Vehicle	Lexus RX-450h

3.2.2 Factors and Levels

The factors are the model type and the models themselves. Table 3.3 shows the levels for these factors. PointRCNN IoU is the same as PointRCNN, but uses the IoU metric for predicting the confidence of each box. Part-A^2 Anchor is Part-A^2 with the anchor-based approach, and Part-A^2 Free is Part-A^2 with the anchor-free approach.

Table 3.3: Factors and levels

Model type	Framework
Non-neural network based	Autoware.AI
Neural network based	OpenPCDet
The models	**Framework**
lidar_naive_l_shape_detect	Autoware.AI
lidar_shape_estimation	Autoware.AI
PointPillars	OpenPCDet
SECOND	OpenPCDet
PointRCNN	OpenPCDet
PointRCNN IoU	OpenPCDet
Part-A^2 Anchor	OpenPCDet
Part-A^2 Free	OpenPCDet
PV-RCNN	OpenPCDet
Classes	**OpenPCDet**
N/A	Autoware.AI
Vehicles	OpenPCDet
Pedestrians	OpenPCDet
Cyclists	OpenPCDet

3.3 Experimental Setup

This section discusses the process by which the data was collected and transformed for use in the experiment. Figure 3.1 shows a visual overview of the entire data processing pipeline.

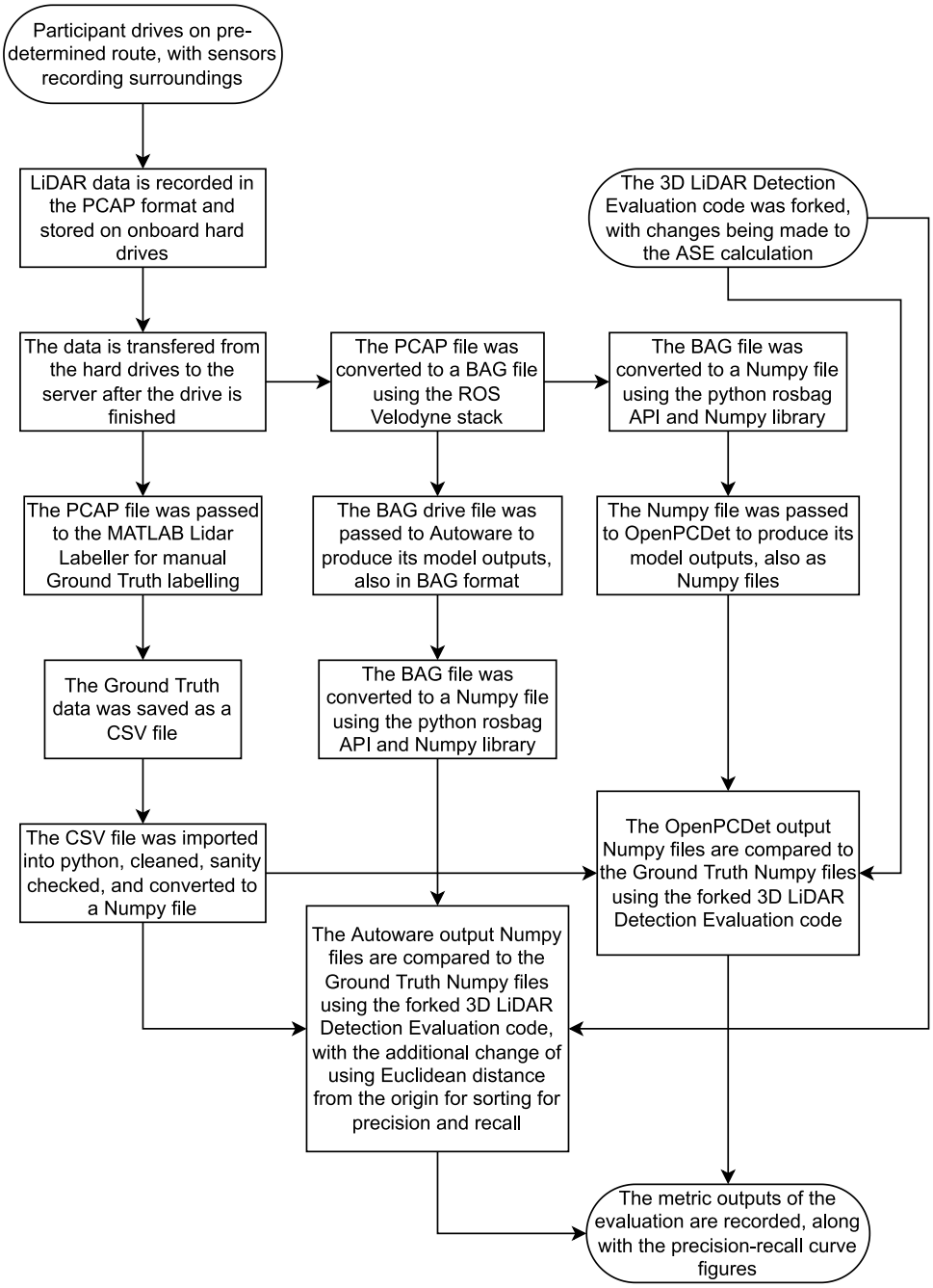

Figure 3.1: Pipeline for processing data from data generation to final results

3.3.1 Data Collection

This section discusses how participants were used to collect drive data and how that was handled and processed for the DBL project that was the basis of this thesis.

Participant Drives

Participants with valid Ontario G or equivalent driving license who are willing to participate in the study are found. The total commitment time is under 4 hours, with approximately 3 hours being dedicated to driving an approximately 1.5 hour pre-determined route twice. The participants complete a safety orientation and a vehicle orientation, and a preliminary test drive to make sure they are fit to proceed.

While driving on the route, a researcher will be accompanying in the front-passenger seat. The researcher will observe and annotate any noteworthy circumstances, such as weather conditions, emergency vehicles, accidents, etc.

Additionally, various sensors will capture the surrounding environment: a forward facing radar sensor, two rear facing radar sensors, a lidar sensor mounted on the roof of the vehicle, two forward facing video cameras, and a forward facing thermal imaging camera.

Data Handling

The LiDAR data is saved as a PCAP (Packet CAPture) file. This file can be read with various software, such as with MATLAB for labeling the GT, or with the ROS Velodyne stack. The PCAP files are stored on a secured server accessible over FTP. The MATLAB Lidar Labeller was used to create GT labels for one specific PCAP file. This file had 17,919 frames, which covers a duration of 29 min 51.9 sec.

ROS is an open source middleware robotics framework that facilitates communications between different modules in a system. It uses a publisher/subscriber model, where modules publish data to a topic and other modules listen to that data by subscribing to that topic. This allows for modular message passing and communication.

The ROS Velodyne stack is used to convert the PCAP file into the ROS-specific BAG file type, as well as to pass that data to Autoware.AI to use. The data outputted by Autoware.AI is also stored as BAG files. The bag files are then read using the ROS Python library in order to convert the data into Numpy files. Each file is used to store a single frame of data. The Numpy file version of the LiDAR data is then passed on to OpenPCDet to use with its models.

The ground truth data was then cleaned. There were instances where the bounding boxes were malformed in various ways. Sometimes, there were bounding boxes that appeared in a frame and disappeared in the next. These bounding boxes did not correspond with any actual object. These were removed, as they were not temporally consistent. Additionally, for each feature, a window five frames wide was passed along the time axis. There were instances where certain values were incorrectly labelled either zero or a standard deviation away from it's window of values. These values were replaced with the average of the window, this average not consisting of itself in the case that it is a zero value. This process smooths out incorrect outliers and removes incorrect zero values at the same time, removing temporal inconsistencies in the ground truth labelled data.

3.4 Integrity Assessment

The GT LiDAR data was converted from the MATLAB format to Numpy files. These files were sanity checked to make sure there were no issues. One sanity check was to make sure that no frames were missing. The frequency of the frames is 10 Hz, so the time intervals between each frame should be around 100 ms. After running the sanity check, Figure 3.2 was generated to show the intervals. Table 3.4 shows the output of calling the *.describe()* method of the Pandas DataFrame containing the data. It shows a summary of the statistics related to the data, most notably, the min, max, mean, and std.

Table 3.4: Statistical summary of the data of the time intervals (ms) between frames

count	mean	std	min	25%	50%	75%	max
17918.0	100.000245	0.219825	99.53	100.08	100.08	100.09	100.65

3.4.1 Algorithm Comparison Method

After the GT data was sanity checked, it was passed on to a forked version of 3D LiDAR Detection Evaluation code [20]. Since the OpenPCDet models were trained on the KITTI dataset, they were only trained to detect objects in front of them. In order to evaluate them on the GT data, the 3D LiDAR Detection Evaluation code was also modified to only consider GT data in the front half of the vehicle when evaluating the OpenPCDet models. When evaluating the Autoware.AI models, classes were not considered, since the Autoware.AI models cannot differentiate between classes. Additionally, the sorting for the

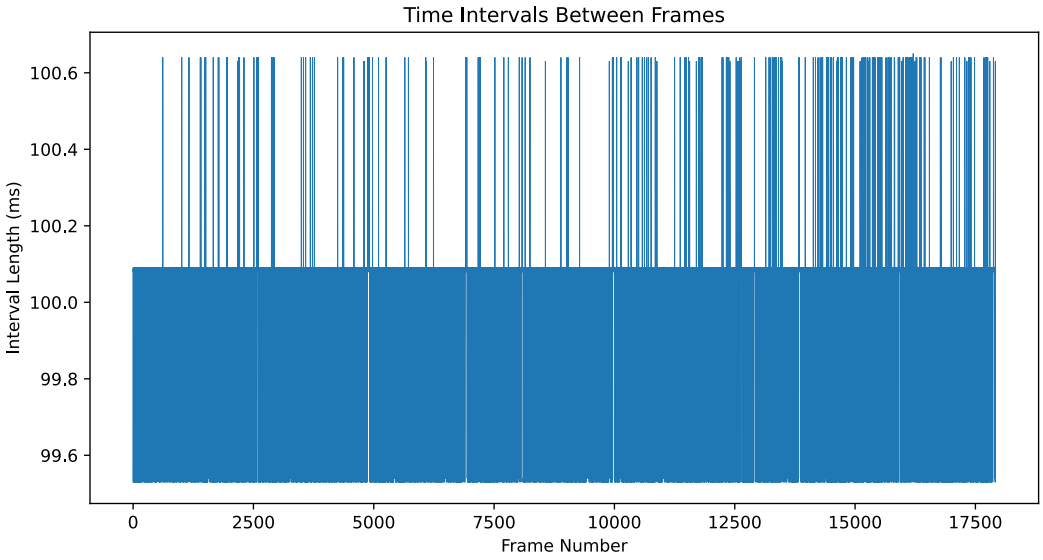

Figure 3.2: Graph of sanity checking time intervals between frames

AP measurement was done on the Euclidean distance from the origin, as the Autoware.AI models do not provide a confidence score. The evaluation produces AP, F_1 score, Average 2D Translation Error (A2TE), Average 3D Translation Error (A3TE), and ASE.

3.5 Results

Below are the results of the experiment performed on the models, split by framework.

3.5.1 Autoware.AI

Table 3.5: Autoware.AI Model Evaluations

Model	AP	F_1 Score	A2TE $[m]$	A3TE $[m]$	ASE $[m^3]$
lidar_naive_l_shape_detect	**0.133**	**0.340**	1.0903	**1.1391**	**0.6940**
lidar_shape_estimation	0.126	0.335	**1.0841**	1.2048	0.8771

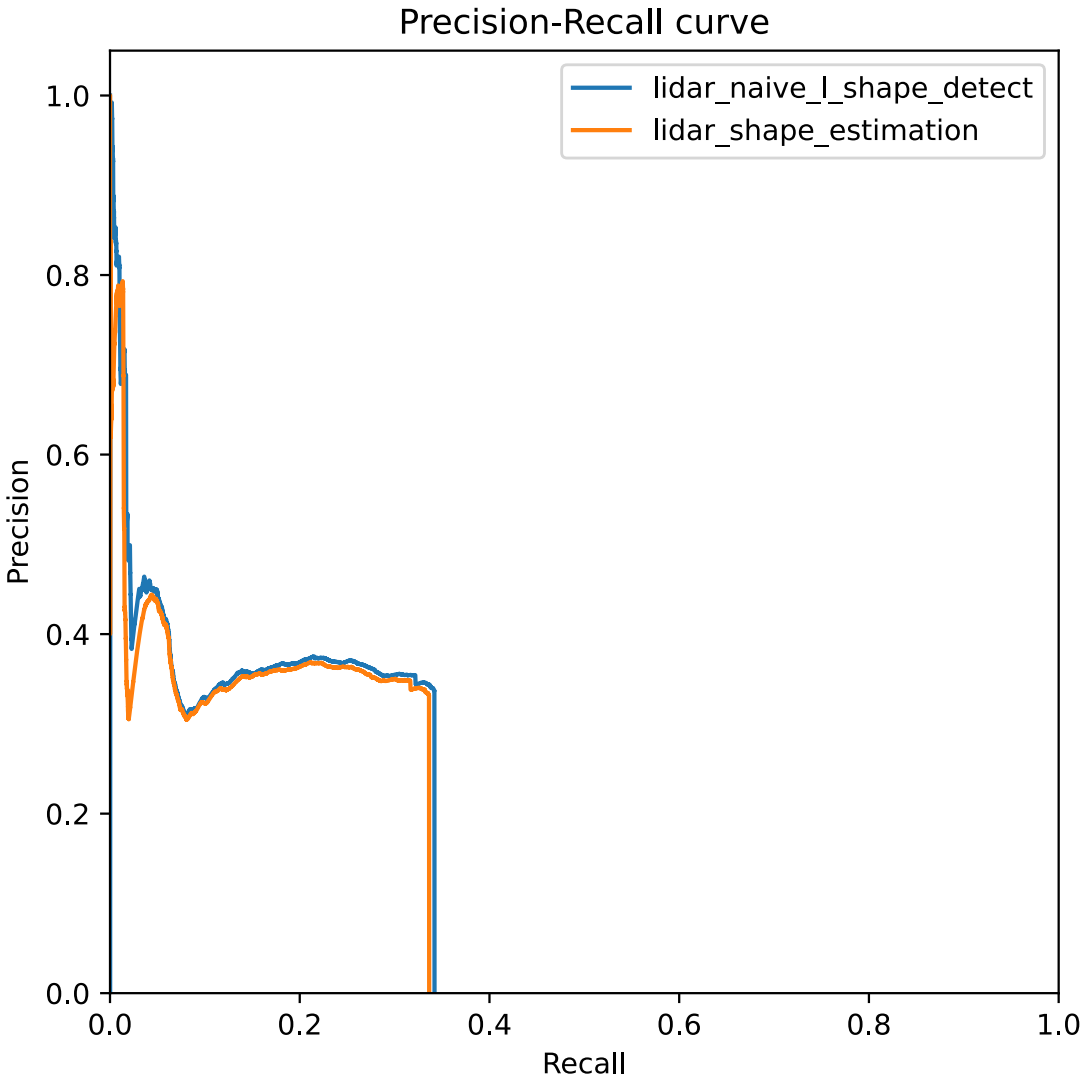

Figure 3.3: Autoware.AI Model Precision-Recall Curves

3.5.2 OpenPCDet

Table 3.6: OpenPCDet Model Evaluations for Vehicle Class

Model	AP	F_1 Score	A2TE $[m]$	A3TE $[m]$	ASE $[m^3]$
PointPillars	0.514	0.575	0.9971	1.0175	0.3182
SECOND	0.516	0.587	1.0251	1.0489	0.3224
PointRCNN	**0.552**	0.645	**0.9832**	**1.0019**	**0.2969**
PointRCNN IoU	0.536	0.627	1.0098	1.0276	**0.2901**
Part-A^2 Anchor	0.545	**0.654**	1.0404	1.0569	0.3256
Part-A^2 Free	0.477	0.618	1.0307	1.0432	0.3265
PV-RCNN	**0.552**	**0.652**	**0.9671**	**0.9856**	0.3106

Table 3.7: OpenPCDet Model Evaluations for Pedestrian Class

Model	AP	F_1 Score	A2TE $[m]$	A3TE $[m]$	ASE $[m^3]$
PointPillars	0.124	0.240	1.2543	1.2726	0.4936
SECOND	**0.273**	**0.392**	**1.2224**	**1.2364**	**0.4204**
PointRCNN	0.168	0.321	**1.2094**	**1.2412**	**0.4179**
PointRCNN IoU	0.149	0.287	1.2499	1.3092	0.4260
Part-A^2 Anchor	0.141	0.262	1.2499	1.2662	0.4858
Part-A^2 Free	0.208	**0.374**	1.2516	1.2654	0.4755
PV-RCNN	**0.225**	0.357	1.2473	1.2604	0.4630

Table 3.8: OpenPCDet Model Evaluations for Cyclist Class

Model	AP	F_1 Score	A2TE $[m]$	A3TE $[m]$	ASE $[m^3]$
PointPillars	0.046	0.110	0.8710	0.8947	0.3873
SECOND	**0.346**	**0.399**	0.8878	0.9142	0.5074
PointRCNN	0.281	**0.419**	0.8573	0.8851	**0.2813**
PointRCNN IoU	0.271	0.385	0.8777	0.8996	0.3366
Part-A^2 Anchor	0.279	0.372	**0.8127**	**0.8427**	0.4215
Part-A^2 Free	**0.309**	0.395	**0.8363**	0.8654	0.3883
PV-RCNN	0.185	0.283	0.8402	**0.8598**	**0.3289**

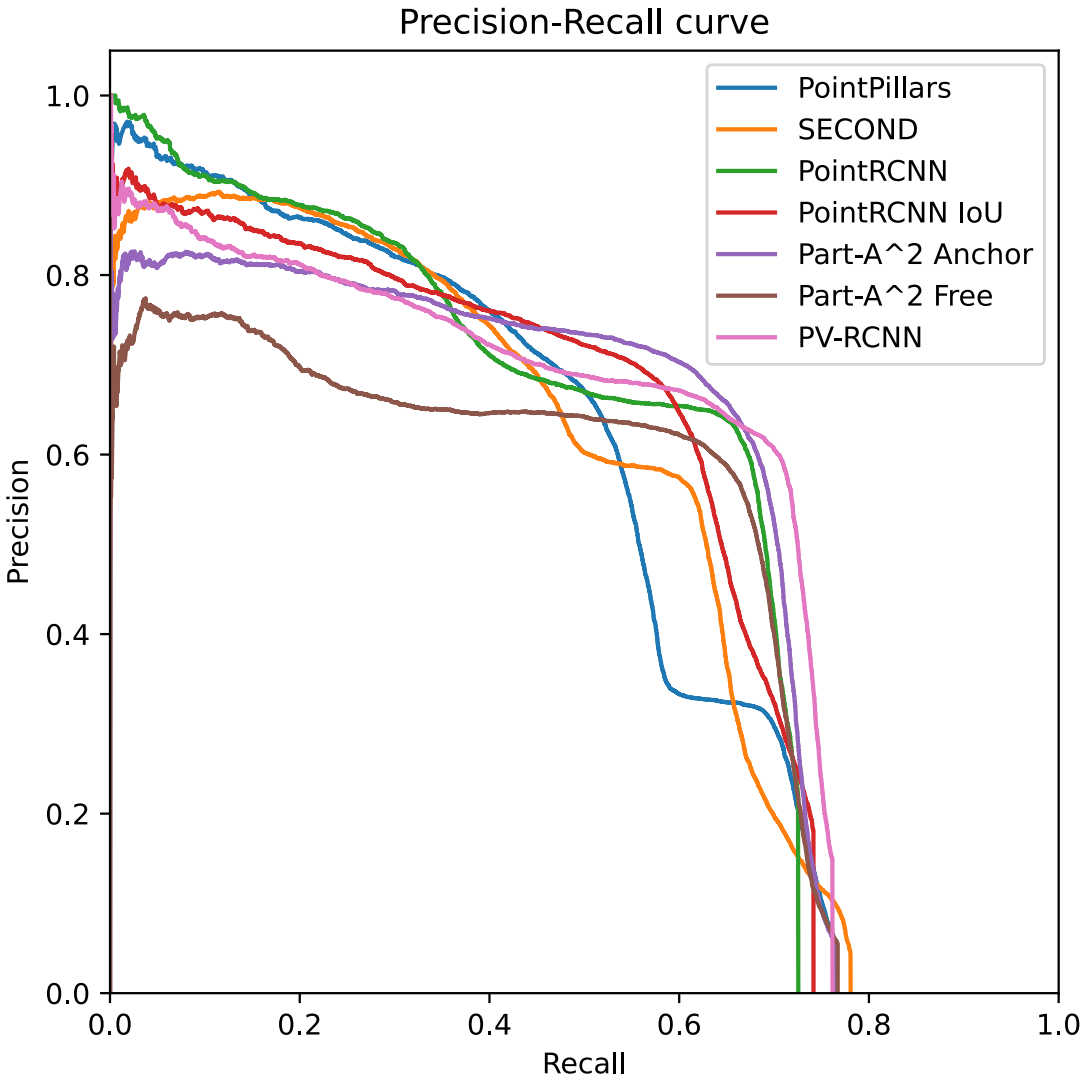

Figure 3.4: OpenPCDet Model Precision-Recall Curves — Vehicle Class

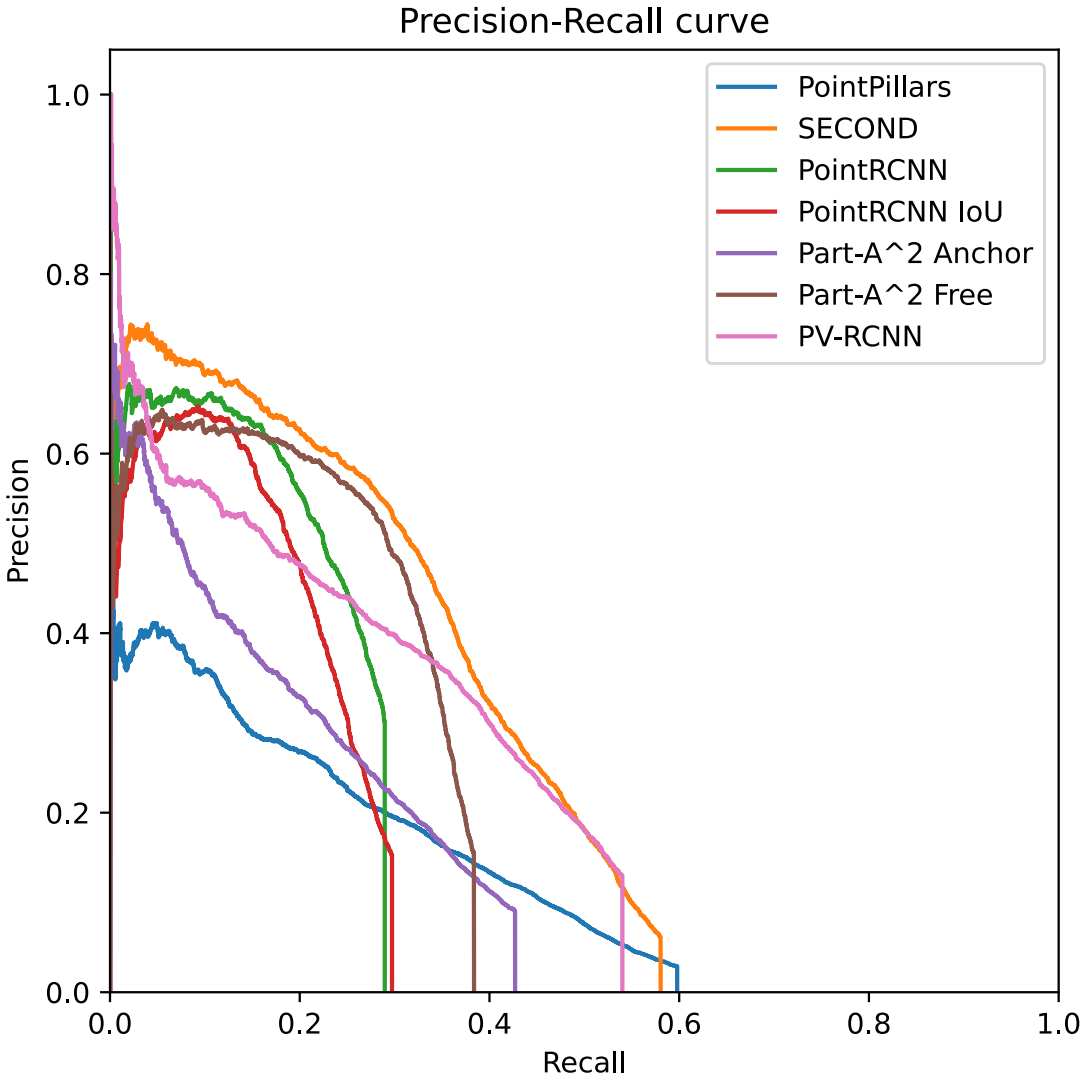

Figure 3.5: OpenPCDet Model Precision-Recall Curves — Pedestrian Class

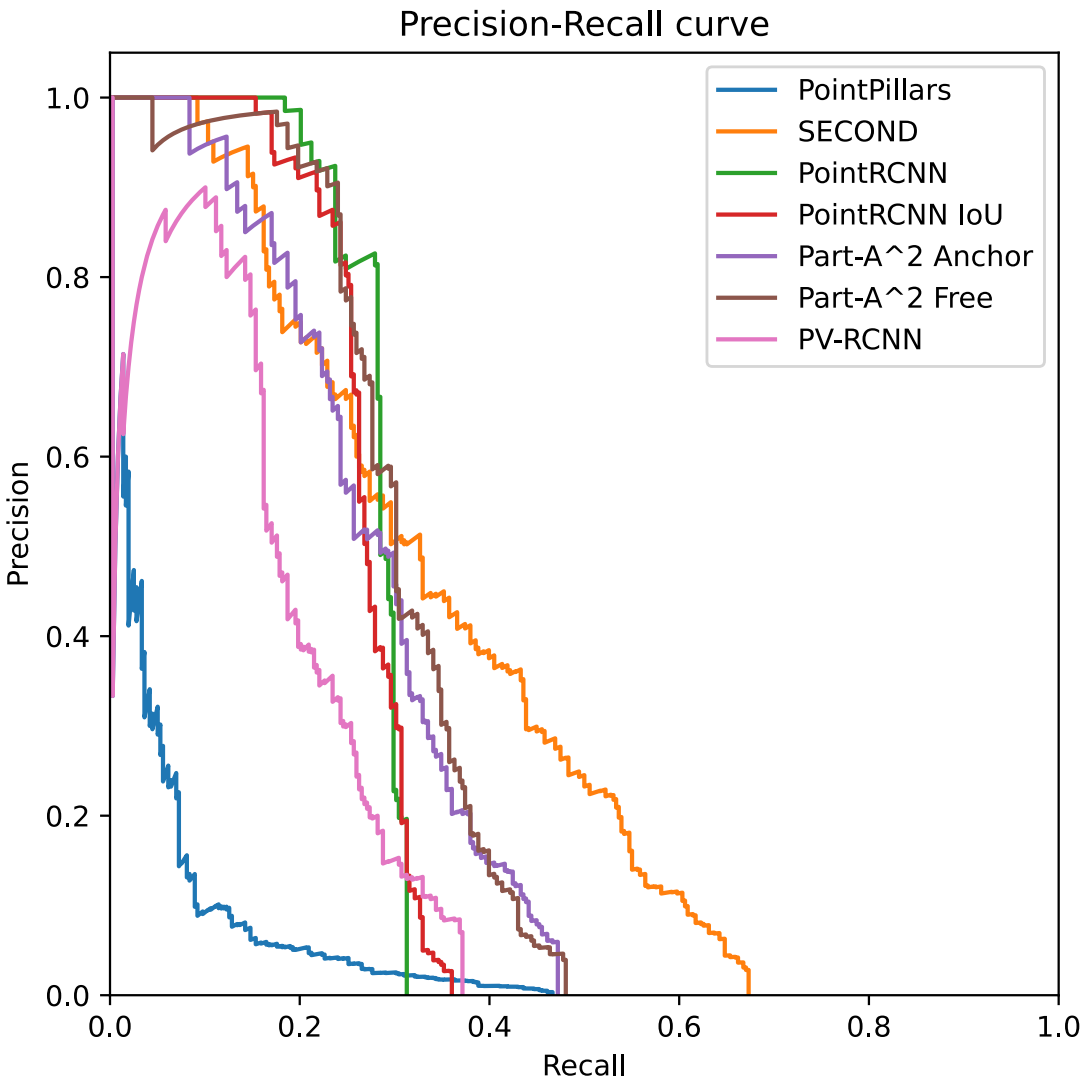

Figure 3.6: OpenPCDet Model Precision-Recall Curves — Cyclist Class

3.6 Discussion

This section discusses the results of the experiments for the Autoware.AI and OpenPCDet models, as well as assumptions made during the experimentation and analysis.

3.6.1 Autoware.AI Results

The two Autoware.AI models, lidar_naive_l_shape_detect and lidar_shape_estimation, performed similarly in almost every metric used in the evaluation. For lidar_naive_l_shape_detect, the AP is 5.26% larger, the F_1 score is 1.47% larger, the 3D ATE is 5.77% smaller, and the ASE is 26.4% smaller. For lidar_shape_estimation, the 2D ATE was smaller by 0.57%. The precision-recall curve, which shows the precision and recall for a given threshold while varying the threshold, shows that lidar_naive_l_shape_detect can maintain a marginally higher precision for the same recall than lidar_shape_estimation, and that the area under its curve (the AP) is also larger. Other than the 26.4% difference with the ASE, the other differences are not statistically significant between the two models.

3.6.2 OpenPCDet Results

Since many different OpenPCDet models are being evaluated, the best two results for each metric are emboldened. This allows us to see which models consistently perform among the best.

Vehicle Class

For the vehicle class, the model which consistently performs the best is PV-RCNN. It has the highest AP and F_1 score, and the lowest 2D and 3D ATE. PointRCNN is another model that also does really well, performing amongst the top 2 in all metrics aside from F_1 score. PointPillars, SECOND, and Part-A^2 Free perform worse, with mostly low AP scores and F_1 scores, high 2D and 3D ATE scores, and high ASE scores.

Pedestrian Class

For the pedestrian class, SECOND is the best performing model in every metric except for A2TE, where it is the second best. No other model performs as well here. PointPillars

consistently performs poorly. This may be because pedestrians are tall, slim objects. In PointPillars, the number of points in a pillar is capped, making tall objects lose more points than short objects. Additionally, each point is augmented with data representing that point's distance from the center line of it's corresponding pillar. For tall objects, there will be more points which share similar values for this data before the pillar point limit is reached, making this data more redundant.

Cyclist Class

For the cyclist class, SECOND performs the best for AP and second best for F_1 score. Part-A^2 Anchor performs the best for A2TE and A3TE. PV-RCNN performs second best for A3TE and ASE. In general, the different models each perform well in a couple of metrics each, except for PointPillars. Again, this may be for the same reasons as with the pedestrian class. Cyclists are narrow and tall, and so have a somewhat similar profile to pedestrians.

3.6.3 Assumptions

An assumption made during the course of the experiment and analysis is that the timespan of the analysis was long enough to accurately measure the performances of the various models. In order to show that this is the case, precision-recall curves for all of the models were made using 100% and the first 90% of the data. The AP values for these models are shown in Tables 3.9, 3.10, 3.11, and 3.12.

Table 3.9: Autoware models AP using 100% and first 90% of data, with percentage difference shown

Model	100%	90%	% Difference
lidar_naive_l_shape_detect	0.133	0.127	4.51%
lidar_shape_estimation	0.126	0.119	5.56%

Table 3.10: OpenPCDet models AP, for the vehicle class, using 100% and first 90% of data, with percentage difference shown

Model	100%	90%	% Difference
PointPillars	0.514	0.494	3.89%
SECOND	0.516	0.494	4.26%
PointRCNN	0.552	0.532	3.62%
PointRCNN IoU	0.536	0.517	3.54%
Part-A^2 Anchor	0.545	0.533	2.20%
Part-A^2 Free	0.477	0.456	4.40%
PV-RCNN	0.552	0.542	1.81%

Table 3.11: OpenPCDet models AP, for the pedestrian class, using 100% and first 90% of data, with percentage difference shown

Model	100%	90%	% Difference
PointPillars	0.124	0.126	1.61%
SECOND	0.273	0.275	0.73%
PointRCNN	0.168	0.169	0.60%
PointRCNN IoU	0.149	0.151	1.34%
Part-A^2 Anchor	0.141	0.143	1.42%
Part-A^2 Free	0.208	0.209	0.48%
PV-RCNN	0.225	0.227	0.89%

Table 3.12: OpenPCDet models AP, for the cyclist class, using 100% and first 90% of data, with percentage difference shown

Model	100%	90%	% Difference
PointPillars	0.046	0.047	2.17%
SECOND	0.346	0.347	0.29%
PointRCNN	0.281	0.283	0.71%
PointRCNN IoU	0.271	0.273	0.74%
Part-A^2 Anchor	0.279	0.282	1.08%
Part-A^2 Free	0.309	0.31	0.32%
PV-RCNN	0.185	0.186	0.54%

The PR-curves comparing 100% data use and 90% data use are shown in Appendix A. Looking at this data, for the Autoware models, the percentage differences of the AP between 100% and 90% data usage are 4.51% for lidar_naive_l_shape_detect and 5.56% for lidar_shape_estimation. All of the OpenPCDet models have smaller percentage differences across all their classes. For the vehicle class, the largest percentage difference is 4.40%, and the smallest is 1.81%. For the pedestrian class, the largest percentage difference is 1.61%, and the smallest is 0.48%. For the cyclist class, the largest percentage difference is 2.17%, and the smallest is 0.29%. These are all very small percentage differences, which show that the AP does not change by much when the amount of data used is changed. Thus, this analysis supports the validity of the original results in Section 3.5.

3.6.4 Application to DBL Project

The original project that this thesis is based on is the DBL project. The comparisons of the different frameworks and models ultimately provide quantitative analysis justifying the use of a specific framework and model for the purposes of 3D object detection from LiDAR data. To this end, the data in this thesis supports the use of PV-RCNN, from the OpenPCDet framework, as the 3D LiDAR-based object detection model.

Chapter 4

Conclusion

In this thesis, an analysis and comparison of various 3D LiDAR-based object detection frameworks and models was performed. The two frameworks analyzed were Autoware.AI, a non-neural network based framework, and OpenPCDet, a neural network based framework. Within Autoware.AI, the lidar_euclidean_cluster_detect algorithm was used as the base clustering algorithm upon which lidar_naive_l_shape_detect and lidar_shape_estimation are built. Within OpenPCDet, a modular framework was used to build all five models and their variations.

4.1 Recommendations

For the Autoware.AI models, lidar_naive_l_shape_detect and lidar_shape_estimation, since there was no associated confidence score to use for thresholding, Euclidean distance from the origin was used instead. The overall AP scores and F_1 scores are far lower than the scores for the OpenPCDet models on the vehicle class, and are instead comparable to the scores of the OpenPCDet results on the pedestrian class.

For the OpenPCDet models, PointPillars struggles the most, possibly due to it viewing the data as a collection of thin pillars, which especially leave out a lot of information from pedestrian and cyclist classes. SECOND does well on the pedestrian class and the cyclist class, having the highest AP in both. PointRCNN does very well with the vehicle class, tying for first place with PV-RCNN in AP, and being amongst the lowest two for the three error metrics. PointRCNN IoU performs worse than PointRCNN on most metrics. Part-A^2 Anchor does well on the vehicle and cyclist classes. Part-A^2 Free does well on the

pedestrian and cyclist classes. PV-RCNN performed the best out of all the models for the vehicle class, and achieved the second highest AP in the pedestrian class. Interestingly, it performed poorly in the cyclist class.

For prediction on vehicles, PV-RCNN is recommended. For prediction on pedestrians, SECOND or PV-RCNN is recommended. For prediction on cyclists, SECOND or Part-A^2 Free is recommended.

4.2 Future Work

For future work, it is recommended to continue looking at the development of the Autoware and OpenPCDet platforms. Autoware.AI is being succeeded by Autoware.Auto. It is based on ROS 2, and will focus on software engineering best practises, improved system architecture, and reproducibility. OpenPCDet is continually developing, with new models constantly coming out. It is the official codebase for Voxel R-CNN [13] and PV-RCNN++ [31], and contains the code for CenterPoint [37] as well. OpenPCDet is constantly adding support for newer 3D LiDAR-based object detection models, and so it is expected that even more models than these will also become available to look at in the future.

References

[1] Autoware-ai. [Online]. Available: https://github.com/Autoware-AI. Accessed: Nov 10, 2021. 6

[2] Difference of normals based segmentation. [Online]. Available: https://pcl.readthedocs.io/projects/tutorials/en/latest/don_segmentation.html. Accessed: Nov 23, 2021. ix, 7

[3] Downsampling a pointcloud using a voxelgrid filter. [Online]. Available: https://pcl.readthedocs.io/projects/tutorials/en/latest/voxel_grid.html. Accessed: Nov 23, 2021. 7

[4] Estimating surface normals in a pointcloud. [Online]. Available: https://pcl.readthedocs.io/projects/tutorials/en/latest/normal_estimation.html. Accessed: Nov 23, 2021. 7

[5] Euclidean cluster extraction. [Online]. Available: https://pcl.readthedocs.io/projects/tutorials/en/latest/cluster_extraction.html. Accessed: Nov 24, 2021. 8

[6] imm_ukf_pda_track. [Online]. Available: https://github.com/Autoware-AI/core_perception/tree/master/imm_ukf_pda_track. Accessed: Nov 28, 2021.

[7] lidar_euclidean_cluster_detect. [Online]. Available: https://github.com/Autoware-AI/core_perception/tree/master/lidar_euclidean_cluster_detect. Accessed: Nov 21, 2021. 6

[8] lidar_naive_l_shape_detect. [Online]. Available: https://github.com/Autoware-AI/core_perception/tree/master/lidar_naive_l_shape_detect. Accessed: Nov 27, 2021. 9

[9] lidar_shape_estimation. [Online]. Available: https://github.com/Autoware-AI/core_perception/tree/master/lidar_shape_estimation. Accessed: Nov 27, 2021. 9

[10] naive_motion_predict. [Online]. Available: https://github.com/Autoware-AI/core_perception/tree/master/naive_motion_predict. Accessed: Nov 29, 2021.

[11] Object detection task - nuscenes. [Online]. Available: https://www.nuscenes.org/object-detection. Accessed: Feb 12, 2021. 21

[12] Velodyne hdl-32e datasheet. [Online]. Available: https://www.mapix.com/wp-content/uploads/2018/07/97-0038_Rev-M_-HDL-32E_Datasheet_Web.pdf. Accessed: Nov 10, 2021. 3, 23

[13] Jiajun Deng, Shaoshuai Shi, Pei-Cian Li, Wen gang Zhou, Yanyong Zhang, and Houqiang Li. Voxel r-cnn: Towards high performance voxel-based 3d object detection. In *AAAI*, 2021. 39

[14] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981. 7

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 2, 10

[16] Benjamin Graham. Spatially-sparse convolutional neural networks. *ArXiv*, abs/1409.6070, 2014. 11

[17] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *ArXiv*, abs/1706.01307, 2017. 11

[18] Yani Andrew Ioannou, Babak Taati, Robin Harrap, and Michael A. Greenspan. Difference of normals as a multi-scale operator in unorganized point clouds. *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 501–508, 2012. 7

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015. 11

[20] Jacob Lambert. 3d lidar bbox detection evaluation. [Online]. Available: https://github.com/jacoblambert/3d_lidar_detection_evaluation. Accessed: Feb 12, 2021. 21, 27

[21] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12689–12697, 2019. 9, 13

[22] W. Liu, Dragomir Anguelov, D. Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. 12, 14

[23] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 11

[24] C. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017. 13

[25] C. Qi, L. Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 14

[26] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, May 2009.

[27] A. S. A. Rachman. 3d-lidar multi object tracking for autonomous driving: Multi-target detection and tracking under urban road uncertainties. 2017. ix, 9, 10

[28] Radu Rusu. Semantic 3d object maps for everyday manipulation in human living environments. *KI - Künstliche Intelligenz*, 24, 11 2010. 8

[29] Matthias Schreier. Bayesian environment representation, prediction, and criticality assessment for driver assistance systems. *at - Automatisierungstechnik*, 65:151 – 152, 2017.

[30] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10526–10535, 2020. 9, 17

[31] Shaoshuai Shi, Li Jiang, Jiajun Deng, Zhe Wang, Chaoxu Guo, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn++: Point-voxel feature set abstraction with local vector representation for 3d object detection. *ArXiv*, abs/2102.00463, 2021. 39

[32] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–779, 2019. 9, 14

[33] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43:2647–2664, 2021. 9, 16

[34] Muhammad Sualeh and Gon woo Kim. Dynamic multi-lidar based multiple object detection and tracking. *Sensors (Basel, Switzerland)*, 19, 2019.

[35] OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. https://github.com/open-mmlab/OpenPCDet, 2020. 9

[36] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors (Basel, Switzerland)*, 18, 2018. 9, 10

[37] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3d object detection and tracking. *CVPR*, 2021. 39

[38] Xiao Zhang, Wenda Xu, Chiyu Dong, and John M. Dolan. Efficient l-shape fitting for vehicle detection using laser scanners. In *2017 IEEE Intelligent Vehicles Symposium*, June 2017. 9

[39] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018. 11, 14

[40] Adam Ziebinski, Rafal Cupek, Hueseyin Erdogan, and Sonja Waechter. A survey of adas technologies for the future perspective of sensor fusion. In Ngoc Thanh Nguyen, Lazaros Iliadis, Yannis Manolopoulos, and Bogdan Trawiński, editors, *Computational Collective Intelligence*, pages 135–146, Cham, 2016. Springer International Publishing. 1

APPENDICES

Appendix A

Precision-Recall Curves Using 100% and First 90% of Data

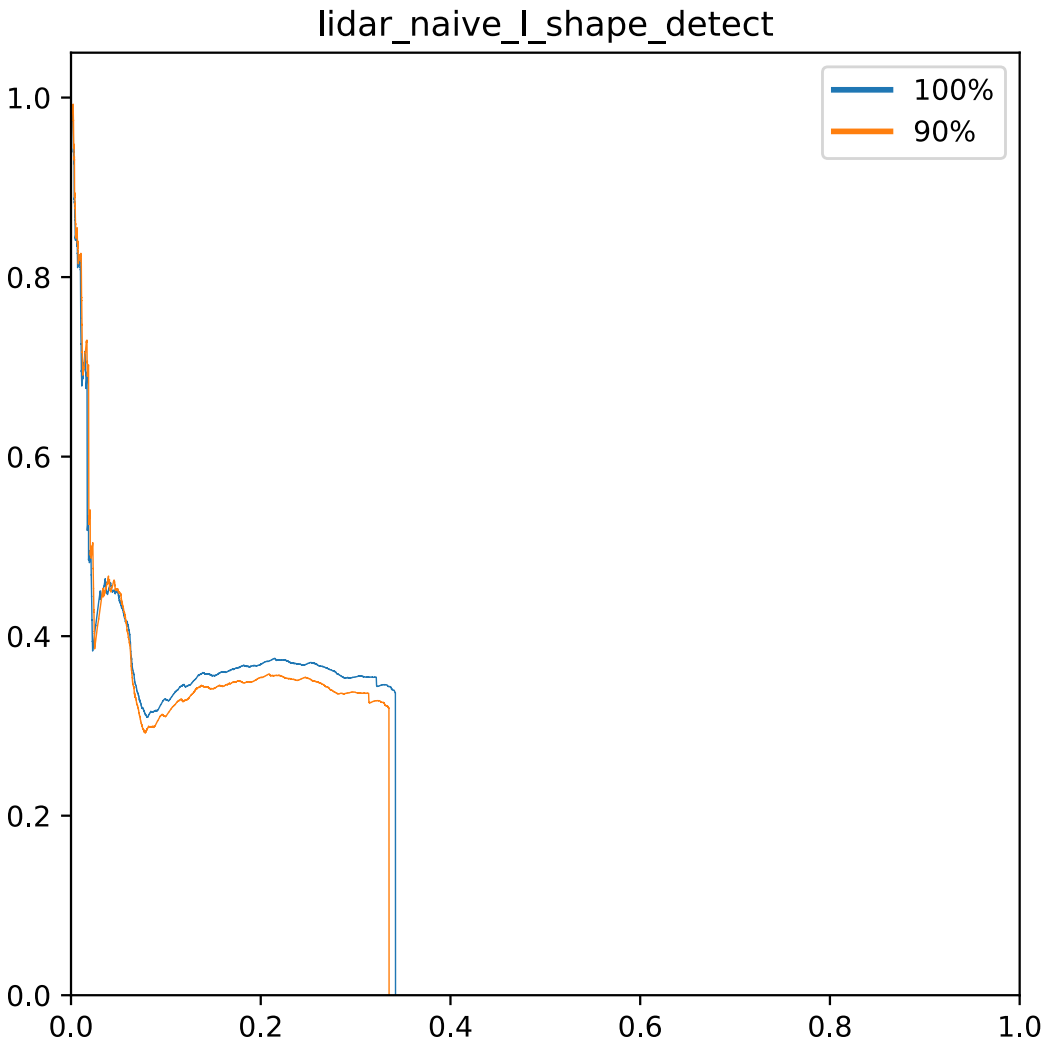

Figure A.1: lidar_naive_l_shape_detect PR-curve using 100% and first 90% of data

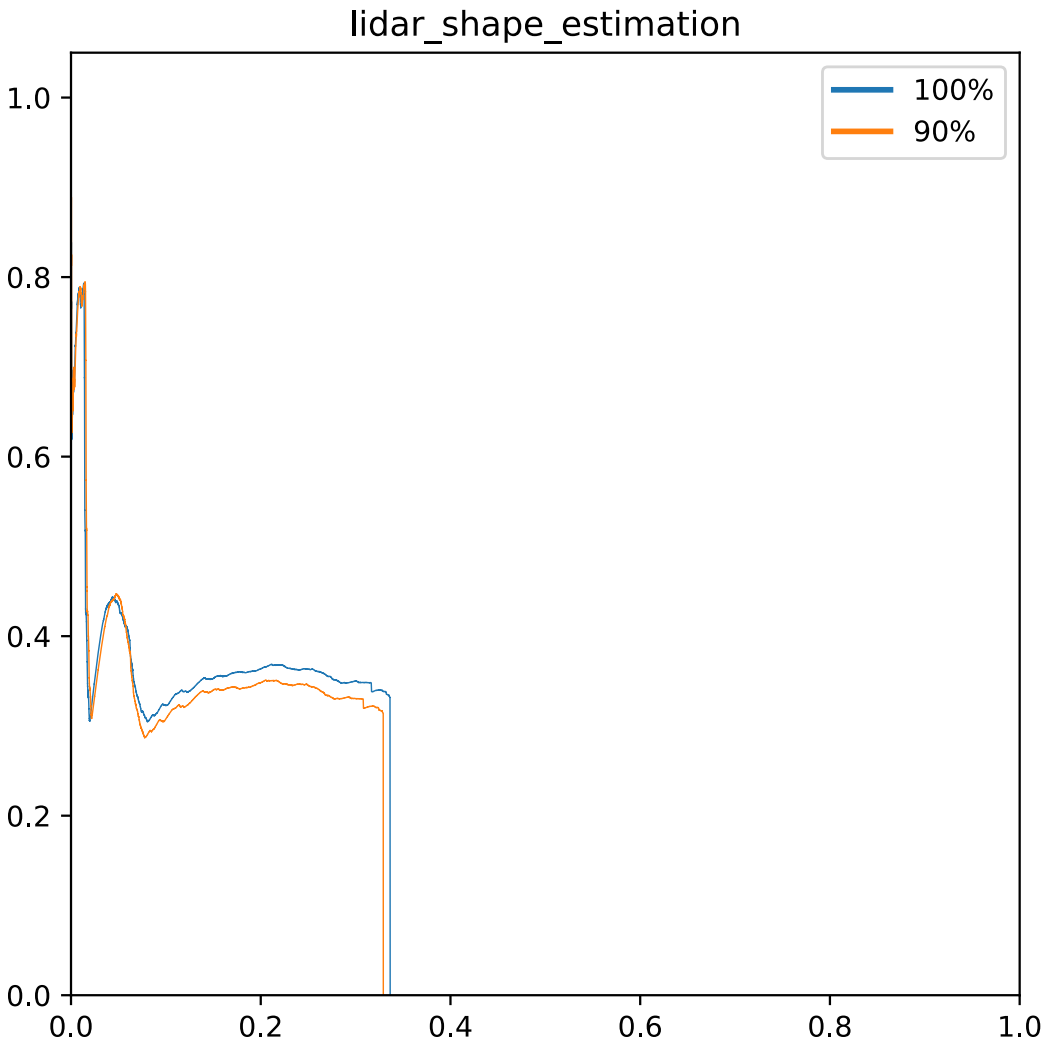

Figure A.2: lidar_shape_estimation PR-curve using 100% and first 90% of data

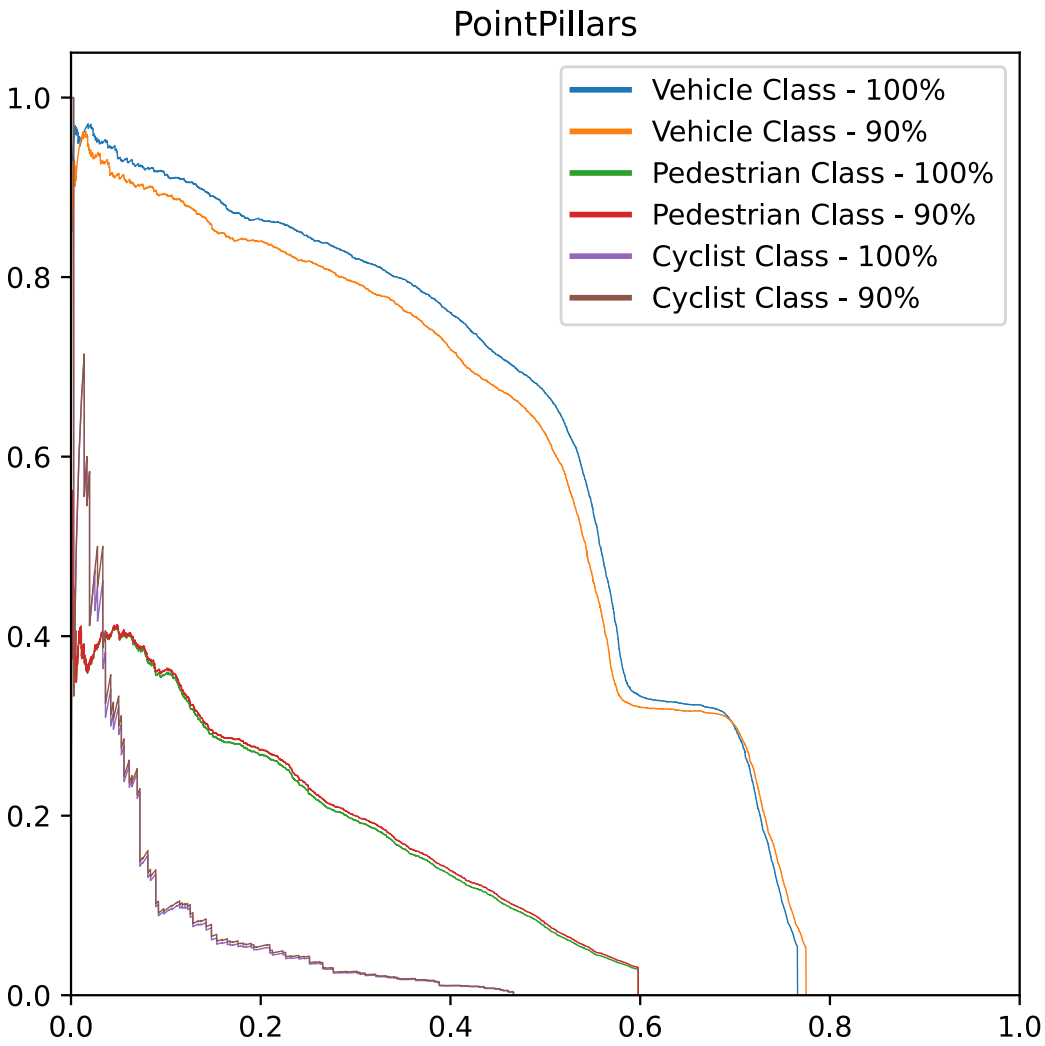

Figure A.3: PointPillars PR-curve using 100% and first 90% of data

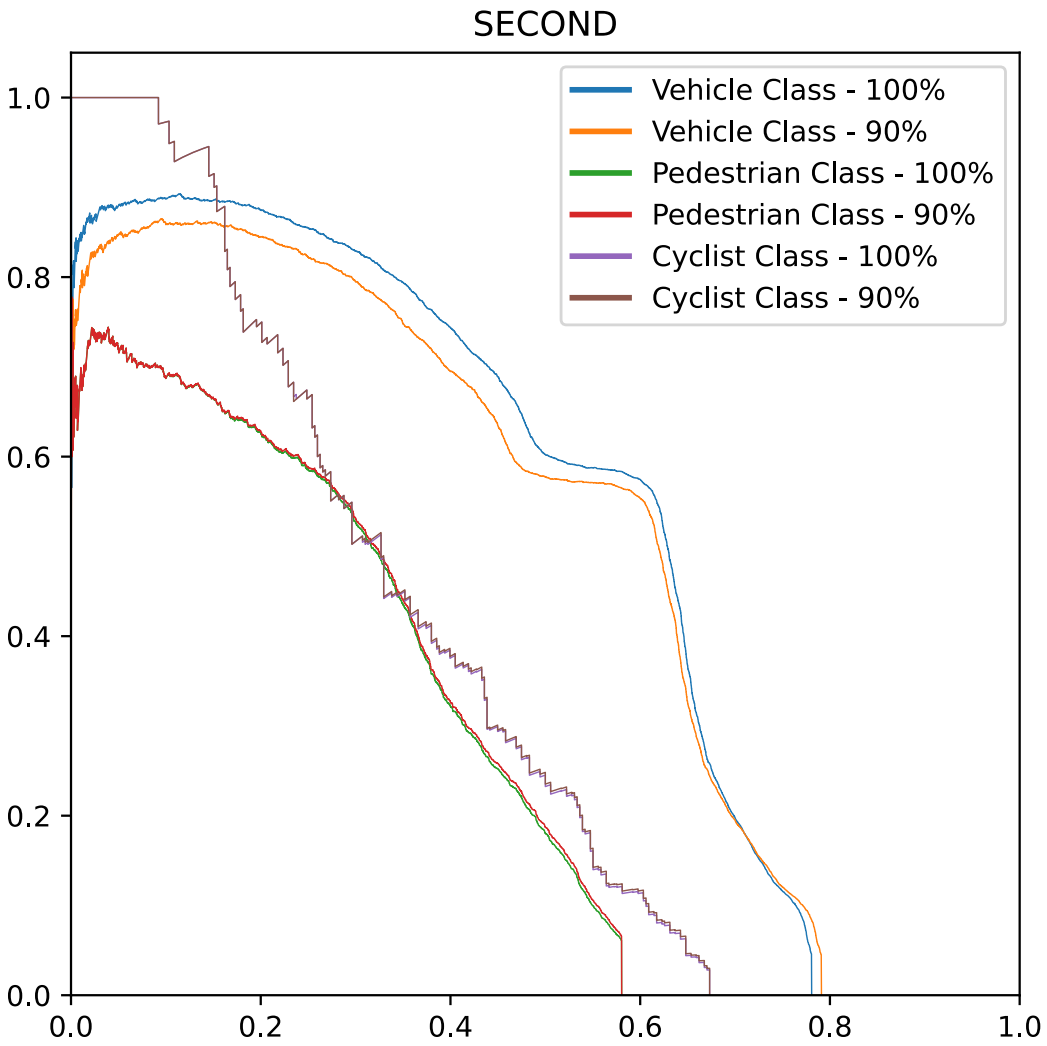

Figure A.4: SECOND PR-curve using 100% and first 90% of data

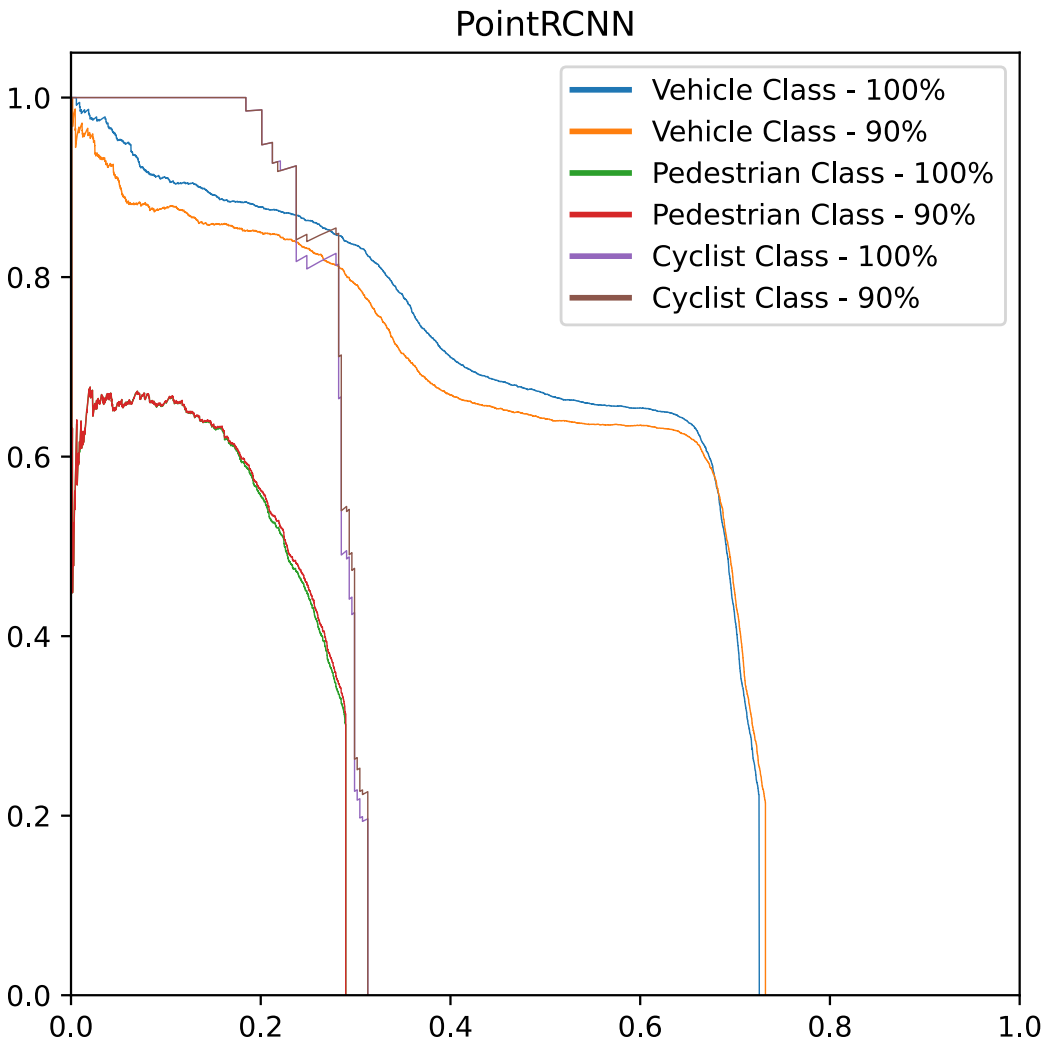

Figure A.5: PointRCNN PR-curve using 100% and first 90% of data

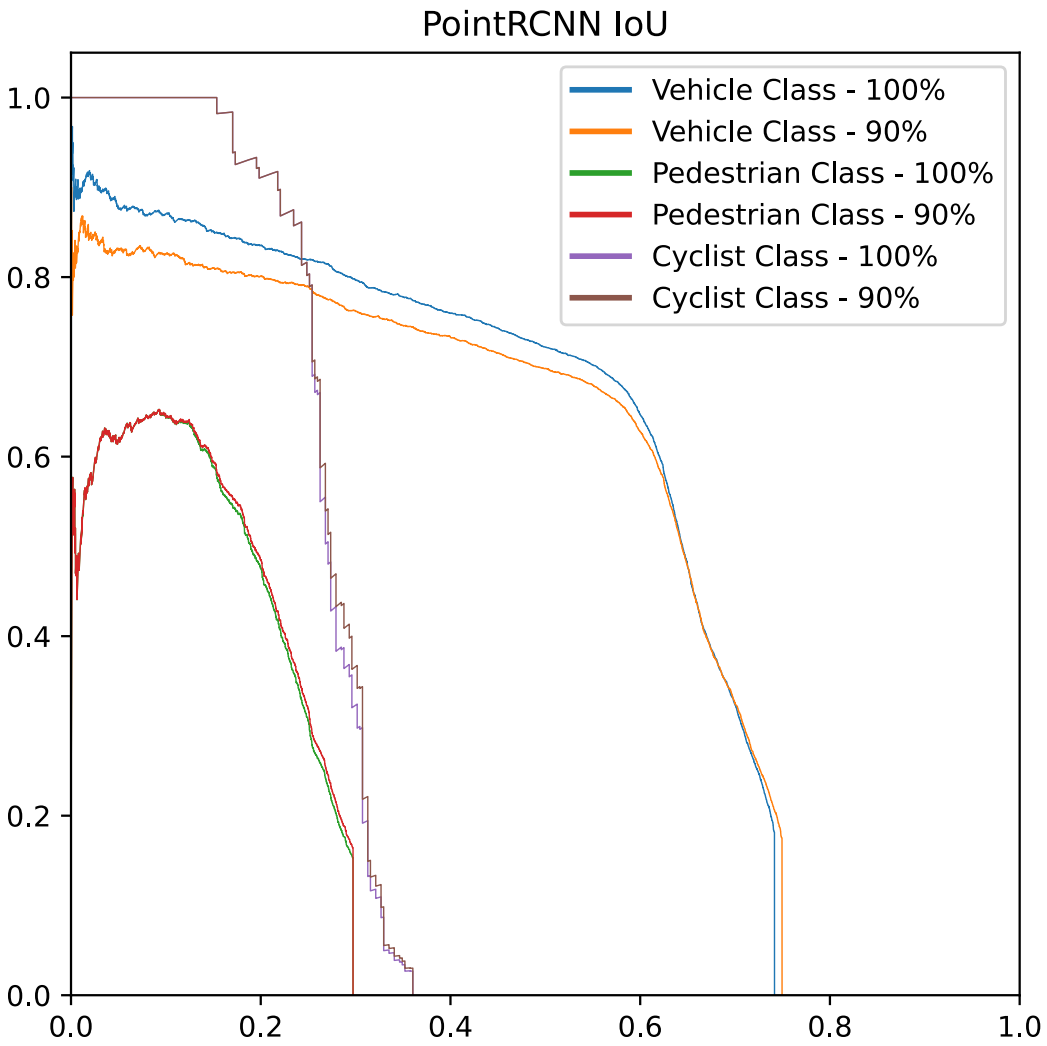

Figure A.6: PointRCNN IoU PR-curve using 100% and first 90% of data

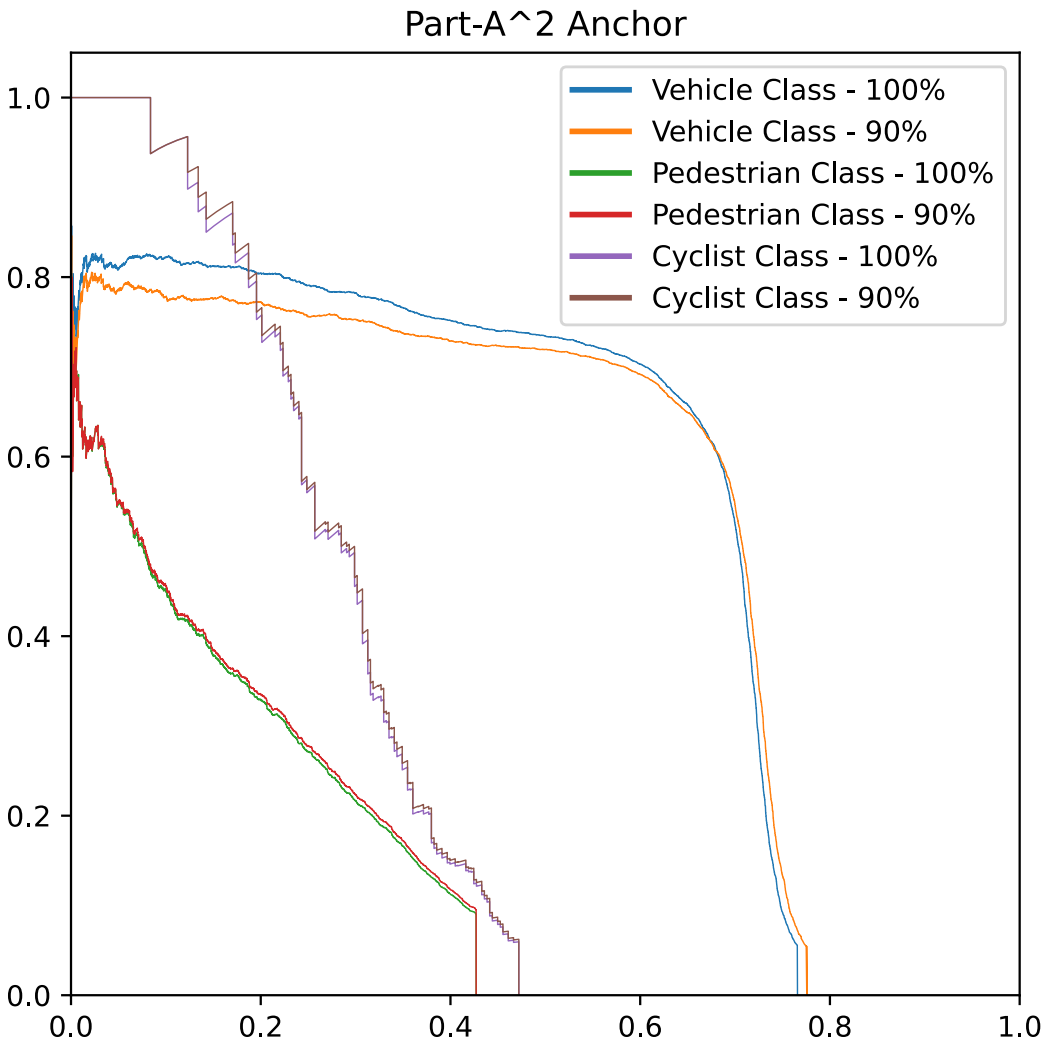

Figure A.7: Part-A^2 Anchor PR-curve using 100% and first 90% of data

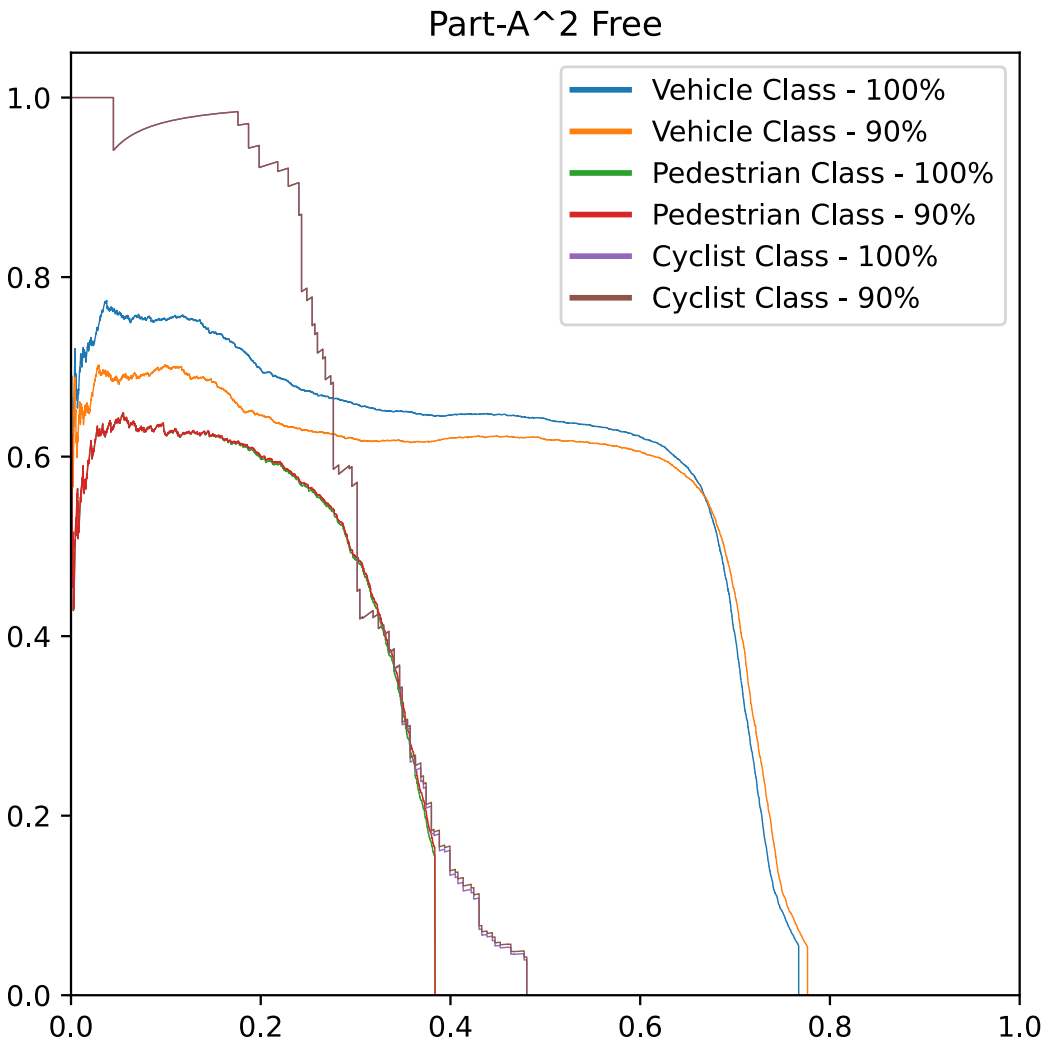

Figure A.8: Part-A^2 Free PR-curve using 100% and first 90% of data

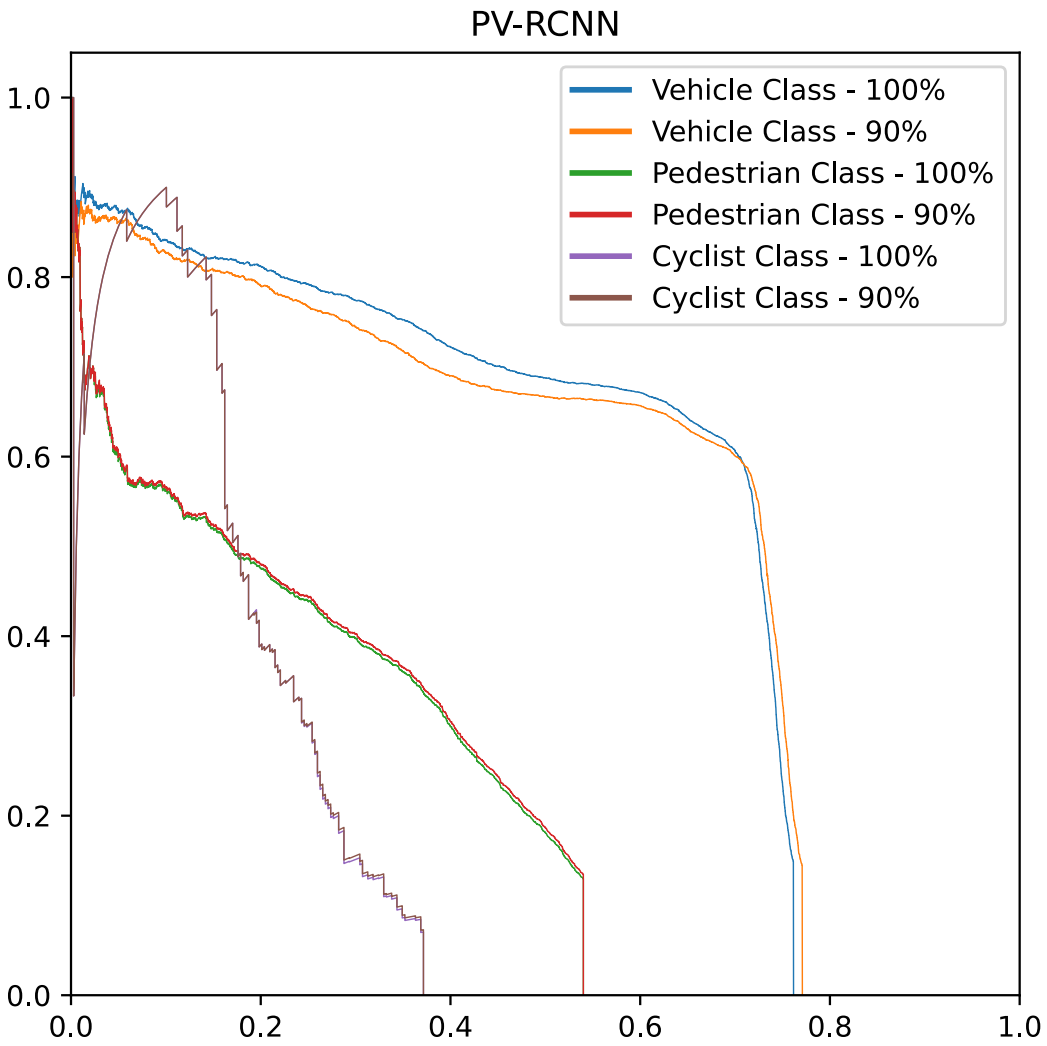

Figure A.9: PV-RCNN PR-curve using 100% and first 90% of data