Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Comparison between Fixed-weight Neural Network and Bayesian Neural Network for 3D Object Detection in Autonomous Driving

Jaswanth Bandlamudi

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 Küstenmacher
M. Sc. Luis Octavio Arriaga Camargo

November 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	Jaswanth Bandlamudi

Abstract

Deep learning methods are capable of learning complex representations from the provided data, thereby solving various real-world problems. Though these methods provide high-quality predictions, they do not provide a reliable estimate of the confidence or uncertainty in the prediction. This uncertainty prediction is especially important in safety-critical applications. In this research and development project, we study an object detection problem in the perception part of the autonomous driving stack. We also investigate whether a reliable uncertainty estimate can be provided as an output of the model.

We precisely chose to solve the problem of 3D object detection using Frustum-PointNet architecture which consumes inputs from LiDAR and camera. The 3D object detection model is trained and tested using the Lyft dataset. The model performed well on the test data with the average box accuracy values of 56.46%, 46.33% and 38.61% on easy, medium, and hard IoU limits respectively.

Additionally, we investigated if a bayesian model of the Frustum-PointNet can be modelled using the Flipout layers from Tensorflow probability library, and the uncertainty in detection is measured using mean of Shannon entropy and the total variance. Surprisingly, the bayesian approach has under-fitted on the same frustums used to train the fixed-weight model of the Frustum-PointNet.

We converted the last two stages of the Frustum-PointNet into the bayesian models and observed that the model did fit well and the performance of the model is similar to the fixed-weight model. To extract epistemic uncertainty, we performed ten Monte-Carlo sampling inference runs and average of Shannon entropies and total variance of the detections is used as the epistemic uncertainty. From the experimental values, the model detected cars with lower epistemic uncertainty, cyclists and pedestrians with higher uncertainty.

Acknowledgements

I sincerely want to thank all my supervisors for the valuable inputs and the freedom provided. A huge credit is due to my third supervisor, Luis Octavio Arriaga Camargo, for his technical insights and directions provided.

A big thanks to my parents, for supporting me through these rough times. I would also like to thank my beloved friends, Santosh Reddy, Sasikiran, and others for reading and reviewing my work. I would like to extend my gratitude to all the front-line health workers, doctors, nurses, administration staff and to all the first responders during the corona pandemic times.

Contents

List of Figures

List of Tables

1

Introduction

Neural Networks (NN) popularity is on the rise due to their effectiveness in learning features from high dimensional data like images [1, 2] and point clouds [3, 4, 5, 6]. Deep learning is a subset of machine learning dealing with neural networks provides black-box model solutions in many fields like Medical Imaging, Autonomous driving, etc. The current advancements in Deep Neural Networks (DNNs) are attributed to Convolutional Neural Networks (CNNs) [7, 8, 9], which are also known as shift and space invariant neural networks.

1.1 Motivation

Autonomous driving is one of the fields which is revolutionized by DNNs [9, 7]. The application of deep learning in the autonomous driving stack can be seen in Figure 1.1. In this work, we concentrate on the 3-Dimensional (3D) object detection task in perception part of the autonomous driving stack.

1.1.1 3D Object Detection

Perceiving the outside environment consists of tasks like lane detection, vehicle detection and Vulnerable Road Users (VRU)[1] detection. As the vehicles are operating

[1]Vulnerable Road Users (VRU) are defined in the European Union Intelligent Transportation Systems Directive as "non-motorised road users, such as pedestrians and cyclists as well as motorcyclists and persons with disabilities or reduced mobility and orientation".

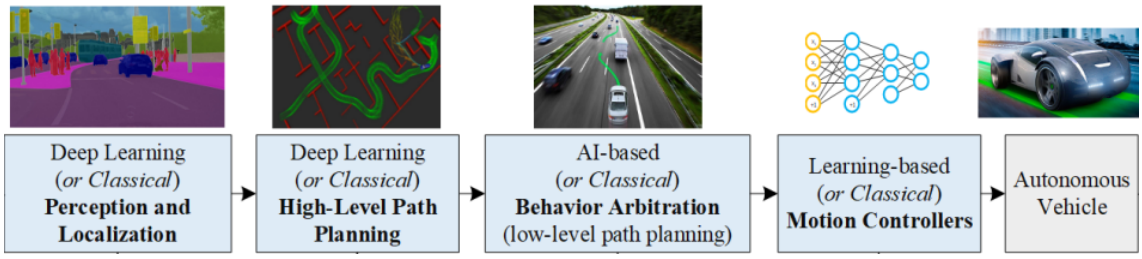


Figure 1.1: Autonomous driving stack [10, pg.4], in the first stage the environment is perceived and the agent positions are regressed. In the second stage, the vehicle performs motion planning and in the third stage based on the previous positions of the agent the feature positions are predicted. In the final stage, motion control commands are regressed which are actual inputs to the vehicle using this software stack

in the 3D world, for better autonomous operation, the perception part assists in knowing the 3D position of the agents[2] around [11]. This task has to be carried out with high accuracy and should be robust to adverse weather and lighting conditions.

1.1.2 Uncertainty Quantification in Neural Networks

There are failures while using DNNs in safety-related tasks. For example, the image-classification model by Google had wrongly classified photos of some humans as gorillas [12]. Another such failure was reported, when a self-driving car from Uber hit a jay-walking pedestrian and resulted in death [13]. Both of these examples show that DNNs are prone to failures. Hence, their usage in safety-critical operations is hindered. To alleviate this problem, researchers proposed the usage of uncertainty [14, 15] as a measure of confidence in the output. This uncertainty in confidence can be extracted using Bayesian Neural Networks [16, 17] and Ensemble methods [18]. The deep learning models used for 3D object detection must include uncertainty as one of their outputs. This measure of uncertainty can act as a valuable input to the later stages of the autonomous driving stack shown in Figure 1.1.

[2]Agents in this work refers to car, pedestrians and cyclist present in the operating environment

1.1.3 Bayesian Neural Networks

Bayesian Neural Networks (BNNs) [17, 19, 16] are a class of neural networks in which the weights are initialized as a distribution $p(w)$ and training includes obtaining a posterior distribution over weights represented by $p(w|D)$, where D is the dataset on which the model is trained. $p(w|D)$ captures the model uncertainty. Additionally, BNNs also permit the knowledge of uncertainty source [20].

1.2 Challenges and Difficulties

The challenges and difficulties of 3D object detection task and BNNs models are outlined in the following sections.

1.2.1 3D Object Detection

- To localize an object in the 3D world, depth information is needed. Hence, the use of cameras doesn't provide satisfactory results.
- The model should be robust to all weather and road conditions.
- The model should be capable of estimating the full object dimensions even in case of a partial occlusions.
- Estimating the center and size of the object with minimum temporal data is hard.

1.2.2 BNNs for 3D Object detection

- Bayesian Neural Networks allow for calculation of epistemic uncertainty i.e. uncertainty due to weights only.
- Uncertainty quantification in multi-stage models for 3D object detection has very limited literature.
- The implementations on simpler classification models don't work well on complex models.

- Majority of 3D object detection tasks are formulated as a combination of classification and regression tasks, whose uncertainty measures are different. Hence, building the Bayesian 3D object detection model is challenging.
- Evaluation criteria for uncertainty measurement is not available for benchmarking.

1.3 Problem Statement

In this work our objective is to

- Train a 3D object detection model on Lyft dataset.
- Convert the model into a Bayesian network
- Train the Bayesian model and evaluate its performances

Our problem statement includes, comparing the performance of the fixed-weight neural network with BNNs in the case of a complex, multi-stage neural network.

The 3D object detection model is trained on Lyft dataset [21] with images from all the six cameras. The accuracy of detections is judged by performing a test similar to KITTI benchmark test [11] on Lyft dataset. Convert the fixed-weight Frustum-PointNet model into a Bayesian model to extract the uncertainty of neural networks in the 3D object detection output from the model.

This work investigates the following research questions:

- Does the model perform at the same level as trained on KITTI dataset when trained on Lyft dataset?
- Can a BNNs version of the 3D object detection be modeled and trained?
- Can the outcomes of the Bayesian model performance be explained ?

Along with answering the above questions we also provide literature review of the 3D object detection methods and uncertainty quantification techniques using BNNs.

2

3D Object Detection

The detection task is key to know the type and position of the agents in the perceived world. This detection is either done in a 2-Dimensional (2D) perspective world or in an original 3-Dimensional (3D) world. We chose to solve the task of 3D object detection as the later stages in the autonomous driving stack as shown in Figure 1.1, need this information to better localize the object with depth accuracy as well.

2.1 Theoretical Background

The main aim of the 3D object detection model is to take RGB image along with depth information from sensors like Light Detection and Ranging (LiDAR), cameras, Radio Detection and Ranging (RADAR), and output a 3D bounding box along with class labels of the agents in the perceived environment. It is also referred to as amodal 3D object detection [7], which means the model should fit a 3D bounding box for the whole object even though only a part of it is visible.

As shown in Figure 2.1, there are multiple agents in the scene some are visible partially. But, knowing their positions is very important for safe autonomous operation. Hence, amodal object detection is important.

The 3D bounding box visually is the smallest possible cuboid placed in the 3D world and encompassing all the parts of an agent and is parameterized by a vector $[x,\ y,\ z,\ l,\ w,\ h,\ \theta]$ and is illustrated in Figure 2.2 [22].

- (x, y, z) represents the center of the bounding box.

Figure 2.1: Sample image from Lyft dataset [21], showing many agents are not fully visible in the scene but still needed to be detected

- (l, w, h) represents the length, width, and breadth of the bounding box.
- θ is the yaw angle of the bounding box (pitch and roll are considered to be not important for automotive applications [23]).

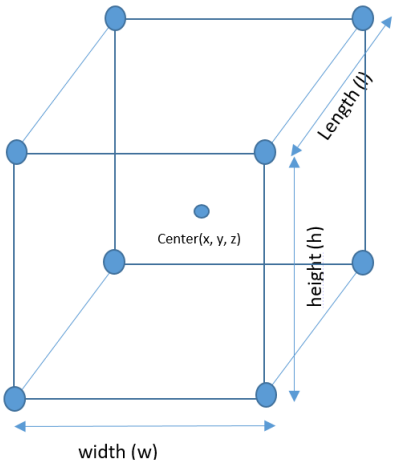


Figure 2.2: 3D bounding box representation with center of the bounding box and its dimensions labelled

2.2 Related Work

The input to the model is generally collected using individual or from multiple sensors like monocular cameras, stereo cameras, RADARs, and LiDARs. But most of the works are done using the data collected by cameras and LiDAR sensors. Various methods proposed for solving 3D object detection problem along with their drawbacks are discussed in Sections 2.2.1, 2.2.2 and 2.2.3.

2.2.1 Camera based Methods

On the subject of 3D object detection methods utilizing cameras, well-known methods use either a monocular camera or a stereo camera for data collection.

2.2.1.1 Monocular Camera

In Mono3D [24] by *Chen et al.*, the methodology proposed is to provide object proposals and use a Convolution Neural Network (CNN) to regress object class and the 3D bounding boxes. The proposals are generated from the context-aware features, segmentation label, and local priors. Deep3DBox [25] proposed by *Mousavian et al.*, uses the fact that the 2D bounding box is nothing but a perspective projection of a 3D bounding box on an image from this information. The authors of Deep3DBox also proposed to use a matured 2D object detector combined with a Deep CNN to additionally regress a 3D bounding box along with the yaw angle.

Deep-MANTA [26] by *Chabot et al.*, which is a multi-stage monocular image-based 3D object detection method proposed for vehicle detection. The first stage consists of CNN, which regresses the 2D bounding box, pixel coordinates of various vehicle parts, and dimensions of the 3D box. The second stage consists of 3D vehicle models, bounding box dimensions, and 3D coordinates of vehicle parts. The available features are used in this stage to compare the values regressed in the first stage and then the true 3D bounding box is estimated by reducing the errors from the proposal to the 3D dataset in the second stage. This approach had resulted in State-Of-The-Art (SOTA) results on the KITTI leader board.

2.2.1.2 Stereo Camera

A stereo camera uses a concept of stereo matching in which the displacement of the corresponding points is used to reconstruct the scene in the form of a depth map. The information provided by the depth map can be further used for 3D bounding box estimation or to generate a pseudo point cloud.

In a work proposed by *Chen et al.*, in the 3DOP [27] method, the point cloud generated is used to extract hand-crafted features and these hand-crafted features are used to total the 3D sliding windows in an energy minimization framework. The topmost K scoring windows are then fed to a modified Fast-RCNN [1] by *Ren et al.*, which regresses the 3D detection. A similar method with a modified ranking algorithm using dual-stream CNN is used in DeepStereoOP [28] by *Pham et al.* and had outperformed the initial 3DOP [27] on the KITTI object detection benchmark [11].

2.2.1.3 Drawbacks

- These methods need a very precise calibration because of heavy dependence on the corresponding points and their relative displacement.

- The depth calculation algorithms are computationally expensive and need more processing power.

- These methods are generally very slow in operation, the better performing DeepStereoOP method needs 3 seconds to process a single image which is not feasible for real-time operation.

2.2.2 LiDAR based Methods

This category of 3D object detection networks aims to represent point clouds similar to images, so that the mature CNN-based object detection models can be used to regress the necessary dimensions. There are two methods in this category which are volumetric grid-based (Voxel-based) and projection maps based.

2.2.2.1 Volumetric grid methods

In the case of images, a convolution operation consists of a two-dimensional kernel and is convolved with an image input which is a 2D input. But point clouds are unstructured and are 3D inputs. Hence, these methods convert the point cloud into a 3D voxel structure and apply a 3D kernel. The voxelization process of the point cloud of the Bunny rabbit can be seen in Figure 2.3.

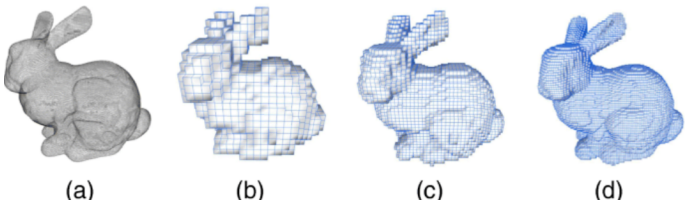


Figure 2.3: Image showing the voxelization of stanford bunny, a) Point cloud, b-d) Voxelization at different resolutions [29]

Vote3D [30] uses voting in order to determine if a cell is occupied or not. For detection purposes, a search in the voxel space is performed and for classification, a linear Support Vector Machine (SVM) is used. Vote3Deep is another method that extends on Vote3D by processing the whole point cloud using a neural network and fixed class sizes are used based on the classification output.

The third method reported by *Zhou et al.* is Voxelnet [29], It consists of three stages a voxel feature extraction layer which is a modified PointNet [3] and outputs a sparse 4D voxel-wise feature vector of pre-defined size. These outputs are fed into 3D convolution layers which results in a 3D feature map and the output is processed by a Region Proposal Network (RPN). The outputs of RPN are the confidence scores and the 3D box anchors.

SECOND [31] by *Yan et al.*, is another method proposed as an improvement to the VoxelNet by using sparse convolution instead of a 3D convolution. This resulted in an improvement in the benchmark results and also an increase in the processing speeds is observed.

2.2.2.2 Projection based methods

Another method to handle the LiDAR point cloud data is a projection map, as a front view map or as a Bird-Eye View (BEV) image rather than as voxels. The projected maps can be used as an image and required operations can be carried on them.

One work proposed using the project map using a Fully Convolutional Network (FCN) is by *Li et al.* [32]. The LiDAR projection encodes the point clouds distance from the ground and the range of the points. This map is processed using a 2D FCN to downsample the provided map and then up samples to regress objectness, which defines whether the point is a vehicle part or not and bounding box predictions and also represents the 3D bounding box edges. This results in multiple estimates of bounding boxes. Hence, the authors deployed a non-maximum suppression to output the best fit for a bounding box. This is an end-to-end model trained on the KITTI dataset.

Another method Complexer-YOLO [33] proposed by *Simon et al.*, uses a BEV projection of the point cloud and uses a modified Fast-RCNN model as the backbone in order to regress the 3D bounding box details. This method has achieved a benchmark 50 fps processing speed but with a deteriorated orientation angle accuracy. These projection methods have a pre-requisite of a dense point cloud and do not perform well on a sparse LiDAR point cloud. Hence *Beltrán et al.* [34] proposed a method in which they tried to normalize the point cloud using the calibration parameters of the LiDAR in use which resulted in a better representation of the point cloud and also provided better generalization ability to the network trained.

2.2.2.3 Drawbacks

- Volumetric methods use 3D convolution kernels which require more operations than a 2D convolution kernel and need more processing power. Also, these methods suffer from higher inference times.
- Voxelization might result in information loss at the bin intersection.
- Many methods fix the sizes of the object detections which in hindsight hinders the detection performance as the sizes might be very irregular.

2.2.3 Fusion based Methods

The best-performing methods for 3D object detection are majorly fusion-based because of the dense input similar to human vision provided by the camera and the depth information provided by the LiDAR point cloud and combining both of them helps to understand the depth of the scene as well as its semantics. Based on the stage at which fusion happens there are early, deep, and sequential fusion methods.

2.2.3.1 Early fusion methods

In early fusion methods, either the raw data collected by the sensors is combined or the processed data are combined. Early fusion is used in the work reported by *Du et al.* [35], a pre-trained 2D detection network proposes a candidate for a bounding box this information is fused with the 3D features from the point cloud and the 3D parameters are regressed with it. In order to extract the 3D features from the point cloud, the authors used pre-existed vehicle models for various types. From the 3D parameters regressed, multiple 3D bounding boxes are generated. A dual-stage CNN is used to extract the true bounding box and their respective objectness score. This method is only used to detect cars, as they are the most common agents present in the real world with less variance in class sizes.

2.2.3.2 Deep fusion methods

Deep fusion methods perform the fusion of the data or the features extracted multiple times in the model. It can also be considered as an early fusion but the fusion happening multiple times.

There are multiple well-performing models using deep fusion. One among those models is MV3D [6] by *Chen et al.*, in which three inputs RGB image, point cloud projected onto the front view, and BEV point cloud are consumed. The front view projection provides information about height, intensity, and distance and the BEV images hold density, intensity, and height information. Then an RPN generates 3D object proposals from BEV only and these proposals are projected to multiple views. As the features extracted from the front view and BEV have various sizes, a region

of interest pooling layer is applied and the output is fed to a deep fusion network which generates 3D bounding box details and the objectness score is regressed.

Another method AVOD is proposed by *Ku et al.* [5], a deep continuous fusion paradigm is utilized where a 2D BEV projection map is fed to a CNN and another CNN extracts feature maps from the monocular image. These features are fed to a parallel network, which consists of an RPN and a detection network similar to the model proposed in MV3D [6] but the main difference is that the RPN consumes the features from BEV and the monocular images as well.

2.2.3.3 Sequential fusion methods

In sequential fusion methods, the proposals generated from one modality are used to extract the necessary data about 3D size or to improve the accuracy of the detection.

This paradigm is used in Frustum-PointNet proposed by *Qi et al.* [36], where a 2D object detector is used to generate a detection in 2D and this 2D detection is used to reduce the search space in order to process only the required part of the point cloud where the object presence is given. This method has multiple stages which are frustum proposal, 3D instance segmentation, and 3D amodal bounding box estimation.

In the frustum proposal stage, the 2D object detection proposal is utilized to generate a frustum which represents a volume of the object under consideration. in the next stage, the frustum generated is fed to an encoder-decoder based instance segmentation network which assigns the labels to the points in the point cloud based on the class label. This helps in finding the objects when there is a presence of partial occlusion. The segmented point cloud is passed to a PointNet based amodal bounding box estimation which regresses the 3D bounding boxes. As the amount of point cloud that's processing is reduced this model is fast and requires less processing power. This method is further extended by *Wang et al.* to develop Frustum Convnet [37] which uses a 3D Fully Connected CNN in place of PointNet in segmentation and bounding box regression network and a multi-view model of the Frustum-PointNet by Cao [38].

2.2.3.4 Drawbacks

- The majority of models used KITTI dataset [11] to train the neural networks which does not consist of varying weather conditions.
- None of the methods have ability to quantify uncertainty on the detections made.

3

Uncertainty Quantification

Deep Neural Networks (DNNs), since their inception have out-performed the conventional statistical approaches in both accuracy and generalizability. But they lack the ability to quantify uncertainty in the outputs. The classification models provide predictive probabilities which are softmax outputs of a model. But the softmax probabilities are wrongfully interpreted as an uncertainty metric. In this Chapter, we portray the neural network as a Bayesian approximation model by utilizing variational inference to train and learn a posterior distribution of weights given the dataset.

Uncertainty of a neural network about the predicted output will be a valuable input to the decision-making process, this uncertainty is introduced mainly by the usage of noisy data for training the neural networks and due to the randomness in choosing the model parameters. The first example in Section 1.1.2 [12] represents the failure due to lack of training data and is referred to as *knowledge uncertainty* or *epistemic uncertainty*. While the second example in Section 1.1.2 [13], represents the failure because of noise in the sensors measurement which is referred to as *data uncertainty* or *aleatoric uncertainty*. Different methods in quantifying uncertainty are illustrated in Figure 3.1.

3.1 Bayesian Neural Networks

In conventional DNNs, weights are modeled as point estimates which is unjustified. These point estimates resulted in the network being deterministic for provided inputs.

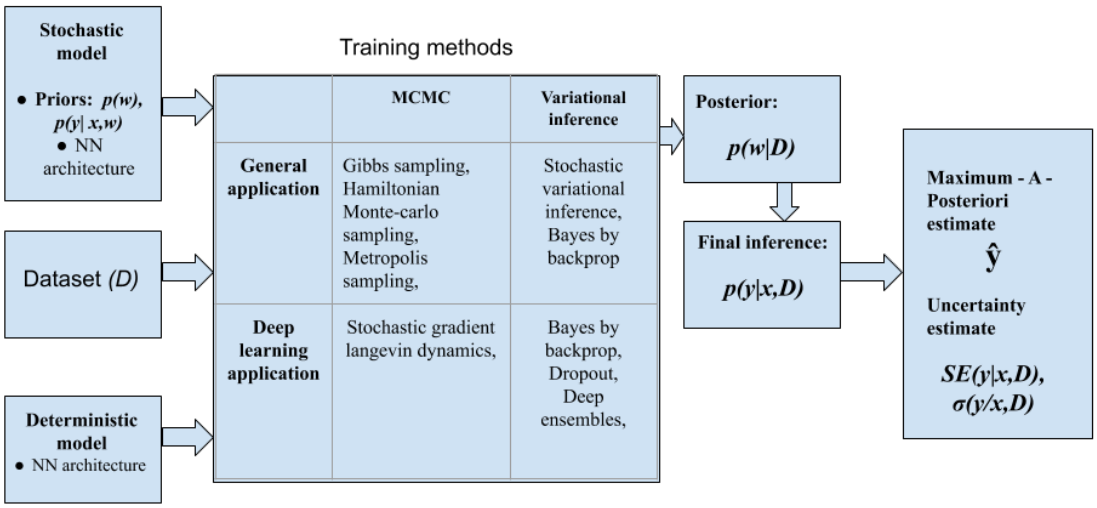


Figure 3.1: Different methods to quantify uncertainty in neural networks

Whereas, Bayesian Neural Networks (BNNs) models the weights as a distribution of weights defined over the given dataset D. This helps in extracting uncertainty by sampling weights from the posterior distribution of weights.

3.1.1 Estimating Posterior Weight Distribution

Let us consider a dataset $D = \left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(N)}, y^{(N)}\right)\right\}$, and w be the weights of the neural network that are to be learnt from the dataset D provided. The posterior distribution $p(w \mid D)$ is obtained by applying Bayes rule as in Equation 3.1

$$p(w \mid D) = \frac{p(D \mid w)p(w)}{p(D)} \tag{3.1}$$

- $p(D \mid w)$ is the likelihood of the dataset provided the weights are known this can also be written as shown in Equation 3.2

$$p(D \mid w) = \prod_{n=1}^{N} p\left(y^{(n)} \mid w, x^{(n)}\right). \tag{3.2}$$

  – In case of regression, the likelihood can be approximated to a Gaussian Distribution $p(y \mid w, x) = N(y; \mu, \sigma)$.

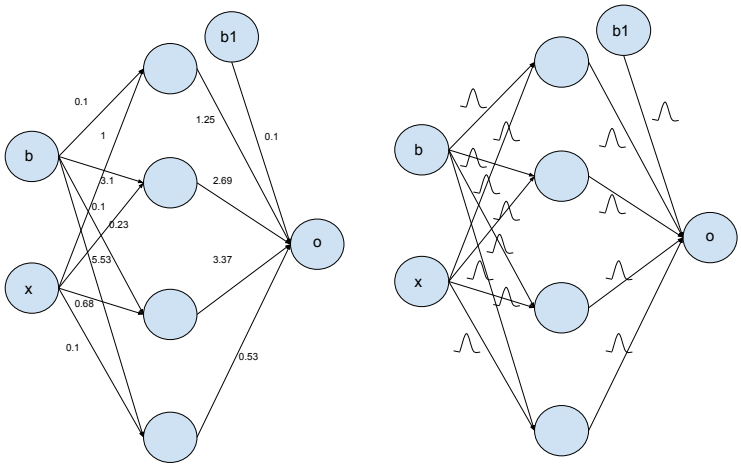


Figure 3.2: Representation of Bayesian neural network showing the weights modelled as distribution generally a gaussian distribution that is parameterized by a mean (μ) and a variance (σ)

  - In case of classification, the softmax outputs of the neural network prediction can be used as likelihood and can be represented as $p(y = c \mid w, x) = f^w(x)^c$.

- $p(w)$ is our belief about the distribution of weights which is generally considered to be Gaussian distribution.

- The denominator $p(D)$ is calculated using $p(D) = \int p(D \mid w)p(w)dw$. But, calculating this is not manageable because of the integration over all the weights. The DNNs generally has millions of weights and integrating all over them is tedious and is computationally intensive.

This sort of probabilistic approach in providing a better way towards inferencing from the unknown input data, while also understanding the amount of variance present in these predictions helps in better and safer usage of DNNs in real-world applications. The model as well as a provision of choosing a prior distribution $p(w)$ which defines our initial belief of the weight parameters. A good initial choice of this

prior over the parameters will also help in balancing out the model complexity to the best fit possible.

As already stated calculating $p(D)$ is not practically possible; hence the posterior $p(w \mid D)$ is approximated using sampling methods like Metropolis-Hastings and Markov-Chain-Monte-Carlo (MCMC) sampling methods. Though these methods have the ability to provide an unbiased approximation, they are slow to converge. Researchers [14, 15, 16] proposed another method called Variational Inference (VI) which though provides a biased approximation but converges faster.

3.1.2 Variational Inference

Graves et al, [39] proposed that the posterior distribution $p(w \mid D)$ can be approximated to be a parameterized distribution $q(w)$ and optimize those distributions over the weight parameters to obtain the best approximation.

The approximation is judged using Kullback-Leibler (KL) divergence [40] as shown in Equation 3.3.

$$\mathrm{KL}(q(x)\|p(x)) \equiv \mathbb{E}_{q(x)}\left[\log \frac{q(x)}{p(x)}\right] = \int q(x) \log \frac{q(x)}{p(x)} dx \tag{3.3}$$

The posterior calculation problem is now posed as an optimization problem of minimizing the KL-divergence between q(w) and $p(w \mid D)$.

$$\begin{aligned}
\mathrm{KL}(q(w)\|p(w \mid D)) &= \mathbb{E}_{q(w)}\left[\log \frac{q(w)}{p(w \mid D)}\right] \\
&= \int q(w) \log \frac{q(w)}{p(w \mid D)} dw \\
&= \int q(w) \log \frac{q(w)p(D)}{p(D \mid w)p(w)} dw \\
&= \int q(w) \log \frac{q(w)}{p(w)} dw + \int q(w) \log p(D) dw - \int q(w) \log p(D \mid w) dw \\
&= \mathrm{KL}(q(w)\|p(w)) + \log p(D) - \mathbb{E}_{q(w)}[\log p(D \mid w)]
\end{aligned} \tag{3.4}$$

In the Equation 3.4[1] the logarithm value $\log p(D)$ is constant, which means minimizing the distance between two distributions is down to maximizing the $\mathbb{E}_{q(w)}[\log p(D \mid w)] - \mathrm{KL}(q(w)\|p(w))$. This is referred to as Evidence Lower Bound (ELBO).

Hence, VI poses the problem of finding the $p(w \mid D)$ into a maximizing ELBO problem. Once the optimization problem is defined we can use Bayes by backprop by *Shridhar et al.* [41], to extract the best approximation of posterior which maximizes the ELBO.

3.1.3 Bayes by Back-propagation

Bayes by Back-propagation was first introduced by *Blundell et al.* [15], which allowed the usage of current back-propagation algorithm work by replacing the weights with a parameterized distribution. In this method, we try to estimate posterior distribution $(q(w)\|D)$, the chain-rule which consists majorly of product and differentiation is applied by sampling a single instance of weight and modify it as per the loss function in Equation 3.4.

Hence, the optimal parameters θ^{opt} which consists of respective mean and variance (μ, σ) are defined as

$$\begin{aligned} \theta^{opt} &= \mathrm{argmin}_\theta \, KL[q(\mathbf{w} \mid \theta)\|P(\mathbf{w} \mid \mathfrak{D})] \\ &= \mathrm{argmin}_\theta \int q(\mathbf{w} \mid \theta) \log \frac{q(\mathbf{w} \mid \theta)}{P(\mathbf{w}P(\mathfrak{D} \mid \mathbf{w}))} \\ &= \mathrm{argmin}_\theta \, \mathbf{KL}[q(\mathbf{w} \mid \theta)\|P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathfrak{D} \mid \mathbf{w})] \end{aligned} \tag{3.5}$$

Let us consider,

$$f(w, \theta) = \log \frac{q(\mathbf{w} \mid \theta)}{P(\mathbf{w})P(\mathfrak{D} \mid \mathbf{w})}$$

,

$$\implies f(\mathbf{w}, \theta) \approx \sum_{i=1}^{n} \log q\left(\mathbf{w}^{(i)} \mid \theta\right) - \log P\left(\mathbf{w}^{(i)}\right) - \log P\left(\mathfrak{D} \mid \mathbf{w}^{(i)}\right)$$

, in which w^i are the samples drawn from the variational posterior $q(w)$.

[1] All the formulae and derivations are adapted versions of the explanation from Master Thesis on Bayesian Convolutional Neural Networks (Bayes-CNNs) by *Shridhar et al.* [19, pg.24]

Substituting this in Equation 3.5 we get,

$$\theta^* = \text{argmin}_\theta \, \mathbb{E}_{q(\mathbf{w}|\theta)} f(\mathbf{w}, \theta) \tag{3.6}$$

But, the above object function cannot be used with a standard back-propagation algorithm. This is due to the inability to achieve a differentiation of the objective function. Hence, *Kingma et al* [42] proposed a reparameterization trick in which the gradients are calculated through general back-propagation and also reducing the variance. Reparameterization, is the ability to reformulate a statistical problem into its equivalent form. In our case, we try to formulate the uncertainty over all parameters to a local uncertainty. This local uncertainty does not depend on the other parameters.

Using reparameterization, we mention the stochastic weight represented by mean μ and σ^2 as a function of another variable sampled from a gaussian ditribution $N(0, 1)$. Hence, $w = f(\varepsilon) = \mu + \sigma \bullet \varepsilon$ where $\varepsilon \models N(0, 1)$. Now instead of random sample from the posterior distribution $q(w)$, we sample from a gaussian distribution with mean 1 and variance 0. The updates are done as follows

$$\begin{aligned}
\Delta\mu &= \tfrac{\delta f}{\delta w} + \tfrac{\delta f}{\delta \mu} \\
\Delta\sigma &= \tfrac{\delta f}{\delta w}\tfrac{\varepsilon}{\sigma} + \tfrac{\delta f}{\delta \sigma} \\
\mu &= \mu - \alpha\Delta\mu \\
\sigma &= \sigma - \alpha\Delta\sigma \\
\theta_o pt &= (\mu_o pt, \sigma_o pt)
\end{aligned} \tag{3.7}$$

In case of CNNs, *Shridhar et al.* proposed to sample layer activations instead of weights, it results in a estimated posterior distribution $q_\theta\left(\theta_{\text{ijhw}} \mid Data\right)$.

$$q_\theta\left(\theta_{\text{ijhw}} \mid Data\right) = \nu\left(\mu_{\text{ijhw}}, \alpha_{\text{ijhw}}\mu^2_{\text{ijhw}}\right)$$

i is the input layer, j is the output layer, (w, h) is the shape of the input tensor.

The resultant equation for sampling the activation is written as

$$b_j = A_{\mathrm{i}} * \mu_{\mathrm{i}} + \varepsilon_j \cdot \sqrt{A_{\mathrm{i}}^2 * (\alpha_{\mathrm{i}} \cdot \mu_{\mathrm{i}}^2)} \tag{3.8}$$

b is the layer activation, $\varepsilon_j \epsilon N(0,1)$, A is the receptive field area. A Bayesian Convolution Neural Network can be seen in Figure 3.3

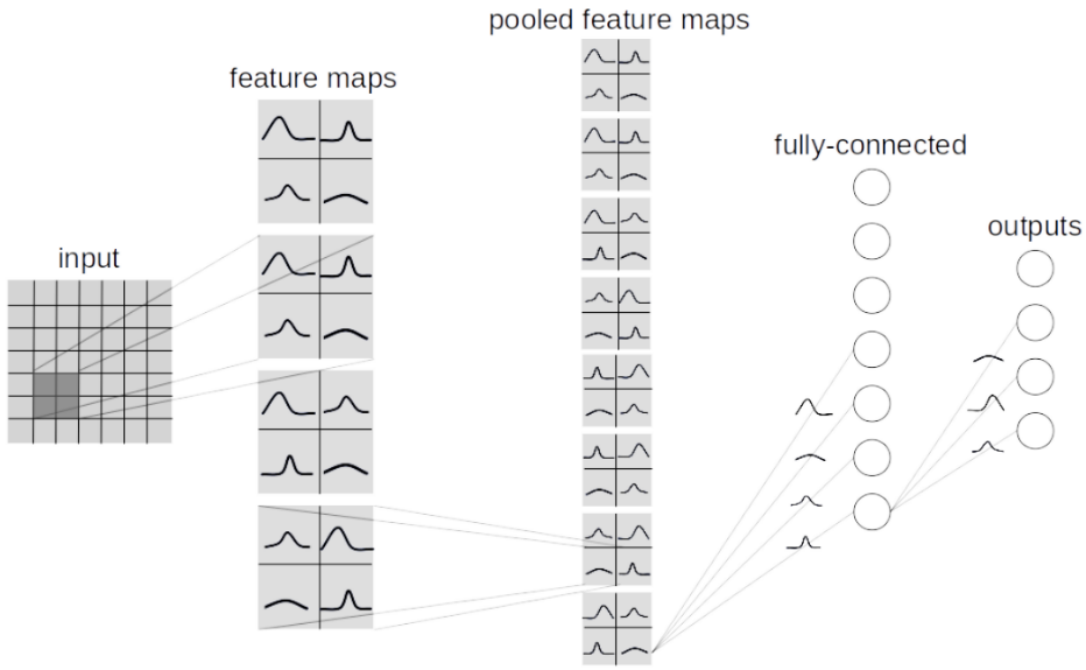


Figure 3.3: Bayesian Convolution Neural Network which depicts the probabilistic feature maps generated from probabilistic kernels as proposed by *Shridhar et al,* [19]

3.1.4 Weight Sampling Methods

In order to perform inference for a new input x^* given a VI approximation $q(w)$ there is a need to sample the weights from the approximation for every inference run performed in order to extract $p\left(y^* \mid x^*, w\right)$. There are multiple weight sampling methods proposed. First of the methods is *Gaussian perturbations* [42] by *Kingma et al.*, where ΔW_{ij} are sampled from the Gaussian distributions modelled as $\mathcal{N}\left(\bar{W}_{ij}, \sigma_{ij}^2\right)$ and using reparameterization trick the weight can be represented as $W_{ij} = \bar{W}_{ij} + \sigma_{ij}\epsilon_{ij}$ where ϵ_{ij} is a value sampled from a normal distribution with mean 0 and variance

1. Dropout is another method in which weights are randomly dropped based on a pre-set drop rate which can be set as hyper-parameter of a model. In the case of a dropout rate of 0.5, we can think of this setting as sampling weights where $n_{sampled-weights} = \frac{n_{total-weights}}{2}$. All the above methods suffer from high variance in the gradient estimates. This is instigated by usage of explicit weight samples by all training samples in a mini-batch. More precisely allotment of sampled weights over all the samples in a mini-batch prompts correlations among gradients of every sample, which cannot be fixed by general averaging.

3.1.4.1 Flipout

The higher variance in the previous methods reported is solved using Flipout [16] introduced in *Wen et al.*. This method aims to uncorrelate the gradients between every sample. This work achieved by making some assumptions which are

1. The sampling of different weights is independent of each other.

2. The posterior distribution is symmetric around 0.

Flipout suggests to use a posterior distribution (q_θ) which is jointly used by each and every weight sample on the mini-batch and an element-wise multiplication is performed by another rank-one sign matrix for every sample. A random sign matrix E which is independent of posterior distribution q_θ. The weight sampling is done by multiplying E and the posterior distribution.

$$W = q_\theta . E$$

With this method we can extract a balanced estimator for the loss gradients, also de-correlating the samples achieving a variance decrease on the sequential updates when averaging over a mini-batch during training. After applying flipout a single neuron can be represented as shown in Equation 3.9

$$y_{\mathrm{i}} = \phi\left(W^T x_{\mathrm{i}}\right) = \phi\left(\left(\bar{W} + q_\theta \cdot r_{\mathrm{i}} s_{\mathrm{i}}^T\right)^T x_{\mathrm{i}}\right) = \phi\left(\bar{W}^T x_{\mathrm{i}} + \left(q_\theta^T \left(x_{\mathrm{i}} \cdot s_{\mathrm{i}}\right)\right) \cdot r_{\mathrm{i}}\right) \tag{3.9}$$

3.1.5 Bayesian Inference

With the availability of the approximation $q(w)$ of $p(w \mid D)$ obtained using VI method we can infer an output y^* for any given input x^* by integrating over the weights available as shown in Equation 3.10.

$$p\left(y^* \mid x^*, D\right) \approx \int p\left(y^* \mid x^*, w\right) q(w) dw \tag{3.10}$$

But, as stated previously integrating over all the weights is not practical. Hence, we depend on sampling methods in which a pre-fixed K number of samples (an additional hyper-parameter) w^k are drawn from the learned approximation $q(w)$. Using each samples values w^k a likelihood of $p\left(y^* \mid x^*, w^k\right)$ is calculated. The final prediction is obtained as average of all the predictions from all the samples weights as shown in Equation 3.11.

$$p\left(y^* \mid x^*, D\right) \approx \frac{1}{K} \sum_{k=1}^{K} p\left(y^* \mid x^*, w^k\right) \tag{3.11}$$

3.2 Ensemble Methods

Deep Ensembles is a non-Bayesian approach for uncertainty estimation, where several networks are trained from scratch differently which leads to each network having different weights. The ensembles can be formed from

1. Training different model architectures [15]
2. Multiple checkpoints obtained while training the model [18]
3. Multiple models by training only part of the model [43]

These ensemble of multiple models gives multiple outputs can be used for uncertainty estimation [44, 43].

3.3 Uncertainty Prediction

As already mentioned in the previous sections the predictive uncertainty is classified into epistemic uncertainty and aleatoric uncertainty [44]. Aleatoeric uncertainty

is further classified into Homoscedastic uncertainty, which stays constant for different inputs and Heteroscedastic uncertainty varies with input and some being more affected because of the noise from the environment.

Epistemic uncertainty is estimated to be uncertainty due to model parameters and is quantized by measuring the variance of the approximated posterior weight distribution $p(w \mid D)$. As the variance increases, the distribution becomes more flatter which represents very high epistemic uncertainty, while a lower variance with the highest peak is the result of a lower epistemic uncertainty. The amount of epistemic uncertainty can be alleviated with an increase in a number of observations and an increase in the range of observations of the type of observations in the task involved. Aleatoric uncertainty is uncertainty because of the noise in the inputs fed to the model and is integral in the observations and thus cannot be reduced with additional annotations and is quantified by employing a distribution over the output of the model.

3.3.1 Quantifying Epistemic Uncertainty in Classification

Uncertainty in classification can be quantified using the Shannon entropy of the final softmax scores in Equation 3.13. As used by *Feng et al.* [14]. For an input of x^* the classification probability of class for N with the availability of the probability score is calculated using Equation 3.12. This probability is used to calculate Shannon entropy (SE) using Equation 3.13

$$\begin{aligned} p\left(c \mid \mathbf{x}^*\right) &= \mathbb{E}_{p(\mathbf{W}|\mathbf{X},\mathbf{Y})} p\left(c \mid \mathbf{x}^*, \mathbf{W}\right) \\ &\approx \frac{1}{N} \sum_{i=1}^{N} s_{\mathbf{x}^*}^i \end{aligned} \tag{3.12}$$

$$\begin{aligned} &SE\left(\mathbf{y}^* \mid \mathbf{x}^*\right) = \mathbb{H}\left(\mathbf{y}^* \mid \mathbf{x}^*\right) \\ &= -p\left(c \mid \mathbf{x}^*\right) \log p\left(c \mid \mathbf{x}^*\right) - p\left(\neg c \mid \mathbf{x}^*\right) \log p\left(\neg c \mid \mathbf{x}^*\right) \\ &\approx -\tfrac{1}{N} \textstyle\sum_{i=1}^{N} s_{\mathbf{x}}^i \cdot \log \tfrac{1}{N} \sum_{i=1}^{N} s_{\mathbf{x}^*}^i - \left(1 - \tfrac{1}{N} \sum_{i=1}^{N} s_{\mathbf{x}^*}^i\right) \log \left(1 - \tfrac{1}{N} \sum_{i=1}^{N} s_{\mathbf{x}^*}^i\right) \end{aligned} \tag{3.13}$$

If the SE value is zero, then the network is most certain of the output and when SE value is higher the network is not certain of the classification result.

3.3.2 Quantifying Epistemic Uncertainty in Regression

In order to find the spatial uncertainty in the bounding box parameters *Feng et al.* [14] proposed to calculate Total Variance (TV), which represents how much each measurement varies from their mean value.

1. Transform the bounding box detections into world reference frame and the mean value of the detections regressed during N forward passes is calculated.
2. Covariance matrix of the detections is calculated using the following the equation $C\left(\mathbf{x}^*\right) = \frac{1}{N}\sum_{i=1}^{N} \hat{\mathbf{v}}_{\mathbf{x}^*}^{i} \hat{\mathbf{v}}_{\mathbf{x}^*}^{i^T} - \mathbf{I}_{\mathbf{x}^*}\mathbf{I}_{\mathbf{x}^*}^{T}$.
3. Total variance is calculating by taking the trace of the covariance matric $C(x^*)$.

The Total variance spans from $[0, +\infty)$, the higher the value represents higher the epistemic uncertainty in regressed bounding box parameters.

3.4 Related Work

The seminal work towards implementing Bayesian concepts in DNNs is proposed by *Buntine et al.* [45], this work approximated the posterior probability by proposing a *maximum-a-posteriori* method. An additional suggestion by them is to utilize second-order distributions for prior probabilities which is supposed to increase the smoothness of the approximated posterior. *Hinton et al.* [46] had proposed a method of regularizing the neural networks which in disguise are also variational methods and helped the neural networks to better generalize and avoid overfitting. One more statement in the same work of our interest is that the addition of Gaussian noise can control the information stored by the weights.

The original work solely on BNNs is done by *Graves et al.* [39], his method includes approximating a variational distribution, and also an efficient sampling is implemented to perform optimization of the neural network. One major observation in his work is that the use of a Gaussian distribution for the prior distribution for weights resulted in decent results in a recognition task. Extending on the above framework of the back-propagation algorithm for BNNs was done by *Blundell et al.* [15] in which the optimization method of maximizing the ELBO in Equation 3.4 is

achieved by fitting a Normal distribution parameterized by a mean and standard deviation over the weights. They named the method Bayes by backprop because the gradient calculated during backpropagation is being used for mean and standard deviation update of the approximated posterior distribution. This work achieved a fixed-weight neural network level performance on the Modified National Institute of Standards and Technology database (MNIST) digit recognition task using a Gaussian Mixture Model of two Gaussians as weight prior and a Gaussian posterior distribution for $P(D|w)$.

The works discussed are of the neural networks in general perceptron form but the major complex tasks are being solved by CNNs. CNNs consists of not only weights of the neural links but also the filters, this problem is first solved by *Shridhar et al.* [41] by placing the probability over the weights in the filters as well and instead of just sampling the weights they implemented a reparameterization [42] technique by sampling the activation because of the acceleration it causes in the subsequent layers. But this kind of reparameterization does not quantify the uncertainty because all examples will have similar weights hence a constrained variance. To address this issue *Wen et al.* [47] proposed a method called Flipout, which manages to have variance reduction by sampling the weights pseudo-independently for every sample. This method is claimed to be the best performing reparameterization technique that works for dense, convolutional, and recurrent layers.

3.5 Drawbacks

- The architectures defined in Section 2 for 3D Object detection are majorly fixed-weight, deterministic in nature.
- The implementations done for uncertainty quantification are majorly single stage and are relatively not complex compared to models in Section 2.

4

Methodology

This chapter contains, a detailed explanation about the dataset and method that are chosen to solve the problem of 3D object detection and our trials in building a Bayesian Neural Network (BNN) model of the architecture used for 3-Dimensional (3D) object detection. In the end, we also describe the experimental setup for training and validating the model.

4.1 Lyft Level-5 Perception Dataset

The research on usage of Deep Neural Network (DNN) in autonomous driving is on the rise and is majorly dependent on the Deep Neural Networks (DNNs) as shown in Figure 1.1. These DNNs need data in order to train. Hence, many technical startups [21, 48, 11], and Original Equipment Manufacturers (OEM) [49, 22] have provided the environment data collected from their own custom sensor suites while driving through various city centers.

We chose to work with the Lyft Level-5 Perception dataset [21]. Detailed explanation and analysis of the dataset can be found in Section 4.1.

Lyft Level-5 Perception Dataset [21] is provided by a mobility service provider Lyft and is collected from its own fleet of vehicles fitted with own sensor suite consists of 7 cameras (Front, Rear, Front-Left, Front-Right, Rear-Left, Rear-Right and Front Zoomed) and 3 Light Detection and Ranging (LiDAR) sensors (Top, Front-Left, Front-Right). The sensor suite can be seen in Figure 4.1.

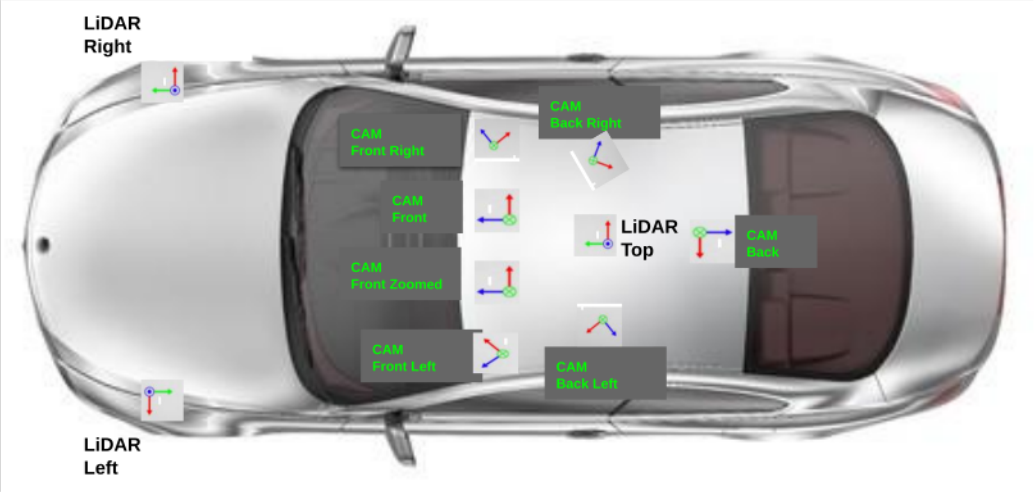


Figure 4.1: Sensor suite used by Lyft to collect data [21]

The dataset is divided into 398 scenes among which 180 scenes form the training dataset and 218 scenes from the test dataset. Each scene consists of 25-45 seconds of acquired data from the environment in which the car traveled and in total has 22680 sample instances resulting in a total of 409248 images and 50148 LiDAR point cloud data along with the above data Lyft has also provided calibration data of all the sensors with respect to the car reference frame is also provided. There are a total of 9 object classes present in the dataset. The sensor synchronization is achieved by triggering the camera when the top LiDAR is collecting the data in the camera Field-Of-View (FOV). This helps in achieving better multi-modal data alignment.

The training dataset in addition has annotations as well which consists of category, attributes, and a 3D bounding box label (X-center, Y-center, Z-center, Length, Width, Height, and Yaw-Angle). The object in a scene is annotated if at least a part of it is visible in a camera frame or at least a single LiDAR point is covering the object. The sample instance can be seen in Figure 4.2.

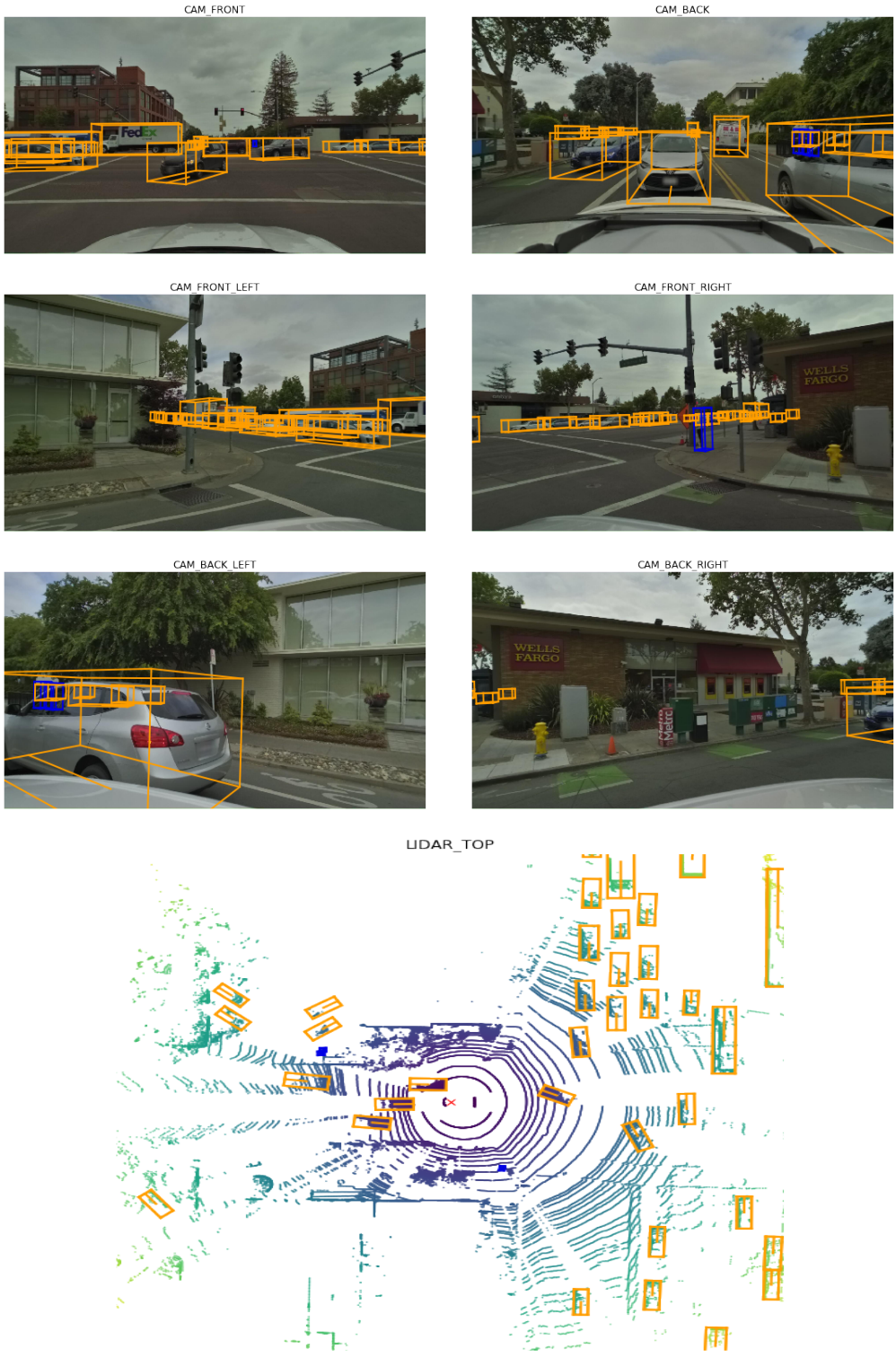


Figure 4.2: Images from a sample instance with annotations in Lyft dataset [21]

Since the ground truth of the test dataset is not available we planned to divide the training dataset into three parts with 60 percent of scenes for training purpose and 20 percent of train scenes for validation purpose and the rest 20 percent for testing purpose. The choice of scenes is done randomly. The dataset consists of 9 categories of agents including Car, Pedestrian, Cyclist, Truck, Emergency vehicle, Bus, Motorcycle, Animal, Other Vehicles. The frequency statistics can be seen in Figure 4.3.

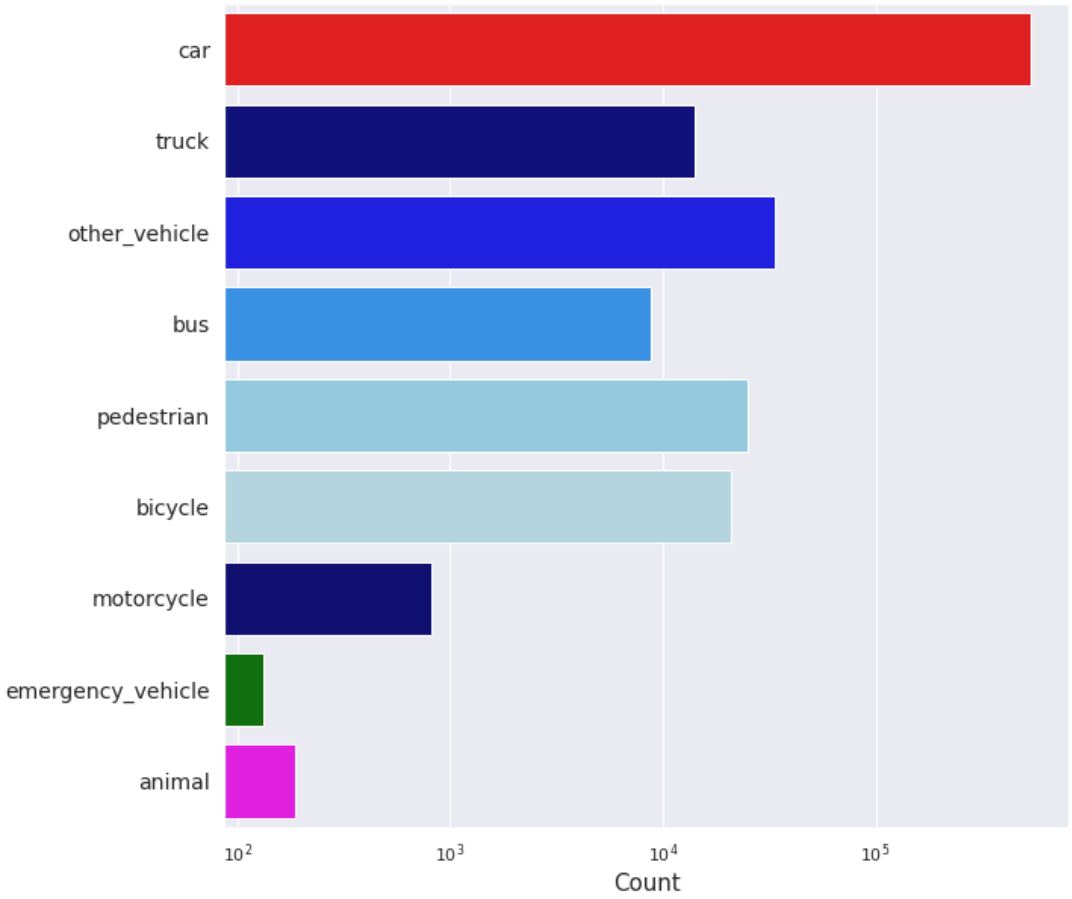


Figure 4.3: Frequency statistics of the classes in Lyft dataset [21] represented in logarithmic scale

4.2 Frustum-PointNet for 3D object detection

The architecture chosen for solving the 3D object detection is Frustum-PointNet proposed by *Qi et al.* [36]. The reason for choosing Frustum-PointNet [36] is its

performance in the case of pedestrian detection on the KITTI 3D object detection benchmark [11]. This method in theory can run in real-time as only parts of the point cloud are processed.

4.2.1 Model Architecture

Frustum-PointNet consumes point cloud data and 2-Dimensional (2D) bounding boxes of all the object instances present in the scene and regresses 3D bounding boxes of all the object instances present in the sample instance. The model pipeline can be seen in Figure 4.4.

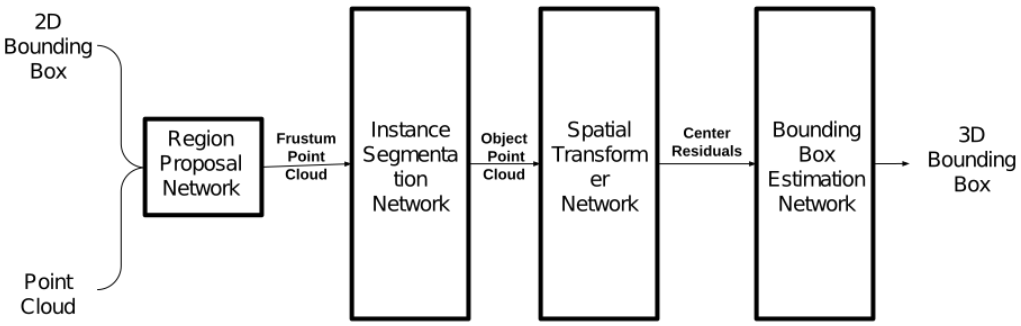


Figure 4.4: Frustum-PointNet Pipeline showing various stages involved to regress the 3D bounding box parameters [36]

The provided 2D bounding box is extended geometrically to form a frustum and the point cloud present in the frustum is extracted. Hence, the name frustum point cloud is given to it. When projected onto the 2D plane the point in the frustum point cloud lies inside the 2D Bounding box. So, a frustum point cloud has points relating to the object of interest.

The next stage is an Instance-Segmentation network based on PointNet [3]. This network classifies every point in the frustum point cloud as belonging to the object or not belonging to the object. The points classified as belonging to a class label form a segmented object point cloud.

The segmented object point cloud is fed into *T-Net* which regresses the center of the bounding box represented by (x, y, z). The output of this model is not

straightforward, rather a residual of the center of the bounding box to the centroid of the frustum point cloud. This indirectly projects the point cloud into an invariant space referenced by the centroid of the point cloud thereby making the model rotational invariant. The frustum point cloud is translated based on the residuals obtained and fed to the next stage. Bounding-Box Net, takes the translated point cloud and outputs the 3D bounding box parameters $(x, y, z, h, w, l, \theta)$.

4.2.1.1 2D Bounding Box Extraction

As already mentioned in Section 4.1 there are only 3D bounding box annotations provided but not 2D bounding boxes. Hence, we took two different paths to extract 2D bounding boxes for the training dataset and test dataset.

For the training dataset, we chose to perform a perspective projection of the 3D bounding box onto the image frame. This is done by first transforming the ego poses into the car's world reference frame and then projecting them from ego pose reference to the sensor under consideration. Then using the visibility mask that is generated based on whether all the corners of the bounding box presence in FOV of the camera is used to filter out the boxes not present in the camera frame.

For the test dataset, we used Fast-RCNN [1] and retrain it on the 2D boxes generated above and used it for extracting the 2D detections on the test images.

4.2.1.2 Frustum Point Cloud Extraction

To extract the frustum point cloud the LiDAR point cloud should be first projected onto the image by performing projective transformations to the point cloud this transformation gives the pixels equivalent of the point cloud.

Let's consider the projected point cloud $p_{UV} = \{(u_1, v_1), \ldots, (u_n, v_n)\}$ and the bounding box be be represented by $(u_{\min}, u_{\max}, v_{\min}, v_{\max})$.Then the point (u_i, v_i) lies in the 2D bounding box if it satisfies the following conditions.$u_{\min} \leq u_i \leq u_{\max}, v_{\min} \leq v_i \leq v_{\max}$ [36].

Then the frustum point clouds are sampled with or without replacement based on the number of points present in the frustum point cloud. This helps in making the frustum equal and maintaining a fixed-sized neural network. A representation of frustum extraction can be seen in Figure 4.5.

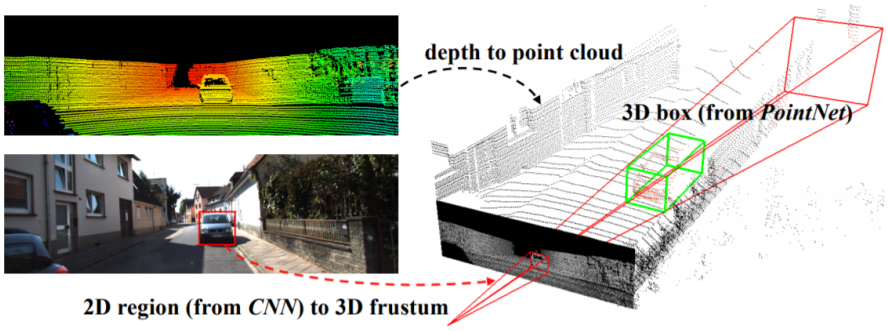


Figure 4.5: Frustum-Extraction showing the extrusion of the point cloud by extending the 2D bounding box into a pyramid frustum [36, pg.1]

After the sampling process is done, the point cloud is normalized by performing the rotational transformation in order to make the extracted point clouds center axis perpendicular to the image plane. The illustration can be found in Figure 4.6.

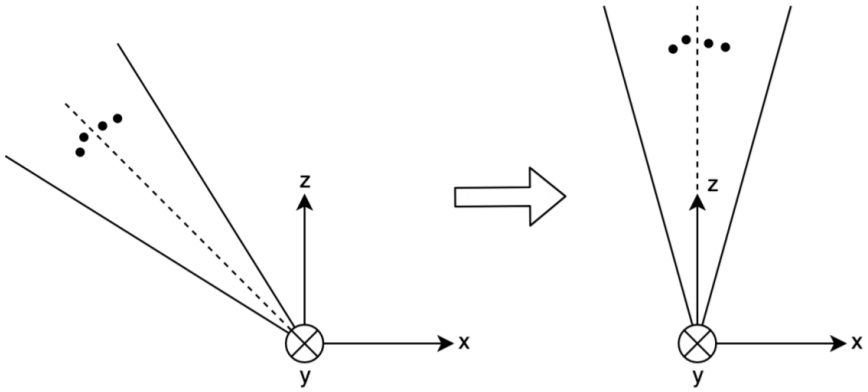


Figure 4.6: Frustum Normalization shows how the pointcloud are transformed inorder to align them to the z-axis [36, pg.6]

4.2.1.3 Instance Segmentation Network

The Instance segmentation network is a version of PointNet [3] which consumes the normalized frustum points and assigns binary classification scores which represent whether the point belongs to the object in the 2D bounding box or not. The illustration of this sub-network can be found in Figure 4.7.

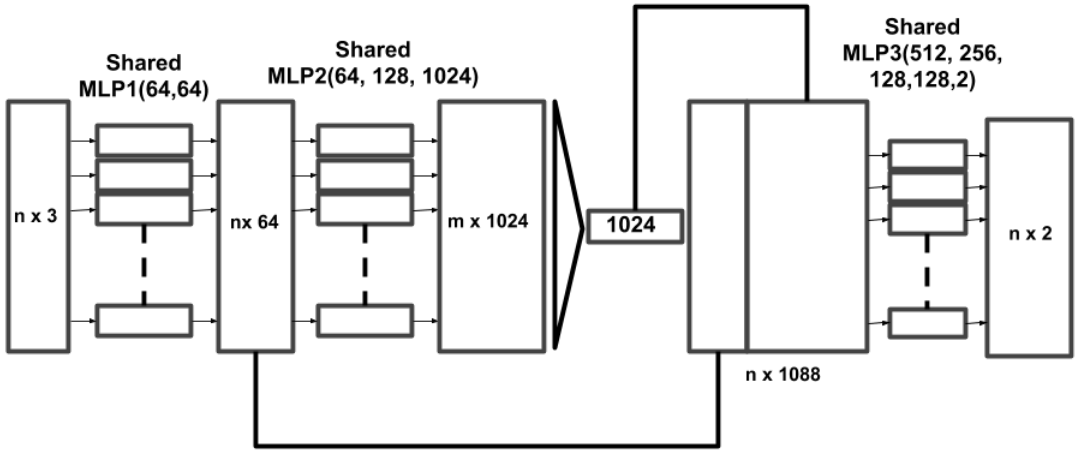


Figure 4.7: Instance segmentation network architecture which classifies each and every point whether it belongs to a object or not. It helps in detecting an object in a cluttered situations [36, SM pg 02]

The network majorly has three Multi-Layered Perceptrons (MLP). MLP1, which has 64 units in the first layer and the second layer has 64 units thereby resulting in a feature vector $\tilde{f}_i^1 \in \mathbb{R}^{64}$. This resulting vector is fed into MLP2 which results in another feature vector $\tilde{f}_i^2 \in \mathbb{R}^{1024}$ this feature vector is then fed into a max-pooling layer which extracts a global vector $f \in \mathbb{R}^{1024}$ which represents the features present in the point cloud. The obtained output from the max-pooling $f \in \mathbb{R}^{1024}$ and $\tilde{f}_i^1 \in \mathbb{R}^{64}$ are continuously concatenated in order to obtain $g_i \in \mathbb{R}^{1088}$ which represents all local features from the first MLP layer and the global features after max pooling operation. This concatenated vector is then passed to MLP3 which has five layers and results in two-class classification scores by using a softmax activation at the end.

4.2.1.4 Spatial Transformer Network (T-Net)

The point cloud obtained from the Instance segmentation is then translated and normalized by subtracting the centroid of the point cloud from each and every point. The normalized frustum point cloud is fed to a Spatial Transformer Network (T-Net) [50] which outputs the bounding box center residuals. Submodels architecture can be seen in Figure 4.8. It is an adapted PointNet [3] model in order to solve the classification model by consuming the raw point clouds. The point in the

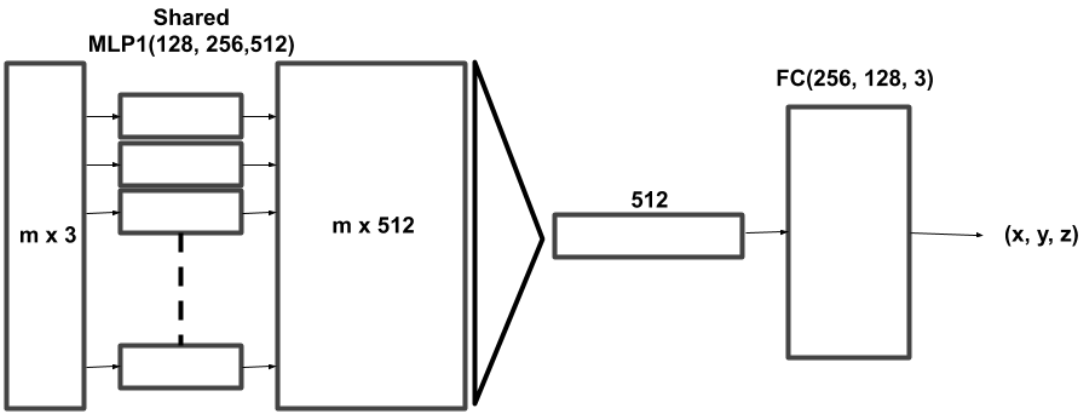


Figure 4.8: Transformer network network calculates the additional delta needed to calculate the exact bounding box center [36]

normalized point cloud is of form $p_i = (x_i, y_i, z_i, r_i)$ but this network uses only (x, y, z) components. As the point cloud data provided in Lyft dataset [21] doesn't have specified reflective value (r) thus removing it from processing will help in better model fitting. The resulting vector is passed through a max-pooling layer which results in a vector $f_i \in \mathbb{R}^{512}$. This vector is finally fed into a cascaded fully connected layer that outputs the residuals needed.

4.2.1.5 Bounding Box Net

Before feeding the point cloud into the bounding box-net the point cloud is translated to the 3D bounding box center frame by subtracting residuals regressed by the T-Net sub-network from all the points in the frustum point cloud. The network is illustrated in Figure 4.9.

The bounding box-net regresses $(x, y, z), (l, w, h)$ and 2*Number of heading bins values. Number of Heading bins is a design setting which will be discussed in later chapters. (x, y, z) represents the center of the bounding box and (l, w, h) represents the dimensions of the 3D bounding box, and the values at the end are used to calculate the heading angle, one of the two outputs is used for calculating the heading bin the angle belongs to and the second output to calculate the residual difference between the center angle and the heading angle.

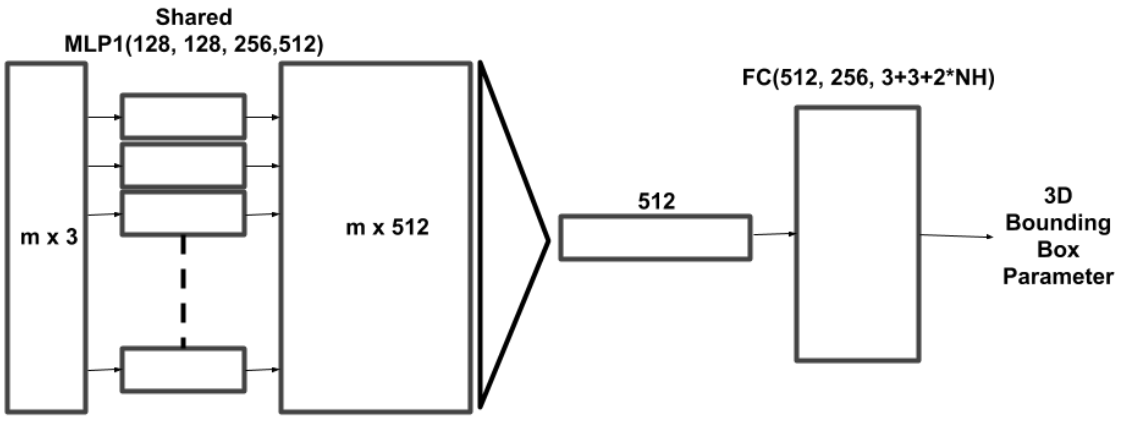


Figure 4.9: Bounding Box Network architecture [36, SM pg.04]

4.3 Design Decisions

In this section, we detail and defend the design choices like data augmentation and sample choices that are made to build the Frustum-PointNet [36] model and training it on the Lyft dataset [21].

4.3.1 Class choice and Data Augmentation

In order to simplify the frustum extraction and model definition, we chose to use the Car, Pedestrian, and Cyclist classes. As seen in Figure 4.3 in Section 4.1 the dataset is suffering from huge imbalance. The number of samples belonging to Car is far more than any other object class. Hence, we chose to perform augmentation using the following two methods.

- Random modification of the 2D bounding boxes by performing translation and scaling. In case of translation, the center (x,y) of the bounding box is moved to a new point $(x+\sigma_1, y+\sigma_2)$, σ_1 is sampled from an uniform distribution defined between $[-0.1w, 0.1w]$ and σ_2 is sampled from an uniform distribution defined between $[-0.1h, 0.1h]$, where (w, h) being width and height of the bounding box. In case of scaling, the width and height are modified by a scale randomly chosen from a uniform distribution with limits [0.9, 1.1].

- By following the third method proposed in this waypoint blog [51], the frustum point cloud after rotational normalization is rotated along the YZ axis for a pre-defined number of times by an angle randomly sampled from a Uniform distribution of $\left[-\frac{\pi}{10}, \frac{\pi}{10}\right]$.

The data augmentation is performed differently for different object classes. if an object belongs to the Car object class it is augmented 2 times and in the case of VRUs, the augmentation is performed 6 times for each sample.As PointNet has inherent rotational invariance, the augmentation also helps in controlling overfitting to a particular as well as boosting the learning performance.

4.3.2 Fixed-weight Frustum-PointNet

The implementation of the Frustum model is carried out in TensorFlow and Keras Functional API and is adapted from the implementation done by *Qi et al.* [36]. The model settings and the number of parameters can be seen in the Tables 4.2 and 4.1.

Parameter	**Count**
Total	1,651,528
Trainable	1,641,412
non-Trainable	10,116

Table 4.1: Parameters in Fixed-weight Frustum-PointNet

Model Setting	**Value**
Number of Points in Frustum point cloud	1024
Minimum number of object points	512
Number of Heading bins	8
Optimizer	Adam
Callback	validation loss based
Learning rate	0.001

Table 4.2: Model Settings used for training Fixed-weight Frustum-PointNet , The learning rate is halved for every 50 epochs

The number of heading bins is chose to be 8 resulting in bin center angles $\left[-\pi, -\frac{3\pi}{4}, -\frac{\pi}{2}, -\frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}\right]$ and the intervals in between the center angles defines their class.

4.3.2.1 Loss Function

The Frustum-PointNet model is trained in an End-to-End fashion. Hence, the three submodels are optimized together using a combined loss function as defined by *Qi et al*, [36].
The loss function is as follows

$$\begin{aligned}\mathcal{L} = \mathcal{L}_{\text{InstanceSeg}} + \lambda\,(\mathcal{L}_T + \mathcal{L}_{Bbox-c} \\ +\mathcal{L}_{Bbox-s} + \mathcal{L}^{reg}_{Bbox-\theta} + \mathcal{L}^{cls}_{Bbox-\theta} + \gamma\mathcal{L}_{\text{corner}}\big)\end{aligned} \tag{4.1}$$

1. $\mathcal{L}_{\text{InstanceSeg}}$ is the loss from the instance segmentation model. It is the negative log-likelihood loss applied to the output of the instance segmentation network and the label derived during the frustum extraction process.

2. $\mathcal{L}_T$ is the loss related to the Transformer network. This loss calculates the distance between regressed center residuals and the center residuals label from the 3D annotation. Huber loss is defined in Equation 4.3.2.1 is used for calculating this loss

3. $\mathcal{L}_{Bbox-c}$ is for the loss induced by the bounding box net. The loss is a Huber loss calculated between the true center of the bounding box label and the regressed center.

4. $\mathcal{L}_{Bbox-s}$ is for the regression loss in regressing the size of the bounding box. It's also calculated using Huber loss between regressed dimensions and the true dimensions of the object in the frame.

5. $\mathcal{L}^{reg}_{Bbox-\theta}$ is for including the loss generated due to improper regression of heading angle residuals and Huber loss is used to calculate this loss.

6. $\mathcal{L}^{cls}_{Bbox-\theta}$ is for calculating the loss due to improper classification of the heading angle bin. Negative loss likelihood is used in order to quantify this loss.

7. $\mathcal{L}_{\text{corner}}$ is the distance between the regressed 8 corners of the bounding box and the 8 corners from the annotations. This loss tends to penalize the loss resulting due to incorrect corner, heading angle, and yaw angle regression. Therefore, this loss has regularizing effects on the model.

Huber Loss

$$\mathcal{L}_{Huber}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & |y - \hat{y}| < 1 \\ |y - \hat{y}| - \frac{1}{2}, & |y - \hat{y}| \geq 1 \end{cases} \tag{4.2}$$

4.3.3 Uncertainty Quantification in Frustum-PointNet

In order to extract epistemic uncertainty, The model architecture is kept the same but the layers in the Fixed-weight model are replaced with Flipout layers from Tensorfloaw probability library as referred to in [17] from the TensorFlow Probability package released by the TensorFlow group. The Bayesian layers inherently have the ability to assign a distribution to the weights and the filters with a distribution hyper-parameter provided as a prior function of weights $\mathcal{N}(0, 1)$ and also implemented the Flipout estimator discussed in Section 3.1.4.1 for sampling during training and reparameterization purposes. This estimates the distribution during the forward pass.

Parameter	**Count**
Total	3,262,216
Trainable	3,252,100
non-Trainable	10,116

Table 4.3: Parameters in Bayesian Frustum-PointNet

The model also monitors Kullback-Liebler (KL) divergence [40] of each and every layer posterior distributions nearness to the prior distribution. The loss function is maintained the same as defined for the fixed-weight neural network mode other than including the KL divergence loss which should be minimized to increase the Evidence Lower Bound (ELBO).

4.3.4 Evaluation

In this section, we will discuss about evaluation metrics for 3D Object Detection and Uncertainty quantification.

4.3.4.1 3D Object Detection Evaluation

For evaluation purposes, we chose to use 3D Average Precision (3DAP) similar to KITTI benchmark [11]. 3D Average Precision is the area under the precision-recall curve. For a fixed 3D Intersection-over-Union (3D-IoU), *Precision* is calculated using the Equation 4.3 and *Recall* is calculated using the Equation 4.5.

$$precision = \frac{TP}{TP + FP} \tag{4.3}$$

$$recall = \frac{TP}{TP + FN} \tag{4.4}$$

Intersection over Union (IoU) is calculated using and a visual illustartion can be seen in figure 4.10

$$\text{IoU} = \frac{\text{area of intersection}}{\text{area of union}}. \tag{4.5}$$

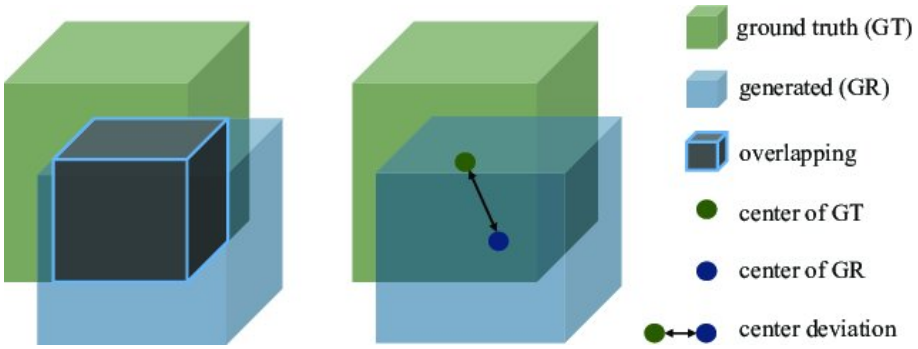


Figure 4.10: Intersection Over Union illustration [52, pg.09]

TP in Equations 4.3 and 4.5 is called True Positive. For judging, if detection is truly positive or not the IoU calculated should be greater than a limit set. The limits we chose can be found in Table 4.4 and FP is False Positive when IoU is less than the set IoU limit. When an object is present in ground truth but not detected by our model it's classified as False Negative (FN).

Then 3D Average Precision is calculated for 11 different recall values chose from 0 to 1 at equal intervals and maximum precision at each recall is calculated. The Average precision is calculated using the Equation 4.6.

Difficulty	Car	Cyclist	Pedestrian
Easy	0.5	0.3	0.3
Moderate	0.6	0.4	0.4
Hard	0.7	0.5	0.5

Table 4.4: IoU limits for evaluation

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} \max_{F: r \geq r} p(\tilde{r}) \tag{4.6}$$

4.3.4.2 Epistemic uncertainty Evaluation

Since, the task of agent classification is done by the 2D object detection network and we used a pre-trained Fast-RCNN [1]. We are not considering epistemic uncertainty in agent classification.

But, Frustum-PointNet [36] depends on using bin based regression of 3D object detection dimensions and the heading angle. So we decided to use Shanon entropy as mentioned in Section 3.3.1 [14] to quantify the epistemic uncertainty in size and heading bin classification. Also, for the final regression of object dimensions with added residuals we used total variance as mentioned in Section 3.3.2 [14] to calculate the epistemic uncertainty in the final detections.

5

Experiments and Results

This chapter consists of detailed explanation about various experiments done. In-order to implement the pipeline for frustum extraction in Section 5.1 and train the Frustum-PointNet model using them in Section 5.2. We converted the Frustum-PointNet model into a Bayesian model using TensorFlow probability library in Section 5.3. But, we observed that the model is under-fitting. We converted the Frustum-PointNet model into partial-Bayesian model, by fixing the weights of instance segmentation model and converting the spatial-transformer network and bounding box network into Bayesian model which fitted during training. The partial-Bayesian model is used for epistemic uncertainty estimation. The results and experimental data of every experiment are explained in detail at the end of each section.

5.1 Frustum Point Cloud Extraction and Normalisation

The aim of the experiment is to implement the pipeline for frustum point cloud extraction as mentioned in Section 4.2.1.2 and verify the pipeline by visualizing the final re-sampled and normalized frustum point cloud. The frustum point cloud should align to z-axis when the frustum rotation and normalization is performed with every frustum consists of 1024 points.

5.1.1 Experimental Procedure

The 3-Dimensional (3D) bounding box annotations in Lyft dataset [21] are provided in reference to global frame. The annotations and the Light Detection

and Ranging (LiDAR) point cloud are transformed to camera frame and the 2-Dimensional (2D) bounding boxes are extracted from the 3D annotations. Frustums are extracted as mentioned in Section 4.2.1.2 and the pipeline is visualized in the following figures.

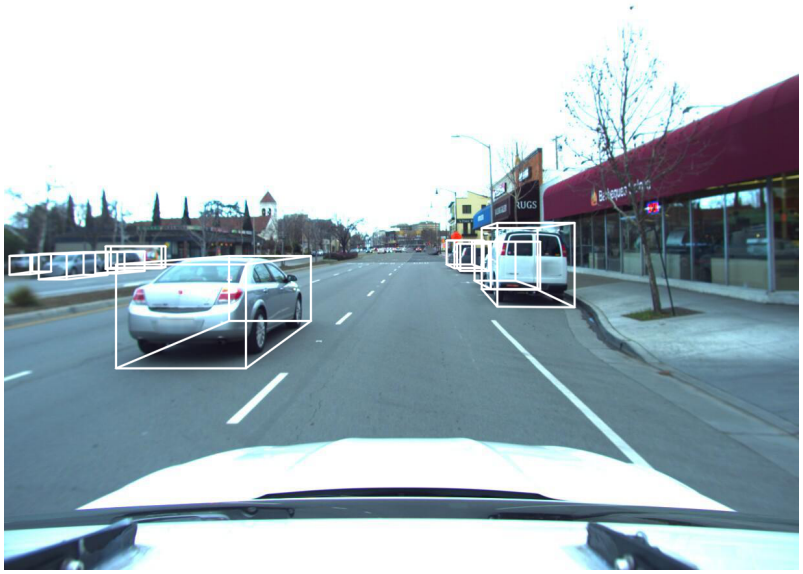


Figure 5.1: Image in a scene with 3D bounding box annotations.

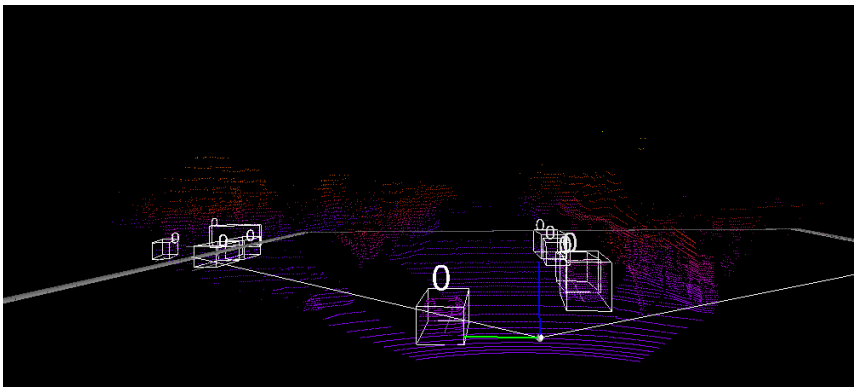


Figure 5.3: Point cloud in a scene with 3D bounding box annotations.

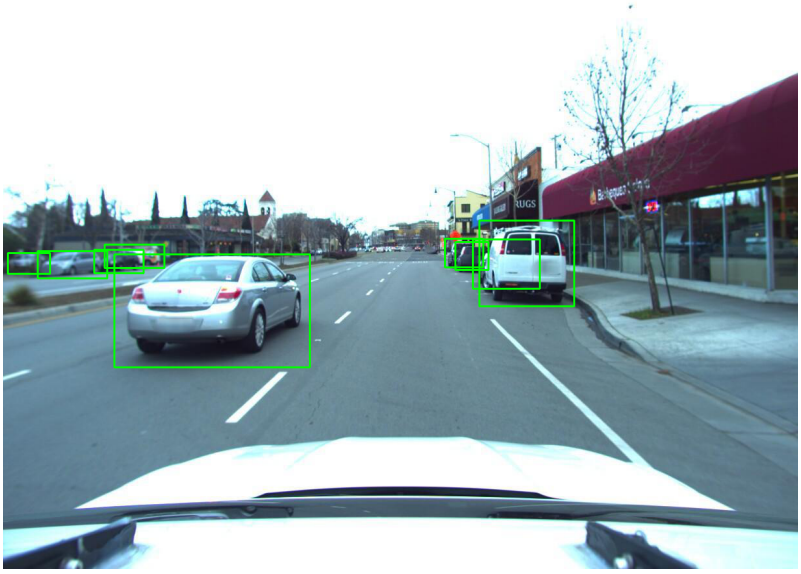
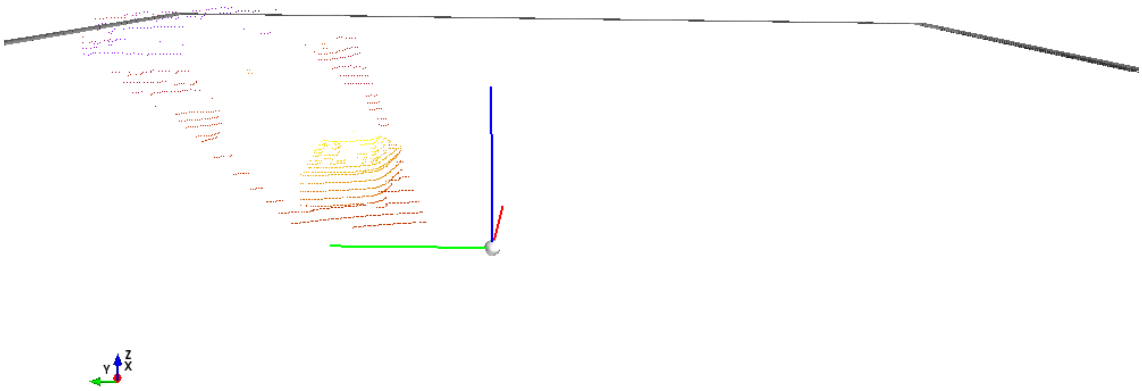
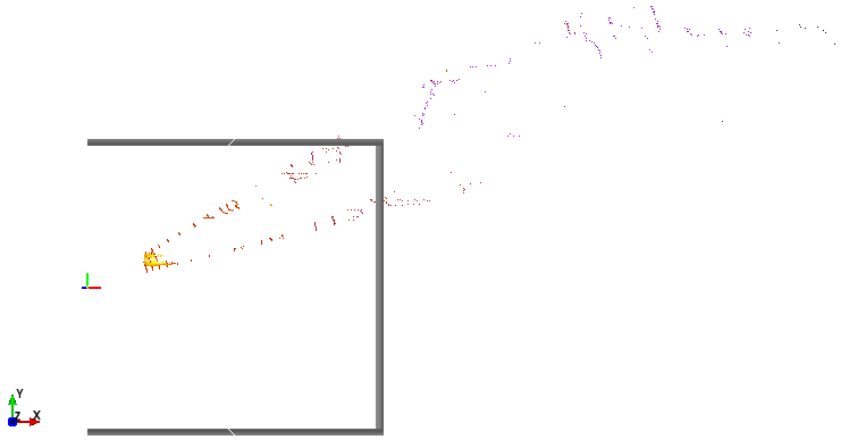


Figure 5.2: 2D bounding box annotations obtained from the projection of 3D bounding box into camera frame.



(a) Frustum point cloud in an ego-vehicle view.



(b) Frustum point cloud in a bird-eye view.

Figure 5.4: Frustum point cloud in different views, from these pictures we can observe that the frustum point cloud is a pyramid representing the LiDAR point cloud spanned by the 2D bounding box of the agent.

5.1.2 Observations

The extracted frustum point cloud after normalization and rotation is visualized in Figure 5.5. From the Figure 5.4, we can observe that the frustum point cloud is aligned at an approximate of 45 degrees to all the axis. In Figure 5.5 from the included axis representation, we can observe that the frustum point cloud is aligned to Z-axis.

The input values along with the frustum point cloud are illustrated in Table 5.1

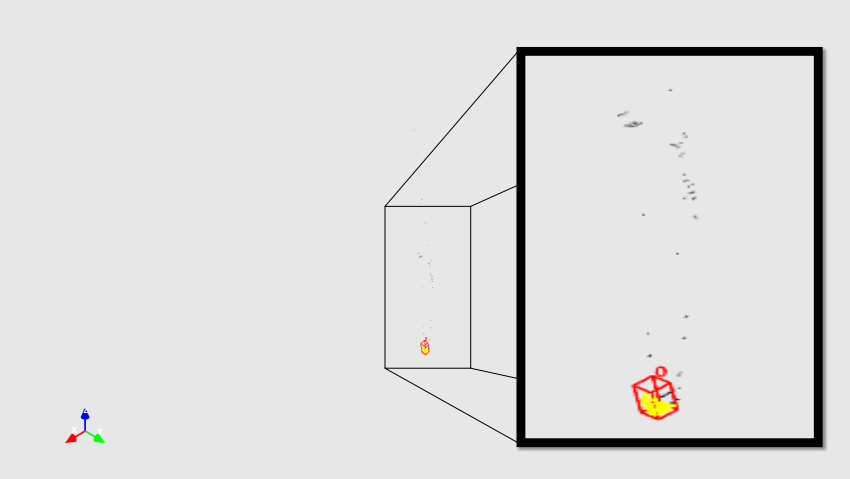


Figure 5.5: Extracted frustum point cloud after normalization, The point cloud is aligned to the Z-axis as opposed to the box alignment in 5.4.

Table 5.1: Feeder values to the model and their shapes, the number of points in the frustum point cloud is a hyper-parameter set to 1024 and the segmented point cloud count is set to 512 .

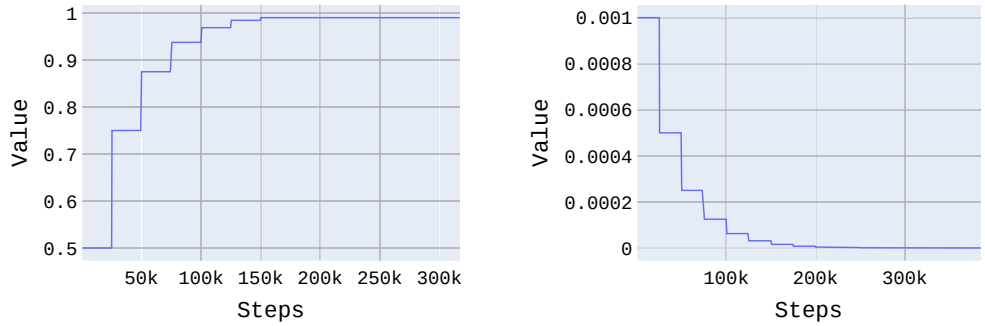
Input	Shape
Frustum point cloud	(1024,4)
One hot vector	(3)
Segmentation label	(512,1)
Center label	(3)
Heading class label	(1)
Heading residuals label	(1)
Size class label	(1)
Size residuals label	(3)

5.2 Training Frustum-PointNet and evaluating it on the test dataset

The main aim of this experiment is to train a Frustum-PointNet using Lyft dataset as per the design decision that are defined in Section 4.3 and report the results based on the evaluation explained in Section 4.3.4.

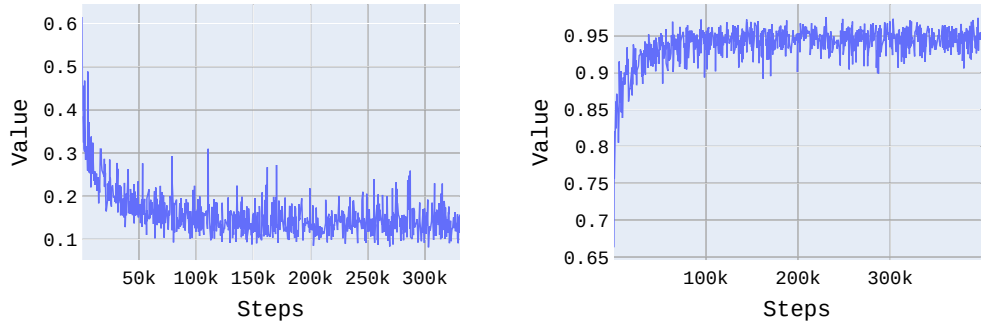
5.2.1 Experimental Procedure

The model is trained using a total of 230,944 samples divided into 7,217 training samples of batch size 32. The training is done for 178 (i,e. Three days) epochs and various values are logged which can be used to assess the accuracy of fit of the models to the frustum data.

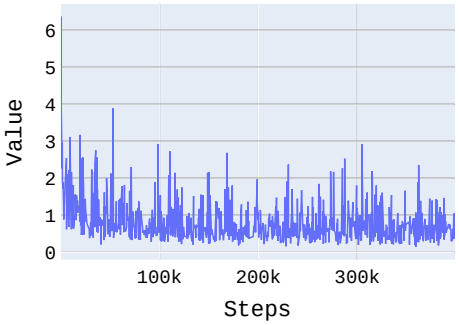


(a) Batch Normalization decay rate (b) Learning rate visualization

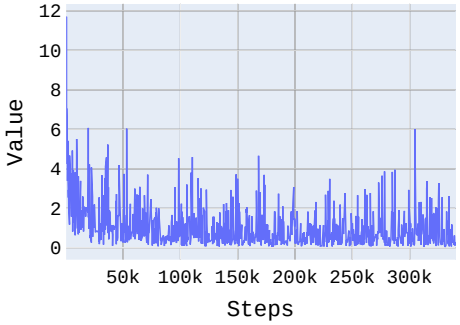
Figure 5.6: Model Setting Visualization, image:a shows the increase in decay rate of the batch normalization layers and image:b shows decrease in learning rate as number of epochs increase



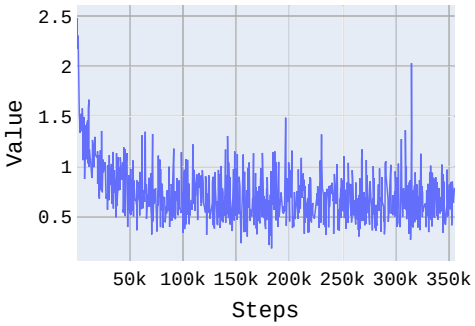
(a) Instance segmentation model loss (b) Instance segmentation model accuracy



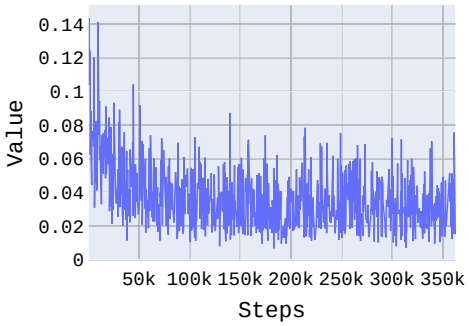
(c) Loss in center prediction after T-Net



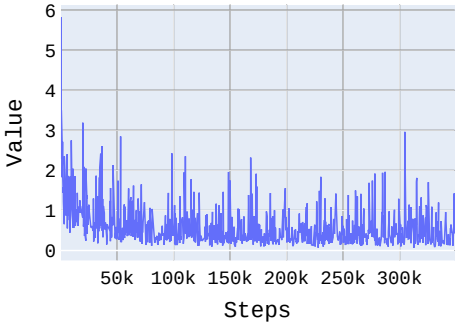
(d) Final Loss in Center Prediction



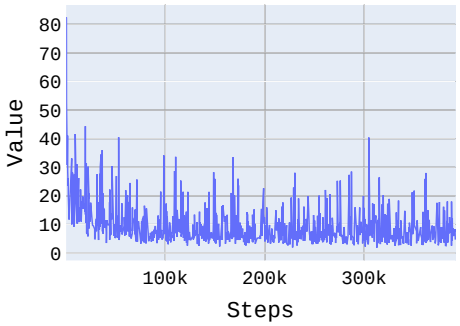
(e) Heading class loss



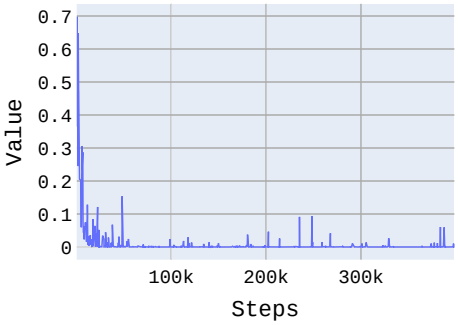
(f) Heading residuals normalized loss



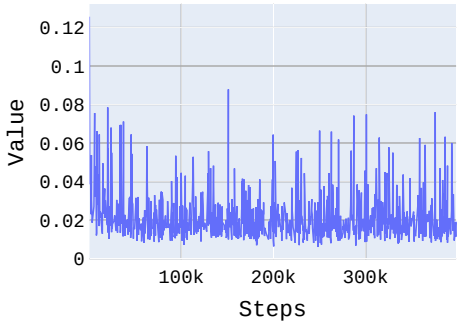
(g) Corners Loss



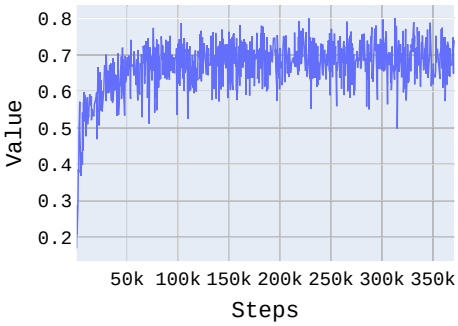
(h) Total Loss



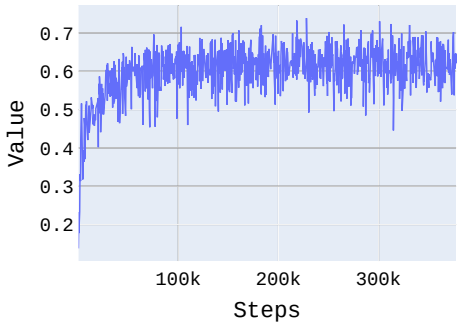
(i) Size Class Loss



(j) Size Residual Normalized Loss



(k) 2-D IoU



(l) 3-D IoU

Figure 5.7: Model parameters visualization, from images a-j we observed that the loss values at every stage of Frustum-PointNet are reducing which means the model is fitting well, images k and l shows increase in IoU values which further corroborates our deduction

5.2.2 Observations

The model is evaluated using the method reported in Section 4.3.4. We used the frustums created from the 2D object proposals extracted from the perspective projection of 3D bounding boxes and from a mature detector as mentioned in Section 4.2.1.1.

5.2.2.1 Quantitative results

The precision-recall curves with 2D proposals generated by perspective projection of 3D bounding Box can be seen in Figure 5.8.

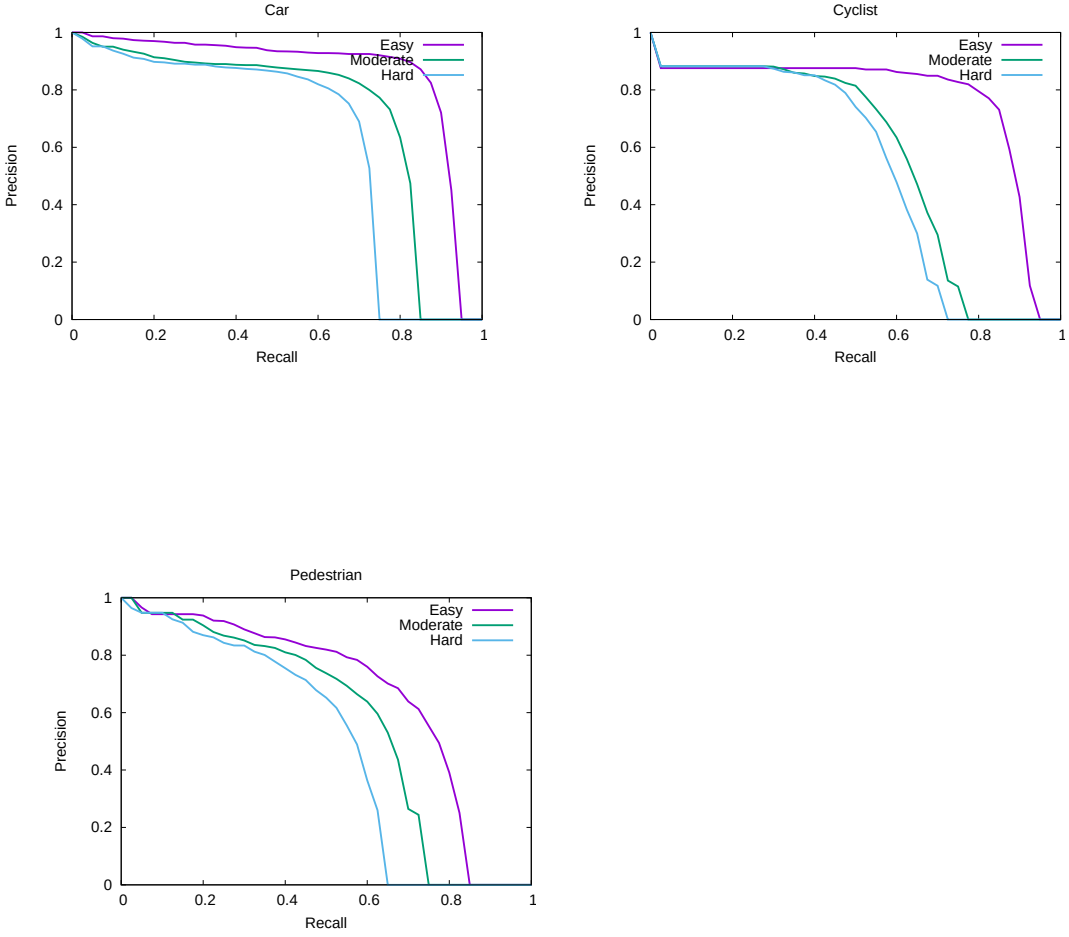


Figure 5.8: Precision-Recall graphs from Lyft 3D detection evaluation

The 3-Dimensional Average Precision (3DAP) values are calculated as stated in Section 4.3.4 and reported in the Tables 5.2 and 5.4.

Table 5.2: 3D Average Precision values for 3D object detection on Lyft dataset [21]

Difficulty	**Car (%)**	**Cyclist (%)**	**Pedestrian (%)**
Easy	80.19	69.79	60.59
Moderate	64.74	42.35	34.93
Hard	58.28	37.28	34.93

Table 5.3: 3D Average Precision values for 3D object detection on KITTI dataset [11]

Difficulty	**Car (%)**	**Cyclist (%)**	**Pedestrian (%)**
Easy	87.28	73.42	55.26
Moderate	77.09	59.97	47.56
Hard	67.90	52.88	42.57

The precision-recall curves with 2D proposals from the images in Lyft test dataset [21] generated by Fast-RCNN [1] using the Detectron object detection API trained on Lyft train dataset [21] and are reported in Figure 5.9.

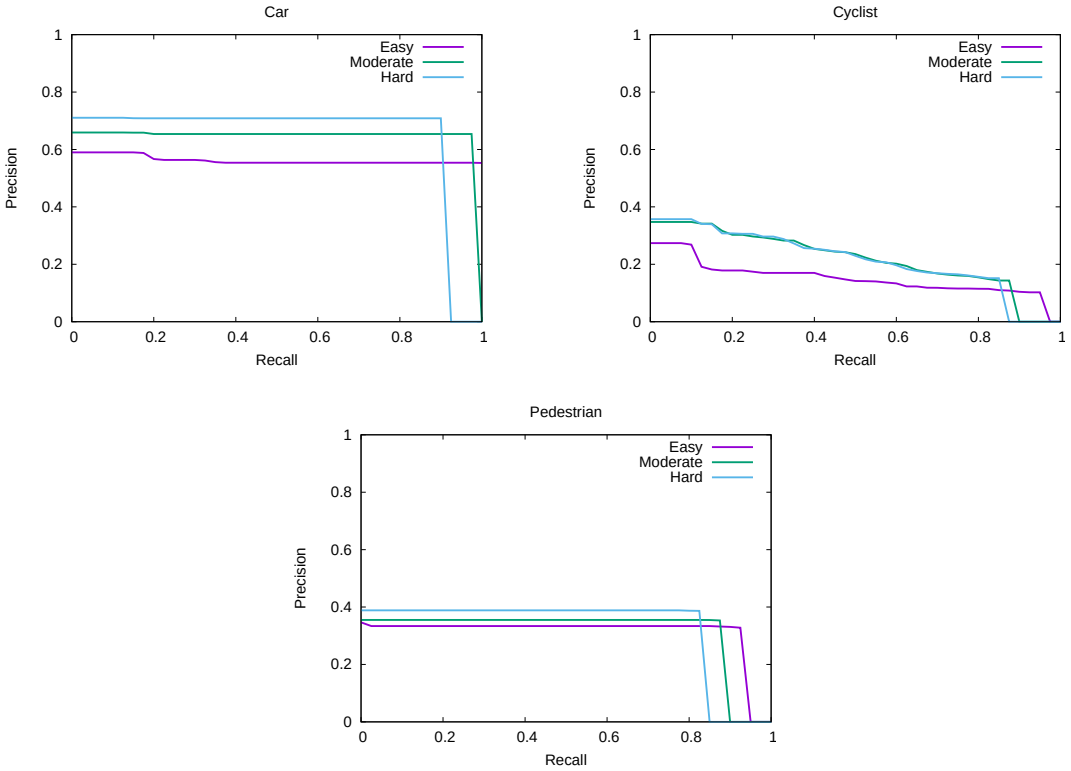


Figure 5.9: Precision-Recall graphs from Lyft 3D detection evaluation using a 2D object detector

Table 5.4: 3D Average Precision values for 3D object detection on Lyft dataset [21]

Difficulty	**Car (%)**	**Cyclist (%)**	**Pedestrian (%)**
Easy	64.82	12.38	14.68
Moderate	58.68	10.92	12.32
Hard	46.68	6.92	9.68

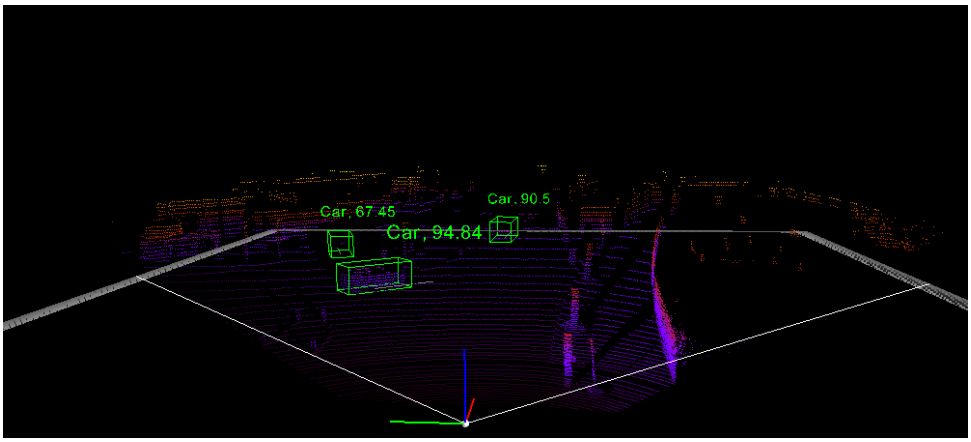
The detection performance is deteriorated when using Fast-RCNN [1] because the image detector cannot detect the objects when there is very low visibility of object in the image but the annotations provided still has the object 3D bounding box as there is at-least one point hitting the less visible object.

5.2.2.2 Qualitative results

The visualizations of the result of a sample from scene-19 and scene-114 of Lyft dataset [21] can be seen in Figure 5.10 and Figure 5.11. In the former image the 3D prediction boxes are visualized in the image plane and in the bottom image the predicted 3D boxes visualized in the LiDAR plane. Since, the sample from scene-114 is a cluttered environment for better understanding of the agent placement we added a BEV projection of the detections as well.

Table 5.5: Colors chosen based on detection probability limits.

Probability (Prob)	**Color**
Prob < 0.5	Red
$0.5 <$ Prob < 0.75	Yellow
Prob > 0.75	Green

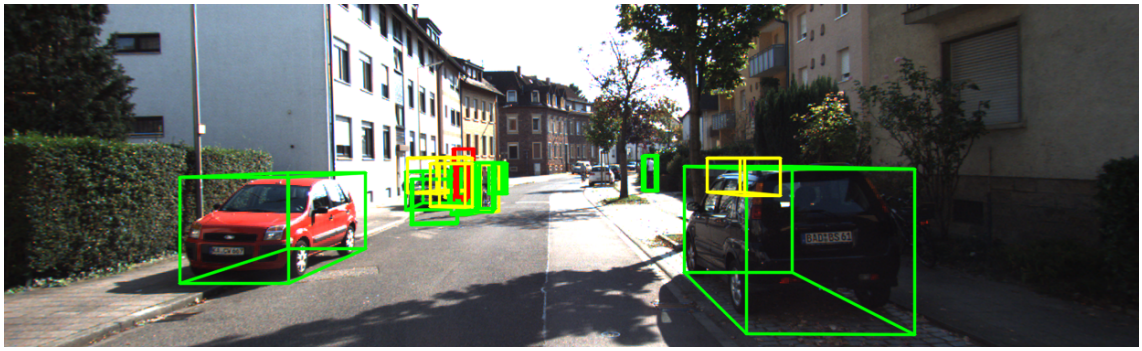


(b) Predictions in LiDAR plane

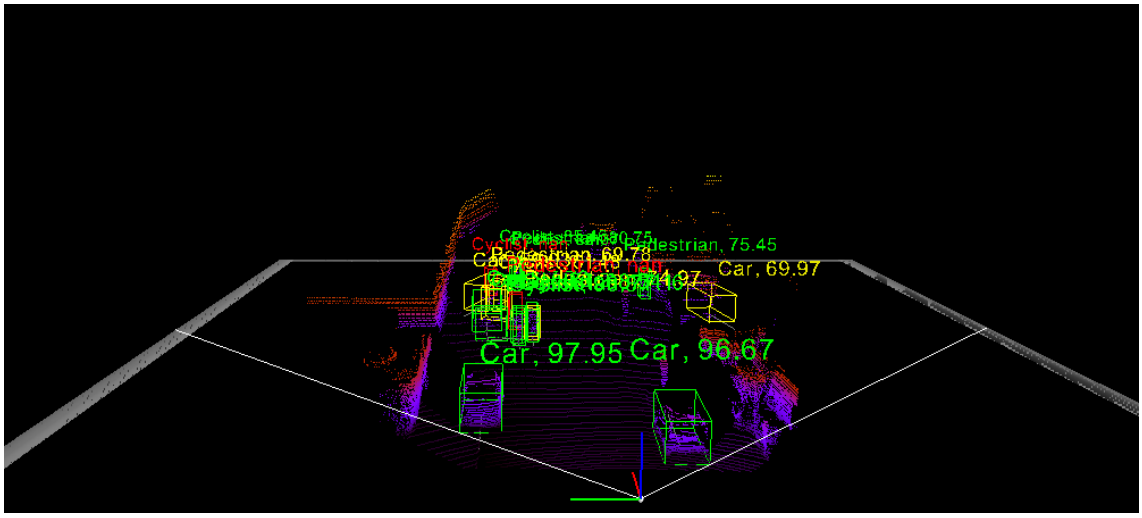
Figure 5.10: Visualization of results from inference of a sample from scene-19 using trained Frustum-PointNet

(a) Predictions in image plane

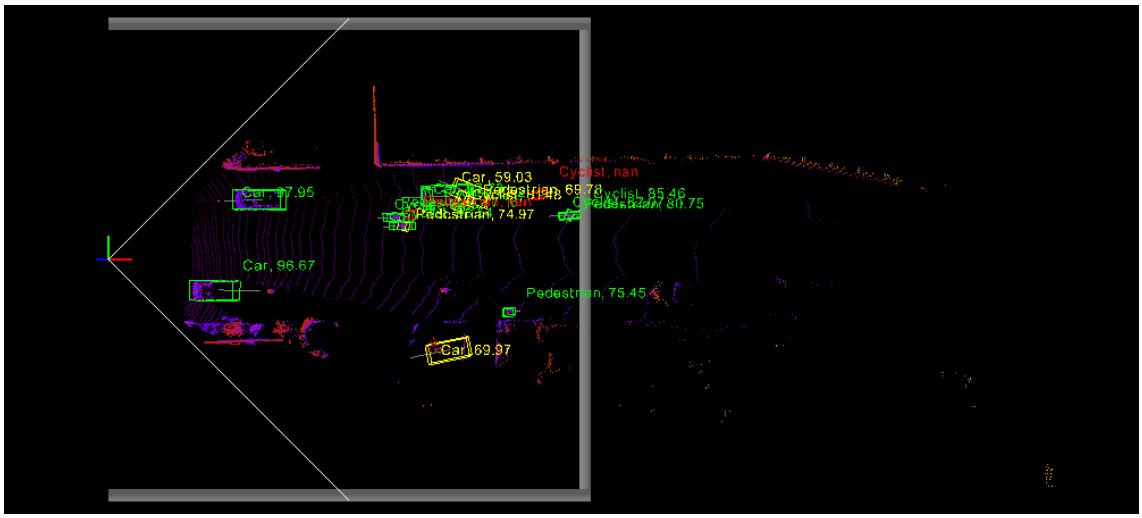
The trained Frustum-PointNet model performed really well on the test dataset. But, we observed that model detected occluded objects with higher probabilities. For example, in Figure 5.11 some samples which are mostly occluded are being detected with confidence greater than 70 percent. This issue can be seen in detail in Figure 5.12, the agents which are greater than 30 meters far and are occluded by many other agents has higher confidence scores. This overly confident results might act as bad inputs to the later stages of the autonomous driving stack.



(a) Predictions in image plane



(b) Predictions in LiDAR plane



(c) Predictions in BEV projection of LiDAR point Cloud

Figure 5.11: Visualization of results from inference of a sample from scene-114 using trained Frustum-PointNet

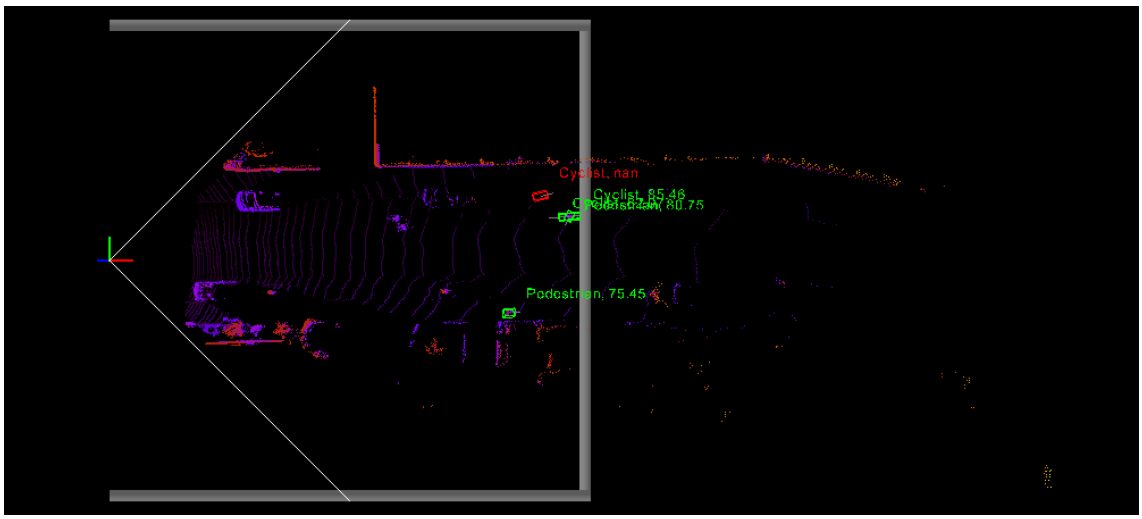


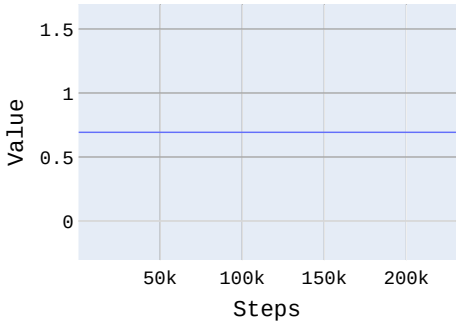
Figure 5.12: The BEV projection visualization shows that the mostly occluded agents in the sample are predicted with high confidence

5.3 Training and Evaluating the Bayesian Frustum-PointNet model

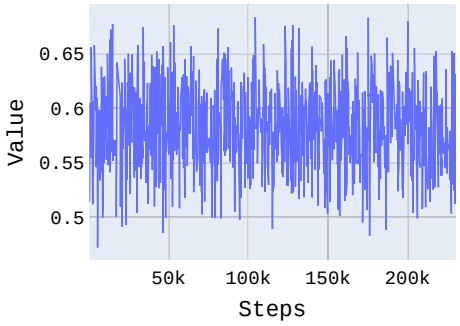
The aim of this experiment is to convert the Frustum-PointNet model [36] into a Bayesian Neural network model which will be referred to as Bayes-FPointNet. Bayes-FPointNet is trained using the frustums extracted in Experiment 5.1. The evaluation and epistemic uncertainty extraction are done based on the criterion described in Section 4.3.4. As the Bayes-FPointNet model is the same as Frustum-PointNet model, with the difference being the way we model the filters and weights. The model should show the same characteristics as the Frustum-PointNet during training and inference.

5.3.1 Experimental Procedure

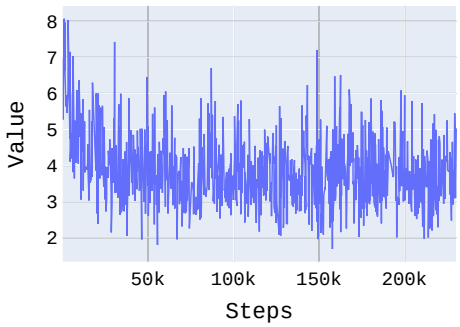
The experimental method is same as mentioned in Section 5.1. From the model logs shown in Figure 5.13.



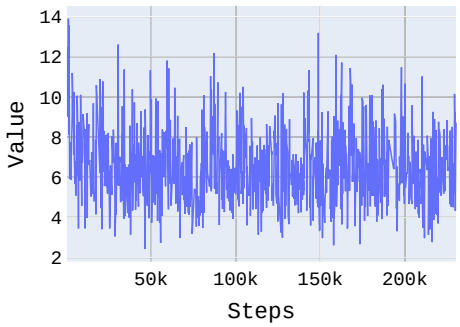
(a) Segmentation model loss



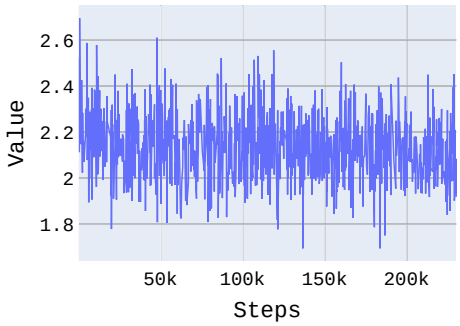
(b) Segmentation model accuracy



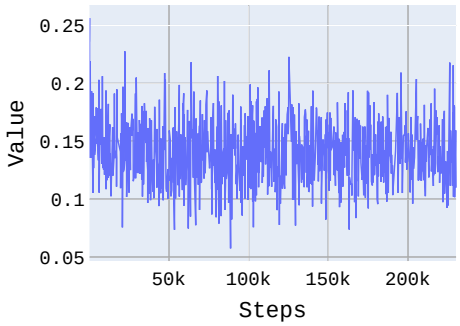
(c) Loss in center prediction after T-Net



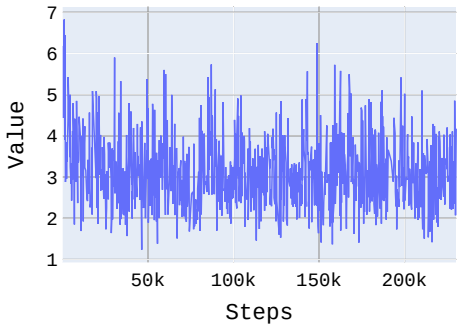
(d) Final Loss in Center Prediction



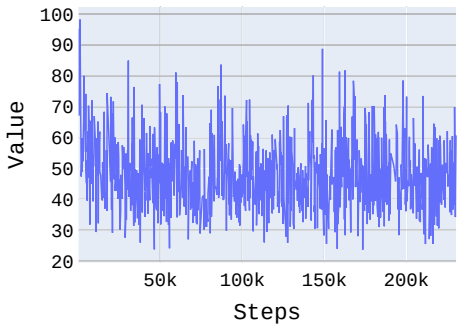
(e) Heading class loss



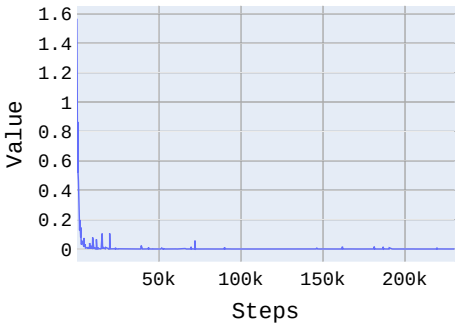
(f) Heading residuals normalized loss



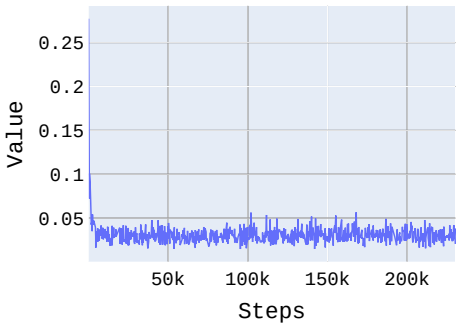
(g) Corners Loss



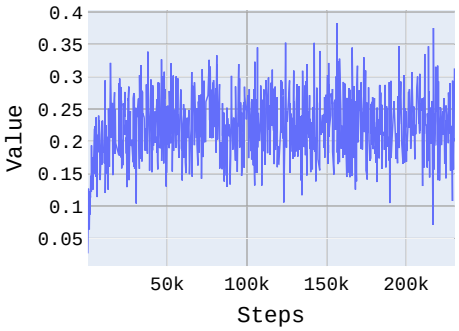
(h) Total Loss



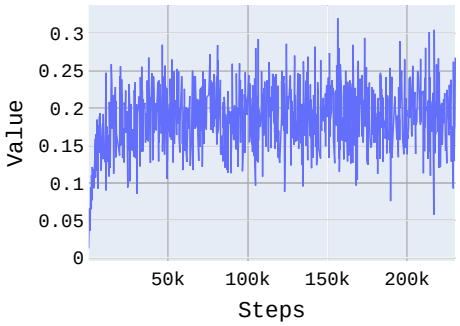
(i) Size Class Loss



(j) Size Residual Normalized Loss



(k) 2-Dimensional Intersection over Union



(l) 3-Dimensional Intersection over Union

Figure 5.13: Model parameters visualization, from images a-j we observed that the loss values at every stage of Frustum-PointNet are not reducing which means the model is not fitting well, images k and l also shows that IoU values are very dynamic and are not converging, which further corroborates our deduction that the model did not fit well.

5.3.2 Observations

The experimental data shows that there is no reduction in the loss values of any of the three stages in the Bayes-FPointNet model. The reason for the model to not converge is still to be fully investigated. But, we believe that it might be due to the models inability to perform a proper inference from the approximated posterior. This occurs when the number of training parameters are high.

5.4 Training and Evaluating a Partial-Bayesian Frustum-PointNet

The aim of this experiment is to convert the spatial-transformer network and bounding box estimation stage of the Frustum-PointNet model into a BNN while using the weights obtained during the Section 5.2 for the instance segmentation network. This results in reduction in number of parameters as shown in Table 5.6, we believe that this helps in finding a better posterior estimation and leads to model convergence. The training and evaluation done are similar to process done in Section 5.2 and Section 5.3. The expectation from this experiment is that the network does not under-fit to the data as seen in the Experiment 5.3. The quantitative performance evaluation should show the similar performance as seen in Figure 5.8.

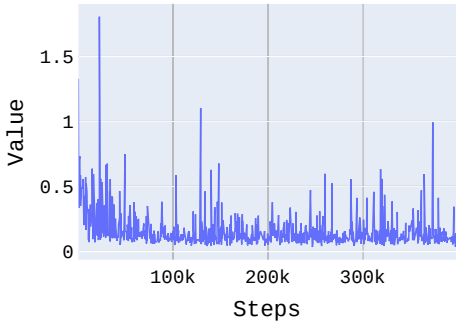
5.4.1 Experimental Procedure

The weights of the first stage of the Frustum-PointNet model are recovered from the model trained in the Section 5.2 and are freezed. So the weights of the instance segmentation network cannot be trained from the data fed. The spatial-transformer network and bounding-box network is re-modelled using flipout layers from the TensorFlow probability library and the losses are adapted from Experiment-3.

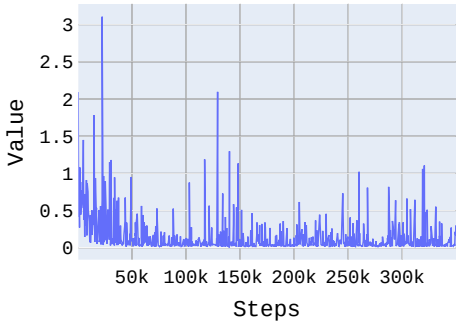
The number of parameters both trainable and non-trainable can be seen in Table 5.6. The model logs visualizations from Figure 5.14 shows that the model had converged and can be used for epistemic uncertainty quantification.

Table 5.6: Parameters in Part-Bayesian Frustum-PointNet

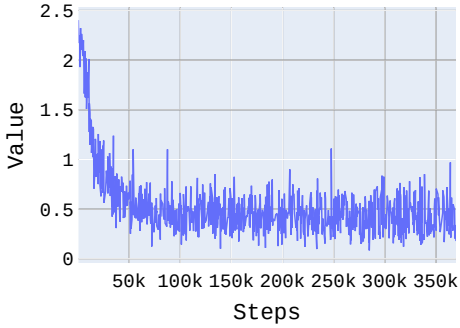
Parameter	**Count**
Total	2,375,496
Trainable	2,365,380
non-Trainable	10,116



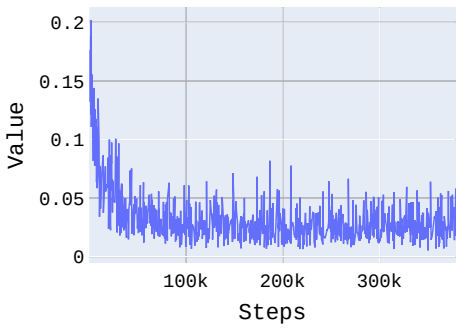
(a) Loss in center prediction after T-Net



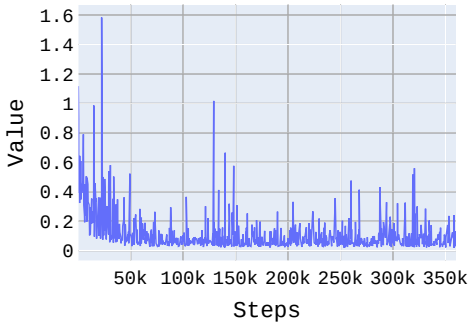
(b) Final Loss in Center Prediction



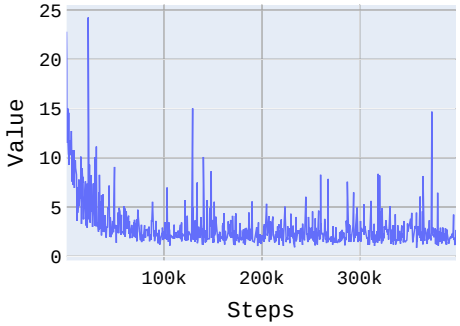
(c) Heading class loss



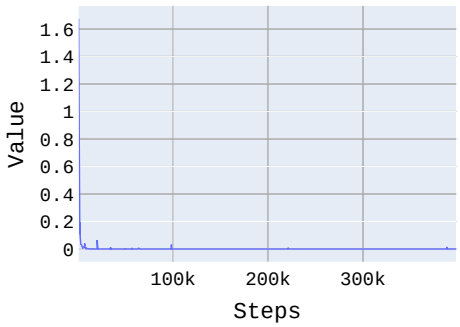
(d) Heading residuals normalized loss



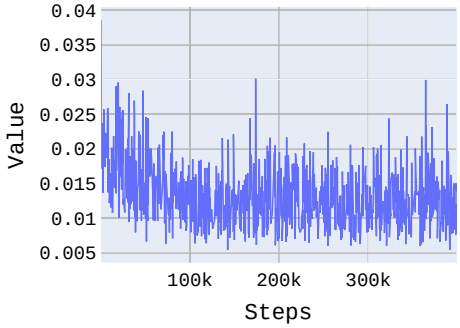
(e) Corners Loss



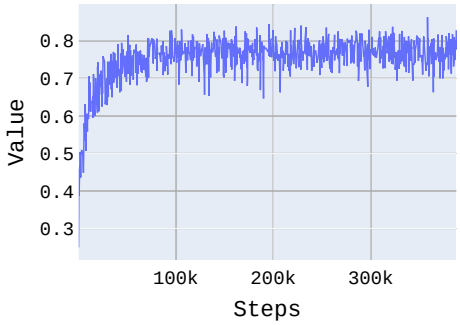
(f) Total Loss



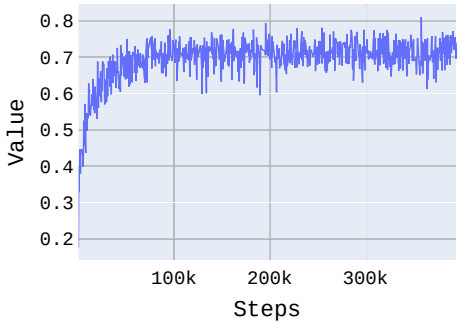
(g) Size Class Loss



(h) Size Residual Normalized Loss



(i) 2-Dimensional Intersection over Union



(j) 3-Dimensional Intersection over Union

Figure 5.14: Model parameters visualization, from images:a-h we observed that the loss values at all stages of Partial Bayesian Frustum-PointNet are reducing which means the model is fitting well, images i and j shows increase in IoU values and reaching a convergence, which further corroborates our deduction.

5.4.2 Observations

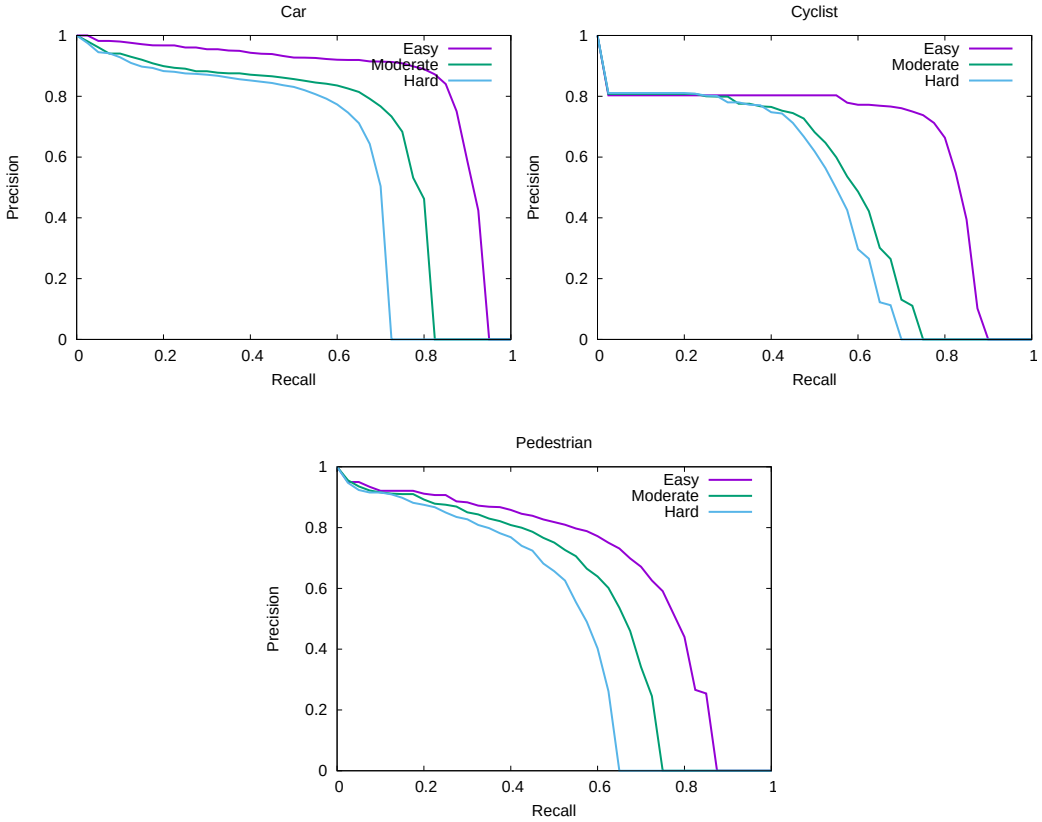


Figure 5.15: Precision-Recall graphs from Lyft 3D detection evaluation, we can deduce that the performance of the model is same as the results in Experiment 5.2

1. The model did fit well, and the Precision-Recall graphs also confirms that the model is performing similar to the fixed-weight model in Experiment 5.2. Hence, we chose to use this model for uncertainty analysis.

2. But, since only the final two stages of the Frustum-PointNet are Bayesian. The epistemic uncertainty obtained from this model will be due to the weights of the spatial-transformer network and bounding box network.

5.5 Extraction of Epistemic Uncertainty and Visualize the Results

The aim of this experiment is to calculate the epistemic uncertainty in the 3D object detection task calculated from the multiple Monte Carlo runs. During these inference runs, weights are sampled from the distribution $p(w)$. The resultant models hold different sampled weights and does form an ensemble like architecture and helps in extracting epistemic uncertainty. The epistemic uncertainty should be quantified using Shannon entropy and Total variance. The values of Shannon entropy should range from zero to 1 $SE \epsilon [0,1]$ and the Total variance should always be positive. We also consider following real-world cases, these cases will help us in understanding the relation between uncertainty and the semantics of the scene.

- The objects near to the ego-vehicle should have lesser epistemic uncertainty.
- In general the number of points in the frustum point cloud related to car are higher than the other agents (i,e cyclist and pedestrian). Hence, we expect the detections related to cars have lesser epistemic uncertainty.
- In case of overlapped objects, the first object should have lesser epistemic uncertainty than the overlapped objects.

5.5.1 Experimental Procedure

The inference loop is run 10 times and detections were captured. Instead of using corners to calculate the epistemic uncertainty as done by *Feng et al* [14], We used the bounding box dimensions, bounding box center and rotation angle to calculate the epistemic uncertainty.

The shannon entropies calculated using the softmax probabilities at each stage and are plotted as follows

Table 5.7: Epistemic uncertainty statistics from the multiple Monte-Carlo runs which shows that the epistemic uncertainty is higher for the pedestrian and the cyclist

Agent	Car	Cyclist	Pedestrian
No of samples	16314	3824	5254
mean variance	1.34	4.48	2.88
mean shannon entropy	0.43	0.88	0.93

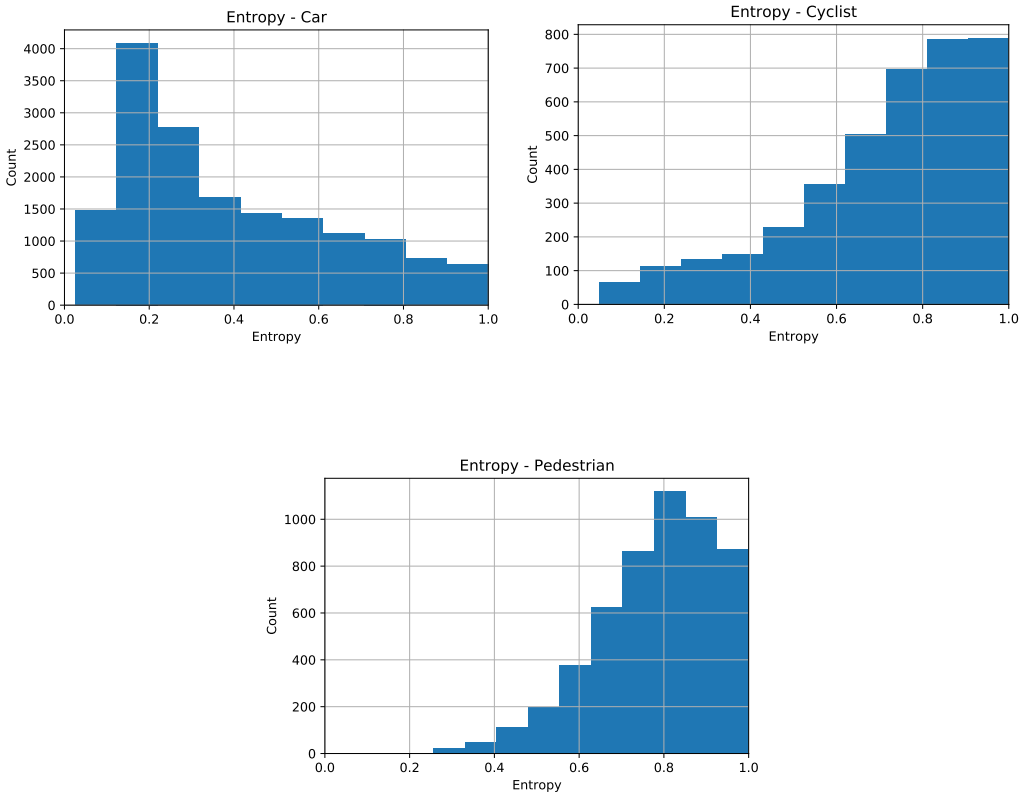


Figure 5.16: Entropies calculated class wise and can be inferred that car class has comparatively low entropy while pedestrians and cyclist has relatively high entropy.

5.5.2 Observations

In this section, we presented various key observations made from the epistemic uncertainty measure for some specific scenes.

The notations presented in the following images for each agent is (object class, detection probability, object entropy). The model is classified to be certain if entropy is less than 0.4 [14].

Table 5.8: Colors chosen based on Shannon-entropy of detection.

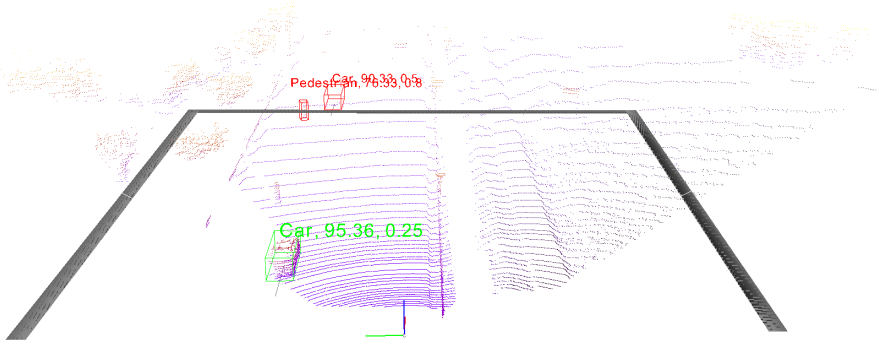
Shannon Entropy (SE)	**Color**
SE < 0.5	Red
$0.5 <$ SE < 0.75	Yellow
SE > 0.75	Green

5.5.2.1 Epistemic uncertainty due to Object being far away from ego-vehicle

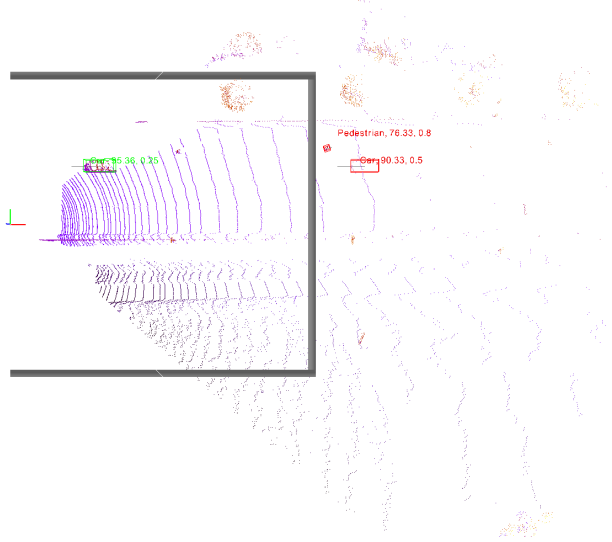
The epistemic uncertainty of the car marked in red is at a distance from the ego-vehicle which might result in

- Reduced number of points in the extracted point cloud.
- Increased in the angle between centroid of the point cloud and the heading angle of the frustum point cloud, which might result in a overlapped bin size

(a) Predictions in image plane.

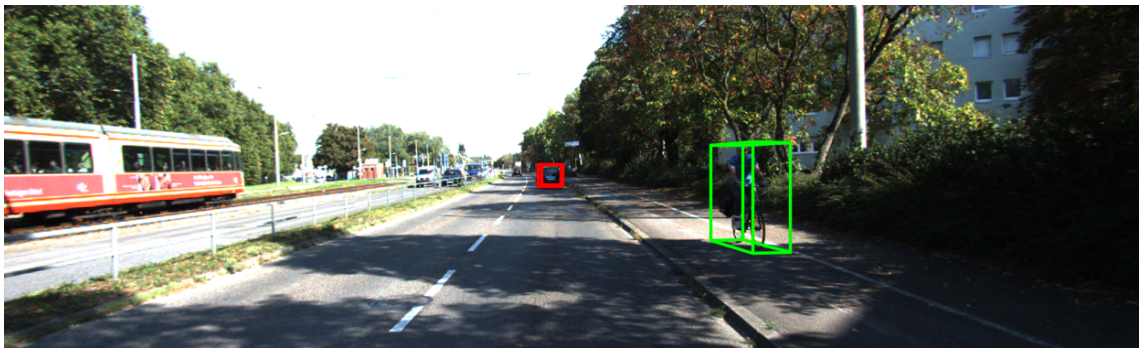


(b) Predictions in LiDAR plane.

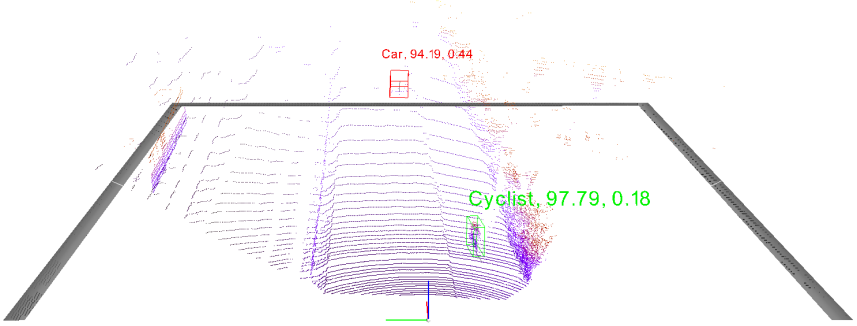


(c) Predictions in BEV projection of LiDAR point cloud.

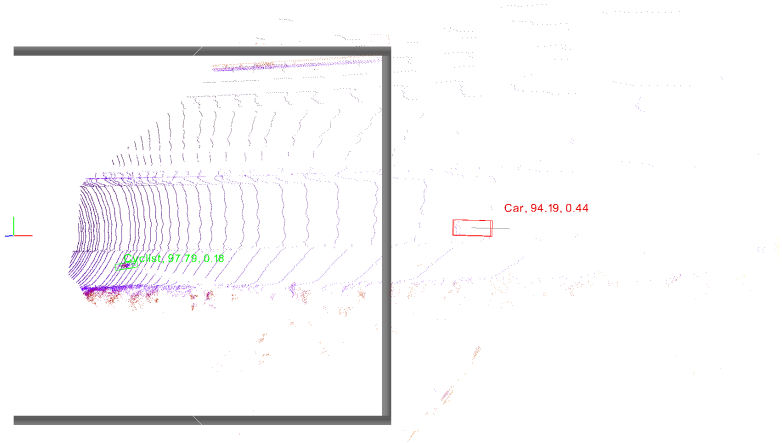
Figure 5.17: Image shows that the epistemic uncertainty of an object increases as the distance from the ego-vehicle increases. The car which is near to the ego vehicle has lower entropy when compared to Pedestrian and Car that are far away.



(a) Predictions in image plane.



(b) Predictions in LiDAR plane.



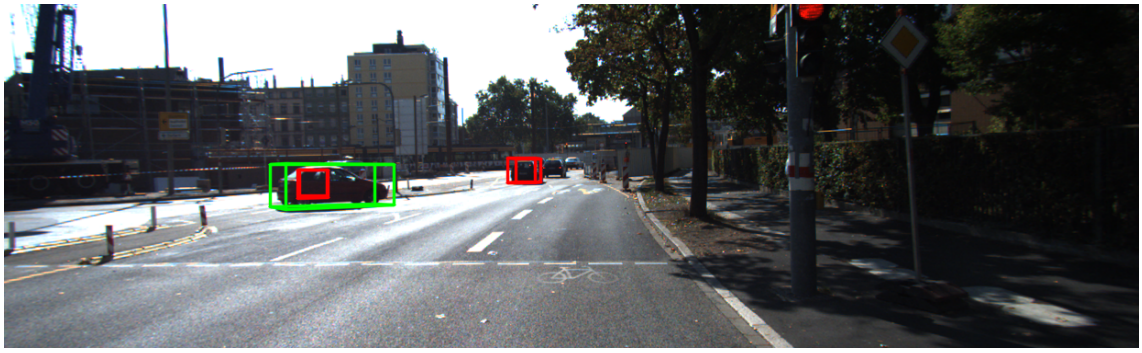
(c) Predictions in BEV projection of LiDAR point cloud.

Figure 5.18: Image shows that the epistemic uncertainty of an object increases as the distance from the ego-vehicle increases. The cyclist near to the ego-vehicle has lower entropy than the car that is far away. This case shows us that the entropy doesn't depend on the agent class but on the distance.

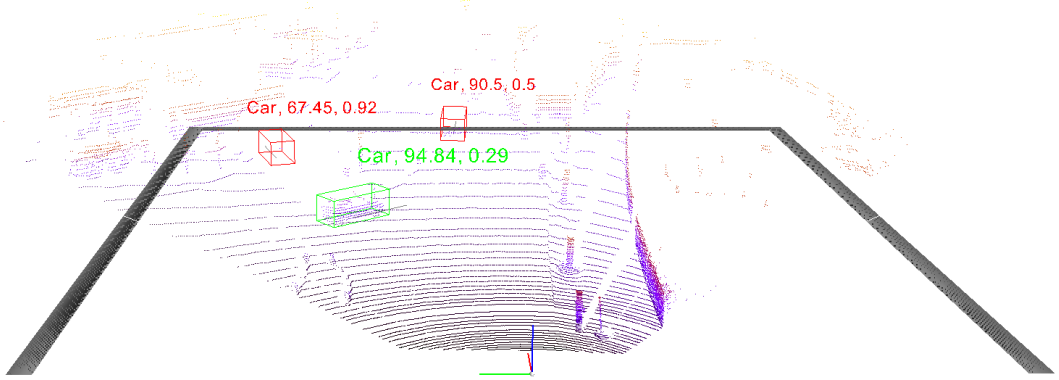
5.5.2.2 Epistemic uncertainty due to one agent blocking the other agent

The epistemic uncertainty of the car marked is high because of the car marked in green is blocking it in Image 5.19. Number of points available are very low in the bounding box, but the detection is made because of the bin size from the class in 2D object detection.

In the scene shown in Figure 5.19, we can observe that one car is blocking the other. Though detection is made but the entropy is 0.92 which says the model is not certain at all with the detection. We can also see that the car which is far away from the ego-vehicle is having high entropy but lesser than the entropy of the car that got blocked

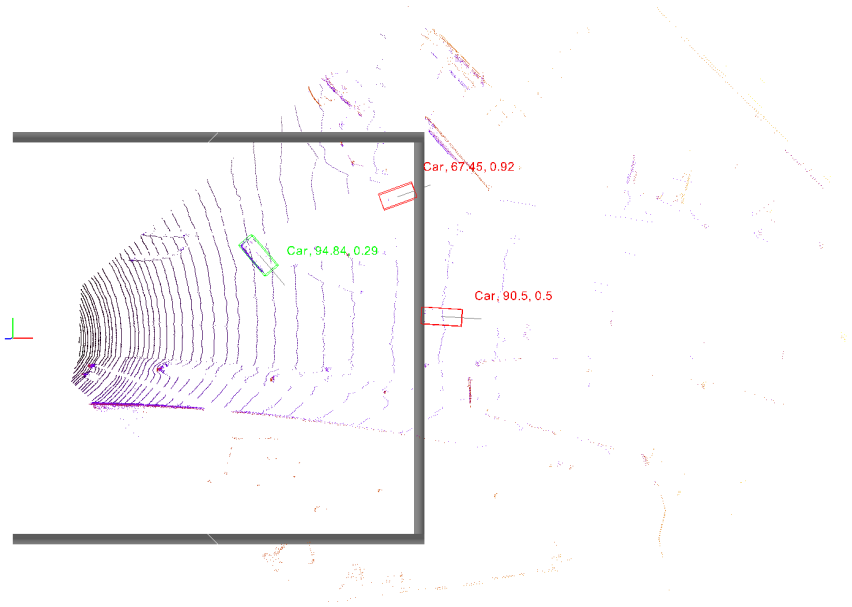


(a) Predictions in image plane.



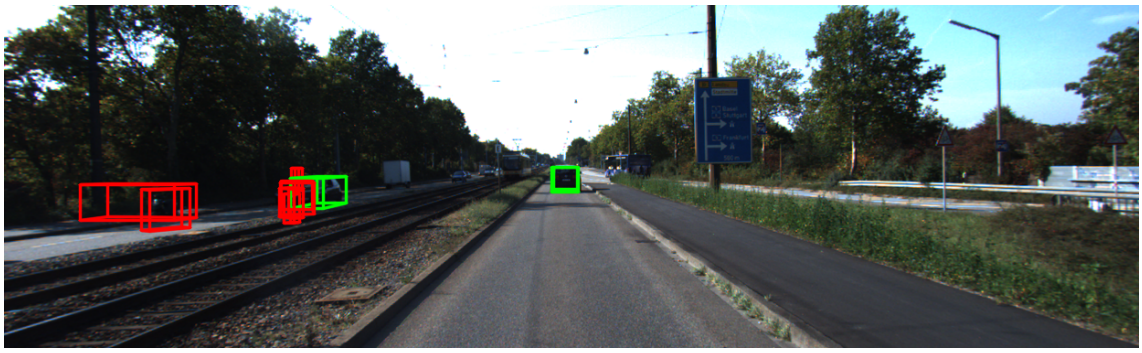
(b) Predictions in LiDAR plane.

Detecting an object even when the number of points available are low is one of the advantages of a Frustum-PointNet model. But the epistemic uncertainty analysis showed that such detection has higher entropy and variance, which means the object

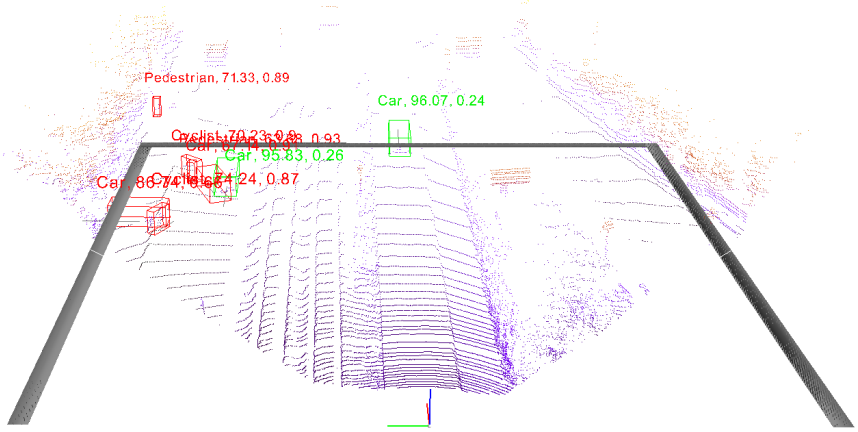


(c) Predictions in BEV projection of LiDAR point cloud.

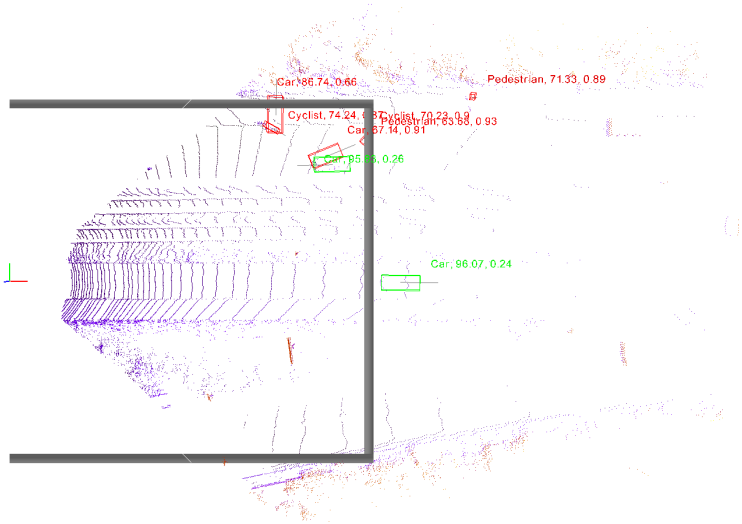
Figure 5.19: Image shows that the epistemic uncertainty is high due to the blockage of one car by another. This case is particularly in our interest since the entropy of object far away is lesser than the entropy of blocked car.



(a) Predictions in image plane.



(b) Predictions in LiDAR plane.



(c) Predictions in BEV projection of LiDAR point cloud.

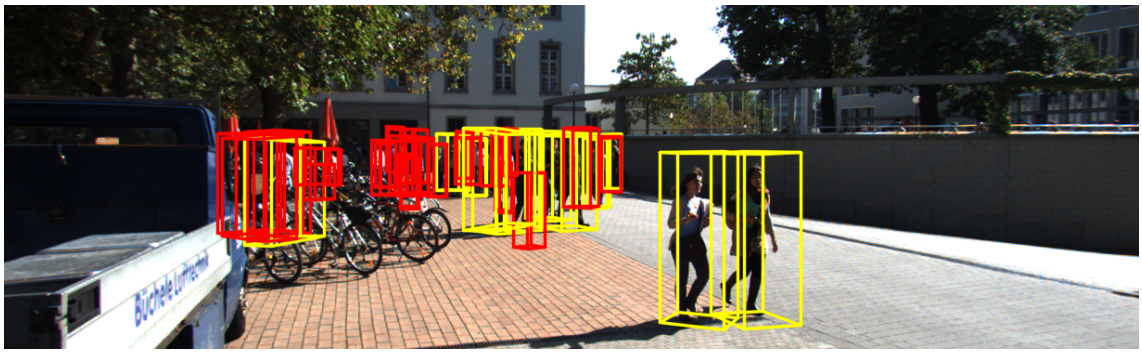
Figure 5.20: Image shows that the epistemic uncertainty is high due to the blockage of one car by another. This case is particularly in our interest since the entropy of object far away is lesser than the entropy of blocked car.

might be present there but its position and orientation are highly unknown and shouldn't be used for further operations in autonomous stack.

5.5.2.3 Epistemic uncertainty due to cluttered environments

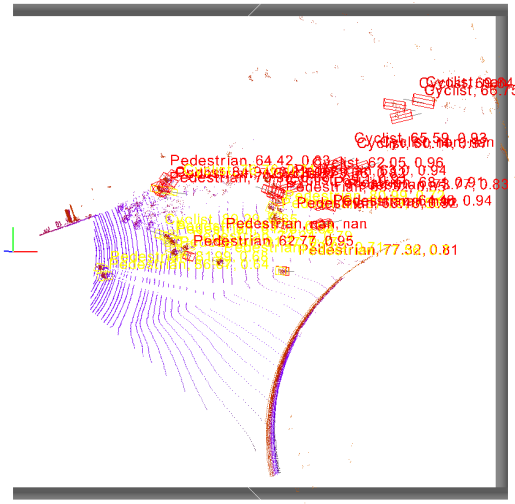
The epistemic uncertainty in a cluttered environment is high in case of every object present in the clutter the entropy in some cases is as high as its maximum value i,e 1.0. The total variance for every object as well is very high when compared to a non-cluttered scene. Because of the overlaps in the detection, the point cloud of one object is affecting the other underlying objects as well which results in high entropy outputs.

The scene in Figures 5.21, 5.22 shows that when there is clutter or a overlap between two detections then the entropies are higher. The case in Figure 5.21 is very interesting because of the Pedestrians are very near to the ego-vehicle but still because of the overlap the entropies are still higher.



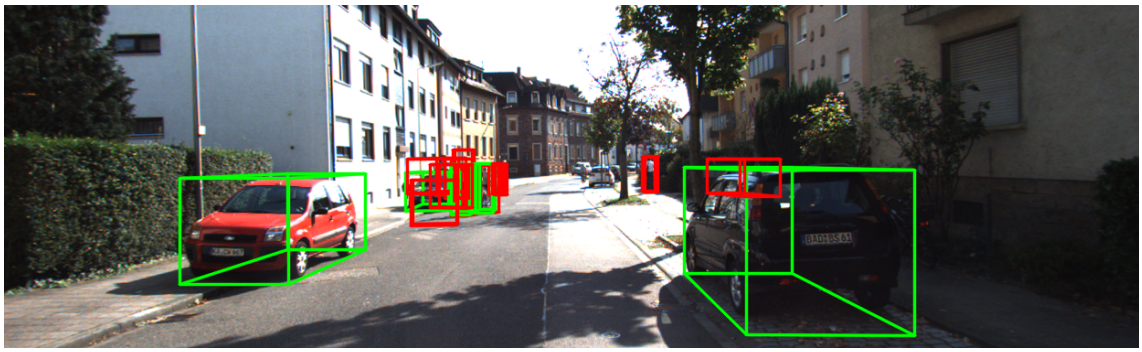
(a) Predictions in image plane.

(b) Predictions in LiDAR plane.

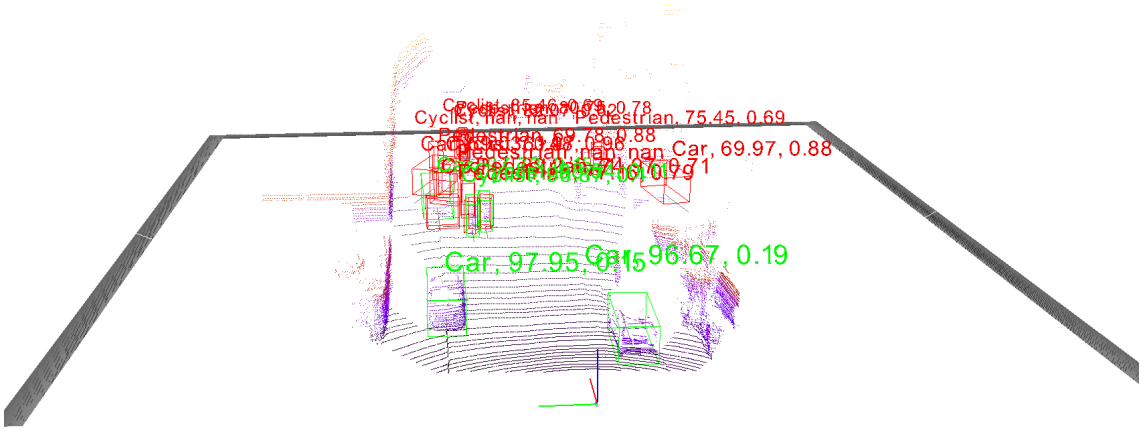


(c) Predictions in BEV projection of LiDAR point cloud.

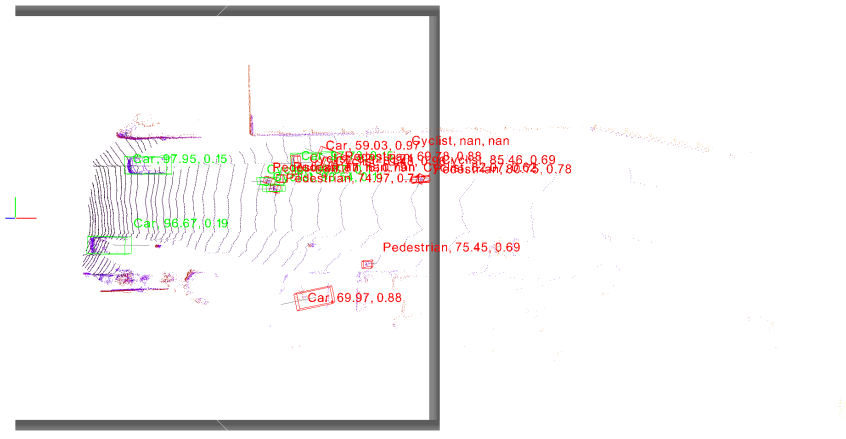
Figure 5.21: Image shows that the epistemic uncertainty is high due to the cluttered environment and even however near the agent is nearer to the ego-vehicle the entropies are still higher



(a) Predictions in image plane.



(b) Predictions in LiDAR plane.



(c) Predictions in BEV projection of LiDAR point cloud.

Figure 5.22: Image shows that the epistemic uncertainty is high due to the cluttered environment.

6

Conclusions

In this research and development project, Frustum-PointNet [36] is modelled to solve the 3-Dimensional (3D) object detection problem in the context of Autonomous driving. The network is trained and tested on Lyft dataset [21]. We observed that the network performed on same level as the network trained using KITTI dataset [11]. The comparison is done based on 3D Average-Precision values calculated using fixed limits set on Intersection-over-Union values.

We also modelled a Bayesian Neural Network based on Frustum-PointNet [36] to extract the uncertainty due to the weights in the Frustum-PointNet [36] model. We also concluded that the Bayesian Neural Network performed better than the Fixed-weight model. But, we observed that the difference in performance is not as expected. We believe that with better tuning of the prior distribution placed on the weights, we can achieve better results.

Further analysis on the extracted uncertainty also showed that the model is over-confident over some samples. We also visualized some samples which represented the models over-confidence and confirm the need for the usage of uncertainty as a measure of confidence.

6.1 Lessons Learned

The lessons learned from the above experiments are as follows,

1. The Bayesian neural network can be modeled but as the complexity of the model

increases (i,e number of trainable parameters) the model tends to under-fit. The reason is still to be investigated.

2. We modelled a partial Bayesian Frustum-PointNet resulting in reduction in number of parameters, this model had performed well and corroborated the claim made in first point.

3. The partial Bayesian model is used to quantify epistemic uncertainty, as we expected initially the entropy in detecting the cyclist and pedestrians is higher than detecting a car.

4. The inference times increased with increase in the number of Monte-Carlo sampling runs. The higher inference times made this approach infeasible for real-time applications like Autonomous driving.

6.2 Future Work

In the future, the possible paths available are

1. We can try a different approach of using a single-stage 3D object detection like Complexer-YOLO [33]. This might help in a reduced number of parameters and full networks epistemic uncertainty quantification can be done.

2. Furthermore, we intended to model the Bayesian Frustum-PointNet using the ensemble [18] or sub-ensemble models [43] which might help in reducing the inference time as well.

3. We can also produce synthetic data using domain adaption techniques in higher uncertainty cases and verify if we can reduce uncertainty or not.

References

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 779–788.

[3] R. Q. Charles, H. Su, M. Kaichun, and L. 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, pp. 77–85.

[4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 5105–5114.

[5] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. 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.

[6] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D Object Detection Network for Autonomous Driving," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6526–6534.

[7] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis, "A survey on 3d object detection methods for autonomous driving applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, pp. 3782–3795, Oct 2019.

[8] A. El Kader Isselmou, G. Xu, S. Zhang, S. Saminu, and I. Javaid, "Deep learning algorithm for brain tumor detection and analysis using mr brain images," in *Proceedings of the 2019 International Conference on Intelligent Medicine and Health*, ser. ICIMH 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 28–32.

[9] Q. Rao and J. Frtunikj, "Deep learning for self-driving cars: Chances and challenges," in *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, ser. SEFAIS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 35–38.

[10] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[11] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[12] Molly Mulshine, "A major flaw in Google's algorithm allegedly tagged two black people's faces with the word 'gorillas'," Online, 2015, (visited on 08/13/2020). [Online]. Available: https://www.businessinsider.com/google-tags-black-people-as-gorillas-2015-7

[13] Sam Levin and Julia C Wong, "Self-driving Uber kills Arizona woman in first fatal crash involving pedestrian," Online, 2018, visited on 08/12/2020.

[14] D. Feng, L. Rosenbaum, and K. Dietmayer, "Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018, pp. 3266–3273.

[15] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 1613–1622.

[16] D. Tran, M. W. Dusenberry, M. V. D. Wilk, and D. Hafner, "Bayesian layers: A module for neural network uncertainty," in *NeurIPS*, 2019.

[17] D. Tran, A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, and D. M. Blei, "Edward: A library for probabilistic modeling, inference, and criticism," *arXiv preprint arXiv:1610.09787*, 2016.

[18] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 6402–6413.

[19] K. Shridhar, F. Laumann, A. L. Maurin, M. Olsen, and M. Liwicki, "Bayesian convolutional neural networks with variational inference," in *Master Thesis*, 2018.

[20] M. Henne, A. Schwaiger, K. Roscher, and G. Weiss, "Benchmarking uncertainty estimation methods for deep learning with safety-related metrics," in *Proceedings of the Workshop on Artificial Intelligence Safety, co-located with 34th AAAI Conference on Artificial Intelligence, SafeAI@AAAI 2020, New York City, NY, USA, February 7, 2020*, ser. CEUR Workshop Proceedings, H. Espinoza, J. Hernández-Orallo, X. C. Chen, S. S. ÓhÉigeartaigh, X. Huang, M. Castillo-Effen, R. Mallah, and J. McDermid, Eds., vol. 2560. CEUR-WS.org, 2020, pp. 83–90.

[21] 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 AV Dataset 2019," online, 2019. [Online]. Available: https://level5.lyft.com/dataset/

[22] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 618–11 628.

[23] Y. Chen, S. Liu, X. Shen, and J. Jia, "Fast Point R-CNN," *ArXiv*, 2019.

[24] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2147–2156.

[25] A. Mousavian, D. Anguelov, J. Flynn, and J. Košecká, "3d bounding box estimation using deep learning and geometry," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 5632–5640.

[26] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teulière, and T. Chateau, "Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1827–1836.

[27] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 1856–1860.

[28] C. C. Pham and J. W. Jeon, "Robust object proposals re-ranking for object detection in autonomous driving using convolutional neural networks," *Image Commun.*, vol. 53, no. C, p. 110–122, Apr. 2017.

[29] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2018, pp. 4490–4499.

[30] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. 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, pp. 1355–1361.

[31] R. Sahba, A. Sahba, M. Jamshidi, and P. Rad, "3d object detection based on lidar data," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, Oct 2019, pp. 0511–0514.

[32] T. Li and X. Zhao, "Cost efficient subcategory-aware cnn for object detection," in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 4202–4206.

[33] M. Simon, S. Milz, K. Amende, and H. Groß, "Complex-yolo: Real-time 3d object detection on point clouds," *ArXiv*, vol. abs/1803.06199, 2018.

[34] Y. Xiang, Wongun Choi, Y. Lin, and S. Savarese, "Data-driven 3d voxel patterns for object category recognition," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1903–1911.

[35] X. Du, M. Ang, S. Karaman, and D. Rus, "A general pipeline for 3d detection of vehicles," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3194–3200, 2018.

[36] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D Object Detection from RGB-D Data," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[37] Z. Wang and K. Jia, "Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection," *ArXiv,*, 2019.

[38] P. Cao, H. Chen, Y. Zhang, and G. Wang, "Multi-View Frustum Pointnet for Object Detection in Autonomous Driving," in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019.

[39] A. Graves, "Practical variational inference for neural networks," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 2348–2356.

[40] S. Kullback and R. Leibler, *Information Theory and Statistics*, , Ed. Dover Publications, 1959.

[41] S. Kumar, L. Felix and L. Marcus, "Uncertainty Estimations by Softplus normalization in Bayesian Convolutional Neural Networks with Variational Inference," *arXiv preprint arXiv:1806.05978,*, 2019.

[42] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2575–2583.

[43] Matias Valdenegro-Toro, "Deep Sub-Ensembles for Fast Uncertainty Estimation in Image Classification," in *Bayesian Deep Learning Workshop. Neural Information Processing Systems (NIPS-2019), befindet sich NeurIPS 2019, December 8-14, Vancouver, BC, Canada*, Dec. 2019.

[44] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5574–5584.

[45] Wray L. Buntine and Andreas S. Weigend, "Bayesian Back-Propagation," *Complex Systems*, vol. 5, p. 603, 1991.

[46] G. E. Hinton and D. van Camp, "Keeping the neural networks simple by minimizing the description length of the weights," in *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, ser. COLT '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 5–13.

[47] Y. Wen, P. Vicol, J. Ba, D. Tran, and R. B. Grosse, "Flipout: Efficient pseudo-independent weight perturbations on mini-batches," *ArXiv*, vol. abs/1803.04386, 2018.

[48] "Waymo Open Dataset: An autonomous driving dataset," https://www.waymo.com/open, 2019, Accessed: 2019-09-15.

[49] J. Geyer, Y. Kassahun, M. Mahmudi, X. Ricou, R. Durgesh, A. S. Chung, L. H. Forest, V. H. Pham, M. Muhlegg, S. Dorn, T. Fernandez, M. Jänicke, S. Mirashi, C. Savani, M. Storm, O. Vorobiov, and P. Schuberth, "A2D2: AEV Autonomous Driving Dataset," online, 2019. [Online]. Available: http://www.a2d2.audi

[50] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2017–2025.

[51] Shuyang Cheng, Yang Song and Peisheng Li, "Using automated data augmentation to advance our Waymo Driver," Technical

Blog, Apr. 2020. [Online]. Available: https://blog.waymo.com/2020/04/using-automated-data-augmentation-to.html

[52] Y. Ge, Y. Xiong, and P. J. From, "Symmetry-based 3d shape completion for fruit localisation for harvesting robots," *Biosystems Engineering*, vol. 197, pp. 188 – 202, 2020.