Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Comparative Study of 3D Object Detection Frameworks Based on LiDAR Data and Sensor Fusion Technologies

Sreenivasa Hikkal Venugopala

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Paul G. Plöger
Dr. Anastassia Kuestenmacher
M.Sc. Deebul Nair

August 12, 2020

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Sreenivasa Hikkal Venugopala

Abstract

Estimating and understanding the surrounding of the vehicle precisely forms the basic and crucial step for the autonomous vehicle. The perception system plays a significant role in providing an accurate interpretation of a vehicle's environment in real-time. Generally, the perception system involves various subsystems such as localization, obstacle (static and dynamic) detection, and avoidance, mapping systems, and others. For perceiving the environment, these vehicles will be equipped with various exteroceptive (both passive and active) sensors in particular cameras, Radars, LiDARs, and others. These systems are equipped with deep learning techniques, that transform the huge amount of data from the sensors into semantic information on which the object detection and localization tasks are performed. For numerous driving tasks, to provide accurate results, the location and depth information of a particular object is necessary. 3D object detection methods, by utilizing the additional pose data from the sensors such as LiDARs, stereo cameras, provides information on the size and location of the object. Based on recent research, 3D object detection frameworks performing object detection and localization on LiDAR data and sensor fusion techniques show significant improvement in their performance. In this work, a comparative study of the effect of using LiDAR data for object detection frameworks and the performance improvement seen by using sensor fusion techniques are performed. Along with discussing various state-of-the-art methods in both the cases, performing experimental analysis, and providing future research directions.

Acknowledgements

I would like to thank my parents H.K. Venugopala, Saroja D. Kulakarni and my sister Vidya Vijay for providing all kinds of support for pursuing my research work.

I would like to thank my supervisors Prof. Dr. Paul G. Plöger, Dr. Anastassia Kuestenmacher, and M.Sc. Deebul Nair, for guiding along the course of the research project and helping in stepping forward in the correct path for completing the research project.

I would like to thank Iman Awaad for providing moral support in difficult times and guiding in enhancing the report with various comments and suggestions.

I would like to thank Swaroop Bhandary, and Shravanthi for their support in various technical aspects.

Contents

List of Figures

List of Tables

1

Introduction

Perception is the process of understanding the scene or environment by interpreting and organizing the information acquired through the sensory system. It is one of the basic building blocks of an autonomous system. In the case of autonomous vehicles, perception refers to the representation of the vehicle's environment. Perception system is categorized into various subsystems such as object detection and tracking system that is responsible for keeping a track of detected objects, mapping system responsible for building maps (both global and local) for path planning and obstacle avoidance, traffic signal detection system responsible for reading and communicating the signal information, and so on. A major challenge in scene understanding is the detection and tracking of the objects with respect to the ego vehicle.

Following properties is a must for the perception system of an autonomous vehicle[16]:

(a) Accurate: the system should provide precise information about the vehicle's environment, which helps in the safe maneuverability of the vehicle.

(b) Robust: the system must be capable of handling unseen situations, adverse weather conditions, and sensor failures.

(c) Real-time: data representation, detection, and classification must be performed in real-time to support other dependent systems.

To move towards achieving these properties, autonomous vehicles are equipped with various automotive-grade sensors such as LiDAR, Radar, cameras, and so on. Each sensors exhibits its superior qualities and inferior qualities under various circumstances. To exploit the complementary properties of these sensors, based on the different driving scenarios the autonomous vehicle chooses to fuse the multi-modal sensor information or to rely on a single sensing modality. In this work, we mainly concentrate on one fundamental problem concerning to perception system, namely, object detection.

Object detection is the process of recognizing, classification, and localization of multiple classes of objects in the vicinity of the sensors. These objects are also referred to as road agents and failing to detect these agents might result in safety-related catastrophic events. Deep learning techniques have shown significant improvement in performance in the detection and classification of the road agents. With a large amount of data from the sensor system, deep learning frameworks are powerful in learning the hierarchical feature representations[32]. Based on the learning capability, many of the proposed methods employ deep learning techniques for performing object detection on single sensing modality and also to fuse multiple sensing modalities and perform object detection.

In this regard, recent research shows that few of the deep neural networks performing object detection on single modality (LiDAR) perform as promising as the sensor fusion based networks. In this research work, a comparative study of 3D object detection frameworks for autonomous vehicles based on LiDAR data and sensor fusion technologies is performed.

This report provides a summary of various state-of-the-art frameworks for 3D object detection, different sensors used in autonomous vehicles and criteria for choosing the sensors, a summary on the multi-modal benchmark datasets, and comparative analysis on the performance of two deep neural network frameworks based on LiDAR data and sensor fusion.

1.1 Problem Statement

The perception system plays an important role in maneuvering the autonomous vehicle safely, without causing any distress to the ego vehicle and its surroundings. An accurate and high-performance perception system is responsible for creating a representation of the vehicle's environment, which includes, localizing the ego vehicle, detection of road and the static objects, detection, and tracking of the dynamic objects for creating a collision-free path for safe maneuvering.

For creating this internal representation, the autonomous vehicles are equipped with many automotive-grade sensors such as cameras, LiDAR, Radar, and so on. These sensors, if used individually, provide the representations in different formats and due to the downsides of the sensors, using individual sensors is not recommended. Alongside, these sensors generate a huge amount of data that is difficult to be handled in real-time conditions. To this extent, deep learning techniques are employed for processing the incoming data and discarding the unnecessary data.

Creating a collision-free path requires precise detection and localizing of the dynamic objects. 3D object detection techniques together with deep learning approaches provide accurate and quicker responses for the detection and estimation of objects. Due to the depth and size information that represents the physical characteristics of the objects, 3D estimations are necessary and are efficient for generating the collision-free representations.

Leveraging the complementary information generated by the multiple sensors, deep learning-based 3D object detection methods generate a high-resolution representation of the environment by fusing the data from multiple sources and estimates the 3D location, size regressions, and orientations of the object. This helps in handling adverse weather effects, lighting conditions, decreasing the processing time of data, tackling occlusions, and avoiding collisions in real-time.

Recent advancement shows that fusing the data from multiple sources at different stages of processing results in rich representations which in turn supports accurate and precise 3D object detection. On the contrary, the LiDAR sensor is capable of providing the 3D data and is not vulnerable to change in lighting conditions, and recent research shows that the 3D detections performed using only LiDAR data involving deep learning techniques provide promising performance similar to sensor fusion-based methods.

In this work, a comparative study on the performance of the frameworks which perform object detection based on LiDAR data and the frameworks which perform object detection based on sensor fusion (multi-modal data) techniques is performed. A detailed analysis on various sensors involved in perception

system, their highlights, and lowlights, along with various state-of-the-art detection frameworks that are based on LiDAR data and sensor fusion techniques are provided. Various experiments are conducted and results are discussed to arrive at a conclusion that supports the comparative study.

1.2 Challenges and Difficulties

Various factors affect the performance of the perception system out of which a few of them pose major challenges. One such challenge is related to sensor limitations and adverse weather conditions. Sensors are the main source for acquiring information on the vehicle's environment, limitations in sensing the environment result in adverse effects on driving tasks which may lead to road accidents. Few sensors will not work as expected under adverse weather conditions (more information is given in Chapter 3) resulting in ineffective perception results.

An additional challenge is posed by the generalization capability of the system across various driving scenarios. For example, the scenarios include driving in the highway, urban conditions, and rural areas. In cases of highways, the traffic will be structured and there will be few object classes to detect and distinguish. In the case of urban and rural driving scenarios, the traffic will not be well structured and there will be multiple classes of objects such as pedestrians, cyclists, vehicles, and few of the background objects like bollards and bins[3].

A further possible technologically challenging scenario is related to obstacle avoidance. For obstacle avoidance, the control signals must be based on the perceptual information generated based on accurate scene interpretation. Even if the vehicle is equipped with some proprioceptive sensors like IMU, GPS, accurate information from the perception system is necessary for localization and determining the obstacle-free path[42].

Another well-posed challenge is occlusion. Occlusion occurs when a certain object blocks another object which is in the view of the ego vehicle resulting in complete or partial visibility of the object. Along with this, the variations in the sizes of the object affects the sensor readings[3].

A few of the major challenges and difficulties faced while performing this work are:

- *Cluster machine handling*: Accessing and configuring the cluster machine for storage of the data, code, and results were initially tricky for the first time user. Copying files from local machine to cluster machine and from cluster machine to local machine using secure copy commands needed support from experts. As a beginner, configurations needed for executing a particular project and submitting the batch job for accessing and utilizing the GPU machines were tricky and difficult.
- *Dataset download and storage*: This research work requires a large data storage capacity for storing the datasets, code, processed data, and the experimental results. The primary storage allocated for a user is around 80 GB and that is not sufficient for storing two datasets and executing two projects. The secondary storage of 5 TB per user is allocated, which was not sufficient for storing the entire dataset, hence we had to make few workarounds by choosing a subset of the dataset at random for conducting the experiments and the access time was more for accessing the secondary storage.
- *Cluster nodes with GPU access*: For one of the inferencing tools, it was required to work with GPUs directly and there are few cluster nodes through which GPU can be directly accessed without the

need for submitting the batch job. But in this case, the inferencing tool needed a display device attached to the machine for inferencing, hence a laptop with GPU was collected from the university to execute the tool and inference the results.

- *Getting the code to run*: For getting the initial code to execute without any failures took a lot of time due to the lack of necessary information provided on executing the code, configuring cluster machine for a particular code and accessing required components such as GPU, more memory (RAM), configuring GPU specific configurations, and delay in responses to the issues created in GitHub pages for the specific issues faced related to the code.
- *Handling WAYMO Open[65] dataset*: The codebase used for experiments here is specific to the KITTI dataset[18], and the WAYMO dataset is in TFRecord format. To use the TFRecord files directly, the code should be modified immensely and that itself consumes a lot of time compared to performing the research. Hence it was converted into KITTI format. With this converted data, it was simpler and similar to use with a few minor modifications to the code.

1.3 Motivation

For a self-driving vehicle, object detection and classification becomes a crucial part of the perception system. Although, various state-of-the-art methods work on different sensor modalities like camera, LiDAR, and fusion of various sensor modalities and provide accurate detections, each of them has its limitations. The main focus of these object detectors should not only be on accurate detections but it should perform in real-time.

In spite of the various challenges and difficulties mentioned under Section 1.2, the 2D object detection techniques that are proposed for autonomous vehicles have been immensely improved and many of these techniques have been published online with an average precision of around 90%[3] based on KITTI dataset[18]. These 2D object detection techniques detect the object in the image plane, but the major disadvantage is that they provide limited information. These techniques do not provide the information related to objects' pose, depth, and occlusion details[3].

In the case of 3D object detection techniques, the size of the object and the position in world frame is obtained using the 3D bounding boxes. The major disadvantage here is that the depth estimation and the regression of extra dimension increases the network complexity. The third dimension which is available in the 3D data provides information related to the localization of the object, regression of the sizes of various objects, and depth information concerning world coordinates[3]. These information provides a better understanding of the vehicle's surrounding in three-dimension which is necessary for autonomous driving tasks.

Recent research in the field of object detection and classification shows that the performance of the deep neural networks performing 3D object detection using LiDAR data is as promising as the performance of the networks based on sensor fusion techniques. This motivates us to perform a comparative study on the performance of two deep learning frameworks that are based on LiDAR data and sensor fusion techniques.

1.4 Contributions

This research provides an overview of various approaches for performing 3D object detection in the field of autonomous driving, along with a description of sensors involved, different sensor fusion techniques, and the datasets. Based on the type of inputs used for the detection frameworks, the object detection frameworks are divided and are presented in further sections. In the end, future work and research challenges are discussed. Contributions of this research can be summarized as follows:

- Summarizing various 3D object detection frameworks based on the type of input data used, along with their contributions and lowlights.
- Summarizing various sensors involved in the perception of the environment along with their selection criteria and summarizing the classical and deep learning approaches for sensor fusion.
- Provide a summary of various multi-modal datasets that are available for research purposes, together with detailed summary of various state-of-the-art 3D object detection frameworks based on LiDAR data and sensor fusion techniques.
- Provide discussions on the selection of datasets and frameworks by describing various factors and conditions together with technical reports on the selected datasets and frameworks.
- A brief description of various metrics involved in the evaluation of 3D object detection frameworks.
- As a whole, a comparative study on the performance of 3D object detection frameworks based on LiDAR data and sensor fusion techniques by performing different experiments along with theoretical knowledge.

This report is further structured as follows. Chapter 2 describes various state-of-the-art 3D object detection frameworks based on type of input. Chapter 3 highlights various sensors, selection criteria, and sensor fusion techniques. Chapter 4 describes various multi-modal datasets, brief summary on 3D detection frameworks, and their selection criteria. Chapter 5 describes technical details on the selected datasets and frameworks. Chapter 6 provides information on various experiments and their qualitative and quantitative results. Finally, with a brief summary of comparative analysis and research challenges, the report is concluded.

2

State-of-the-Art

The outbreak of research in the field of object detection using deep convolutional neural network started when AlexNet[29] created a record by winning the ImageNet Large Scale Visual Recognition Challenge[11] in the year 2012[66]. Generally, objects are detected and classified based on the intersection over union (IOU) of the ground truth data and the estimated data, and this estimation usually is based on the classification probability and bounding box regressions.

State-of-the-art deep learning frameworks generally follow either of the two approaches, namely, two-stage object detection and single-stage object detection. In two-stage object detection, the region of interest or region proposals are extracted in the first stage and then the objects are classified and in single-stage object detection, the framework directly maps the features into bounding boxes and classifies the objects. These two-stage detectors require high inference time and very complex training, whereas single-stage detectors are easier to optimize and are faster, but they compromise on accuracy[16].

Researches have been proposed and performed in the field of object detection and classification based on the data from various sensing modalities and reported on the strengths and weaknesses of different approaches for the problem. In this chapter, we discuss and summarize various deep learning approaches providing solutions to the object detection problem using single modalities as well as sensor fusion techniques.

2.1 Image Based Frameworks

State-of-the-art 2D object detection frameworks that are based on camera images have been proven for their performance. These 2D object detectors mainly detect objects in the image plane. You Only Look Once YOLO[50] is one of the first single-stage detectors which uses the deep CNN for creating the feature maps and a fully-connected neural network for bounding box predictions and multi-class probabilities. This design makes it faster, low memory consumption, and reduces computation cost[78]. Another faster approach is Single Shot multi-box Detector (SSD)[40], these use high-resolution feature maps from the early stages of deep CNN for improving the performance on detection localization.

Even though the computational cost and the performance of single-stage detectors in real-time is better, the performance of the two-stage detectors using the region proposal networks (RPN) is unmatched in terms of accuracy for the object recognition and localization tasks and also in recent years, the computational cost of these detectors are also improved[78]. R-CNN[21] is one of the initial two-stage detectors whose

pipeline consisted of RPN and deep learning techniques for classification. The pipeline of R-CNN is expressed in three parts namely, proposal generation, feature extraction, and region classification. Initially, the pipeline generates many proposals for the input image and based on selective search the proposals are selected and a deep CNN is used for converting them to feature maps, and then a classifier is used for classification[21].

Although the accuracy and performance of 2D object detectors are good, the detection of the objects in the image plane is not sufficient for autonomous driving conditions. For more accurate driving tasks, the 3D localization of objects and objects size estimation is necessary. To obtain these parameters, researchers have proposed a few methods that involve the estimation of 3D bounding boxes using deep neural networks[3]. An RPN based method named Mono3D[8] which utilizes various parameters like location priors, semantics, context, and hand-engineered features were proposed. Here the features are computed by performing searches in 3D space and are filtered using Non-Maximum Suppression (NMS), and these proposals are given to Fast R-CNN[20] for 3D bounding box regression[8].

A method called 3D Voxel Pattern (3DVP)[71] alleviate the effect of occlusion based on object reasoning. In this method, authors represent the input as a set of RGB intensities, a set of voxels for 3D shape, and occlusion mask. This representation improvises the recovery of objects that are occluded or truncated. The clustering of the patterns in the data is performed to obtain the 3D localization and pose estimation[71]. This method is highly dependent on the performance of RPN, to overcome this, another method called SubCNN[72] was proposed by the same authors which utilizes CNN for accessing the class information at the RPN level and generates region of interests (ROI) using which 3D estimations are performed[72].

Most of the methods that perform 3D estimations on monocular images by performing either exhaustive search on 3D space, by clustering the patterns in data, or 3D templates use only images from the front-facing camera which is not sufficient for accurate driving of the vehicle. To overcome this, a method named 360Panoramic[10] was proposed, this uses panoramic images as input and estimates dense depth maps and performs standard object detection methods. Due to a lack of availability of a panoramic image dataset, they provide benchmark results on synthetic dataset[10].

To overcome the expensiveness property of LiDAR, an approach named Pseudo-LiDAR[63] has been proposed recently. This method is based on depth estimation using the stereo images. Here the depth estimations are performed using the stereo disparity estimation algorithms and a depth map is generated. From this depth map, the 3D location of each pixel is calculated and all points are projected in to 3D coordinates resulting in a dense 3D map which is similar to the LiDAR dense map. On this pseudo-LiDAR map, 3D detection methods are applied to obtain the detections[63]. Due to low accurate depth estimation for faraway objects by Pseudo-LiDAR[63], an alternative approach named Pseudo-LiDAR++[77] was proposed. This method uses a Stereo Depth Network (SDN) and an alternative cost function called Depth Cost Volume (DCV) that significantly improves the estimation of depth. All these depth estimation still depends on the initial depth estimations from the 2D stereo images and thus it is difficult to match the accuracy and reliability of the LiDAR sensor[77].

A major drawback of the methods performing 3D estimation on monocular images is the lack of availability of depth information which limits the detection and localization of the objects especially

when the objects are far or occluded[3] and since monocular cameras are sensitive to changes in lighting and weather conditions the performance will be compromised. The above-mentioned object detection frameworks based on cameras, are summarized in Table 2.1.

Framework	Method	Limitation
YOLO [50]	2D detector, single-stage detector, uses deep CNN for feature maps and fully connected network for classification and regression	No depth information
SSD [40]	2D detector, high-resolution feature maps from early stages of deep CNN	No depth information
Mono3D [8]	3D detector, performs search in 3D space, uses NMS for feature selection, Fast R-CNN for 3D regression	No prior depth information, poor localization of objects
3DVP [71]	3D detector, voxel based representation, RPN for proposal generation.	Highly dependent on RPN proposal generation
360Panoramic [10]	3D detector, adapts CNN for 3D detection, depth estimation on monocular images	Built on synthetic data, objects near to camera are not detected
Pseudo-LiDAR [63]	3D detector, depth estimation using stereo disparity estimation and 3D depth maps are created.	Low accuracy on long range objects
Pseudo-LiDAR++ [77]	3D detector, uses Stereo Depth Network for depth estimation.	3D depth estimated using 2D depth from stereo images.

Table 2.1: Summary of image based object detection frameworks.

2.2 LiDAR Based Frameworks

To overcome the lack of availability of direct depth measurements from cameras, an alternative sensor named LiDAR is used for 3D perception. Various methods have been proposed for solving the 3D object detection problems based on the 3D data from LiDAR sensors. This 3D data is often referred to as point cloud data. Based on the different approaches of processing this data and utilizing them for object detection, the 3D detection methods can be sub-categorized as projection-based, volumetric representations, and point-nets[3].

Projection-based detector mainly works by projecting the 3D point cloud on to a 2D image plane. Projection on to a 2D image plane can be performed by projecting on three different 2D shapes, namely, a plane[58], a cylindrical surface[34], or a spherical surface[70]. After projecting on to these surfaces a state-of-the-art 2D detector is utilized for object detection and by performing the dimension and position regression the 3D bounding boxes are recovered[3]. A method that uses a fully convolutional network

(FCN) and creates a 2D cylindrical map by projecting the point cloud data for prediction of 3D bounding boxes was proposed by Bo Li et al[34]. In this approach, along with the 2D map, the height and distance of the point were also encoded and provided as input to 2D FCN which performs the convolutional operations and provides the bounding box predictions point-wise. A non-maximum suppression (NMS) strategy is utilized to filter out the FCN proposals and provide the predictions[34].

While a few methods used these traditional projection techniques, a few research works such as Complex-YOLO[55] and BirdNet[4] used a different type of projection named Bird-Eye View (BEV) for 3D proposal generation. Complex-YOLO[55] method represents the point cloud by encoding it into a 2D cell based on the maximum and minimum height of the point and then uses a Faster R-CNN[51] framework as a backbone network and modifies the refinement stage to provide the 3D bounding box estimations[55]. While the BirdNet[4] method represents the point cloud data using the height, intensity, and density channels, and the location and orientation of the objects are estimated using convolutional neural networks and 3D estimations are obtained by post-processing techniques[4].

To realize the generalization capability of these methods based on the point cloud density, an approach for normalizing the density channel was proposed by BeltrÃąn et al[4] and this normalization is performed based on the LiDAR's parameters. This approach provides a uniform representation of the input point cloud resulting in improving the generalization capabilities. This forms one of the requirements for safety-critical systems of autonomous vehicle[3].

Although these models use the softmax normalization for providing the prediction score, the sum of probabilities due to the softmax normalization will be one, this might not provide the prediction confidence[3]. To overcome this Feng et al[15] proposed a method based on the probabilistic approach which represents two kinds of uncertainties namely epistemic and aleatoric in the classification task. Aleatoric uncertainty is also known as statistical uncertainty and their presence is unknown and epistemic uncertainty is also known as systematic uncertainty[67]. This systematic uncertainty is related to the accuracy of detection and statistical uncertainty is related to the object's distance and occlusion[15].

Due to the loss of information during the projection of 3D point cloud on to a 2D image plane, the detection estimation of the methods based on projection is not reliable and also there will be no spatial information encoding explicitly.

The second way of representing the point cloud inputs is using the volumetric representations, the methods which use this kind of representation generally assumes that the object or the scene is described in a 3D grid, or using voxels, where each of the units will be associated with an attribute. Due to this representation, the methods will encode the information on the shape of the object explicitly[3]. One of the downsides of this representation is that if objects are less in a scene, most of the volumes will be empty and the efficiency of the methods will be reduced while processing these empty volumes.

A few methods were proposed for solving the object detection problem using volumetric representations. A method named 3DFCN[33] which uses a binary volumetric representation of the point cloud data and estimates the 3D detections for objects vehicle only[33]. This method works similar to [34] and uses 3D convolutions for estimations. To improve the performance and extend to other object classes an approach called Vote3Deep[12] was proposed, where the bounding box sizes for each class are fixed and the network

is directly trained on these 3D fragments of data to increase the performance[12].

Generally, the total number of sparsely distributed 3D points in a point cloud will be fluctuating. The conventional pipelines of deep neural networks assume that the input will be of fixed size, to avoid this downside, and to reduce the loss of information in point cloud by projecting techniques the third method called point-nets are used for the representation of point clouds. The first conclusive results using this representation method were produced by an approach called VoxelNet[79]. VoxelNet works directly on raw point clouds, it generates voxel-wise features based on the subsets of the raw point cloud. Here a voxel feature encoder (VFE) network layer is used to convert the fluctuating number of points in each voxel into a fixed-size feature vector and this grid of feature vectors are provided as input to the region proposal network for object detection[79].

The above method uses 3D dense convolutions for its operations and its computational cost increases making it difficult for real-time use cases. To address this issue, a method called SECOND[75] which utilizes sparse convolutions for 3D detection was proposed. Here sparse convolution networks are used for extracting the information from LiDAR's z-axis before down-sampling it and also introduces GPU based algorithms for sparse convolution for increasing the speed of training and inference. The output of the sparse convolutions is provided as input to RPN for 3D detections and estimations[75]. Due to the utilization of GPU and sparse convolutions, the computational cost is reduced and can inference in real-time.

Another approach named IPOD[76] was proposed which works on raw point clouds without making any approximations to it. Here first semantic segmentation is performed on the images and segmentation results are then projected on to raw point clouds to extract all positive points and finally the proposals are generated. Using the NMS approach the proposals are reduced and using the IOU regressions and predefined anchor sizes the predictions are extracted[76]. Due to the involvement of many computations, this method is not suitable for real-time scenarios.

An alternative method called PointRCNN[54] was proposed for object detection on a raw point cloud. This approach consists of two stages, the first stage is used for the generation of 3D proposals using a bottom-up approach, and the second stage is used for refining the generated proposals in canonical coordinates and obtain the final predictions. Stage-1 generates a high-quality 3D proposal by segmenting the complete point cloud into the foreground and background points, stage-2 transforms each proposal to canonical coordinates and learns spatial features and to match with the global features for accurate bounding box regressions[54]. Due to this process, the performance of this framework increases but the inference time is more than that of SECOND[75].

A method named PointPillars[31] a fast encoder approach to encode the raw point cloud into vertical columns called pillars was proposed and these encoded features can be used with any 2D convolutional detection frameworks. This method first encodes the raw point cloud into pseudo images by using a Pillar Feature Encoder Network and uses a 2D convolution backbone network for representing high-level features and perform detection using a detection head similar to single-shot detectors. Because of the encoding technique, this method outperforms all the other LiDAR-based techniques in all aspects[31].

Among all the ways of representing the point cloud, projection-based techniques are popular since it

can be used with traditional image-based object detectors, but due to loss of information due to projecting, the point-net based methods which use raw point cloud is preferred. Although there is still a need for the investigation of new techniques for utilizing complete scene as input[3]. The aforesaid 3D object detection frameworks that are based on LiDAR data are summarized in Table 2.2.

Framework	**Method**	**Limitation**
VeloFCN [34]	Uses FCN on BEV projection, the output is 3D bounding box regression and likelihood of an object present there	Detects large objects like vehicles, not good at detecting smaller objects
Complex-YOLO [55]	Encodes point cloud in 2D cell, it is an extension of single-shot detector for 3D detection and regression	Not accurate due to the use of second refinement stage
BirdNet [4]	Different generalization and normalization technique are used to accommodate various LiDAR models	Accepts only 3 channel inputs, and no information on intensity and density is used.
3DFCN [33]	Represents point clouds using binary volumetric representations	Detects large objects like vehicles, not good at detecting smaller objects
Vote3Deep [12]	Provides convolutional algorithms for exploiting sparse voxel features	Assumes fixed size of bounding box for all objects, limiting its performance
VoxelNet [79]	Works directly on the raw point cloud, convert it into voxel feature vectors using VFE and uses RPN for detection tasks.	3D dense convolutions are involved, computation complexity increases and not suitable for real-time operation
SECOND [75]	Uses sparse convolution operation, introduces GPU based algorithm making it faster	Deficiency of LiDAR sensors are not addressed, Low performance on smaller objects like pedestrians
IPOD [76]	First performs segmentation on images, using this result to generate proposals using RPN for object detection	Not efficient due to the involvement of various complex computations
PointRCNN [54]	Involves two operation stages, one for 3D proposal generations and other for refinement.	Inference time is more than SECOND
PointPillars [31]	Fast encoder approach to encode point cloud into pillars, uses 2D convolution operations	Low detection accuracy on smaller objects like pedestrians when above 48 meters range

Table 2.2: Summary of LiDAR based object detection frameworks.

2.3 Sensor Fusion Based Frameworks

Discrimination of various classes of objects is necessary for the task of object detection and classification. The texture of the object is an important factor for class discrimination and it is not provided by LiDAR's point cloud data and it is provided by a monocular camera. On the contrary, a monocular camera does not provide depth information which is a major requirement for size and location estimation of objects. Similarly, the camera can aid in the detection of the objects that are far from ego vehicle but as the distance increases the sparsity of the LiDAR points increases and reduces detection capabilities. To utilize and exploit these complementary properties of these sensing modalities many methods were proposed for fusing the data from different sensors and perform object detection.

In general, there are three schemes for fusing the data from different modalities, viz, early fusion, late fusion, and deep fusion[9]. In early fusion, the data from different modalities are fused at the beginning of the process and generates a new representation that depends on all modalities. In late fusion, data is processed separately until the last stage of the process and then fused. In deep fusion, the data is fused hierarchically in various layers of neural network[9]. Further information on these approaches are provided in Chapter 3.

Instead of fusing the data from camera and LiDAR directly, there are few approaches that perform a fusion of RGB images along with depth images and optical flow to perform object detection in 2D space. One such framework was proposed by Gonzalez et al[22] to perform pedestrian detection in 2D space. Instead of using the LiDAR data directly, first, they convert into high-density depth images and then fuse with RGB images to extract multi-view and multi-class objects and for each view, a random forest classifier is trained and ensembles methods applied for object detection[22]. Another approach was proposed by Enzweiler et al[13] where they perform early fusion for RGB images along with depth maps to perform 2D detections. In this approach, the depth maps are obtained from stereo cameras, and they use multi-layer perceptrons and linear SVMs for detection and classification of objects[13]. The main focus of these two approaches is on the detection of pedestrians.

Initial promising results in performing 3D object detection by fusing multi-modal data and multiple views was first presented by the architecture named Multi-View 3D Object Detection (MV3D)[9], where the authors make use of RGB images along with projection-based approaches for representing the LiDAR data for detection and classification of objects. Here LiDAR data is projected on to the BEV space and using RPN the 3D proposals are generated. These proposals are then projected on to the feature maps from multiple views. Using ROI, features related to each view are extracted. Using deep fusion technique the region-wise proposals are fused and the final layer provides the classification probabilities and 3D regressions are performed[9]. The main drawback of this approach is due to the RPN proposals generated on the sparse LiDAR data, even though the objects are visible in the camera frame, they were not detected.

To overcome the drawback of MV3D architecture, a framework called Aggregate View Object Detection (AVOD)[30] was proposed. In this approach, instead of using multi-view data representation channels, only RGB images and LiDAR data represented using a bird's eye view were used. RPN was used to extract the region proposals on both images and bird's eye view data resulting in generating high recall

proposals. These proposals were then fused by performing early fusion techniques. A feature pyramid network (FPN) extensions were implemented to upsample the feature maps resulting in the detection of smaller objects as well[30]. The drawback of this approach was that the model was able to detect objects only in front-view due to the use of front-view images.

Another approach named Deep Continuous Fusion[36] that utilizes the continuous fusion technique for the fusion of a bird's eye view representation of LiDAR data and images was proposed. In this approach, authors learn to project the images on the BEV space and utilizes the continuous convolution layers for fusing the images with LiDAR feature maps[36]. Another similar approach named LaserNet++[43] where the images were fused with the range view feature maps of LiDAR data to exploit the detection and classification of small objects using their dense feature maps.

An alternative approach that fuses the multi-modal data on multi-level feature extraction was proposed by Liang et al[37] performs multi-task multi-sensor fusion for 3D object detection. This approach is a two-stage fusion architecture. In the first stage, sparse depth image and RGB image are concatenated, and point-wise features are extracted, and using point-wise feature fusion technique, the point-wise features from images are fused with LiDAR BEV point-wise features. In the second stage, ROI feature fusion is performed and 2D and 3D regression are performed[37].

An alternative strategy that exploits two PointNet architecture for classification and detection of objects in 3D space named Frustum PointNet[47] was proposed, which extrapolates the 2D detections performed on the monocular images onto the 3D space where the point cloud data is fused. In this approach, the region proposals are extracted from the monocular images and then they are extrapolated onto 3D space, this results in 3D frustum region proposals. Using these frustum proposals the 3D point cloud data is trimmed and the search space is reduced. First PointNet is used to remove the background clutters in the point cloud that does not enclose in any of the frustum proposals and second PointNet is then applied for classification and regression[47].

The drawback of this approach is that since it utilizes the extrapolated frustums, most of the points in point cloud might not be included and resulting in data loss. To overcome this drawback, an approach named Frustum ConvNet[64] which exploits the voxel-level features was proposed. Similar to the Frustum PointNet[47] approach, here the region proposals from 2D images are extrapolated into 3D space. Based on these proposals the point clouds are grouped and aggregated them as frustum level feature vectors and are treated similar to voxel-level feature vectors. Using FCN layers to fuse the frustum level features by performing up-sampling and down-sampling across the frustum axis resulting in higher frustum resolution feature maps. Using a detection head on these feature maps, 3D detection and regression is performed[64].

Various sensor fusion techniques and different approaches discussed above exploit the complementary information from various sensing modalities resulting in improving the accuracy of 3D object detection and bounding box regression task. The aforementioned 3D object detection frameworks are summarized in Table 2.3.

Framework	Method	Limitation
On-Board Object Detection [22]	Random forest classifier for multi-view, ensemble method for object detection	Object detection in 2D space, specific for pedestrian detection
Mixture-of-Experts [13]	Early fusion of RGB images with depth maps from stereo camera, MLP and LinearSVM for object detection and classification	Object detection in 2D space, specific for pedestrian detection
MV3D [9]	Performs object detection on multi-view, utilizes deep fusion technique for data fusion	Far objects are not detected due to the use of RPN on BEV, low performance on smaller objects
AVOD [30]	RPN is used on both input modalities, utilizes early fusion technique for data fusion	Detects objects only in front view
ContFuse [36]	Performs continuous fusion of multi-modal data, projects images onto BEV space for fusion	Images are projected onto BEV, the performance on small object detections is comparatively low
Multi-task Multi-sensor Fusion [37]	Multiple fusion techniques involved. Point-wise fusion and ROI fusion is performed for both 2D and 3D regression	Results on small object detection are not provided for comparison
F-PointNet [47]	Utilizes preobtained 2D detections, uses two PointNet layers one for point cloud cleaning and other for detection and classification	Inference time is more, depends on 2D detection results.
F-ConvNet [64]	Treats the extrapolated region proposals from 2D detections as voxel feature vectors and uses RPN for 3D regression	Depends on 2D detection results for extrapolating the voxel feature vectors

Table 2.3: Summary of sensor fusion based object detection frameworks.

From the discussions presented in this chapter, it is inferred that due to the lack of availability of the depth information, and sensitivity to the lighting conditions, the image-based 3D object detection frameworks are not reliable for self-driving vehicles. On the other hand, LiDAR sensors are costlier compared to monocular cameras and provide direct depth information which helps in the estimation of size regressions and localization of the objects, which in turn results in better performance for the 3D object detection. On the contrary, LiDAR sensors do not provide fine textural information, and the sparsity increases as the range increases which affects the detection performance. To utilize the complementary properties of these sensors, sensor fusion based frameworks are proposed and these frameworks take the advantage of fusing the complementary information and provide better performance in real-time scenarios making it reliable for self-driving cars.

More information on the various characteristics of the sensors that are involved in perceiving the vehicle's environment, along with different criteria based on which the sensors are chosen, and various approaches of sensor fusion for utilizing the complementary features of the sensors are discussed in the upcoming chapter (Chapter 3).

3

Sensor Fusion

3.1 Sensors

Sensors are the major and primary source of information for interpretation of the environment for an autonomous vehicle. These vehicles are equipped with various automotive graded sensors which include both exteroceptive and proprioceptive sensors. In this chapter, we provide more information on various sensors used for perception in autonomous vehicles and the criteria for selecting these sensors.

3.1.1 Sensing Modalities

A. Camera:

Images captured from cameras provide comprehensive information in their field of view. Different cameras namely monocular camera, stereo camera, thermal camera, time-of-flight (TOF) camera, and so on are used in autonomous vehicles. The information from these images is in the form of pixel intensities which provides the texture and shape of the object[3]. This texture and shape information is necessary for classifying the road signs, various classes of objects, and lane markings.

A disadvantage of using a monocular camera is that it lacks in providing the depth information which is necessary for estimation of the size of the object and its position accurately. But this can be overcome by using the stereo camera setup, where two identical cameras are placed at a distance apart and their optical axis is aligned. The depth information is calculated by applying matching algorithms to determine the concurrences in the images from both cameras. But this setup requires more processing time and energy.

To overcome the consumption of processing time, the TOF cameras are used, which works on the method by calculating the delay in the time of flight of the modulated infrared pulses to infer the depth information, but the downside of this camera is that the resolution is low when compared to the stereo cameras[25].

The weakness of cameras is that they are sensitive to lighting conditions and weather conditions. To suppress the effect of lighting conditions, infrared cameras can be used, these are robust to changes in the amount of light like dawn and dusk. But these cameras do not provide the depth information directly[3].

B. LiDAR:

LiDAR sensing generates high-resolution data by emitting and receiving thousands of laser beams. These laser beams hit the objects and bounces back to the receiver in the sensor. The time of flight of these laser beams is used to determine the distance of the object from the source. Based on the intensity of the reflected beams, the texture of the objects can be obtained. Each of these laser beams is in the infrared spectrum range and are emitted in many different angles covering 360° generating highly accurate models of the vehicle's surrounding in 3D.

Generally, the LiDAR data is stored in the format known as point cloud data (PCD). The samples of the point cloud are not distributed uniformly as compared to the pixels of images. As these are active sensors, these do not require any external illumination factors, so these sensors are more reliable during night time as well as in case of sunny weather conditions.

The main problem with LiDAR is that they generate a very large amount of data in less time, for example considering a Velodyne HDL-64 LiDAR sensor, this sensor uses an array of rotating beams to generate 3D point clouds covering 360° and up to a distance of 120 meter radius. Per frame, this produces around 120,000 points resulting in almost 1,200 million points per second at a rate of 10 Hz per frame making this difficult to process in real-time[3].

One tricky behavior of the LiDAR sensor is that, if two sensors with the same wavelength are used they tend to generate fake echoes of the objects resulting in the detection of objects nearer or further than the actual position. As the range of the laser beams increases, the sparsity of the data increases and does not provide the fine texture of the object.

C. Radar:

Radar work on the concept of Doppler effect, by measuring the runtime of the radio signal, the radial velocity of the object is determined. The distance between the object and the ego vehicle can also be measured by utilizing the total time taken by the radio wave to hit the object and reflect back to the receiver. Radar sensors are mainly used in the scenario of blind spot detections, automated parking, and determining the distance of the objects with respect to ego vehicle. Due to the high penetrable nature of radio waves, they offer good performance under varying weather conditions and also provide more accurate readings for detection of objects in short-range[53]. Another advantage is that the installation of the Radar equipment does not need additional space or complex constraints, they are easy to mount on vehicles[53]. Since Radar uses shorter wavelengths, the smaller objects will not be detected and due to low resolution, the fine texture of the objects will not be generated and it is very challenging to use Radar for classification of the objects. Another drawback is the lack of precision, limited field of view, and chances of generating false positives due to the echo in received signal[53].

D. Ultrasonic sensors:

This sensor uses high-frequency sound waves in order to determine the object's distance. These are mainly used in the detection of objects that are near to the ego vehicle and also in scenarios of low-speed driving

viz automated parking. These sensors are very sensitive and the accuracy can be easily affected by a change in temperature, dust in the air, and also these waves are absorbed by soft materials such as fabrics. These sensors are compact and can be relied upon to measure the distance in various weather conditions like rain, snow, or fog and at night time as well.

3.1.2 Sensor Selection Criteria

Here we discuss various criteria on selecting the sensors described in the previous section.

A. Range:

The range of a sensor is defined as the maximum and minimum values it can measure of a certain applied parameters. LiDAR and Radar sensors come in various sensor ranges say, short-range, mid-range, and long-range. These sensors detect objects that are at a distance of a few meters up to more than 200 meters. Few LiDAR systems have difficulties in detecting objects that are too close to the vehicle but Radars can detect objects that are less than a meter away from the vehicle. On the other hand, monocular cameras do not provide distance information but stereo cameras can be trusted to provide distance information of up to 80 meters approximately.

B. Spatial Resolution:

On the account of the laser beams of the short-wavelength emitted by LiDAR, the scans will have a spatial resolution of around 0.1°. This results in a high-resolution 3D scan of the vehicle's environment and resulting in the precise classification of objects. On the other hand, as distance increases the Radars will not be able to resolve the smaller objects. For cameras, spatial resolution is defined by the image pixel size and its signal-to-noise ratio.

C. Robustness:

Since both Radar and LiDAR are active sensors, they are robust to darkness and even though the performance of LiDAR in the daytime is good, it is even better at night time due to the absence of ambient sunlight which affects the reception of laser beams. Radars have excellent robustness for the adverse weather conditions, they are not affected by rain, snow, fog, or any other such disturbances. Since cameras are passive sensors, they are not relied on during night time and LiDAR and cameras are easily affected by adverse weather, as the adversity increases the performance decreases.

D. Object Classification:

Cameras are an excellent source for performing the multi-class classification of objects because of the textural information from images. In contrast, LiDAR provides high-density 3D scans that also provide the texture of objects and helps in the classification of various objects. On the contrary, Radars are not that good at the classification of objects.

E. Measure Speed:

Using the Doppler frequency shift in the radio waves, Radars can directly determine the speed of the object. In LiDAR, by tracking the same object and measuring the distance successively the distance can be calculated but it is not reliable, and cameras are used to measure the distance to collision based on the displacement of the objects in the image plane.

F. Cost:

Due to the wide usage of Radar and camera systems in the automotive industry, these are cost-effective and LiDAR sensors are comparatively costlier to other sensors. Due to hardware and computational expenses, stereo cameras are also expensive.

G. Size and Computational Requirements:

Radars and monocular cameras are of smaller size and can be integrated into the vehicle without much space required but stereo cameras and LiDARs are bulkier and require special requirements for integrating them into vehicles. The cost of preprocessing the data from LiDAR and Radar is typically high when compared to preprocessing the images.

The summary with respect to the sensors and their characteristics discussed in this chapter are summarized in Table 3.1.

Sensors	**Sensitive to light**	**Sensitive to Weather**	**Depth**	**Measure Speed**	**Cost**	**Range**	**Color**	**Object Classification**
LiDAR	x	✓	✓	indirectly	high	long mid short	x	✓
Radar	x	x	✓	directly	medium	long mid short	x	x
Camera	✓	✓	x	indirectly	low	x	✓	✓
Stereo Camera	✓	✓	✓	indirectly	high	mid (< 100 m)	✓	✓
Thermal Camera	x	✓	x	indirectly	medium	x	x	✓
Ultrasonic	x	x	✓	indirectly	low	short	x	x

Table 3.1: Summary on sensors involved in perceiving the vehicle's environment and their characteristics.

3.2 Sensor Fusion Techniques

Sensor fusion is basically the integration of data from multiple sources in order to provide or generate a specific and comprehensive unified representation of the data[24]. Sensor data fusion helps in improvisation of measurements from more than one source of data than individual sources of data. Sensor fusion helps in handling of the redundant data and increases system accuracy along with improving the integrity of

the system[53]. Sensor fusion can be adapted in many ways depending on the time of fusion, and sensor data can be integrated on various levels say, data level, feature level, and decision level[24].

Here data-level fusion refers to integrating the sensor data directly, feature level fusion refers to combining the representative features from multi-sensor data, and decision level fusion refers to fuse the decision from each sensor modal[17]. Sensor fusion is a multidisciplinary task and based on its application, data sources, and other criteria it can be classified into many categories, and detailed information on this is provided in the literature [7]. In this report, sensor fusion techniques for the application of the perception system are divided into three categories, (1) Probabilistic approaches, (2) Gaussian approaches, and (3) Deep learning approaches.

3.2.1 Probabilistic Approaches

Probabilistic approaches generally model the uncertainties using the probability theory. Here we briefly discuss the widely used Bayesian filters (Kalman filter) for sensor fusion. Given the prior information related to the system and the measuring devices, along with all the measurement data from the sensors, Kalman filters[27] provides an optimal approximation of the system state. Because of the filtering capability and modeling the uncertainties, Kalman filters are widely used in many applications especially in autonomous navigation systems for accurate localization of the system, and tracking of the dynamically moving objects.

The Kalman filter basically consists of two models, one is the process model and the other is the measurement model. The process model by assuming the future state as a linear function of the present state, describes the transition of the process state and it is defined in the below equation[17].

$$x_t = Ax_{t-1} + Bu_{t-1} + w_{t-1} \tag{3.1}$$

where,

x_t - System state at time t

A - State transition matrix

B - Relates to optional control input u_t to state

w_t - process noise

The measurement model provides the measurement at time step t, by assuming the observations as a linear function of the state and is defined in below equation[17].

$$z_t = Hx_t + v_t \tag{3.2}$$

where,

z_t - Measurement at time t

x_t - System state at time t

H - Relates state to measurement

v_t - Measurement noise

Here w_t and v_t are independent Gaussian white noises. Along with the estimation of the system states, Kalman filters also provide the posterior estimation of error covariance and is defined in below equation[17].

$$P_t = (I - K_t H)P_t^- \tag{3.3}$$

where,

P_t - Posteriori estimate of error covariance

P_t^- - Priori estimate of error covariance

K_t - gain matrix

Unfortunately, the system states and their transitions are dynamic in nature and are rarely linear, to solve this nonlinearity we use the extensions of Kalman filters such as Extended Kalman filters (EKF) and Unscented Kalman filters (UKF). EKF and UKF both linearize the non-linear states and performs the state estimations. EKF uses Taylor series expansion techniques for linearizing the non-linear system states and UKF performs stochastic linearization using a weighted statistical regression process[17]. But the computational complexity of both these approaches is high.

3.2.2 Gaussian Approaches

Gaussian processes are referred to as stochastic processes where any random subset of a random variable is jointly Gaussian distributed[60]. These are basically non-parametric Bayesian models and have continuous representation that helps in modeling the uncertain data and spatially correlated data. Gaussian processes are mainly characterized by a mean function and a covariance function or also known as a kernel that helps in modeling the distributions. Given the dataset, these kernel functions model how closely the data is related.

The main idea of the Gaussian process is to introduce the prior information directly in function space and it can be defined as a set of random variables with a finite number of data points are Gaussian distributed[17]. The introduction of this prior to the function space helps in representing the prior beliefs. Basically, combining this prior information along with the data results in posterior estimations.

Gaussian processes are also highly used in machine learning applications for classification and regression tasks along with dimensionality reduction tasks. Since these approaches are non-parametric models, that help in direct combination of functions or their derivatives[17]. Initially, an approach for fusing multi-modal data was proposed by Pearlmutter et al[44], which performed a generalized transformation on the Gaussian process priors by utilizing the linear transformation, along with the use of transformed Gaussian priors to estimate the derivatives of the noisy measurements and fusing the data[44]. A detailed report on various approaches, their performance along with advantages and disadvantages are provided in a literature review published by Vasudevan[60].

3.2.3 Deep Learning Approaches

For fusing the sensor data with deep learning approach, generally, we need to have firm answers for the following three questions, (1) What to fuse - tells about what data to fuse, (2) How to fuse - how to fuse the data and what operations to be implemented for fusing the data, and (3) When to fuse - this is similar to the three-stage of data fusion proposed by Hall et al[24]. In this section, we will provide a brief summary of the various approaches by answering the above three questions.

What to Fuse:

Due to the lack of availability of fine textural information and lack of precision of the Radar sensor, there is not much research published on fusing its data with other main perception sensors such as LiDAR and camera. Here we will mainly see various ways of representing, and preprocessing of the data from these two sensors.

Considering LiDAR point cloud data, there are three main ways for preprocessing the data, (1) Discretization of the 3D data into 3D voxels and allocating points to those voxels, (2) Directly learning features on continuous vector space, and (3) Projection methods as discussed in Section 2.2. In the first approach, the continuous point cloud data is discretized into 3D voxel representation which helps in preserving the 3D shape of the data, but due to the sparsity of the point cloud most of the voxels remain empty and processing these is time-consuming and increases complexity. To overcome this, a framework named VoxelNet[79] introduced a voxel feature encoding (VFE) layer for processing the point cloud data efficiently for 3D detections.

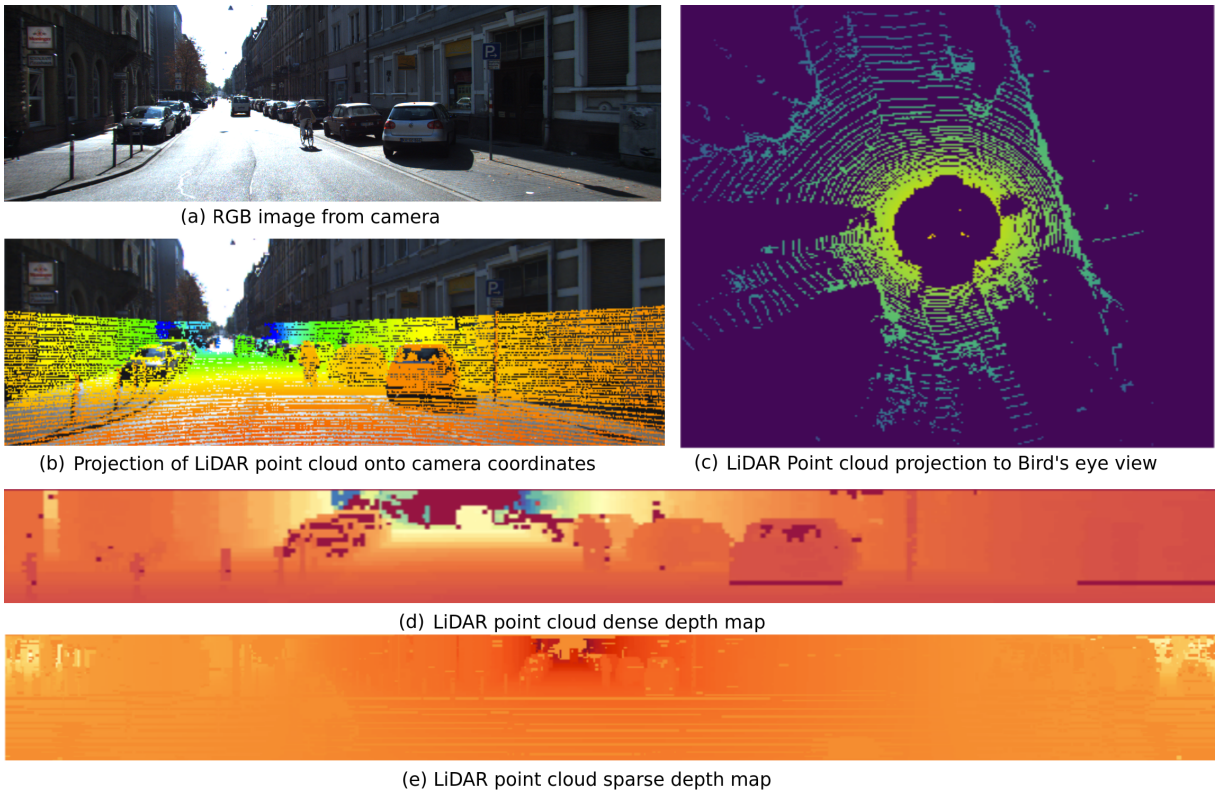
In the second approach, PointNet architecture was utilized to learn the features directly from point cloud data and perform 3D object detections and later this architecture was improvised and adapted by Charles et al[47] for fusing with RGB images and perform 3D detections and classifications. A new convolution operator named Parametric Continuous Convolution[62] that aggregates the points based on weighted sum and another approach where the χ transformations[35] are learned before applying the point cloud features to CNN was proposed.

In projection-based approaches, 3D point clouds are projected onto 2D grid-based feature maps so as to take advantage of the computational efficiency of 2D convolutions. There are three major 2D grid-based features maps namely, spherical maps, bird's eye view (BEV), and camera-plane maps (CPM). Spherical maps are obtained by projecting the point cloud onto a sphere that is characterized by azimuth and zenith angles[16]. This helps in the dense representation of point clouds. CPM is obtained by projecting point cloud onto camera coordinates by utilizing camera calibrations. Bird's eye view preserves the actual information on the length and width of the object and also helps in avoiding occlusion. This method of projection provides the position of an object with respect to ground plane in turn making it easier for localization tasks[16].

Unlike point cloud data, images from the camera provide fine textural information of objects and the scene. But there is a high probability of object occlusions and variations in the scale of the object in 2D space. To overcome these issues, an approach named Orthographic Feature Transformation[52] was

proposed to project the camera features onto the BEV projections of point cloud data. Similarly, a few other approaches have been discussed in Section 2.3.

We can summarize the different approaches on processing the point cloud data and camera images into three approaches, (1) fusion of feature maps extracted from 2D convolutions separately on point cloud data and RGB images, (2) fusion of feature maps obtained from point cloud data using 3D RPN and camera images, and (3) projecting camera images on to BEV representation of point cloud data or projecting LiDAR data on camera coordinates and perform object detection. Various projection approaches, and depth maps of LiDAR can be seen in Figure 3.1.



(a) RGB image from camera

(b) Projection of LiDAR point cloud onto camera coordinates

(c) LiDAR Point cloud projection to Bird's eye view

(d) LiDAR point cloud dense depth map

(e) LiDAR point cloud sparse depth map

Figure 3.1: RGB image and various projection approaches for LiDAR representations in 2D. Each image representing different projection methods.

How to Fuse:

In deep learning models, there are few approaches to fusing or combining the data from multiple modalities. Since we are mainly concentrated on the LiDAR and camera sensors, we can represent these modalities as M_l and M_c and their feature maps from the t_{th} layer of neural network is represented as $f_t^{M_l}$ and $f_t^{M_c}$. We will denote the transformation function applied for these feature maps as $Tr_t(\cdot)$ [16]. The summarization of various approaches for performing data fusion are enumerated below.

1. *Addition*: Performs element-wise join operation.

$$f_t = Tr_{t-1}(f_{t-1}^{M_l} + f_{t-1}^{M_c}) \tag{3.4}$$

2. *Average Mean*: Performs element-wise mean operation.

$$f_t = Tr_{t-1}(0.5 * f_{t-1}^{M_l} + 0.5 * f_{t-1}^{M_c}) \tag{3.5}$$

3. *Concatenation*: Concatenates the feature maps as shown in Equation 3.6. Usually for CNNs, these are flattened and then concatenated along the rows of feature maps[16].

$$f_t = Tr_{t-1}((f_{t-1}^{M_l})^\frown(f_{t-1}^{M_c})) \tag{3.6}$$

4. *Ensembles*: Performs ensemble operations as shown in Equation 3.7 and fuse the feature maps. This approach is mainly used for ROI fusion.

$$f_t = Tr_{t-1}(f_{t-1}^{M_l}) \cup Tr_{t-1}(f_{t-1}^{M_c}) \tag{3.7}$$

5. *Mixture of Experts*: This approach explicitly learns the weights of the feature maps. Initially, the feature maps are processed by separate networks and the output of these networks is then averaged with weights that are predicted using the gated network, and finally, the output is combined using simple addition operation[16].

$$f_t = w^{M_l} * (f_{t-1}^{M_l}) + w^{M_c} * (f_{t-1}^{M_c}),\ with\ w^{M_l} + w^{M_c} = 1 \tag{3.8}$$

The Equations from 3.4 to 3.8 are referenced from [16].

When to Fuse:

Because of the hierarchical representation of features by deep neural networks, it offers many options for data fusion such as early fusion, late fusion, and middle fusion[16]. Similar to the previous subsection, we will use the same mathematical parameter notations for describing the fusion techniques. Addition to that, we will define the fusion operation of two feature maps as $f_t = f_{t-1}^{M_l} \oplus f_{t-1}^{M_c}$[16].

1. *Early Fusion*: In this fusion technique, either raw data from various modalities or the preprocessed data are fused. This method can be referred to as data-level fusion provided Hall et al[24]. Main advantage of using this approach is its computational capability and time complexity. Here the features from each modality are learned together at an early stage and it exploits all the data provided by each modality. Due to its computational capability, it requires less memory for preprocessing various modalities, and this results in the inflexibility of the models. This is because, in case if one of the modalities is replaced by another modality, then the whole network needs to be trained to start

from preprocessing. These techniques are highly sensitive to spatial-temporal data misalignment, this might occur because of the wrong calibration of sensors, sampling rates, and defective sensors[16]. This fusion technique can be described mathematically as in Equation 3.9.

$$f_T = Tr_T\big(Tr_{T-1}\big(\cdots Tr_t\big(\cdots Tr_2(Tr_1(f_0^{M_l} \oplus f_0^{M_c}))\big)\big)\big) \tag{3.9}$$

2. *Late Fusion*: This technique fuses the individual decisions generated from individual sensing modalities. This method can be referred to as the decision level fusion provided by Hall et al[24]. Because of the fusion of the decision of individual modalities, these are highly flexible and modular. This is because in case a sensor needs to be replaced, then only that part of the network has to be retrained and final decisions are fused. Due to this the computation and memory cost increases and the mid-level features will be lost for final decision making. Mathematically this technique can be represented as,

$$f_T = Tr_T^{M_l}\big(Tr_{T-1}^{M_l}\big(\cdots Tr_1^{M_l}(f_0^{M_l})\big)\big) \oplus Tr_T^{M_c}\big(Tr_{T-1}^{M_c}\big(\cdots Tr_1^{M_c}(f_0^{M_c})\big)\big) \tag{3.10}$$

3. *Middle Fusion*: This technique can be considered as the hybrid of the above two fusion techniques. Here mid-level features from different sensors are fused, that is, the features generated from intermediate levels of individual networks are fused. This method can be referred to as feature level fusion technique provided by Hall et al[24]. This technique helps in learning of intermediate feature representation from different modalities at different depths. Even though these methods are highly flexible, the complexity of finding an optimal depth for starting the data fusion is high. There are few variations of middle fusions are available, namely mid-level deep fusion and short-cut fusion and are described in [9], [23], and [61]. Mathematically this technique can be represented as, considering we start fusion at a depth of l_*,

$$f_T = Tr_T\big(\cdots Tr_{l_*+1}\big(Tr_{l_*}^{M_l}\big(\cdots Tr_1^{M_l}(f_0^{M_l})\big) \oplus Tr_{l_*}^{M_c}\big(\cdots Tr_1^{M_c}(f_0^{M_c})\big)\big)\big) \tag{3.11}$$

The Equations from 3.9 to 3.11 are referenced from [16]. The above three fusion techniques are represented in Figure 3.2.

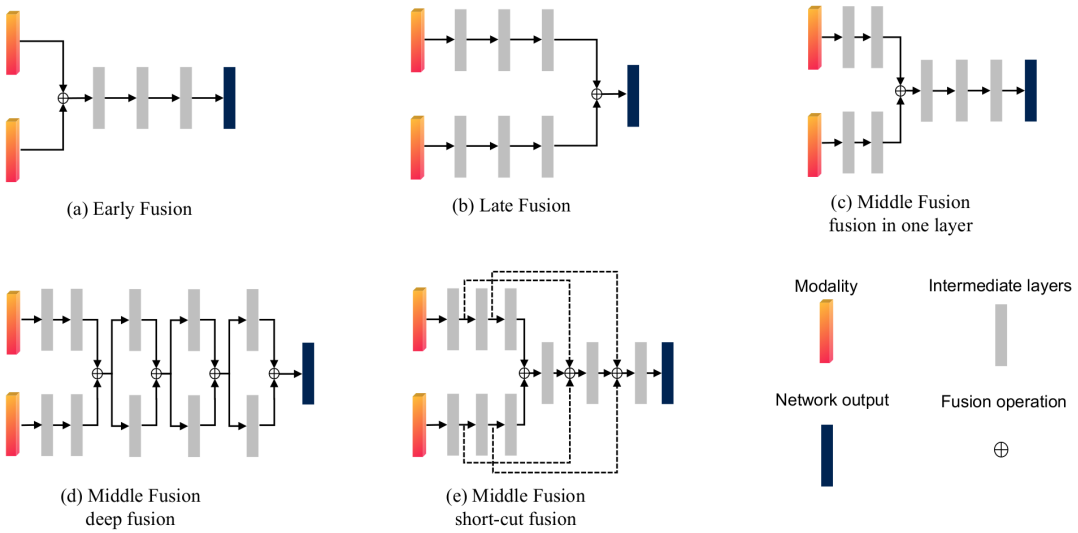


Figure 3.2: Image representing the early fusion, late fusion, and various middle fusion techniques [16, p. 9].

4

Solution Approach

The fundamental goal of this research is to provide a comparative analysis of the effect of using only the LiDAR sensor and the effect of using sensor fusion techniques on 3D object detection and classification task. From previous chapters, we have information on various sensors involved in the perception of the vehicle's environment, their advantages, and drawbacks, along with the information on various methods producing promising results for 3D detection and classification problem. In this chapter, brief information on various large-scale multi-modal datasets, 3D object detection frameworks that are based on LiDAR data and sensor fusion techniques, and their selection criteria for performing various experiments are presented.

4.1 Datasets

Generally, dataset refers to the set of data that can be used for performing various tasks such as, supervised learning applications, unsupervised applications. Almost all the published methodologies based on deep neural networks for 3D object detection and classification tasks utilizes supervised learning techniques. For supervised learning techniques, the dataset plays a very important role in the training and evaluation of the models. For a supervised learning technique, the more the data better the model learns to handle unseen events or scenarios. For improving the performance of these detection frameworks real-world datasets are important.

As discussed in previous chapters, applications related to autonomous vehicles such as object detection, localization of the vehicle, map building, requires input from multiple sources. In this section, a discussion on various multi-modal datasets that are available for research purposes, along with the selection of the datasets for performing the experimental analysis is presented.

4.1.1 Multi-Modal Datasets

The first large-scale dataset for the autonomous driving task which was released for public usage was KITTI dataset[18]. The initial version of the KITTI dataset was released in 2013 and from then on there are many datasets made available. For all the datasets that are available, the KITTI dataset is widely used and it is the benchmark for most of the researches. Further, we will discuss various large-scale datasets available publicly for autonomous vehicles application.

KITTI dataset was recorded using a station wagon by attaching various sensing modalities such as high-resolution stereo camera (both color and grayscale), Velodyne 3D laser scanner (LiDAR), and high-precision GPS/IMU inertial navigation system[19]. The latest version of this dataset was published in the year 2015. In this dataset, objects are categorized into various classes, for instance, 'Car', 'Pedestrian', 'Person (sitting)', 'Cyclist', 'Tram', and more. Here an almost equal amount of data for both training and testing purpose are provided. There are 7481 training frames and 7518 testing frames including 2D annotations, 3D annotations, and pixel-level segmentation information are available. The total size of the available dataset is 180GB[19].

An initial large-scale dataset named Oxford RobotCar[41] was published in the year 2016. The sensor suite consisted of visual cameras including fisheye and stereo cameras, 2D and 3D LiDARs, GPS, and inertial sensors. The dataset consisted of 11,070,651 frames of stereo cameras and 3,226,183 frames of 3D LiDARs. The major downside of this dataset is that there were no annotations provided[41].

In 2019, there were many large scale datasets made available for the public for research purposes. One such dataset is BLVD[74], this dataset consists of 120,000 frames with almost 249,129 objects with only 3D annotations and labeled into different categories for example 'Vehicles', 'Pedestrian', and 'Rider'. The sensor suite installed on the vehicle to record the dataset included LiDAR and visual stereo cameras[74]. Another large-scale dataset named H3D (Honda 3D)[46] was published with 27,721 frames and around 1,071,302 objects annotated only in 3D and labeled into categories such as 'Car', 'Pedestrian', 'Cyclist', 'Truck', 'Misc', 'Animals', 'Motorcyclists', 'Bus'. The sensor suite consisted of a 3D LiDAR and a visual camera[46].

A dataset named ApolloScape[26] was recorded in various regions of China with a sensor suite consisting of visual stereo cameras, 3D LiDAR, GNSS, and inertial sensors. This dataset provides around 143,906 frames and 89,430 objects categorized into 'Rover', 'Car', 'Pedestrian', 'Truck', and many more classes. In this dataset, the information on 2D and 3D annotations along with pixel-level segmentation, lane markings, and instance segmentation details are available[26]. A dataset named LyFT[28] that has over 1,300,000 3D annotations was released in 2019. The sensor suite consisted of a 3D LiDAR and visual camera for recording over 350 scenes that are 60 to 90 minutes long and the objects are categorized into 'Car', 'Pedestrian', 'Traffic lights', and more classes[28].

The nuScenes[5] dataset with almost 25 object classes was recorded with a sensor suite of six visual cameras, five Radars, and one 3D LiDAR for recording complete 360° view of the ego vehicle. The dataset consists of only 3D annotations and object classes such as 'Traffic Cones', 'Animals', 'Temporary Traffic Barrier', and many more. The dataset was collected in two different regions namely Boston and Singapore to show the diversity in driving conditions. The dataset consists of nearly 1000 scenes with 1.4 million frames with camera and Radar, and over 390,000 frames of 3D LiDAR data[5]. A dataset called WAYMO Open[65] dataset which was recorded using a sensor suite consisted of a complex sensor setup was released in late 2019. The sensor suite consisted of one 360° mid-range 3D LiDAR, five visual cameras, five Radars, and four short-range 3D LiDARs for complete 360° coverage of the ego vehicles surrounding. The dataset consists of both 3D and 2D annotations with various classes of objects for instance 'Vehicles', 'Pedestrian', 'Sign', and 'Cyclist'. The dataset consists of 1000 segments for training and 150 segments for evaluation,

each frame recordings is of 20 seconds duration consisting of both camera and LiDAR frames[65]. All the datasets mentioned above are summarized in Table 4.1.

4.1.2 Selection of Dataset

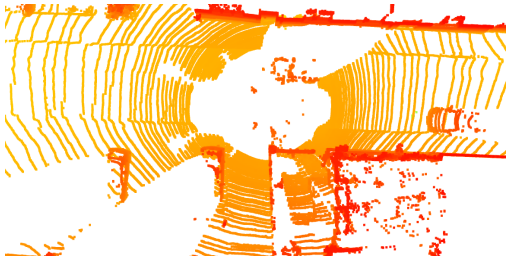
In this section, we will discuss the selection of the datasets for performing various experiments that support the argumentations provided in this research. For conducting experiments we select two distinct, diverse, and complex datasets to analyze the impact of the variations in the data on 3D detection algorithms.

Until today, all the published deep learning approaches are based on the KITTI dataset. The availability of the dataset is convenient and easily accessible. There are many development kits available for this dataset to understand, visualize, and experiment with. The data format is straight forward where the images are in portable network graphics (PNG) format and are available in both color as well as grayscale versions, corresponding LiDAR scans for each image is available in floating point binary format where each point in the data is represented by (x, y, z) coordinates and additional value of reflectance. Corresponding 2D and 3D annotations for each individual images are provided in text file format for easy access and corresponding calibration file that consists of intrinsic and extrinsic camera calibrations along with Velodyne to camera calibrations and Velodyne to IMU calibrations are provided. To make it easier for developers, the dataset is already partitioned into training (with labels) and testing (without labels) subsets. Evaluation metrics are also provided for various tasks such as 2D and 3D object detections, semantic segmentation, object tracking, and more. These evaluation metrics are used as a benchmark for various other datasets as well.

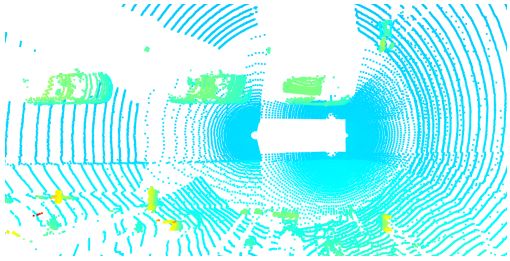
Considering the other datasets discussed above, Oxford RobotCar is a large dataset but no annotations are provided, and annotating this dataset will consume a lot of time and cannot be done by one person. Other datasets for instance BLVD, H3D, ApolloScape, do not provide the complete 360° coverage using visual sensors. These datasets usually use front facing stereo cameras and three visual cameras facing front and sideways. The datasets such as LyFT and NuScenes have occlusion effects on objects that are too close to the ego vehicle due to single long-range LiDAR mounted on top of the vehicle. WAYMO dataset does not have this problem as it is equipped with short-range LiDARs.

The other dataset which is under consideration is the WAYMO Open dataset. This is one of the large-scale and complex datasets that are available for research purposes. This dataset has been recorded under various driving conditions, various scenarios, different weather conditions, different times of day and night, and has more objects annotated in both 3D and 2D with different class labels. This dataset is provided in TFRecord format which is basically used for storing the sequential binary records. Here each TFRecord file has images, point cloud, calibration (both camera and LiDAR), and label information. All the other datasets discussed above are equipped with only one 3D LiDAR mounted on top of the vehicle for capturing 360°, but the roof of the car will affect the field of view for the objects that are too close to the vehicle. In the WAYMO Open dataset, there are additional short-range LiDARs attached at for edges of the vehicle to overcome this issue with closer objects. This can be seen in Figure 4.1. WAYMO Open dataset also provides 3D and 2D annotations for all 1000 segments of the training data collected.

Based on the arguments discussed above we choose the KITTI dataset and WAYMO Open dataset for performing various experiments in this research. A detailed description on KITTI dataset and WAYMO Open dataset is provided in upcoming Chapter 5.



(a) LiDAR point cloud without short range LiDAR and no information on the objects that are near to ego vehicle.



(b) LiDAR point cloud with short range LiDAR and more information on the objects that are near to ego vehicle.

Figure 4.1: Images representing the advantages of having short-range LiDARs along with a long-range LiDAR in recording 360° data over a single long-range LiDAR.

4.2 3D Object Detection Frameworks

As introduced in Chapter 2 under Section 2.2 and 2.3, there are several state-of-the-art deep learning-based 3D object detection frameworks. Here we will discuss the approach, strengths, and weaknesses of the frameworks that help in choosing the appropriate 3D detection frameworks for performing the comparative study. Based on the drawbacks related to image only frameworks discussed in previous chapters and lack of availability of direct depth information with images, in this section, we will discuss the frameworks that perform object detection on LiDAR data and sensor fusion techniques.

4.2.1 LiDAR Based

The main advantage of using LiDAR is because of its accuracy and precision. LiDAR provides a three-dimensional position of a point or an object. This helps in localization and obstacle avoidance by knowing the distance to impact from various objects. These are not affected by headlights of oncoming vehicles or the bright sunlight, these are also effective in dark and during night times. With these favorable characteristics of LiDAR, the first state-of-the-art detection framework with promising results were from VoxelNet[79].

VoxelNet directly works on the sparse point cloud data by eliminating the need for manual feature engineering. VoxelNet architecture (shown in Figure 4.2) is subdivided into three networks, (1) Feature Learning Network, (2) Convolutional Middle Layers, and (3) RPN layer. In Feature Learning Network, first, the sparse 3D space is equally partitioned into voxels based on the range of distance, height, and width, based on the availability of the points in each voxel, the points are grouped and due to the varying number of points in each voxel, random sampling is performed on voxels that have points that are more than the specified value. This helps in computational efficiency and a decrease in the imbalance of points.

The main advantage of this approach is introducing the Voxel Feature Encoding (VFE) layers. In VFE layer corresponding centroid is calculated for all the points in a voxel and these points are augmented using the relative offset with respect to the centroid. These are then processed using the FCN layers to transform into feature space to encode the surface information contained in the particular voxel. Similarly, the empty voxels undergo the same process as other voxels. Finally, the output feature of this layer is a combination of point-wise features, and the aggregated features stacking all the outputs from each VFE layer to encode the point features in voxel to learn the descriptive shapes. These are then represented as sparse tensors and provided as input to Convolutional Middle Layers. In this layer, the 3D convolution operation is performed along with batch normalization and the ReLU layer. Here the sparse tensors will be aggregated and progressively expanded by adding context to shape descriptions. These feature maps are then processed using RPN and finally, two maps are generated one as probability score map and other as regression map[79].

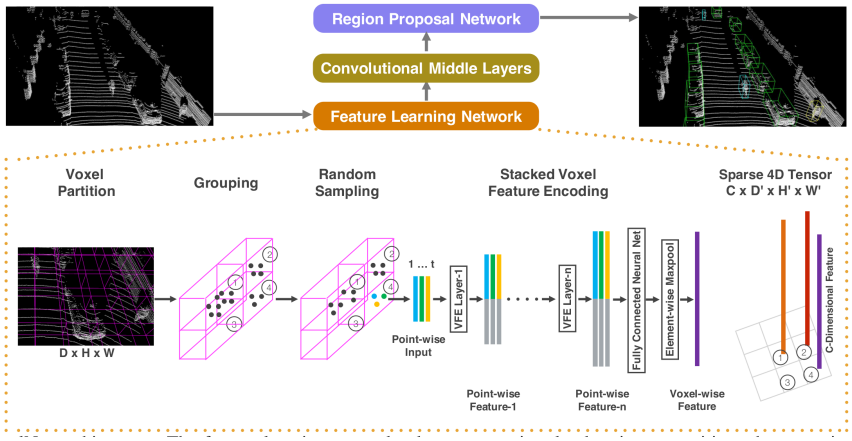


Figure 4.2: Architectural overview of VoxelNet[79, p. 2] representing the feature learning network, convolutional middle layers, and RPN.

SECOND[75] introduces sparse convolution operations that reduce the complexity and computational requirements compared to VoxelNet. Here basically the information from the z-axis is extracted before the 3D data is downsampled, which reduces the loss of data. The architecture (shown in Figure 4.3) consists of three components, (1) voxel feature extractor, (2) sparse convolutional middle layer, and (3) RPN. Initially, point cloud grouping is performed using the same procedure proposed by VoxelNet and modifying it based on the number of points per voxel. This offset is varied based on the type of object being detected, for the detection of vehicles the offset value is more compared to the offset value for the detection of pedestrians and cyclists. Here, similar to VoxelNet, the VFE layer is utilized for extracting the voxel features. In the sparse convolutional middle layer, the information from the z-axis is learned and converts the 3D data into 2D BEV images. This layer consists of many submanifold convolution layers along with a sparse convolution layer that downsamples the z-axis to create a dense feature map[75]. These feature maps are provided as input to RPN, here RPN has three stages, each stage begins with down-sampling followed

by convolution operations and finally batch normalization and ReLU layers. Finally, the output of each stage is upsampled and concatenated to form a single feature map. On this feature map, a convolution operation is performed for the prediction of class, and regression offsets[75].

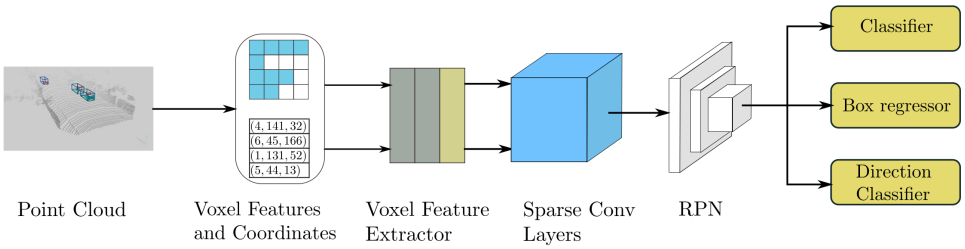


Figure 4.3: Architectural overview of SECOND detector[75, p. 4] representing VFE layer, sparse convolutional layers, and RPN.

IPOD[76] introduces a different way of extracting proposals directly from the raw point cloud. First, using the annotations 2D semantic segmentation is performed on images. These segmentation results are then projected on to the raw point cloud for the generation of proposals. Using the NMS approach, point-based proposals based on positive proposals are extracted and the background points are removed. Architecture (shown in Figure 4.4) is divided into a backbone network that is based on PointNet++[49] architecture which helps in processing the raw point cloud. This layer consists of several set abstraction layers and feature propagation layers that help in gathering local features and enlarging them. The second subcomponent is the Proposal Feature Generation, each proposal has two parts, the first part is cropped from the extracted feature maps and the second part is the canonized coordinates of the selected points. the final feature map is the concatenation of these two parts. The third subcomponent is the Bounding-Box Prediction Network, here PointNet++ is used for prediction of class, size ratios, and orientation information[76].

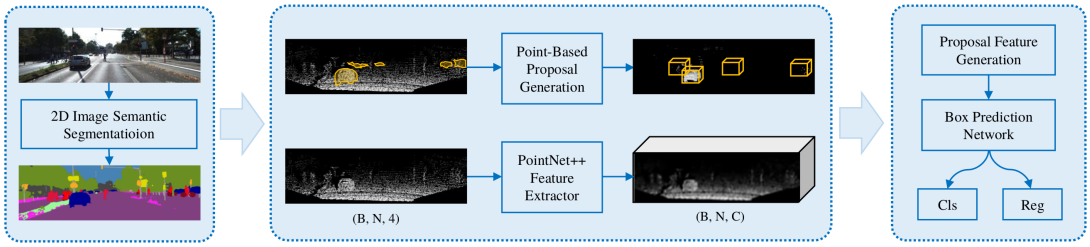


Figure 4.4: Architectural overview of IPOD[76, p. 2] representing subsampling network, proposal generation network, and backbone network.

PointRCNN[54] performs 3D detection by performing point cloud segmentation. The whole architecture (shown in Figure 4.5) is divided into two stages. The first stage performs a bottom-up approach for generating 3D proposals and the second stage refines these proposals in canonical coordinates for generating final results. In the first stage, the segmentation masks are extracted from the annotated 3D bounding

boxes and are considered to be foreground points. Here, for learning the point cloud representations PointNet++ backbone network is utilized. Using these segmentation masks, the foreground segmentation on point cloud is learned which in turn helps in learning the contextual information. In this bottom-up approach, proposals are directly generated from the foreground points, that is both segmentation and proposal generation happen hand in hand and using NMS techniques top 100 proposals are preserved for the refinement stage. Based on these proposals, the points are pooled again to obtain a highly accurate location and orientation of objects. These pooled points and the proposals are presented as input for the canonical refinement stage. These inputs are then transformed using the canonical transformations and this helps in learning the local spatial features for each proposal. This refinement stage combines these local spatial features and the global semantic features from the first stage for bounding-box regression and confidence refinement[54].

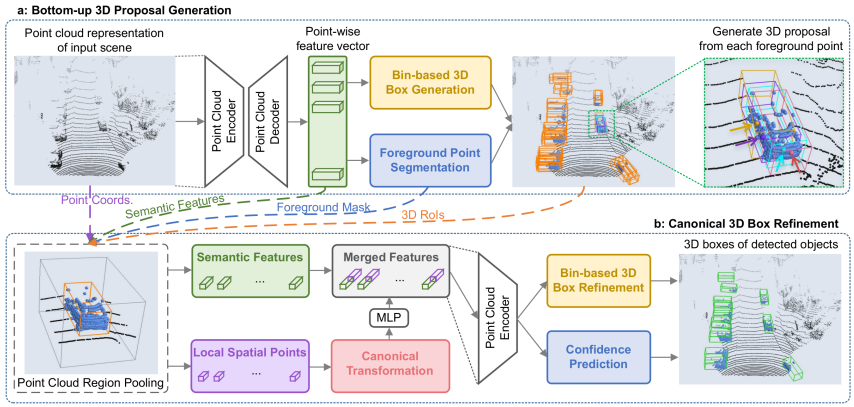


Figure 4.5: Network illustration of PointRCNN [54, p. 3] representing bottom-up layer and refinement layer.

PointPillars[31] introduces a fast encoding technique for raw point clouds for 3D object detection and classification. Here the point clouds are preprocessed and organized into vertical columns called pillars. On these pillars, the proposed encoders utilize PointNets for learning point cloud representations, and 2D convolution detection architecture is used for detection tasks. The architecture (shown in Figure 4.6) is basically divided into three subcomponents, (1) Feature Encoder Network, (2) 2D convolutional backbone, and (3) detection head. The first subcomponent is used to convert the raw point cloud into a pseudo-image. The point cloud is disjoined into an equally spaced grid in an x-y plane this results in pillar representation, each point in the pillar is then augmented with respect to the arithmetic mean of all the points in that pillar. Based on the number of non-empty pillars and the number of points per pillar a dense tensor is created. On this tensor, a simplified PointNet is applied to perform the linear operation, followed by batch normalization and ReLU and this is followed by a max operation to generate a pillar index tensor. These are then scattered back on to the location of the original pillar resulting in a pseudo-image. The backbone layer is similar to VoxelNet which is composed of two stages, the first network is a top-down network that generates features at an increasingly small spatial resolution and the

second network performs upsampling and concatenation of features from the first network. The detection head is based on SSD, here the 2D IOU is matched with the prior boxes generated[31].

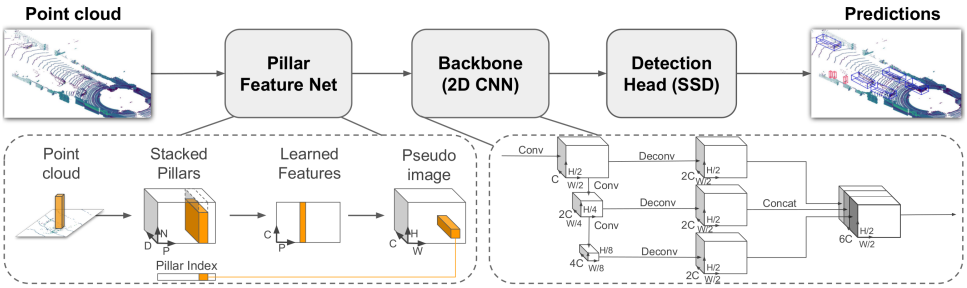


Figure 4.6: Architectural overview of PointPillars[31, p. 2] representing pillar feature network, backbone network, and a detection head.

4.2.2 Sensor Fusion Based

Cameras provide comprehensive details in its field of view along with the fine texture of the objects. RGB images are more powerful in representing the color factors like traffic signals or signals from other vehicles and helps in inferring the signboards. But, the downside of using only cameras is that they are sensitive to lighting and weather conditions. Since images are on a 2D plane, no depth information will be available and there is a high probability of occlusion effect on objects. On the contrary, LiDAR directly provides depth information and the occlusion effects are minimized, along with that LiDAR does not provide fine textural information of the scene but it is not affected by lighting conditions. To take advantage of these complementary properties, sensor fusion techniques are incorporated to provide a better understanding of the vehicle's environment and improve the accuracy of object detection and collision avoidance. Here, we will go through state-of-the-art sensor fusion frameworks that have published promising results in tackling the problem in hand.

MV3D[9] implements the fusion of different projection views of the sparse point cloud data and the RGB images to detect 3D oriented bounding boxes. The network (shown in Figure 4.7) is divided into two subnetworks, one is a 3D proposal network and the other is a region-based fusion network. Here, the sparse point cloud is represented in two views one is the BEV, and the other is front view projections, along with these two views RGB images are considered as well. The first subnetwork generates the 3D proposals on BEV projection of point cloud, this has three advantages, one it preserves the object's physical size, second tackles occlusion problems, and third is that object lies on the ground plane making it easier for localization of objects. Here during the training, the IOU between the anchor boxes generated and the ground truth are compared and if it is above 0.7 then these are treated as positive and if it is below 0.5 then these are treated as negative and the ones in middle are ignored. Using the NMS approach, the proposals are reduced and only the top 2000 proposals are considered for further processing. In the second subnetwork, ROI pooling is performed by projecting the 3D proposals onto the different views and the features are fused by utilizing the deep fusion technique (element-wise mean operation). On the final

fused feature map, orientation and classification regressions are performed[9].

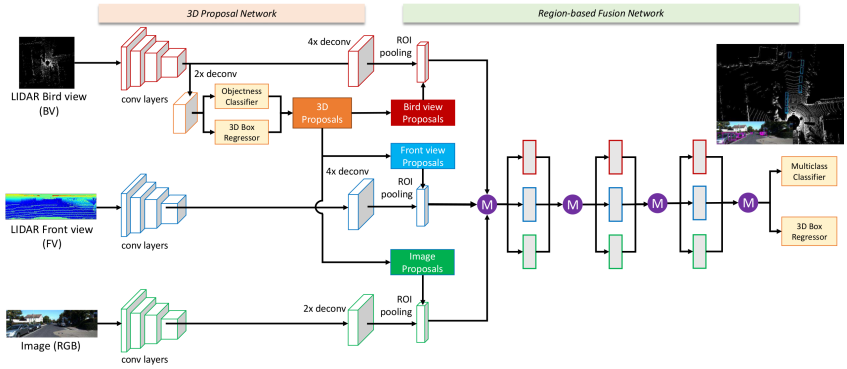


Figure 4.7: Architectural overview of MV3D[9, p. 2] representing 3D proposal network and region based fusion network.

AVOD[30] implements a feature extractor similar to feature pyramid networks (FPN) for better localization of small object classes. The network (shown in Figure 4.8) is divided into two parts, one is the implementation of RPN for performing the fusion of feature maps and the other is the detection head. Here, the point cloud is represented in BEV space using a similar approach as MV3D. The feature extractor is composed of two components, one is the encoder that is built by performing modifications on the VGG-16[57] framework to downsample the input data and the other is the decoder which is built based on FPN, that learns to upsample the feature maps to the original size as input. The output of these feature extractors is high-resolution feature maps and the same feature extractor is applied on both input modalities. Based on the anchors from ground truth, the feature maps for each modality are cropped and resized, and due to a large number of anchors, 1x1 convolution layers are implemented to maintain the memory constraints. These cropped and resized feature maps are of equal size and are fused using the element-wise mean operation to generate 3D proposals. These fused feature maps are then passed through the fully connected layers for generating the objectness score for the objects and perform orientation computations[30].

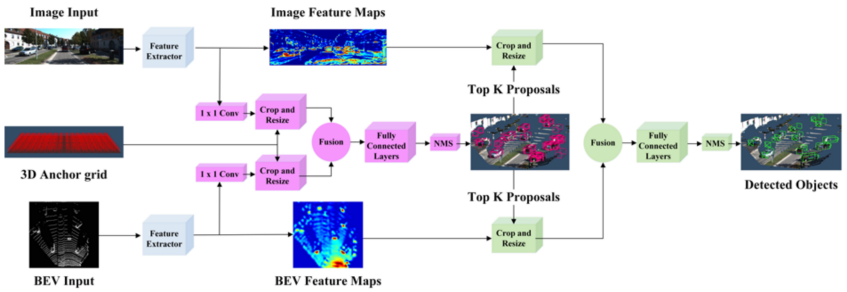


Figure 4.8: Architectural overview of AVOD[30, p. 2] representing feature extractors, RPN, and detection network.

ContFuse[36] proposes a continuous fusion approach that encodes discrete-state image features and the continuous geometric information from the point cloud. Here the features are learned by projecting the image features onto the BEV space of point cloud data. Using a convolutional network, the image features are extracted and are interpolated to fuse with the convolution layers of LiDAR and create a dense BEV feature map. For performing these operations, the architecture (shown in Figure 4.9) is divided into two streams, one for extracting features from images and other for extracting features from BEV space. These two streams are connected using the continuous fusion layers. The idea of the continuous fusion layer is to generate a dense BEV feature map by utilizing the feature maps from images and the point cloud. Given a few points from the point cloud, these layers map the pixel features from the image feature map to these points from point cloud to create a dense BEV feature map. These dense BEV feature maps are readily fused with the BEV feature maps from the second stream. The dense BEV feature map from stream one will be of the same size as the BEV feature map from the second stream, hence the element-wise summation operation is performed for fusion. The final detection head consists of a 1x1 convolutional layer for providing the detection outputs[36].

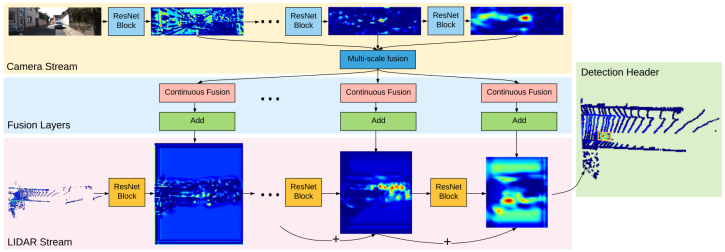


Figure 4.9: Architectural overview of ContFuse[36, p. 4] representing feature extractors for image and LiDAR, and continuous fusion layers.

Multi-Task Multi-Sensor Fusion[37] performs 2D and 3D object detection along with ground estimation and depth completions. Here, multiple perception tasks for instance ground estimation, depth completions, and object detections are performed at once for having better feature representations that in turn improves the detection accuracy. The network (shown in Figure 4.10) performs multiple tasks at once, the first task is to perform multi-sensor fusion by combining the point-wise features and the ROI based feature maps. The second task is to perform the ground estimations and the third task is to perform the depth completion and fuse this information to obtain the dense point-wise fusion features. For the first task, the features are extracted from image and BEV space using two separate backbone networks, and point-wise fusion is performed to obtain dense feature maps. This feature map is used for predicting the 3D detections for each BEV voxel using 2D convolutions. These detections are then reduced by applying the NMS approach to obtain high-quality 3D detections and 2D detections. These are then refined by applying ROI-wise feature fusion for generating accurate 2D and 3D detections. Along with this, the ground estimation is performed by first extracting the point-wise height with respect to the ground, based on point indices, and subtract it from the original height and create a new BEV related to ground level. This is given as input to the LiDAR backbone along with the original BEV and during the

3D regression phase, the original height is replaced back. This helps in better localization of objects in 3D space. For the depth completion task, the dense depth estimations for each pixel in the image are performed and pseudo-LiDAR data are generated for better depth estimation and finding the correct correspondence for the pixel at longer distance[37].

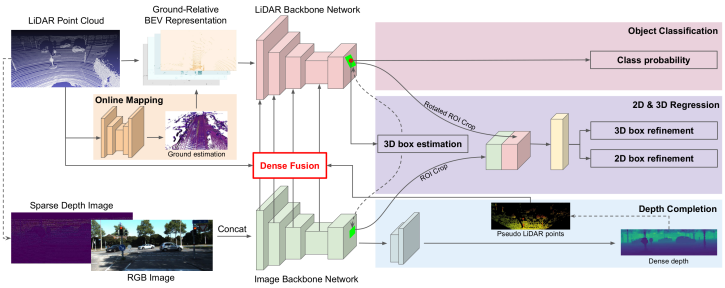


Figure 4.10: Architectural overview of Multi-Task Multi-Sensor Fusion[37, p. 3] representing the subtasks of ground estimations, depth completion and multi-sensor fusion.

F-PointNet[47] utilizes the advantages of 2D region proposal from 2D object detectors. The network (shown in Figure 4.11) performs three operations, the first operation is the generation of frustum proposals, second is 3D instance segmentation, and third is 3D amodal bounding box estimations. Using the camera projection matrix, the 2D bounding boxes estimated on the images are transformed into frustum that defines the 3D search space. The points within each of the frustums forms a frustum point cloud. These are then normalized to have uniform orientations and all this is performed in the frustum proposal generation phase. The 3D instance segmentation is performed on the point cloud using the PointNet[48] architecture. Here, the frustum point cloud is given as the input and the network predicts the likelihood of a point belonging to the object of interest. Here, each frustum corresponds to one object of interest, and other points are classified as background points. After segmentation, the points corresponding to the object of interest are classified as foreground points and the masks are extracted. For bounding box estimations, box regression PointNet is employed which takes these 3D masks of the objects as input and performs the classification tasks[47].

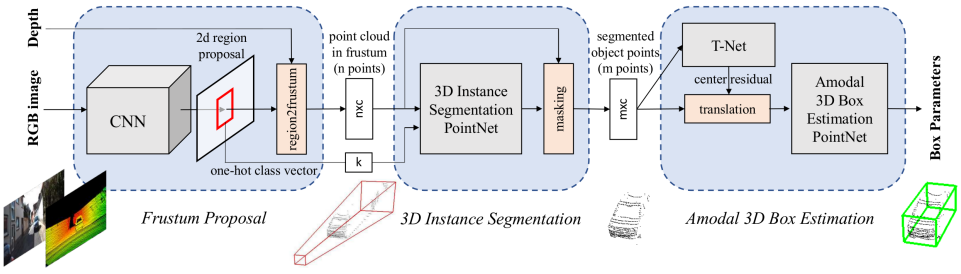


Figure 4.11: Architectural overview of F-PointNet[47, p. 3] representing frustum proposa generation, 3D instance segmentation, and amodal 3D box estimations.

F-ConvNet[64] performs amodal 3D object detections on point cloud by utilizing the 2D region

proposals from RGB images. Here, the point-wise features are aggregated as frustum-level features and are stacked up to form an array of features represented as feature maps, these are then processed using a fully convolutional network that fuses the frustum-level features spatially and performs the continuous estimations of orientations of 3D boxes. First, the points in the point cloud are grouped based on the RGB image associated with it and the 2D region proposals. Here, by sliding a parallel plane that is perpendicular to the frustum axis, a sequence of frustums is generated. For one single 2D proposal one frustum will be generated. The points inside these frustums are grouped together. The architecture (shown in Figure 4.12) consists of a frustum-level feature extractor, a fully convolutional network (FCN) for the fusion of frustum-level features, and a detection head. In frustum-level feature extractor, for each frustum, a PointNet[48] is applied for extracting point-wise features and is then aggregated into an array for processing by the FCN layer. In the FCN layer, the 2D feature map from PointNets is processed using convolution and deconvolution operations, and at the final layer, the features across the frustums are fused by convolution and down-sampling operations. The final feature map is the concatenation of all the feature maps, which in turn helps in 3D estimations. A detection head is built using the two parallel convolution layers for performing classification and regression[64].

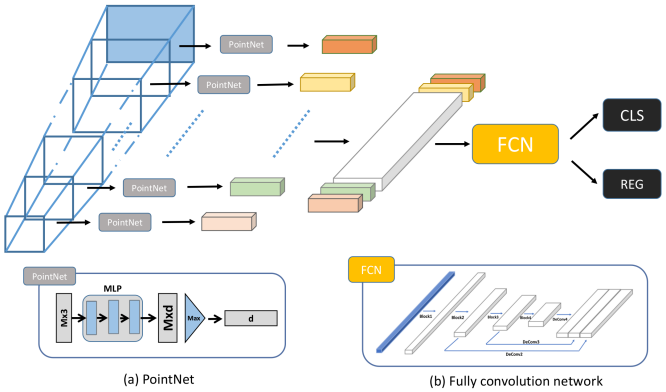


Figure 4.12: Architectural overview of F-ConvNet[64, p. 2] representing PointNet layers and FCN layers along with detection head.

4.2.3 Selection of Framework

In this section, we will discuss various aspects based on which the 3D object detection frameworks are selected for conducting the experiments. The objective of this work is to provide a comparative analysis of the impact of using a single modality (LiDAR) and using sensor fusion (LiDAR and camera) techniques on 3D object detection. For conducting various experiments, for supporting the objective of this research we select two deep learning-based object detection frameworks of which one takes only LiDAR data as input and the other takes multiple modalities (LiDAR and camera) as input and performs object detection.

First, we will discuss selecting the LiDAR-based framework. The first promising performance was published by VoxelNet, it introduced VFE layers for processing the raw point cloud and RPN layer for

detection tasks. But the issue with this approach is the use of 3D convolution operations, this increases the computational complexity and resource consumption. The inference time was too high making it not compatible to use in real-time application. To overcome this drawback, the SECOND framework introduced the usage of sparse convolution operations on the raw point cloud and algorithms using GPU based computations, this gradually decreased the inference time. Here, due to sparse convolution operations, smaller objects such as pedestrians were not detected properly and the detection accuracy was too low, and more false positives are reported for small objects.

IPOD extrapolates the instance segmentation performed on 2D images to generate the 3D proposals based on which the detection and classification are performed. This addresses the problem with small object detection but due to the involvement of complex tasks, the inference time is more which is not good for real-time application. PointRCNN performs 3D point cloud segmentation based on 3D annotations for proposal generation and then performs canonical transformations for classification and regression. The inference time is more than the SECOND framework and low accuracy on the detection of small objects.

PointPillars introduces a new approach for processing the raw point cloud by representing them in the column features called pillars and are two-dimensional vectors. These include a fast encoding technique that encodes the raw point cloud into a pseudo image. This makes it easier and faster for performing 2D convolution operation and then an SSD based detection head is utilized for detection and classification tasks. The inference time is too low and is faster than all previous approaches. The published results outperform all the previous LiDAR-based methods and are comparable to most of the state-of-the-art fusion-based methods. Here the detection accuracy of large objects is similar to that of sensor fusion methods up to a range of 70 meters and for small object detection up to a range of 48 meters, and the detection accuracy of PointPillars is more compared to previous methods.

Similarly, the first promising results using sensor fusion techniques were first published by MV3D, which exploits the advantage of combining multiple views for performing object detection. Here, the features from three different views are fused using deep fusion techniques. The issue with this method is that the far objects are not detected even though they are visible in images. This is because of the use of RPN only in BEV space, and the inference time is too high. AVOD extracts features from both the input modalities and fuses the features using element-wise fusion operation and performs early fusion. Here, only the front view images are considered and the point cloud is cropped to fit only the front view and limits the object detection only to the front view.

ContFuse performs continuous fusion operations, where the images are projected onto BEV space by performing interpolation. Due to the projection of images onto BEV space, the accuracy of detection of small objects is comparatively less and moreover the code replicating the results published in the research paper is not available. Multi-Task Multi-Sensor Fusion technique makes use of information from different tasks and fuses it to obtain a high-resolution feature map that improves the accuracy of detection immensely. This approach does not show any drawback and has published state-of-the-art results. The only problem here is the code to replicate the results published is not available and it takes a lot of time for replicating the code with the knowledge of the architecture and functionality of each component provided in the research paper.

F-PointNet uses two PointNet architecture for performing the segmentation and bounding box estimations. Here, the segmentation and grouping of the point cloud solely depend on the 2D proposals generated on images. The downside of this method is that it is not an end-to-end learning method for estimation of oriented boxes and the final estimations have highly relied on too few foreground points that are grouped based on 2D proposals. The inference time is more compared to previous methods.

F-ConvNet fuses the frustum-level features extracted by grouping the point cloud based on the extraction of frustums that depends on 2D region proposals. Here, each 3D point is identified that corresponds to the pixel level information from 2D proposals, and frustums are generated. On these sequences of frustums, a series of PointNet is applied and the output is stacked into arrays on which using the FCN these frustum-level features are fused by performing convolution and sampling operations. This method was published in 2019 and reported the state-of-the-art performances outperforming all the previous methods. But the only drawback here is the grouping of point cloud depends on 2D proposals. This method has reported high performance even for small object detection and the inference time is lower compared to others.

Based on the arguments provided in this section, we choose PointPillars framework to perform experiments related to LiDAR only object detection and Frustum ConvNet (F-ConvNet) framework to perform sensor fusion based object detection. The only reason for choosing the F-ConvNet framework over the Multi-Task Multi-Sensor Fusion method is that the code is available for research purposes. Further details on the architecture and implementation details are presented in the upcoming Chapter 5.

Datasets	**Sensors**				**Year Published**	**Data Annotations**			**Category**	**Size**
	LiDAR	**Camera**	**Radar**	**GNSS**		**3D**	**2D**	**Pixel level seg**		
KITTI	✓	✓	x	✓	2015	✓	✓	✓	Car, Pedestrian Person (sitting), Cyclist, Tram	7481 training, 7518 testing frames, 80,256 objects
Oxford RobotCar	✓	✓	x	✓	2016	x	x	x	No annotations	11,070,651 camera frames, 3,226,183 LiDAR frames
H3D	✓	✓	x	x	2019	✓	x	x	Car, Cyclist, Animals, Bus	27,721 frames 1,071,302 objects
ApolloScape	✓	✓	x	✓	2019	✓	✓	✓	Rover, Car, Pedestrian, Truck	143,906 frames 89,430 objects
BLVD	✓	✓	x	x	2019	✓	x	x	Vehicles, Rider, Pedestrian	120,000 frames 249,129 objects
LyFT	✓	✓	x	x	2019	✓	x	x	Car, Pedestrian, Traffic lights	360+ scenes (each 60-90 min)
NuScenes	✓	✓	✓	x	2019	✓	x	x	25 different categories	1.4M frames with camera and Radar, 390,000 frames with LiDAR
WAYMO	✓	✓	x	x	2019	✓	✓	x	Vehicle, Pedestrian, Sign, Cyclists	1150 segments each 20 sec long

Table 4.1: Summary of large-scale multi-modal datasets discussed in this research work.

5

Evaluation

In this chapter, a detail discussion on the datasets, various conditions under which the dataset was recorded, information on a sensor suite for dataset collection, metrics for the evaluation of object detection frameworks, along with the two deep neural network frameworks that are opted for conducting the experiments are presented.

5.1 KITTI

In this section, information on sensor suite, recording conditions, and the dataset format referenced from [19] are discussed. The KITTI dataset was recorded using a VW Passat station wagon in and around the city of Karlsruhe, Germany. The dataset consists of various data for instance camera images, laser scans, calibrations, annotations, high-precision GPS/IMU information. The vehicle was equipped with two sets of stereo camera setup that consisted of 2 PointGray Flea2 grayscale camera and 2 PointGray Flea2 color camera along with 4 Edmund Optics lenses with an opening angle of 90°, one 64 beam Velodyne HDL-64E rotating 3D laser scanner rotating at a frequency of 10 Hz with angular resolution of 0.09 and up to 2 cm distance accuracy. Laser scanner collected almost 1.3 million points per second up to a range of 120 meters with a horizontal field of view of 360° and vertical field of view of 26.8°, one OXTS RT3003 inertial and GPS navigation system running at 100 Hz with a resolution of $0.02m/0.1°$. The length between the two cameras in a stereo setup is 54 cm. To store the recorded data, the vehicle was equipped with a computer with two six-core Intel XEON X5650 processors along with a hard disk of the capacity of 5 Terabytes[19]. The sensor setup is represented in Figure 5.1.

The raw dataset is grouped into various categories for example road, city, residential, campus, and person. The complete dataset is divided into many categories depending on the sensor data and various task for instance, there is a subset of data only for stereo evaluation, optical flow evaluation, scene flow evaluation, tasks such as depth completion and prediction, map building, object detection and tracking in 2D and 3D, road/lane detection, semantic/instance segmentation, and the raw data. The dataset was recorded during daytime so there is no much variation in the lighting conditions of the data.

The images are stored with lossless compression using 8-bit PNG files. For every frame provided, a text file that consists of the information related to GPS/IMU is provided. The text file consists of 30 different values of geographic coordinates that includes altitude, orientation, velocity, angular rates, accelerations, and satellite information for each frame. Here the angular rates and acceleration information

is represented using two coordinates, one with respect to the vehicle that is (x, y, z) and other is with respect to the surface of the earth at that particular instance that is (f, l, u) and for missing information, it is mentioned as -1 in last three entries. The laser scans are stored as floating-point binaries. Each point is represented with (x, y, z) coordinates along with a reflectance value.

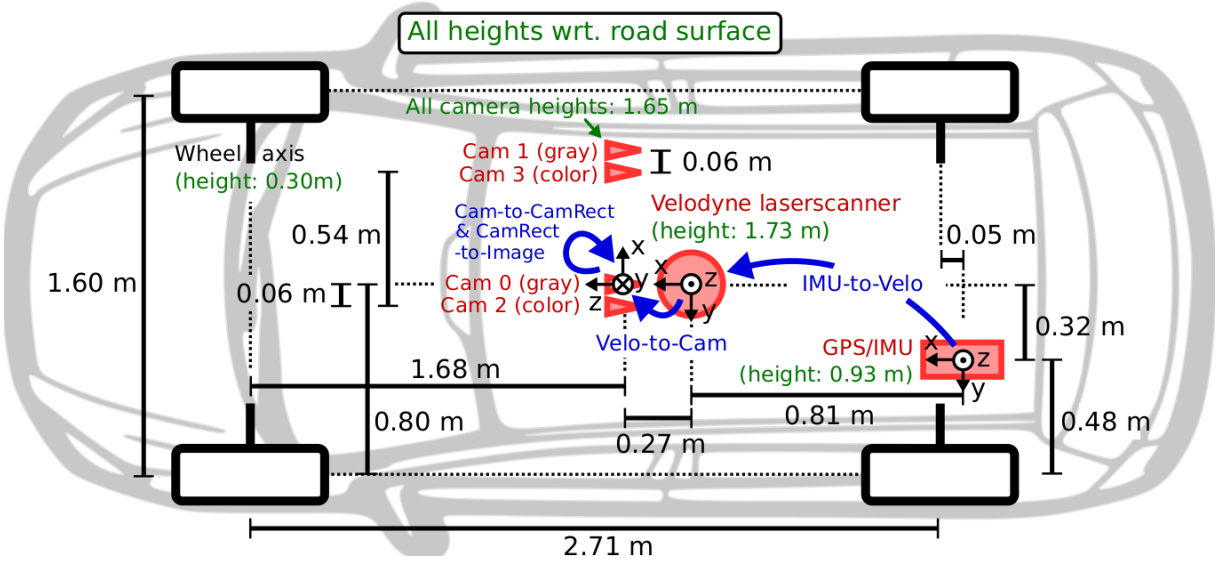


Figure 5.1: Overview of KITTI sensor suite setup[19, p. 2] along with their coordinate system illustrations. Measurement represented in green color are with respect to the ground surface[19]

All the dynamic objects that are in the field of view of the reference camera are annotated with 3D bounding boxes. These objects are categorized into classes such as 'Car', 'Pedestrian', 'Cyclist', 'Van', 'Truck', 'Person (sitting)', and 'Misc'. Each annotated object class is provided with the bounding box information along with the 3D size (length, width, and height), the rotation and translation information, the yaw angle, level of occlusion, and truncation information. Some sample images and their labels are provided in Figure 5.2.

As mentioned above, one single label represented by KITTI dataset consists of various fields that hold various information for instance, 'type' holds the information on what class the object belongs, 'truncated' - holds values between 0 and 1 representing the percentage of truncation of the object and if its value is 2 then it is ignored by manual labeling, 'occluded' - indicates occlusion state of the object, values ranges from 0 to 3, 0 is non-occluded, 1 is partially occluded, 2 is highly occluded, and 3 is unknown, 'alpha' - an observation on the orientation of the object, 'bbox' - 2D bounding box coordinates (hold 4 values), 'dimensions' - represents the dimension of object (height, width, length) in meters, 'location' - provides x,y,z in camera coordinates, 'rotation_y' - rotation around y-axis in camera coordinate, along with these fields there are other fields for tracking of objects such as 'track_id', 'frame', and for storing the detection results as well.

Sensor Calibrations

The sensors were well calibrated everyday before starting the recordings. As shown in Figure 5.1, each sensor has different coordinate direction, for cameras, $x = right,\ y = down,\ z = forward$, for LiDARs, $x = forward,\ y = left,\ z = up$, and for GPS/IMU, $x = forward,\ y = left,\ z = up$. For synchronizing these sensors, timestamps of laser scanner were used as a reference point and each complete spin is referred to as a frame. Due to the high frequency of GPS/IMU, the data recorded closest to a particular frame is used.

All the cameras are axis-aligned and lie on the x-y plane. The calibration file provided consists of the intrinsic and extrinsic calibration information for each camera along with the rectification matrix, transformation matrix from Velodyne to the camera, and the transformation matrix from IMU to Velodyne. Detailed information on the calibration of IMU/GPS to Velodyne can be referred to in the literature[19].

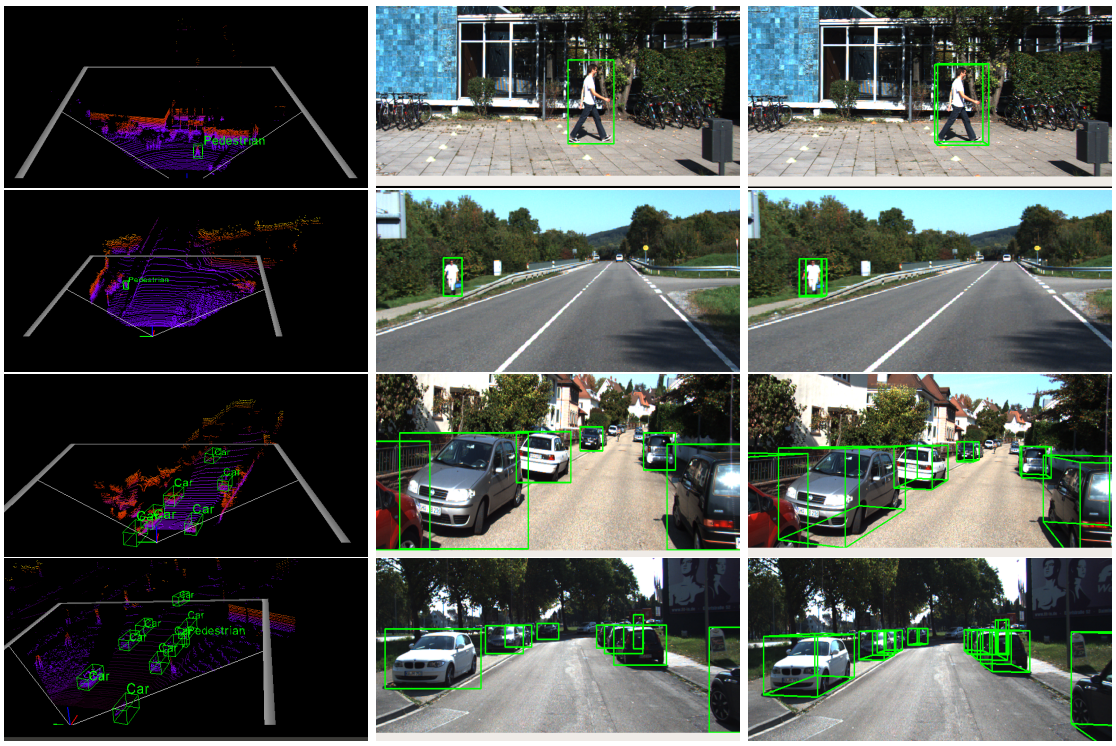


Figure 5.2: Sample images and LiDAR projection from KITTI dataset, with ground truth bounding boxes and projected 3D labels.

The projection of a 3D point $X = (x, y, z, 1)^T$ on to the camera coordinates represented by $Y = (u, v, 1)^T$ is given as (referenced from [19]),

$$Y = P_{rect}^{(i)} X \tag{5.1}$$

where, $P_{rect}^{(i)}$ is the projection matrix, it is given as (referenced from [19]),

$$P_{rect}^{(i)} = \begin{pmatrix} f_u^{(i)} & 0 & c_u^{(i)} & -f_u^{(i)} b_x^{(i)} \\ 0 & f_v^{(i)} & c_v^{(i)} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

where,

$b_x^{(i)}$ - Baseline with respect to reference camera.

$f_u^{(i)}$ and $f_v^{(i)}$ - Focal length

$c_u^{(i)}$ and $c_v^{(i)}$ - Principal point at which the optic axis intersects with the image plane.

But, the images are rectified, for projecting the point into camera coordinates, we need to consider the rectification matrix $R_{rect}^{(0)}$ as well as shown in Equation 5.2 (referenced from [19]).

$$Y = P_{rect}^{(i)} R_{rect}^{(0)} X \tag{5.2}$$

Based on the amount of visibility of the object and the height of the bounding box, the KITTI dataset provides three difficulty levels for measuring the performance of object detectors. These three difficulty levels are as follows and the example for each category is represented in Figure 5.3

1. *Easy*: Bounding box = Minimum height of 40px, Maximum occlusion level = Fully visible, and Maximum truncation level = 15%

2. *Moderate*: Bounding box = Minimum height of 25px, Maximum occlusion level = Partly occluded, and Maximum truncation level = 30%

3. *Hard*: Bounding box = Minimum height of 25px, Maximum occlusion level = Highly occluded, and Maximum truncation level = 50%

(a) Easy

(b) Moderate

(c) Hard

Figure 5.3: Example depicting easy, moderate, and hard difficulty levels for KITTI data evaluation.

5.2 WAYMO Open

In this section, information on sensor suite, recording conditions, and the dataset format referenced from [59] and [65] are discussed. WAYMO Open dataset is the most diverse and largest publicly available multi-modal dataset for autonomous vehicle applications. The WAYMO fleet is equipped with many in-house high-resolution cameras and multiple high-quality LiDAR sensors. Each vehicle in the fleet is equipped with one mid-range LiDAR on top of the vehicle, four short-range LiDARs on four sides of the vehicle, and five high-resolution cameras on front and sides of the vehicle. Sensor setup and the coordinate system information is given in Figure 5.4. The mid-range LiDAR has a vertical field of view of $[-17.6°, +2.4°]$ and limited to the range of 75 meters, and the four short-range LiDARs have a vertical field of view of $[-90°, +30°]$ and limited to the range of 20 meters. The high-resolution cameras generate the images of size 1920×1280 and the cameras that are connected on side-left and side-right generate the images of size 1920×1040, and all the cameras have the horizontal field of view of $\pm 25.2°$ [59].

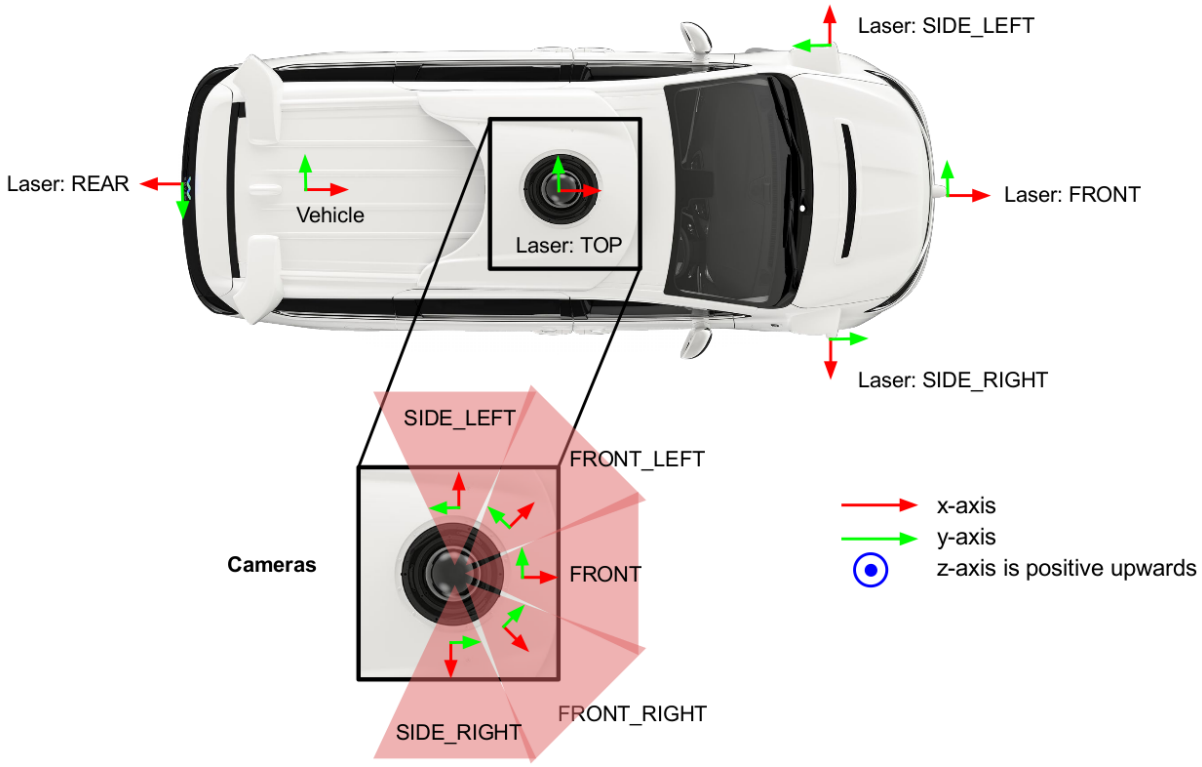


Figure 5.4: Overview of WAYMO sensor suite setup along with their coordinate system illustrations[59, p. 3].

To make this dataset more diverse, it has been recorded in multiple cities such as San Francisco, Phoenix, and Mountain View, and different driving conditions namely driving in daylight, nighttime, in rain, in fog, in snow, and construction areas as well. The dataset provides a large number of hand-

annotated 3D labels and 2D labels unlike most other datasets just provide projected 3D labels. The dataset consists of around 12 million LiDAR and camera bounding box annotations respectively. Along with normal LiDAR data, it is also provided in the form of range images, along with accurate vehicle pose of each pixel in the range image. There are 1000 training and validation scenes, and 150 testing scenes, each of which are 20 seconds long.

The dataset is complex and uses different frame coordinates for representing each sensor data. The global frame is with respect to which the data are recorded are set prior to vehicle motion where $x = east,\ y = north,\ z = up$, the vehicle frame moves along the vehicle $x = forward,\ y = left,\ z = up$. For each of the sensors, a sensor frame is defined as the 4×4 transformation matrix that transforms the data from the sensor frame to the vehicle frame. For LiDARs the z-axis points upwards and the other two axes depend on the particular LiDAR and same for the cameras as well, this can be seen in Figure 5.4. The image frame is defined for every camera and in a 2D coordinate system, where the origin lies at the top-left corner and $x = image\ width,\ y = image\ height$ [59]. Spherical coordinate system (range, azimuth, inclination) for the LiDAR is based on the Cartesian coordinate system (x, y, z) in the LiDAR sensor frame and it is calculated using the below equations (referenced from [59]).

$$range = \sqrt{x^2 + y^2 + z^2} \tag{5.3}$$

$$azimuth = atan2(y, x) \tag{5.4}$$

$$inclination = atan2(z, \sqrt{x^2 + y^2}) \tag{5.5}$$

The objects are categorized into classes such as 'Vehicles', 'Pedestrians', 'Cyclists', and 'Sign'. The 3D bounding boxes are exhaustively annotated and each of which is represented by $(cx, cy, cz, l, w, h, \theta)$, where, (cx, cy, cz) are center coordinates, (l, w, h) are length,width, and height, and θ denotes the heading angle. The camera images are also exhaustively annotated and each represents (cx, cy, l, w), where, (cx, cy) represents a center pixel of the bounding box and (l, w) represent the length of the bounding box along horizontal axis and width of the bounding box along the vertical axis of image frame[59]. Some sample images and their labels are provided in Figure 5.5.

As discussed above, the dataset provides both 2D and 3D annotations for each object in both the camera frame and LiDAR frame. In the camera frame, the label is of the 4-DOF bounding box and in the LiDAR frame, the label is of the 7-DOF bounding boxes. These annotations are categorized into LEVEL_1 and LEVEL_2 based on the human annotator and various statistics of objects.

The dataset is provided in the form of TFRecord files, where each TFRecord file consists of information on LiDAR scans of all 5 LiDARs, images that are compressed using JPEG compressions, LiDAR scans in the form of range images, sensor calibration, and annotations. Along with the information of various properties related to each pixel in the range image such as *range* - distance between origin in LiDAR sensor and the point, *intensity* - based on reflectance value, *elongation* - useful for classification of spurious objects (like dust, rain, fog), *no label zone* - specifies if the point lies in no label zone, *vehicle pose* - at the particular instance of recording, and *camera projections* - for projection of LiDAR point to camera coordinates[59].

The range images and camera projections can be seen in Figure 5.7 and Figure 5.6. A detailed description of various challenges, tasks, and gaps for research based on the WAYMO Open dataset is provided in the literature [59].

Figure 5.5: Sample images and LiDAR projection from WAYMO dataset, with ground truth bounding boxes and 3D labels in LiDAR frames.

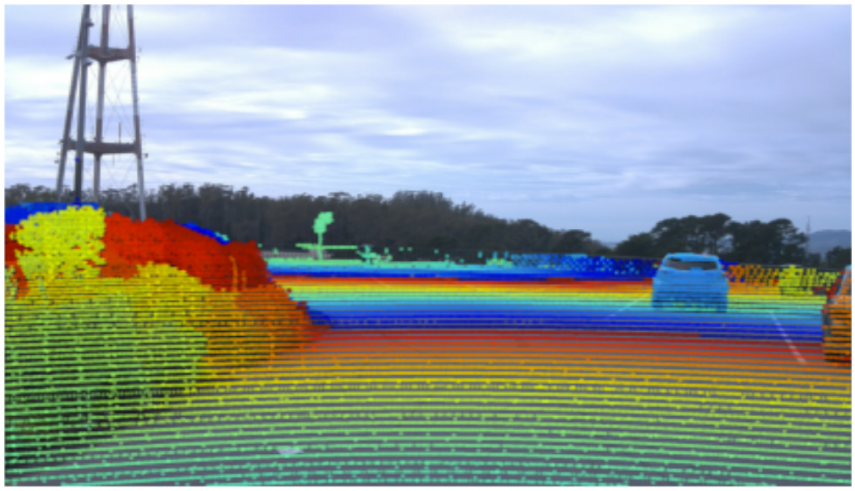
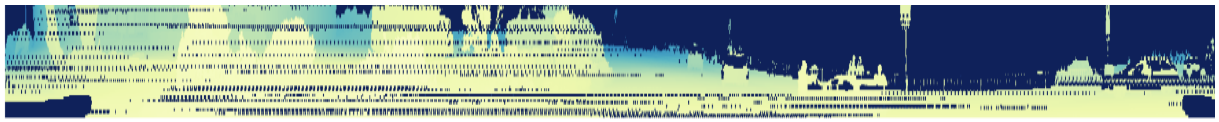
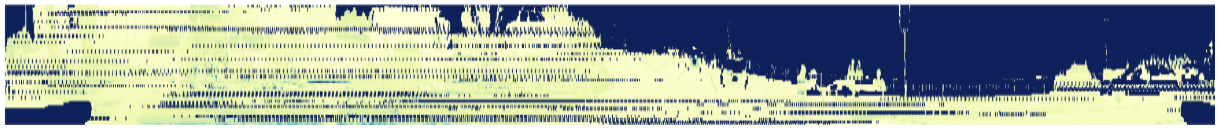


Figure 5.6: Projection of range image of LiDAR data on to a 2D image.

(a) Range property of LiDAR range image



(b) Intensity property of LiDAR range image



(c) Elongation property of LiDAR range image

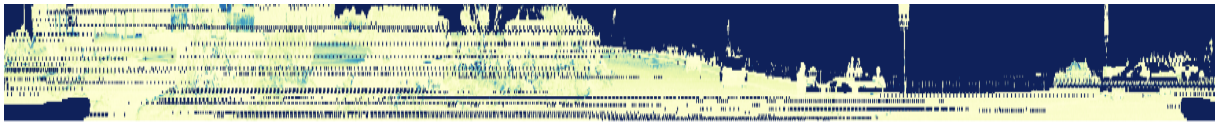


Figure 5.7: Sample range images of LiDAR point cloud from WAYMO Open dataset.

5.3 Metrics for Object Detection

Evaluation of object detection techniques mainly relies on the calculation of precision and recall values. Before defining what precision and recall is with respect to object detection, we will see the definitions of a few parameters in terms of object detection that are relevant for calculating precision and recall values.

- *True Positive (TP)*: Correct detection of the existent object that is defined by ground-truth label.
- *True Negative (TN)*: Correct detection of the nonexistent object or detection of an object in a wrong location.
- *False Positive (FP)*: Incorrect detection of the nonexistent object.
- *False Negative (FN)*: Not detecting the existing object or existing ground-truth labels.

Due to a large number of bounding boxes predicted per image, say RPN generates a large number of proposals for an image, and these should not be detected for an image, the TN case will not be utilized. Precision is defined as the capability of a model to detect or recognize only the relevant object. Precision (P) is the percentage of correct positive predictions[45]. Recall is the capability of a model to detect or to recognize all the relevant cases. Recall (R) is the percentage of correct positive predictions in the presence of all the ground-truth values[45]. Equation 5.6 and 5.7 represents precision and recall.

$$P = \frac{TP}{TP + FP} = \frac{True\ Positive}{All\ Detections} \tag{5.6}$$

$$R = \frac{TP}{TP + FN} = \frac{True\ Positive}{All\ Ground\ truth} \tag{5.7}$$

Intersection Over Union

The above parameters depends on determining which prediction is correct and which prediction is incorrect. A common way of checking the correctness of the detection is by checking the Intersection over Union (IOU) parameter. It is calculated as the ratio between the area of overlap (intersection) of the ground-truth bounding box and the predicted bounding box to the total area covered (union) by these two bounding boxes. IOU representation is depicted in Figure 5.8. Generally, we define a threshold value t, if $IOU > t$, then we classify that detection to be correct detection and if $IOU < t$, then it is classified as incorrect detection. This metric is highly used for bounding box evaluations.

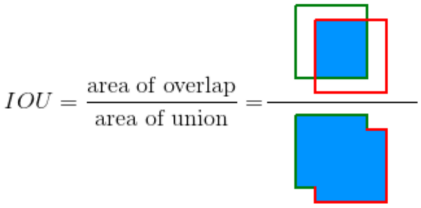


Figure 5.8: Image depicting the calculation of Intersection over Union (IOU)[45, p. 2].

Precision v/s Recall curve

A trade-off between the precision and recall values of the various confidence values that are associated with the predicted bounding boxes by a model is observed through the precision v/s recall curve (PR curve). Generally, the precision of a model is higher when the false positive rates are lower. In the case of object detection, due to the presence of many predicted bounding boxes for an object, many of the positives may be missed and this results in increasing the false negatives, in turn, resulting in increasing the recall values. On the contrary, if recall value increases, precision decreases. But, for a good object detection model, along with the detection of only the relevant object, it should also detect all the ground truth bounding boxes as well. So, an object detector model is considered to be good if its precision stays high even when its recall values are high. Hence, the area under the curve here provides high precision and high recall values[45]. A sample precision v/s recall curve is visualized in Figure 5.9.

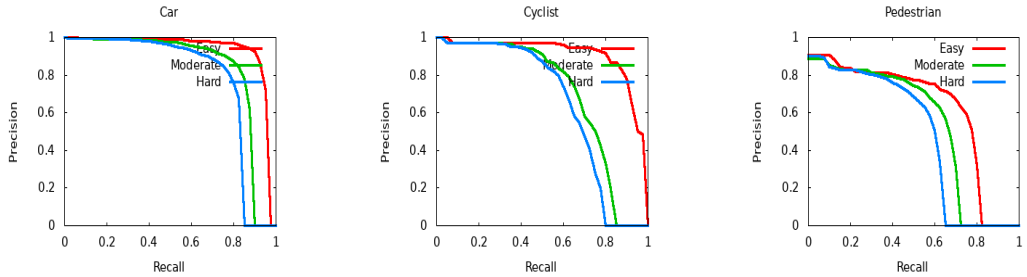


Figure 5.9: Sample precision v/s recall curves generated from Frustum ConvNet model for 'Car', 'Cyclist', and 'Pedestrian' classes

Average Precision

In practical applications, the PR curve will not be smooth and most of the time it will by varying making it difficult to calculate the area under the curve. To overcome this, a metric called average precision was introduced. Average precision is a numeric metric that helps in comparing various detectors based on their values. The AP basically summarizes the shape of the PR curve and is calculated by taking the mean of precision values at a set of eleven equally spaced recall values as shown in Equation 5.8[14].

$$AP = \frac{1}{R_{11}} \sum_{r \in \{0, 0.1, ..., 1\}} P_{interp}(r) \tag{5.8}$$

where,

$P_{interp}(r)$ - Interpolated precision value.

R_{11} - 11-point interpolation points and it is equal to 11.

Interpolation of precision at each recall level r is performed by considering the maximum precision value measured for which the corresponding recall value exceeds r as shown in Equation 5.9[14].

$$P_{interp}(r) = \underset{r':r' \geq r}{max} P(r') \tag{5.9}$$

Where, $P(r')$ is the measured value of precision at a recall value r'.

The main reason for 11-point interpolation is to minimize the effect of the zigzag effect of the PR curve. This is done to penalize the models that do not provide a precision value at all recall points. The main reason for introducing this metric is to compare different methods and to assign a rank based on the performance[14].

The average precision AP_i over all the N classes in a dataset can be calculated by using the mean average precision (mAP) and is given as (referenced from [45]),

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{5.10}$$

Until October 2019, the official KITTI dataset implemented an 11-point interpolation for calculating average precision. The intervals for the recall values start at 0, this means that a single correct prediction is sufficient to obtain an average precision of 100%, that is, the evaluation produces $\frac{1}{11} \approx 0.0909$ over entire dataset[56]. This might show a better performance for the models but not the quality of the algorithms. To overcome this Andrea et al[56] proposed to perform 40-point interpolation instead of 11-point out of provided 41-points. This is done to eliminate the glitch encountered at the lowest recall bin[56]. This is shown in Equation 5.11. From October 2019 onwards, the KITTI official dataset updated their test servers.

$$AP = \frac{1}{R_{40}} \sum_{r \in \{1/40, 2/40, 3/40, ..., 1\}} P_{interp}(r) \tag{5.11}$$

Average Orientation Similarity

The orientation similarity of detection at a particular recall value is a normalized value of the cosine similarity which is defined as shown in Equation 5.12[18].

$$s(r) = \frac{1}{|D(r)|} \sum_{i \in D(r)} \frac{1 + cos\Delta_{\theta}^{(i)}}{2} \delta_i \tag{5.12}$$

where,

$D(r)$ - Set of all object detections at recall r.

$\Delta_{\theta}^{(i)}$ - Difference in angle between the ground-truth and estimated orientations of the i_{th} detection.

δ_i - Penalty score.

$$\delta_i = \begin{cases} 1 & i_{th} \textit{ detection assigned to a ground truth bounding box} \\ 0 & i_{th} \textit{ detection not assigned to a ground truth bounding box} \end{cases}$$

Average Orientation Similarity (AOS) is used to assess the performance of the 3D detection frameworks for detection and 3D orientation of the objects. This is defined as shown in Equation 5.13[18]. The AOS is also known as Average Heading Similarity[30].

$$AOS = \frac{1}{11} \sum_{r \in \{0, 0.1, ..., 1\}} \underset{r':r' \geq r}{max} s(r') \tag{5.13}$$

AP Bird Eye View and AP3D

AP_{BEV} helps in evaluating whether the prediction of an object is accurate with respect to the ground-truth labels by performing the IOU of prediction with respect to the ground-truth from BEV space. This metric helps in evaluating the conditions for collision avoidance. AP_{3D} counts the IOU of 2 cuboids one from prediction and other from the ground-truth and adds the information of the height of object and up-down based on the object's location. AP_{BEV} basically evaluate the projection of 3D bounding boxes on to BEV space[6].

For 3D localization of objects, generally, the 3D bounding boxes are projected on to BEV space that results in the oriented BEV bounding boxes. So evaluating these bounding box predictions using AP_{BEV} results in determining the accuracy of localization as well. The other name of AP_{BEV} is AP_{loc} for BEV bounding boxes[9].

Metric Proposed by WAYMO[59]

Accurate heading predictions form the basic and critical for tasks such as object tracking and analysis of the prediction of the dynamic objects. For this predictions, general AP does not provide information

on heading of the objects, hence a metric named Average Precision Heading (APH) is proposed and is defined as

$$APH = 100 \int_0^1 max\{h(r')|r' \geq r\}dr \tag{5.14}$$

Here, $h(r)$ is calculated similarly to the calculation of average precision at a particular recall value, in addition to that, each true positive detection are associated with heading accuracy and is defined as $min(|\theta' - \theta|, 2\pi - |\theta' - \theta|)/\pi$, where, θ' and θ are the angles of the predicted heading and the ground-truth heading and is mentioned in radians in range of $[-\pi, \pi]$[59].

The implementation of this metrics considers a set of predictions whose scores are normalized between 0 and 1, and uniformly samples a fixed number of threshold scores. For each of the sampled threshold score, it performs a Hungarian matching between the predictions whose score is above the threshold and the ground truths, this maximizes the overall IOU between the matched pairs. WAYMO also proposes to add the conservative precision values between the two recall scores if the difference between the two consecutive recall values is greater than the threshold value. This helps in avoiding the overestimation of AP due to the sparse PR curve[59].

5.4 PointPillars

In this section, the network architecture and the implementation details from [31] are discussed. PointPillars architecture implements a novel approach for encoding the raw point cloud in a faster way. PointNets are utilized for encoding the point clouds into vertical columns named pillars for prediction of the oriented 3D bounding boxes for the objects. The encoded features will be in 2D and in the form of pseudo-image. This enables in performing end-to-end learning using 2D convolutions. This approach poses several advantages, first, the complete information represented by the point clouds are grasped by learning the features instead of relying on fixed encoders. Second, there is no need for hand-tuning the voxel binning in a vertical direction. Finally, encoding point clouds in the form of pillars enables to use the 2D convolutions that are efficient with GPU.

Major contributions here are, first, novel encoding technique of raw point clouds and end-to-end learning techniques for 3D oriented bounding box detection. Second, computations on pillars by utilizing the dense 2D convolutions that enable faster inference. Finally, conducting various experiments on the KITTI benchmark dataset and provide state-of-the-art results with inference time of 62 Hz which is 2 to 4 times faster than other encoding techniques.

Network Architecture

PointPillars directly accepts the raw point clouds as input and provides the estimations of oriented 3D bounding boxes for objects. To this extent, the network involves three subcomponents, (1) A feature encoder network to encode the raw point cloud into pseudo-images, (2) 2D convolutional layers, and (3) Detection head. The architecture is depicted in Figure 5.10.

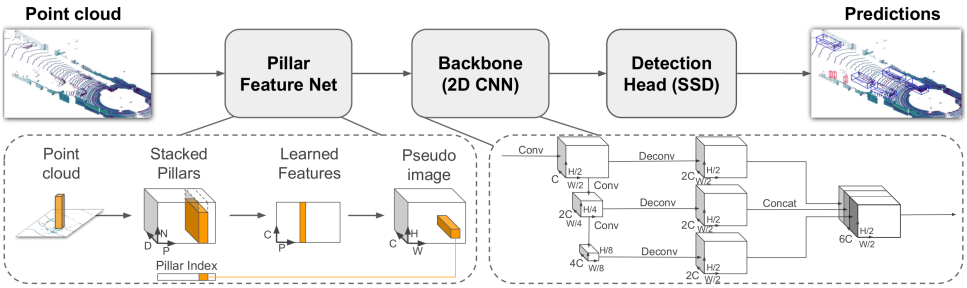


Figure 5.10: Architectural overview of PointPillars[31, p. 2] representing pillar feature network, backbone network, and a detection head.

Initially, the weights of the network are randomly initialized. The encoder network provides an output of 64 features ($C = 64$). The backbone network, that is the second subcomponent (2D convolutional layer) is same for detection of 'Car' and 'Pedestrian/Cyclist' except for the change in the stride of the first block, that is for 'Car' stride $S = 2$ and for 'Pedestrian/Cyclist' stride $S = 1$. The backbone network consists of three blocks of configuration, $Block1(S, 4, C)$, $Block2(2S, 6, 2C)$, and $Block3(4S, 6, 4C)$. Each of these blocks is upsampled by following configurations, $Up1(S, S, 2C)$, $Up2(2S, S, 2C)$, and $Up3(4S, S, 2C)$ and the features from these three upsamples are concatenated to generate a $6C$ feature vector that is provided for detection head[31].

For modeling the loss functions, the ground truth labels and the predicted bounding boxes are represented by $(x, y, z, w, l, h, \theta)$. The residual differences between the ground truth and predicted boxes are calculated as below and are referenced from [31].

$$\Delta x = \frac{x_{gt} - x_a}{d_a}, \Delta y = \frac{y_{gt} - y_a}{d_a}, \Delta z = \frac{z_{gt} - z_a}{h_a}$$

$$\Delta w = log\frac{w_{gt}}{w_a}, \Delta l = log\frac{l_{gt}}{l_a}, \Delta h = log\frac{h_{gt}}{h_a}$$

$$\Delta\theta = sin(\theta_{gt} - \theta_a)$$

$$d_a = \sqrt{w_a^2 + l_a^2}$$

Where,

x_{gt}, y_{gt}, z_{gt} - Ground truth axis

x_a, y_a, z_a - Predicted (anchor) axis

w_{gt}, l_{gt}, h_{gt} - Ground truth width, length, and height

w_a, l_a, h_a - Anchor box width, length, and height

$\theta_{gt} - \theta_a$ - Yaw angle of ground truth and anchor boxes.

The total localization loss is defined as (referenced from [31]),

$$L_{loc} = \sum_{b \in \{x,y,z,w,h,l,\theta\}} SmoothL1(\Delta b) \tag{5.15}$$

The classification loss is taken from [39] and is defined as

$$L_{cls} = -\alpha_a (1 - p_a)^{\gamma} log p_a \tag{5.16}$$

Where,

p_a - Class probability of anchor

α - Weighting factor which is equal to 0.25

γ - tunable parameter in range 0 to 5, and is equal to 2

For calculating the loss in directions L_{dir}, variations of the anchor and ground truth boxes with softmax classification loss incorporated as in [75]. The total loss is defined as (Equation 5.17 referenced from [31])

$$L = \frac{1}{N_{pos}} (\beta_{loc} L_{loc} + \beta_{cls} L_{cls} + \beta_{dir} L_{dir}) \tag{5.17}$$

Where,

N_{pos} - Number of positive anchors

β - Constant coefficients and $\beta_{loc} = 2$, $\beta_{cls} = 1$, and $\beta_{dir} = 0.2$.

For optimizing the loss function, ADAM optimizer with an initial learning rate of $2e^{-4}$ and decaying it by a factor of 0.8 for every 15 epochs for over 160 epochs with a batch size of 2 on the validation set is implemented[31].

Network Implementation

As discussed earlier, the network is subdivided into (1) A feature encoder network to encode the raw point cloud into pseudo-images, (2) 2D convolutional layers, and (3) Detection head.

A. Feature Encoder Network

This accepts the 3D raw point cloud as input and provides the encoded pseudo-image in 2D. Consider a point p in the point cloud that is characterized by (x, y, z, r) in the Cartesian coordinate system. The first step here is to discretize the point cloud into a set of pillars P by creating an evenly spaced grid in the x-y plane and there is no need for hyper-parameter tuning for binning in the z-axis as no voxelization technique involved. The points in each pillar are then augmented with respect to the distance from the point and the arithmetic mean of all points, and the offset from the center of the pillar and are denoted by $(x_c, y_c, z_c, x_p, y_p)$. This augmented point p will now be of nine dimension with $D = 9$ [31].

Due to the sparsity of the raw point cloud, most of the pillars will be empty and to oversee this effect and create a dense tensor, a limit on the number of non-empty pillars per sample (P) and the number of points per pillar (N) is imposed and a dense vector (D, P, N) is created. Using simplified PointNets, this dense tensor is processed to create a tensor of size (C, P, N) where C is the stacked columns. By performing max operation this tensor is converted into a 2D tensor of size (C, P) creating the encoding of the point cloud. After encoding, these are scattered on to the original pillar locations thus generating the pseudo-image of size (C, H, W) with H and W as the height and width[31].

B. Backbone Network

The backbone network here involves a 2D convolutional layer which is similar to the one from VoxelNet[79]. It consists of two parts, one is the top-down network for generating the features at increasingly small spatial resolution and the other performs the upsampling and concatenation of features from the first part of the network. The top-down network is built using a series of blocks $Block(S, L, F)$, where, S - stride, L - number of 3x3 2D convolution layers, and F - number of output channels. Each of these blocks is followed by batch normalization and ReLU functions[31].

The final features from each of the top-down network blocks are combined using upsampling and concatenation. The features are first upsampled from an initial stride to the final stride using the transposed 2D convolution operation with F final features and these are passed through batch normalization and ReLU functions. These are then concatenated to generate a final Feature map.

C. Detection Head

Here, a single shot detector (SSD)[40] based approach is incorporated for performing 3D detections on the final feature map. By utilizing 2D IOU, the ground truth boxes and the predicted boxes are matched and with this 2D match, the height and the elevation parameters become additional regression parameters[31].

Published Results

The published average precision on different classes from the literature [31] is provided in Table 5.1.

	$AP_{AOS}(\%)$			$AP_{BEV}(\%)$			$AP_{3D}(\%)$		
	Easy	*Moderate*	*Hard*	*Easy*	*Moderate*	*Hard*	*Easy*	*Moderate*	*Hard*
Car	90.19	88.76	86.38	88.35	86.10	79.83	79.05	74.99	68.30
Pedestrian	58.05	49.66	47.88	58.66	50.23	47.19	52.08	43.53	41.49
Cyclist	82.43	68.16	61.96	79.14	62.25	56.00	75.78	59.07	52.92

Table 5.1: Published results of PointPillars architecture referenced from [31]

5.5 Frustum ConvNet

In this section, the network architecture and the implementation details from [64] are discussed. Frustum ConvNet (F-ConvNet) proposes a novel approach for amodal 3D object detection on point clouds based on 2D region proposals. F-ConvNet basically aggregates the point-wise feature as frustum-level features and arrays them to form a feature map and fuses these features by utilizing the fully convolutional network to perform amodal 3D object detection and orientation regressions. Here, the pixels present in the 2D proposals are matched with the 3D points in the point clouds, and for each region proposals a sequence of frustums are generated by sliding along the frustum axis (as shown in Figure 5.11). These frustums holds the group of local points. The frustum axis refers to the optical axis of the image plane.

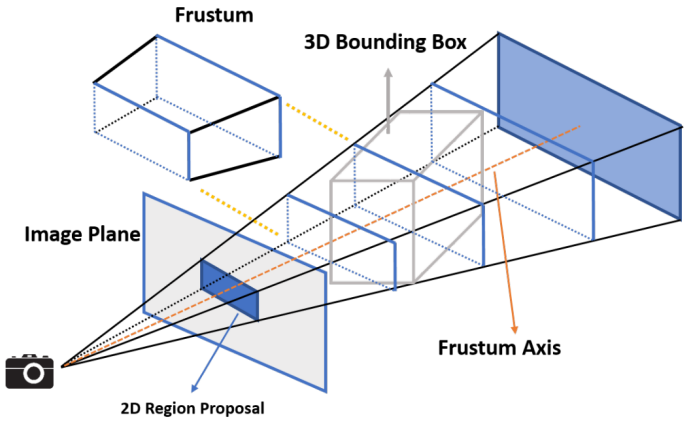


Figure 5.11: Generation of sequence of frustums by sliding along frustum axis[64, p. 1].

The major contributions of this method are, first, an amodal 3D detector that groups the point cloud based on sliding frustum for aggregating the point-wise features for use of fully convolutional network. Second, proposes different variants of F-ConvNets that include an FCN that extracts multi-resolution frustum features and refines using F-ConvNet on reduced 3D space. Finally, it is implemented on the assumption of no prior knowledge of a working 3D environment that makes it a dataset agnostic[64].

Network Architecture

Frustum ConvNet fuses the frustum-level features using fully convolutional layers by aggregating the point-wise features that are extracted using the 2D region proposals. The architecture is depicted in Figure 5.12.

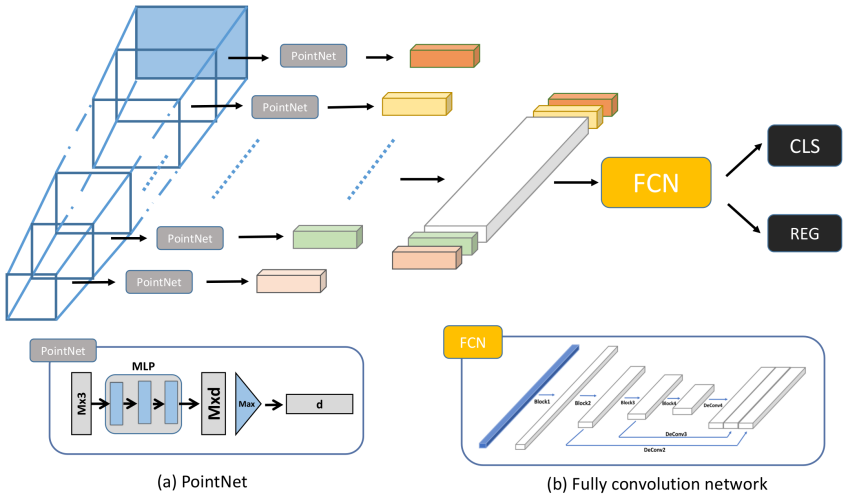


Figure 5.12: Architectural overview of F-ConvNet[64, p. 2] representing PointNet layers and FCN layers along with detection head.

The frustum-level features are extracted using the modified PointNet[48] that stacks three fully connected layers followed by a final layer for aggregating the features of each individual point as a frustum-level feature vector by performing element-wise max pooling operations. The fully convolutional network (FCN) consists of four convolution blocks and three deconvolution layers. Configuration details of the FCN layers are given in Figure 5.13.

Name	Kernel size/Filter no./Striding/Padding
Block1	3×128 / 128 / 1 / 1
Block2	3×128 / 128 / 2 / 1 3×128 / 128 / 1 / 1
Block3	3×128 / 256 / 2 / 1 3×256 / 256 / 1 / 1
Block4	3×256 / 512 / 2 / 1 3×512 / 512 / 1 / 1
Deconv2 Deconv3 Deconv4	1×128 / 256 / 1 / 0 2×256 / 256 / 2 / 0 4×512 / 256 / 4 / 0
Merge_conv2 Merge_conv3 Merge_conv4	1×256 / 128 / 1 / 0 1×512 / 256 / 1 / 0 1×1024 / 512 / 1 / 0

Figure 5.13: Configuration details of FCN layer of Frustum ConvNet referenced from [64].

Here, the ground truth is denoted as $\{x_c^g, y_c^g, z_c^g, l^g, w^g, h^g, \theta^g\}$, where, (x_c^g, y_c^g, z_c^g) are box center coordinates, (l^g, w^g, h^g) are three side lengths of bounding box and θ^g is the yaw angle. The anchor boxes are denoted as $\{x_c^a, y_c^a, z_c^a, l^a, w^a, h^a, \theta^a\}$, where, (x_c^a, y_c^a, z_c^a) are centroid of anchor box, (l^a, w^a, h^a) are three side lengths of anchor box and θ^a is the yaw angle[64]. With these the residual differences between the anchor boxes and the ground truth labels are calculated as in below equations (referenced from [64]).

$$\Delta x = x_c^g - x_c^a, \Delta y = y_c^g - y_c^a, \Delta z = z_c^g - z_c^a$$

$$\Delta w = \frac{w^g - w^a}{w^a}, \Delta l = \frac{l^g - l^a}{l^a}, \Delta h = \frac{h^g - h^a}{h^a}$$

$$\Delta \theta = \theta^g - \theta^a$$

Regression loss is based on the Euclidean distance, and the discrepancy in the sizes and the orientation is modeled using SmoothL1 regression loss. For regularizing the box regressions, a corner loss implemented by [47] is adapted. Together with the focal loss for classifications, the total loss is modeled using these three losses[64].

For optimizing the loss functions, ADAM optimizer with a weight decay of $1e^{-4}$ with starting learning rate of $1e^{-3}$ and decaying by a factor of 10 for every 20 epochs of total 50 epochs is implemented[64].

Network Implementation

The implementation of the method involves four stages, (1) Grouping of point cloud, (2) Frustum-level feature extraction, (3) Fully convolutional layers, and (4) Detection head.

A. Grouping of point cloud

The grouping of the sparse point clouds results in the generation of sequences of frustums along the same frustum axis. This method utilizes the RGB images associated with the point cloud and the 2D region proposals obtained from off-the-shelf 2D object detectors to generate a series of frustums. These sequences are generated by sliding a pair of parallel-planes along the same frustum axis with equal strides. These parallel-planes are perpendicular to the frustum axis. For each of the 2D region proposals, a sequence of frustums are generated using this method. All the points inside a frustum are grouped together and are categorized as foreground points.

B. Frustum-level Feature Extraction

Consider a 2D region proposal, a sequence of T frustums are generated by sliding along the frustum axis with a stride of s, and one of the frustum in the sequence, there are M points associated with it. To extract the point-wise features PointNets are applied to each frustum which results in T point-wise feature vectors[64].

C. Fully Convolutional Network (FCN)

The individual frustum level features from the previous stage (total T) are stacked up in the form of arrays to create a feature map of size $T \times d$. The FCN consists of layers of convolution and deconvolution blocks. Except for the first block, all other blocks of convolution blocks downsamples the 2D feature map,

these convolution operations and down-sampling results in fusing the frustum level features. The output of each convolution block is upsampled by a corresponding deconvolution block to a higher resolution and all the feature maps from the deconvolution blocks are concatenated to form a feature map of the frustum dimension. The individual feature maps of deconvolution blocks will be of different lengths, concatenating them results in estimating the 3D boxes for object instances that are unknown and have varying sizes.

The final feature map of the FCN layer is a reduced version by a power of 2 of the initial feature map from the PointNet layer. In refinement extension of F-ConvNet, on the same 2D region proposals a sequence of $T/2$ frustums can be generated using $2s$ strides. By applying PointNet on these frustums the resulting feature map will be of size $T/2 \times d$ and by doubling the height of frustums results in covering the same 3D space and its feature map being compatible with the corresponding FCN resulting in the feature map of size $T/2 \times 2d$. A final convolution layer is used for resizing this into original size which can be placed in FCN with no change to other FCN layers. This procedure is referred to as multi-resolution feature integration variant[64].

D. Detection Head

The final feature map from the FCN layer is provided as input to two parallel convolution layers, one for classification estimations and other for regression.

Published Results

The published average precision on different classes from the literature [64] is provided in Table 5.2.

	$AP_{BEV}(\%)$			$AP_{3D}(\%)$		
	Easy	*Moderate*	*Hard*	*Easy*	*Moderate*	*Hard*
Car	89.69	83.08	74.56	85.88	76.51	68.08
Pedestrian	58.90	50.48	46.72	52.37	45.61	41.49
Cyclist	82.59	68.62	60.62	79.58	64.68	57.03

Table 5.2: Published results of Frustum ConvNet architecture referenced from [64]

6

Experiments and Results

To perform the comparative analysis, along with describing various aspects that support the comparison of two different models and their performances based on sensors, approaches, techniques, preprocessing, and so on theoretically, to support the theoretical argument, a few experiments are conducted. In this chapter, the details on the experimental setup, experiments conducted, and their results are discussed.

6.1 Experimental Setup

The deep neural network models chosen for performing experiments are available for research purposes on GitHub (PointPillars and F-ConvNet). These network models require large datasets for training and validating, and these datasets are downloaded from KITTI official website and WAYMO official website. To train, validate, and test these deep neural network models high performance computing systems are required. For conducting the experiments the scientific computing platform called cluster machines is utilized.

The cluster machines are equipped with 80 GB of primary storage space and 5 TB of secondary storage space per user and are also limited based on the total number of files stored. The cluster machine consists of several nodes to which a batch job is submitted through which the training and validations are performed. For performing these experiments, we require the cluster nodes that have GPU processor and huge RAM for faster computations. These cluster nodes are equipped with 192 GB of RAM, Intel Xeon Gold 6130 (Skylake EP) processor with 16 cores, and Nvidia Tesla V100 GPU (detailed information is available at [2]).

The downloaded KITTI dataset is divided into training and testing folder that consists of information related to 7481 training sets and 7518 testing sets respectively. The training folder consists of four subfolders, namely, 'calib' - consists of calibration information for each frame in text files, 'image_2' - consists of images from one of the four cameras in PNG format (generally it is front left camera), 'label_2' - consists of label information for each image in text files, and 'velodyne' - consists of laser scans in the form of floating binaries (.bin). The testing folder consists of a test set in a similar folder structure except for the 'label_2' folder. All images will not be considered for training and validations, only the images with corresponding label files are considered.

The total size of the KITTI dataset including training and testing folder is around 50 GB. Along with this, the two deep learning models generate around 10 GB to 15 GB of data including the input pickle

files, storing preprocessed input data, logging information, and storing results. Due to this, only one experiment was executed using primary storage at a time.

The WAYMO Open dataset is in TFRecord format and they are not compatible to be used with the selected frameworks directly as compared to KITTI. The information from these TFRecord files is extracted into the KITTI format and placed in two folders as training and testing. A few modifications in the codebase needs to be performed with respect to camera calibrations of reference camera for computing. The total size of the WAYMO dataset is around 1 TB in a compressed state, one compressed file of the dataset is of 25 GB to 30 GB of TFRecord files. Extracting information from these files requires larger data space, due to this a random subset of the complete dataset is utilized.

Since PointPillars is based on only LiDAR data, in detection visualization provided in further sections, the images represent the ground-truth boxes and its corresponding point cloud shows detection results. More details on different configurations for each of the deep learning frameworks and modifications in the codebase for the WAYMO dataset are provided in further sections.

6.2 Experiments

For observing the impact on the performance of 3D object detection frameworks by using only LiDAR data over sensor fusion techniques various experiments were designed and conducted, for instance, to check how good the object detectors are at handling occlusions, truncations, change in range of LiDAR data, and so on. In this section, a description of various experiments and their results are discussed along with the quantitative analysis of the results of each experiment. The qualitative analysis is provided in Section 6.3.

6.2.1 Performance Impact With Complete KITTI Dataset

In this experiment, we evaluate the performance of the PointPillars framework that performs 3D object detection based only on LiDAR data and F-ConvNet framework that performs 3D object detection based on sensor fusion technique, for detection of object classes 'Car', 'Pedestrian', and 'Cyclist'. As described in the literature ([31], [64]), two different models are trained for each of the framework, one model for detection of object class 'Car' and other model for detection of object classes 'Pedestrian', and 'Cyclist'. Here, the complete training set of KITTI dataset is divided into 3712 training samples and 3769 validation samples. The main idea of this experiment is to observe the impact of mixed data (that includes, occluded objects, various driving conditions, and so on) on 3D object detection with LiDAR data and to check how much of performance improvement is seen by employing sensor fusion techniques. Below, the configuration details for each framework is explained.

PointPillars

Here, the maximum number of pillars or vertical columns is set to 12000 with a maximum of 100 points per pillar. For modeling of the loss functions, anchor boxes and the ground truth boxes are matched using 2D IOU with two thresholds, if the IOU ratio is greater than the matched threshold then it is categorized

as positive and if the IOU ratio is lesser than the unmatched threshold then it is categorized as negative, and the ones that are in between these two thresholds are neglected. At inferencing time, the NMS IOU threshold of 0.5 is configured[31].

The feature encoder network is configured with 64 filters. The RPN is configured with number of layers as $[3, 5, 5]$ with layer strides of $[2, 2, 2]$ for object class 'Car' and $[1, 2, 2]$ for object classes 'Pedestrian', and 'Cyclist', along with number of filters $[64, 228, 256]$ with upsample strides of $[1, 2, 4]$, and upsample filters of $[128, 128, 128]$. These configurations are the same for both models.

For loss function optimization, ADAM optimizer with an initial learning rate of $2e^{-4}$ and decaying it by a factor of 0.8 for every 15 epochs for over 160 epochs with a batch size of 2 on the validation set is configured[31].

For the detection of 'Car' object class, the network is configured with the LiDAR range as $(0, 70)$, $(-40, 40)$, and $(-3, 1)$ in meters along (x, y, z) direction along with anchor sizes of height, length, and width as $[1.56, 3.9, 1.6]$ in meters with anchor strides as $[0.32, 0.32, 0.0]$ with a z center as -1 meters. For anchor matching the matched threshold of 0.6 and unmatched threshold of 0.45 is configured. For the detection of 'Pedestrian' and 'Cyclist' object classes, the network is configured with the LiDAR range as $(0, 48)$, $(-20, 20)$, and $(-2.5, 0.5)$ in meters along (x, y, z) direction along with anchor sizes of height, length, and width as $[1.73, 0.8, 0.6]$ in meters for 'Pedestrian' class and $[1.73, 1.76, 0.6]$ in meters for 'Cyclist' class with anchor strides as $[0.16, 0.16, 0.0]$ with a z center as -0.6 meters. For anchor matching the matched threshold of 0.5 and unmatched threshold of 0.35 is configured.

F-ConvNet

For generating 2D region proposals FPN[38] based framework is used which is the same as the one used by [47]. These proposals are augmented by performing scaling and translating during the training and for each of the region proposal the point cloud is randomly sampled and is limited to 1024 points for the first stage and for the final refinement stage, it is limited to 512 points. For categorizing the points into foreground and background, the size of the ground truth boxes are shrunk by a factor of 0.5 and if the center of anchor box falls inside shrunk ground truth boxes then these are categorized as foreground points and the other points as background points for creating the positive and negative examples[64].

For optimizing the loss function, the ADAM optimizer with a weight decay of $1e^{-4}$ with starting learning rate of $1e^{-3}$ and decaying by a factor of 10 for every 20 epochs of total 50 epochs is implemented. For the detection of all three classes of objects, the maximum depth of 70 meters is configured.

For the detection of 'Car' object class, for the first stage, each frustum is configured with four resolutions with strides of $[0.25, 0.5, 1.0, 2.0]$ and the height resolution as $[0.5, 1.0, 2.0, 4.0]$ with a number of points per frustum as 1024 and a total number of frustums in a sequence as 280. For the refinement stage, each frustum is configured with four resolutions with strides of $[0.1, 0.2, 0.4, 0.8]$ and the height resolution as $[0.2, 0.4, 0.8, 1.6]$ with a number of points per frustum as 512 and a total number of frustums in a sequence as 140. For the detection of 'Pedestrian' and 'Cyclist' object classes, for the first stage, each frustum is configured with four resolutions with strides of $[0.1, 0.2, 0.4, 0.8]$ and the height resolution as $[0.2, 0.4, 0.8, 1.6]$ with a number of points per frustum as 1024 and a total number of frustums in a

sequence as 700. For the refinement stage, each frustum is configured with four resolutions with strides of $[0.05, 0.1, 0.2, 0.4]$ and the height resolution as $[0.1, 0.2, 0.4, 0.8]$ with a number of points per frustum as 512 and a total number of frustums in a sequence as 350.

The training and validation times with the experimental setup (Section 6.1), for PointPillars it was approximately around 21 hours and for F-ConvNet it was approximately around 8 to 9 hours. The quantitative results on performance on validation set is given in Table 6.1, the variation in the loss for classification, localization, and orientation curves for both frameworks are given in Figure 6.2 and Figure 6.3, and detection results are visualized in Figure 6.1 and Figure 6.5, along with few failure cases visualized in Figure 6.4 and Figure 6.6.

		Frustum ConvNet			**PointPillars**		
		Easy	*Moderate*	*Hard*	*Easy*	*Moderate*	*Hard*
Car	$AP_{AOS}(\%)$	**98.13**	**90.04**	**87.66**	90.63	88.64	86.83
	$AP_{BEV}(\%)$	**90.36**	**88.84**	80.13	90.02	87.49	**85.48**
	$AP_{3D}(\%)$	**89.25**	**79.05**	**77.15**	86.37	76.84	71.73
Pedestrian	$AP_{AOS}(\%)$	**69.31**	**63.06**	**56.30**	36.35	35.00	33.94
	$AP_{BEV}(\%)$	68.55	60.34	56.45	**74.05**	**68.73**	**63.64**
	$AP_{3D}(\%)$	61.63	53.50	48.39	**67.93**	**61.21**	**56.80**
Cyclist	$AP_{AOS}(\%)$	**86.34**	**71.17**	**69.14**	84.73	64.77	61.88
	$AP_{BEV}(\%)$	**87.97**	**70.83**	**66.78**	82.94	62.54	59.38
	$AP_{3D}(\%)$	**85.88**	**68.24**	**63.07**	79.78	59.46	56.39

Table 6.1: Detection results on KITTI validation set for Frustum ConvNet and PointPillars, showing the performance on training and validating on full training dataset.

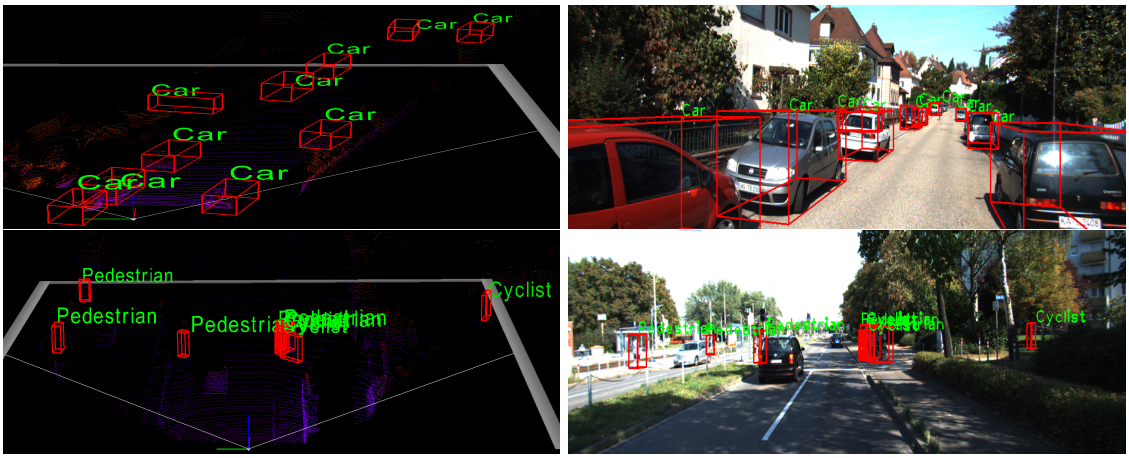
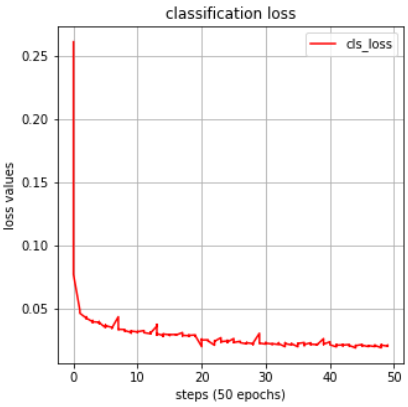
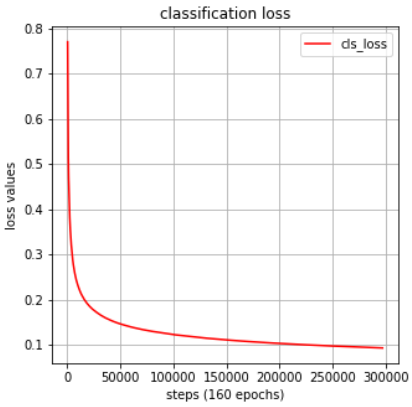


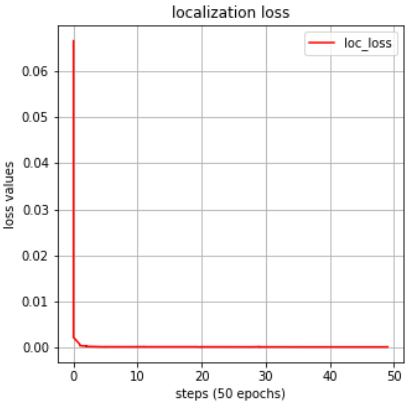
Figure 6.1: Detection results for Frustum ConvNet with IOU = 0.7 for 'Car' and IOU = 0.5 for 'Pedestrian/Cyclist'.



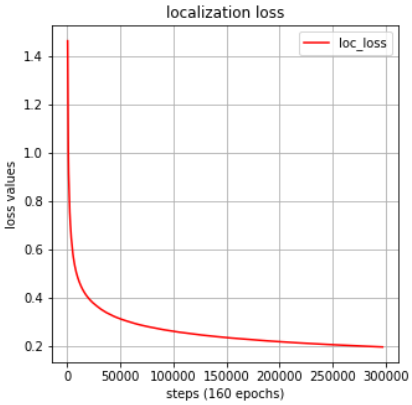
(a) classification loss for Frustum ConvNet



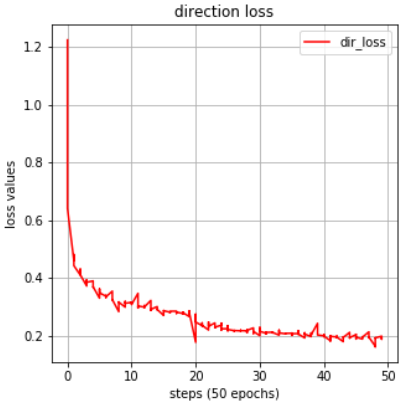
(b) classification loss for PointPillars



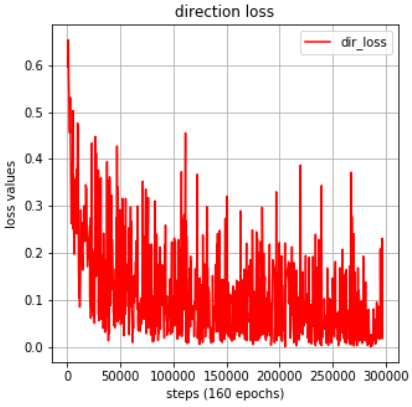
(c) localization loss for Frustum ConvNet



(d) localization loss for PointPillars

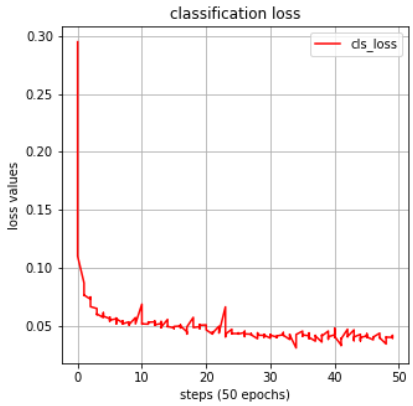


(e) Direction loss for Frustum ConvNet

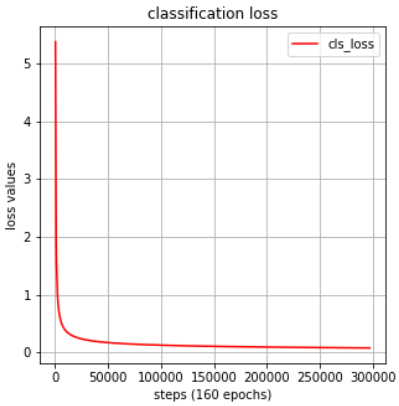


(f) Direction loss for PointPillars

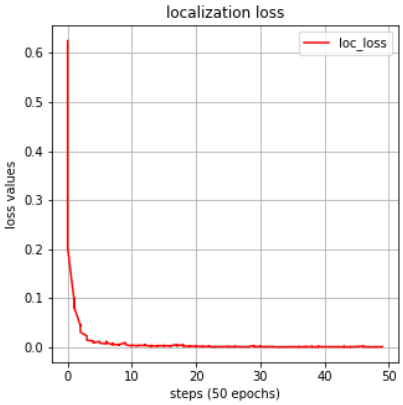
Figure 6.2: Variations of losses for both the models detecting object class 'Car'.



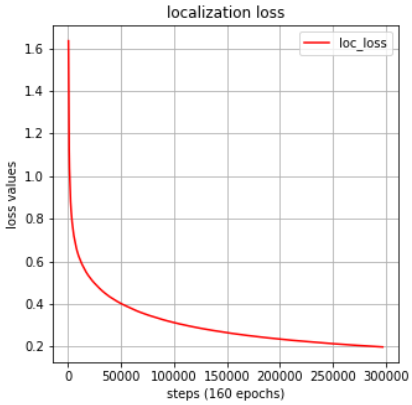
(a) classification loss for Frustum ConvNet



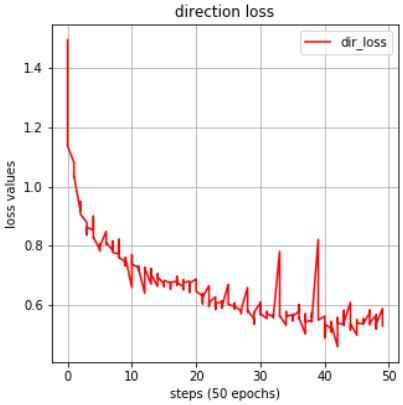
(b) classification loss for PointPillars



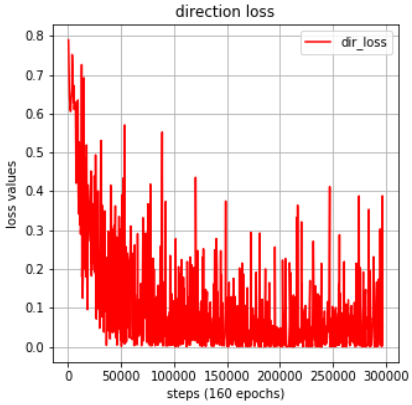
(c) localization loss for Frustum ConvNet



(d) localization loss for PointPillars



(e) Direction loss for Frustum ConvNet



(f) Direction loss for PointPillars

Figure 6.3: Variations of losses for both the models detecting object classes 'Pedestrian/Cyclist'

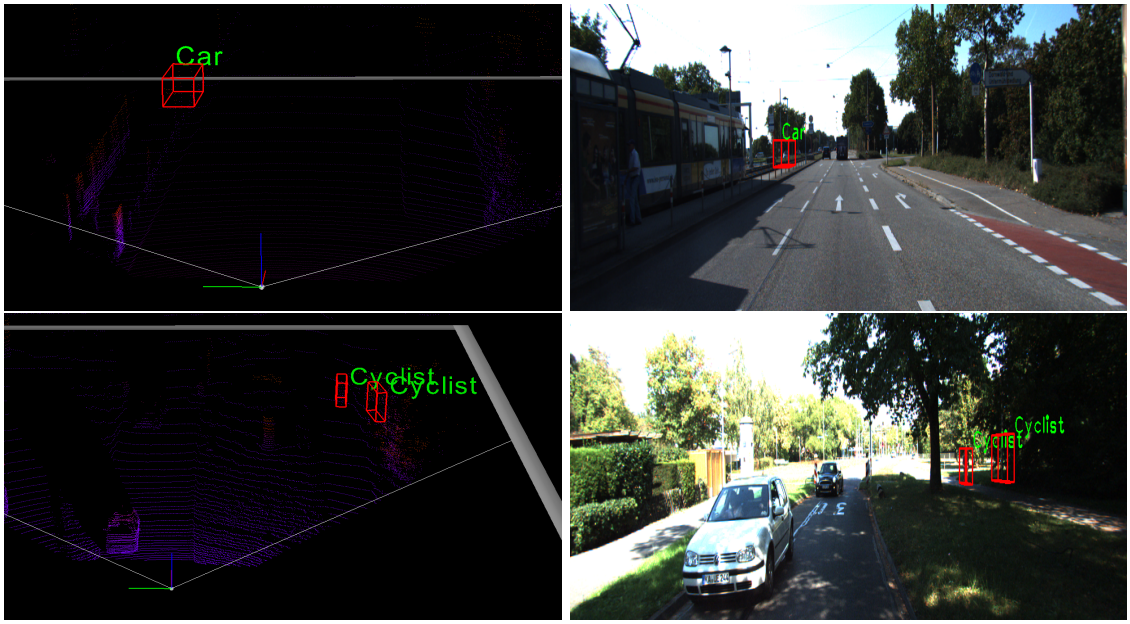


Figure 6.4: Miss-Detection (Failure cases) results for Frustum ConvNet with IOU = 0.7 for 'Car' and IOU = 0.5 for 'Pedestrian/Cyclist'.

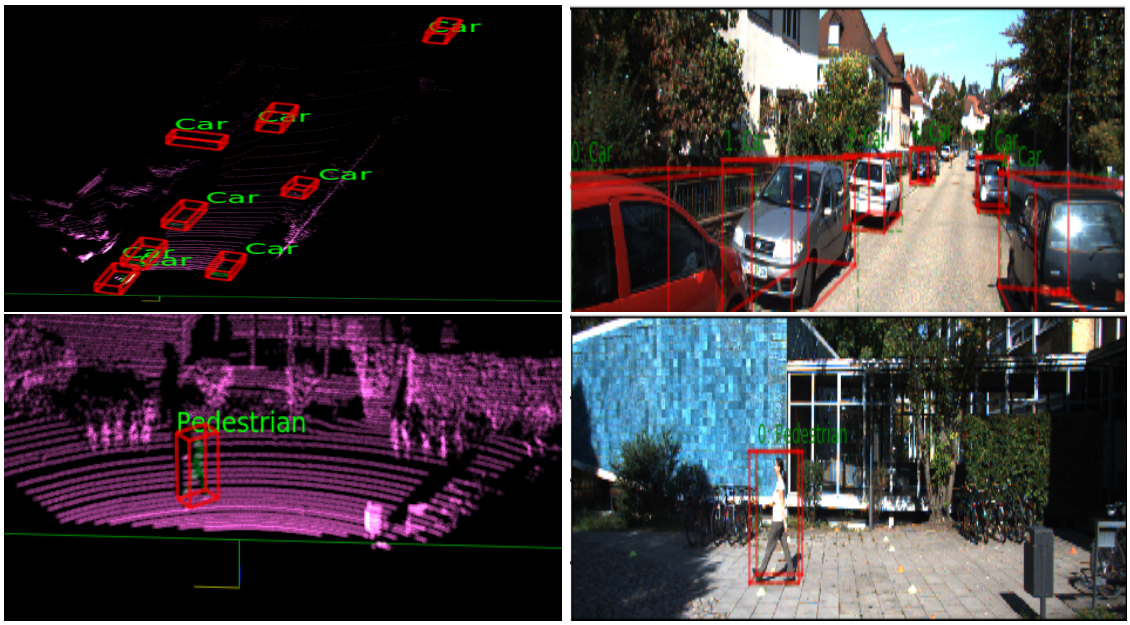


Figure 6.5: Detection results for PointPillars with IOU = 0.6 for 'Car' and IOU = 0.45 for 'Pedestrian/-Cyclist'.

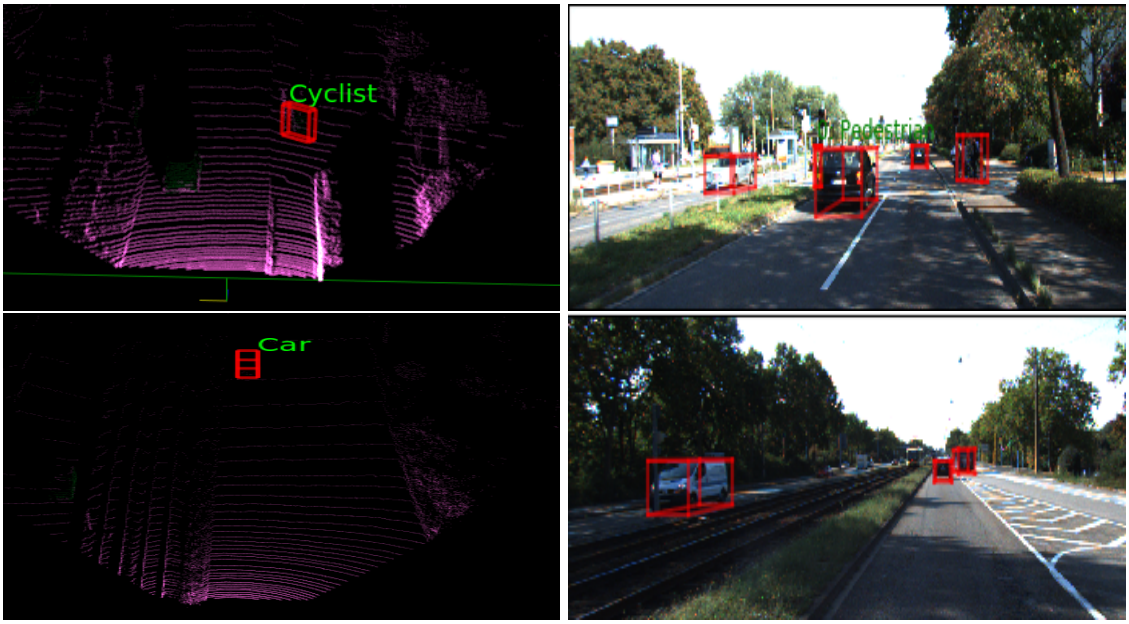


Figure 6.6: Miss-Detection (Failure cases) results for PointPillars with IOU = 0.6 for 'Car' and IOU = 0.45 for 'Pedestrian/Cyclist'

6.2.2 Performance For Handling Occluded Objects

Tackling of occluded objects is one of the main subtasks for object tracking. For tracking, first, the objects need to be detected and classified. For this purpose, the capability of the frameworks in handling the occluded objects is examined.

In this experiment, we evaluate the performance of the two object detection frameworks for the handling of the occluded objects. As discussed in Section 5.1, the KITTI dataset provides four levels of occlusion information for an object depending upon the percentage of occlusion. Level-0 is for non-occluded objects, Level-1 is for partially occluded objects, level-2 is for Highly occluded objects, and Level-3 is for unknown objects. The main idea here is to check the performance of the LiDAR framework on the handling of occluded objects and how much of the performance is improved by the use of sensor fusion frameworks.

For this experiment, all the configurations mentioned in Experiment 6.2.1 remains the same. Here, all four network models are trained with 3712 training samples that include all type of objects (occluded and non-occluded) and are validated only on the subset of the validation samples that consists of only occluded objects from Level-1 to Level-3. Here, the training and validation times vary a little compared to the previous experiment. For PointPillars it took around 18 hours approximately and for F-ConvNet it took around 6-7 hours approximately.

The quantitative analysis of object detection evaluation metrics is provided in Table 6.2, and visualization of detection results are provided in Figure 6.7 and Figure 6.8.

		Frustum ConvNet			PointPillars		
		Easy	*Moderate*	*Hard*	*Easy*	*Moderate*	*Hard*
Car	$AP_{AOS}(\%)$	**98.07**	**90.07**	**87.57**	90.58	88.40	86.67
	$AP_{BEV}(\%)$	**90.43**	**88.90**	80.10	90.14	87.38	**85.07**
	$AP_{3D}(\%)$	**89.20**	**78.79**	**76.48**	86.65	76.54	69.91
Pedestrian	$AP_{AOS}(\%)$	**67.66**	**61.06**	**54.27**	42.64	42.00	39.97
	$AP_{BEV}(\%)$	69.38	61.17	56.71	**71.63**	**66.31**	**60.27**
	$AP_{3D}(\%)$	62.75	56.27	49.24	**64.28**	**58.62**	**52.46**
Cyclist	$AP_{AOS}(\%)$	**88.08**	**72.33**	**70.05**	84.11	65.57	61.70
	$AP_{BEV}(\%)$	**88.83**	**70.40**	**65.00**	83.87	64.61	60.39
	$AP_{3D}(\%)$	**88.30**	**68.98**	**63.90**	80.61	60.55	55.99

Table 6.2: Detection results on occlusion subset of KITTI validation set for handling of occlusion with Frustum ConvNet and PointPillars.

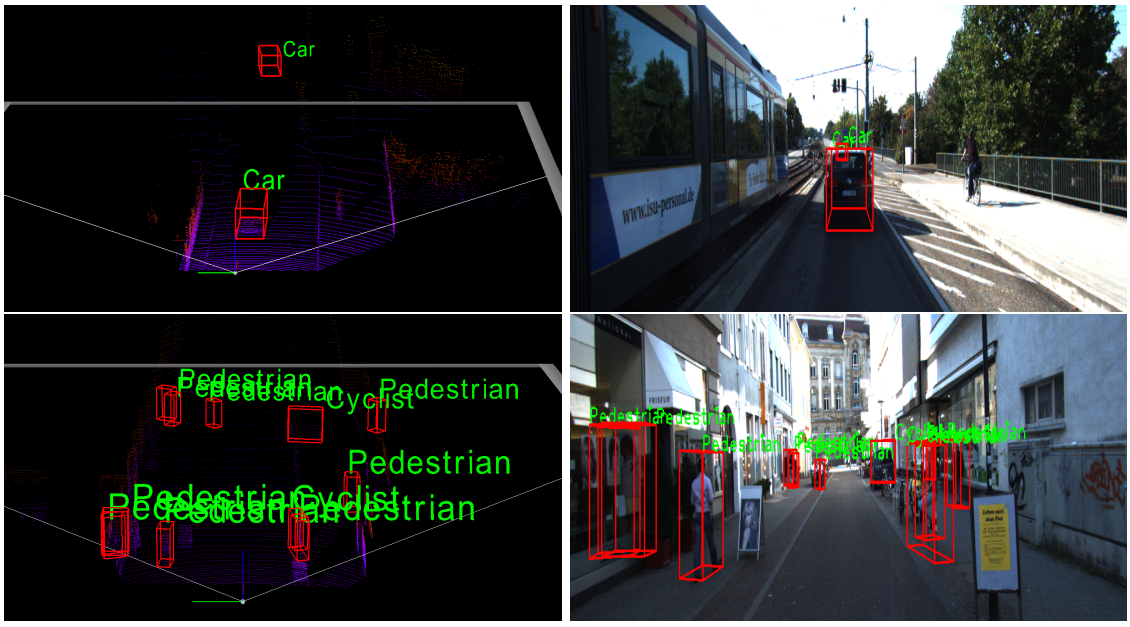


Figure 6.7: Qualitative result analysis on handling occluded objects for Frustum ConvNet.

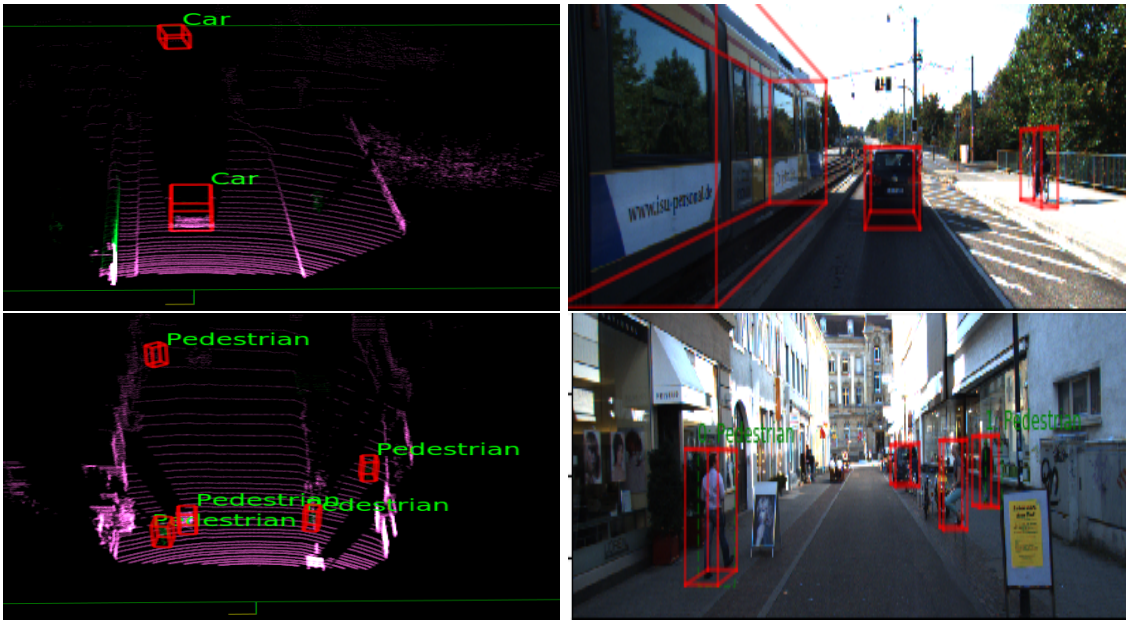


Figure 6.8: Qualitative result analysis on handling occluded objects for PointPillars.

6.2.3 Performance on Truncation Handling

When the vehicle is moving, from one frame to another frame there will be many objects that are partially visible, sometimes the vehicles or objects (pedestrians or cyclists) that are in front left side or front right side of the ego vehicle are probably partially visible and poses a high risk for collision. Keeping track of these objects plays a major role in collision avoidance tasks and in lane changing scenarios.

These objects that are partially visible are categorized as truncated objects by the KITTI dataset. As discussed in Section 5.1, based on the percentage of visibility, the values of truncation ranges from 0 to 1 and if the value is 2, then these are ignored by manual labeling. The main objective of this experiment is to evaluate the performance of the detection of truncated objects. Here, all four models are trained with 3712 training samples that include all type of objects (all levels of truncations) and are validated only on the subset of the validation samples that consists of only truncated objects. The training and validation times are similar to the previous experiment.

The quantitative analysis of object detection evaluation metrics is provided in Table 6.3, and visualization of detection results are provided in Figure 6.9 and Figure 6.10.

		Frustum ConvNet			PointPillars		
		Easy	*Moderate*	*Hard*	*Easy*	*Moderate*	*Hard*
Car	$AP_{AOS}(\%)$	**97.94**	**89.87**	87.12	90.49	89.07	**87.14**
	$AP_{BEV}(\%)$	**97.29**	**89.15**	80.05	90.52	89.00	**87.25**
	$AP_{3D}(\%)$	**89.54**	**85.41**	**77.39**	87.67	78.41	75.33
Pedestrian	$AP_{AOS}(\%)$	**69.98**	**63.92**	**59.90**	38.89	38.39	35.91
	$AP_{BEV}(\%)$	**69.93**	**66.18**	**59.22**	66.03	63.29	57.92
	$AP_{3D}(\%)$	**66.34**	**59.41**	**51.69**	56.36	53.68	48.37
Cyclist	$AP_{AOS}(\%)$	**88.05**	**67.93**	**66.78**	80.30	62.36	58.50
	$AP_{BEV}(\%)$	**93.63**	**68.34**	**62.89**	80.97	61.18	57.32
	$AP_{3D}(\%)$	**86.66**	**63.95**	**58.71**	78.33	58.61	54.85

Table 6.3: Detection results on truncation subset of KITTI validation set for handling of truncation with Frustum ConvNet and Pointpillars.

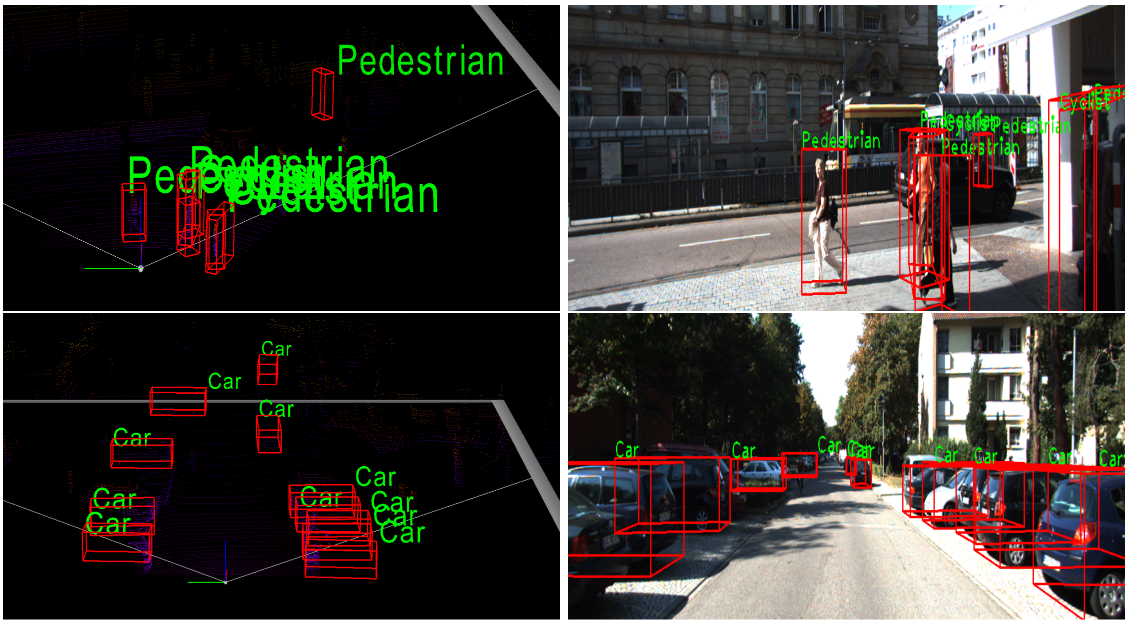


Figure 6.9: Qualitative result analysis on handling truncated objects for Frustum ConvNet.

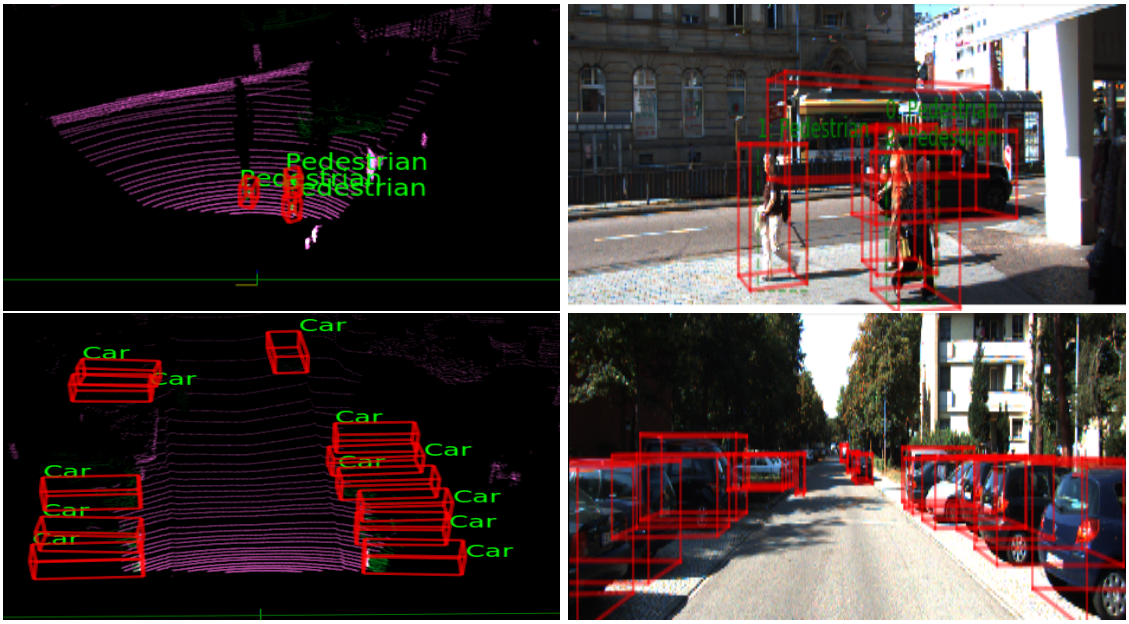


Figure 6.10: Qualitative result analysis on handling truncated objects for PointPillars.

6.2.4 Effect of LiDAR Range on Detection Performance

One of the adverse properties of the LiDAR sensor is that the point cloud becomes more sparse as the range increases. This increase in sparsity directly impacts on the detection performance of the frameworks. Generally, most of the published frameworks that use LiDAR data as one of the sources or only source for object detection, limits the range of the point cloud to 70 meters. The detection performance up to 70 meters range provides good performance and in most of the frameworks that uses only LiDAR for the detection of large objects such as 'Car', 'Truck', the point cloud range is limited to 70 meters and for smaller objects such as 'Pedestrian', 'Cyclist', the point cloud range is limited to around 50 meters.

As seen from the previous experiments, the performance of detection of small objects ('Pedestrian', 'Cyclist') by PointPillars shows similar results with respect to AP_{BEV} and AP_{3D} at a depth of 48 meters when compared to Frustum ConvNet at a depth of 70 meters, in this experiment we mainly concentrate on the detection of object class 'Car' at a higher depth of 100 meters and 120 meters. The main objective here is to observe the effect of change in LiDAR range on LiDAR based framework when compared to the sensor fusion based framework.

In this experiment, for the detection of object class 'Car', the range of LiDAR is increased from 70 meters to 100 meters and then from 100 meters to 120 meters. Because of the region proposals from the 2D images, Frustum ConvNet shows better results compared to PointPillars. Because of the 2D region proposals, the points belonged to objects that are at a greater range are also grouped and processed, and their 3D estimations are obtained, unlike in PointPillars, due to the sparsity, many proposals are missed out and objects are not estimated.

Few sample detections are provided in Figure 6.11 and Figure 6.13 for Frustum ConvNet and Figure 6.12 and Figure 6.14 for PointPillars, and the performance metrics are provided in Table 6.4.

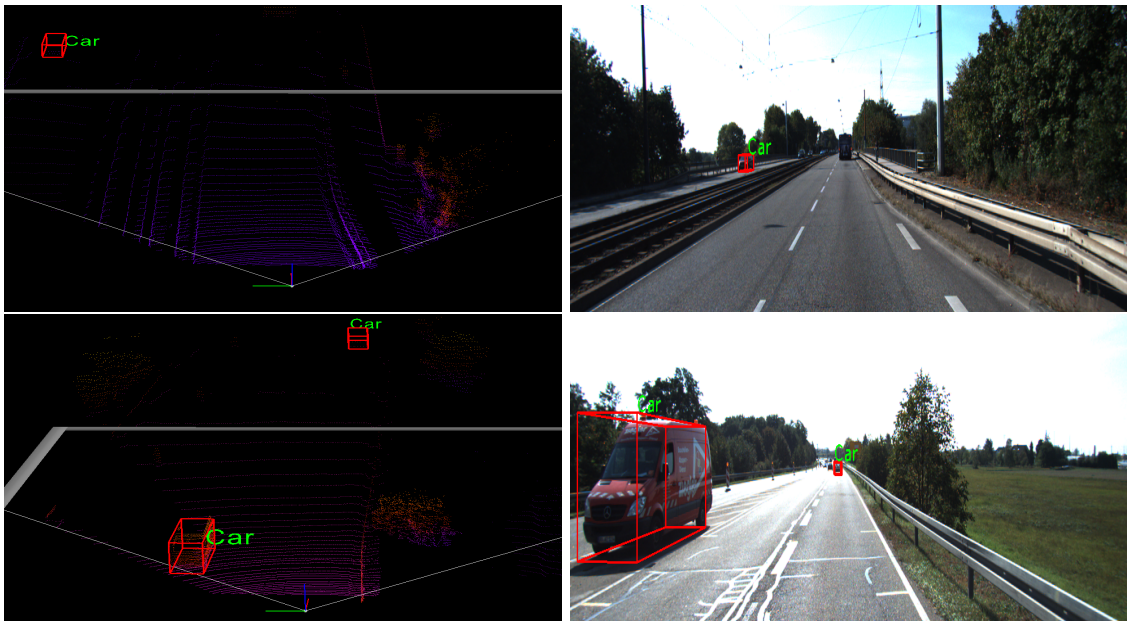


Figure 6.11: Qualitative result analysis for object class 'Car' at a depth of 0-100 meters for Frustum ConvNet.

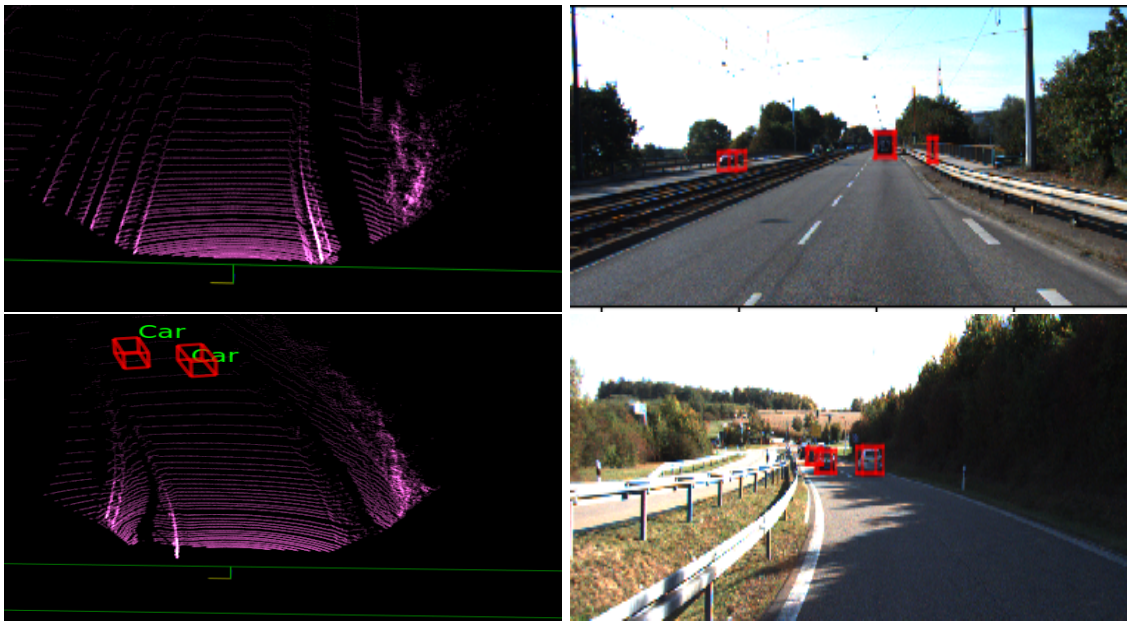


Figure 6.12: Qualitative result analysis for object class 'Car' at a depth of 0-100 meters for PointPillars.

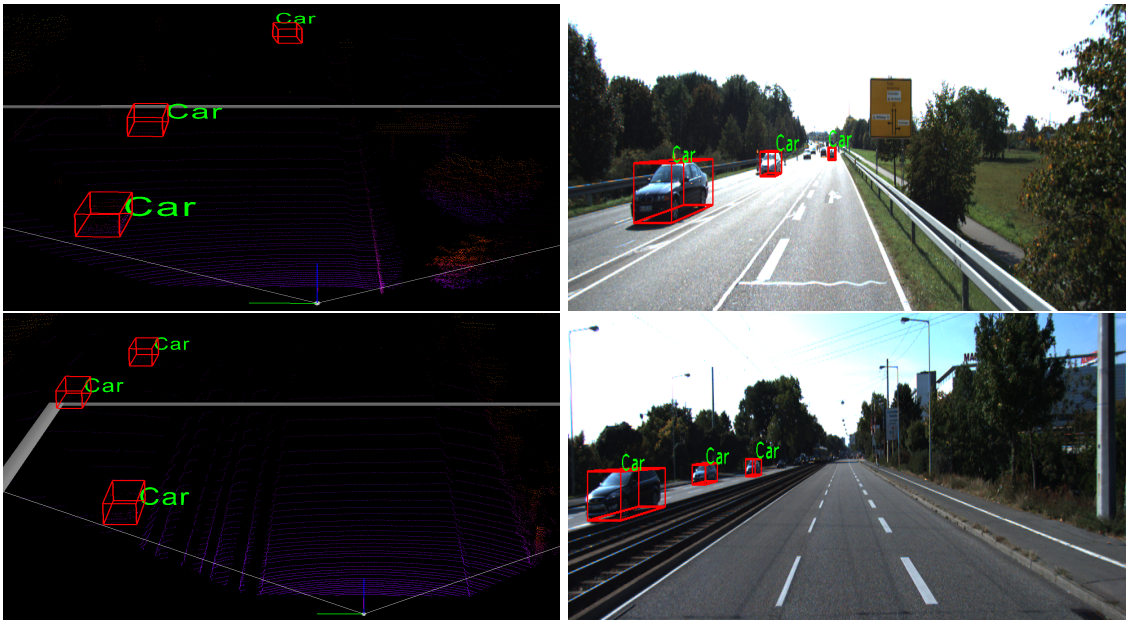


Figure 6.13: Qualitative result analysis for object class 'Car' at a depth of 0-120 meters for Frustum ConvNet.

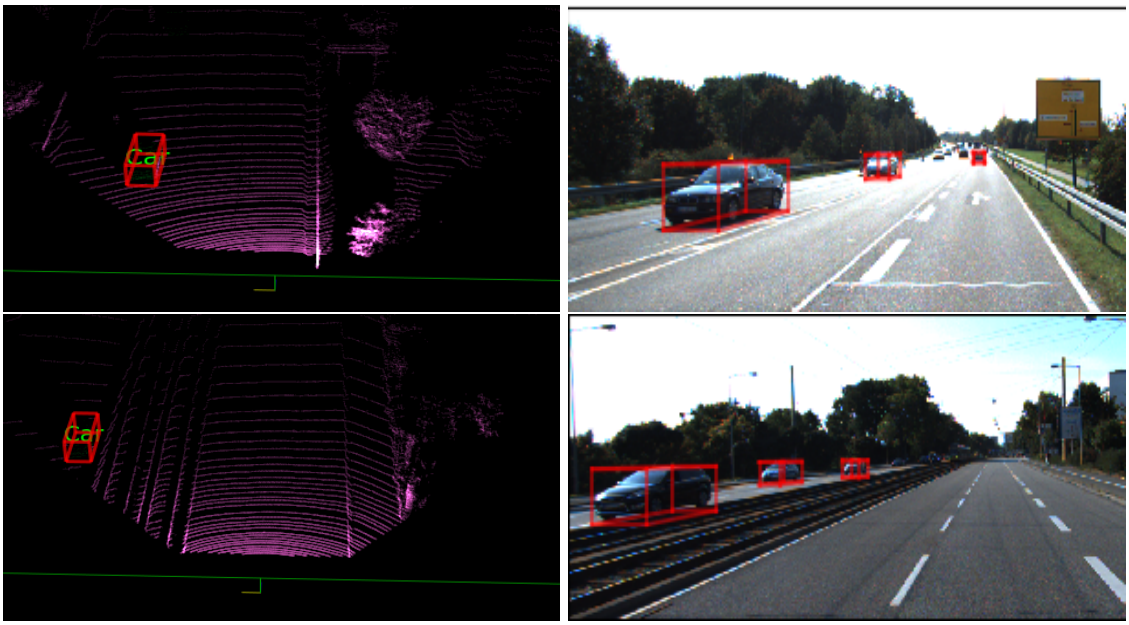


Figure 6.14: Qualitative result analysis for object class 'Car' at a depth of 0-120 meters for PointPillars.

		Frustum ConvNet ('Car')			**PointPillars ('Car')**		
		Easy	*Moderate*	*Hard*	*Easy*	*Moderate*	*Hard*
Range 0-100 meters	$AP_{AOS}(\%)$	**98.29**	**90.00**	**87.62**	89.60	75.78	74.08
	$AP_{BEV}(\%)$	**90.34**	**88.79**	**80.07**	89.51	71.11	69.71
	$AP_{3D}(\%)$	**89.16**	**78.66**	**76.30**	73.20	51.55	48.65
Range 0-120 meters	$AP_{AOS}(\%)$	**98.26**	**90.03**	**87.73**	87.78	61.13	60.11
	$AP_{BEV}(\%)$	**90.28**	**88.69**	**80.00**	82.79	54.37	53.25
	$AP_{3D}(\%)$	**89.37**	**78.76**	**76.20**	65.67	42.79	41.24

Table 6.4: Detection results for object class 'Car' at different depths of LiDAR data in KITTI dataset.

6.2.5 Effect of Class Imbalance on Frameworks

For classification tasks, equal distribution of classes is important and the imbalance in the distribution of classes forms the most influential factor on the performance of the frameworks. Keeping this in mind, this experiment is designed to observe the effect of a dominating class of objects over other classes. All the previous experiments consist of two models for each framework, one for estimating the 3D coefficients of object class 'Car' and other model for estimating the 3D coefficients of object classes 'Pedestrian' and 'Cyclist'.

In the KITTI dataset, object class 'Car' dominates other object classes. The total number of objects belongs to object class 'Car' is 14357, with 'Pedestrian' count of 2207, and 'Cyclist' count of 734 along with other objects. This imbalance in the distribution of classes affects the performance of a classifier or object detection framework when trained using a single model. In this experiment, the frameworks are directly targeted for observing the impact of class imbalance.

Here, one single model per detection network is trained and validated on a complete dataset for all three object classes. With the same configurations as mentioned in Experiment 6.2.1, two models, one for each network is created and this helps in analyzing the effect of imbalance in the distribution of classes in the training set. The main objective here is to create a single model that is capable of handling the effects of imbalance of class distributions and learn to detect and classify the various classes of objects.

Due to the creation of a single model for all three classes of objects, the training and validation time is slightly increased and it is approximately 25 to 28 hours for the PointPillars framework and 12 to 15 hours for the Frustum ConvNet framework. Below images provides information on variation of loss functions (Figure 6.16), visualizations of sample detection results are shown in Figure 6.15 and Figure 6.17 and average precision metrics are shown in Table 6.5.

		Frustum ConvNet			PointPillars		
		Easy	*Moderate*	*Hard*	*Easy*	*Moderate*	*Hard*
Car	$AP_{AOS}(\%)$	52.78	51.44	44.82	**90.04**	**87.22**	**84.46**
	$AP_{BEV}(\%)$	52.38	44.82	44.38	**89.19**	**86.27**	**80.97**
	$AP_{3D}(\%)$	44.84	43.67	42.51	**77.99**	**68.09**	**65.89**
Pedestrian	$AP_{AOS}(\%)$	**36.83**	**31.96**	**29.82**	1.20	0.86	0.84
	$AP_{BEV}(\%)$	**35.73**	**29.78**	**28.60**	0.42	0.38	0.38
	$AP_{3D}(\%)$	**34.08**	**28.51**	**26.83**	0.26	0.34	0.34
Cyclist	$AP_{AOS}(\%)$	**38.74**	**36.16**	**34.69**	0.39	14.63	13.93
	$AP_{BEV}(\%)$	**42.18**	**37.55**	**35.35**	0.07	13.37	12.69
	$AP_{3D}(\%)$	**41.79**	**36.32**	**34.44**	0.04	12.22	11.89

Table 6.5: Detection result showing the effect of class imbalance on Frustum ConvNet and PointPillars.

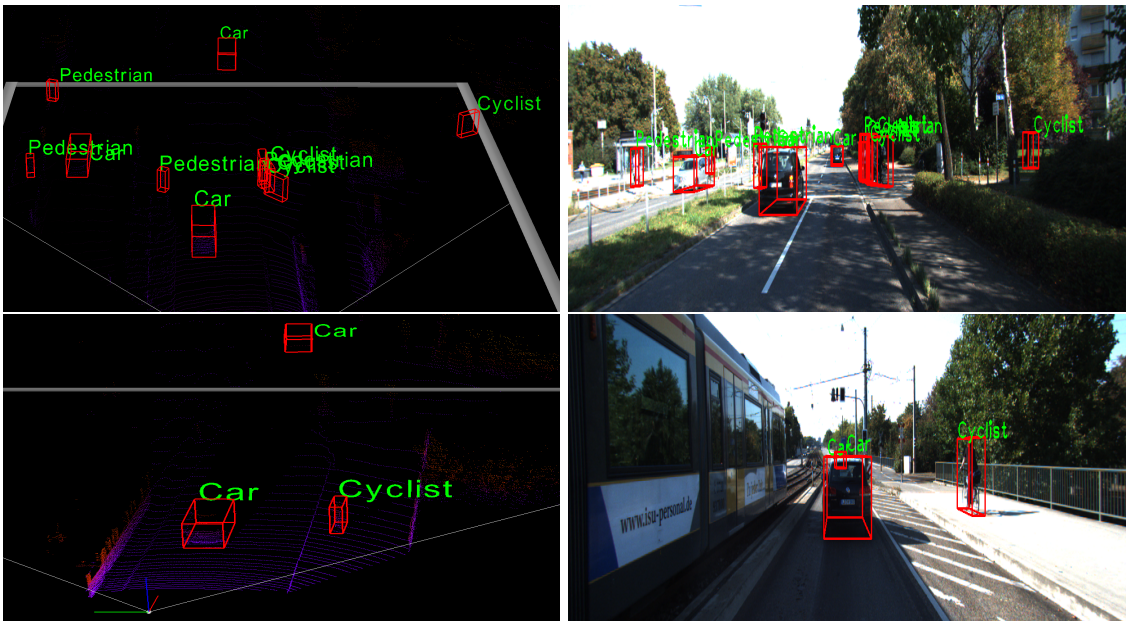
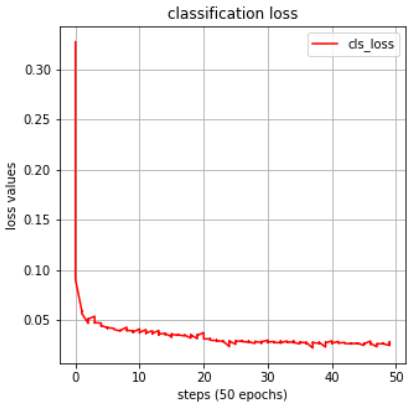
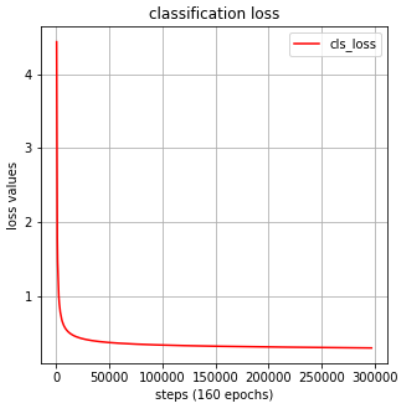


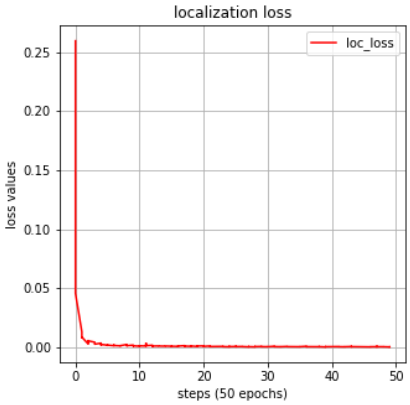
Figure 6.15: Qualitative result analysis for Frustum ConvNet depicting the imbalance of classes.



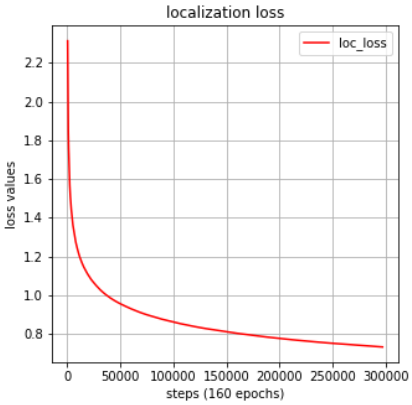
(a) classification loss for Frustum ConvNet



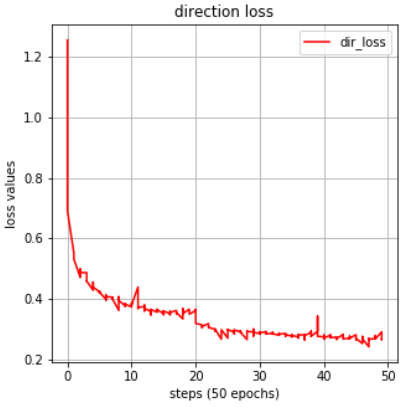
(b) classification loss for PointPillars



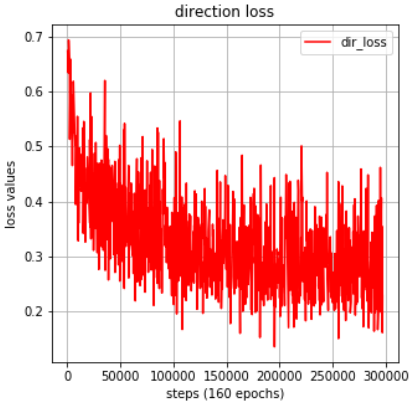
(c) localization loss for Frustum ConvNet



(d) localization loss for PointPillars



(e) Direction loss for Frustum ConvNet



(f) Direction loss for PointPillars

Figure 6.16: Variations of losses for both the models depicting the effect of class imbalance

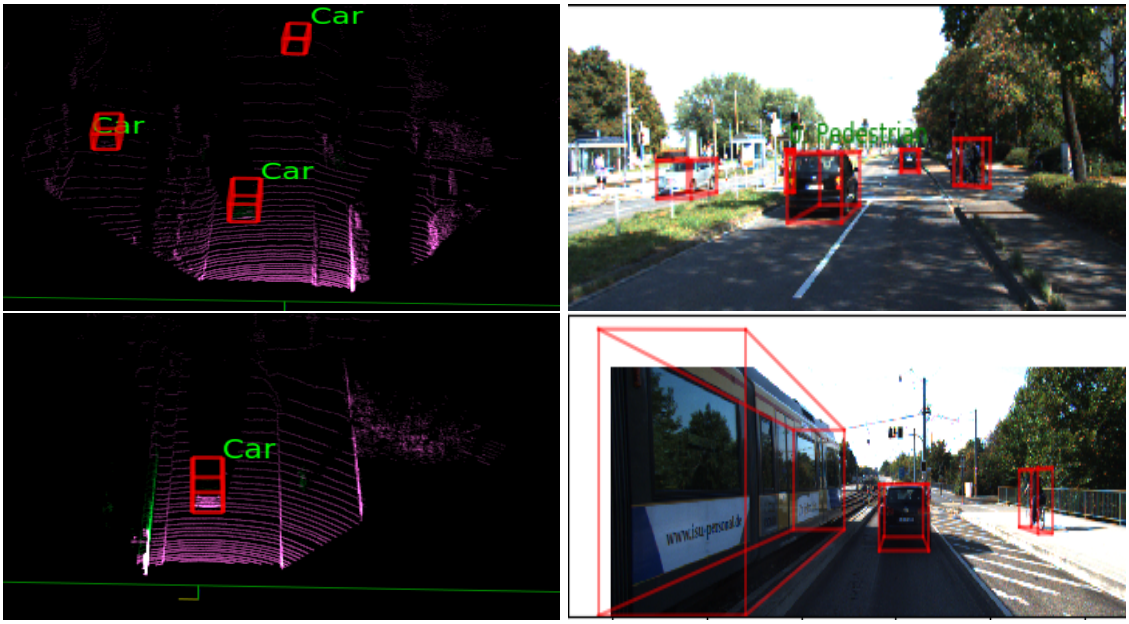


Figure 6.17: Qualitative result analysis for PointPillars depicting the imbalance of classes.

6.2.6 Effect of Complex Dataset on LiDAR Framework

Here, we utilize the complexity of the WAYMO Open dataset for observing its effect on the PointPillars framework. The complexity of the WAYMO Open dataset is already discussed in Chapter 5 (Section 5.2). Compared to KITTI, the WAYMO dataset has more diverse data and also has varying weather conditions along with a large number of objects. For example, for a total training set consisting of 4931 data files, the total number of objects for 'Car' class is 26693, 'Pedestrian' is 12218, and 'Cyclist' is 115. Compared to KITTI, these numbers are huge considering the number of data files. Along with this, it consists of varying weather conditions such as rain, fog, sunny, and varying lighting conditions such as dawn and dusk.

In this experiment, we used the same configuration as provided in the Experiment 6.2.1 except for the dimensions of the anchor boxes. For object class 'Car' the dimensions were $(2.09, 4.69, 1.80)$ and $(0.6, 1.76, 1.73)$ for object class 'Cyclist', and $(0.89, 0.96, 1.73)$ for object class 'Pedestrian' in meters with respect to width, length, and height. In WAYMO, all object classes such as 'Car', 'Truck', 'Bus', and so on are labeled as 'Vehicles', hence the update of configuration for dimensions of anchor boxes.

One of the major challenges faced while performing this experiment was to adapt the dataset formats and make it compatible with PointPillars without making many changes to the codebase. WAYMO dataset is provided in the form of TFRecord files whereas the KITTI dataset has separate folders for camera images, calibrations, labels, and Velodyne scans. PointPillars framework is designed based on KITTI format and all the calculations, projections, preprocessing, and other operations depend on the terminologies and formats as provided in the KITTI dataset. To directly use the TFRecord files in place

of KITTI format, it is a huge risk and time-consuming task for changing a lot of code.

Instead, the data from the TFRecord files were extracted into separate folders to match the KITTI format. This reduced a lot of effort and risk of injecting the bug. Due to the availability of five different cameras, it was quite confusing and complex to extract the information like calibrations, and annotations for each individual camera frames from LiDAR annotations. Due to the initial bugs, preprocessing and training of the network used to take more than 3 days for completing around 20 epochs out of 160 epochs for a subset of the WAYMO dataset with 7500 data files. This training time was reduced after fixing the bugs with respect to the calibration extraction and transformation matrix miss-match between the labels and camera coordinates.

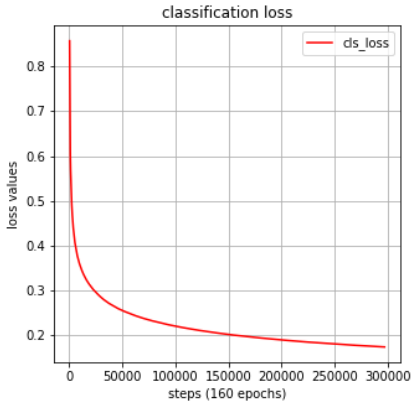
After these updates, the training time was reduced drastically, it took around 23 to 25 hours for completing the training and validations for the subset with 5000 data files. Along with these changes, there were few dataset-specific changes that needed to be done with respect to camera calibrations. For instance, for the projection of annotations from camera coordinates to LiDAR, in KITTI the front left camera is used as a reference and corresponding to that reference camera the calibrations ('Calib/P2') are used. In WAYMO, the front-facing camera is used as a reference, and calibrations ('Calib/P0') needs to be used.

Along with these code changes, the text files in which the data to be considered for training and validations are mentioned, that is, the training and validation split is mentioned are also updated. 60% of the complete training set is considered for training and the remaining 40% for validation.

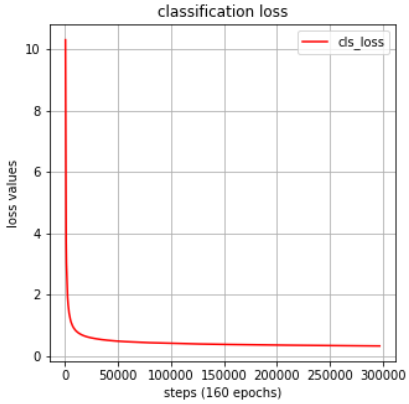
With these updates, the training and validations were carried out. For evaluation of the performance of the framework, KITTI evaluation code is utilized since the WAYMO Open dataset is converted into KITTI format and there is not much difference with respect to WAYMO Open dataset metrics for evaluation of 3D object detection. The successful and failure detection cases are visualized in Figure 6.19, along with variations in the loss function in Figure 6.18 and the quantitative results in Table 6.6.

	$AP_{AOS}(\%)$			$AP_{BEV}(\%)$			$AP_{3D}(\%)$		
	Easy	*Moderate*	*Hard*	*Easy*	*Moderate*	*Hard*	*Easy*	*Moderate*	*Hard*
Car	27.22	26.98	26.98	61.42	60.84	60.84	51.67	51.53	51.53
Pedestrian	13.25	13.16	13.16	22.69	22.51	22.51	20.24	20.18	20.18
Cyclist	6.84	6.01	6.01	23.25	20.11	20.11	23.00	20.11	20.11

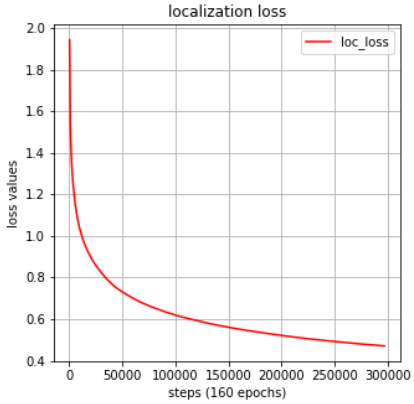
Table 6.6: Detection results for PointPillars on WAYMO Open dataset.



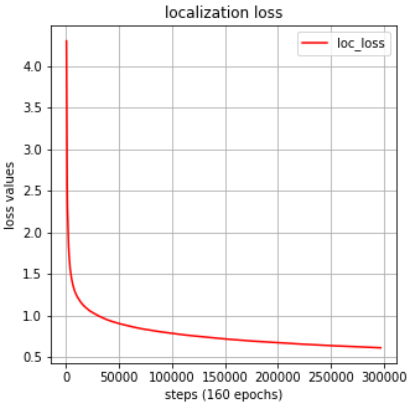
(a) classification loss for object detection class 'Car'



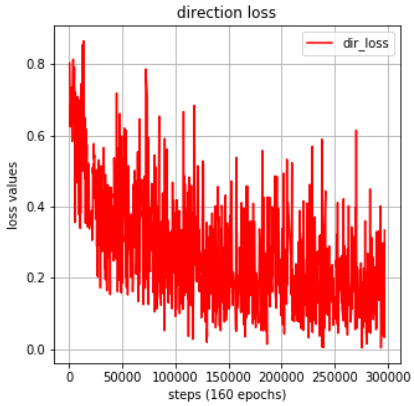
(b) classification loss for object detection class 'Pedestrian/Cyclist'



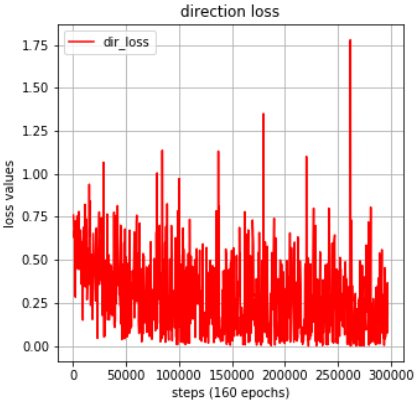
(c) localization loss for object detection class 'Car'



(d) localization loss for object detection class 'Pedestrian/Cyclist'



(e) Direction loss for object detection class 'Car'



(f) Direction loss for object detection class 'Pedestrian/Cyclist'

Figure 6.18: Variations in loss functions for the PointPillars models depicting the effect of complex dataset (WAYMO Open).

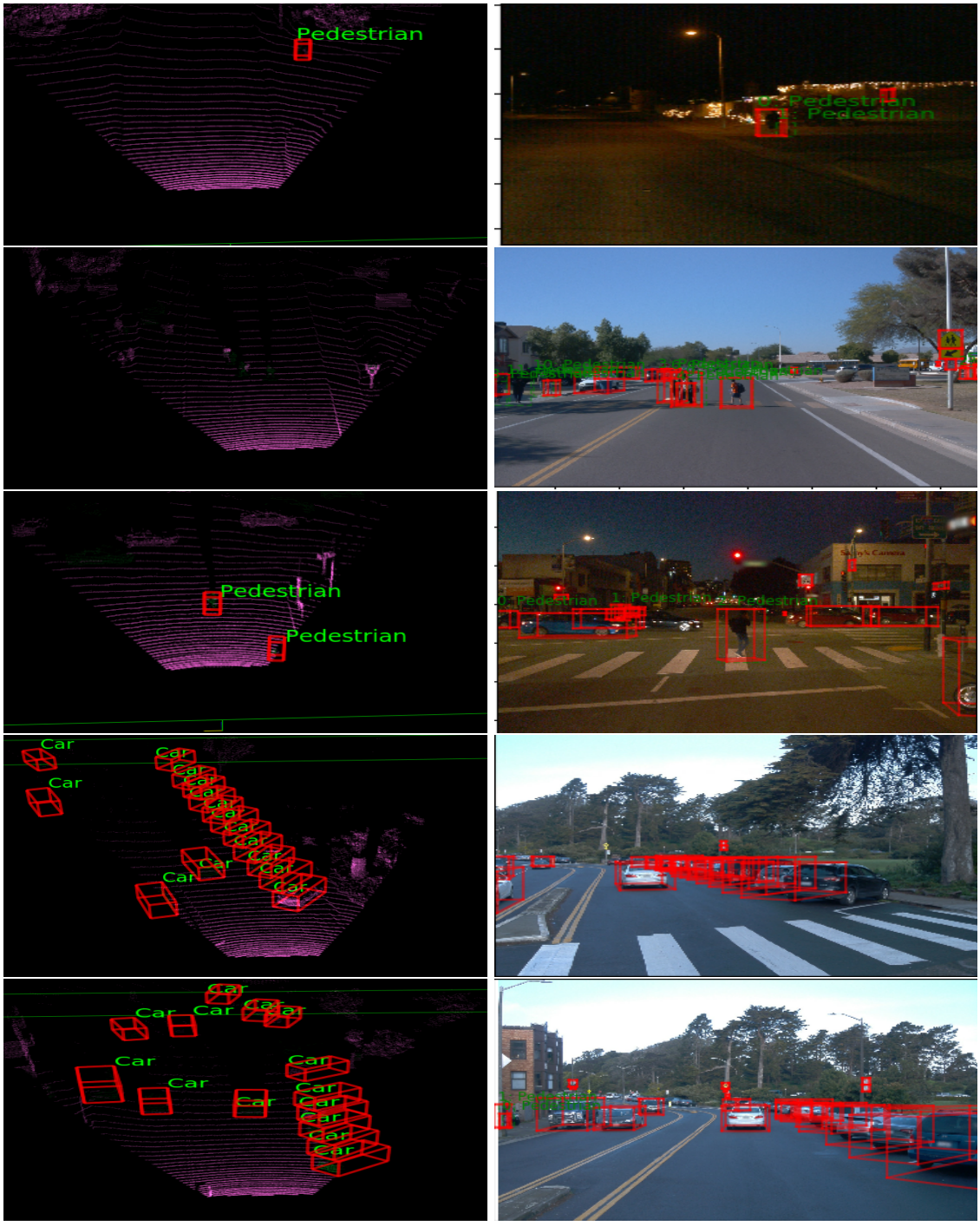


Figure 6.19: Qualitative result analysis showing success and failure cases for PointPillars on WAYMO Open dataset.

6.3 Result Analysis

For a more clear comparison of the results, the average precision values are highlighted for each of the metrics provided in the previous section. For qualitative analysis of the frameworks, the corresponding visualizations are provided as well. In this section, the discussion on the outcome of each experiment and the comparative analysis of both approaches are presented.

Experiment 6.2.1 is more of a baseline for performing other experiments, where, both the frameworks are trained and validated on a complete KITTI training dataset that consists of all kinds of objects, similar training environment, and similar optimization functions so that the results can be compared. As seen in Table 6.1, the results are quite comparable for both the frameworks except for the orientation metrics. This is because, in sparse point clouds, it is difficult to determine the orientation of small objects without the textural information. This forms a major advantage for sensor fusion technique. Even though the results for the detection of 'Pedestrians' for LiDAR framework is shown better than sensor fusion framework, in reality, the range of the LiDAR data matters and the result shown for LiDAR framework is for the range of 48 meters and for sensor fusion framework it is 70 meters. Due to utilizing the features from images in the sensor fusion framework, the AP_{BEV} of 56.45% at 70 meters is much better than 63.64% at a range of 48 meters.

There are few failure cases reported for both frameworks in Figure 6.4 and Figure 6.6. Without the textural information, it is more challenging for the detection of 'Pedestrians', as they can be easily be confused with similar narrow vertical features like poles, and are misclassified. With camera features, this lowlight can be easily tackled. Fine textural features for the objects from images help in estimating a better orientation of objects and the same can be seen in direction loss graphs.

As shown in Figure 6.2 for optimization of loss curves for the detection of 'Car', compared to the classification and localization loss curves, the direction loss curve for the LiDAR based framework shows large variations and this makes it difficult for orientation estimations. As shown in Figure 6.3, with respect to the optimization of loss curves for smaller object detection, it can be seen that even with the advantage of sensor fusion techniques, the estimations of orientation are trickier and a small fluctuations are seen when compared to LiDAR based framework.

The advantage of using LiDAR is that it provides the depth information and the objects or scene can be visualized from a higher angle that majorly helps in avoiding occlusions and truncations. This together with features from image supports better detection of objects. With similar training conditions as the first experiment, Experiments 6.2.2 and 6.2.3 were conducted, except for the validation set. The validation set consisted of only objects that are occluded and truncated respectively. Since both the frameworks use LiDAR data, the average precision for both the frameworks is approximately similar. But, because of the fusion of image features with point cloud features, the detection results are better as shown in Figure 6.7 for occlusion handling and in Figure 6.9 for truncation handling.

Basically, for LiDAR-based frameworks, points that belong to a particular object are grouped. This works well with larger objects such as 'Car' or 'Vehicles', even when they are occluded. For example, as shown in the top right image of Figure 6.7, from image plane the car which is in front is completely occluded by its preceding car but this can be easily distinguished with point cloud and estimations for

these objects are easier compared to other smaller objects. For smaller objects such as 'Pedestrians', the grouping of points in point cloud does not yield as better results since these can be easily mass grouped with other objects especially when they are occluded or partially visible. For example, as shown in the bottom left image of Figure 6.8, most of the detections are missed out as they were miss grouped and misclassified. Although the average precision for pedestrian detection for the LiDAR framework looks comparatively better than the sensor fusion framework, as discussed earlier, these values are for detection at a range of 48 meters and are not good compared to average precision of sensor fusion framework at 70 meters.

Because of color and texture information of the smaller objects from camera images, extracting these features and fusing with features from point cloud helps in precise detections and classification of the smaller objects as well. As shown in the bottom left image of Figure 6.7, many of the occluded pedestrians are detected and classified correctly. In the case of truncated objects, there will be fewer points compared to the original number of points per object. This makes it even harder for the detection of smaller objects for LiDAR only frameworks as shown in Figure 6.10 and the same is supported by the average precision values provided in Table 6.3.

In Experiment 6.2.4, we observe the effect of change in the LiDAR range on the detection of objects. As the range of LiDAR increases, the sparsity of the point cloud increases making it difficult to detect the objects that are far away from the ego vehicle. As seen in previous experiments, detection results for smaller objects at a distance of 48 meters by LiDAR-based framework are as similar to the detection results at 70 meters by sensor fusion techniques, and there are higher chances that the small objects will be considered as miscellaneous points as the sparsity increases. Hence in this experiment, we do not check for the detection of smaller objects at longer distances.

Initially, the range was changed from 0-70 meters to 0-100 meters for the detection of bigger objects such as 'Car', and an increase in 30 meters does not result in more sparse data compared to 70 meters. So this does not affect much on detection performance, although due to the injection of more space between the points the detection performance of the point cloud based framework reduces and since the features from image support the evidence of the presence of an object at a longer distance, the detection performance of sensor fusion framework does not get affected much. This can be seen in Figure 6.11 and Figure 6.12. For example, considering the images in Figure 6.11, the proposals from the images were fused with the partial detections from LiDAR data and the objects that are far from ego vehicles are detected and similarly there is a failure case shown for point cloud based framework in top images of Figure 6.12, due to the lack of supporting evidence and sparsity of point cloud, the objects at a longer distance are not detected.

Similarly, for supporting this argument further, the range was increased from 0-100 meters to 0-120 meters. This increase of 50 meters range compared to the original range, introduces a lot of sparsity into the data (an example can be seen in Figure 6.20). Similar to the argument above, as shown in the Figure 6.14, the point cloud based framework detects the objects that are in an optimal range from ego vehicle and fails to detect the objects that are too far from the vehicle. Similarly, the increase in range does not affect the sensor fusion technique as shown in Figure 6.13.

Experiment 6.2.5 is designed to check the performance of the frameworks in handling multiple classes of objects using a single model. As seen in previous experiments, for each of the framework there were two models trained one for 'Car' detection and other for 'Pedestrian' and 'Cyclist' detection, in this experiment a single model is trained for each of the frameworks for observing the capability of handling multiple classes of objects. These frameworks are designed to handle one class of objects at a time and the configurations provided are not compatible with training multiple classes. To conduct this experiment, the existing configurations were modified to make it compatible with the frameworks to train the models. This experiment was designed to check if these particular frameworks (PointPillars and F-ConvNet) are capable of handling multiple classes at a time.

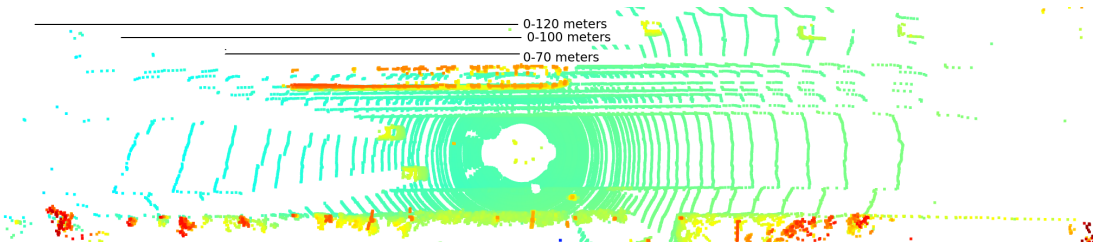


Figure 6.20: Image showing the increase in sparsity of point cloud data as the range increases. The distance inscribed in image are approximations.

As shown in Table 6.5, the average precision for detection of 'Car' is similar to the baseline for PointPillars, but in case of detection of smaller objects, it produces the worst performance. This is because of the problem with the assignment of the target, that is, for detection of each class, target anchors needs to be assigned, but in case of PointPillars, the target anchors of 'Pedestrians' and 'Cyclists' are assigned to object class 'Car'. This is because the detection of small objects like 'Pedestrians' and 'Cyclists' requires smaller receptive fields and to do this implementation of a new network that extracts features from a sparse point cloud is necessary. That is, the feature extractor network for 'Car' does not yield good performance for feature extraction for 'Pedestrians' and 'Cyclists'. If this is corrected, then the framework produces good performance for the detection of multiple classes of objects in a single model.

But in the case of F-ConvNet, the same feature extractor head is utilized for the extraction of features from the point cloud using the region proposals from camera images. Hence, the performance with multiple classes of objects is quite better when compared to PointPillars as shown in Table 6.5. As shown in Figure 6.15, F-ConvNet is capable of detecting all objects in a single model without the need of any change in the network architecture, but PointPillars required an update in feature extractor network that results in modifying the existing network architecture and Figure 6.17 shows sample detection examples for PointPillars framework. As shown in Figure 6.16, since F-ConvNet detects all three classes of objects, slight variations in the three loss functions are seen and for PointPillars only the direction loss variations are seen and other losses are similar to the baseline experiment.

In Experiment 6.2.6, the PointPillars framework is trained and validated using the WAYMO Open dataset to observe the capability of handling complex driving scenarios, varying lighting conditions, and different weather conditions. KITTI dataset is not as diverse and complex as the WAYMO dataset. This

experiment is not conducted using F-ConvNet because of the absence of necessary input requirements. For F-ConvNet, the information on 2D proposals is provided prior to the training of the network, which is obtained by training a separate 2D object detection model. These results were available for the KITTI dataset and due to time constraints, it is not able to prepare for WAYMO and hence it is not conducted with F-ConvNet.

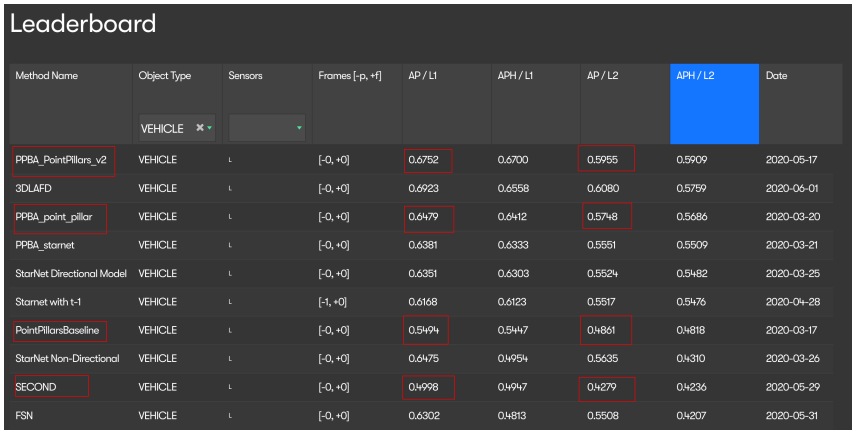
With PointPillars there were few difficulties in the beginning as explained under Experiment 6.2.6. In the WAYMO dataset, the density of the objects per frame is too high compared to KITTI, and the rate of occlusion of objects is high. But this does not affect much in case of the detection of bigger objects like 'Car'. The PointPillars framework and the configurations available are designed to handle the KITTI dataset, hence modifying few parts of configurations and codebase as discussed earlier (under Experiment 6.2.6) does not improve the performance of the framework.

This is because, in KITTI, there are different labels for different vehicle type, for example, there are labels such as 'Car', 'Truck', 'Van', and so on, but in WAYMO, all these classes are labeled as 'Vehicles'. With respect to PointPillars framework, re-annotating the WAYMO dataset is not an option, hence all the objects which were labeled as 'Vehicles' are considered as 'Car', and along with this for assigning the target anchors the size of anchor boxes are varied and this conflicts with initial feature extractions from the point cloud. To adjust to these new changes, the network architecture was needed to be modified and due to timing constraints, it was not able to produce an optimal feature extractor for obtaining better performance. Although the feature extractor was not modified, the performance results with respect to the detection of 'Car' are quite good when compared to KITTI.

With respect to the detection of smaller objects, the performance is reduced by almost 30 to 40%, because of the overlapping receptive fields due to high occlusion rate. As discussed earlier, for handling the smaller objects in case of overlapping objects, more sophisticated feature extractors are necessary. Due to this, the performance of the detection of 'Pedestrians' and 'Cyclists' are reduced. Due to the high density of the objects, the orientation regressions needs to be improved as well. Even though the loss curves (Figure 6.18) for classification and localization show significant improvement over the training period, there were many failure cases observed especially with small object detection. Few sample cases for detection performance, failure cases, and the ground truth are provided in Figure 6.19.

Even though the performance on WAYMO dataset is comparatively lesser with respect to KITTI dataset, the average precision of detection of 'Car' represented in Table 6.6 is quite comparable to the results of the frameworks based on PointPillars presented in leader-board[1] at WAYMO website for 3D object detection competition. A sample screen shot can be seen in Figure 6.21.

From the above discussion, it is seen that the performance of LiDAR based framework on the detection of big objects such as 'Car' is similar to that of sensor fusion framework even in various conditions and scenarios, but the performance with respect to detection of smaller objects is not quite comparable. The introduction of a new sensing modality such as camera along with LiDAR improvises the detection capabilities and improves the confidence level of the object detection framework.



Leaderboard

Method Name	Object Type	Sensors	Frames [-p, +f]	AP / L1	APH / L1	AP / L2	APH / L2	Date
	VEHICLE							
PPBA_PointPillars_v2	VEHICLE	L	[-0, +0]	0.6752	0.6700	0.5955	0.5909	2020-05-17
3DLAFD	VEHICLE	L	[-0, +0]	0.6923	0.6558	0.6080	0.5759	2020-06-01
PPBA_point_pillar	VEHICLE	L	[-0, +0]	0.6479	0.6412	0.5748	0.5686	2020-03-20
PPBA_starnet	VEHICLE	L	[-0, +0]	0.6381	0.6333	0.5551	0.5509	2020-03-21
StarNet Directional Model	VEHICLE	L	[-0, +0]	0.6351	0.6303	0.5524	0.5482	2020-03-25
Starnet with t-1	VEHICLE	L	[-1, +0]	0.6168	0.6123	0.5517	0.5476	2020-04-28
PointPillarsBaseline	VEHICLE	L	[-0, +0]	0.5494	0.5447	0.4861	0.4818	2020-03-17
StarNet Non-Directional	VEHICLE	L	[-0, +0]	0.6475	0.4954	0.5635	0.4310	2020-03-26
SECOND	VEHICLE	L	[-0, +0]	0.4998	0.4947	0.4279	0.4236	2020-05-29
FSN	VEHICLE	L	[-0, +0]	0.6302	0.4813	0.5508	0.4207	2020-05-31

Figure 6.21: Leader-board from WAYMO website [1]. Highlighted methods and values are comparable with results presented in Table 6.6.

7

Conclusions

Perception system of an autonomous vehicle is the basic and complex system that creates an internal representation of vehicle surroundings, generates the collision-free path for safe maneuverability of the vehicle, generates commands for obstacle avoidance and emergency braking, and many such tasks. 3D object detection and classification forms the basis for generating all this information in real-time and with high confidence values. Perceiving the environment using various sensors is the source for representing the environment of the vehicle on which object detection is performed.

Keeping the objective of this research in mind, that is, to perform various analysis, literature studies and experimental analysis to perform the comparative study on 3D object detection frameworks based on LiDAR data and sensor fusion technology, various sensors responsible for perceiving the environment along with their advantages and disadvantages, selection criteria, and various sensor fusion techniques are discussed in Chapter 3, various multi-modal datasets available publicly, along with state-of-the-art methods performing 3D object detection together with the criteria for selecting them are discussed in Chapter 4. Metrics for evaluating the frameworks and the technical report on the selected datasets and frameworks are discussed in Chapter 5.

Chapter 6 provides the detailed information on the experimental setup, various experiments conducted for supporting the objective of this research and detailed analysis on the results of the experiments by comparing their performances are discussed. With this information at hand, the use of LiDAR data over other sensors has various advantages such as estimating the depth of the object, distance to impact, physical structure, and location on the ground level that helps in localizing the object with respect to the ego vehicle and more. But there are also few lowlights such as distortion in signal, sparsity, and generation of the huge amounts of unnecessary data points.

In spite of these highlights and lowlights, the 3D object detection frameworks based on LiDAR data have proven to perform better in some scenarios, in the detection of bigger objects, and detection of smaller objects at a smaller range. But in real-time scenarios, it is not recommended to concentrate only on bigger objects or to keep varying the range of sensors for the detection of smaller objects. To overcome these lowlights of LiDAR frameworks, various sensor fusion techniques are discussed to show the improvement in object detection by utilizing the complementary effects of various sensors by fusing the data from multiple sources.

The textural information from RGB images helps in distinguishing between the pedestrian and a

narrow vertical object even at a longer distance. Information like these are fused with the LiDAR data to improve the detection performance even at a longer distance. This fusion also helps in overcoming the sparsity issues associated with LiDAR. With fusion techniques, there is no need for varying the range of sensors for small object detection and also they provide real-time performance with high confidence values. This argument is supported by the detailed analysis of the experimental results from Section 6.3.

7.1 Lessons Learned

Performing this research resulted in learning many lessons starting from handling datasets to learning many different approaches for detecting objects. Few of the lessons learned are listed below:

(a) Various representation techniques for representing the data, different projection techniques for projecting LiDAR data and camera images.

(b) Different approaches, methods, and techniques for 3D object detection based on camera images, LiDAR data, and fusion of these two sensor data.

(c) Various sensors and their characteristics together with their complementary behavior along with various approaches to fusing the sensor data.

(d) Importance of sensor calibrations in fusing the data and projecting from one coordinate to other coordinate.

7.2 Future Work

Based on the highlights and lowlights discussed in this research work, following future research directions are enumerated:

(a) Safety critical points with respect to perception system in case of a sensor failure, noise in measurement, or effects of adverse weather conditions are to be researched.

(b) Implementing more sophisticated algorithms for the detection of smaller objects such as pedestrians in LiDAR data without confusing them with narrow objects.

(c) Improvising the better visibility for an autonomous vehicle by introducing Radar sensors along with cameras and LiDARs for far object detection and distance estimations.

(d) Improving the performance of PointPillars framework with the WAYMO Open dataset for detection of 'Pedestrian/Cyclist' and improving detection performance as a whole.

(e) Instead of providing the pre-generated 2D region proposals as input to the F-ConvNet, implementing an RPN for extracting the 2D proposals and using these proposals to improve the detection performance as a whole and in particular for smaller object detection.

(f) Implementing an algorithm, to mask the effect of adverse weather conditions on sensor data and perform object detection and classification.

Glossary

Active sensors	Sensors that emits energy into the environment and measures the response.
Bounding box	Bounding boxes are used for labeling the objects and their locations in an image. A rectangle is used in case of 2D and a cuboid in case of 3D labeling.
Ego vehicle	It is the reference vehicle. The autonomous vehicle in this research is referred to as ego vehicle.
Exteroceptive sensors	Sensors that measure the values which are external to the system.
Frustum axis	A frustum for a proposal, is generated by sliding a parallel plane from image plane that are perpendicular to the optical axis, that is also known as frustum axis[64].
Maneuverability	Mobility + Number of independent wheel reconfiguration.
Mobility	Number of independent motion directions immediately achievable.
NMS	Non-maximum suppression is an edge thinning technique, it is applied for finding the locations with sharpest change in intensity values[69].
Passive sensors	Sensors that measures the ambient energy which enters the sensors from environment.
Proprioceptive sensors	Sensor that measure the values that are internal to the system.
ROI	For object detection, ROI specifies borders of an object under consideration[68].

RPN	Region Proposal Network generates a number of proposals that will be considered by a classifier for checking the existence of an object.
Strides	Stride is a controlling parameter of CNN that controls the number of pixels shifts given the input matrix. When it is set to 1 then the filter moves 1 pixel at a time, when it is set to 2 then the filter moves 2 pixels at a time and so on..
Voxel	Voxel represents a point in 3D space.

Acronyms

2D	2 Dimensional
3D	3 Dimensional
3DVP	3D Voxel Pattern
AOS	Average Orientation Similarity
AP	Average Precision
APH	Average Precision Heading
AVOD	Aggregate View Object Detection
BEV	Bird's Eye View
CNN	Convolutional Neural Networks
ContFuse	Deep Continuous Fusion
CPM	Camera Plane Maps
DCV	Depth Cost Volume
DOF	Depth of Field
EKF	Extended Kalman Filters
FCN	Fully Convolutional Network
FPN	Feature Pyramid Network
GB	Gigabyte
GNSS	Global navigation Satellite System
GPS	Global Positioning System
GPU	Graphical Processing Unit
IMU	Inertial Measurement Unit
IOU	Intersection Over Union
IPOD	Intensive Point-based Object Detector

JPEG	Joint Photographic Experts Group
LiDAR	Light Detection and Ranging
MLP	Multi Layer Perceptrons
Mono3D	Monocular 3D
MV3D	Multi-View 3D Object Detection
NMS	Non-Maximum Suppression
PCD	Point Cloud Data
PNG	Portable Network Graphics
R-CNN	Region-based CNN
Radar	Radio Detection and Ranging
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RGB	Red Green Blue
ROI	Region of Interest
RPN	Region Proposal Network
SDN	Stereo Depth Network
SECOND	Sparsely Embedded Convolutional Detection
SSD	Single Shot Detectors
SVM	Support Vector Machines
TB	Terabyte
TOF	Time-of-Flight
UKF	Unscented Kalman Filters
VFE	Voxel Feature Encoding
VW	Volks Wagon

YOLO You Only Look Once

A

Creating Baseline Networks

A.1 PointPillars

Below are the instructions to create a baseline model for 3D object detection using LiDAR data. These instructions are specific to be used in the cluster machine (mentioned in Section 6.1).

Dataset

Download the KITTI dataset from the given link in Appendix C and arrange it into the folder structure shown in Figure A.1. The training folder consists of 7481 training data files and the testing folder consists of 7518 testing data files. The images in the image_2 folder are used only for visualization purposes and to reduce the point cloud into the front view. This is because the KITTI dataset has only front-facing cameras and labels are in camera coordinates. And these reduced point cloud data will be stored in the velodyne_reduced folder.

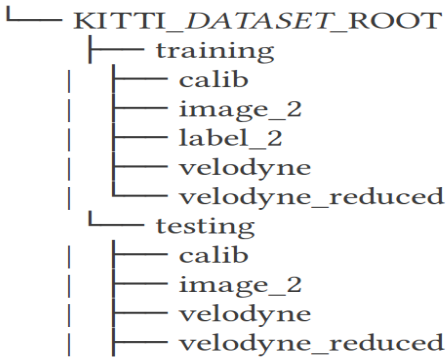


Figure A.1: Folder structure for storing KITTI dataset for creating PointPillars baseline model.

Setup And Codebase

It is recommended to use either Anaconda environment or virtual environments for setting up the system to execute the code. Download the code from the link in Appendix C and install the following python packages.

- Python 3.7
- pytorch torchvision and its dependencies
- google-sparsehash
- fire
- shapely
- pybind11
- protobuf
- scikit-image
- numba
- pillow
- tensorboardX
- libboost-all-dev

Since the code for PointPillars is developed on SECOND framework codebase, an additional sparse convolutional network from Facebook needs to be installed and the instructions are provided in the same GitHub page.

Once the installation of the above packages is complete, set up the Cuda for numba and add the path of the second.pytorch as python path in ∼/.bashrc file as shown in Figure A.2.

```
export NUMBAPRO_CUDA_DRIVER=/usr/local/cuda-10.2/targets/x86_64-linux/lib/stubs/libcuda.so
export NUMBAPRO_NVVM=/usr/local/cuda-10.2/nvvm/lib64/libnvvm.so
export NUMBAPRO_LIBDEVICE=/usr/local/cuda-10.2/nvvm/libdevice

export PYTHONPATH=$PYTHONPATH:'xxxx/xxx/xxxx/Pointpillars/second.pytorch'

export PATH=/usr/local/cuda-10.2/bin${PATH:+:${PATH}}
export CPATH=/usr/local/cuda-10.2/include:$CPATH
export LD_LIBRARY_PATH=/usr/local/cuda-10.2/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

Figure A.2: Cuda and numba setup for PointPillars.

Data Preprocessing

Dataset preparation and preprocessing takes place in 3 steps as given below:

1. Creating info files that consist of all the necessary information required for training and validating the network such as annotation, camera calibration, image, point cloud data. This is created by executing the below command:

   python create_data.py create_kitti_info_file –data_path=KITTI_DATASET_ROOT

2. Generating reduced point cloud as mentioned in the previous section. These are generated by using the below command:

   python create_data.py create_reduced_point_cloud –data_path=KITTI_DATASET_ROOT

3. Generating ground-truth information. These are generated by using the below command:

   python create_data.py create_groundtruth_database –data_path=KITTI_DATASET_ROOT

Once these files are generated, the links for the generated files need to be updated along with the dataset folder in the configuration files (second.pytorch/second/configs/pointpillars/car/xyres_16.porto and second.pytorch/second/configs/pointpillars/ped_cycle/xyres_16.porto).

Training and Validation

Once the preprocessing is complete, the training and validation of the network are performed using below the commands:

python pytorch/train.py train –config_path= path to config file –model_dir=/path/to/model_dir

python pytorch/train.py evaluate –config_path= path to config file –model_dir=/path/to/model_dir

The config_path (any of the config files mentioned above) is where the configurations are mentioned with which the model is created and model_dir is for generating the output and logs for the model.

Inferencing

For inferencing the output, execute the following command and provide the required information as requested by the inferencing window.

python ./kittiviewer/viewer.py

A.2 Frustum ConvNet

Below are the instructions to create a baseline model for 3D object detection using sensor fusion technique. These instructions are specific to be used in the cluster machine (mentioned in Section 6.1).

Dataset

Download the KITTI dataset from the given link in Appendix C and arrange it into the folder structure shown in Figure A.3. For F-ConvNet the downloaded data should be placed inside a folder named 'data' inside codebase.

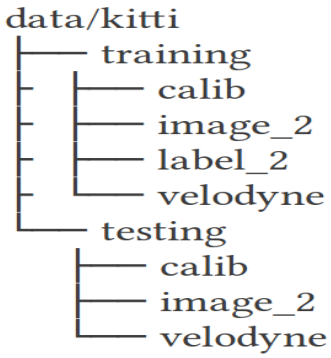


Figure A.3: Folder structure for storing KITTI dataset for creating F-ConvNet baseline model.

Setup And Codebase

Download the code from the link in Appendix C and install the following python packages for Frustum ConvNet. It is recommended to use a separate Anaconda environment or virtual environment for setting up an environment for Frustum ConvNet.

- Python 3.6+
- pytorch torchvision and its dependencies
- opencv
- yaml
- pybind11
- pickle
- tensorboardX
- libboost-all-dev

Once the codebase is downloaded and the libraries are installed, navigate to the folder named 'ops' and execute the following two files in order using the commands below. This will compile a few of the c++ code files required for box estimations. To compile these files, the compiler should support the GCC 8.1.0 version or higher.

bash clean.sh

bash make.sh

Training and Validation

Training the network involves two stages as given below:

A. First Stage

First, the pickle files are created for training, and validation based on the input files present in folder 'frustum-convnet/kitti/image_sets/' and 'frustum-convnet/kitti/rgb_detections/', these files need not be changed or modified for KITTI dataset. The pickle files are created using the below command and the files are generated at 'kitti/data/pickle_data'.

python kitti/prepare_data.py –car_only –gen_train –gen_val –gen_val_rgb_detection

The option 'car_only' is to generate information files related to object 'Car' and it can be changed to 'people_only' for 'Pedestrian' and 'Cyclist' detection. Once these files are generated, training and evaluation can be performed by using below commands:

python train/train_net_det.py –cfg config file OUTPUT_DIR output/car_train

python train/test_net_det.py –cfg config file OUTPUT_DIR output/car_train TEST.WEIGHTS output/car_train/model_0050.pth

The configuration file are present in 'frustum-convnet/cfgs/' folder. The output directory can be modified to any location where the output will be stored.

All the above steps can be performed by executing the bash script as shown in below command:

bash scripts/car_train.sh

B. Refinement Stage

Based on the results generated by the first stage, the second stage utilizes the output files and creates new pickle files for the refinement stage. These pickle files will be stored under 'kitti/data/pickle_data_refine' folder and are generated using below command.

python kitti/prepare_data_refine.py –car_only –gen_train –gen_val_det –gen_val_rgb_detection

Once these files are generated, training and evaluation of the refinement stage are performed using the below commands.

python train/train_net_det.py –cfg config file OUTPUT_DIR output/car_train_refine

python train/test_net_det.py –cfg config file OUTPUT_DIR output/car_train_refine TEST.WEIGHTS output/car_train_refine/model_0050.pth

For the refinement stage, the configuration files will be in the 'frustum-convnet/cfgs/' folder and the file name starts with 'refine_*'. Similar to the first stage the output directory can be modified to any location where the output will be stored.

All the above steps can be performed by executing the bash script as shown in the below command:

bash scripts/car_train_refine.sh

Both stages can be performed by using the below command. For 'Pedestrian/Cyclist' detection replace 'car_all.sh' by 'people_all.sh'.

bash scripts/car_all.sh

Inference

For inferencing the output, the output will be generated in the same label format of the KITTI dataset, any standard visualization tools that supports the KITTI dataset can be used here. One of the recommended visualization tools is [73].

B

WAYMO Data Handler

B.1 WAYMO to KITTI Conversion

Below are the instructions to extract the data from TFRecord files of the WAYMO Open dataset into the KITTI dataset format.

Dataset

Download the subset of the WAYMO Open dataset from the link associated with 'WAYMO Open' under 'Multi-modal Datasets' in Appendix C. For training purpose download the data under the 'Training' tab and for testing purpose download the data under the 'Validation' tab. The downloaded data will be compressed using '.tar' format and decompressing these files results in many TFRecord files (.tfrecord).

Setup And Codebase

Download the scripts from the conversion of WAYMO Open dataset into KITTI dataset format from the link provided under the 'Software Tool' section of Appendix C and install the following python libraries.

- Tensorflow 1.15.0
- waymo-open-dataset
- OpenCV-python

Conversion

Navigate into the folder 'Waymo_Kitti_converter' and execute the below command for the conversion of WAYMO data into KITTI format.

python Waymo_to_kitti.py –source=source path–dest=output folder path –all

Here, the source path refers to the folder containing the TFRecord files, and the output folder path refers to the folder in which the converted data will be stored. The option '–all' generates point cloud, images, labels, and camera calibration in separate folders. Along with this, there are three more options

given, '–velo' generates point cloud information with calibration and labels, '–img' generates camera images with camera calibration and labels, and '–oclu' generates point cloud, images, labels, and camera calibration with basic occlusion information. The occlusion level here is defined by two values in label files, value 1 for occluded objects and value 0 for non-occluded objects.

Output Folder

The output folder is represented in Figure B.1. The output folder consists of four main folders named 'Calibration', 'Camera', 'Labels', and 'Velodyne'. The folder 'Calibration' consists of two subfolders, 'Calib' - consists of calibration for each individual camera in separate folders, and 'Calib_all' - consists of all calibrations related to all cameras and LiDAR in a single file for each frame respectively. The folder 'Camera' has five subfolders based on their mounting angle 'Front', 'Front_left', 'Front_right', 'Side_left', 'Side_right'. The folder 'Labels' has two subfolders, 'Label' - consists of labels for each individual camera, and 'Labels_all' - consists of all labels in a single file for each frame. The folder 'Velodyne' consists of point cloud information for each frame. Each calibration file consists of information on all five camera's intrinsic matrix ('P0' to 'P4'), rectification matrix ('R0_rect'), and transformation matrix from vehicle frame to all five camera frames ('Tr_velo_to_cam_0' to 'Tr_velo_to_cam_4') The subfolders named from 0 to 4 corresponds to the respective reference cameras as given below.

Front	-	0
Front_left	-	1
Front_right	-	2
Side_left	-	3
Side_right	-	4

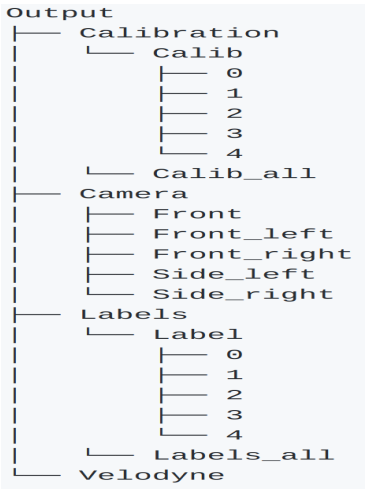


Figure B.1: Output folder structure of WAYMO to KITTI converter tool.

B.2 PointPillars With WAYMO

In this section, the instructions on creating the PointPillars network model with the WAYMO Open dataset are illustrated.

Dataset

Once the WAYMO Open dataset is extracted into KITTI format, all the data in the output folder is not required for the training of PointPillars network. Here, the data corresponding to only one reference camera is selected, that is, the images, labels, camera calibrations corresponding to reference camera 'Front' are selected along with the point cloud data and placed it in the format similar to KITTI as shown in Figure A.1.

Codebase Modifications

First, the text files under the folder 'second.pytorch/second/data/ImageSets/' needs to updated to match the new data files. The training and validation sets are divided as 60% and 40% respectively. In original PointPillars code, due to the use of front left camera as a reference all the calculations with respect to the intrinsic matrix in camera calibrations were referred to 'P2' and since with WAYMO the reference camera used is the 'Front' camera, all these references needs to be updated to 'P0'.

Along with these changes, since the WAYMO Open dataset does not provide IMU information, the 'Tr_imu_to_velo' transformation matrix references in calibration files are removed from the code. With these code changes and the configuration changes mentioned under Experiment 6.2.6, the models with PointPillars are generated.

C

Links

Frameworks

F-ConvNet	-	https://github.com/zhixinwang/frustum-convnet
PointPillars	-	https://github.com/nutonomy/second.pytorch

Multi-modal Datasets

Oxford RobotCar	-	http://robotcar-dataset.robots.ox.ac.uk/downloads/
Honda 3D (H3D)	-	https://usa.honda-ri.com/hdd/introduction/h3d
KITTI	-	http://www.cvlibs.net/datasets/kitti/
ApolloScape	-	http://apolloscape.auto/scene.html
BLVD	-	https://github.com/VCCIV/BLVD/
LyFT	-	https://self-driving.lyft.com/level5/data/
NuScenes	-	https://www.nuscenes.org/download
WAYMO Open	-	https://waymo.com/open/download/

Software Tool

WAYMO to KITTI Converter	-	https://github.com/Sreeni1204/Waymo_Kitti_converter

Miscellaneous

Cluster configuration	-	https://wr0.wr.inf.h-brs.de/wr/hardware/hardware.html
WAYMO Leader-board	-	https://waymo.com/open/challenges/3d-detection/

References

[1] 3D Detection - Waymo. https://waymo.com/open/challenges/3d-detection/, July 2020. (Accessed on 12/07/2020).

[2] WR Cluster Hardware. https://wr0.wr.inf.h-brs.de/wr/hardware/hardware.html, March 2020. (Accessed on 12/03/2020).

[3] Eduardo Arnold, Omar Y. Al-Jarrah, Mehrdad Dianati, Saber Fallah, David Oxtoby, and Alex Mouzakitis. A Survey on 3D Object Detection Methods for Autonomous Driving Applications. *IEEE Trans. Intell. Transp. Syst.*, 20(10):3782–3795, 2019. doi: 10.1109/TITS.2019.2892405. URL https://doi.org/10.1109/TITS.2019.2892405.

[4] Jorge Beltrán, Carlos Guindel, Francisco Miguel Moreno, Daniel Cruzado, Fernando García, and Arturo de la Escalera. BirdNet: A 3D Object Detection Framework from LiDAR Informati. In *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, pages 3517–3523. IEEE, 2018. doi: 10.1109/ITSC.2018.8569311. URL https://doi.org/10.1109/ITSC.2018.8569311.

[5] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A multimodal dataset for autonomous driving. *Computing Research Repository CoRR*, abs/1903.11027, 2019. URL http://arxiv.org/abs/1903.11027.

[6] Yingjie Cai, Buyu Li, Zeyu Jiao, Hongsheng Li, Xingyu Zeng, and Xiaogang Wang. Monocular 3D Object Detection with Decoupled Structured Polygon Estimation and Height-Guided Depth Estimation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 10478–10485. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/6618.

[7] Federico Castanedo. A Review of Data Fusion Techniques. *The Scientific World Journal*, 2013: 704504, Oct 2013. ISSN 2356-6140. doi: 10.1155/2013/704504. URL https://doi.org/10.1155/2013/704504.

[8] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3D Object Detection for Autonomous Driving. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2147–2156. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.236. URL https://doi.org/10.1109/CVPR.2016.236.

[9] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D Object Detection Network for Autonomous Driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6526–6534. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.691. URL https://doi.org/10.1109/CVPR.2017.691.

[10] Grégoire Payen de La Garanderie, Amir Atapour Abarghouei, and Toby P. Breckon. Eliminating the Blind Spot: Adapting 3D Object Detection and Monocular Depth Estimation to 360 ˆ\circ ◦ Panoramic Imagery. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, volume 11217 of *Lecture Notes in Computer Science*, pages 812–830. Springer, 2018. doi: 10.1007/978-3-030-01261-8_48. URL https://doi.org/10.1007/978-3-030-01261-8_48.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848. URL https://doi.org/10.1109/CVPR.2009.5206848.

[12] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pages 1355–1361. IEEE, 2017. doi: 10.1109/ICRA.2017.7989161. URL https://doi.org/10.1109/ICRA.2017.7989161.

[13] Markus Enzweiler and Dariu M. Gavrila. A Multilevel Mixture-of-Experts Framework for Pedestrian Classification. *IEEE Trans. Image Process.*, 20(10):2967–2979, 2011. doi: 10.1109/TIP.2011.2142006. URL https://doi.org/10.1109/TIP.2011.2142006.

[14] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vis.*, 88(2):303–338, 2010. doi: 10.1007/s11263-009-0275-4. URL https://doi.org/10.1007/s11263-009-0275-4.

[15] Di Feng, Lars Rosenbaum, and Klaus Dietmayer. Towards Safe Autonomous Driving: Capture Uncertainty in the Deep Neural Network For Lidar 3D Vehicle Detection. In *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, pages 3266–3273. IEEE, 2018. doi: 10.1109/ITSC.2018.8569814. URL https://doi.org/10.1109/ITSC.2018.8569814.

[16] Di Feng, Christian Haase-Schuetz, Lars Rosenbaum, Heinz Hertlein, Fabian Duffhauss, Claudius Gläser, Werner Wiesbeck, and Klaus Dietmayer. Deep Multi-modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges. *Computing Research Repository CoRR*, abs/1902.07830, 2019. URL http://arxiv.org/abs/1902.07830.

[17] Shimin Feng. *Sensor fusion with Gaussian processes.* PhD thesis, University of Glasgow, UK, 2014. URL http://theses.gla.ac.uk/5626/.

[18] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 3354–3361. IEEE Computer Society, 2012. doi: 10.1109/CVPR.2012.6248074. URL https://doi.org/10.1109/CVPR.2012.6248074.

[19] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *I. J. Robotics Res.*, 32(11):1231–1237, 2013. doi: 10.1177/0278364913491297. URL https://doi.org/10.1177/0278364913491297.

[20] Ross B. Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1440–1448. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.169. URL https://doi.org/10.1109/ICCV.2015.169.

[21] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 580–587. IEEE Computer Society, 2014. doi: 10.1109/CVPR.2014.81. URL https://doi.org/10.1109/CVPR.2014.81.

[22] Alejandro González, David Vázquez, Antonio M. López, and Jaume Amores. On-Board Object Detection: Multicue, Multimodal, and Multiview Random Forest of Local Experts. *IEEE Trans. Cybern.*, 47(11):3980–3990, 2017. doi: 10.1109/TCYB.2016.2593940. URL https://doi.org/10.1109/TCYB.2016.2593940.

[23] Qishen Ha, Kohei Watanabe, Takumi Karasawa, Yoshitaka Ushiku, and Tatsuya Harada. MFNet: Towards real-time semantic segmentation for autonomous vehicles with multi-spectral scenes. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 5108–5115. IEEE, 2017. doi: 10.1109/IROS.2017.8206396. URL https://doi.org/10.1109/IROS.2017.8206396.

[24] David Hall and James Llinas. An Introduction to Multisensor Data Fusion. *Proceedings of the IEEE*, 85:6 – 23, 02 1997. doi: 10.1109/5.554205.

[25] Stephen Hsu, Sunil Acharya, Abbas Rafii, and Richard New. Performance of a Time-of-Flight Range Camera for Intelligent Vehicle Safety Applications. In *Advanced Microsystems for Automotive Applications 2006*, pages 205–219, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33410-1. URL https://doi.org/10.1007/3-540-33410-6_16.

[26] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. The ApolloScape Dataset for Autonomous Driving. In *2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT,*

USA, June 18-22, 2018, pages 954–960. IEEE Computer Society, 2018. doi: 10.1109/CVPRW.2018.00141. URL http://openaccess.thecvf.com/content_cvpr_2018_workshops/w14/html/Huang_The_ApolloScape_Dataset_CVPR_2018_paper.html.

[27] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960. ISSN 0021-9223. doi: 10.1115/1.3662552. URL https://doi.org/10.1115/1.3662552.

[28] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Lyft level 5 perception dataset 2020, 2019. URL https://level5.lyft.com/dataset/.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Association for Computing Machinery ACM*, 60(6):84–90, 2017. doi: 10.1145/3065386. URL http://doi.acm.org/10.1145/3065386.

[30] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L. Waslander. Joint 3D Proposal Generation and Object Detection from View Aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 1–8. IEEE, 2018. doi: 10.1109/IROS.2018.8594049. URL https://doi.org/10.1109/IROS.2018.8594049.

[31] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 12697–12705. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.01298. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Lang_PointPillars_Fast_Encoders_for_Object_Detection_From_Point_Clouds_CVPR_2019_paper.html.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nat.*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. URL https://doi.org/10.1038/nature14539.

[33] Bo Li. 3d fully convolutional network for vehicle detection in point cloud. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 1513–1518. IEEE, 2017. doi: 10.1109/IROS.2017.8205955. URL https://doi.org/10.1109/IROS.2017.8205955.

[34] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle Detection from 3D Lidar Using Fully Convolutional Network. In *Robotics: Science and Systems XII, University of Michigan, Ann Arbor, Michigan, USA, June 18 - June 22, 2016*, 2016. doi: 10.15607/RSS.2016.XII.042. URL http://www.roboticsproceedings.org/rss12/p42.html.

[35] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution On X-Transformed Points. In *Advances in Neural Information Processing Systems*

31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada, pages 828–838, 2018. URL http://papers.nips.cc/paper/7362-pointcnn-convolution-on-x-transformed-points.

[36] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep Continuous Fusion for Multi-sensor 3D Object Detection. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*, volume 11220 of *Lecture Notes in Computer Science*, pages 663–678. Springer, 2018. doi: 10.1007/978-3-030-01270-0_39. URL https://doi.org/10.1007/978-3-030-01270-0_39.

[37] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun. Multi-Task Multi-Sensor Fusion for 3D Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 7345–7353. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00752. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Liang_Multi-Task_Multi-Sensor_Fusion_for_3D_Object_Detection_CVPR_2019_paper.html.

[38] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature Pyramid Networks for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 936–944. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.106. URL https://doi.org/10.1109/CVPR.2017.106.

[39] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2999–3007. IEEE Computer Society, 2017. doi: 10.1109/ICCV.2017.324. URL https://doi.org/10.1109/ICCV.2017.324.

[40] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, volume 9905 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2016. doi: 10.1007/978-3-319-46448-0_2. URL https://doi.org/10.1007/978-3-319-46448-0_2.

[41] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The Oxford RobotCar dataset. *I. J. Robotics Res.*, 36(1):3–15, 2017. doi: 10.1177/0278364916679498. URL https://doi.org/10.1177/0278364916679498.

[42] Roberto Manduchi, Andres Castano, Ashit Talukder, and Larry H. Matthies. Obstacle Detection and Terrain Classification for Autonomous Off-Road Navigation. *Auton. Robots*, 18(1):81–102, 2005. doi: 10.1023/B:AURO.0000047286.62481.1d. URL https://doi.org/10.1023/B:AURO.0000047286.62481.1d.

[43] Gregory P. Meyer, Jake Charland, Darshan Hegde, Ankit Laddha, and Carlos Vallespi-Gonzalez. Sensor Fusion for Joint 3D Object Detection and Semantic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 1230–1237. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPRW.2019.00162. URL http://openaccess.thecvf.com/content_CVPRW_2019/html/WAD/Meyer_Sensor_Fusion_for_Joint_3D_Object_Detection_and_Semantic_Segmentation_CVPRW_2019_paper.html.

[44] Roderick Murray-Smith and Barak A. Pearlmutter. Transformations of Gaussian Process Priors. In *Deterministic and Statistical Methods in Machine Learning, First International Workshop, Sheffield, UK, September 7-10, 2004, Revised Lectures*, volume 3635 of *Lecture Notes in Computer Science*, pages 110–123. Springer, 2004. doi: 10.1007/11559887_7. URL https://doi.org/10.1007/11559887_7.

[45] Rafael Padilla, Sergio Netto, and Eduardo da Silva. A Survey on Performance Metrics for Object-Detection Algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, 07 2020. doi: 10.1109/IWSSIP48289.2020.

[46] Abhishek Patil, Srikanth Malla, Haiming Gang, and Yi-Ting Chen. The H3D Dataset for Full-Surround 3D Multi-Object Detection and Tracking in Crowded Urban Scenes. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 9552–9557. IEEE, 2019. doi: 10.1109/ICRA.2019.8793925. URL https://doi.org/10.1109/ICRA.2019.8793925.

[47] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum PointNets for 3D Object Detection From RGB-D Data. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 918–927. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00102. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Qi_Frustum_PointNets_for_CVPR_2018_paper.html.

[48] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 77–85. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.16. URL https://doi.org/10.1109/CVPR.2017.16.

[49] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5099–5108, 2017. URL http://papers.nips.cc/paper/7095-pointnet-deep-hierarchical-feature-learning-on-point-sets-in-a-metric-space.

[50] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern*

Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pages 779–788. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.91. URL https://doi.org/10.1109/CVPR.2016.91.

[51] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 91–99, 2015. URL http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.

[52] Thomas Roddick, Alex Kendall, and Roberto Cipolla. Orthographic Feature Transform for Monocular 3D Object Detection. In *30th British Machine Vision Conference 2019, BMVC 2019, Cardiff, UK, September 9-12, 2019*, page 285. BMVA Press, 2019. URL https://bmvc2019.org/wp-content/uploads/papers/0328-paper.pdf.

[53] Francisca Rosique, Pedro J. Navarro, Carlos Fernández, and Antonio Padilla. A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research. *Sensors*, 19(3):648, 2019. doi: 10.3390/s19030648. URL https://doi.org/10.3390/s19030648.

[54] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 770–779. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00086. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Shi_PointRCNN_3D_Object_Proposal_Generation_and_Detection_From_Point_Cloud_CVPR_2019_paper.html.

[55] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-YOLO: Real-time 3D Object Detection on Point Clouds. *Computing Research Repository CoRR*, abs/1803.06199, 2018. URL http://arxiv.org/abs/1803.06199.

[56] Andrea Simonelli, Samuel Rota Bulò, Lorenzo Porzi, Manuel Lopez-Antequera, and Peter Kontschieder. Disentangling Monocular 3D Object Detection. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1991–1999. IEEE, 2019. doi: 10.1109/ICCV.2019.00208. URL https://doi.org/10.1109/ICCV.2019.00208.

[57] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

[58] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 945–953. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.114. URL https://doi.org/10.1109/ICCV.2015.114.

[59] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. *Computing Research Repository CoRR*, abs/1912.04838, 2019. URL http://arxiv.org/abs/1912.04838.

[60] Shrihari Vasudevan. Data fusion with Gaussian processes. *Robotics Auton. Syst.*, 60(12):1528–1544, 2012. doi: 10.1016/j.robot.2012.08.006. URL https://doi.org/10.1016/j.robot.2012.08.006.

[61] Jingdong Wang, Zhen Wei, Ting Zhang, and Wenjun Zeng. Deeply-Fused Nets. *Computing Research Repository CoRR*, abs/1605.07716, 2016. URL http://arxiv.org/abs/1605.07716.

[62] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep Parametric Continuous Convolutional Neural Networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2589–2597. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00274. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Wang_Deep_Parametric_Continuous_CVPR_2018_paper.html.

[63] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark E. Campbell, and Kilian Q. Weinberger. Pseudo-LiDAR From Visual Depth Estimation: Bridging the Gap in 3D Object Detection for Autonomous Driving. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 8445–8453. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00864. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Wang_Pseudo-LiDAR_From_Visual_Depth_Estimation_Bridging_the_Gap_in_3D_CVPR_2019_paper.html.

[64] Zhixin Wang and Kui Jia. Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection. *Computing Research Repository CoRR*, abs/1903.01864, 2019. URL http://arxiv.org/abs/1903.01864.

[65] WAYMO. Waymo Open Dataset: An autonomous driving dataset, 2019. URL https://www.waymo.com/open.

[66] Wikipedia contributors. Alexnet — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=AlexNet&oldid=960230052. [Online; accessed 6-June-2020].

[67] Wikipedia contributors. Uncertainty quantification — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=Uncertainty_quantification&oldid=955461996. [Online; accessed 12-June-2020].

[68] Wikipedia contributors. Region of interest — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=Region_of_interest&oldid=962514676. [Online; accessed 4-August-2020].

[69] Wikipedia contributors. Canny edge detector — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=Canny_edge_detector&oldid=968976334. [Online; accessed 4-August-2020].

[70] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 1887–1893. IEEE, 2018. doi: 10.1109/ICRA.2018.8462926. URL https://doi.org/10.1109/ICRA.2018.8462926.

[71] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Data-driven 3D Voxel Patterns for object category recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1903–1911. IEEE Computer Society, 2015. doi: 10.1109/CVPR.2015.7298800. URL https://doi.org/10.1109/CVPR.2015.7298800.

[72] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Subcategory-Aware Convolutional Neural Networks for Object Proposals and Detection. In *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017*, pages 924–933. IEEE Computer Society, 2017. doi: 10.1109/WACV.2017.108. URL https://doi.org/10.1109/WACV.2017.108.

[73] Kui Xu. kuixu/kitti_object_vis: Kitti object visualization (birdview, volumetric lidar point cloud). https://github.com/kuixu/kitti_object_vis, July 2020. (Accessed on 12/07/2020).

[74] Jianru Xue, Jianwu Fang, Tao Li, Bohua Zhang, Pu Zhang, Zhen Ye, and Jian Dou. BLVD: Building A Large-scale 5D Semantics Benchmark for Autonomous Driving. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 6685–6691. IEEE, 2019. doi: 10.1109/ICRA.2019.8793523. URL https://doi.org/10.1109/ICRA.2019.8793523.

[75] Yan Yan, Yuxing Mao, and Bo Li. SECOND: Sparsely Embedded Convolutional Detection. *Sensors*, 18(10):3337, 2018. doi: 10.3390/s18103337. URL https://doi.org/10.3390/s18103337.

[76] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. IPOD: Intensive Point-based Object Detector for Point Cloud. *Computing Research Repository CoRR*, abs/1812.05276, 2018. URL http://arxiv.org/abs/1812.05276.

[77] Yurong You, Yan Wang, Wei-Lun Chao, Divyansh Garg, Geoff Pleiss, Bharath Hariharan, Mark E. Campbell, and Kilian Q. Weinberger. Pseudo-LiDAR++: Accurate Depth for 3D Object Detection in Autonomous Driving. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BJedHRVtPB.

[78] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access*, 8:58443–58469, 2020. doi: 10.1109/ACCESS.2020.2983149. URL https://doi.org/10.1109/ACCESS.2020.2983149.

[79] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4490–4499. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00472. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Zhou_VoxelNet_End-to-End_Learning_CVPR_2018_paper.html.