Pointwise and Instance Segmentation for 3D Point Cloud

Sanket Gujar

A Thesis submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

April 2019

APPROVED:

[illegible]

Professor Michael Gennert

[illegible]

Professor Jacob Whitehill

[illegible]

Professor Berk Calli

Abstract

The camera is the cheapest and computationally real-time option for detecting or segmenting the environment for an autonomous vehicle, but it does not provide the depth information and is undoubtedly not reliable during the night, bad weather, and tunnel flash outs. The risk of an accident gets higher for autonomous cars when driven by a camera in such situations. The industry has been relying on LiDAR for the past decade to solve this problem and focus on depth information of the environment, but LiDAR also has its shortcoming. The industry methods commonly use projections methods to create a projection image and run detection and localization network for inference, but LiDAR sees obscurants in bad weather and is sensitive enough to detect snow, making it difficult for robustness in projection based methods.

We propose a novel pointwise and Instance segmentation deep learning architecture for the point clouds focused on self-driving application. The model is only dependent on LiDAR data making it light invariant and overcoming the shortcoming of the camera in the perception stack. The pipeline takes advantage of both global and local/edge features of points in points clouds to generate high-level feature. We also propose Pointer-Capsnet which is an extension of CapsNet for small 3D point clouds.

Acknowledgments

I would like to express my gratitude to my advisor Prof. Michael Gennert and Prof. Jacob Whitehill for their guidance, direction, and support for the duration of this work. I would like to thank Prof. Berk Calli for being part of the committee. I would like to thank my lab members for their constant support and guidance. I would also like to acknowledge the Academic & Research Computing group at Worcester Polytechnic Institute for providing computing support that contributed to the results reported within this report. Lastly, I would like to thank my family and friends for their support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The camera is the cheapest and computationally real time option for detecting or segmenting the environment for an autonomous vehicle, but it does not provide depth information and is surely not reliable during night, bad weather, tunnel flash out etc. The risk of an accident gets higher for autonomous cars when driven with a camera in such situations. The industry has been relying on LiDAR for the past decade to solve this problems and focus on depth information of the environment. Industry leaders like Waymo and Uber ATG now even started using there in-house manufactured LiDARs instead of relying on other manufacturers.

There has been lot of attention given to research and investment in the autonomous vehicle industry which lead to significant improvement in LiDAR technology. The LiDAR market is going to hit **5,204.8 Million** USD soon and new

Figure 1.1: Camera Challenges: Bad weather and dynamic lighting condition are a big challenge for camera based autonomous driving.

silicon valley startups like Luminar Technologies and Ouster have released their LiDAR product which can detect up to 200m and are giving a good competition to the currently dominating LiDAR company Velodyne.

The Investment, resources and the drastic improvement ensures the technology will be a dominating factor for the autonomous vehicle industry in the future, but the issue commonly faced with LiDAR is more on the computational efficiency of perception stack. Running the current pointwise algorithm [9, 10] in real time with the input feed of more than 1M points per second needs lots of hardware optimization. Pointwise classification or segmentation with so many points is computationally expensive and will not be real time.

It is possible to build a self-driving car with camera alone but it will not always be safe and having safe autonomy means driving with fewer errors. Different sensor modalities have different strengths and weaknesses. Cameras suffer from difficulty to adjust and detect object in low-light and high dynamic range scenarios. Radar suffers from limited resolution and artifacts due to multi-path and doppler ambiguity. LiDAR sees obsurants as it is sensitive enough to detect snow, making it difficult for robustness in projection based methods.

1.1 Projection methods

The Projection methods [11, 1]are the fastest methods for object detection for autonomous cars but are not robust enough. In projection methods, the points are projected on a plane to obtain a 2d projected image and the model tries to localize the target given the projected image. The most common projection used are birds eye view projection and front projection. The front projection is made by projecting all the points on the front view plane. The points will not overlap over each other but, if two target objects lie close enough they might be projected as a single entity in the front projection confusing the model. Birds eye projection is made by projecting all the points on the ground plane. Here points may overlap over each other and the priority of the class of the points decides which point will be projected on the plane.

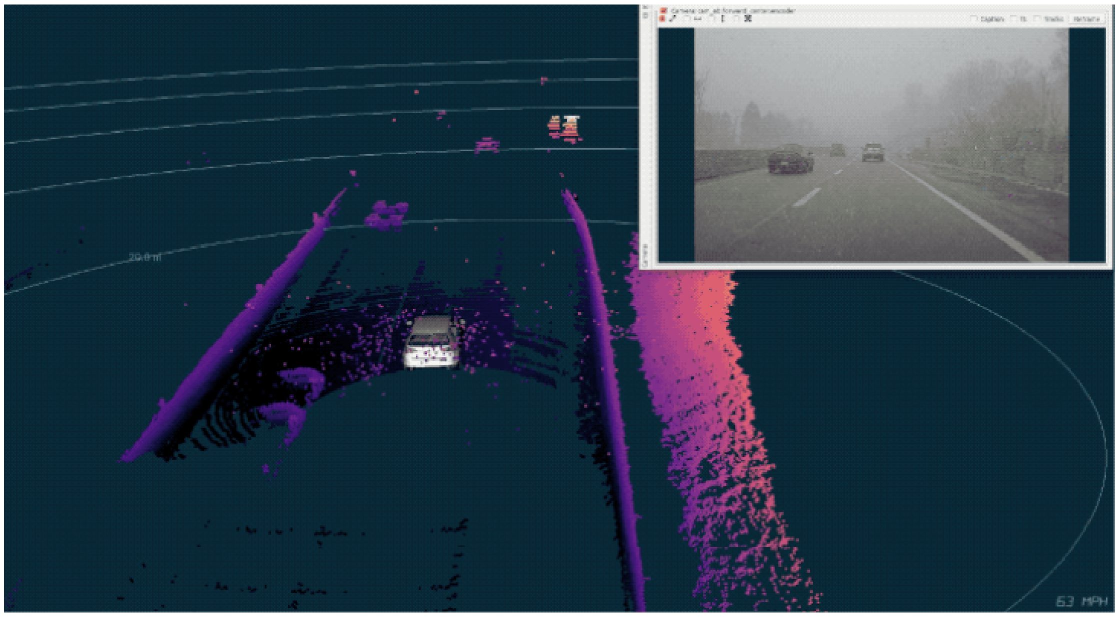


Figure 1.2: LiDAR sensitive to snow which makes it difficult for projection methods to detect vehicles

It is fast for real time application and has shown good results [5] with detecting vehicles but it still has drawbacks (fig: 1.3, 1.4) making it not reliable. [1] takes Birds eye projection as well as the Front projection and fuses the features with RGB image features to regress the 3d box coordinates

1.1.1 When will projection based methods fail

The projection methods are fast but are likely to fail given:

1. The vehicles are close to one another making their projections to look like a single object in the front projection.

2. Pedestrians and poles are represented as a small compact cluster in birds eye view projection making it hard for the model to detect them (fig 1.4)

3. The difficulty of differentiating between large and small vehicles just on projections as the back part of vehicles is projected just as a line for any vehicle

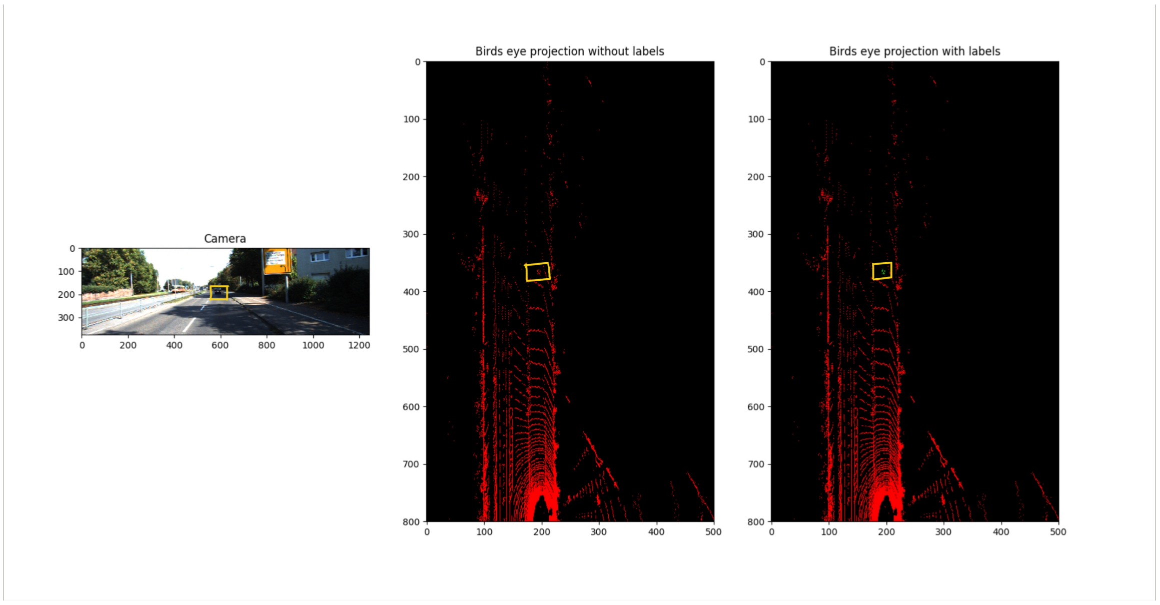


Figure 1.3: Birds eye view projection drawbacks
The car (yellow box) is represented by a small number of points in the point cloud and birds eye projections samples down it even further making it difficult to detect by the model.

irrespective of their dimensions.

4. Weather conditions like rain and snow as LiDAR is sensitive enough to detect this point adding a uniform noise distribution in the projections.

1.2 Proposal

We believe an independant LiDAR perception stack will be more robust to the challenges faced by autonomous vehicles given its scan invariance to the environment lighting condition. If we summarize the section 1.1, most of the methods just evaluate their model on a point cloud with less than ten thousand points and also, there is no method that did 3D semantic segmentation for a large point cloud with more than 60 thousand points per frame. Given the LiDAR point cloud will have at least have more than 60 thousand points per frame, we need a more scalable algorithm to deal

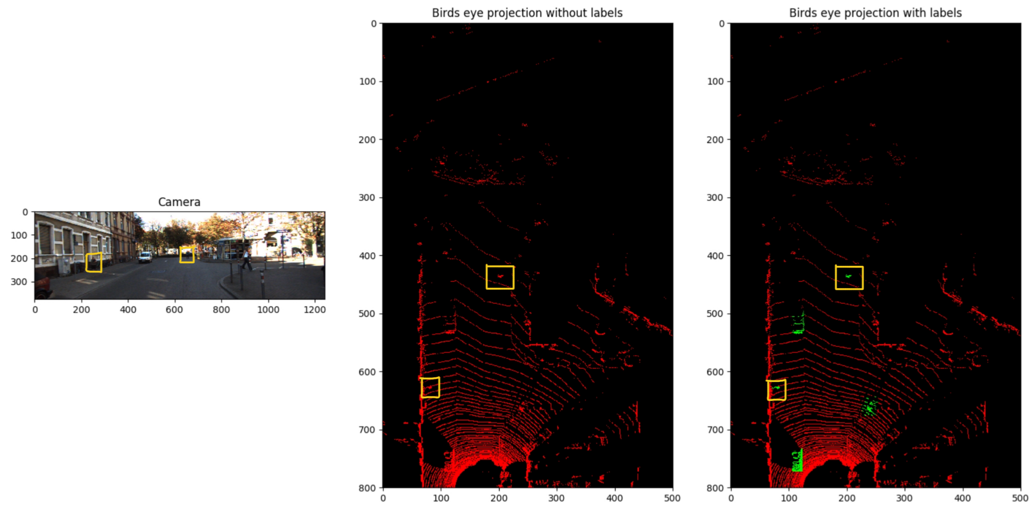


Figure 1.4: Birds eye view projection drawbacks
The pedestrains (yellow boxes) are respresented as a small compact cluster in the projection.

with this scenario. Considering scenarios where lighting conditions are not appropriate for camera model to work robustly, we need a pipeline that should be invariant to the environment conditions to work appropriately. Implementing a pipeline that works on LiDAR data alone will provide us the same result irrespective of the lighting conditions. The pipeline will be more reliable for driving at night as the scans will be not affected by the day/night conditions. Most of the autonomous vehicles pipeline also deploys two independent models for perception using camera and LiDAR and use their output to cross-verify the detections. We focused on doing semantic segmenting for vehicles and pedestrians from the lidar point clouds.

Along with segmentation, instance segmentation is also an important issue for LiDAR data. We proposed a image independent pipeline for instance segmentation for LiDAR. We believe a centroid is a distinctive character for each instance in the point cloud. We tried to predict the segment id as well as a vector $(\hat{x}, \hat{y}, \hat{z})$ which will point towards the centroid of the object.

The previous proposed methods uses convolutions to learn features, but convolutional neural networks fails to address the spatial relationship between features because of max-pooling, which is also the important symmetric functions used by the state-of-the-arts methods[9, 10]. Also, the convolutional architectures also need huge datasets to generalize.

We proposed a model architecture of extension of Capsule network [12] for 3D object classification. We believe that capsules will help us preserve the orientation and spatial relationship of the extracted features.

1.3 Dataset

We used multiple dataset for training and evaluation of out methods. We used Kitti dataset for Pointer semantic and instance segmentation experiments and Modelnet40 for the Pointer capsnet experiment.

1. **KITTI Vision Benchmark Suite**

   Kitti[2] have been the most popular dataset for benchmarking in the autonomous vehicle research industry. The dataset was captured by driving around the mid-size city of Karlsruhe, in rural areas and on highways. The 3D object detection benchmark (fig: 1.5) consists of 7841 trained images and 7518 test images as well as their corresponding point clouds. For evaluation they use precision-recall curves and for ranking the methods they use average precision. The annotated objects in the dataset are cars, vans, trucks, pedestrians, person sitting, cyclists, and trams. The kitti dataset has 3d bounding box annotation in camera frame and not the point-wise labels. We tried to convert this bounding box from camera to LiDAR frame from the calibration matrix provided and label each point in the bounding box as the class label. The bounding box were not precise enough resulting in labelling of surrounding unknown points as class labels. A sample of labelled frame can be seen in fig. 1.5.

   For instance segmentation we created our in-house dataset using Kitti. After transforming the bounding box in LiDAR frame we calculated the centroid of

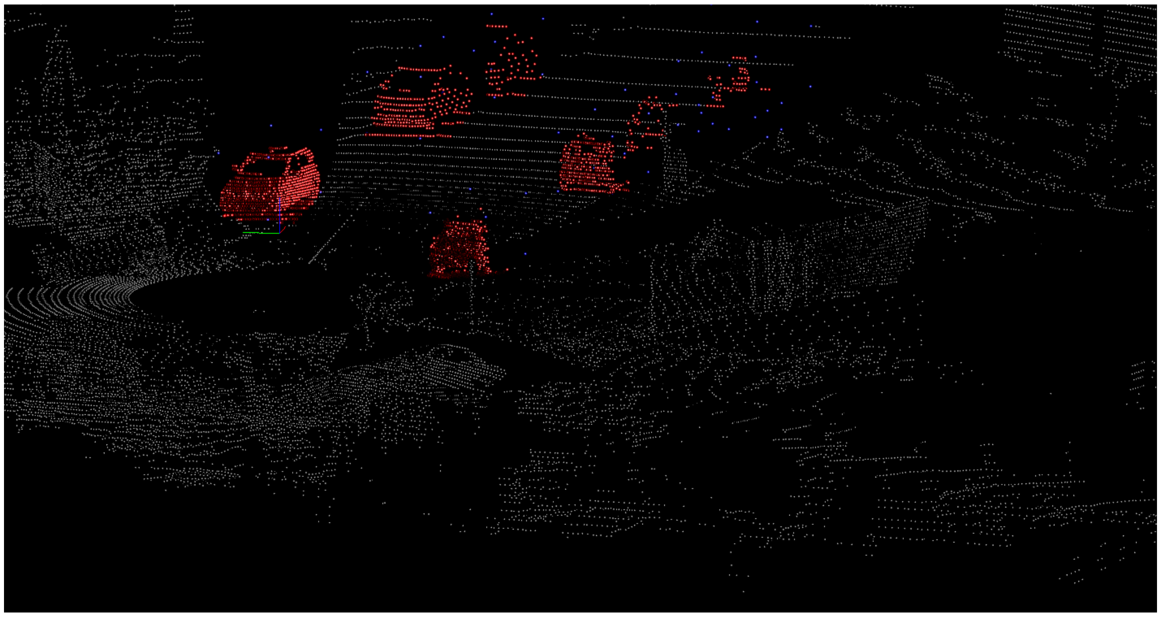


Figure 1.5: Point cloud semantic labelling in Kitti

each frame and labelled each point with a vector pointing from the point to its respective centroid. The unknown class point were not labelled. An example of instance labels can be seen in fig. 1.6 and 1.7.

2. **ModelNet40**

   ModelNet[13] provides comprehensive clean collection of 3D CAD models for objects (fig. 1.8). It is available in two subsets of 10 popular class subset (ModelNet10) and 40 class subset (ModelNet40). ModelNet had been popularly used to benchmark the models on point clouds and also analyze the cost variations after certain operations[12, 10, 12]. We used ModelNet40 to test Capsule Net operations before applying to the large point clouds.

1.4 Application

Lots of applications uses lasers to scan and analyze the environment or product entities. When they use laser they create a demand for 3D object detection algorithms for better analysis. The industry where 3D object detection are of critical importance

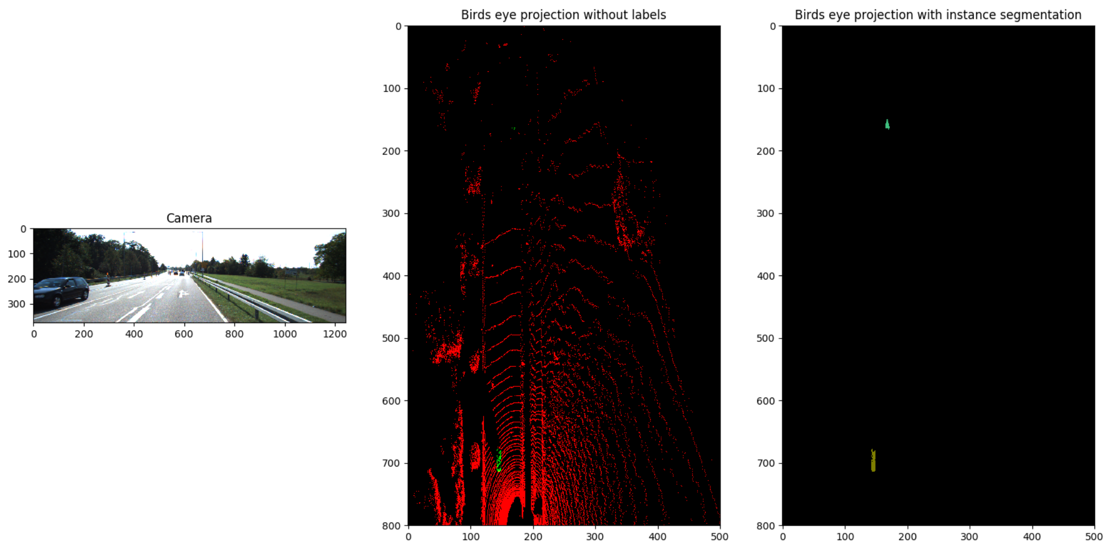


Figure 1.6: Instance Segmentation on Kitti dataset

are:

1. **Self Driving Industry**: 3D object detection is critical to the perception stack of a self driving car to detect and localize the obstacles in real world coordinates. They need to correctly identify vehicles and pedestrians in very possible scenario to avoid unexpected outcomes. With new LiDAR in the market, they enable the vehicle to see up to 200m, helping the vehicle to maintain a high speed and still have enough time to respond to unexpected states.

   The main advantage of LiDAR is the vehicle can drive at night with the same precision in the day irrespective of the lighting conditions. This makes the autonomous stack more stable and reliable while driving during night or sudden change in light intensity.

2. **Urban Planning and Surveys**: LiDAR scans are used more often to analyze urban areas, remote terrains, damage inspections etc. Drones are now widely used to inspect buildings, bridges, caves etc which are not safely reachable for humans. The 3D object detection algorithm can segment the buildings, bridges, vehicles creating a faster pipeline for analysis for researchers.

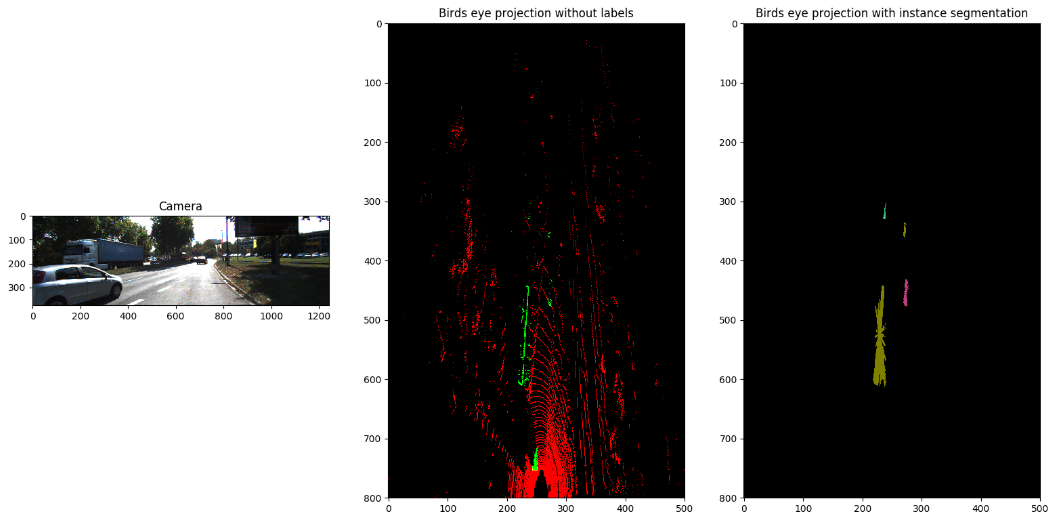


Figure 1.7: Instance Segmentation on Kitti dataset

3. **Health care**: 3D reconstructions are popularly used in diagnosis systems in digital microscopy and analysis of CT scans [6, 7].

4. **Manufacturing industry**: laser scanners are used to check fault in productions in real time. They are essential for quality assurance in industries producing for aerospace, defence, and medical sectors. 3D object detection algorithms can be used for localizing the object pose for the robot to make it more efficient for grabbing and placing in difficult situations.

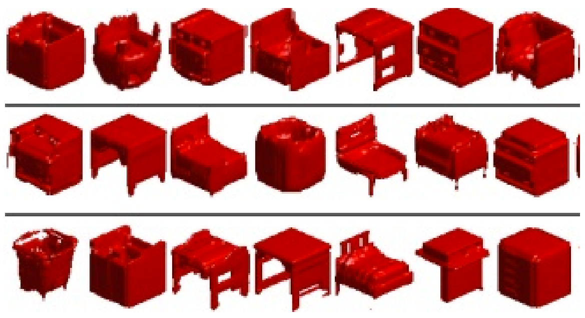


Figure 1.8: ModelNet Dataset Sample

Chapter 2

Literature Survey

2.1 3D Point wise Methods

Alternative to projection methods are the pointwise methods, where the point is represented by just its 3 coordinates (x,y,z) and additional dimensions can be added such as normal, intensity etc, depending on the availability.

Pointnet [9] was the initial approach for a novel type of neural network that directly consumes unordered point clouds, which also takes care of the permutation invariance of points in the point cloud. Pointnet can do object classification, part segmentation, and scene semantic parsing. The main feature of Pointnet is its robustness with respect to input perturbations and corruptions. Pointnet has 3 main key modules (Fig 2.1), which were use of symmetric function, a local and global information combination structure, and two joint alignment networks that aligns both input points and point features. The most important key was the use of the single symmetric function, MaxPooling, which inputs a single vector which is invariant to the permutations of the input vector orders.

Pointnet failed to capture local structure and generalize to complex scenes, so the authors modified the architecture resulting in Pointnet++ [10]. The intuition of Pointnet++ came from the basic CNN structure where its lower level neurons have smaller receptive fields whereas larger level have larger receptive fields. The ability to abstract local patterns along the hierarchy allows better generality to unseen cases.

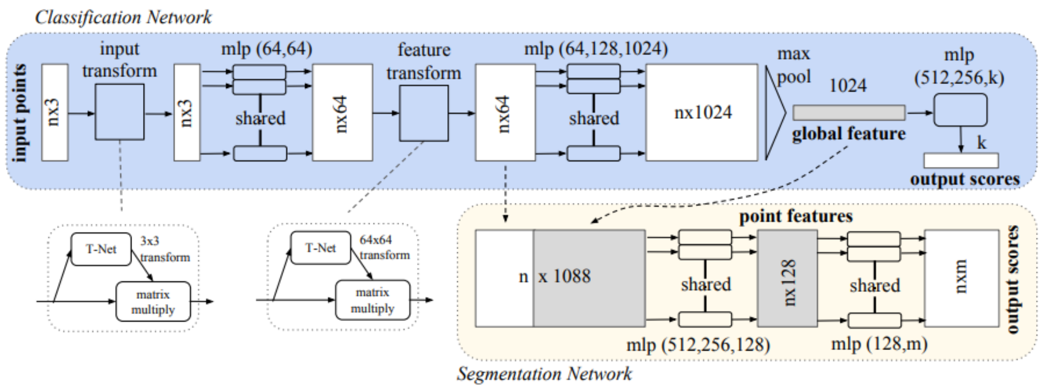


Figure 2.1: Pointnet architecture

Pointnet++ is a hierarchical network that applies Pointnet recursively on a nested portioning of the input point cloud. It proposes novel set learning layers to adaptively combine features from multiple scales from varying densities. Similar to CNNs, Pointnet++ extracts local features from small neighborhoods and groups them into larger units and processes the groups to produce higher level features. This process is recursive until we obtain the feature of the whole point set.

The two main issues addressed by Pointnet++ were the partitioning of point set, and abstraction of points or local features through a local feature learner. Both these issues are correlated as the partitioning of the point set has to produce common structure partitions, so that the weights of the local features can be shared. Pointnet++ uses Pointnet as the local feature learner.

The hierarchical structure (Fig : 2.2) is composed by a number of set abstraction levels. The set abstraction layers consist of three layers: Sampling layer, Grouping layer and Pointnet layer.

Recently, there was one more architecture inspired from Pointnet known as EdgeConv[12]. EdgeConv, instead of working on individual points, exploits the geometric structure by constructing a local neighborhood graph and applying convolution-like operation on the edge connecting the neighborhood pair of points. It thus has the property of translation-invariance and non-locality.

Here the graph is not fixed as it is dynamically updated after each layer of the

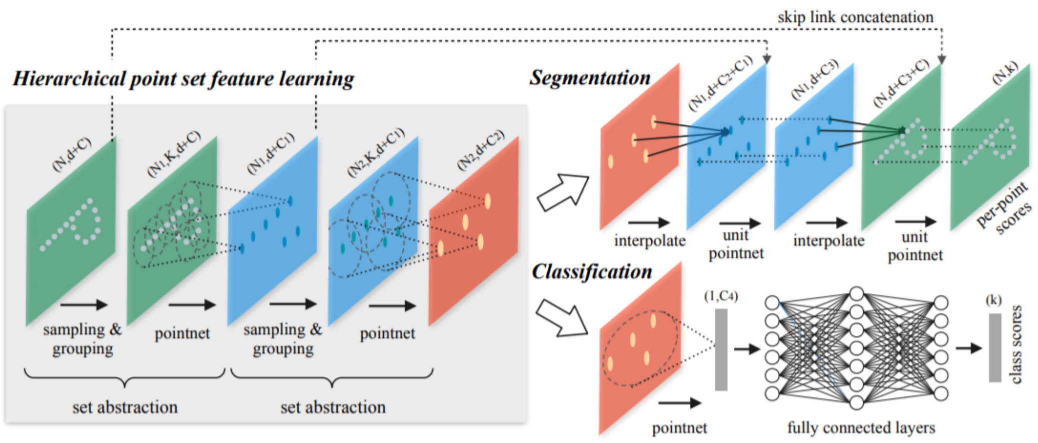


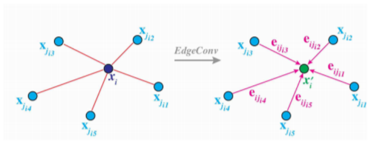
Figure 2.2: Pointnet++ Architecture

network, that is the k-nearest neighbors (kNN) of a point changes after each layer is calculated from the sequence of embedding.

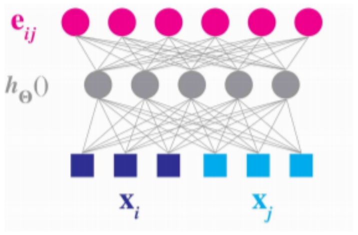
EdgeConv (fig :2.3) applies channel-wise symmetric aggregation operation (ψ) on the edge features associated with all the edges emanating from each vertex. The edge features are defined as $E_{ij} = H(x_i, x_j)$, where H is some parametric non-linear function parametrized by the set of learnable parameters. Here, H will be MLP for the model. So the output of EdgeConv at the i^{th} vertex is $x_i^{'} = \underset{j:(i,j)\in\varepsilon}{\psi} H_\theta(x_i, x_j)$. Where ψ is channel-wise symmetric aggregation operation on the edge features associated with all the edges emanating from each vertex.

The choice of edge function has a crucial influence on the properties of the resulting EdgeConv operation (table 2.1)

Besides changing the input, there were approaches to change the convolution operators for point cloud too. Parametric Continous Convolution[15] is a learnable operator that operates over non-grid structured data and exploits parametrized kernel functions that span the full continuous vector space. It can handle arbitrary data structures as long as its support relationship is computable. The continuous convolution operator is based on Monte-Carlo integration, so given particular function f and g with a finite number of input points y_i sampled from the domain so the convolution at any arbitrary point x will be approximated as $h(x) = \int_{-\infty}^{\infty} f(y)g(x-y)dy \approx$



(a) EdgeConv Operation: The output of EdgeConv is calculated by aggregating the edge features associated with edges from each connecting vertex.



(b) Computing edge feature E_{ij} from a point pair x_i, x_j

Figure 2.3: Edge Conv Operations

Edge Function	Property
$H(x_i, x_j) = \theta_j x_j$ $\theta = (\theta_i, ..., \theta_k)$	Classical Euclidean Convolution
$H(x_i, x_j) = H(x_i)$	Encoding global information of the local neighboring structure. (Pointnet)
$H(x_i, x_j) = H(x_j - x_i)$	Encodes the local information, considering the shape as collection of small patches and losing the global information.
$H(x_i, x_j) = H(x_i, x_j - x_i)$	Asymmetric edge function which combines the global shape structure(Xi) and local shape features $(x_j - x_i)$

Table 2.1: Edge Functions

$\sum_i^N \frac{1}{N} f(y_i) g(x - y_i)$

Function g is parameterized such that each point in the support domain is assigned a value (Kernel weights), such parameterization is infeasible for continuous convolution, since the function g will be defined over an infinite number of points. The solution is to model g using parametric continuous functions using multi-layer perceptrons (MLP) as the approximator, because MLPs are expressive and capable of approximating continuous function, so $g(z; \theta) = MLP(z; \theta)$. The kernel function g(z,) spans the full continuous support domain while remaining parameterized by a finite number of parameters.

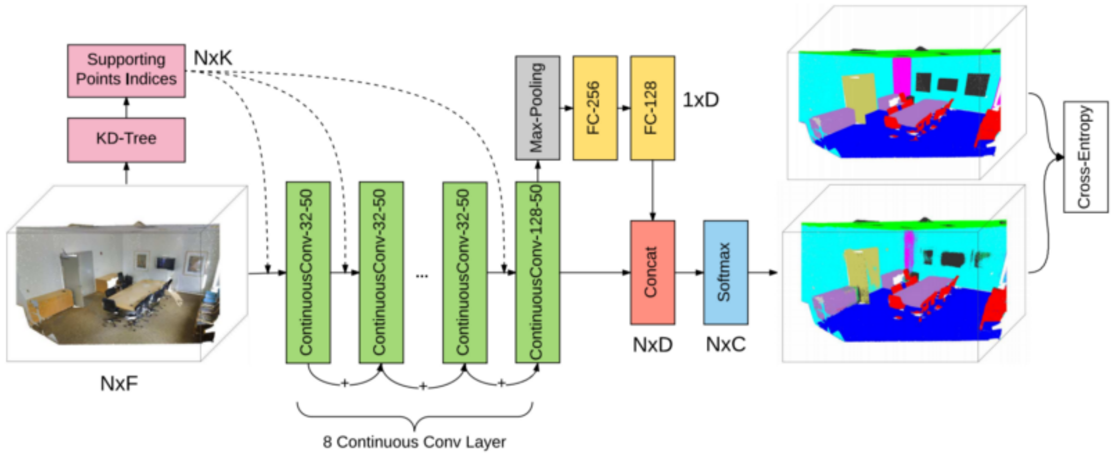


Figure 2.4: Continuous Parametric Convolution Architecture

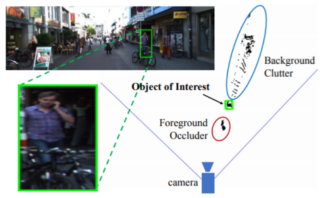
2.2 3D Instance Segmentation

Instance segmentation is also an important problem for autonomous vehicle, where we are not only segmenting the points but also counting the number of objects per class and classifying the point belong to them. There had been a very few approaches in this direction but mostly with the use of RGB image along with LiDAR. Frustum Pointnet [8] is a novel framework for RGB-D data based object detection. Instead of solely relying on 3D proposals, this method leverages both mature 2D object detection and advanced 3D deep learning for object localization. It leverages mature 2D object detector to propose 2D object regions in RGB images as well as to classify objects. So, with a known camera projection matrix, a 2D box can be lifted to a frustum that defines a 3D search space. We collect all the points in frustum to form a frustum point cloud.

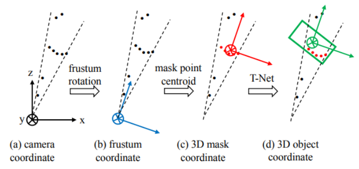
The frustum may be oriented towards any direction leading to a large variation in the placement of point clouds. So we normalize the frustums by rotating them towards center view such that the center axis of the frustum is orthogonal to the image plane. Similar to Mask-RCNN, which achieves instance segmentation by binary classification of pixels in the image region, it does 3D instance segmentation using a Pointnet-based network on point clouds in frustums. It predicts the 3D bounding box center in the local coordinate system: 3D mask coordinates.

The network takes frustum point cloud as input and predicts a score for each point for how likely the point belongs to the object of interest. In multi-class detection case, it also leverages the semantics from a 2D detector for better instance segmentation. So the network can use this prior to finding geometries that look like the object of interest. In the architecture, they encode the semantic category as a one-hot vector (k dimensional for the pre-defined k categories) and concatenate the one-hot vector to the intermediate point cloud features.

After 3D instance segmentation, points which are classified as the object of interest are extracted. They further normalize the points to boost the translational invariance of the algorithm and translate the point clouds into a local coordinate by subtracting XYZ values by its centroid. The coordinate transformation and frustum rotation are critical for 3D detection results.



(a) Frustum proposal generation



(b) Coordinate System for point cloud

Figure 2.5: Frustum Pointnet

Given the segmented object points, this module will estimate the object amodal oriented 3D bounding box by running a box regression Pointnet together with a preprocessing transformer network. A light-weighted regression Pointnet (T-net) is used to estimate the true center of the complete object and transform the coordinate such that the predicted center becomes the origin. The T-net can be thought of as a special type of spatial transformer network (STN). The box estimation network predicts amodal bounding boxes for objects given an object detection in 3D object coordinate. They parametrized the 3D bounding box by its center (c_x, c_y, c_z), size (h, w, l) and heading angle θ.

The center residual predicted by the box estimation network is combined with previous center residual from the T-net and the masked points centroid to recover an absolute center.

2.3 Capsule Network

Convolutional neural networks have been a dominant approach to object detection but still have many drawbacks. A notable example is CNN faces difficulty in generalizing to novel viewpoints. Capsules[3] solves some inefficiences by converting pixel intensities into vectors of instantiation parameters of recognized fragments and then applying transformation matrices to the fragments to predict the instantiation parameters of larger fragments. Here each layer is divided into many small groups of neurons called capsules. A capsule [12] is a group of neurons which performs computation on the input and encapsulates it into a vector which is capable of representing a different property of the same entity.

In the capsules, the length of the activity vector represents the probability that the entity exists and orientation represents the instantiation parameters. Capsule network is equivariant to the input instead of being invariant. So if the object is rotated at an angle, the capsule will be able to identify that the object is rotated at an angle without extra augmented training required. CapsNet preserves the geometric dependence of features to be translational invariant.

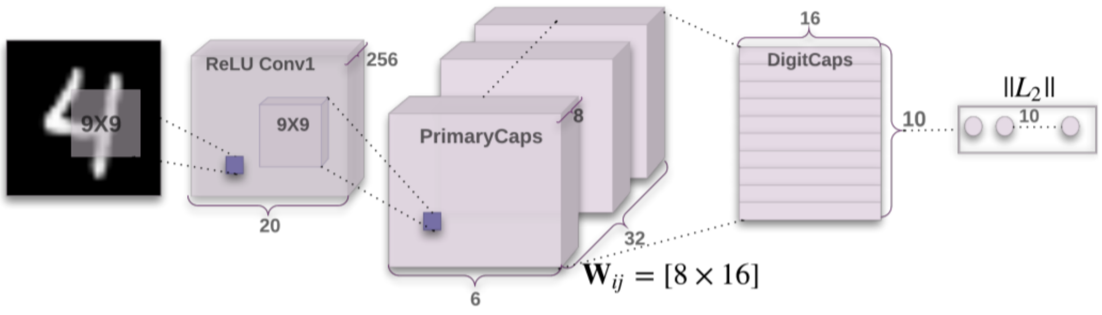


Figure 2.6: CapsNet architecture with 3 layers for MNIST data

2.3.1 Routing

The output of a capsule is a vector making it possible to use powerful dynamic routing to ensure all the output of the capsules are sent to an appropriate parent. Each parent capsule computes a prediction vector by multiplying its output by a weight matrix. If the prediction vector has a large scalar product with the output of a possible parent, there is a top-down feedback which increases the coupling coefficient for that parent and decreases for other parents[12].

Where c_{ij} is the coupling coefficient and is determined by the iterative dynamic routing process. The coupling coefficient between capsule i and all the capsules in the above layers sums up to 1 and is determined by a routing softmax whose initial logits b_{ij} are the log probabilities that capsule i should be coupled to capsule j.

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})} \tag{2.1}$$

The log priors can be learned discriminatively at the same time as all other weights. The agreement is measured as the scalar product $a_{ij} = v_j \hat{u}_{j|i}$. The agreement is treated as a log likelihood and is added to the initial logits b_{ij} before computing the new values of the coupling coefficient.

For all the first layers of capsules the total input s_j is a weighted sum over all prediction vectors $\hat{u}_{j|i}$ from capsules in the layer below and is produced by multiplying the output u_i of a capsule in the layer below with a weight matrix W_{ij}

$$s_j = \sum_i c_{ij} \hat{u}_{j|i}, \qquad \hat{u}_{j|i} = W_{ij} u_i \tag{2.2}$$

2.3.2 Squash

The length of the output vector of a capsule represents the probability that the entity represented by the capsule is present in the current input. A non-linear squashing function is used to ensure that short vectors get shrunk to almost zero length and long vectors get shrunk to a length slightly below 1. Squash is the activation function applied to the capsule. For a capsule total input s_j the output v_j is defined as

$$v_j = \frac{\parallel s_j \parallel^2}{1 + \parallel s_j \parallel^2} \frac{s_j}{\parallel s_j \parallel^2} \tag{2.3}$$

2.3.3 Results

CapsNet achieved state-of-the-art results on MNIST dataset and also proved to learn more robust representation for each class than a traditional CNN. It also achieved more accuracy on affine transformed dataset as compared to the traditional CNN. Although it achieved a 10.6% error rate on CIFAR10[4] dataset when tested with an ensemble of 7 similar models, it is similiar to the error rate when standard CNN were applied to CIFAR10[14].

Chapter 3

Methodology

Considering scenarios where lighting conditions are not appropriate for camera model to work robustly, we need a pipeline that should be invariant to the environment conditions to work appropriately.

Implementing a pipeline that works on LiDAR data alone will provide us the same result irrespective of the lighting conditions. The pipeline will be more reliable for driving at night as the scans will be not affected by the day/night conditions. Most of the autonomous vehicles pipeline also deploys two independent models for perception using camera and LiDAR and use their output to cross-verify the detections.

We believe a independant LiDAR perception stack will be more robust to the challenges faced by autonomous vehicles given its scan invariance to the environment lighting condition. If we summarize the section 1.1, most of the methods just evaluate their model on a point cloud with less than ten thousand points and also, there is no method that did 3D semantic segmentation for a large point cloud with more than 60 thousand points per frame. Given the LiDAR point cloud will have at least consist of more than 60 thousand points per frame, we need a more scalable algorithm to deal with this scenario.

3.1 Pointer Features

The points in the point cloud can be represented by two properties which is its position in the frame (global feature) and the distribution of its neighboring points (local features). We needed to develop an architecture that can embed both local and global features (fig. 3.1) of the point cloud and would also learn to weight the importance of these features to generate strong high level features that would make the learning faster and easier.

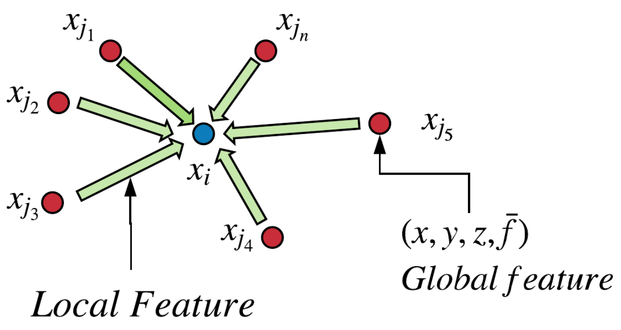


Figure 3.1: Point cloud features

Global Features

- Consist of real 3D world coordinates x, y, z and feature provided by the sensor like intensity, phase of wave, rgb value etc.
- Is a feature of a single point.

Local Features

- Consist of unit vector pointing from its neighbours to the point. $x_i - x_j$, where x_j is the neighbors of the point x_i.
- Is a feature of a single point depending on its neighbors.

Consider a F-dimensional point cloud with n points, denoted by $X = (x_1, x_2..., x_n) \in \mathbb{R}^F$. In the simplest setting of F = 3, each point contains 3D coordinates $x_i =$

(x_i, y_i, z_i); it is possible to include additional coordinates representing intensity, color, surface normal, and so on bu they are sensor dependent. Generally the dimension F represents the feature dimensionality of a given layer.

x_i is a point in the point clouds and x_j is the neighboring point in the point cloud. we can regard x_i as the central pixel and $x_j : (i, j) \in \varepsilon$ as a patch around it.
We define global features p_{ij} with function g_θ which is a parametric non-linear function parametrized by the set of learnable parameters θ

$$p_{ij} = g_\theta(x_i, x_j)$$
$$g_\theta : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}^{F'}$$

We define local features q_{ij} with function h_θ which is also a parametric non-linear function parametrized by the set of learnable parameters θ

$$q_{ij} = h_\theta(x_i, x_i - x_j)$$
$$q_\theta : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}^{F'}$$

We define the fusion feature T_{ij} of local features q_{ij} and global feature p_{ij} with function M

$$T_{ij} = M(p_{ij}, q_{ij})$$

Here M is a learnable function which can be weighted sum or a convolutional layer or a concatenation layer

Finally, we define the Pointer operation by applying a channel-wise symmetric aggregation operation $\square$ ($\sum$ or max)

$$\boxed{x_i^{l+1} = \underset{j:(i,j)}{\square} T_{ij}^l}$$

3.2 Pointer Block

As per section 3.1 and fig. 3.2, we we used KD-tree to find the find the neighbors for each point in the point cloud. After finding the nearest neighbors we will get feature tensor of shape $N \times k \times f$ which will be passed to two different networks which will calculate the global and local features of the point cloud. We do weighted sum of high level features generated by these two networks and then perform max pooling to obtain feature of shape $N \times O$.

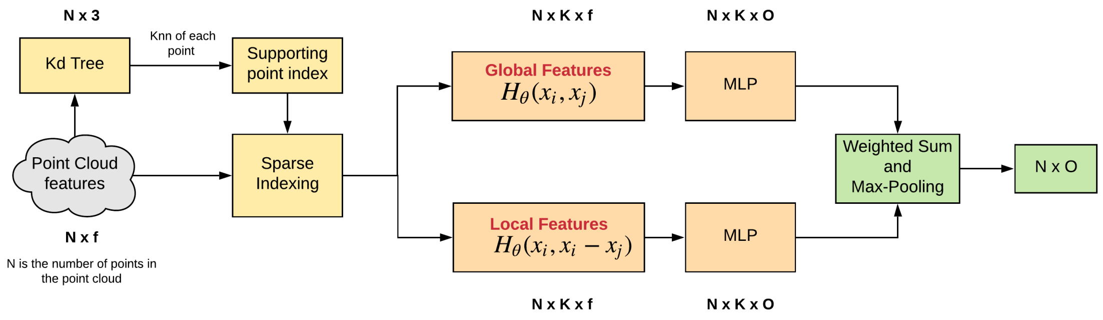


Figure 3.2: Pointer main block

3.3 Pointer Semantic Segmentation

The main challenge is to do 3D point wise semantic segmentation using LiDAR data alone. Pointer generated features can be used to segment vehicles and pedestrians from the LiDAR point cloud. We do so by concatenating pointer blocks and using a fully connected layer to output class score for each of the point in the point cloud. For the best model we used 4 pointer concatenated block with 3 layer of convolutional layers to predict the class score. We used skip connection throughout the architecture and were critical to convergence. A frame mostly consists of unknown labelled points consisting of roads, bushes, tree, etc. which we do not care about. A small portion of the frame consisted of the target class labels like vehicles and truck and the samples were even more less for pedestrians. We used weighted cross-entropy as loss due to deal with the imbalance in class labels. The weights for this class label were decided pragmatically upon the target class accuracy performance.

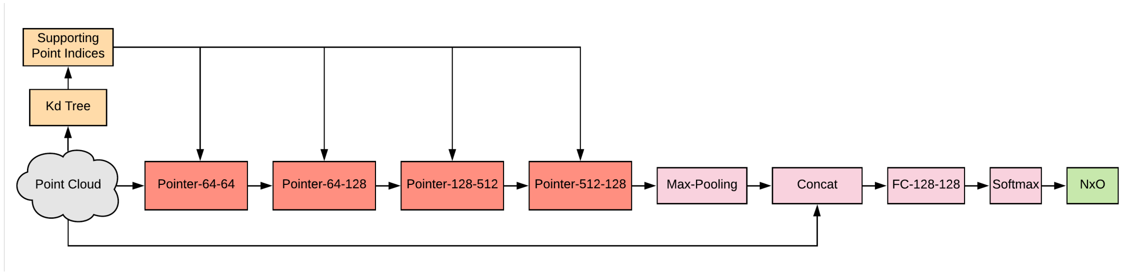


Figure 3.3: Pointer vector Instance segmentation

We used two metrics for evaluating the performance. The total accuracy which is the total correct prediction ratio for the whole point cloud. The target class accuracy is only for the target class which consist of vehicles, pedestrians, etc. We compared out performance with Pointnet++ and Edgeconv on the kitti dataset. Pointer's overall accuracy was lower than the two network but the target class accuracy was significantly better as shown in table 3.1 and fig. 3.4, 3.5, and 3.6.

Model	Accuracy	Target Class Accuracy
Pointnet++	97.12	45.34
EdgeConv	95.20	75.20
Pointer	94.88	**83.40**

Table 3.1: Semantic Segmentation Accuracy on Kitti Test Dataset

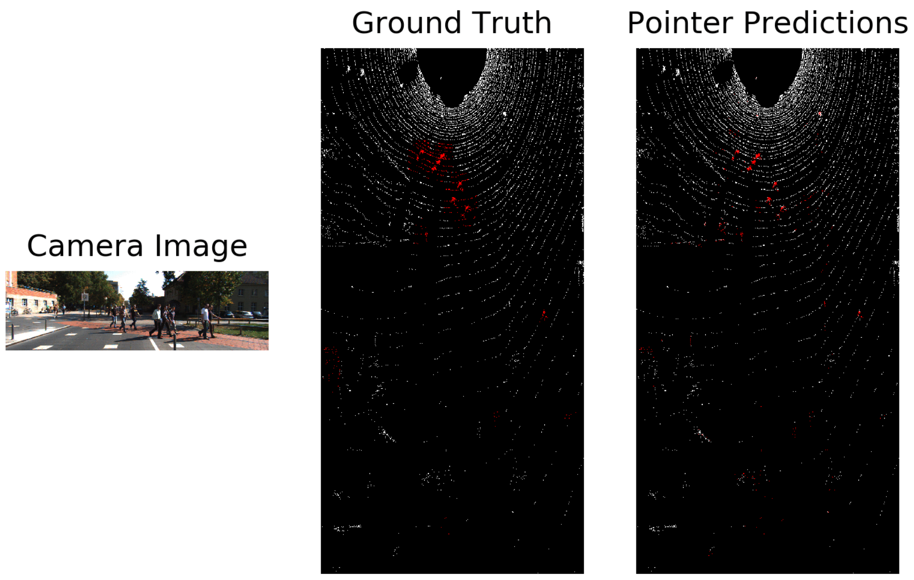


Figure 3.4: Pointer semantic segmentation result

3.4 Pointer Instance Segmentation

Along with segmentation, detecting different instance is also important for tracking and prediction pipeline and instance segmentation using only LiDAR was not a well explored till yet. We carried two major experiments to do instance segmentation using pointer using LiDAR data alone.

3.4.1 Clustering Instance segmentation

We tried to create a cluster for each instance in the frame and create a batch with this cluster to pass it to the network. The network architecture (fig. 3.7) was the same for modification was only to apply clustering to form cluster for each instance

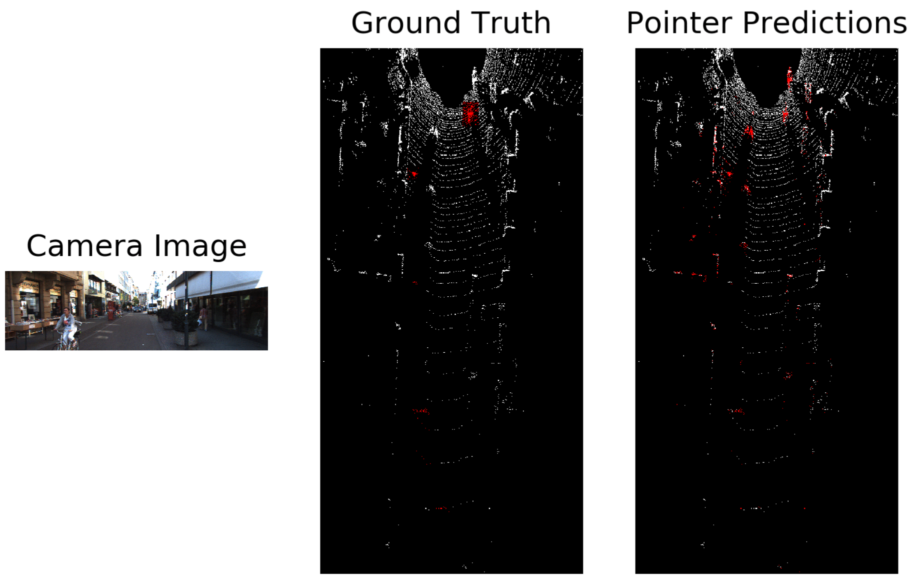


Figure 3.5: Pointer semantic segmentation result

and then pass it to the network. We choose Bayesian Gaussian mixture clustering and choose batch size as 20 from the statistics of the dataset. The clustering did not perform well as it would give a cluster with more than two instance in or a clusters with the vehicles parts divided in them. The accuracy of the semantic segmentation for this cluster was also poor (table 3.2, fig. 3.9, 3.10) given the poor clusters. We would try to improve the clustering to get better results for this approach.

3.4.2 Vector Instance Segmentation

For the second approach, we believed a centroid of a instance is a distinctive character for each instance in the point cloud. We tried to predict the segment id as well as a vector $(\hat{x}, \hat{y}, \hat{z})$ which points towards the respective centroid of the object. It helps in

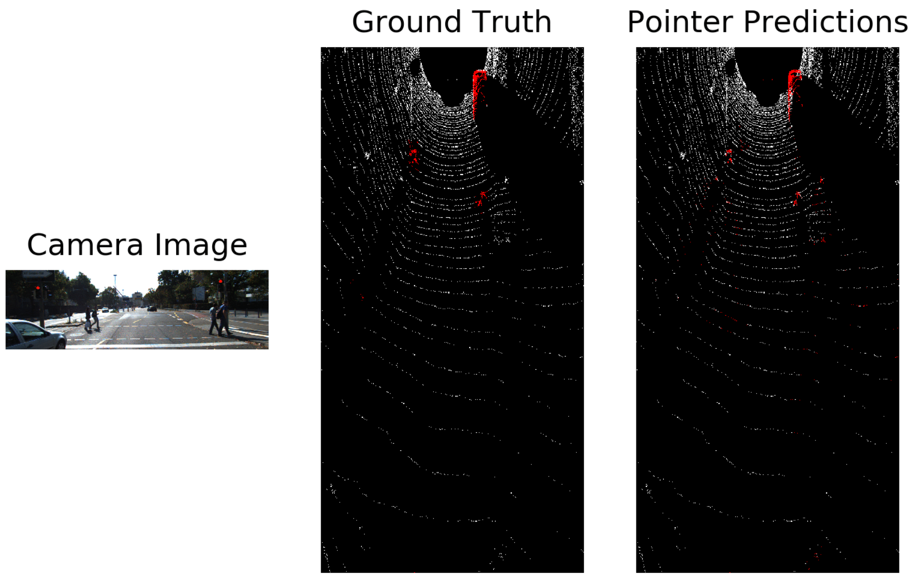


Figure 3.6: Pointer semantic segmentation result

differentiating two instances which are close to each other as there centroids will at a distance greater than the points which are close to each other. We created in-house training and testing dataset to include the vector in ground truth in the Kitti dataset.

The network architecture (fig 3.8) was the same from the semantic segmentation experiment, we split the the network after generating high level features into two networks, where one network was to predict the class label for each label as in semantic segmentation and the other was to predict the vector to the centroid for each point. We used cosine similarity loss for the vector prediction and weighted cross entropy loss for the semantic prediction. We used sum of these two losses to train the network. The accuracy of the prediction (table 3.2, fig. 3.11, 3.12, 3.13) was similar to the semantic segmentation accuracy and we got better instance segmentation till a a safe

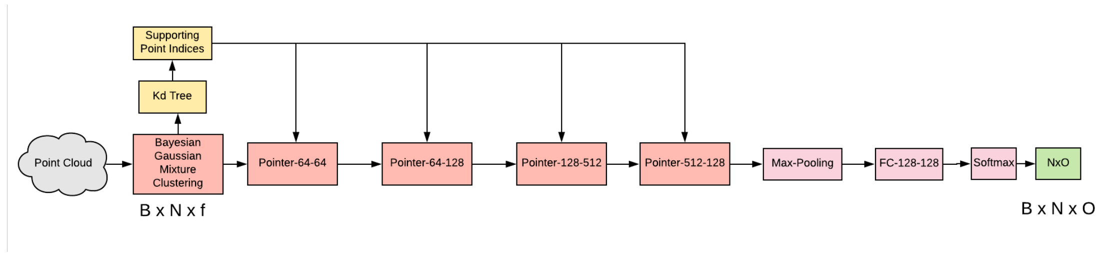


Figure 3.7: Pointer clustering instance segmentation

distance than the previous experiment.

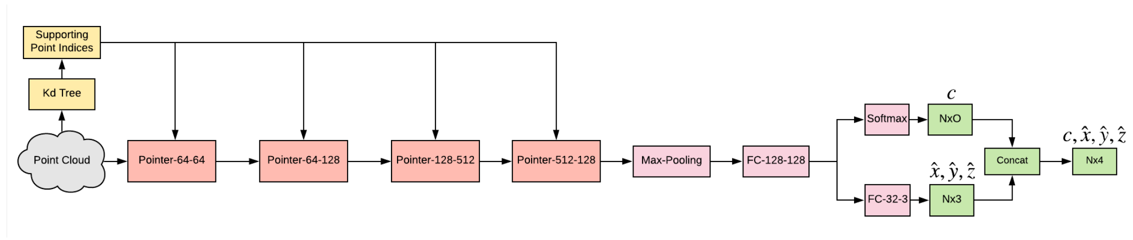


Figure 3.8: Pointer clustering instance segmentation

Model	Accuracy	Target Class Accuracy
Pointer Clustering	91.59	40.62
Pointer Vector	93.38	82.91

Table 3.2: Instance Segmentation Accuracy

3.5 Pointer Capsnet

Internal data representation of a convolutional neural network does not take into account important spatial hierarchies between simple and complex objects. Geoffrey Hinton argued that in order to correctly do classification and object recognition, it is important to preserve hierarchical pose relationships between object parts. Capsules encode probability of detection of a feature as the length of their output vector and the state of the detected feature is encoded as the direction in which that vector points to. When detected feature moves around the image or its state somehow changes, the probability still stays the same (length of vector does not change), but its orientation changes. The original capsule relies on the existence of a spatial relationship between elements in the feature map. Whereas such features are lost in point permutation invariant formulation of 3D pointwise classification methods. We tried to extend capsule network for 3D point clouds by given the capsules the features extracted by pointer network.

We propose to use pointer features of the point cloud as the input feature to the CapsNet and carryout classification experiments with ModelNet40 dataset. As CapsNet preserves the spatial relationship and orientation of the extracted features, we believe it will help the network learn faster with less samples and also to generalize to multiple viewpoints which is a dominant problem for 3D point cloud. The architecture (fig. 3.14) is similar to Capsnet but the input features are just obtained using pointer instead of a convolutional layer. We tried to experiment with rotationally augmented samples of Modelnet40 and tried to compare the accuracy with Pointnet[9], Pointnet++ and Edgeconv (table 3.3). The network was hard to train (fig. 3.15) and took more time to converge and we were unable to beat the accuracy of baselines.

Model	Accuracy
Pointnet	89.2
Pointnet++	90.7
EdgeConv	92.2
3D Capsule (with Edgeconv)	92.7
Pointer Capsnet	**71.29**

Table 3.3: Classification Accuracy on Modelnet40

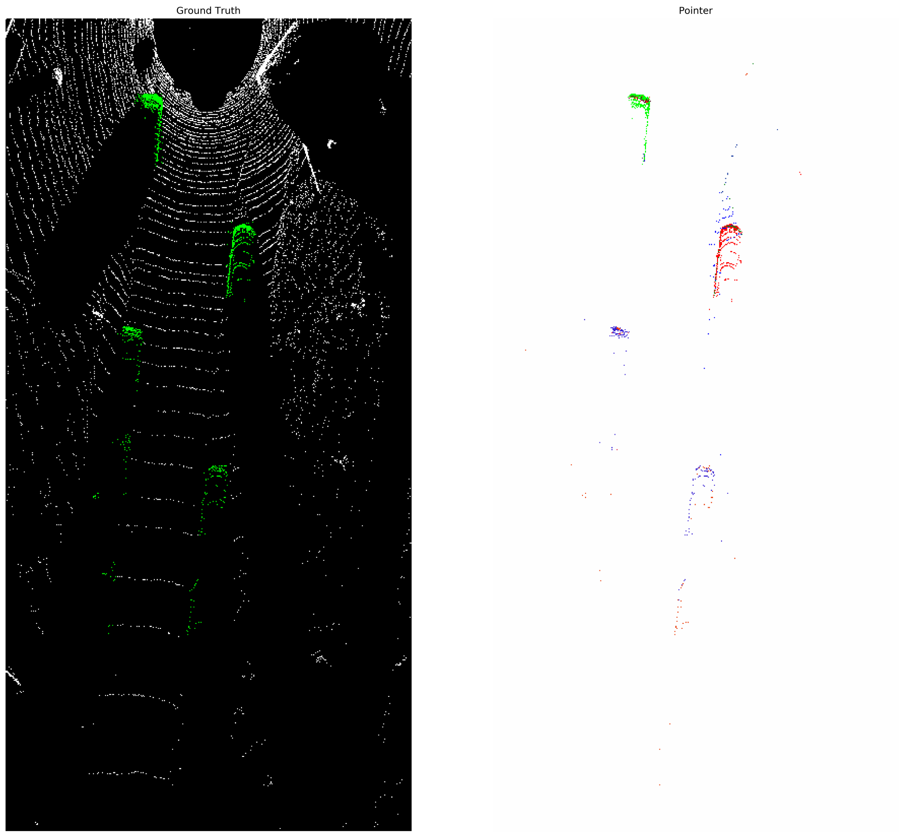


Figure 3.9: Pointer clustering instance segmentation result

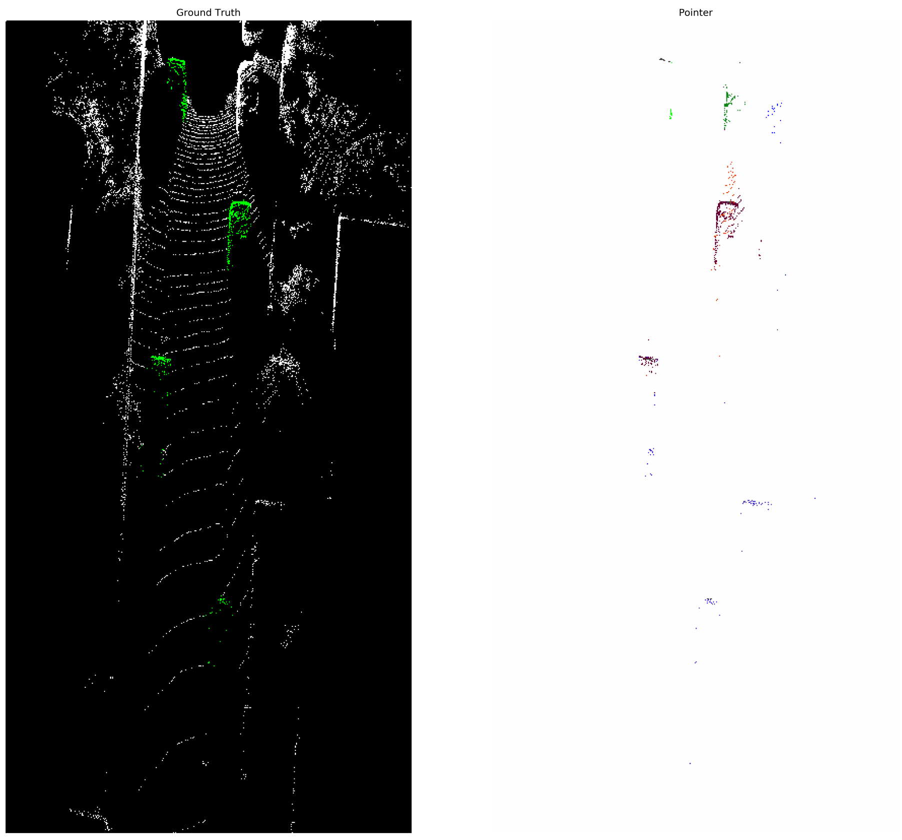


Figure 3.10: Pointer clustering instance segmentation result

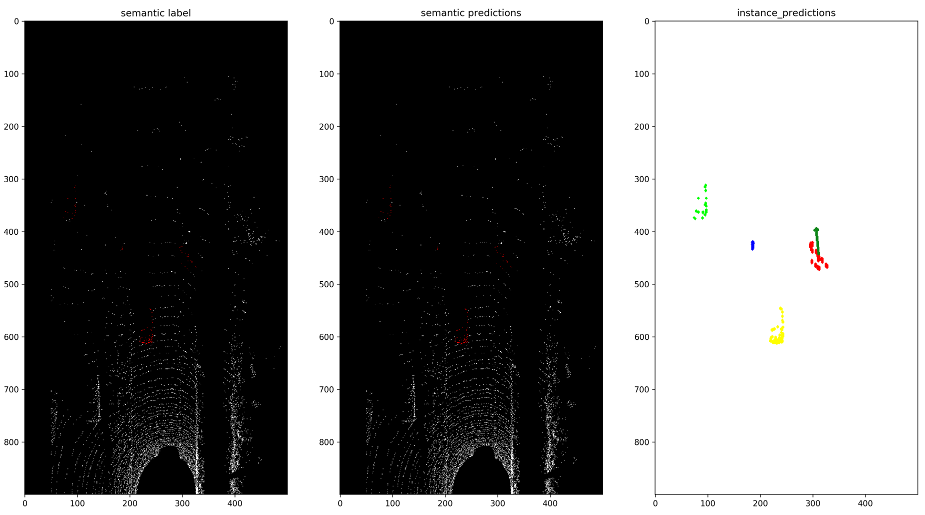


Figure 3.11: Pointer vector instance segmentation result

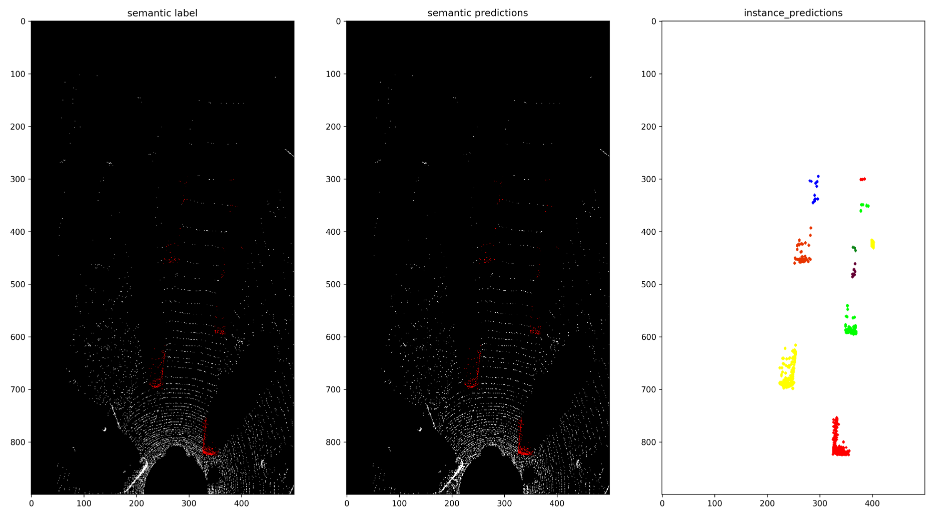


Figure 3.12: Pointer vector instance segmentation result

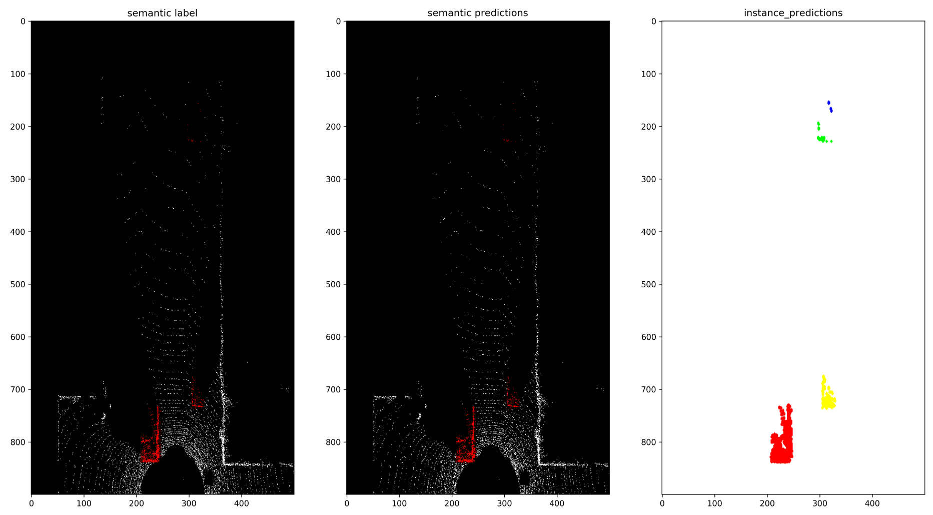


Figure 3.13: Pointer vector instance segmentation result

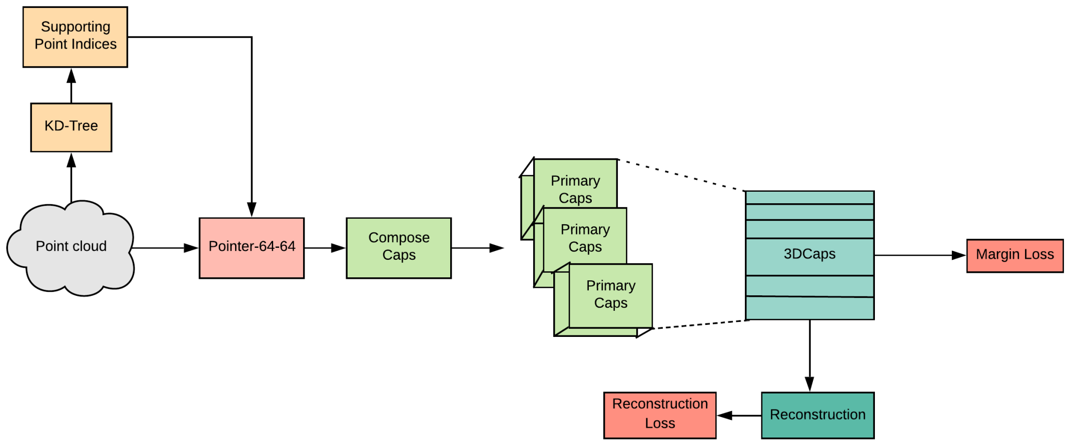


Figure 3.14: Pointer Capsnet Architecture

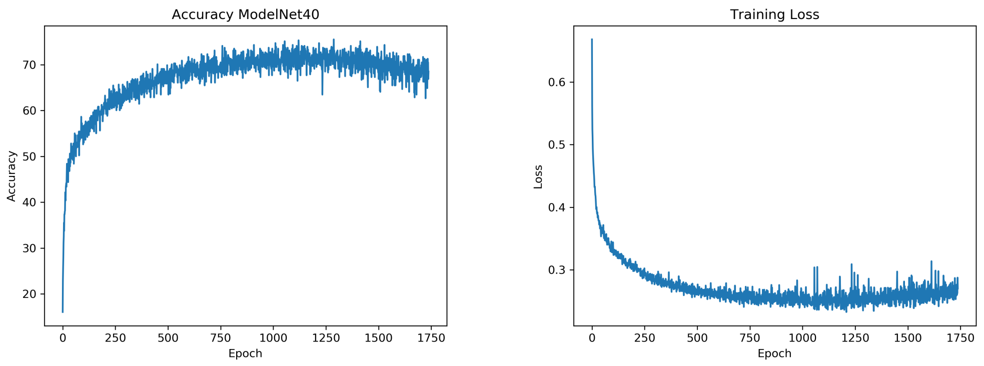


Figure 3.15: Pointer Capsnet training loss and validation accuracy

Chapter 4

Conclusion and Future work

Pointer can contribute to the development of self-driving vehicle perception stack to make roads more safer for pedestrains. Pointer is a more robust and camera independent pipeline for segmenting vehicles and pedestrian for an autonomous vehicle perception stack. Pointer is invariant to lighting conditions giving the same performance and accuracy during day/night scenarios. Pointer is one of the initial approach to do end-to-end instance segmentation using LiDAR data alone making it beneficial for tracking and planning systems.

Pointer still needs improvement in efficient sampling methods to reduce the number of input points to the network cutting down the computational need. Reducing model size and inference time is required to make it more optimal for real time inference. More experiments with different clustering algorithm need to done in order to improve clustering to improve both vector and clustering instance segmentation. The Pointer features to CapsNet were not accurate enough to provide good accuracy for Modelnet dataset and needs further detailed experiments and analysis for further improvement. LiDAR only perception algorithms are essential to drive at night and Pointer would help making the self driving cars more safe.

Bibliography

[1] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. *CoRR*, abs/1611.07759, 2016.

[2] Andreas Geiger. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3354–3361, Washington, DC, USA, 2012. IEEE Computer Society.

[3] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. In *ICANN*, 2011.

[4] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[5] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[6] Peter M Maloca, J. Emanuel Ramos de Carvalho, Tjebo Heeren, Pascal W Hasler, Faisal Mushtaq, Mark Mon-Williams, Hendrik P.N. Scholl, Konstantinos Balaskas, Catherine Egan, Adnan Tufail, Lilian Witthauer, and Philippe C. Cattin. High-performance virtual reality volume rendering of original optical coherence tomography point-cloud data enhanced with real-time ray casting. *Translational Vision Science Technology*, 7(4):2, 2018.

[7] Jeremy P Metcalf and Richard C Olsen. Photogrammetric point cloud and lidar fusion for improved building delineation (conference presentation). In *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XXIV*, volume 10644, page 106440E. International Society for Optics and Photonics, 2018.

[8] Charles Ruizhongtai Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. *CoRR*, abs/1711.08488, 2017.

[9] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[10] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017.

[11] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015.

[12] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018.

[13] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920. IEEE Computer Society, 2015.

[14] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[15] Chris Zhang, Wenjie Luo, and Raquel Urtasun. Efficient convolutions for real-time semantic segmentation of 3d point clouds.

Chapter 5

Appendix

5.1 Visual Results

This section contains some birds eye view projection result from Kitti dataset of the architectures used for benchmarking.

5.1.1 Pointnet

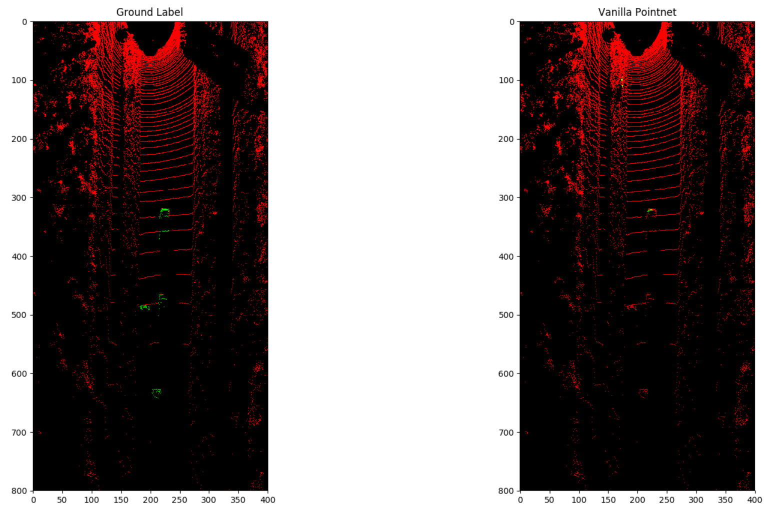


Figure 5.1: Pointnet results

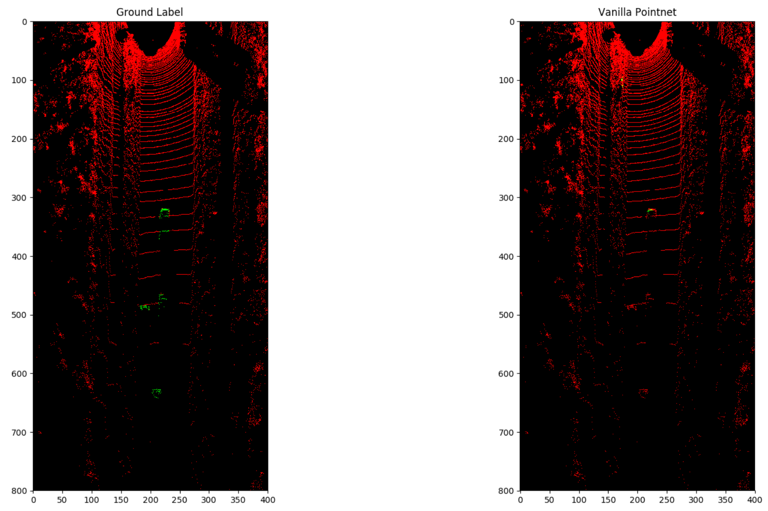
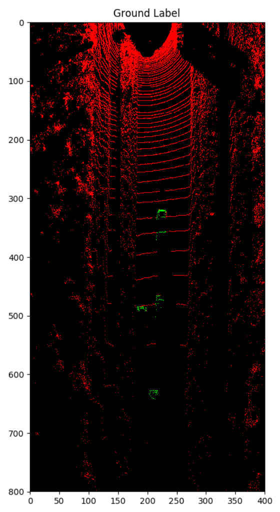


Figure 5.2: Pointnet results



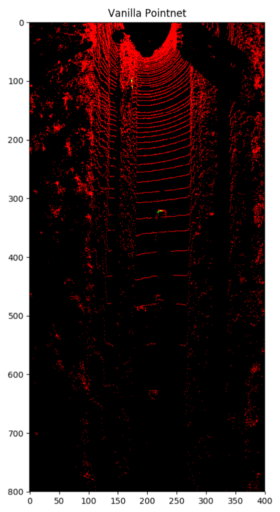


Figure 5.3: Pointnet results

5.1.2 Pointnet++

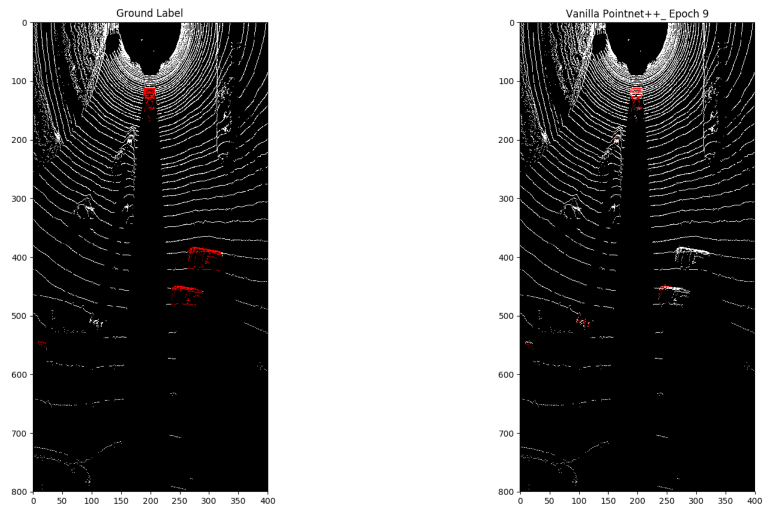


Figure 5.4: Pointnet++ results

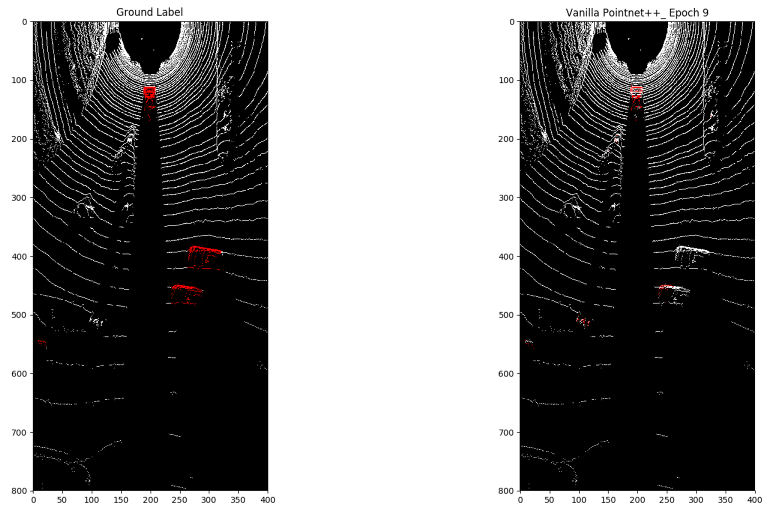


Figure 5.5: Pointnet++ results

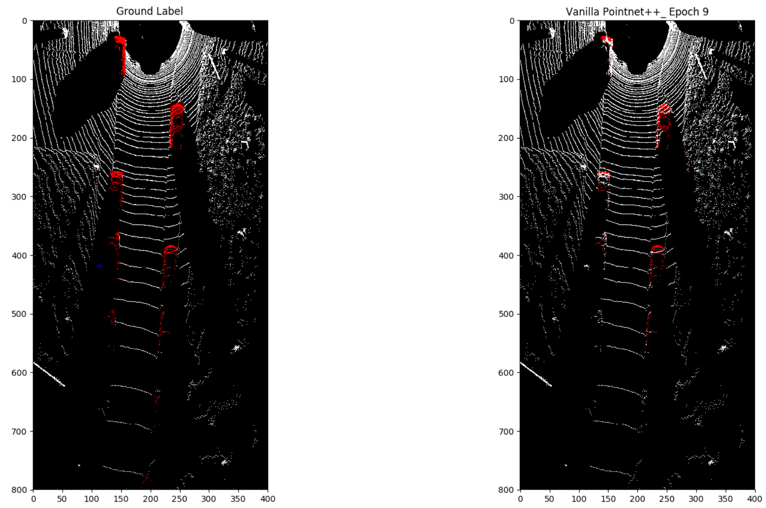


Figure 5.6: Pointnet++ results

5.1.3 EdgeConv

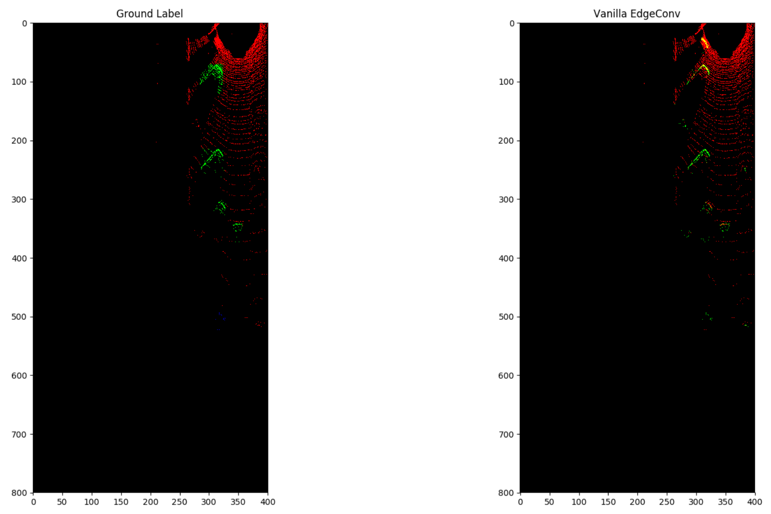


Figure 5.7: EdgeConv results

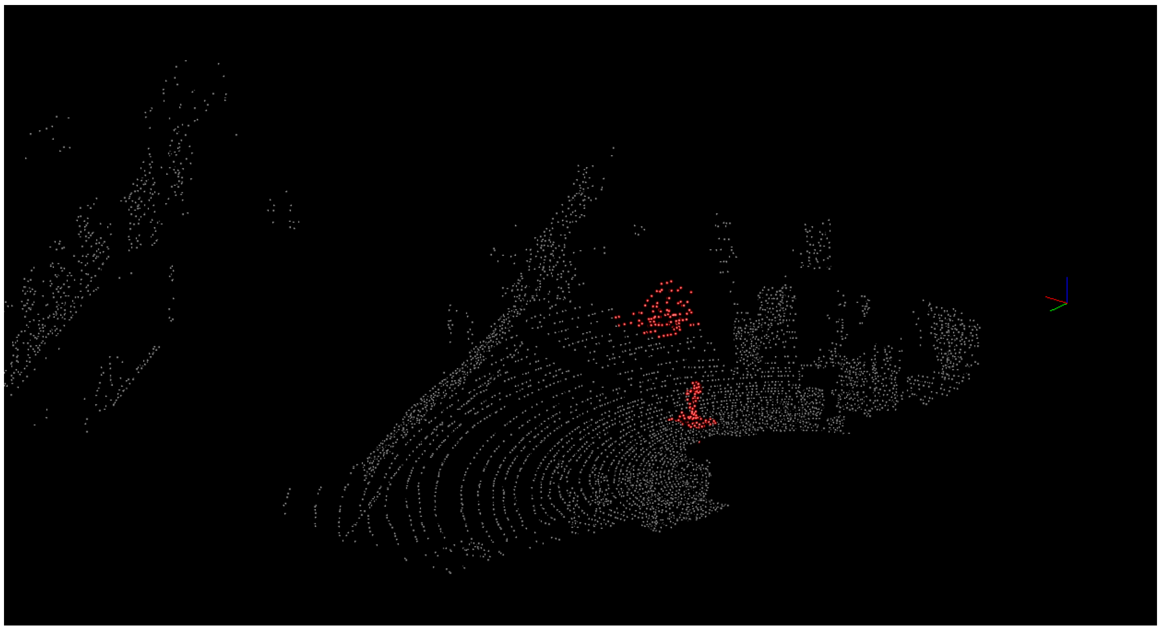


Figure 5.8: EdgeConv results

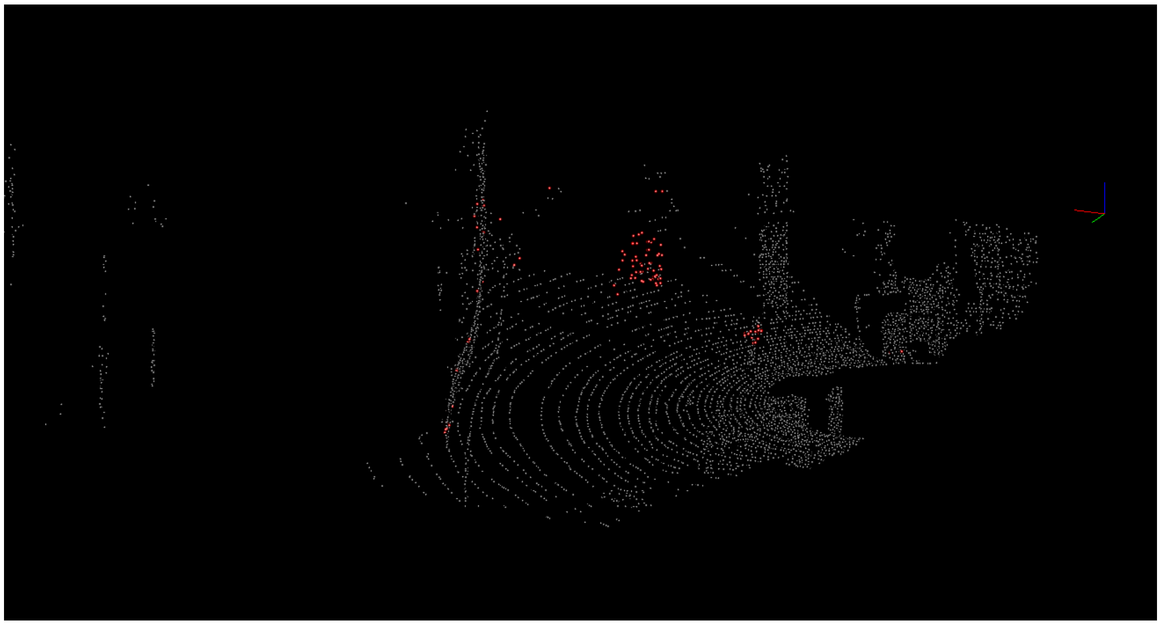


Figure 5.9: EdgeConv results

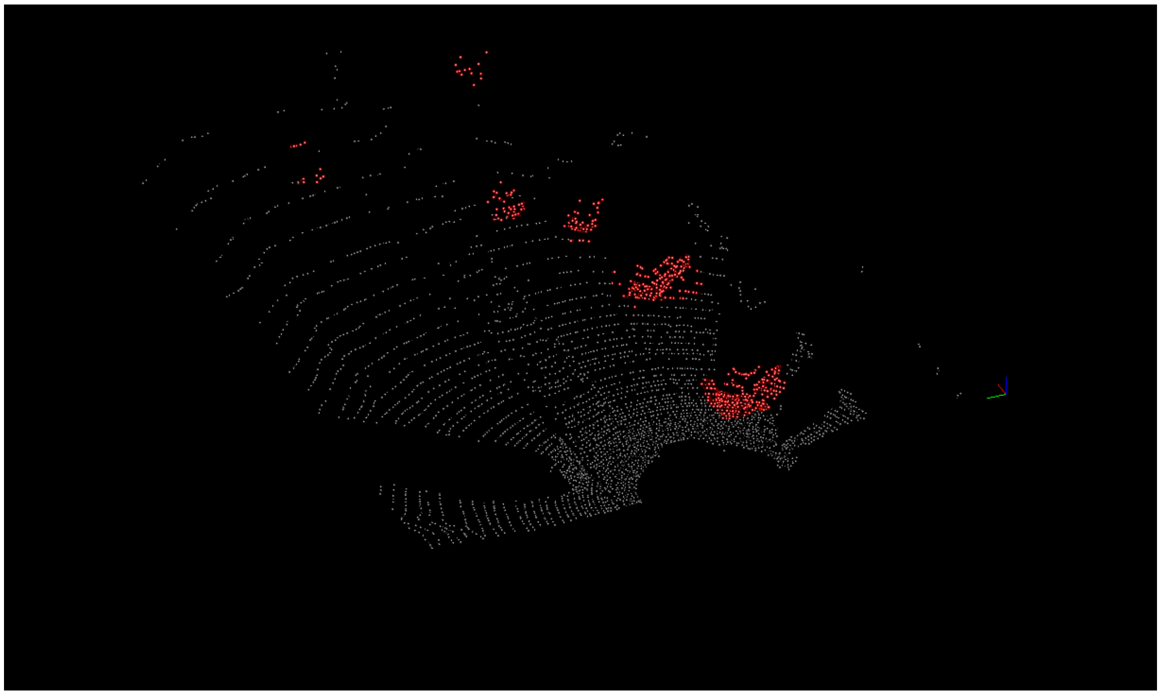


Figure 5.10: EdgeConv results