Exploring Local Delaunay Graph based Neural Network for 3D Object Detection

BIVASH PANDEY

Submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science, Honours Computing Science

at

Saint Mary's University
Halifax, Nova Scotia

Approved: Dr. Jiju Poovvancheri
Supervisor

Approved: Dr. Somayeh Kafaie
Reader

Approved: Dr. Somaye Arabi Naree
Reader

Date: April 26, 2022

Acknowledgments

I would like to thank Dr. Jiju Poovvancheri for his support, advice, and encouragement throughout my undergraduate years. I find myself lucky to have such a great mentor without whom this thesis would not have been possible. I am truly grateful to Dr. Somaye Arabi Naree and Dr. Somayeh Kafaie for all the invaluable feedback and suggestions. I am indebted to all the support of my parents that I would never be able to repay back. Finally, I would like to thank everyone who helped me directly or indirectly to make this journey possible.

Bivash Pandey
April 26, 2022

In loving memory of my grandmother.

Abstract

Exploring Local Delaunay Graph based Neural Network for 3D Object Detection

By

Bivash Pandey

Three-dimensional object detectors are essential components of the perception systems of autonomous robots. Most of the 3D object detection methods are point-based, voxel-based, or point-voxel-based (uses both point-based and voxel-based approaches). Recently, few works have used graph neural networks for 3D object detection and achieved promising results. Representation of a point cloud as a graph preserves the irregularity and removes the cost associated with the transformation into voxel grids or projection into a bird's eye view. In this thesis, we study the effects of point cloud encoding using local Delaunay graphs by embedding it in an existing graph neural network (Point-GNN). We perform experiments on the KITTI benchmark and find that graph-based methods achieve higher or comparable accuracy. Our results are comparable with most of the state-of-the-art methods. Further, our study indicates that the selection of the right type of proximity graph representation is crucial for real-time 3D object detection.

April 26, 2022

Contents

Chapter 1

Introduction

1.1 Introduction

Object detection is a method to detect certain classes of objects in the input data. The main goal is to identify objects and their locations in the input data (image, video, point cloud, etc.). In the last decade, we have witnessed remarkable results using Convolution Neural Network (CNN) based methods [1, 2, 3, 4, 5] for two-dimensional (2D) object detection. However, the environment that we interact with on a daily basis is three-dimensional (3D). 2D images lack depth information and are not a great choice for perception in real-world applications like autonomous driving, robot vision, etc. The convolution applied to the 2D images cannot be applied directly to 3D data because, unlike images, points in 3D space do not have a regular grid-like structure. Even placing a point cloud in a regular grid and applying convolution either causes information loss or unnecessary computation in empty voxels [6]. The development and affordability of 3D sensors (LiDAR sensor, RGB-D camera, etc.) as well as the availability of 3D datasets (KITTI [7], Waymo [8], etc.) have gained the interest of researchers in the field of computer vision, robotics, deep learning, and augmented reality. The data from these 3D sensors usually contains the coordinates of the points and additional properties like intensity, depth, RGB values, etc. Unlike the image (Figure 1.1), the point cloud (Figure 1.2) is irregular and unstructured, and therefore

the point cloud needs to be processed before feeding into the object detector.

Point cloud detectors can be categorized into voxel-based [9, 10, 11, 12, 13], point-based [14, 15, 16, 17], point-voxel-based [18, 19, 20, 21], and graph-based [6, 22, 23, 24, 25, 26]. There are fewer works that have looked into graph representation of the point cloud for 3D object detection. Point-GNN [6] encodes the point cloud as a radius-neighbour graph and SVGA-Net [25] constructs the complete graph for each local node. In this thesis, we explore the possibilities of representing the point cloud using proximity graphs. We consider the local Delaunay graph to capture the features in the proximity; the term Delaunay triangulation and Delaunay graph are used interchangeably, and it is described with examples in the next chapter. In this work, we make use of an existing graph neural network, i.e., Point-GNN [6] architecture to study the impact of the local Delaunay graph on the performance of 3D object detection in terms of accuracy and computation.

1.2 Challenges

There is a wide variety of challenges in extracting essential information with high accuracy from the point clouds. Some of the properties of point clouds that make 3D object detection a challenging problem are listed below.

1. **Irregularity:** Point cloud data in 3D space are not evenly distributed. Pixels in the image are arranged in a particular order, whereas points in the point cloud do not have any fixed ordering.

2. **Sparsity:** The volume occupied by the point cloud is sparse. Points near the lidar sensor are dense whereas points away from the lidar sensor are sparse which can be clearly seen in Figure 1.2. When such point clouds are transformed into voxels grid, most of the grids are empty and the computation performed on those empty voxels is wasted.

3. **Large number of points:** A point cloud usually contains a large number of points. While inspecting a few of the point clouds randomly from the KITTI [7]

dataset, we have found more than 100K points in each point cloud. Therefore the method operating on the point cloud should be able to process data efficiently.

Figure 1.1: Sample image from KITTI [7] dataset.

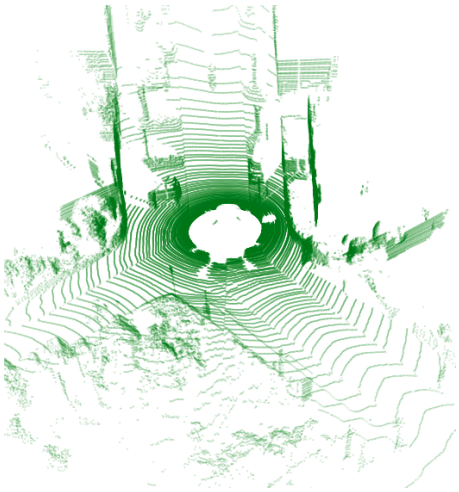


Figure 1.2: Sample point cloud from KITTI [7] dataset.

To overcome the above challenges, we investigate encoding the point cloud into the local Delaunay graph. Representing the point cloud in a graph structure preserves the irregularity of a point cloud. Moreover, it removes an overhead cost that comes into play while converting irregular point clouds into a grid structure. To process the point

cloud efficiently, we initially downsample the point cloud using voxel downsample [27].

1.3 Contributions

In this thesis, we analyze how local Delaunay graphs can affect the performance of 3D object detection from the point clouds. Our contributions can be summarized as follows:

- We use a local Delaunay graph (LDG) based representation of input point clouds to extract the structural information for 3D object detection.
- We investigate the possibilities of LDG representation by embedding it in the existing Point-GNN architecture (Figure 3.1).
- We perform a thorough experimental study on the KITTI [7] dataset to analyze the performance of the LDG based 3D object detection.

1.4 Organization

This thesis is organized into multiple sections as shown below:

- **Introduction:** This chapter introduces the studied problem, motivations, challenges, and our contributions.
- **Background:** This chapter includes the explanation and overview of different concepts required to have an in-depth understanding of graph-based object detection proposed in this thesis.
- **Related Work:** This chapter provides a literature review of the different methods and approaches for 3D object detection.
- **Local Delaunay Graph based Neural Network:** This chapter explains the proposed approach used in this thesis. Furthermore, we explain in detail how we use the local Delaunay graph to encode the point clouds.

- **Experimental Results:** This chapter demonstrates the results of the experiments performed on the KITTI dataset.
- **Conclusion and Future Work:** This chapter concludes our work and discusses the potential future directions to improve the model accuracy.

Chapter 2

Background

In this section, we provide the readers with basic terminology, definitions, concepts, and examples used in various discussions of this thesis.

2.1 Object Detection

We have defined object detection in the previous chapter. Let us look at an example of object detection in Figure 2.1 that shows the detection of cars in the input data. Our main focus in this thesis is 3D object detection from point clouds.

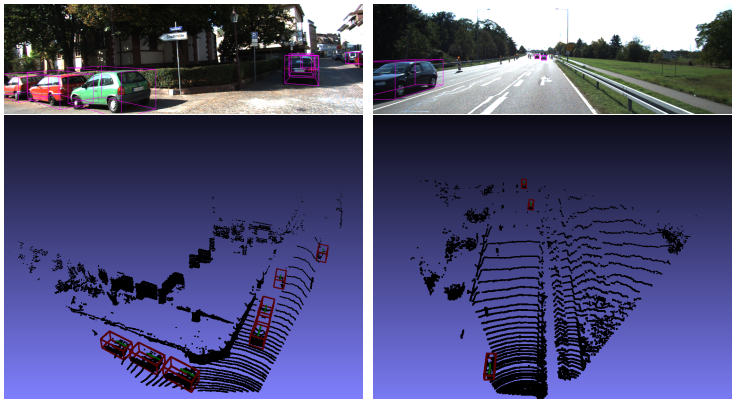


Figure 2.1: Object detection on the KITTI [7] dataset.

2.2 Point Cloud

Before diving into the point cloud, let us understand what a point is in 3D space. A point can be simply defined as a position in the space with x, y, and z coordinates. Further, the collection of such points in the space is known as point cloud. Let us denote a point p_i as a vector consisting of coordinates values x_i, y_i, and z_i; therefore, a point can be represented as a triple $p_i = (x_i, y_i, z_i)$.

Along with the coordinate values, other features can be associated with a point like reflectance, depth, RGB values, etc. Let r_i be the reflectance value of i^{th} point then, we can rewrite the point vector as $p_i = (x_i, y_i, z_i, r_i)$. A point cloud consisting of N points can be defined as: $P = \{p_i | i = 1, 2, 3....n\}, p_i \in \mathbb{R}^4$.

2.3 Voxel

Voxel is the smallest unit of volume on a discrete uniform grid in 3D space. A point cloud can be divided into equally spaced voxels. Let the length, width, and height of a point cloud boundary be represented by L, W, and H, and that of each voxel be represented by l, w, and h. Then the size of the 3D voxel is : $L' = L/l$, $W' = W/w$, $H' = H/h$. An example of voxel partitioning is shown in Figure 2.2.

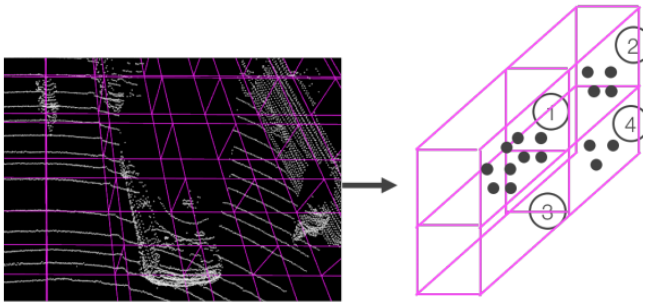
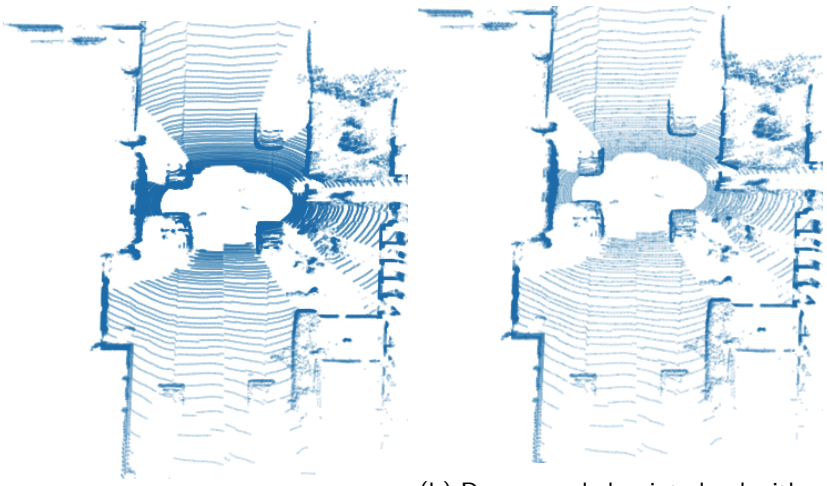


Figure 2.2: Division of 3D space into voxel grids. Figure clearly shows an uneven number of points in different voxels. Source: [9].

2.4 Downsampling

A point cloud usually contains a large number of points. In the case of the KITTI [7] dataset, there are more than 100K points in each point cloud. Constructing a graph with such a large number of points is computationally expensive. Therefore, before constructing the graph, the point cloud is downsampled to reduce the number of points with minimal loss of information. Random downsampling initially groups the point into voxels, and selects a point randomly within a voxel grid. Downsampling still preserves the structure of the point cloud. In Figure 2.3(b), all the points inside a voxel size of 0.1 are converted to a single point. Moreover, it can be clearly seen that the downsampled point cloud looks similar to that of the original, albeit containing much lesser points.



(a) Original point cloud.

(b) Downsampled point cloud with a voxel size of 0.1 in Open3D [27].

Figure 2.3: Voxel downsampling.

2.5 Graph

A graph can be defined as a tuple $G(V, E)$ where V represents the set of vertices and E represents the set of edges. A graph shows the relationship between the vertices

and edges. A graph with n vertices can be represented as an adjacency matrix A of size $n \times n$. For the unweighted and undirected graph:

$$A_{ij} = \begin{cases} 1 & : \text{if there is an edge between } v_i \text{ and } v_j \text{ i.e. } (v_i, v_j)\, \epsilon\, E, \\ 0 & : \text{otherwise.} \end{cases} \tag{2.1}$$

Adjacency matrix of a Delaunay graph shown in Figure 2.6(a) is:

Table 2.1: Adjacency Matrix.

	A	B	C	D	E	F	G	H
A	0	1	0	0	0	0	0	1
B	1	0	1	0	0	0	0	1
C	0	1	0	1	0	1	1	1
D	0	0	1	0	1	1	0	0
E	0	0	0	1	0	1	1	1
F	0	0	1	1	1	0	1	0
G	0	0	1	0	1	1	0	1
H	1	1	1	0	1	0	1	0

2.6 Radius Neighbour Graph

Given a point cloud $P = \{p_i | i = 1, 2, 3....n\}$, $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ consisting of N points, a radius neighbour graph is represented as $G = (P, E)$, where $p_i \in P$ is the set of vertices and E is the set of edges that connects vertices within a radius r.

$$E = \{(p_i, p_j)\,|\, d_{ij} < r\} \tag{2.2}$$

where $d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$

2.7 Delaunay Graph

A Delaunay triangulation $D(T)$ [28, 29] in 2D can be defined as the triangulation of the set of points P such that no point lies inside the circumcircle (Figure 2.4) of a triangle. Let us look at some of the examples in Figure 2.5 to understand what constitutes Delaunay triangulation and what does not. Furthermore, the above definition of 2D can be extended to 3D [30]. Delaunay triangulation in 3D can be defined as the triangulation of the set of points P such that no point lies inside the circumsphere of a tetrahedron. Delaunay triangulation of a few 2D points and 3D points is shown in Figure 2.6. We define the Delaunay graph as $G_d(V_d, E_d)$ where V_d is the set of points in $D(T)$ and E_d consists of the edges of triangles in $D(T)$.

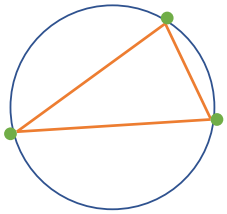
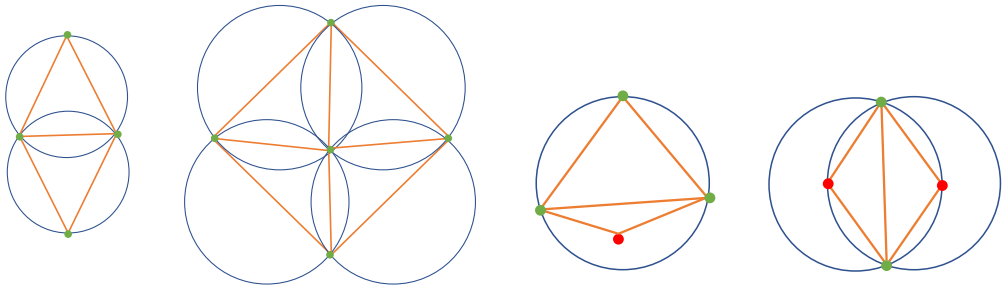
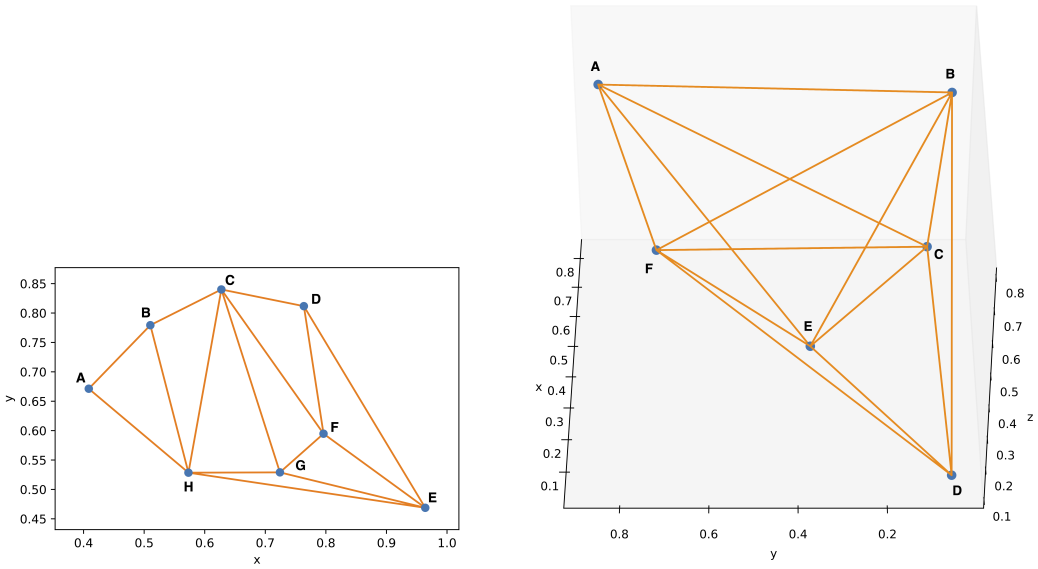


Figure 2.4: Circumcircle of a triangle.



(a) Delaunay triangulation. (b) Non Delaunay triangulation.

Figure 2.5: Illustration of Delaunay and non-Delaunay triangulation. Red points in the Figure 2.5(b) do not statisfy the condition of Delaunay triangulation as they lie inside the circumcircle of a triangle.



(a) Delaunay triangulation in 2D.

(b) Delaunay triangulation in 3D.

Figure 2.6: Delaunay triangulation in 2D (a) and 3D (b).

2.8 Graph Neural Network (GNN)

GNN is a special type of neural network that processes graph-structured data. A vertex in a graph can have any number of features. In our case, intensity is the feature associated with each point. CNNs cannot be applied directly to graph-structured data because of the irregular structure. However, GNNs are capable of handling graph data and performing node level as well as graph level classification and detection. As indicated in Equation 2.3, GNN iteratively updates the node features by aggregating features from the neighbourhood [31].

$$h_u^t = \phi^{t-1}(h_u^{t-1}, \lambda^{t-1}(\{h_v^{t-1}, \forall\, v\, \epsilon\, N(u)\})) \tag{2.3}$$

$N(u)$ is the set of neighbours of node u. λ is the aggregation function that aggregates the features from all the vertices $v \epsilon N(u)$ whereas ϕ is the update function that updates

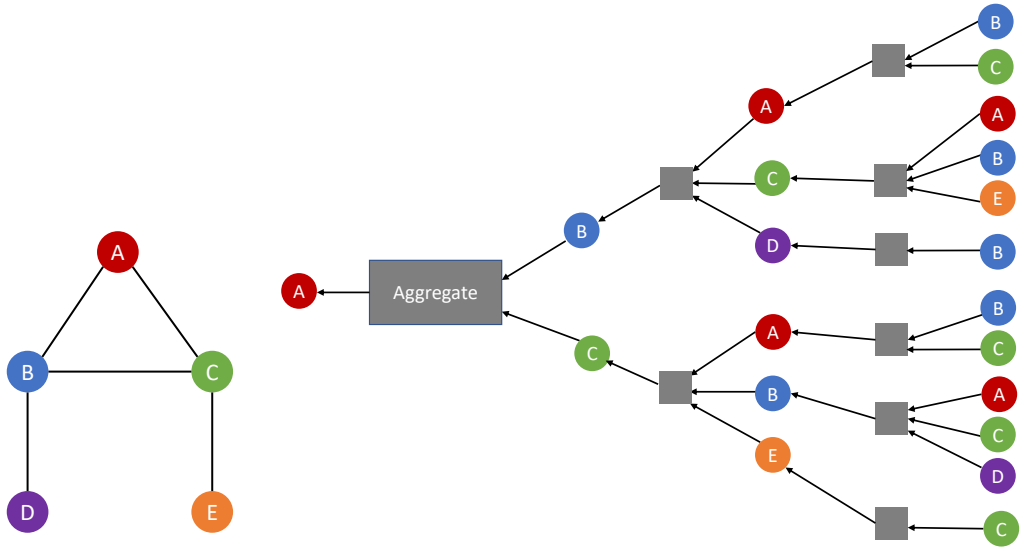
the feature of node u at the t^{th} iteration by combining the aggregated features with its own features from the previous step.

The simplest form of GNN can be viewed as a message-passing [31] and defined as:

$$h_u^t = \sigma(W_u\, h_u^{t-1} + \sum_{v \epsilon N(u)} W_v\, h_v^{t-1}) \tag{2.4}$$

W_u and W_v are trainable weight matrices. σ is the non-linear activation function (Sigmoid, ReLU, Tanh, etc). The main goal of the activation function is to introduce non-linearity and help the network learn the complex functions. h_u^t denotes the updated feature representation of node u at t^{th} step.

In Figure 2.7, we illustrate how a node aggregates features from its neighbours.



(a) Input graph. (b) Feature aggregation.

Figure 2.7: Feature aggregation by node 'A' from its neighbours. This is an example of three layers feature aggregation.

Chapter 3

Related Work

In this chapter, we review different approaches for 3D object detection from the point cloud.

3.1 3D Object Detection

3D object detection is an approach to detect objects in 3D data (e.g., LiDAR data) with 3D bounding boxes and finding the category in which that particular object belongs. Advancement in computer hardware and availability of datasets like KITTI [7], Shapenet [32], ApolloScape [33], Waymo [8], Lyft [34], A2D2 [35], nuScenes [36], and Objectron [37] has gained the interest of research community in the direction of 3D object detection. 3D object detection has many applications in real life such as autonomous driving [38, 39], robot vision [40], and augmented reality [41]. We group different 3D object detection methods into four categories: voxel-based, point-based, point-voxel-based, and graph-based.

3.2 Voxel-based methods

Since the point cloud data obtained from the LiDAR sensor is irregular, convolution cannot be applied directly to the irregular data. Therefore, voxel-based methods con-

vert the irregular data to regular data by placing the points in the voxel grids to extract features using CNN in 3D. VoxelNet [9] is an end-to-end architecture that removes the need for manual feature engineering by operating directly on the sparse point cloud. PointPillars [10] places the point clouds in the vertical structure and makes use of 2D convolutional layers in the pillars for 3D object detection. Using the same voxel size might not be efficient to learn feature representation. To eliminate this issue Voxel-FPN [11] uses multi-scale voxel partitioning and removes the need of finding the perfect voxel size for a better result. Voxel R-CNN [12] achieves comparable accuracy as the point-based methods using a two-stage approach which consists of a 3D backbone network, a 2D bird-eye view Region Proposal Network, and a detection head. Moreover, [12] illustrates that the precise positioning of raw points is not necessary to achieve higher performance in 3D detection. The Triple Attention module of the TANet [13] enhances the discriminative points whereas suppresses the unstable points of the target. It uses a triple attention module (point-wise, channel-wise, and voxel-wise) which are then stacked to achieve the multi-level feature attention.

3.3 Point-based methods

Unlike voxel-based methods, point-based methods directly process the point clouds without converting them into a grid structure [9, 42] or other forms like bird's eye view (BEV) [43, 44, 45]. PointNet [14], the pioneering method to learn from points, uses raw point clouds as input and generates class labels or per point segment labels as output for the classification and segmentation task. It uses multi-layer perceptrons (MLPs) to extract features from the point clouds and achieves a decent accuracy. Later PointNet++ [46] improved PointNet [14] by capturing the local structures which was one of the limitations of PointNet [14]. Moreover, PointNet++ has less error rate with higher accuracy as compared to PointNet. These two methods handle the irregular point cloud. PointRCNN [15] operates in 2 stages. First, it generates the 3D proposals from the raw point cloud by segmenting the point cloud into foreground points and then learns local spatial features along with global semantic features for 3D bounding box refinement. 3DSSD [16] is a lightweight point-based single-stage 3D

object detector that reduces inference time by removing feature propagation layers and refinement modules. LiDAR R-CNN [17], a second-stage detector that can be plugged into any 3D object detector, solves the problem of proposal sizes being ignored by learned features in the point-based methods. Point-based 3D detectors are high performing as compared to voxel-based, however, that accuracy comes at the price of higher computational cost [12].

3.4 Point-Voxel-based methods

Point-Voxel CNN [18] leverages the power of the point-based and the voxel-based methods. It is memory-efficient because input data are represented in points, and are less irregular as convolutions are performed on voxels. Unlike PVCNN [18], Fast Point R-CNN [19] initially uses voxel representation to get the initial results and then uses light-weight PointNet to improve the predictions from the first step. Furthermore, to make each point aware of its context information, Fast Point R-CNN [19] uses an attention mechanism which in turn improves the accuracy. PV-RCNN [20] benefits from 3D voxel CNN for multi-scale feature representations and 3D proposals, whereas from a PointNet-based network for location information. Unlike others, HVPR [21] exploits voxel-based and point-based features to acquire 3D representation as a pseudo image. Moreover, HVPR [21] introduces a memory module to augment point-based features and an Attentive Multi-Scale Feature module to generate scale-aware features of the irregular point clouds. However, there is a possibility that these kinds of hybrid methods might face the drawback of both representations [6].

3.5 Graph-based methods

In 2017, Simonovsky et al. [22] proposed a new approach where a graph is constructed from the point cloud. Each point is treated as a vertex and each directed edge (j, i) is formed by connecting vertices i to all the vertices j in a neighbourhood. Later, [23, 24] used k-nearest neighbour (k-NN) graph for the point cloud classification and part segmentation task. There are a few methods that encode the point cloud as a

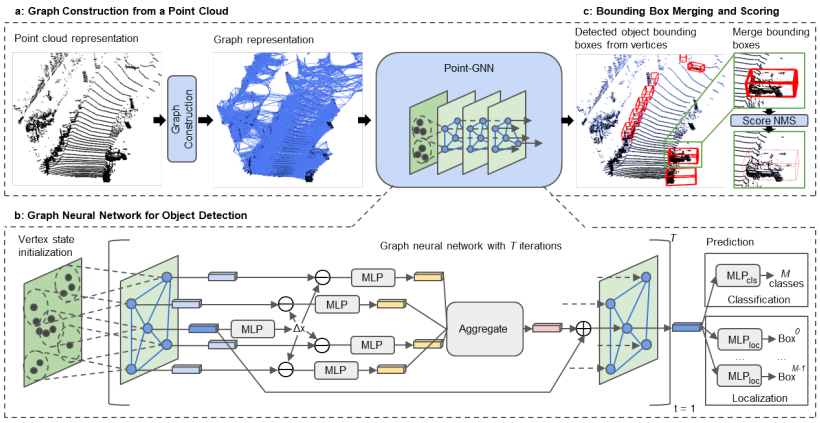


Figure 3.1: Point-GNN architecture. Source: [6].

graph for 3D object detection. Point-GNN [6] is one of the earliest methods to use a graph-based approach for 3D object detection. It requires more time and memory to construct graphs using all the points in a point cloud as vertices. Therefore, Point-GNN (Figure 3.1) initially downsamples the point cloud and encodes it as a fixed radius-neighbour graph which eliminates the need of making a point cloud regular. Point-GNN [6] updates the node features by repeatedly extracting features from its neighbours and predicts the category and bounding box by using vertex feature values after T iterations. Furthermore, it introduces an auto-registration mechanism to minimize translation invariance. SVGA-Net [25] constructs a local complete graph among the points within a certain radius r and a global k-NN graph between voxels. It uses an attention mechanism to enhance the features with the help of a local complete graph and a global k-NN graph. Thakur et al. [26] use graph representation to encode the point cloud and masked attention for feature aggregation by assigning variable weights to the nodes.

Graph-based methods have higher accuracy, however they are not the fastest when it comes to inference time. Inference time of SVGA-Net [25] is 62 ms and that of Point-GNN [6] is 643 ms. PointPillars [10], a voxel-based method has an inference time of 16.2 ms which is almost real-time. Therefore, there is still room to improve inference time of graph-based approaches.

Chapter 4

Local Delaunay Graph based Neural Network

Previously, the Delaunay graph has been used in clustering, segmentation [47], and classification [48]. To the best of our knowledge, there exists no graph-based method for 3D object detection that makes use of the Delaunay graph to encode the point clouds. Our work makes use of the existing Point-GNN [6] architecture (Figure 3.1) and modifies the graph representation to test the performance of the local Delaunay graph for 3D object detection tasks. Our major contribution is the graph construction step and the experimental study; other steps are similar to that of Point-GNN [6].

4.1 Graph Construction

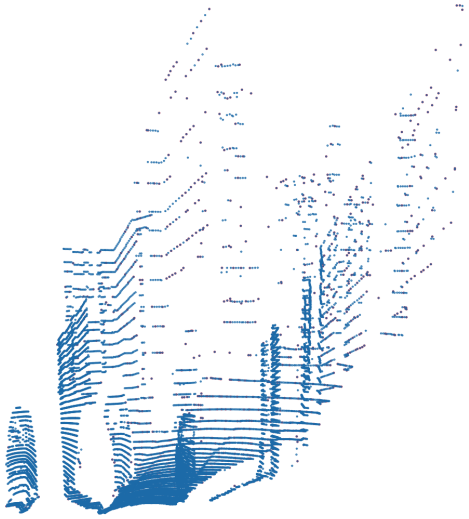
Let $X = \{x_i | i = 1, 2, 3....n\}, x_i \in \mathbb{R}^3$ be N points and $F = \{f_i | i = 1, 2, 3....n\}, f_i \in \mathbb{R}^k$ be the features associated with corresponding points in a point cloud P. Then a point cloud can be denoted as $P = \{p_i | i = 1, 2, 3....n\}$, where $p_i = (x_i, f_i)$. Each point has both 3D coordinates and k features. In the case of the KITTI [7] dataset, the reflectance value is the feature associated with each point.

Since the point cloud contains a large number of points, constructing the graph straight out of those points by considering each point as a vertex is computationally expensive. Therefore, the input point cloud is initially downsampled using the voxel downsampling method [27]. Let P' be the downsampled point cloud. For each point p_j in P', we find the neighbours of p_j within radius r in the original point cloud P. Let P_n be the neighbours of p_j, $p_j \in P'$. Finally, we construct the Delaunay graph from the vertices of P_n.

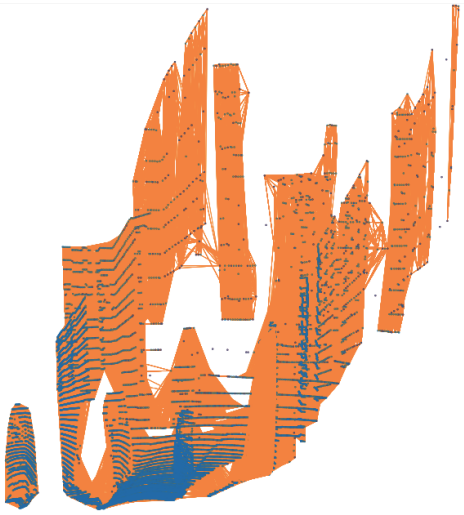
Instead of constructing the Delaunay graph locally, there is a possibility to construct the Delaunay graph of a point cloud globally without finding the neighbours within a certain radius. It will obviously be faster than the local Delaunay graph. However, we then get longer edges in the graph which do not contribute to the feature aggregation for 3D object detection. To preserve the local features by aggregating features from nearby neighbours we used the local Delaunay graph. For five or more points in P_n, we construct a local Delaunay graph; however, when there are less than five points in P_n, we form a graph by connecting the center point with other points in P_n rather than neglecting those points.

After the construction of the Delaunay graph using points in P_n, we keep only those edges that are connected to p_j (vertex used for finding the neighbours). Therefore the number of edges in this local Delaunay graph are less than in the radius neighbour graph. However, constructing the local Delaunay graph takes a longer time than the radius neighbour graph. The number of vertices, edges, and time to construct the graph are shown in Table 5.4. Once we finish the local Delaunay graph representation, we pass the graph to GNN. We have shown the local Delaunay graph construction steps in Figure 4.1 and the architecture of our approach in Figure 4.3. Figure 4.2 shows the visualization of the local radius neighbour graph and the local Delaunay graph.

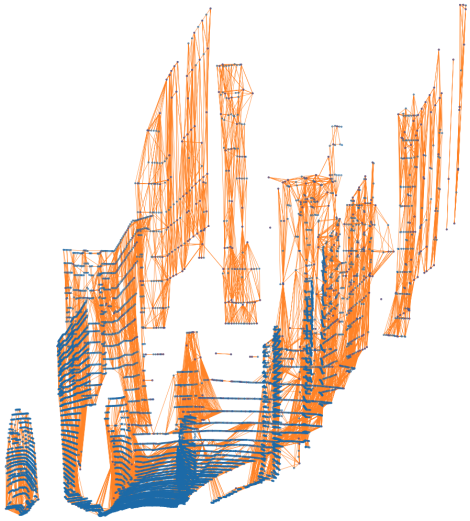
Delaunay graph in 3D forms the tetrahedra. Unlike other graphs, there is an opportunity to exploit additional feature information from the Delaunay graph like the area of triangles and the volume of tetrahedra. These extra features might help to improve the accuracy of the model.



(a) Input point cloud in blue and downsampled in red.

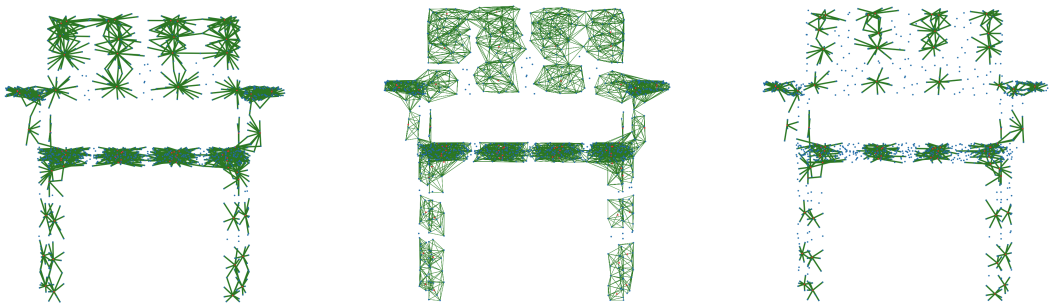


(b) Local Delaunay graph construction.



(c) Only edges connected to the downsampled key points.

Figure 4.1: Graph construction steps. Orange lines in (b) and (c) represent the edges. Graph from step (c) is passed to GNN.



(a) Radius Neighbour graph. (b) Local Delaunay graph. (c) Local Delaunay graph with edges connected to center vertex.

Figure 4.2: Local radius neighbour graph vs local Delaunay graph. Point cloud [49].

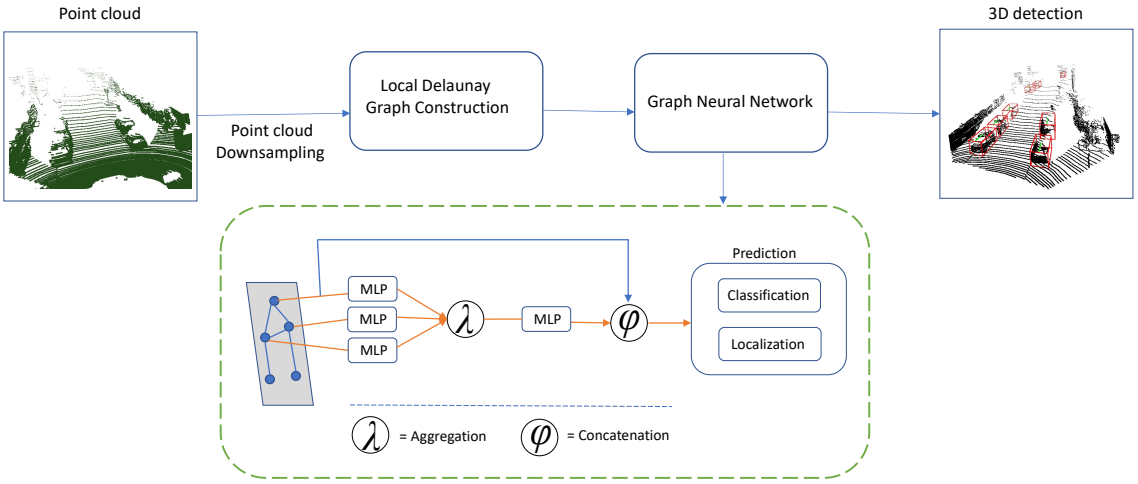


Figure 4.3: The architecture of the proposed approach. This approach uses a local Delaunay graph for the point cloud representation.

4.2 Graph Neural Network for Object Detection

A GNN updates the vertex features by aggregating features from its neighbours as shown in Equation 2.3 and Figure 2.7. Our work uses the Point-GNN architecture

(Figure 3.1), therefore the GNN used in this thesis is exactly the same as described in the Point-GNN [6]. We reproduce the relevant details of Point-GNN in the subsequent paragraphs. For object detection, we modify the GNN from Equation 2.3 to Equation 4.1.

$$h_u^t = \phi^{t-1}(h_u^{t-1}, \lambda(\{f^{t-1}(x_{uv}, h_v^{t-1})\})), (u,v)\,\epsilon\, E \tag{4.1}$$

where, $x_{uv} = x_v - x_u$, is the difference between relative coordinates of the neighbours. f^{t-1}, λ, and, ϕ are the differentiable functions. f^{t-1} computes the edge features between two vertices x_v and x_u by using relative coordinates of the neighbours. A relative coordinate is used to achieve the translation invariance. λ is the aggregation function that aggregates the edge features. ϕ updates the feature of node u at t^{th} iteration by combining the aggregated edge features with its own node features from the previous step $(t-1)$.

Translation invariance achieved with the relative coordinates was not desirable. Therefore, [6] introduces an auto-registration mechanism for the better translation invariance.

$$h_u^t = \phi^{t-1}(h_u^{t-1}, \lambda(\{f^{t-1}(x_{uv} + \Delta x_u^{t-1}, h_v^{t-1})\})) \tag{4.2}$$

where, $\Delta x_u^{t-1} = c^{t-1}(h_u^{t-1})$ is the coordinate offset of the vertex and c^{t-1} computes the offset with the help of the center vertex. ϕ^{t-1}, f^{t-1}, and c^{t-1} are represented using multi-layered perceptrons. Finally, after the completion of GNN iterations, the refined feature values of a vertex are used to predict the class and the bounding box of the object.

4.3 Loss Function

In this thesis, we have used the same loss functions as Point-GNN [6]. Total loss is the combination of classification loss, localization loss, and regularization loss. For the classification loss, average cross-entropy loss is used. Let M be the total number of classes and $\{\rho_j | j = 1, 2, 3, ..., M\}$ be their probabilities. Then the average cross-

entropy loss is given as in Equation 4.3.

$$L_{cls} = \frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_j^i \, log(\rho_j^i) \tag{4.3}$$

where y^i is the class label and ρ^i is the predicted probability of the node i. We parameterize a 3D bounding box as $(x, y, z, l, h, w, \theta)$, where (x, y, z) is the center of a bounding box, (l, h, w) is the dimension of the box (length, height, and width), and θ is the yaw angle. Let $b_p = (x_p, y_p, z_p, l_p, h_p, w_p, \theta_p)$ be the predicted bounding box and $b_t = (x_t, y_t, z_t, l_t, h_t, w_t, \theta_t)$ be the ground truth of a bounding box. Then the localization loss is the difference between the predicted bounding box and its ground truth. For a vertex inside the bounding box, the average localization loss is computed using Huber loss [50] as shown in Equation 4.4.

$$L_{loc} = \frac{1}{N} \sum_{i=1}^{N} \sum_{b \in (x,y,z,l,h,w,\theta)} L_{huber}(b_p - b_t) \tag{4.4}$$

Finally, L1 regularization is added for reducing the over-fitting of the model. Then the total loss becomes:

$$L_{total} = \alpha L_{cls} + \beta L_{loc} + \gamma L_{reg} \tag{4.5}$$

where, α, β, and γ are constants. We have not modified the parameters and hyper-parameters other than the training steps to compare the experimental results with consistency.

4.4 Implementation

We used Point-GNN [6] code which is available on GitHub [51]. We have used Python 3.7.7, Numpy [52], Scipy [53], Scikit Learn [54], Tensorflow [55], Open3D [27], Mesh-Lab [56] for the representation of point clouds using local Delaunay graph. Although the original code was written with Tensorflow 1.15, we were unable to run the program with the same version because of the conflicts with other packages in the Linux

server. Therefore, we installed Tensorflow 2.6 and used its backward compatibility feature to run the program.

4.5 Training

We have trained different models (T1, T2, T3) to detect cars in the LiDAR data using the local Delaunay graph. T1, T2, and T3 refer to a model with 1, 2, and 3 GNN layers respectively. Most of the parameters are kept the same as that of Point-GNN [6]. We trained T1 with a batch size of 4 whereas T2 and T3 were trained with a batch size of 2. The initial learning rate was set to 0.125 with a decay rate of 0.1 every 400K steps. Moreover, we trained the proposed network up to 1250K steps with loss weights $\alpha = 0.1, \beta = 10, \gamma = 5e - 07$. The Delaunay graph was constructed locally with a neighbourhood of $r = 4m$. While training, the maximum number of neighbours per vertex was set to 256 and the point cloud was downsampled with a voxel size of $0.8m$. Whereas during inference, all the neighbours were considered and the point cloud was downsampled with a voxel size of $0.4m$. It took around 20-22 days to train a model on the Cedar server.

Chapter 5

Experimental Results

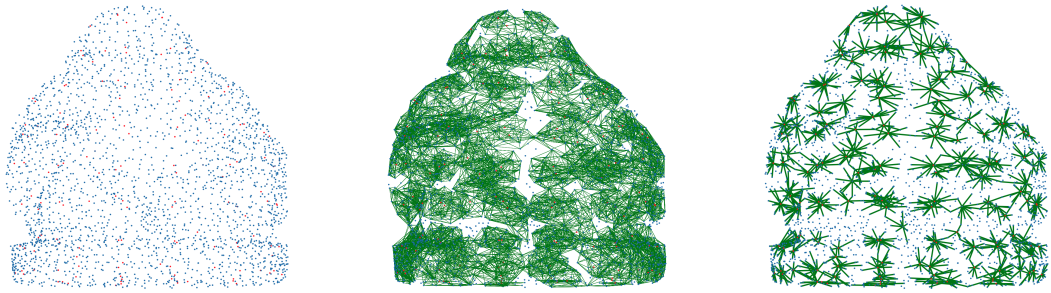
5.1 System

We have used Compute Canada (www.computecanada.ca) resources to train all the models used in this thesis. A model trained in ACENET (https://ace-net.ca/) cluster has used 2 RTX 6000 GPUs of 16 GB each with a batch size of 4 whereas models trained in Cedar (https://www.westgrid.ca/) cluster have used 1 Volta GPU of 32 GB with a batch size of 2.

5.2 Dataset

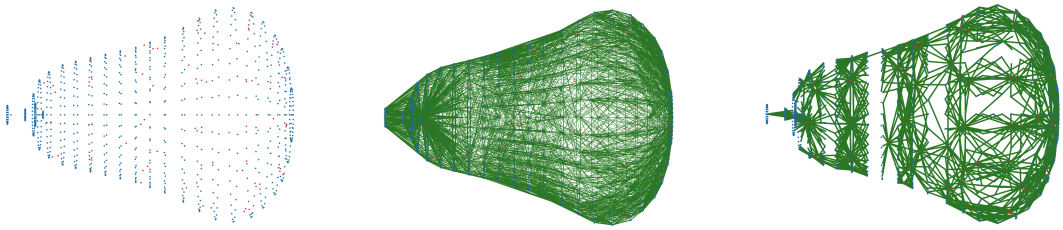
We used KITTI [7] dataset to test the performance of the local Delaunay based 3D object detection model. It contains 7481 training data and 7518 testing data. A sample of point cloud data visualization from the KITTI dataset is shown in Figure 1.2.

5.3 Graph Visualization



(a) Point cloud. (b) Local Delaunay graph. (c) Edges connected to the center vertex.

Figure 5.1: Graph construction steps.



(a) Point cloud. (b) Local Delaunay graph. (c) Edges connected to the center vertex.

Figure 5.2: Graph construction steps.

In Figure 5.1 and Figure 5.2, we have shown the steps involved in graph construction. Step (a) shows the initial point cloud in blue and the downsampled point cloud in red. Step (b) represents the graph construction using the local Delaunay graph. Finally, step (c) shows only edges of the local Delaunay graph that are connected to the center vertex. The point clouds used in Figure 5.1 and Figure 5.2 are downloaded from [57].

5.4 Results

In this section we present both the quantitative and qualitative results and compare our results with other 3D object detection methods. We have used KITTI validation split to generate the results. In our experiments, we have trained our network using a train split of 3260 samples and evaluated the results in the validation split of 3769 samples. The split files (train_car.txt, val.txt) are downloaded from [51].

5.4.1 Quantitative Results

To obtain the average precision results, we have used code from [58]. We have evaluated the average precision result on Easy, Moderate, and Hard levels of difficulty using the KITTI [7] dataset. In the Easy level, the size of each bounding box is higher and objects are fully visible. In the Moderate level, objects are partially occluded with smaller bounding boxes. In the Hard level, objects are difficult to see.

Precision and Recall is defined mathematically as [59]:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.2}$$

where,
TP = True Positive, denotes object was detected when it was supposed to be.
FP = False Positive, denotes object was detected when it was not supposed to be.
FN = False Negative, denotes object was not detected when it was supposed to be.

Average Precision (AP) is the area under the precision-recall curve [59].

$$\text{AP} = \int_0^1 p(r)\, dr \tag{5.3}$$

Table 5.1: The Average Precision (AP%) comparison of car detection on the KITTI validation split.

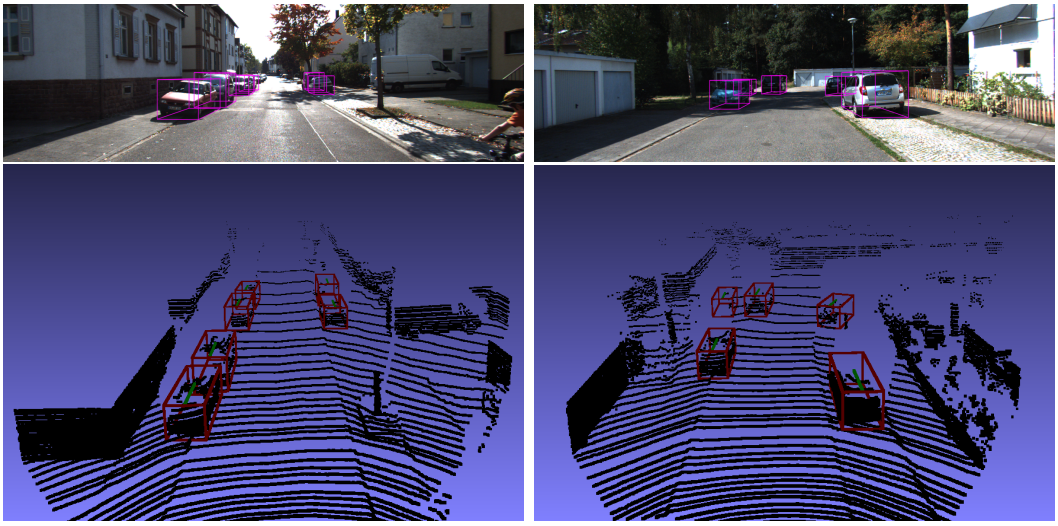
Method	Car 3D AP%			Car BEV AP%		
	Easy	Moderate	Hard	Easy	Moderate	Hard
VoxelNet [9]	81.97	65.46	62.85	89.60	84.81	78.57
Voxel-FPN [11]	88.27	77.86	75.84	90.20	87.92	86.27
Voxel R-CNN [12]	89.41	84.52	78.93	-	-	-
TANet [13]	87.52	76.64	73.86	-	-	-
PointRCNN [15]	88.88	78.63	77.38	-	-	-
3DSSD [16]	89.71	79.45	78.67	-	-	-
F-PVCNN [18]	85.25	72.12	64.24	-	-	-
Fast Point R-CNN [19]	89.12	79.00	77.48	90.12	88.10	86.24
HVPR [21]	91.14	82.05	79.49	-	-	-
PointGNN [6]	87.89	78.34	77.38	89.82	88.31	87.16
SVGA-Net [25]	90.59	80.23	79.15	90.27	89.16	88.11
Thakur et al. [26]	83.54	74.47	63.84	90.12	87.05	75.48
Ours	87.65	77.67	76.27	89.91	87.69	86.38

Table 5.1 shows the comparison of our proposed approach with other existing methods in the car category using validation split. Unlike most other methods, we train it on the subset of train split in which samples without a car are excluded. The accuracies of other methods reported in Table 5.1 are extracted from the respective paper. Our graph-based approach sits in the middle for all the difficulty levels. And our results are almost close to the state-of-the-art methods. Our method performs better than voxel-based methods (like VoxelNet [9], TANet [13]), Point-voxel-based methods (like F-PVCNN [18]), and graph-based methods (like Thakur et.al. [26]). On the easy difficulty level of BEV (Bird's Eye View), our method surpasses Point-GNN [6].

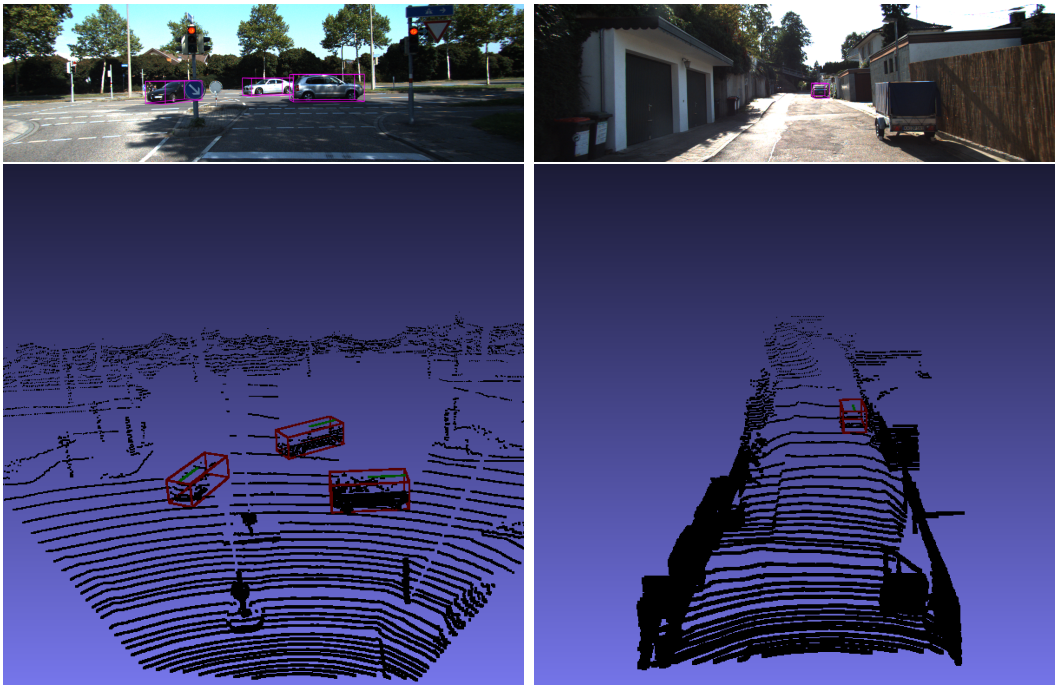
In this study, we still have not utilized the full potential of the local Delaunay graph by removing the edges not connected to the center vertex. We tried to incorporate those edges as well, but we got an unexpected error while integrating our graph in the Point-GNN code [51]. Therefore, we planned to drop the edges not connected to the center vertex.

5.4.2 Qualitative Results

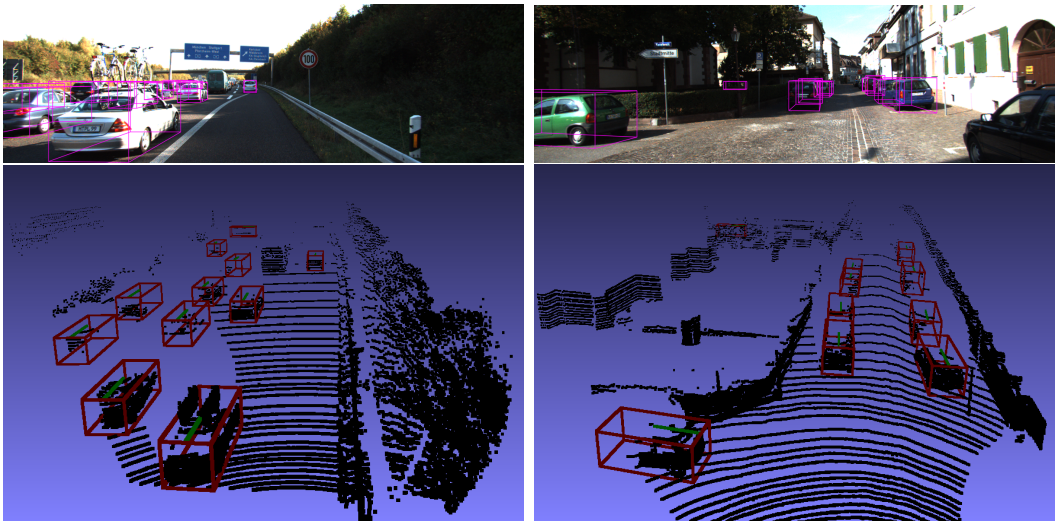
To create the visualization of our prediction results, we used code from [60] and visualized point clouds in MeshLab [56]. We demonstrate our prediction results in Figure 5.3 where we visualize 3D bounding boxes both on point clouds and images. We can clearly see our method can detect in varying lighting conditions as well as in occluded conditions. In Figure 5.3(c, right) and in Figure 5.3(d, left), our model is capable of detecting cars that are not clearly visible in the images. These detection results show that cars at a distant are also detected, not just in the proximity. Moreover, Figure 5.3(c) illustrates that even the occluded cars are detected perfectly. Not only the numeric results, but the visualization also displays a great detection result.



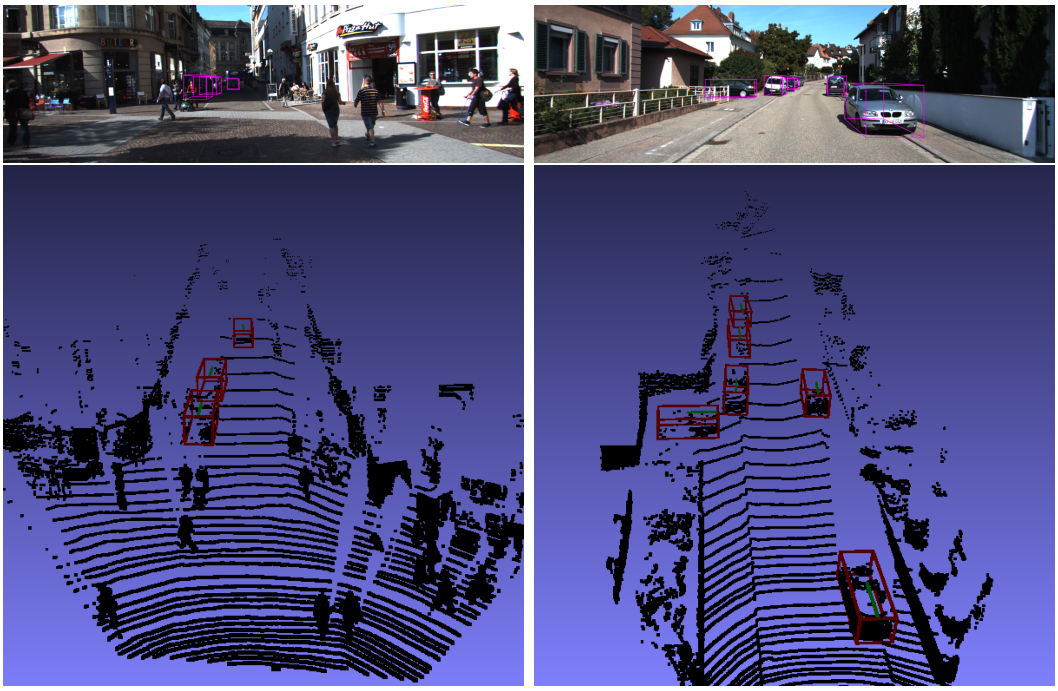
(a)



(b)



(c)



(d)

Figure 5.3: Qualitative analysis on the KITTI validation set.

5.5 Ablation Study

Further, we studied how the number of layers in GNN affects the performance. We even performed the comparison of total inference time between Point-GNN and our approach. Moreover, we looked into the number of edges and time taken to generate the edges in the radius neighbour graph and the local Delaunay graph. Finally, we explored the effect of different radii on accuracy.

5.5.1 Number of Layers in GNN

Here, we study the relationship between the number of layers in GNN and the Average Precision (AP) value both in 3D and BEV. We have shown our results in Table 5.2. We have found that increasing the number of layers from 1 to 3 increases the accuracy both in 3D as well as in BEV.

Table 5.2: The Average Precision (AP%) comparison on different number of GNN layers.

Number of layers	Car 3D AP%			Car BEV AP%		
	Easy	Moderate	Hard	Easy	Moderate	Hard
1	83.31	73.65	67.10	88.93	85.50	82.99
2	85.28	75.91	73.44	89.53	87.29	85.30
3	87.65	77.67	76.27	89.91	87.69	86.38

5.5.2 Inference Time

Using our approach, the average processing time for one sample in the validation set takes about 9.94 seconds which is pretty high as compared to 643 ms of Point-GNN. In our case, most of the time is spent on creating the Delaunay graph representation of the point cloud. It takes almost 98.5% (9.79 seconds) of the total time. This illustrates the construction of the local Delaunay graph is expensive.

If we compare the time without the graph generation step, our approach performs faster than Point-GNN. Point-GNN takes 522 ms whereas ours take 150 ms. Less

number of edges generated by our graph representation might be the reason for faster inference time in the later stage. Taking graph representation time into account, we can say that this approach would not be practical for applications like autonomous vehicles where it requires lower inference time.

5.5.3 Number of Edges

We have tested the number of edges generated by a local radius-neighbour graph and a local Delaunay graph. The time to construct those graphs is measured locally in an 8 GB MacBook air without any GPU. This time does not necessarily align with the graph generation time during inference performed with GPUs and higher RAM. The time and the number of edges shown in Table 5.3 and Table 5.4 are based on the average of 10 tests. For this test, the radius is set to $1m$ and the maximum number of neighbours to 256. We performed the experiment on 3 different sub-samples of the point cloud [7].

Table 5.3: Number of edges in a graph using local radius neighbour graph.

Subsample #	Number of vertices	Number of Edges	Time (seconds)
1	18739	94307	0.027
2	18988	90897	0.033
3	19946	96555	0.028

Table 5.4: Number of edges in a graph using local Delaunay graph.

Subsample #	Number of vertices	Number of Edges	Time (seconds)
1	18739	13645	2.082
2	18988	17215	2.223
3	19946	16666	2.293

5.5.4 Average Precision with Different Radii

We have compared the accuracy of the two LDG based models with 2 GNN layers trained up to 65200 steps. One of the models uses a radius of $2m$ whereas another uses a radius of $4m$. The result in Table 5.5 shows that a smaller radius leads to lower accuracy. It might be because of the fact that a node aggregates features from the less number of points in the smaller neighbourhood.

Table 5.5: The Average Precision (AP%) comparison on different radii.

Radius	Car 3D AP%			Car BEV AP%		
	Easy	Moderate	Hard	Easy	Moderate	Hard
$2m$	70.82	61.84	55.13	87.45	77.42	75.02
$4m$	83.53	67.60	64.30	88.90	82.96	77.26

Chapter 6

Conclusion and Future Work

In this study, we have used the local Delaunay graph for the representation of point clouds and analyzed its performance for 3D object detection. We have clearly seen that its accuracy on validation split of KITTI benchmark is close to Point-GNN [6] and higher than some other point-based and voxel-based methods as shown in Table 5.1. For the BEV easy category, the proposed method performs better than Point-GNN in terms of accuracy. However, the accuracy of our approach comes at a price of higher computational time due to the Delaunay construction algorithm.

6.1 Conclusion

This study shows the accuracy of our method is higher than few other methods as illustrated in Table 5.1. While comparing it with Point-GNN [6], accuracy has not been improved for 3D detection. However, in the Easy difficulty level of BEV, the proposed model showed better accuracy. One possible reason for slightly lower accuracy of our model could be the exclusion of edges not connected to the center vertex. Moreover, we found that the local Delaunay graph representation is a lot slower as compared to a radius-neighbour graph representation. This is because the local Delaunay graph construction is computationally expensive. Although our graph-based approach yields higher accuracy, speed could be an issue when implemented in real-life applications

like autonomous vision. Further optimizations could be possible to enhance the speed of the local Delaunay construction method.

6.2 Future Work

In this work, we have not exploited the full potential of the local Delaunay graph. Firstly, all the edges of the local Delaunay graphs can be used to find the true accuracy of this architecture. By using Delaunay graph representation of the point clouds, we have access to additional properties like the volume of tetrahedra as well as the area of triangles. Therefore, the possibility of exploiting volume and area features can be explored further. Moreover, this same representation can be experimented with different tasks like classification and segmentation. The model can be tested on different datasets like Waymo [8] and nuScenes [36]. Additionally, further optimizations can be considered to reduce the inference time without compromising the accuracy.

Bibliography

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[4] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[5] B. B. Traore, B. Kamsu-Foguem, and F. Tangara, "Deep convolution neural network for image recognition," *Ecological Informatics*, vol. 48, pp. 257–268, 2018.

[6] W. Shi and R. Rajkumar, "Point-gnn: Graph neural network for 3d object detection in a point cloud," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1711–1719.

[7] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3354–3361.

[8] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2446–2454.

[9] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4490–4499.

[10] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

[11] H. Kuang, B. Wang, J. An, M. Zhang, and Z. Zhang, "Voxel-fpn: Multi-scale voxel feature aggregation for 3d object detection from lidar point clouds," *Sensors*, vol. 20, no. 3, p. 704, 2020.

[12] J. Deng, S. Shi, P. Li, W. Zhou, Y. Zhang, and H. Li, "Voxel r-cnn: Towards high performance voxel-based 3d object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 2, 2021, pp. 1201–1209.

[13] Z. Liu, X. Zhao, T. Huang, R. Hu, Y. Zhou, and X. Bai, "Tanet: Robust 3d object detection from point clouds with triple attention," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 11 677–11 684.

[14] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[15] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 770–779.

[16] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3dssd: Point-based 3d single stage object detector," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 040–11 048.

[17] Z. Li, F. Wang, and N. Wang, "Lidar r-cnn: An efficient and universal 3d object detector," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7546–7555.

[18] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[19] Y. Chen, S. Liu, X. Shen, and J. Jia, "Fast point r-cnn," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9775–9784.

[20] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "Pv-rcnn:

Point-voxel feature set abstraction for 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 529–10 538.

[21] J. Noh, S. Lee, and B. Ham, "Hvpr: Hybrid voxel-point representation for single-stage 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 605–14 614.

[22] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3693–3702.

[23] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *Acm Transactions On Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[24] Y. Li, H. Chen, Z. Cui, R. Timofte, M. Pollefeys, G. S. Chirikjian, and L. Van Gool, "Towards efficient graph convolutional networks for point cloud handling," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 3752–3762.

[25] Q. He, Z. Wang, H. Zeng, Y. Zeng, S. Liu, and B. Zeng, "Svga-net: Sparse voxel-graph attention network for 3d object detection from point clouds," *arXiv preprint arXiv:2006.04043*, 2020.

[26] S. Thakur and J. Peethambaran, "Dynamic edge weights in graph neural networks for 3d object detection," *arXiv preprint arXiv:2009.08253*, 2020.

[27] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3d: A modern library for 3d data processing," *arXiv preprint arXiv:1801.09847*, 2018.

[28] B. Delaunay *et al.*, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[29] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 4, pp. 469–483, 1996.

[30] P. Maur, "Delaunay triangulation in 3d," *Technical Report, Departmen. of Computer Science and Engineering*, 2002.

[31] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 51–55, 2020.

[32] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[33] X. Huang, P. Wang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, "The apolloscape open dataset for autonomous driving and its application," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 10, pp. 2702–2719, 2019.

[34] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska, "One thousand and one hours: Self-driving motion prediction dataset," https://level-5.global/level5/data/, 2020.

[35] J. Geyer, Y. Kassahun, M. Mahmudi, X. Ricou, R. Durgesh, A. S. Chung, L. Hauswald, V. H. Pham, M. Mühlegg, S. Dorn *et al.*, "A2d2: Audi autonomous driving dataset," *arXiv preprint arXiv:2004.06320*, 2020.

[36] 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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[37] A. Ahmadyan, L. Zhang, A. Ablavatski, J. Wei, and M. Grundmann, "Objectron: A large scale dataset of object-centric videos in the wild with pose annotations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7822–7831.

[38] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[39] J. Zhang, H. Liu, and J. Lu, "A semi-supervised 3d object detection method for autonomous driving," *Displays*, vol. 71, p. 102117, 2022.

[40] G. Du, K. Wang, S. Lian, and K. Zhao, "Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review," *Artificial Intelligence Review*, vol. 54, no. 3, pp. 1677–1734, 2021.

[41] M. Ortega, E. Ivorra, A. Juan, P. Venegas, J. Martínez, and M. Alcañiz, "Mantra: An effective system based on augmented reality and infrared thermography for industrial maintenance," *Applied Sciences*, vol. 11, no. 1, p. 385, 2021.

[42] J. Sun, Y.-M. Ji, F. Wu, C. Zhang, and Y. Sun, "Semantic-aware 3d-voxel centernet for point cloud object detection," *Computers & Electrical Engineering*, vol. 98, p. 107677, 2022.

[43] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 7652–7660.

[44] S. Casas, W. Luo, and R. Urtasun, "Intentnet: Learning to predict intention from raw sensor data," in *Conference on Robot Learning*. PMLR, 2018, pp. 947–956.

[45] A. Sagar, "Aa3dnet: Attention augmented real time 3d object detection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 628–635.

[46] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[47] J. Kim and J. Cho, "Delaunay triangulation-based spatial clustering technique for enhanced adjacent boundary detection and segmentation of lidar 3d point clouds," *Sensors*, vol. 19, no. 18, p. 3926, 2019.

[48] H. Wang, Y. Zhang, W. Liu, X. Gu, X. Jing, and Z. Liu, "A novel gcn-based point cloud classification model robust to pose variances," *Pattern Recognition*, vol. 121, p. 108251, 2022.

[49] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[50] P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73 – 101, 1964.

[51] WeijingShi, "Point-gnn," GitHub, (accessed on 2021-12-20). [Online]. Available: https://github.com/WeijingShi/Point-GNN

[52] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

[53] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones,

R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[55] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[56] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, G. Ranzuglia *et al.*, "Meshlab: an open-source mesh processing tool." in *Eurographics Italian chapter conference*, vol. 2008. Salerno, Italy, 2008, pp. 129–136.

[57] jijup, "Shapehull3d," GitHub, (accessed on 2022-04-15). [Online]. Available: https://github.com/jijup/Shapehull3D/

[58] 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)*. IEEE, 2018, pp. 1–8. [Online]. Available: https://github.com/asharakeh/kitti_native_evaluation/tree/f2f70059e643556bb825166fdc4dfaf7d1a798a0

[59] J. Hui, "map (mean average precision) for object detection," Mar 2018, (accessed on 2022-04-16). [Online]. Available: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

[60] yeyang1021, "Kitti_viz_3d," GitHub, (accessed on 2022-04-10). [Online]. Available: https://github.com/yeyang1021/KITTI_VIZ_3D