ABSTRACT

SINGH, ABHIHEK RANJAN. PointPillars++: An Encoder for 3-D Object Detection and Classification from Point Clouds. (Under the direction of Dr. Seth Hollar).

3-D Object detection and classification are integral parts of Autonomous Driving Systems. In this work, designing a fast yet reliable object detection system for the task of autonomous driving is considered. 3-D data, unlike 2-D data, has various representations such as 3-D voxels, point clouds, and 3-D images. Due to point clouds' irregular and permutation invariant nature, most researchers tend to favor using 3-D voxels or 3-D images. However, both 3-D voxels and 3-D images are unnecessarily voluminous, making deep learning tasks on them computationally expensive. In their seminal paper Qi et al. (2017a) provided a unified architecture that can be used to perform deep-learning on point clouds. The introduction of PointNet sparked research into utilizing the sparsity of point cloud to reach computationally economic algorithms for performing various deep learning tasks on 3-D data such as 3-D object detection, classification, and segmentation. Lang et al. (2019) utilized PointNet Qi et al. (2017a) to encode the point cloud into a pseudo-image and use existing 2-D detection and classification techniques (Liu et al. 2016) to perform 3-D object detection with the help of only point clouds. PointPillars (Lang et al. 2019) reached the remarkable speed of 62 Hz, which was three times better than the second-best while maintaining the state-of-the-art accuracy. Although since PointPillars utilizes PointNet, it fails to capture the local substructure induced by the metric space points live in, limiting its ability to recognize fine-grained patterns and generalizability to common scenes. Qi et al. (2017b) propose PointNet++ to get around this issue by utilizing hierarchical feature learning.

This work can be viewed as an effort to utilize the findings of Qi et al. (2017b) and Lang et al. (2019) in creating a 3-D object detection and classification system for autonomous vehicles. This thesis proposes PointPillars++: an encoder for 3-D object detection and classification from point clouds, which utilizes the sparsity of point clouds and hierarchical feature learning of PointNet++. Extensive experimentation shows that PointPillars++ outperforms existing encoders in accuracy by a significant margin while still maintaining a workable speed of 12 Hz.

PointPillars++: An Encoder for 3-D Object Detection and Classification from Point Clouds

by
Abhihek Ranjan Singh

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Electrical Engineering

Raleigh, North Carolina
2021

APPROVED BY:

Dr. Gregory Bottomley	Dr. Tianfu Wu

Dr. Seth Hollar
Chair of Advisory Committee

DEDICATION

I dedicate this work to my parents for their never-ending support and belief in me, my younger brother, who has always motivated me to take on challenges out of my comfort zone, and my friends over here and overseas, whose company kept me going.

BIOGRAPHY

Abhishek was born in a small town located in the Deoria district of Uttar Pradesh, India, and was brought up in the Khandwa district of Madhya Pradesh, India. He attended Maulana Azad National Institute of Technology (MANIT) in Bhopal, India, and was conferred an undergraduate degree in Electrical Engineering in 2018. He was vice-chair of IEEE MANIT Student branch during the academic year 2017-18, where his team was awarded the Darrel Chong students' activity award for their exemplary work in AARAMBH, which was an initiative to help underprivileged children get access to technological education. He was also the leader of the electrical design team for Solaracers, which was taking part in the Electrical Solar Vehicle Competition, 2016 (ESVC'16). Under his leadership, the team won the best design award and reached the overall fourth position in the nationwide competition. After Graduation, he worked for Toppr Technologies Pvt. Ltd. as a Product Developer for the year 2018-19.

Abhishek enrolled at the NC State University in Fall 2019 and since then has worked with EcoPRT, improving the software pipeline and researching multiple approaches to solve the 3D-Object detection and classification Problem.

ACKNOWLEDGEMENTS

This thesis can be viewed as an effort to utilize the findings of Qi et al. (2017b) and Lang et al. (2019) in designing a 3-D object detection and classification system for an autonomous vehicle.

Firstly, I would like to thank my advisor **Dr. Seth Hollar** for trusting me and allowing me to work on the EcoPRT project. Dr. Hollar has supported me by providing a research assistantship and guided me whenever I was stuck on any technical problem. Working under him has been a great learning experience. He also helped me with the thesis and guided me over the time of development. Lastly, I would like to express my gratitude for his patience, enthusiasm, and immense knowledge.

Secondly, I would like to thank my thesis committee members **Dr. Gregory Bottomley** and **Dr. Tianfu Wu** for all of their guidance and feedback throughout this process. Their lectures and coursework helped me understand a great deal about utilizing Machine Learning and Deep learning to solve 3-D object detection.

Thirdly, I would like to thank **Deveshwar Hariharan** for helping me build a solid basis for this work and continuously guiding me in the right direction. His support, both as a senior and a friend, was highly vital to completing this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

1.1 Motivation

1.1.1 The Rise of Autonomous Vehicles

Between 2012 and 2021, the number of registered vehicle in US increased by 40.8 million (Hedges and Company 2021) which is a 16 percent increase, compared to that the US population has increased by a mere 6 percent (MacroTrends 2021).With the increase in vehicles, congestion, parking and road accidents have become even more pressing. In the US alone, the number of fatal crashes during 2015-2019 has reached an alarming number of 165,556 (Administration 2021).

Autonomous Vehicles can provide a solution to some of the problems mentioned above by providing more efficient planning and vehicle control (Cottam 2018). In the past few years, autonomous driving has seen a significant rise in popularity and investment (Krishnan 2021). Many tech giants have either launched their own self-driving car or started investing in startups related to autonomous vehicle technology (Cheung 2018).

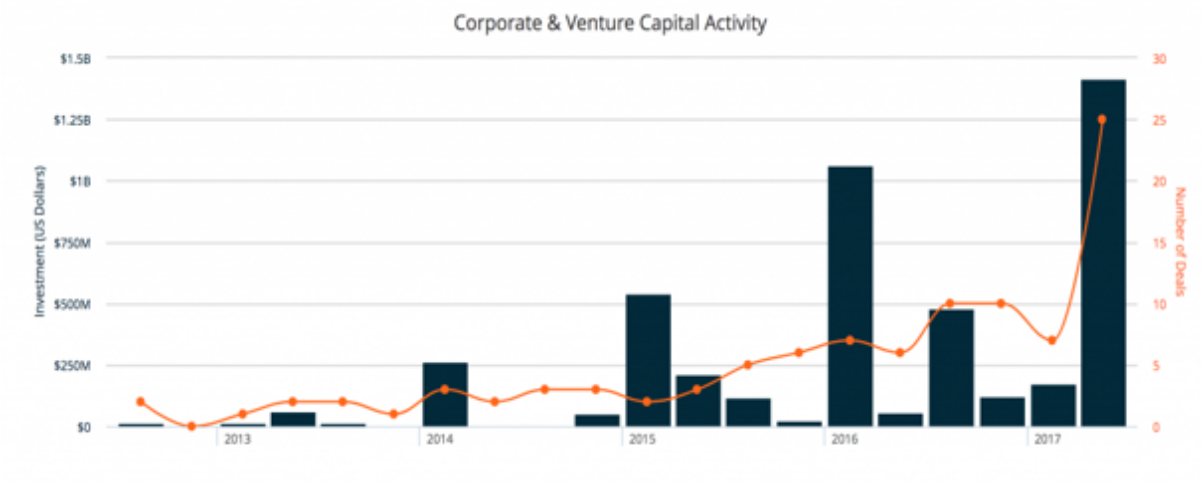

Figure 1.1: The rise of investment in Autonomous Vehicle technology | Image Credit: Krishnan (2021)

Figure 1.2: World's top companies working on Autonomous vehicle technology | Image Credit: Cheung (2018)

1.1.2 Perception for Autonomous Vehicles

Even though Autonomous Vehicles are now more real than ever, designing one is not a trivial task (see Figure 1.3). One of the significant challenges in designing any autonomous vehicle is characterized by perception.

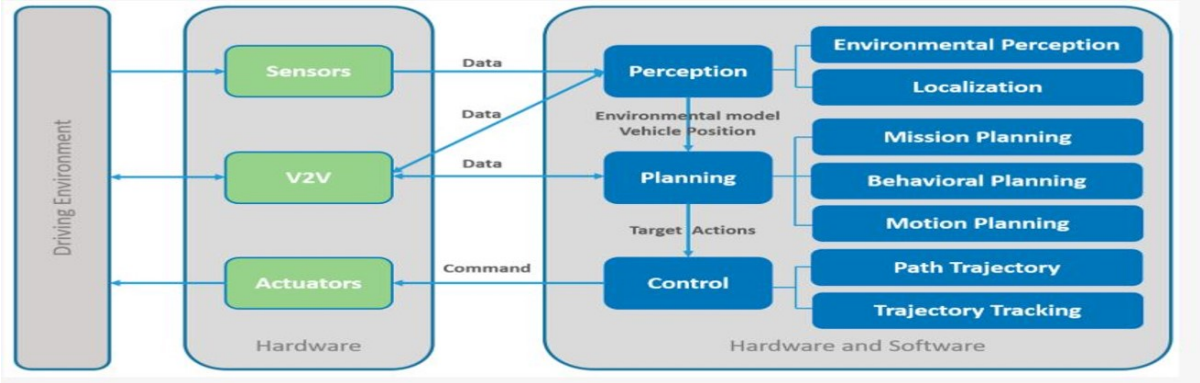

Figure 1.3: A typical Autonomous Vehicle Software Stack | Image Credit: Rosique et al. (2019)

While humans have been able to solve the problem of perception with only a few types of proprioceptive and exteroceptive sensors (see Figure 1.4), the autonomous driving systems of today typically require a large number of exteroceptive sensors coupled with a significant amount of computing power (see Figure 1.5 and Figure 1.6). As can be seen in Figure 1.6, the task of perception can be modularized into various sub-tasks such as object detection, lane detection, vehicle state estimation, and occupancy grid map generation. Most of these sub-tasks can be classified in either of two categories, i.e., finding information about the vehicle or finding information about vehicle surroundings. These tasks require information processing on 3-D data, making 3-D object detection the core problem to be solved by the perception module of any autonomous vehicle.

1.1.3 Other Application of 3-D Object Detection

The ability to detect and classify a 3-D object is not only beneficial for autonomous vehicles but also finds its application in various other fields such as national defense and security (Hua et al. 2018), medical research (Singh et al. 2020), and remote sensing (Nevalainen et al. 2017). One of the rather exciting applications of 3-D object detection is its use in

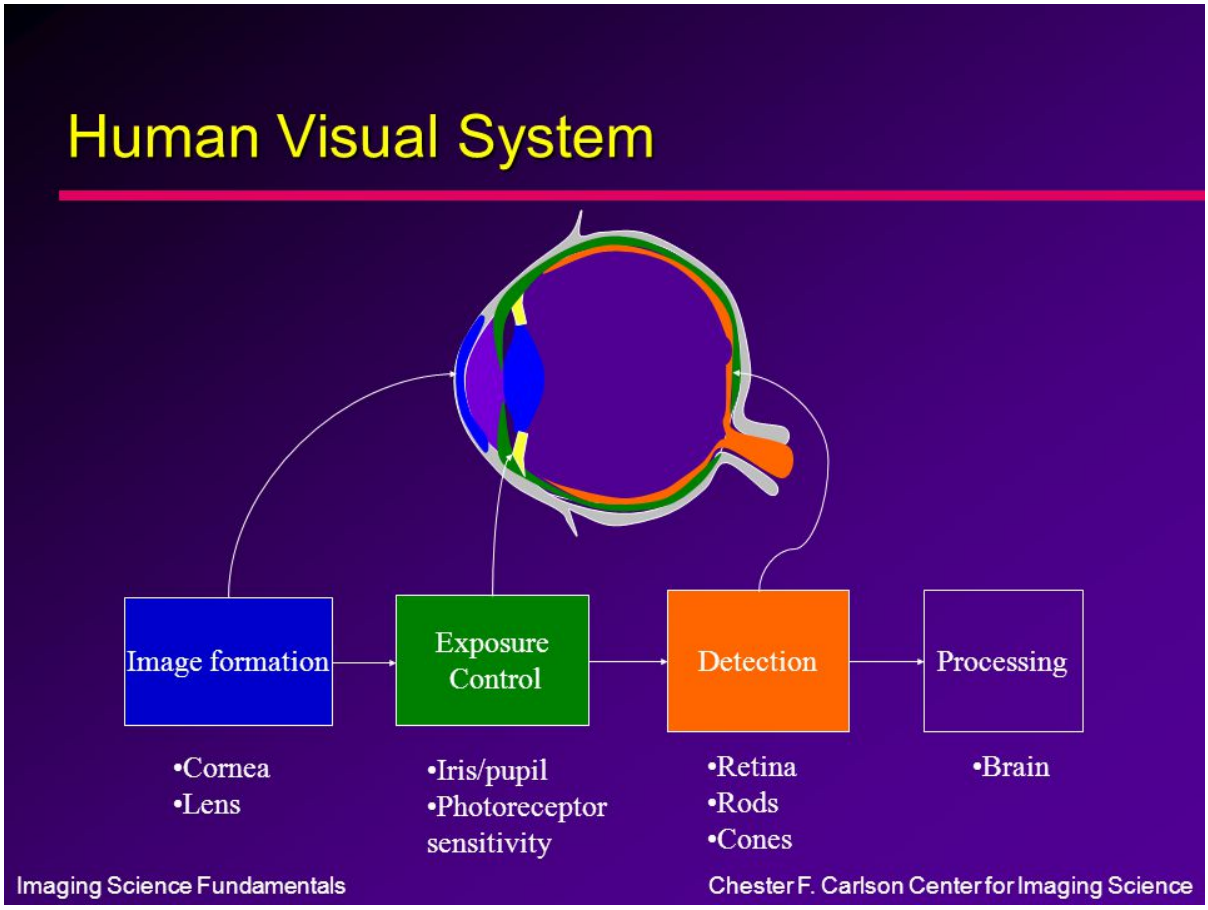

Figure 1.4: Human Visual System | Image Credit: (Carlson 2021)

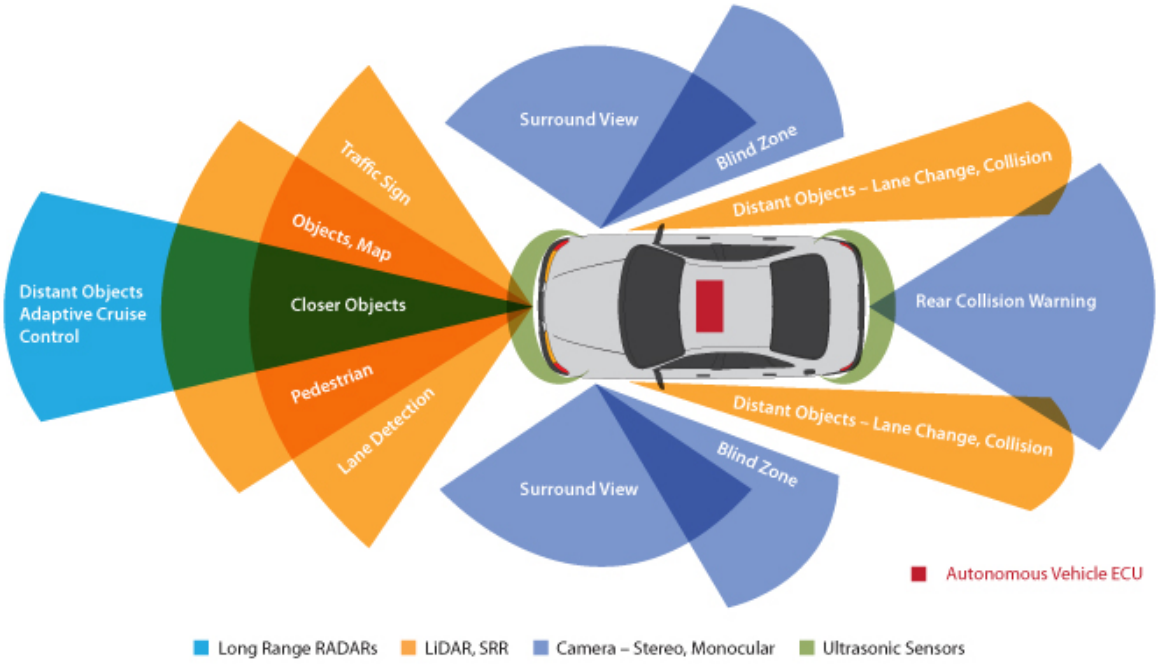

Figure 1.5: A typical autonomous vehicle sensor requirement | Image Credit: (Elxsi 2020)

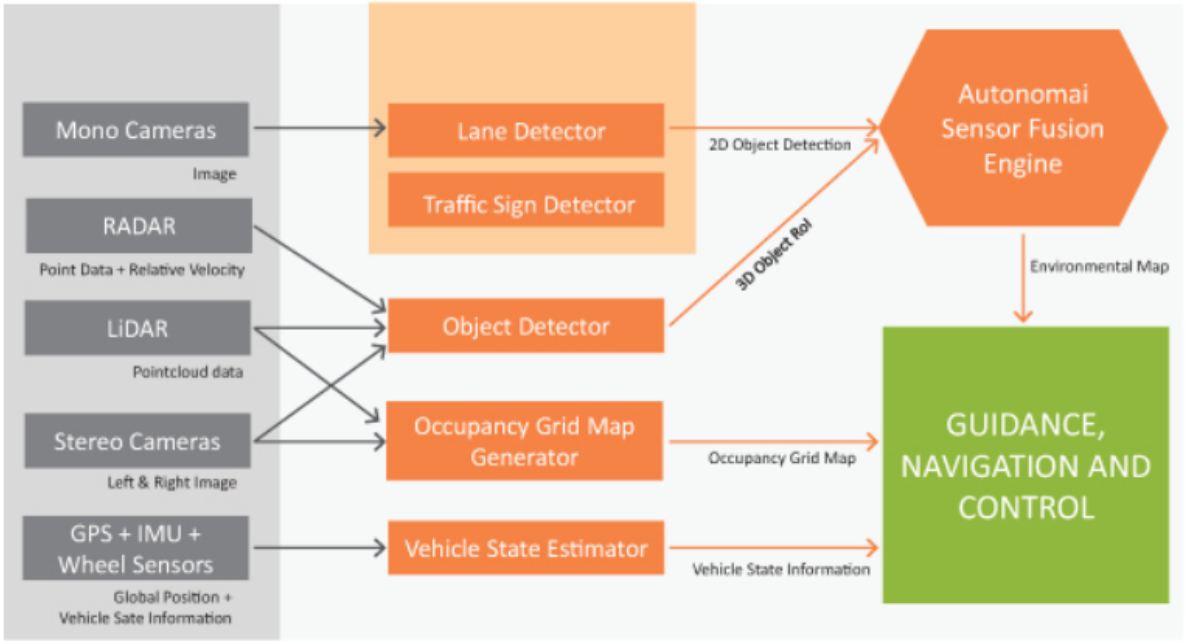

Figure 1.6: A typical perception module for an autonomous vehicle | Image Credit: (Elxsi 2020)

the classification of tree species composition of forest stand (Xu et al. 2020) which can be used for assessing ecosystem health, understanding successional processes, and digitally displaying forest biodiversity.

All of these applications are fascinating in their own right. However, the primary focus of this thesis is the application of 3-D object detection and classification for the case of autonomous driving. Although keeping the concept of transfer learning in mind, this work, with some modification, can also be extended to study the problems mentioned above.

1.2 Definitions

This section contains some important definitions which are helpful in understanding the content of this thesis.

Definition 1 (LiDAR) *LiDAR, which stands for light detection and ranging, is a remote sensing method that uses light in the form of pulsed laser to generate 3D geometric data.*

Definition 2 (Point Cloud) *A point cloud is a set of data points in space. It can be used to represent 3D geometric data. The data structure can be represented as a* $[N, D]$ *tensor where N represents the number of points and D represents the dimension associated with each point. In general for 3-D data generated by LiDAR has D equal to four where each point position has its set of Cartesian Coordinates (X, Y, Z) and reflectance (R).*

Definition 3 (Voxels) *A voxel is a unit of graphic information that defines a point in three-dimensional space. Since a pixel (picture element) defines a point in two dimensional space with its x and y coordinates , a third z coordinate is needed. In 3-D space, each of the coordinates is defined in terms of its position, color, and density.*

Definition 4 (Voxelization) *Voxelization is the process of converting a data structures that store geometric information in a continuous domain (such as a 3D triangular mesh) into a rasterized image (a discrete grid). Voxelization can be used to convert a point cloud to a 3-D voxels.*

Definition 5 (pillars) *A pillar in the context of pointpillars is a voxel with its height equal to the height of the point cloud canvas.*

Definition 6 (Speed) *In this work unless explicitly mentioned speed refers to time taken by the algorithm to run from end-to-end, that is from processing a 3-D geometric data to*

producing the bounding boxes and classification score on the provided 3-D geometric data. It is measured in frames per second (fps) or Hertz(Hz). (All speeds are evaluated on the same hardware, that is a NVIDIA-1080 Ti)

Definition 7 (Ego) *Ego is just another name for an autonomous vehicle and has been used interchangeably in this work.*

1.3 EcoPRT project at NCSU

North Carolina State University at Raleigh is a public university in North Carolina that houses over 34000 (University 2021) students. Most calsses are located on the Main campus or the Centennial campus, and thousands of students travel between them to attend classes every day. The university bus service "Wolfline" is becoming insufficient due to students' ever-growing population and is set to burden the current transit system. As a result, N.C. State University has been looking out for other options in its mobility plans. A light rail system was suggested in 2012 (Seth Hollar 2021) for handling traffic in peak hours, but the development cost of such a system was too high and was put on hold. In the year 2013, Dr. Seth Hollar and Marshal Brain suggested a cost-effective and efficient alternative to the proposed light rail system, and the Economical Personal Rapid Transit (EcoPRT) the project was initiated. The project's idea was to establish a PRT system between the Main and the Centennial Campus of the university by connecting D.H. Hill Library and James B. Hunt Library.

I got involved with the EcoPRT project in 2019, by that time they had already developed two generations of prototypes. When I got involved the work was already going on the third generation prototype. One major shortcoming of the previous generations was their inability to detect and classify dynamic objects in their path. This thesis explores a solution to this problem by proposing a novel encoder to perform the task of 3-D object detection.

1.4 Background and Literature Review

Unlike 2-D data, 3-D data can be represented in various forms, such as voxels, point clouds, and image grids. The most ubiquitously used data structure for 3-D image is point cloud. Point Clouds are a set of unordered data points in space taken from the surface of objects. They are simple and unified structures which are easy to learn from. However, point clouds are just a set of points and therefore invariant to permutation of its members. Due to

Figure 1.7: EcoPRT Project| Image courtsey:Ben Webber

the irregular nature of point clouds many researchers prefer converting them into more regular data structures such as 3-D voxels or image grids as required by typical Deep-Net Architectures. This in turn renders data unnecessarily voluminous, making the 3D convolution computationally expensive compared to 2D convolution.

The 3-D convolutional networks' arresting performance on the point cloud data-sets sparked investigation into alternative mathematical treatments that specifically exploit the sparsity and the per-mutability of point clouds. In their seminal paper Qi et al. (2017a) argued that to exploit the point cloud's unordered structure, we can use a symmetric function that yields the same result regardless of the input's order. Qi et al. (2017a) successfully demonstrated that the structure they developed was immutable to permutations in the input while also accurate. Qi et al. (2017a) argued that Neural Networks are essentially function approximator and if they could constrain the network to approximating a function which is permutation invariant then the above mentioned issue can be solved. Since symmetric functions like addition and maxpooling are permuation invariant, Qi et al. (2017a) decided to make their network emulate them. They achived this by introducing a module called PointNet Vanilla (see Figure 1.8) specifically for processing point clouds.

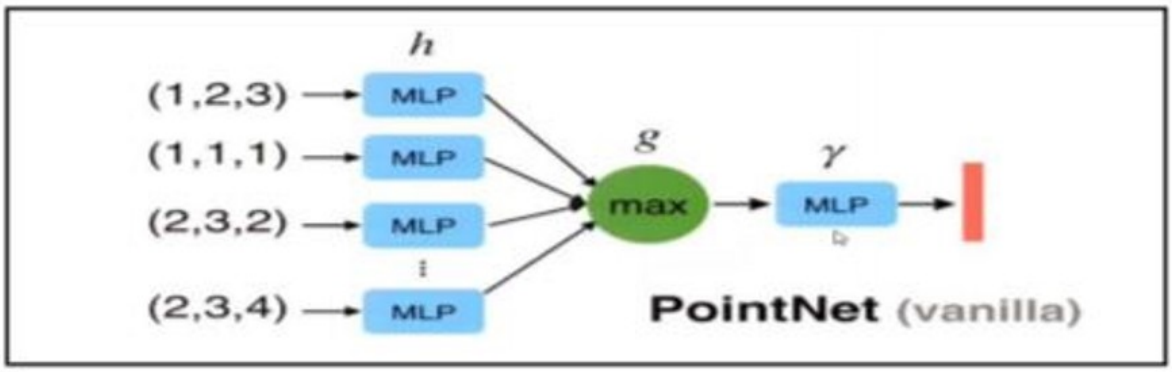

Figure 1.8: PointNet Vanilla Architecture | Image courtsey:(Qi et al. 2017a)

The introduction of PointNet paved the way for more research into the different configurations of the vanilla structure (Qi et al. 2017b; Zhou and Tuzel 2017; Ku et al. 2018; Qi et al. 2018; Li et al. 2018; Danzer et al. 2019; Li et al. 2019) which can be used for deep learning on 3-D data. The basic structure of PointNet was great at identifying/classifying objects and shapes; however, it significantly under-performed when it came to segmentation tasks.

Many papers have addressed this specific issue where they choose portions of the point cloud and processed them independently to improve the performance on datasets that require semantic segmentation. These models, however, suffer from speed and accuracy issues. VoxelNet, by Zhou and Tuzel (2017), was among the first architectures proposed as an end-to-end model to address semantic segmentation and classification within the same example. The SECOND architecture proposed by Yan et al. (2018) introduced the sparse convolution middle layer to the VoxelNet (Zhou and Tuzel 2017) architecture that significantly increases the model's speed. Lang et al. (2019) improved upon the work of Zhou and Tuzel (2017) by proposing a fast encoder called pointpillars (see Figure 1.9) which improved the speed by a significant margin while still maintaining the accuracy to state of the arts standards.

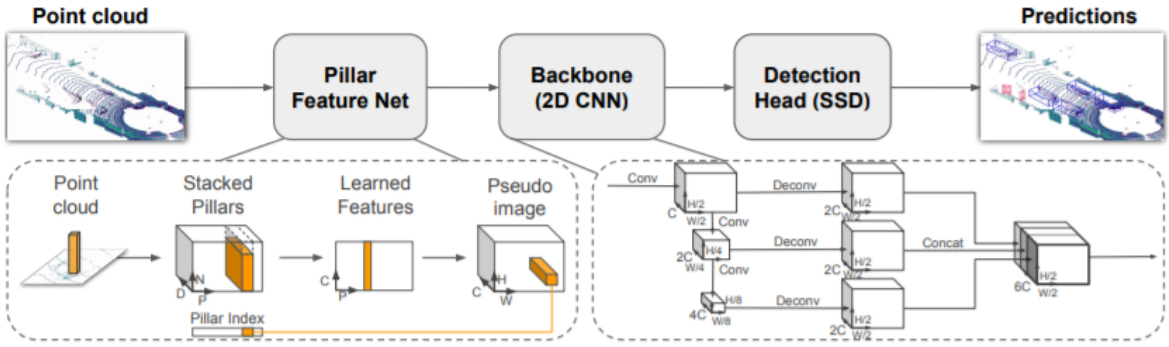

Figure 1.9: PointPillars Network Architecture | Image courtsey:(Lang et al. 2019)

1.5 Problem Statement

Lang et al. (2019) made two critical changes to the SECOND(Yan et al. 2018) architecture improving their model's speed. They specified a new form of voxelization where they only voxelized the X and Y dimensions. This binning resulted in the creation of pillars. They limited the maximum number of pillars formed in an example and the number of points within each pillar to retain dimensional consistency. Therefore, they calculated local features for their points based purely on their location concerning the pillar each point was binned into. While this solution is computationally cheaper, it is not computing the

features that are necessarily most useful.

This work proposes to replace this layer with a custom layer that uses the set abstraction and feature propagation modules proposed in Qi et al. (2017b). The set abstraction layer essentially computes features with the same dimensionality as PointPillars' Pillar Feature Net; however, unlike the original, the features calculated by this layer are done through a deep learning module that does not just take into account the features of the pillar the point belongs to, it includes the points that are in the neighborhood of the points. The set abstraction layer computes features in multiple scales in the same layer. This is analogous to passing an image through multiple kernel sizes in a computation.

1.6 Structure of thesis

This thesis has five chapters comprised of Introduction, PointPillars++, Method, Results, and conclusion to the thesis. Following paragraph provide concise information about each chapter.

Chapter 1 is an Introduction to the work, it mainly consists of Motivation behing studying this particular problem, Important definitions required to understand this work, Background information and Literature review and description of the problem which will be tackled in this work.

Chapter 2 is an explanation of one of the possible solutions to the problem of 3D-Object Detection and Classification, PointPillars++. In this Chapter the Network architecture of PointPillars++ is explained in more detail, set-abstraction layer, feature-propagation layer, Maxpooling, Pseudo Image Formation, 2D-Backbone and Detection head are all discussed in details.

Chapter 3 informs about the methodology used to reach the results. It consists details about implementation such as Network Architecture and Optimization methodology. Dataset, Data Augmentation, Network Training and Hardware used are also discussed.

Chapter 4 explains the results of the experiment and is comparative study with other existing architecture. Quantitative Analysis with respect to KITTI 3D-Object detection benchmark and KITTI BeV benchmark is presented, final output image is provided for a Qualitative analysis.

Chapter 5 presents the conclusion and summary of the research. This chapter also discusses future works which can be performed.

CHAPTER

2

POINTPILLARS++

Pointpillars++ is an extention of Pointpillars(Lang et al. 2019) architecture. In this network, the Pillar feature encoder part of the Pointpillars(Lang et al. 2019) is replaced with a new encoder based upon the work of Qi et al. (2017b) while keeping everything else virtually the same.

2.1 Network Architecture

The PointPillars++ is an end-to-end network; therefore, it accepts point clouds as input and gives bounding boxes with classification scores for each class, i.e., car, pedestrian, and cyclist, as output. We can broadly divide the PointPillars++ architecture into three parts: Pillar Feature Encoder, Backbone (2D-CNN), and the Detection Head (Single Shot Detector) as shown in Figure 2.1. Similar to Lang et al. (2019) the Pillar Feature Encoder converts a point cloud to a pseudo-image which is then used by 2D Backbone to make it ready for the Detection Head. The Detection Head then outputs bounding boxes and scores for all the available classes.

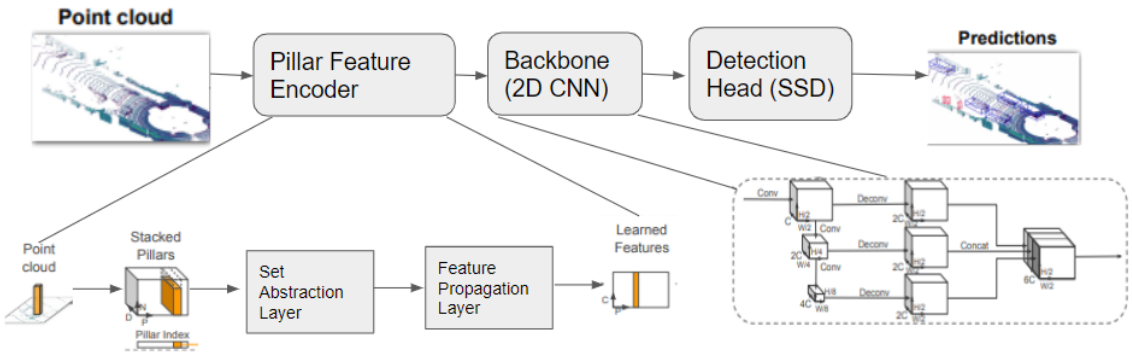

Figure 2.1: Network Architecture for PointPillars++

2.1.1 Pillar Feature Encoder: Pointclouds to Pseudo-Image

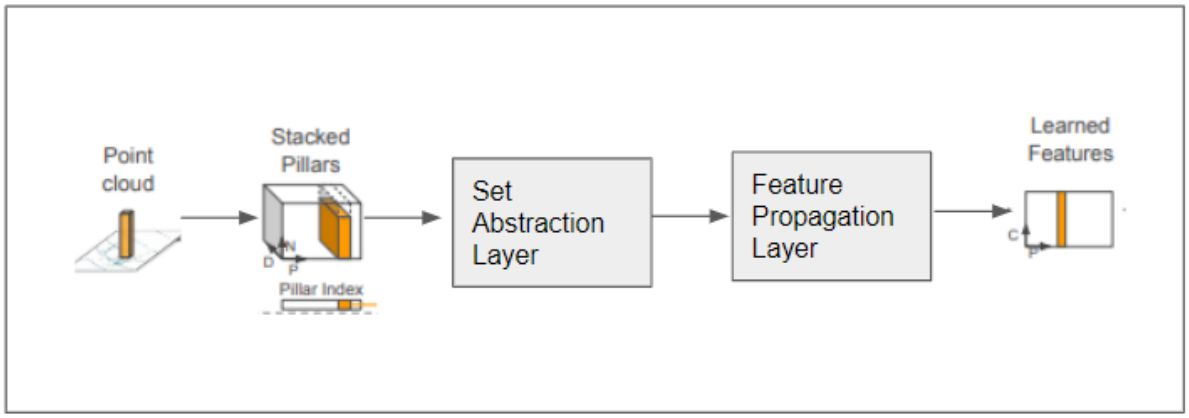

Figure 2.2: Architecture for Pillar Feature Encoder

Initially, the network computes the pillar structures similar to Lang et al. (2019) where the data is sampled and padded to the shape (P, N_p, D) where D is the number of dimensions (usually, 3 or 4 but can be augmented up to 9), P is the maximum number of pillars, and N_p is the maximum number of points per pillar.

After the voxelization, the data is then unrolled to the shape (N, D), where N is the total number of points$(N = N_p * P)$, and passed into the set abstraction layer, left side in Figure 2.3.

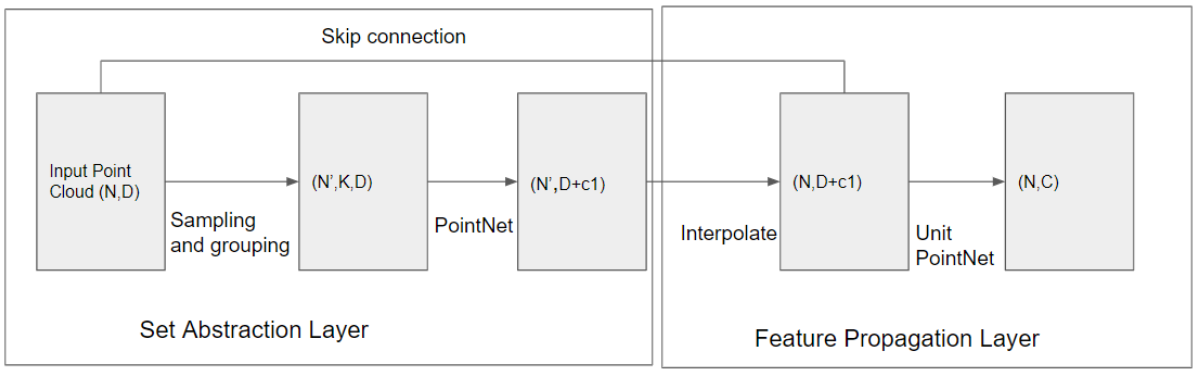

Figure 2.3: The first three blocks are part of Set Abstraction Layer while the next two are part of Feature Propogation Layer, this part of the architecture is same as Qi et al. (2017b)

Set Abstraction layer

Set Abstraction layer takes in the tensor of shape (N, D), which is sampled to obtain N' points, called centroids, based on the iterative farthest point sampling technique. Each of these points has a group of points K associated with them within a radius of r meters. These groups are then individually passed through PointNet layers that extract features for each group. These features are concatenated to respective centroids resulting in a tensor of size $(N, D + c1)$, where c1 is the extra features generated using the mentioned points. The output from this layer is passed through a feature propagation layer, similar to Qi et al. (2017b).

Feature Propogation layer

This layer interpolates the features learned for each centroid to each of the original points based on an inverse distance weighted average method to retrieve all the initially passed points to the module with all the additional features creating a tensor of size $(N, D + c1)$ This tensor is then passed through a unit pointnet to finally get (N, C) tensor. This output is reshaped back to the (P, N_P, C).

MaxPooling and Pseudo-Image formation

The previous layer's output is passed through a MaxPooling layer, Which performs a max operation over the points on (P, N_P, C) to obtain (P, C).

Once the encoded tensor (P, C) is obtained, the features are scattered back to the original pillar location to create a pseudo-image of size (C, H, W) where H, W indicate the height

and width of canvas respectively.

Multi-scale Grouping(MSG):Dealing with Non-Uniform Sampling Density

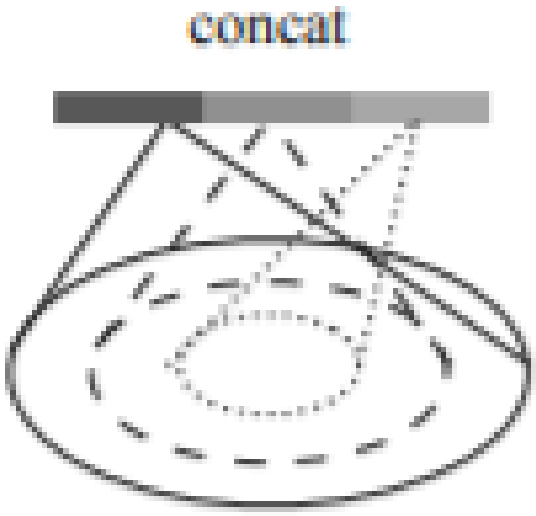

Figure 2.4: Multi-scale grouping(MSG)(Qi et al. 2017b)

It is common for point clouds to come with non-uniform density in different areas. This issue can be dealt with by using Multi-scale grouping (Qi et al. 2017b). As shown in Figure 2.4, a simple way to capture the non-uniform features on the multi-scale level is using different scales for sampling. The network can be trained to learn an optimized strategy for combining the multi-scale features; I am using the same strategy used by Qi et al. (2017b) to obtain the multi-scale features.

2.1.2 Backbone

The output of the Pillar Feature Encoder [C,H,W](pseudo-image) is then used as an input for the 2D Backbone similar to Zhou and Tuzel (2017) and Lang et al. (2019). Here the pseudo-image is passed through 2 sublayers, one of which computes top-down features while the other combines these features and upsamples them as shown in Figure 2.5.

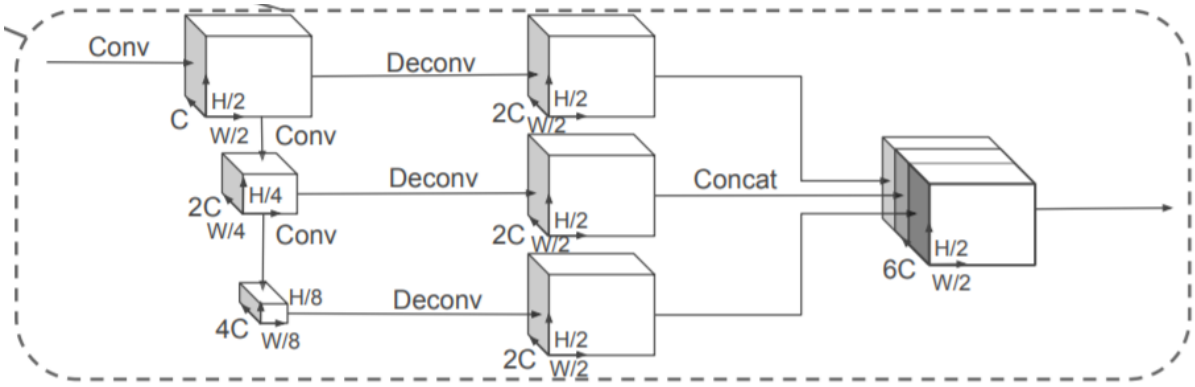

Figure 2.5: 2D Backbone

The parameters of the top-down layer can be expressed by blocks such as Block(S, L, F). Where S is the stride measured relative to the original input pseudo image, L is the dimensions of the convolution layer, and F is the number of output channels followed by batchnormalization and ReLU. The stride length used for the images are adjusted based on the input size to retain a fixed output shape; this is done by maintaining the stride of the first convolution at $\frac{S}{S_i n}$, where $S_i n$ is the stride of input blob and every subsequent convolution at stride 1.

The output of the top-down layer is upsampled, and the upsampled output is passed through a BatchNormalization layer and ReLU layer to result in features of different stride which are then concatenated to form features which will be used by detection head.

2.1.3 Detection Head(SSD)

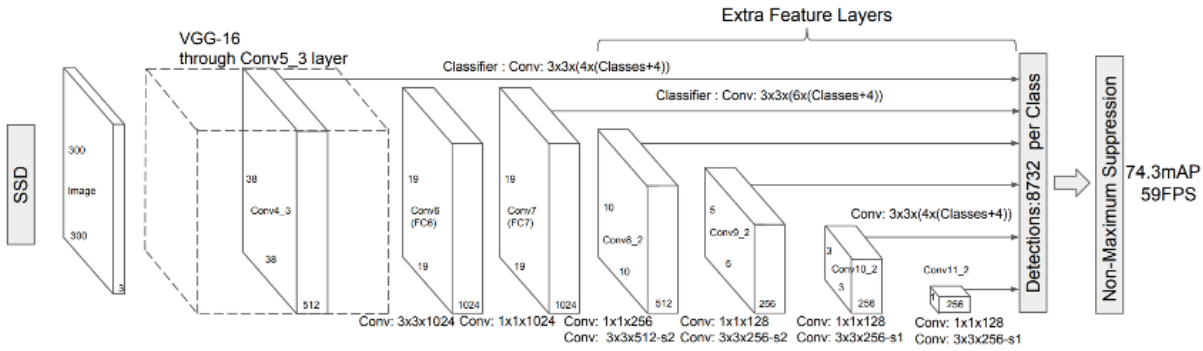

Figure 2.6: Detection Head Architecture

For the detection head I am using the Single Shot Detector (Liu et al. 2015) same as Lang et al. (2019) which uses intersection over Union (IoU) (Everingham et al. 2009) to match prior boxes to ground truth. The dimensions of the box were not used for matching but are calculated additionally using regression.

The architecture of SSD can be seen in Figure 2.6. Here it can be seen that SSD uses the feed-forward CNN approach to produce bounding boxes and scores for each class instance; after this, boxes with low scores are removed, and non-max suppression is applied on the resulting space to get the desired output.

CHAPTER

3

METHOD

To do a comparative analysis I decided to reproduce the results of Lang et al. (2019) and compare it with the results obtained on the pointpillars++ (with and without using Multi-Scale Grouping).

3.1 Implementation Details

In this section I describe the different Network Architectures of PointPillars++ and the optimization algorithms used for those architectures.

3.1.1 Network Architecture

I have used two different architectures for PP++, one which uses Multi-Scale Grouping(MSG) and the other which uses the Single-Scale Grouping(SSG). Both the architecture share the same 2D-Backbone and Detection heads. The major differentiating element for both the architectures is the way sampling is done for the Pillar Feature Encode.

Instead of pre-training the network, all weights were initialized randomly using a uniform distribution as in He et al. (2015)

Pillar Formation

l is a point in a point cloud with coordinates x, y, z and reflectance r. Just like Lang et al. (2019) the point cloud is first discretized into an evenly spaced grid in the x-y plane, creating a set of pillars P with $|P| = B$. The point in each pillar are then augmented with $x_c, y_c, z_c, x_p, and\, y_p$ where c subscript denotes distance to the airthmetic mean of all points in the pillar and the p subscript denotes the offset from the pillar x,y center. The augmented Lidar point l is now $D = 9$ dimensional.

The set of pillars will be mostly empty due to sparsity of the point cloud, and the non-empty pillars will in general have few points in them. For example, at $0.16^2 m^2$ bins the point cloud from an HDL-64E velodyne has $6k-9k$ non-empty pillars in the range typically used in KITTI for approximately 97% sparsity. This sparsity is exploited by imposing a limit both on the number of non-empty pillars per sample P and on the number of points per pillar N to create a dense tensor of size (D, P, N). If a sample or pillar holds too much data to fit in this tensor the data is randomly sampled. Conversly, if a sample or pillar has too little data to populate the tensor, zero padding is applied. Here the value of $P = 1200, N = 100, D = 9$ has been taken.

Pillar Feature Encoder with Single-Scale Grouping(SSG)

The tensor (D, P, N) is then reshaped into tensor $(D, P * N)$ and $N' = 64$ points are sampled from $P * N$ points. These points are passed throught the set abstraction layer where the radius for ball point query is taken as $r = 0.2m$. The shape of the pointnet in the setabstraction layer is set to be $[64, 64]$ with input channel equal to $6+6$. The feature propogation layer has an input channel of $64+9$ and the final pointnet has layers $[64, 64]$ which returns 64 features making the final output tensor be of shape $[N * P, 64]$.

Pillar Feature Encoder with Multi-Scale Grouping(MSG)

Similar to the above case here also the Tensor of shape (D, P, N) is first reshaped into a tensor of shape $(D, P * N)$ and then $N' = 64$ points are sampled from $P * N$ points. But now the ball query is applied at two different levels that is at radii $r = 0.1$ and $r = 0.2$. Both of these are then passed through two sets of point clouds which are shaped $[32, 64]$ and $[64, 128]$ respectively. This whole structure forms the setabstraction layer ouptup of which is then passed through the feature propogation layer which is parameterized by number of input channels $= 128+64+9$ and the pointnet of shape $[128, 64]$. The final feature map in

this case is also same as the previous case that is a tensor of shape $[N*P,64]$.

Pillar to pseudo-Image

The output of both the PFN layer of both the models is a tensor of shape $[N*P,64]$ which is then reshaped into $[P,N,64]$. This tensor then is passed through a maxpooling layer which gives the final encoded output tensor of shape $[P,64]$. Ones encoded, the features are scattered back to the original pillar locations to create a pseudo-image of size $[C,H,W]$ where $C=64$ and H and W indicate the height and width of the canvas.

2D-Backbone and Detection Head

Both the encoders have C=64 output features. These Output features are passed through 2D-Backbone. The car and pedestrian/cyclist backbones are the same except for the stride of the first block (S=2 for car, S=1 for pedestrian/cyclist). The 2-D backbone consists of 3 Blocks i.e. Block1(S,4,C), Block2(2S,6,2C), and Block3(4S,6,4C). Each block is upsampled by the following upsampling steps: Up1(S,S,2C), Up2(2S,S,2C), and Up3(4S,S,2C). Then the features of Up1,Up2, and Up3 are combined together to create 6C features for the Detection Head. The Detection Head used is same as Yan et al. (2018).

3.1.2 Optimizer and loss function

Both the networks are trained with the help of Adam optimizer(Kingma and Ba 2017), and the same loss function which is used in pointpillars(Lang et al. 2019). For the localization regression loss I am using the same loss function as used in SECOND(Yan et al. 2018). The following box encoders are used for regression target:

$$\Delta x = \frac{x^{gt}-x^{a}}{d^{a}}, \quad \Delta y = \frac{y^{gt}-y^{a}}{d^{a}}, \quad \Delta z = \frac{z^{gt}-z^{a}}{h^{a}}$$

$$\Delta w = \log\left(\frac{w^{gt}}{w^{a}}\right), \quad \Delta l = \log\left(\frac{l^{gt}}{l^{a}}\right), \quad \Delta h = \log\left(\frac{h^{gt}}{h^{a}}\right)$$

$$\Delta\theta = sin\left(\theta^{gt}-\theta^{a}\right)$$

where ground truth boxes and anchors are defined by (x,y,z,w,l,h,θ) and x^{gt} , x^{a} are respectively the ground truth and anchor boxes and $d^{a}=\sqrt{((w^{a})^{2}+(l^{a})^{2}}$. The total localization

loss is:

$$L_{loc} = \sum_{b \in (x,y,z,w,l,h,\theta)} SmoothL1(\Delta b)$$

For the localization loss we are using the focal loss(Lin et al. 2018):

$$L_{cls} = -\alpha_a (1-p^a)^{\lambda} log(p^a)$$

here the p^a is class probability of an anchor and $\alpha = 0.25$, $\lambda = 2$ which was the setting mentioned in the original paper. The total loss is therefore:

$$L = \frac{1}{N_{pos}} (\beta_{loc} L_{loc} + \beta_{cls} L_{cls} + \beta_{dir} L_{dir})$$

where N_{pos} is the number of positive anchors and $\beta_{loc} = 2$, $\beta_{cls} = 1$, and $\beta_{dir} = 0.2$.

3.2 Experimental Setup

This section contains the details of experimental setup like what data was used, what GPU was used to train the network, the experimental settings and data augmentation.

3.2.1 Dataset

All experiments are done using KITTI object detection benchmark dataset(Geiger et al. 2012). Although the KITTI benchmark contains both lidar point clouds and images, for the purpose of training PP++ only the lidar-point clouds are required and hence used. The samples are originally divided into 7481 training and 7518 testing samples. To tune the hyper-parameters the training set is further split into 3712 training and 3769 validation samples(Chen et al. 2015). After finalizing the hyper-parameters the network is retrained on 6733 training samples and 734 validation sample for creating the submission result. The KITTI benchmark consists labeled data of car, pedestrian and cyclists. Since the ground truth objects were only annotated if they are visible in the image, we follow the standard convention(Engelcke et al. 2017; Chen et al. 2017) of only using lidar points that project into the image.

3.2.2 Data Augmentation

As shown in Yan et al. (2018) and Engelcke et al. (2017), data Augmentation is a critical component to getting a good performance on KITTI detection bechmark. I am using the same technique as used by Lang et al. (2019). First, as per Yan et al. (2018), a lookup table of ground truth 3D boxes for all classes and the associated point clouds that falls inside these 3D boxes is created. Then for each sample, 15, 0, 8 ground truth samples are randomly selected for cars, pedestrians, and cyclists respectively and are placed into the current point cloud. Next, all ground truth boxes are individually augmented. Each box is rotated (uniformly drawn from $[-\pi/20, \pi/20]$) and translated (x,y, and z independently drawn from $N(0, 0.25)$) to increase the training set.

Finally, just like Lang et al. (2019) two sets of global augmentations are performed which are jointly applied to the point clouds and all the boxes. First, the random mirroring flip is performed along the x-axis(Yang et al. 2019), and a global rotation and scaling(Zhou and Tuzel 2017; Yan et al. 2018). Finally a global translation with x,y,z drawn from N(0,0.2) to simulate localization noise is applied.

3.2.3 Settings

Unless mentioned specifically the xy resolution of 0.16m, the max number of pillars(P): 1200, and the max number of points per pillar(N):100 is used as default for the entire experiment. For anchoring and matching the strategy of Zhou and Tuzel (2017) is used which is same as of Lang et al. (2019). Each class anchor is described by a width, length, height, and z center, and is applied at two orientations: 0 and 90 degrees. Anchors are matched to ground truth using the 2D IoU with the following rules. A positive match is either the highest with a ground truth box, or above the positive match threshold, while a negative match is below the negative threshold. All other anchors are ignored in the loss.

Same as Lang et al. (2019) axis aligned non-max suppression (NMS) with an overlap threshold of 0.5 IoU is applied which provides similar performance to rotational NMS, but is much faster.

Car

The x, y, z range is [(0, 70.4), (-40, 40), (-3, 1)] meters respectively. The car anchor has width, length, and height of (1.6, 3.9, 1.5) m with a z center of -1 m. Matching uses positive and negative thresholds of 0.6 and 0.45.

Pedestrian and Cyclist

The x, y, z range of [(0, 48), (-20, 20), (-2.5, 0.5)] meters respectively. The pedestrian anchor has width, length, and height of (0.6, 0.8, 1.73) meters with a z center of -0.6 meters, while the cyclist anchor has width, length, and height of (0.6, 1.76, 1.73) meters with a z center of -0.6 meters. Matching uses positive and negative thresholds of 0.5 and 0.35

3.2.4 Hardware used

Unless explicitly mentioned the hardware used for training the network and evaluating performance is a gtx 1080 Ti.

3.3 Training the Network

Both the networks were trained on two GTX 1080 Tis for 148480 steps. Batch size used for training the network was 1. Each epoch consisted of 928 steps making the entire training to be of 160 epochs. The network was evaluated after every 37120 steps for the performance. It was found that for PointPillars++ with SSG the optimal number of steps required for training were 148480 as can be seen in Figures 3.3 and 3.1.

For PointPillars++ with MSG the optimal number of steps for training were a little less then 148480 as can be seen from Figures 3.4 and 3.2.

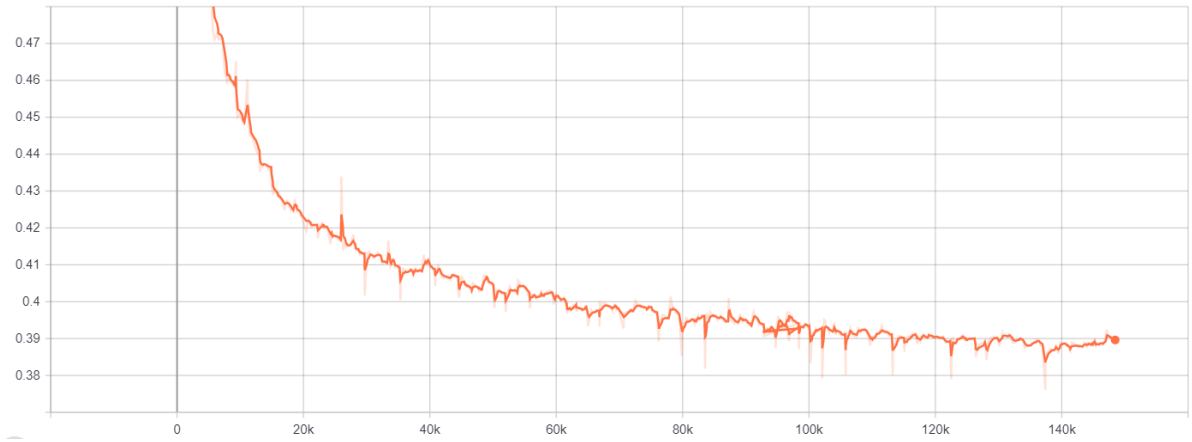

Figure 3.1: The classification loss per step while training PointPillars++ with SSG

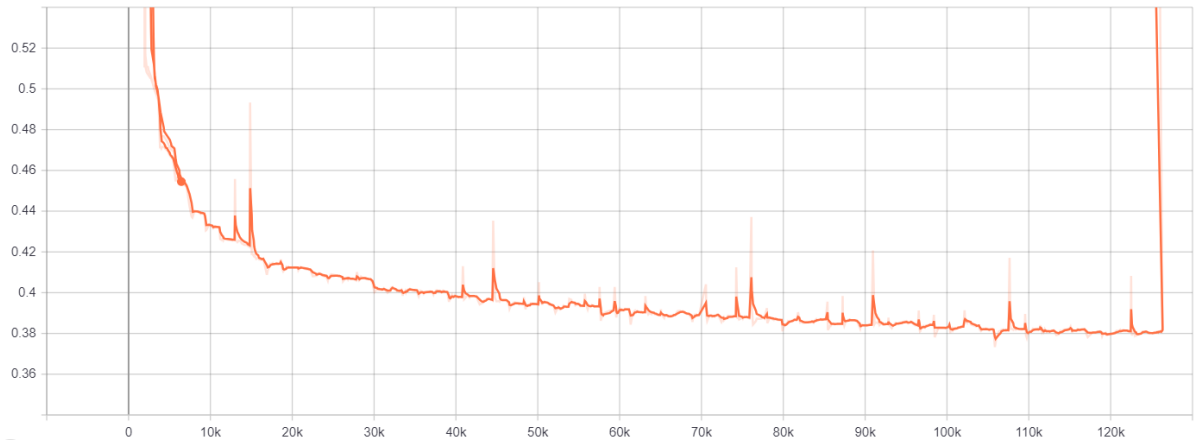

Figure 3.2: The classification loss per step while training PointPillars++ with MSG

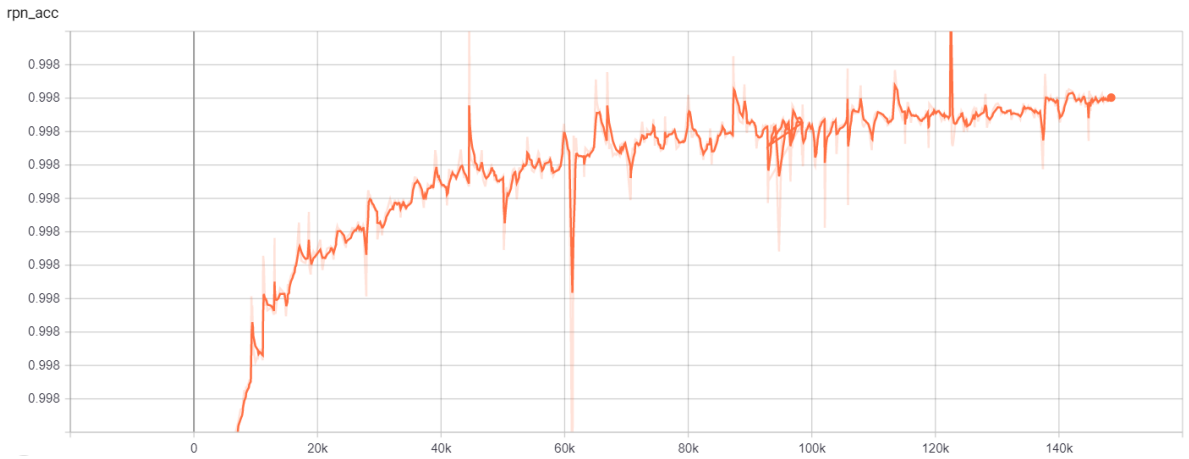

Figure 3.3: The reason proposal network accuracy per step while training PointPillars++ with SSG

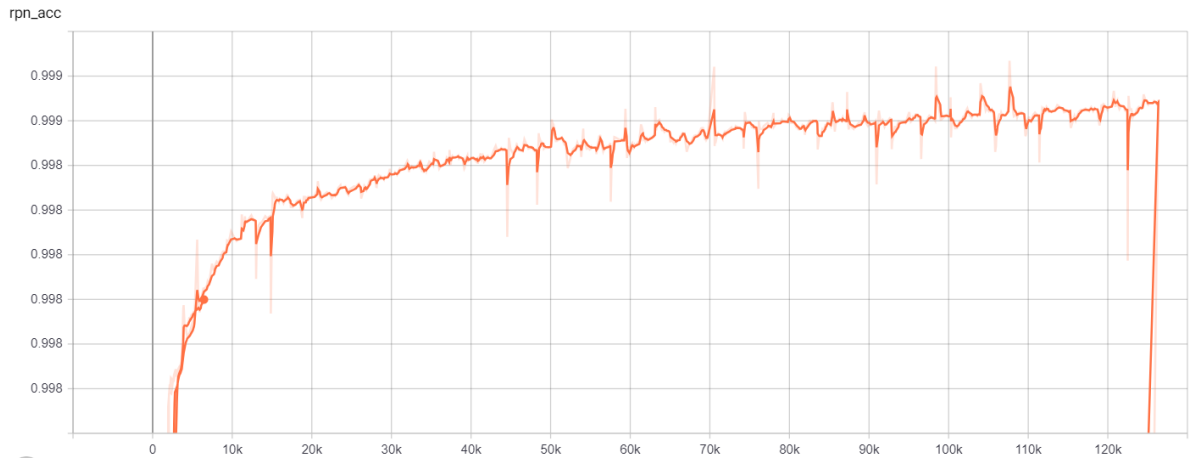

Figure 3.4: The reason proposal network accuracy per step while training PointPillars++ with MSG

After training the SSG network for 160 epochs and MSG network for 130 epochs the evaluation of both the networks on a KITTI 3-d object detection benchmark and KITTI birds eye view detection benchmark was done.

CHAPTER

4

RESULTS

This work can be viewed as an effort to utilize the findings of Qi et al. (2017b) and Lang et al. (2019) in creating a 3-D object detection and classification system for an autonomous vehicle. To find out the effectiveness of the created 3-D object detection system (PointPillars++), a comparative study is performed between PointPillars++, PointPillars, VoxelNet and SECOND.

4.1 Quantitative Analysis

The performance of PP++ with other state of the art methods has been evaluated with the help of KITTI evaluation detection metrics which are: 3-D detection benchmark and Birds Eye View (BeV) detection benchmark.

KITTI evaluates the 3-D object and bird's eye view detection performance using the PASCAL criteria. Far objects are thus filtered based on their bounding box height in the image plane. As only objects also appearing on the image plane are labeled, objects in don't care areas do not count as false positives. It is noted that the evaluation does not take care of ignoring detections that are not visible on the image plane - these detections might give

rise to false positive. For cars overlap of 70% is required while for pedestrians and cyclists 50% overlap also works.

4.1.1 3-D detection benchmark

Table 4.1: Comparison between Voxelnet (Zhou and Tuzel 2017), SECOND(Yan et al. 2018), PointPillars (Lang et al. 2019), and PointPillars++(MSG and SSG) on the KITTI test 3D detection benchmark.

Method	mAP	Car			Pedestrian			Cyclist		
		Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
VoxelNet	49.05	77.47	65.11	57.73	39.48	33.69	31.5	61.22	48.36	44.37
SECOND	56.96	83.13	73.66	66.20	51.07	42.56	37.29	70.51	53.85	46.90
PP	59.20	79.05	74.99	68.30	52.08	43.53	41.49	**75.78**	59.07	52.92
PP++(SSG)	66.02	90.77	89.31	88.37	52.79	**46.70**	**45.06**	71.90	55.89	53.44
PP++(MSG)	**66.83**	**92.09**	**90.21**	**89.43**	**54.63**	46.63	43.11	69.40	**59.49**	**56.53**

KITTI has divided there evaluation on three strata that is easy, medium and difficult. As can be seen from Table 4.1, both variants of PP++, that is the PP++ with MSG and PP++ with SSG, outperforms other state of the art methods across all classes and difficulty strata except for the easy cyclist stratum. It is also worth noting that in general PP++ with MSG does better than PP++ with SSG. The mean average precision(mAP) of 66.83 in case of PP++ with MSG and 66.02 in case of PP++ with SSG is a significant improvement compared to the current state of the art method (59.02 for PP).

4.1.2 BeV detection benchmark

As shown in Table 4.2 PP++ again outperforms other state of the art methods in terms of the mean average precision(mAP) and in all difficulty strata of Car class. But when comparing across the Pedestrian and Cyclist classes PP outperforms PP++ across all difficulty stratum.

4.1.3 Real Time Inference

As indicated in the Table 4.3, PP++ with SSG and PP++ with MSG are able to achieve the speed of 12.4 Hz and 10.3 Hz respectively. The main inference steps are as follows. First,

Table 4.2: Comparison between Voxelnet (Zhou and Tuzel 2017), SECOND(Yan et al. 2018), PointPillars (Lang et al. 2019), and PointPillars++(MSG and SSG) on the KITTI test Birds Eye view detection benchmark.

Method	mAP	Car			Pedestrian			Cyclist		
		Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
VoxelNet	58.25	89.35	79.26	77.39	46.13	40.74	38.11	66.70	54.76	50.55
SECOND	60.56	88.07	79.37	77.95	55.10	46.27	44.76	73.67	56.04	48.78
PP	66.19	88.35	86.10	79.83	**58.66**	**50.23**	**47.19**	**79.14**	**62.25**	**56.00**
PP++(SSG)	66.33	90.77	**89.65**	**89.07**	52.85	46.89	45.31	71.90	56.33	54.22
PP++(MSG)	**66.36**	**91.13**	88.94	88.27	54.77	47.07	42.36	70.19	59.64	54.87

Table 4.3: Comparison between Voxelnet (Zhou and Tuzel 2017), SECOND(Yan et al. 2018), PointPillars (Lang et al. 2019), and PointPillars++(MSG and SSG) on the real time inference speed.

Method	Modality	Speed(Hz)	mAP
VoxelNet	LIDAR	4.4	49.05
SECOND	LIDAR	20	56.69
PP	LIDAR	62	59.20
PP++(SSG)	LIDAR	12.4	66.02
PP++(MSG)	LIDAR	10.3	66.83

the point cloud is loaded and filtered based on range and visibility in the images. Then, the points are organized in pillars and decorated. Next, the tensor is uploaded to GPU, encoded and gets scattered to a pseudo-image. This pseudo image is then processed by backbone and detection heads. Finally non-max suppression is applied on the CPU for a total run-time of 81 and 97 ms for PP++ with SSG and PP++ with MSG respectively.

4.2 Qualitative Analysis

The qualitative result is provided in Figure 4.1. While PointPillars++ is only trained on lidar point clouds, for ease of interpretation the 3-D bounding boxe prediction from the image perspective. Figure 4.1 shows the detection results with respect to 3-D bounding boxes. The inference is produced with the help of kitti visualizer, which restricts the use of non max suppression and hence there are multiple false negatives which are present in the Figure. The prediciton for cars are particularly accurate and common failure mode include false positives which can be occluded with the help of non max suppression.

Figure 4.1: From left to right: 2-D image of Ground Truth, Point Cloud of Ground Truth with Bounding Boxes, The result after inference

CHAPTER

5

CONCLUSION

5.1 Summary of Results

After extensive experimentation it is been observed that both the variants of PointPillars++ perform significantly better then PointPillars and other state-of-the-art algorithms on KITTI 3D-detection benchmark and KITTI birds eye view detection benchmark. The mean Average precision of PointPillars++ with SSG and PointPillars++ with MSG are 66.02 and 66.83 respectively on 3-D detection benchmark and 66.33 and 66.36 respectively on Birds Eye View detection benchmark. KITTI ranks the performance of each algorithm based on how the algorithm is doing on the medium difficulty strata, PP++ with SSG reaches the average precision, measured with respect to 3-D detection benchmark, of $(89.31, 46.70, 55.89)$ on the medium difficulty strata for car,pedestrian and cyclist respectively while PP++ with MSG is able to reach the average precision of $(90.21, 46.63, 59.49)$ on the same difficulty strata and classes.

When measuring the average precision with respect to birds eye view detection benchmark for the medium difficulty stratum of car, pedestrian, and cyclist, PP++ with SSG achieves $(89.65, 46.89, 56.33)$ while PP++ with MSG achieves $(88.94, 47.07, 59.64)$.

Both PP++ with MSG and PP++ with SSG achieve are able to achieve a realtime inference speed greater then $10Hz$ (PP++(MSG) with 10.3 Hz and PP++(SSG) with 12.4 Hz).

5.2 Overarching Conclusion

PointPillars (Lang et al. 2019) reached remarkable speed of 62 Hz which was 3 times better than the second best while maintaining the state-of-the-art accuracy. Although since PointPillars utilizes pointnet, it fails to capture the local substructure induced by the metric space points live in, limiting its ability to recognize fine-grained patterns and generalizability to common scenes. In this work I propose PointPillars++ which can be used to solve the above mentioned problem. After extensive experimentation it can be shown that PointPillars++ is able to outperform PointPillars in terms of mean average precision on both Birds Eye View detection benchmark and 3D-object detection benchmark (see Tables 4.1 and 4.2).

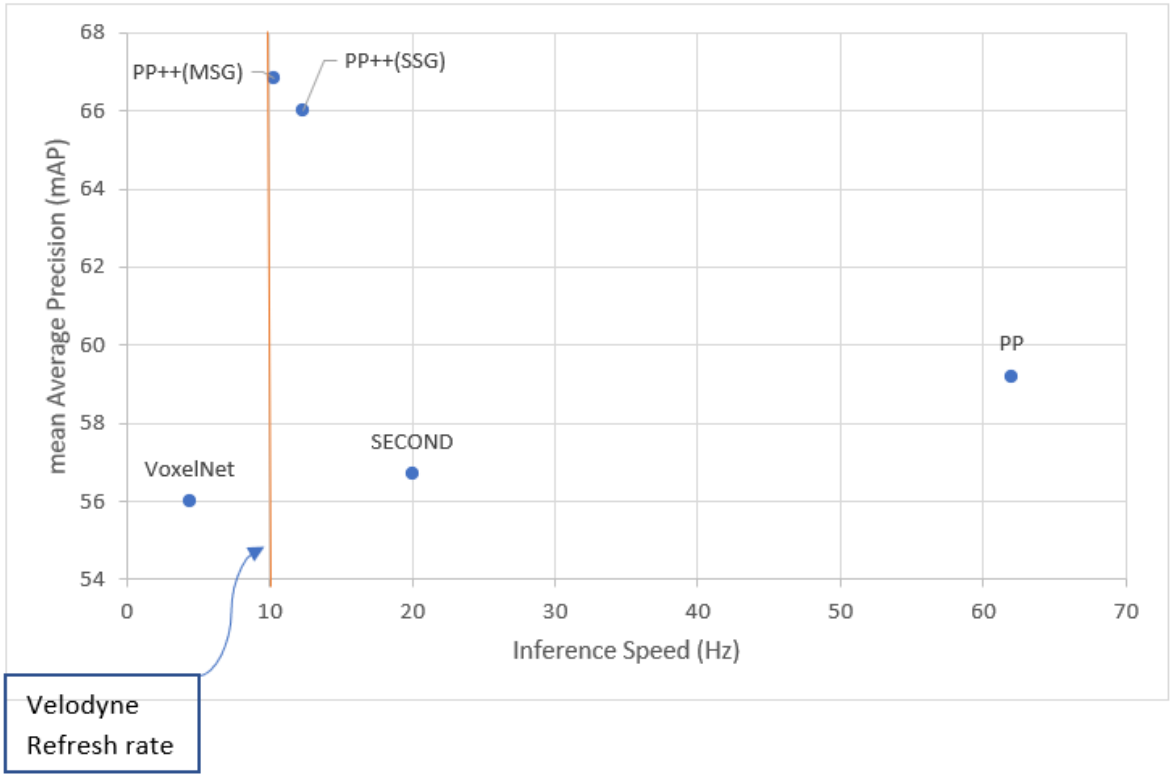

Figure 5.1: 3D-object detection benchmark performance(mAP) vs inference speed(Hz) on KITTI test set across pedestrian,cyclist and car classes. The red line represents the refresh rate of a HDL-32E velodyne sensor.

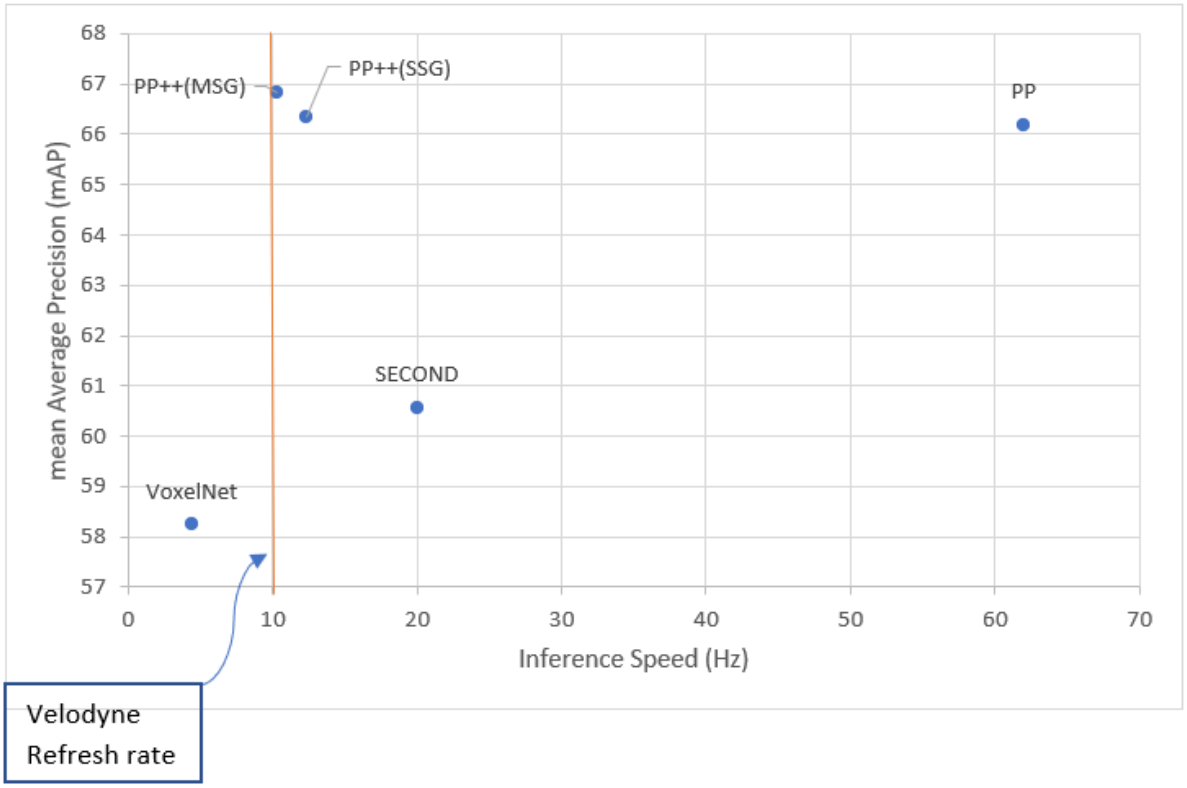

Figure 5.2: Birds eye View detection benchmark performance(mAP) vs inference speed(Hz) on KITTI test set across pedestrian,cyclist and car classes. The red line represents the refresh rate of a HDL-32E velodyne sensor.

It is essential to consider the inference speed of any algorithm which is to be used for 3D-Object detection in an autonomous vehicle. And it is evident that both variants of PP++ are comparatively on the slower side (see Table 4.3) but it is also imperative to consider that the speed of real time inference depends not only on the speed of algorithm but also on the refresh rate of the sensor. Most lidar sensors have a refresh rate between 5-20 Hz. In the ecoPRT project we are using velodyne HDL-32E for point cloud generation. The optimal refresh rate for this sensor is kept at 10 Hz and since both PP++ with MSG and PP++ with SSG perform significantly better then other state of the art methods in terms of mAP and are faster then 10 Hz (see Figures 5.1 and 5.2), they can be considered a viable replacement for existing methods.

5.3 Future Work

5.3.1 Increasing the speed of PointPillars++

Even though PointPillars++ is sufficiently fast for a lidar sensor with 10 Hz refresh rate, it can become a bottleneck in future when faster and more robust lidar sensors become available. Also, having a faster algorithm can help reduce the required computational power, ultimately resulting in the reduction of cost. Although it is not trivial but the speed of PointPillars++ can be improved by using a combination of following techniques.

Increasing the X-Y resolution of point cloud

In their work Lang et al. (2019) show that by increasing the x,y resolution of point cloud they can also increase the speed of real time inference. This increase in speed comes with the cost of reducing the mean average precision. In this work the x-y resolution of $0.16^2 m^2$ has been used, In future works the performance can be evaluated with x-y resolutions of $(0.20^2 m^2, 0.24^2 m^2, and 0.28^2 m^2)$.

Reducing the number of sampled point in the set-abstraction layer of Pillar Feature Encoder

The slowest part of the PointPillars++ algorithm is ball query in set-abstraction layer on all the sampled points. In this work the number of sampled point is taken as 64, in future works the effect of reducing these points can be studied. Doing so should improve the speed of the algorithm by a significant amount but it will also affect the mean average precision and this trade off needs to studied.

5.3.2 Evaluating the performance of PointPillars++ on NuScenese Dataset

The nuScenes dataset is a public large-scale dataset for autonomous driving developed by the team at Motional. It contains 10 times as much data as KTITTI Object detection dataset. In future the network can be trained on nuScense dataset and evaluated on nuScenes 3-D object detection benchmark to compare the performance of PointPillars++ with other state of the art methods. Since the amount of data in nuScense is very large it is advisable to use the pretrained network to reach a faster convergence which will allow the reduction of training time.

REFERENCES

Administration, N. H. T. S. (2021). Fars encyclopedia.

Carlson, C. F. (2021). Imaging science fundamentalschester f. carlson center for imaging science the human visual system the eye. - ppt download.

Chen, X., Kundu, K., Zhu, Y., Berneshawi, A. G., Ma, H., Fidler, S., and Urtasun, R. (2015). 3d object proposals for accurate object class detection. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Chen, X., Ma, H., Wan, J., Li, B., and Xia, T. (2017). Multi-view 3d object detection network for autonomous driving.

Cheung, K. (2018). World's top 33 companies working on self driving cars - algorithm-x lab.

Cottam, B. J. (2018). Transportation planning for connected autonomous vehicles: How it all fits together. *Transportation Research Record*, 2672(51):12–19.

Danzer, A., Griebel, T., Bach, M., and Dietmayer, K. (2019). 2d car detection in radar data with pointnets. *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 61–66.

Elxsi, T. (2020). Tata elxsi - autonomai.

Engelcke, M., Rao, D., Wang, D. Z., Tong, C. H., and Posner, I. (2017). Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks.

Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J., and Zisserman, A. (2009). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–308. Printed version publication date: June 2010.

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

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

Hedges and Company (2021). Us vio vehicle registration statistics: See how many cars in the us.

Hua, X., Wang, X., Wang, D., Huang, J., and Hu, X. (2018). Military object real-time detection technology combined with visual salience and psychology. *Electronics*, 7(10).

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Krishnan, N. (2021). How autonomous vehicles drive technological innovation | cleantech group.

Ku, J., Mozifian, M., Lee, J., Harakeh, A., and Waslander, S. (2018). Joint 3d proposal generation and object detection from view aggregation.

Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. (2019). Pointpillars: Fast encoders for object detection from point clouds.

Li, X., Guivant, J., Kwok, N., and Xu, Y. (2019). 3d backbone network for 3d object detection. *ArXiv*, abs/1901.08373.

Li, Y., Bu, R., Sun, M., Wu, W., Di, X., and Chen, B. (2018). Pointcnn: Convolution on $\mathcal{X}$-transformed points.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2018). Focal loss for dense object detection.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C. (2015). SSD: single shot multibox detector. *CoRR*, abs/1512.02325.

MacroTrends (2021). U.s. population 1950-2021 | macrotrends.

Nevalainen, O., Honkavaara, E., Tuominen, S., Viljanen, N., Hakala, T., Yu, X., Hyyppä, J., Saari, H., Pölönen, I., Imai, N. N., and Tommaselli, A. M. G. (2017). Individual tree detection and classification with uav-based photogrammetric point clouds and hyperspectral imaging. *Remote Sensing*, 9(3).

Qi, C. R., Liu, W., Wu, C., Su, H., and Guibas, L. J. (2018). Frustum pointnets for 3d object detection from rgb-d data.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space.

Rosique, F., Navarro, P. J., Fernández, C., and Padilla, A. (2019). A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3).

Seth Hollar, N. P. (2021). A new low cost, efficient, self-driving personal rapid transit system.

Singh, S. P., Wang, L., Gupta, S., Goli, H., Padmanabhan, P., and Gulyás, B. (2020). 3d deep learning on medical images: A review.

University, N. C. S. (2021). Fast facts – institutional strategy and analysis.

Xu, Z., Shen, X., Cao, L., Coops, N. C., Goodbody, T. R., Zhong, T., Zhao, W., Sun, Q., Ba, S., Zhang, Z., and Wu, X. (2020). Tree species classification using uas-based digital aerial photogrammetry point clouds and multispectral imageries in subtropical natural forests. *International Journal of Applied Earth Observation and Geoinformation*, 92:102173.

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

Yang, B., Luo, W., and Urtasun, R. (2019). Pixor: Real-time 3d object detection from point clouds.

Zhou, Y. and Tuzel, O. (2017). Voxelnet: End-to-end learning for point cloud based 3d object detection.