A study of PointNet and PointNet based approaches for Object Detection and Classification in Autonomous Driving

Abhishek Ranjan Singh
Department of ECE
North Carolina State University Raleigh, NC
arsingh3@ncsu.edu

Adarsh Puri
Department of ECE
North Carolina State University Raleigh, NC
apuri3@ncsu.edu

Abstract- 3-D Object detection and classification are integral parts of Autonomous Driving Systems [4]. In this work we consider the problem of designing a fast yet reliable object detection system for autonomous driving task. 3-D data unlike 2-D data has various representations such as 3-D voxels, point cloud etc. Recent literature suggests that directly using point clouds for performing various tasks such as classification, part segmentation or semantic segmentation results in a faster system which is able to maintain the accuracy to state-of-the-arts standards [1]. In this work we are researching the viability of such claim by first evaluating the original work on point nets and then comparing PointPillars [7], a PointNet based encoder, with other state of the art approaches. PointPillars uses a PointNet based encoder for converting the point cloud into a sparse 2-D image, Extensive experimentation shows that doing this not only increases the speed but also maintains the accuracy to state-of-the-art standards. Despite only using lidar, PointPillar's full detection pipeline significantly outperforms the state of the art, even among fusion methods, with respect to the 3D KITTI benchmark. This detection performance is achieved while running at 62 Hz: a 2-4-fold runtime improvement.

I. Introduction

3-D Object Detection and classification is an integral part of the autonomous driving task. In Autonomous Driving systems, the data is often acquired using sensors called LIDARS (Light Detection and Ranging) which capture images in the form of point clouds. Point Clouds are a set of unordered data points in space taken from the surface of objects. They are simple and unified structures which are easy to learn from. However, Point Clouds are just a set of points and therefore invariant to permutation of its members.

Due to the irregular nature of Point Clouds many researchers prefer converting them into more regular data formats such as 3D Voxels or image grids as required by typical Deep Net Architectures. This in turn renders data unnecessarily voluminous. Using such data reduces the chance of achieving a fast-enough speed.

Driving is a time sensitive task and hence requires quick decision making on the part of driver, this implies that any autonomous driving system not only needs to be accurate but also fast if it wants to surpass or at least reach human level performance.

In their original work Charles Qi and Hao Su claim that PointNet [1] can solve the issue of speed by directly processing point clouds with the help of symmetric functions which are invariant to order of input and hence are effective on point clouds which are unordered data sets. In their original work they show that PointNet can effectively be used for the task of Image classification, Part Segmentation and Semantic Segmentation.

After their work was published various researchers tried to utilize their approach for the task of object detection. Most of the approaches can be divided into one of the two categories; fixed encoders tend to be fast but sacrifice accuracy, while encoders that are learned from data are more accurate, but slower. Among all these approaches PointPillars stands out as it not only outperforms other methods in terms of speed but also is able to reach the state-of-the-art accuracy [Figure 1][7].

In this study we first try to reproduce the results of [1] while studying about what makes this approach work. After that we try to reproduce the results of [7] on KITTI DATASET [6].

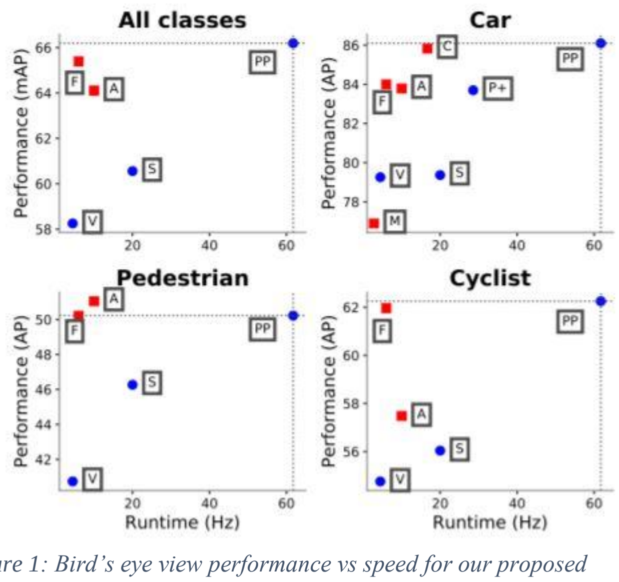
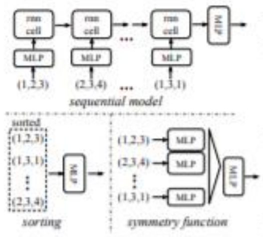


Figure 1: Bird's eye view performance vs speed for our proposed PointPillars, PP method on the KITTI [6] test set. Lidar-only methods drawn as blue circles; lidar & vision methods drawn as red squares. Also drawn are top methods from the KITTI leader-board: M: MV3D [8], An AVOD [9], C: ContFuse [10], V: VoxelNet [11], F: Frustum PointNet [12], S: SECOND [13], P+ PIXOR++ [14]. PointPillars [7] outperforms all other lidar-only methods in terms of both speed and accuracy by a large margin. It also outperforms all fusion based method except on pedestrians. Similar performance is achieved on the 3D metric (Table 1).

II. Deep Learning on Unordered Datasets

From a data structure point of view, a point cloud is an unordered set of vectors. While most works in deep learning focus on regular input representations like sequences (in speech and language processing), images and volumes (video or 3D data), not much work has been done in deep learning on point sets.

In order to make a model invariant to input permutation, three strategies exist: 1) sort input into a canonical order; 2) treat the input as a sequence to train an RNN, but augment the training data by all kinds of permutations; 3) use a simple symmetric function to aggregate the information from each point. Here, a symmetric function takes n vectors as input and outputs a new vector that is invariant to the input order. For example, + and ∗ operators are symmetric binary functions. While sorting sounds like a simple solution, in high dimensional space there in fact does not exist an ordering that is stable w.r.t. point perturbations in the general sense. This can be easily shown by contradiction. If such an ordering strategy exists, it defines a bijection map between a high-dimensional space and a 1d real line. It is not hard to see, to require an ordering to be stable w.r.t point perturbations is equivalent to requiring that this map preserves spatial proximity as the dimension reduces, a task that cannot be achieved in the general case. Therefore, sorting does not fully resolve the ordering issue, and it's hard for a network to learn a consistent mapping from input to output as the ordering issue persists. As shown in experiments [Figure 2], it was found that applying an MLP directly on the sorted point set performs poorly, though slightly better than directly processing an unsorted input.



	accuracy
MLP (unsorted input)	24.2
MLP (sorted input)	45.0
LSTM	78.5
Attention sum	83.0
Average pooling	83.8
Max pooling	**87.1**

Figure 2: Three approaches to achieve order invariance. Multilayer perceptron (MLP) applied on points consists of 5 hidden layers with neuron sizes 64,64,64,128,1024, all points share a single copy of MLP. The MLP close to the output consists of two layers with sizes 512,256.

The idea to use RNN considers the point set as a sequential signal and hopes that by training the RNN with randomly permuted sequences, the RNN will become invariant to input order. However in "OrderMatters" [21] the authors have shown that order does matter and cannot be totally omitted. While RNN has relatively good robustness to input ordering for sequences with small length (dozens), it's hard to scale to thousands of input elements, which is the common size for point sets.

III. PointNet: A Neural Network Architecture for Point Clouds (Unordered Data)

PointNet tries to use the third approach that is use of symmetric function for achieving order invariance.

A. The Model

The full PointNet network architecture is visualized in [Figure 3], where the classification network and the segmentation network share a great portion of structures. Please read the caption of Figure 3 [1] for the pipeline. This network has three key modules: the max pooling layer as a symmetric function to aggregate information from all the points, a local and global information combination structure, and two joint alignment networks that align both input points and point features.

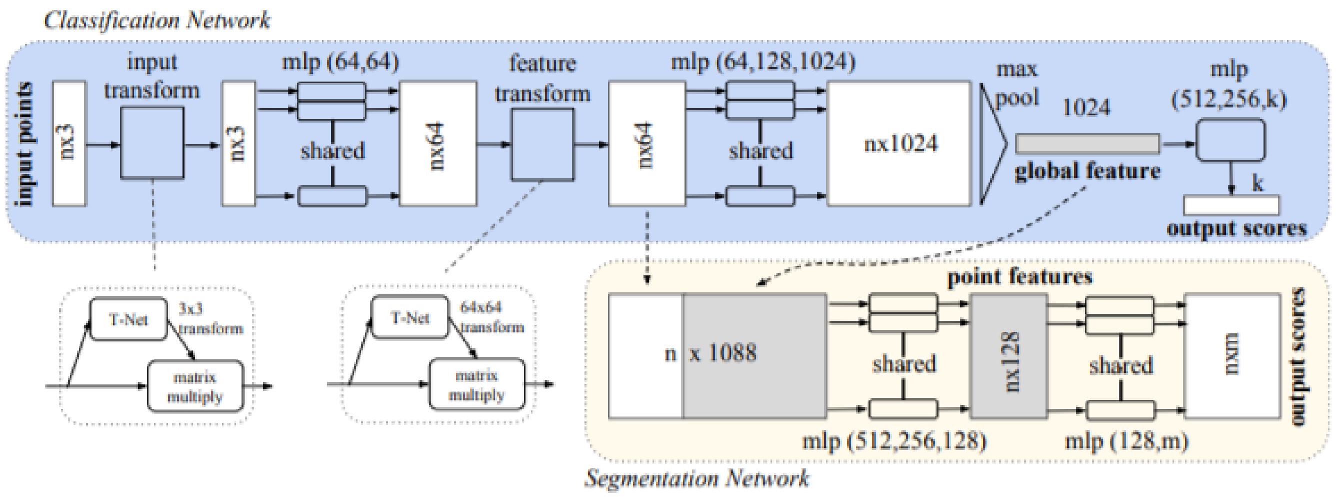


Figure 3: PointNet Architecture. The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

B. *Experimentation details*

We are using the same architecture as [1] for the task of classification. We are emulating the experiment conditions of [1] including the dataset (ModelNet40 [23] for classification and Standard 3D semantic parsing dataset [22] for semantic segmentation). To produce these results, we used GTX 1660 Ti and Tensorflow version 1.2 on Linux 16.0.4.

C. *Results for the task of classification*

The results are represented in [1]. As you can see that point net does perform as was mentioned in [1].

Table 1:Classification results on ModelNet40[23] Dataset

Model	Input	Accuracy avg. class	Accuracy overall
Original PointNet	Point Cloud	86.2	89.2
Our PointNet	Point Cloud	83.33	88.6

D. *Results for the task of semantic segmentation*

The results are represented in [1]. As you can see that point net does perform as was mentioned in [1]

Table 2: Results on semantic segmentation in scenes. Metric is average IoU over 13 classes (structural and furniture elements plus clutter) and classification accuracy calculated on points

Models	Mean IoU	Overall accuracy
Original PointNet	47.71	78.62
Ours PointNet	45.55	77.23

IV. POINTPILLARS

In this section we discuss in details about how point pillars are used to for the task of object detection. We compare our results with original work. We have virtually used the same structure and dataset to produce our results as was mentioned in [].

A. *The Model*

The PointPillar architecture has 3 stages; Pillar Feature Net, Backbone 2D CNN and the Detection Head [Figure 2]. For the Pillar Feature net, we have used a pretrained PointNet based autoencoder and have used KITI dataset to train the network.

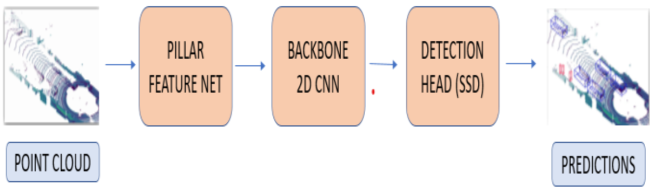


Figure 4:Model Architecture with processing Point Cloud as input and showing predictions for cars, pedestrians and cyclists as output

PointPillars accepts Point Clouds as input and encode them into pseudo images. This is done by the Feature encoder (Pillar Feature Net). The pseudo image is then fed into a 2D Convolutional Backbone followed by a Detection Head (Single Shot Detector for our project). These output bounding boxes and k scores for classification of cars, pedestrians and cyclists. Thus, it also eliminates the problem of dealing with expensive 3D convolutional layers.

1. Pillar Feature Net: To apply a 2D convolution architecture, the Pillar Feature net has to convert the input Point Cloud to a pseudo image. Each point in the Point Cloud is represented by the coordinate L(x,y,z). First the Point Cloud is discretized into evenly spaced grids in the x-y plane creating a set of pillars as shown below.

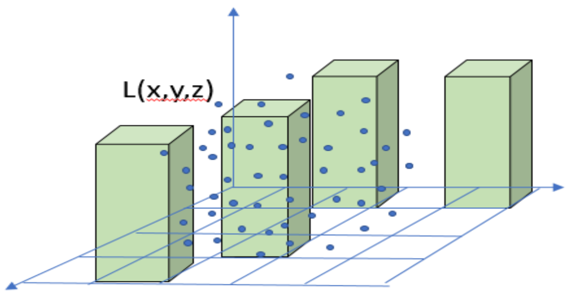


Figure 5: Vertical Pillars constructed from Point Clouds

The point is then augmented to (x_c ,y_c ,z_c), and (x_p ,y_p) where c denotes distance to arithmetic mean of all points in a pillar and p denotes offset from the pillar center. Hence each point L becomes D=9 dimensional. This sparsity of the Point Clouds is exploited by imposing a limit both on the number of non-empty pillars per sample (P) and on the number of points per pillar (N) to create a dense tensor of size (D, P, N). A simplified version of PointNet (PointNet Vanilla []) is then used for each point, where for each point a linear layer is applied followed by BatchNorm and ReLU to generate a (C, P, N) sized tensor. In this project we are using the weights of a pretrained encoder part of PointNet based autoencoder [15] to initialize the above mentioned PointNet part of our Pillar feature extractor. This is followed by a max operation over the channels to create an output tensor of size (C, P). Once encoded, the features are scattered back to the original pillar locations to create a pseudo-image of size (C, H, W) where H and W indicate the height and width of the canvas.

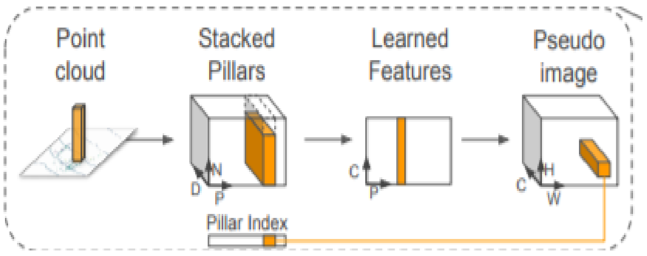


Figure 6: Pillar feature Net Architecture

2. 2D Convolution Backbone: The backbone has two sub-networks: one top-down network that produces features at increasingly small spatial resolution and a second network that performs upsampling and concatenation of the top-down features as shown below.

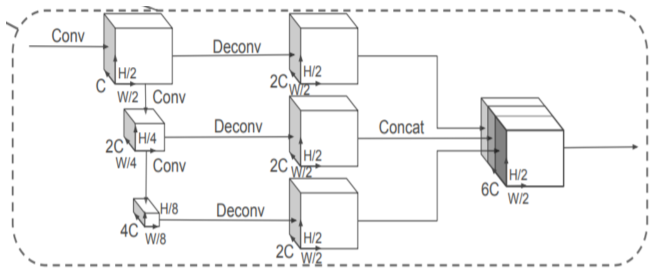


Figure 7: 2D Convolution Backbone

The top-down backbone can be characterized by a series of blocks Block(S, L, F). Each block operates at stride S (measured relative to the original input pseudo-image). A block has L 3x3 2D conv-layers with F output channels, each followed by BatchNorm and a ReLU. The first convolution inside the layer has stride S Sin to ensure the block operates on stride S after receiving an input blob of stride Sin. All subsequent convolutions in a block have stride 1. The final features from each top-down block are combined through upsampling and concatenation as follows. First, the features are upsampled, Up (Sin, Sout, F) from an initial stride Sin to a final stride Sout (both again measured wrt. the original pseudo-image) using a transposed 2D convolution with F final features. Next, BatchNorm and ReLU are applied to the upsampled features. The final output features are a concatenation of all features from different strides.

3. Detection Head: In this project, we use the Single Shot Detector (SSD) [17] setup to perform 3D object detection. We match the prior boxes to the ground truth using 2D Intersection over Union (IoU) [18] similar to SSD. Bounding box height and elevation are not used for matching; instead given a 2D match, the height and elevation become additional regression targets.
   a. SSD Architecture: The SSD follows feed-forward convolutional network approach that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections. The early network layers are based on a standard architecture used for high quality image classification (truncated before any classification layers), called the base network. Auxiliary structure is then added to the network to produce detections. We add convolutional feature layers to the end of the truncated base network. These layers decrease in size progressively and allow predictions of detections at multiple scales. The convolutional model for predicting detections is different for each feature layer. Each added feature layer produces a fixed set of detection predictions using a set of convolutional filters as shown on top of the SSD network architecture in Fig. 9. For a feature layer of size m × n with p channels, the basic element for predicting parameters of a potential detection is a 3 × 3 × p small kernel that produces either a score for a category, or a shape offset relative to the default box coordinates. At each of the m × n locations where the kernel is applied, it produces an output value. The bounding box offset output values are measured relative to a default box position relative to each feature map location.

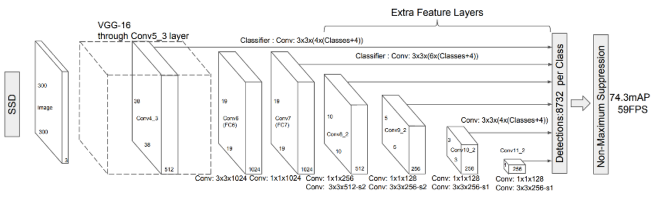


Figure 8: SSD Architecture

   b. Matching Approach In SSD: During training we determine which default boxes correspond to a ground truth detection and train the network accordingly. For each ground truth box, we are selecting from default boxes that vary over location, aspect ratio, and scale. We begin by matching each ground truth box to the default box with the best jaccard overlap. Unlike MultiBox, we then match default boxes to any ground truth with jaccard overlap higher than a threshold (0.5). This simplifies the learning problem, allowing the network to predict high scores for multiple overlapping default boxes rather than requiring it to pick only the one with maximum overlap.

B. *Baseline*

There are various models which can be used to do object detection and classification with the help of LIDAR data. Some methods even use a combination of LIDAR data (Point Clouds) and camera data (Images). Since we are only using LIDAR for training our model, we decided to compare the result with other models which only use LIDAR.

We are using the results which are mentioned at [6] for the VoxelNet [11], SECOND [13] and PointPillars [7] on 3-D Detection Benchmark. We are mainly focusing our comparison with the results presented in [7] for PointPillars on KITTI DATASET.

V. Experimental Section

We are using the codebase of Second available at [16] for producing our results. We have produced our results on NVIDIA GTX 1080 Ti with the help of Pytorch version 1.3.1 for training the model (The reported speed is obtained with the help of TensorRT, which is a library for optimized GPU inference).

A. Metrics

There are various metrics which can be used to evaluate the performance of a 3-D object detection task such as bird's eye view (BEV), 3D, 2D, and average orientation similarity (AOS). We are using 3D-detection benchmark as mentioned by KITTI detection benchmark [6] for evaluation of our model. We are also comparing the training time required for each model.

B. Model Selection

For training our model we are essentially using the same hyperparameters as mentioned in [7]. As with one exception we are essentially using the same structure as [7].

a. Network:
Most of the (except PointNet part of Pillar Feature Net) weights in our network were initialized randomly using a uniform distribution [19]. The PointNet part of the Pillar Feature Net was initialized using the weights from [15].
Similar to [7] The encoder network has C = 64 output features. The car and pedestrian/cyclist backbones are the same except for the stride of the first block (S = 2 for car, S = 1 for pedestrian/cyclist). Both network consists of three blocks, Block1(S, 4, C), Block2(2S, 6, 2C), and Block3(4S, 6,4C). Each block is upsampled by the following upsampling steps: Up1(S, S, 2C), Up2(2S, S, 2C) and Up3(4S, S, 2C). Then the features of Up1, Up2 and Up3 are concatenated together to create 6C features for the detection head.

b. Loss:
We use the same loss functions introduced in SECOND [13]. Ground truth boxes and anchors are defined by (x,y,z,w,l,h,θ). The localization regression residuals between ground truth and anchors are defined by:

$$\Delta x = \frac{x^{gt} - x^a}{d^a}, \Delta y = \frac{y^{gt} - y^a}{d^a}, \Delta z = \frac{z^{gt} - z^a}{h^a}$$
$$\Delta w = \log \frac{w^{gt}}{w^a}, \Delta l = \log \frac{l^{gt}}{l^a}, \Delta h = \log \frac{h^{gt}}{h^a}$$
$$\Delta \theta = \sin\left(\theta^{gt} - \theta^a\right),$$

where x^{gt} and x^a are respectively the ground truth and anchor boxes and $d^a = \sqrt{(w^a)^2 + (l^a)^2}$. The total localization loss is:

$$\mathcal{L}_{loc} = \sum_{b \in (x,y,z,w,l,h,\theta)} \text{SmoothL1}\left(\Delta b\right)$$

Since the angle localization loss cannot distinguish flipped boxes, we use a softmax classification loss on the discretized directions [13], L_{dir}, which enables the network to learn the heading. For the object classification loss, we use the focal loss [20]:

$$\mathcal{L}_{cls} = -\alpha_a \left(1 - p^a\right)^\gamma \log p^a,$$

where p^a is the class probability of an anchor. We use the original paper settings of $\alpha = 0.25$ and $\gamma = 2$. The total loss is therefore:

$$\mathcal{L} \ = \frac{1}{N_{pos}} \left(\beta_{loc}\mathcal{L}_{loc} + \beta_{cls}\mathcal{L}_{cls} + \beta_{dir}\mathcal{L}_{dir}\right),$$

where N_{pos} is the number of positive anchors and $\beta_{loc} = 2$, $\beta_{cls} = 1$, and $\beta_{dir} = 0.2$.
To optimize the loss function, we use the Adam optimizer with an initial learning rate of $2*10^{-4}$ and decay the learning rate by a factor of 0.8 every 15 epochs and train for 100 epochs. We use a batch size of 2 for validation set and 4 for our test submission.

c. Settings:
Unless explicitly varied in an experimental study, we use an xy resolution: 0.16m, max number of pillars(P): 12000, and max number of points per pillar (N): 100. We use the same anchors and matching strategy as [11]. Each class anchor is described by a width, length, height, and z center, and is applied at two orientations: 0 and 90 degrees. Anchors are matched to ground truth using the 2D IoU with the following rules. A positive match is either the highest with a ground truth box, or above the positive match threshold, while a negative match is below the negative threshold. All other anchors are ignored in the loss. At inference time we apply axis aligned non maximum suppression (NMS) with an overlap threshold of 0.5 IoU. This provides similar performance compared to rotational NMS, but is much faster. Car. The x, y, z range is [(0, 70.4), (-40, 40), (-3, 1)] meters respectively. The car anchor has width, length, and height of (1.6, 3.9, 1.5) m with a z center of -1 m. Matching uses positive and negative thresholds of 0.6 and 0.45. Pedestrian & Cyclist. The x, y, z range of [(0, 48), (-20,20), (-2.5, 0.5)] meters respectively. The pedestrian anchor has width, length, and height of (0.6, 0.8, 1.73) meters with a z center of -0.6 meters, while the cyclist anchor has width, length, and height of (0.6, 1.76, 1.73) meters with a z center of -0.6 meters. Matching uses positive and negative thresholds of 0.5 and 0.35.

C. Performance and Comparision to Baseline

In this section we present the results of our model and compare it with the literature.

a. Quantitative Analysis: The KITTI dataset is stratified into easy, moderate, and hard difficulties and the official KITTI leaderboard is ranked by performance on moderate.
As shown in Table [1], Our Model performs on par with original PointPillars and outperforms other methods in terms of speed and accuracy. It also requires relatively a smaller number of epochs (100) and time for reaching the same performance compared to original model.

Table 3: Result on KITTI test 3D detection Benchmark

Method	Modality	Speed	MAP	Car			Pedestrian			Cyclist		
				Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
VoxelNet	Lidar	4.4	49.05	77.47	65.11	57.73	39.48	33.69	31.5	61.22	48.36	44.37
SECOND	Lidar	20	56.69	**83.13**	73.99	66.22	51.07	42.56	37.29	70.51	53.85	46.90
PointPillars	Lidar	62	59.20	79.05	**74.99**	**68.30**	52.08	**43.53**	41.49	**75.78**	59.07	**52.92**
PointPillars(Our)	Lidar	**62**	**59.45**	81.05	71.99	67.23	**54.89**	42.00	**43.01**	73.28	**60.00**	51.00

b. Qualitative Analysis: We provide qualitative result in Figure [7]. The figure inference has a lot of red boxes but if we remove the ones which have a score of less than 0.5 we will be able to get the a pretty good result. Due to the limitations of Kitt Viewer [16] we weren't able to do that.

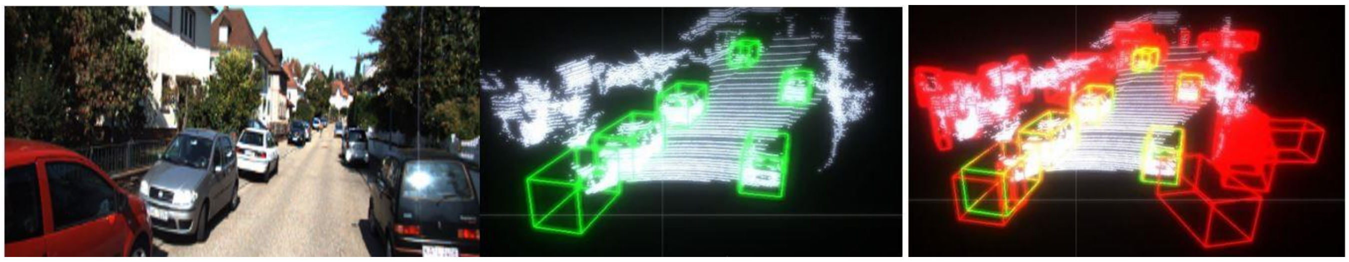


Figure 9: First pic is the original pic, second is the annotated Lidar Point Cloud and the third one is the prediction result

References

[1] R. Q. Charles, H. Su, M. Kaichun and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 77-85.

[2] https://www-fars.nhtsa.dot.gov/Main/index.aspx

[3] Department for Transport, "Research on the Impacts of Connected and Autonomous Vehicles (CAVs) on Traffic Flow: Summary Report".

[4] A. C. Serban, E. Poll and J. Visser, "A Standard Driven Software Architecture for Fully Autonomous Vehicles".

[5] P. Viswanath, S. Nagori, M. Mody, M. Mathew and P. Swami, "End to End Learning based Self-Driving using JacintoNet".

[6] http://www.cvlibs.net/datasets/kitti/

[7] https://arxiv.org/abs/1812.05784

[8] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In CVPR, 2017

[9] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander. Joint 3d proposal generation and object detection from view aggregation. In IROS, 2018

[10] M. Liang, B. Yang, S. Wang, and R. Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In ECCV,2018

[11] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In CVPR, 2018

[12] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. In CVPR, 2018

[13] Y. Yan, Y. Mao, and B. Li. SECOND: Sparsely embedded convolutional detection. Sensors, 18(10), 2018

[14] B. Yang, M. Liang, and R. Urtasun. HDNET: Exploiting HD maps for 3d object detection. In CoRL, 2018

[15] https://github.com/charlesq34/pointnet-autoencoder

[16] https://github.com/traveller59/second.pytorch

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. In ECCV, 2016.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. International Journal of Computer Vision, 2010

[19] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In ICCV, 2015

[20] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. PAMI, 2018

[21] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. arXiv preprint arXiv:1511.06391, 2015.

[22] http://buildingparser.stanford.edu/images/3D_Semantic_Parsing.pdf

[23] https://modelnet.cs.princeton.edu/