3D Object Detection and Tracking using Graph Neural Networks on Waymo Dataset

Ahmed Bahnasy
Technical University in Munich
ahmed.bahnasy@tum.de

Abstract

Recent approaches for 3D object detection have made a tremendous progress due to the development of deep learning. this also highly influences the progress in other tasks like 3D Multi-object tracking (MOT) which is an indispensable component to any autonomous system. Recent work uses a standard tracking- by-detection pipeline, where feature extraction is first performed independently for each object in order to compute an affinity matrix. Then the affinity matrix is passed to the Hungarian algorithm for data association. This matching can be seen as discrete optimisation process and there is no learning it. A key factor of this standard pipeline is to learn discriminative features for different objects, by relying on a robust detector in order to reduce confusion during data association. In this task, we experiment with improving the discriminative feature learning for MOT: by first build a robust object detector and second instead of obtaining features for each object independently, we try applying learning module for tracking by introducing the Graph Neural Network which enables feature interaction mechanism. As a result, the feature of one object is informed of the features of other objects so that the object feature can lean towards the object with similar feature and deviate from objects with dissimilar features, leading to a more discriminative feature for each object.

1. Introduction

Robust 3D perception becomes an essential component in any state-of-the-art autonomous system. 3D detection has a series of challenges: First, point-clouds are sparse, the points are unevenly distributed across the 3D space, and most of the 3D space are without measurements. Second, 3D objects have a wide range of sizes, shapes and aspect ratios, for example, in autonomous driving domain, buses, trucks and cars are elongated, pedestrians are tall and cyclists are nearly planer. These challenges makes applying ideas from 2D domain to 3D domain not straight forward. Trying to use the anchor based approach followed in 2D domain by assigning a different anchor for each object orientation increases the computational cost and may introduce a large number of of potential false-positive detections. The main challenge for linking up 2D and 3D domains lies in the way the objects are represented [43]. Representing objects as points greatly simplifies 3D recognition [45]. In this task, we tried to build a 3D that detects the centers of the objects and their properties using a key-point detector [47]. In specific we followed the same approach in [45, 9] by using any standard Lidar-based backbone network, for example, PointPillars or VoxelNet, to build an intermediate representation for the input point cloud. This representation is then flattened into BEV map using a standard image-based key-point detector to find the object centers [47]. The rest of the 3D bounding box parameters, i.e., 3D Size, orientation, and velocity, are then regressed from the center location feature. according to [45], center-based representation has several advantages: First, it reduces the detector search space since points has no orientation, unlike bounding boxes. Second, center-based representation simplifies the tracking downstream task.

Much like 3D perception, Multi-object tracking (MOT) is a crucial component for autonomous driving [18]. Tracking-by-detection paradigm is the dominant approach when it comes to online MOT problems [3, 36]. The main idea is to apply object detector to all frames and extract features independently from each detected object. Then pairwise similarity is computed between objects and a Hungarian algorithm [16] is used to solve the problem. The key idea in this approach is to learn discriminative features for the objects with different identities to reduce the confusion in the matching. One key observation is that feature extraction is done independently and there is no interaction between features during feature extraction.According to [37], independent feature extraction is sub-optimal for discriminative feature learning. This idea is very important for MOT, given the fact that an object in the current frame can be matched to at most one object in the previous frame. So,

if the similarity between two features increased, then the pairwise similarity between the rest of the objects and any of these two objects should be decreased to avoid confusion for matching. Based on that observation, we followed [37] by implementing a Graph Neural Network to do the feature interaction. The idea is to construct a graph with each node being the object feature. Then, each node can update its feature by aggregating features from other nodes during layer propagation step. the object feature is now not isolated and can be adapted with respect to other features.

We tested the trained models on Waymo dataset [30]. For the detection part, among the different experiments, we show that voxelization approach is still the best approach to process sparse LiDAR point clouds. Center-based 3D box representation is better than anchor based 3D box representation by 2 mAP. Our best trained achieves 0.71 mAP for vehicle and pedestrian and 0.59 for cyclists classes. For the tracking part, our best graph neural network trained model achieves 0.45 MOTA for Vehicle, 0.41 MOTA for Pedestrian class and 0.38 for Cyclist class, which is unfortunately slightly below the baseline model (0.59 MOTA), but we believe that GNNs are more capable of achieving more than. Due to time limit and some technical difficulties in the implementation of training and validation pipelines for tracking, we couldn't scale the GNN training to utilise best practices used in training deep learning models like increasing batch size and make use of Batch Normalization techniques and parallel training and inference. Another important note here is that the beforementioned results for tracking are calculated based on a subset of the validation data, evaluating the whole validation set takes too long as we've to process the frames in a sequential fashion, frame by frame.

2. Related Work

3D Object Detection is all about predicting the three dimensional rotated bounding boxes [10, 17, 23, 41, 43, 44]. Most 3D-based methods either use point cloud data directly or require converting these data into 3D grids or voxels instead of generating BEV representations. In [33], point cloud data are converted into voxels containing feature vectors, and then a novel convolution-like voting-based algorithm is used for detection. Vote3Deep [6] leverages feature voting in [33] to efficiently process the sparse 3D point-cloud in equally spaced 3D voxels. VoxelNet [49] uses a PointNet [24] inside each voxel to generate a unified feature representation from which a head with 3D sparse convolutions [12] and 2D convolutions produces detections. SECOND [41] simplifies the VoxelNet and speeds up sparse 3D convolutions. PIXOR [42] project all points onto a 2D feature map with 3D occupancy and point intensity information to remove the expensive 3D convolutions. PointPillars[17] replaces all voxel computation with a pillar representation, a single tall elongated voxel per map location, improving backbone efficiency. MVF [48] and Pillarod [35] combine multiple view features to learn a more effective pillar representation. In this task we followed [45, 9] by focusing on the output representation and employ any 3D encoder to get the representation.

Online Multi-Object Tracking is mostly addressed using Tracking by Detection paradigm. The performance is highly impacted by two factors: object detection quality and discriminative feature learning. After the affinity matrix is computed based on the pairwise similarity of learned discriminative feature, the problem of online MOT could be addressed as as a discrete optimization problem and could be solved using as bipartite matching problem using the Hungarian algorithm [16]. To obtain discriminative feature, prior work mostly focuses on the feature selection. Among different features, it turns out that motion and appearance are the most discriminative features. Early work employs hand-crafted features such as spatial distance [21] and Intersection of Union (IoU) [4] as the motion feature, and use color histograms as the appearance feature. Recently, Convolutional Neural Networks have been used to extract appearance feature [1, 7]. Regarding the motion feature, many filter based approaches has been used [3, 36]. Deep learning approaches has been introduced for motion feature in [1]. we followed the same idea in [38, 37] by exploring both the appearance and motion feature with focus on the 3D space.

Graph Neural Networks is used as way to improve discriminative feature learning for MOT after showing promising performance in many fields [13, 32, 15, 2, 19, 46]. GNNs was first proposed by [11] to directly process graph-structured data using neural networks. The major component of the GNNs is the node feature aggregation technique, with which node can update its feature by interacting with other nodes. With this technique, significant success has been achieved in many fields using GNNs such as semantic segmentation[5], action recognition [26], single object tracking [8], person re-identification [39], point cloud classification and segmentation [**?**]. The majority of the work deals with object features as independent and isolated from other features. By leveraging node aggregation technique of the GNNs, object features could be iteratively evolved so that the feature of different objects can be more discriminative. The idea of feature interaction was first introduced in [14] to encode context information for object detection in the spatial domain, Although a temporal relation network is proposed in the follow-up work [40], the feature of a tracked object is only aggregating from its past trajectory and no interaction with other object features exist. In This task we followed [45] by implementing a generic feature interaction framework that can model any kind of interaction in

both spatial and temporal domains.

3. Network Architecture

During the early stages of this task, we tried to use Votenet [22] as a 3D object detector. Despite the promising results of this architecture on Indoor datasets, it performs very poorly on outdoor datasets like KITTI and Waymo. More information about these experiments are stated in Appendix A

3.1. Detection

Point Cloud Encoder. We followed the same approach of up to date SOTA detectors [27, 34, 45] on Waymo dataset [30] by, first, employing a 3D encoder that quantizes the point-cloud into regular bins. A point-based network then extracts feature for all points inside a bin. The 3D encoder then pools these features into its primary feature representation. Second, the output of the encoder is processed by a backbone network. The output of a backbone network is a map-view feature-map $\mathbf{M} \in \mathbb{R}^{W \times L \times F}$ of width W and length L with F channels in a map-view reference frame. Both width and height directly relate to the resolution of individual voxel bins and the backbone network's stride. Common Common backbones used for this task are VoxelNet [41, 42] and PointPillars [17]. A number of separate heads modules take the feature-map $\mathbf{M}$ and predict a class-specific heat-map, object size, a sub-voxel location refinement, rotation, and velocity. All outputs are dense predictions. Figure 1 shows the overall Detection Network.

Center heatmap head. This head produces a heatmap peak at the center location of any detected object. This head produces a K-channel heatmap $\hat{Y}$, one channel for each of K classes. During training, it targets a 2D Gaussian produced by the projection of 3D centers of annotated bounding boxes into the map-view. Following [47], a focal loss is used.

Regression heads. Several object properties are stored at the center-features of the objects: sub-voxel refinement $o \in \mathbb{R}^2$, height above the ground $h_g \in \mathbb{R}$, the 3D size $s\ in \mathbb{R}^3$, and a heading rotation angle $(\sin(\alpha), \cos(\alpha)) \in \mathbb{R}^2$. Sub-voxel refinement is meant to reduce the quantization error. Heading prediction uses the sine and cosine of the heading angle as a continuous regression target.

3.2. Tracking

Given M tracked objects $o_i \in O$ at frame t where $i \in 1, 2, ..., M$ and N detected objects $d_j \in D$ in frame $t+1$ where $j \in 1, 2, ..., N$, the idea of MOT is all about associating existing tracked objects M from previous frame t with new detected objects N in the current frame $t+1$ by learning discriminative feature from O and D and then find the correct matching based on the pairwise feature similarity.

Following [37], we implemented a 3D appearance extractor and 3D motion extractor which are both applied to all objects in O and D and then the concatenated together. A graph neural network is then implemented that takes as input the concatenated object and construct a graph with node being the object feature in frame t and $t+1$. Then, the graph neural network iteratively aggregates the node feature from the neighbourhood and computes the affinity matrix for matching using edge regression. Following the [28], the 3D detection is parametrized as a tuple of $d = x, y, z, l, w, h, \theta$ where (x, y, z) denotes the object center in 3D space, (l, w, h) denotes the object size and θ is the heading angle. The tracked objects O use the same parametrization alongside an additional ID. the feature extractors are depicted in Figure 2.

3.2.1 Feature Extractor

For appearance feature extractor, LiDAR point cloud as used as the appearance cue. the point cloud enclosed by the 3D detection box is extracted and then PointNet [24] is applied to obtain the feature. The same appearance feature extractor is applied on both the tracked and the detected objects. For the motion feature extractor, different methods are used. for tracked objects, they have temporal data in form of an associated trajectory in the past frames. while detected objects do not have. for Tracked objects o_i, an LSTM is used to take as input the objectś detections in the past T frames to obtain the feature. For detected object d_j, Two-Layer MLP (Multi-Layer-perceptron) is used that takes the detection in frame $t+1$ as input to extract the feature. The final feature for tracked and detected objects is obtained by concatenating the 3D motion and appearance features. The final motion and appearance vectors are adjusted to have the same dimensionality to ensure fair contribution for both features in the final representation of the object.

3.2.2 Graph Neural Network for Data Association

Graph Construction. After feature extraction, we end up with M features for tracked objects in frame t and N features for detected objects in frame t+1. A graph, of total nodes N+M, is constructed with each node being an object feature. an important step is to define the neighbourhood for every node in the graph. A naiive approach is to make it fully connected. The downside of this approach is the computational complexity. Based on the prior knowledge about online MOT, where the matching should be done across frames and the possible matching should be done with only nearby frames, the edges are constructed between pairs of

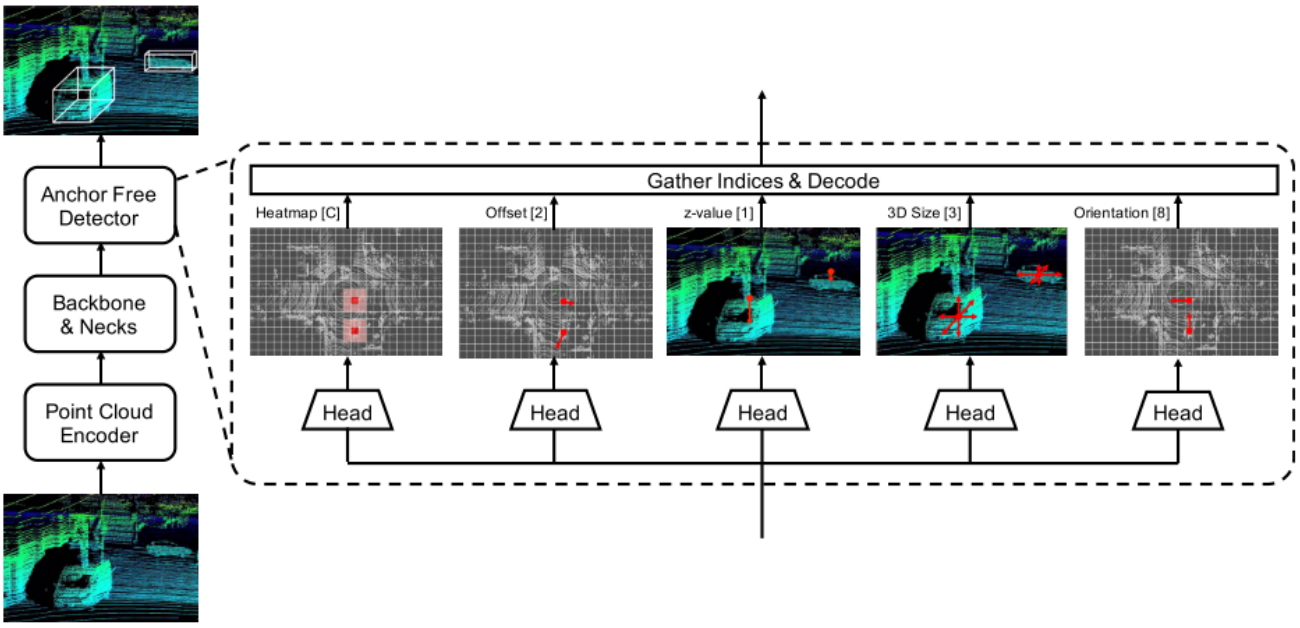

Figure 1. Overall Architecture of the Detection Network. source [9]

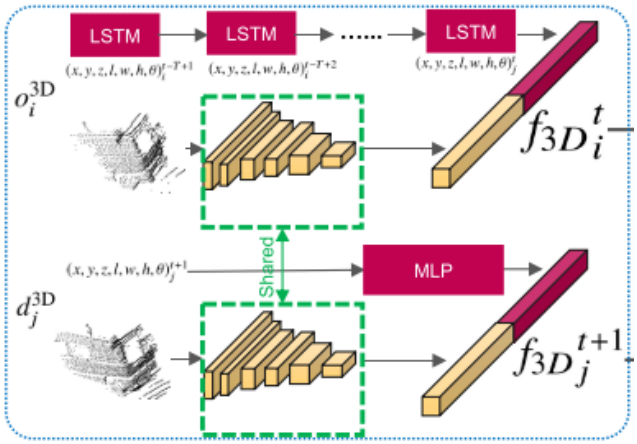

Figure 2. source [37]

nodes in different frames and only if the distance between their detection centers is less than $Dist_{max}$.

Affinity Matrix using Edge Regression. $N \times M$ matrix is needed for solving the online MOT. The affinity matrix is constructed based on the pairwise similarity of the features extracted from M tracked objects and N detected objects. This process is called Edge Regression in context of GNN. Conventional metrics used in MOT community are cosine similarity and negative L2 distance between features pairs. Following [37], we implemented a learnable regression module by employing a two-layer MLP that takes the difference of two node features as input and outpus a scaler value between 0 and 1 as the pairwise similarity score:

$$A_{ij} = Sigmoid(\sigma_2(ReLU(\sigma_1(n_i^t - n_j^{t+1})))) \quad (1)$$

where σ_1 and σ_2 are two different linear layers. In addition, n_i^t and n_j^{t+1} are two node features in different frames where $j, i \in 1, 2, , M, j \in 1, 2, , N$

Node feature Aggregation. The heart of GNN is aggregation step which model the feature interaction. it iteratively update the node feature by aggregating features from the node neighbourhood in every layer of the GNN. Among the vast aggregation rules in the literature, we choose four rules to analyze their impact on the performance of MOT.

The first type is applying a general message passing scheme, the second type is using GraphConv operator [2], the third type is using EdgeConv operator [**?**] and the fourth type is using Graph attention mechanism []

$$n_i^{'t} = \sum_{j \in \mathcal{N}(i)} \sigma(ReLU(n_i^{t+1})) \quad (2)$$

$$n_i^{'t} = \sigma_4(ReLU(n_i^t)) + \sum_{j \in \mathcal{N}(i)} \sigma(ReLU(n_i^{t+1})) \quad (3)$$

$$n_i^{'t} = \sigma_4(ReLU(n_i^t)) + \sum_{j \in \mathcal{N}(i)} \sigma(ReLU(n_i^{t+1} - n_i^t)) \quad (4)$$

$$n_i^{'t} = \sigma_4(ReLU(n_i^t)) + \sum_{j \in \mathcal{N}(i)} \sigma(ReLU(A_{ij}(n_i^{t+1} - n_i^t))) \quad (5)$$

where $\mathcal{N}(i)$ is a set of neighbourhood nodes in frame $t+1$ with respect to the node i in frame t. σ_3, σ_4 are linear layers which have different weights across layers of the GNN. The weight A_{ij} is obtained from the affinity matrix of the current layer.
In Eq. 2, node feature us update by aggregating feature from only the neighbourhood nodes, the node feature itself is lost after aggregation. In Eq. 3, the previous limitation is addressed by adding self loops in the graph to consider node feature during aggregation. In Eq 4, feature from neighbourhood is replaced with the difference of the features between the node itself and the neighbourhood node. In Eq. 5, attention weight is added from the affinity matrix to the feature aggregation so that the network can focus on the neighbourhood with higher affinity score.

3.2.3 Losses

Following [37], two losses are implemented and used in all K graph layers during the training. The first loss is the batch triplet loss L_{tri}, and the second loss is affinity Loss L_{aff}. The entire loss function L:

$$L = \sum_{K=0}^{K-1} (L_{tri}^{k} + L_{aff}^{k}) \tag{6}$$

Bath Triplet Loss. Applying Batch Triplet Loss to node feature in every layer of the GNN is needed for learning discriminative features for matching. for node n_i^t that has a matched node in n_j^{t+1}, the batch triplet loss in each layer is:

$$L_{tri} = \max(\left\|n_i^t - n_j^{t+1}\right\| - \min_{\substack{d_s \in D \\ id_i \neq id_s}} \left\|n_i^t - n_j^{t+1}\right\| - \min_{\substack{o_r \in O \\ id_r \neq id_j}} \left\|n_i^t - n_j^{t+1}\right\| + \alpha, 0) \tag{7}$$

where α is the margin of the triplet loss. n_s^{t+1} is a node in frame $t+1$ that has a different ID from node n_j^{t+1} and n_i^t. Similarly, n_i^t and n_j^{t+1}. In the case that n_i^t does not have a matched node in frame $t+1$ with the same ID, the first is deleted for the positive pair and only the remaining two negative terms in the loss are considered.

Affinity Loss. Additional affinity loss L_{aff} is applied to supervise the final output of the network, i.e., the predicted affinity matrix A. The affinity loss consists of two parts and represented as follows:

$$L_{aff} = L_{bce} + L_{ce} \tag{8}$$

First, since the ground truth affinity matrix A^g can only have integer 0 or 1 in all the entries, the prediction can be formulated as a classification problem. The first part could be represented as a binary cross entropy loss L_{bce} that is applied on each entry of the predicted affinity matrix A.

$$L_{bce} = \frac{-1}{MN} \sum_{i}^{M} \sum_{j}^{N} (A_{ij}^{g} \log A_{ij} + (1 - A_{ij}^{g}) \log(1 - A_{ij})) \tag{9}$$

Second, each tracked object o_i^t in frame t can only have either one matched detection d_j^{t+1} or no match at all. Thus, each row and column of the A^g can only be a one-hot vector or an all-zero vector. For all rows and columns that have a none-hot vector in A^g, cross entropy loss L_{ce} is applied. For example, the column A_j^g in ground truth affinity matrix is a one-hot vector and the loss L_{ce} for the jth column is:

$$L_{ce} = \frac{-1}{M} \sum_{i}^{M} A_{ij}^{g} \log(\frac{\exp A_{ij}}{\sum_{i}^{M} \exp A_{ij}}) \tag{10}$$

4. Experiments

4.1. Waymo Dataset.

Waymo Open Dataset [] contains 798 training sequences and 202 validation sequences. it has 5 classes, among them 3 dynamic (moving) classes (Vehicles, Pedesterians, Cyclists) which we consider in our experiments. The point clouds are captured with a 64 lanes LiDAR, which produces about 180k LiDAR points every 0.1s. The official 3D detection evaluation metrics include the standard 3D bounding box mean average precision (mAP) and mAP weighted by heading accuracy (mAPH). we present our results based on mAP only. The mAP is based on an IoU threshold of 0.7 for vehicles and 0.5 for pedestrians. For 3D tracking, the official metrics are Multiple Object Tracking Accuracy (MOTA) and Multiple Object Tracking Precision (MOTP). We present our results based on MOTA only. The official evaluation toolkit also provides a performance breakdown for two difficulty levels: LEVEL 1 for boxes with more than five Lidar points, and LEVEL 2 for boxes with at least one LiDAR point.

4.2. Data Augmentation

For data augmentation, we use random flipping along both X and Y axis, and global scaling with a random factor from [0.95, 1.05]. We use a random global rotation between $[-\pi/4, \pi/4]$. We also use the ground-truth sampling [41] to deal with class imbalance problem, which copies and pastes points inside an annotated box from one frame to another frame. We train the model with batch size of 4 for 20 epochs on GTX1070 ti GPU.

4.3. Implementation Details

Detector. The implemented detection model uses a detection range of [−75.2m, 75.2m] for the X and Y axis, and

[−2m, 4m] for the Z axis. Following [**?**], Voxel size is (0.1m, 0.1m, 0.15m). We followed [41, 45] by replacing the expensive 3D convolutional middle layers by sparse convolutions [12]. All heads outputs share a first 3×3 convolutional layer, Batch Normalization, and ReLU. Then, each output uses its own branch of two 3×3 convolutions separated by a batch norm and ReLU.

For the optimizer, we used Adam with the default parameters $\beta_1 = 0.9$, $\beta_2 = 0.99$ and learning rate 0.03. We used One Cycle Policy [29] for training to speed up the convergence of the model. The learning rate is chosen based on the method described in [29].

Tracker. Both appearance extractor and motion extractor feature vector dimensionality is set to 128. For appearance branch, PointNet is used with six 1D Convolutional layers that maps the input point cloud with size of $N \times 5$ to $N \times 128$. Then a max pooling operation is applied along the points features to obtain the final appearance feature with dimensionality of 128. for motion feature, two layer LSTM with a hidden size of 128 and number of past frames T=5 is used for tracked objects. For detected objects, two-layer MLP (7, 64,128) is used. a max distance $Dist_{max} = 5$ is set to define the neighbourhood for constructing the graph. four graph layers, each with the same dimensionality of 256 are used. For edge regression, two-layer MLP with hidden feature dimension (256,64,1) is used. the margin value α for the batch triplet loss is set to 10.

Tracking Baseline. A velocity-based closest distance matching, which doesn't require separate model motion, is used as a baseline. it is based on the velocity predictions of the detected objects. it predicts the difference in object position between the current and the past frame. At inference time, the object centers in the current frame are projected back to the previous frame by applying the negative velocity estimate and then matching to the tracked objects is done in a greedy way by closest distance matching.

4.4. Main Results

We show per class comparisons with the state-of-the-art methods in table 1. The Detector results are shown in the Fig /reffig:predictions

4.5. Ablation Study

Augmentations. We show the impact of some augmentations in 2

Feature Extractor Table 3 shows the effect of each individual feature extractor and the combination between them. it is obvious from the results that combining both improves the performance.

Model	mAP ↑
HorizonLiDAR3D [34]	0.83
CenterPoint [45]	0.78
PV-RCNN [27]	0.78
Ours	**0.709**
SECOND [41]	0.34
StarNet [20]	0.44

Table 1. Comparison with some state-of-the-art. the table shows mAP evaluation metric

Method	mAP ↑
Baseline	0.51
+ Filter Empty Boxes	0.57
+ GT Sampler	0.66
+ Rotation	0.70

Table 2. Ablation studies for Detection

Extractor	MOTA ↑
A	0.42
M	0.44
A+M	0.45

Table 3. Effect of feature extractors: A (appearance), M (motion)

Node Aggregation	MOTA ↑
Mean aggregator	0.35
GCN [15]	0.43
SAGEConv [13]	0.42
GATConv [32]	0.44
AGNNConv [31]	0.44
EDGEConv [**?**]	0.44
Affinity Attention	0.45

Table 4. Effect of different Node aggregators

Number of layers Figure 4 shows the effect of number of GNN Layers on the MOTA score.

Node Aggregation. Table 4 shows the effect of different node aggregators

References

[1] Erkan Baser, Venkateshwaran Balasubramanian, Prarthana Bhattacharyya, and Krzysztof Czarnecki. Fantrack: 3d multi-object tracking with feature association network. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1426–1433. IEEE, 2019.

[2] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[3] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016*

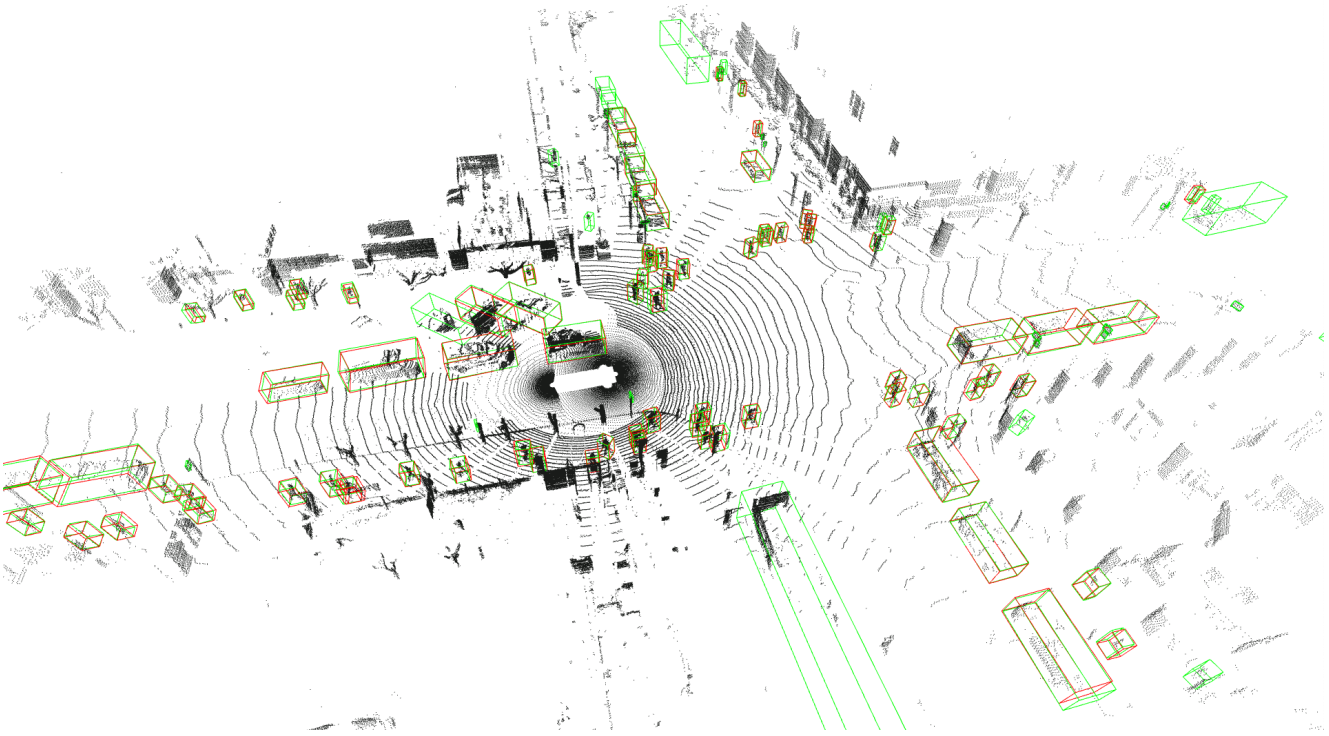

Figure 3. Results of the Detector (Red: predictions, Green: Ground Truth)

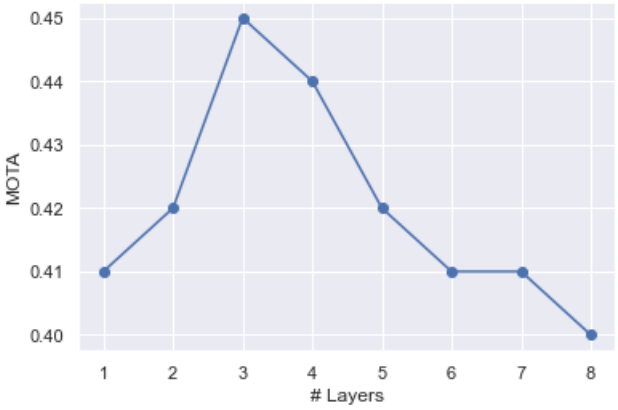

Figure 4. Effect of GNN Layers on MOTA score

IEEE international conference on image processing (ICIP), pages 3464–3468. IEEE, 2016.

[4] Erik Bochinski, Volker Eiselein, and Thomas Sikora. High-speed tracking-by-detection without using image information. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2017.

[5] Yunpeng Chen, Marcus Rohrbach, Zhicheng Yan, Yan Shuicheng, Jiashi Feng, and Yannis Kalantidis. Graph-based global reasoning networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 433–442, 2019.

[6] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1355–1361. IEEE, 2017.

[7] Davi Frossard and Raquel Urtasun. End-to-end learning of multi-sensor 3d tracking by detection. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 635–642. IEEE, 2018.

[8] Junyu Gao, Tianzhu Zhang, and Changsheng Xu. Graph convolutional tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4649–4659, 2019.

[9] Runzhou Ge, Zhuangzhuang Ding, Yihan Hu, Yu Wang, Sijia Chen, Li Huang, and Yuan Li. Afdet: Anchor free one stage 3d object detection. *arXiv preprint arXiv:2006.12671*, 2020.

[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[11] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[12] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the*

IEEE conference on computer vision and pattern recognition, pages 9224–9232, 2018.

[13] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

[14] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3588–3597, 2018.

[15] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[16] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[17] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019.

[18] 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 *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 3569–3577, 2018.

[19] Federico Monti, Michael M Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. *arXiv preprint arXiv:1704.06803*, 2017.

[20] Jiquan Ngiam, Benjamin Caine, Wei Han, Brandon Yang, Yuning Chai, Pei Sun, Yin Zhou, Xi Yi, Ouais Alsharif, Patrick Nguyen, et al. Starnet: Targeted computation for object detection in point clouds. *arXiv preprint arXiv:1908.11069*, 2019.

[21] Hamed Pirsiavash, Deva Ramanan, and Charless C Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *CVPR 2011*, pages 1201–1208. IEEE, 2011.

[22] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[23] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 918–927, 2018.

[24] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas 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*, pages 652–660, 2017.

[25] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[26] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with directed graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7912–7921, 2019.

[27] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng 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*, pages 10529–10538, 2020.

[28] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 770–779, 2019.

[29] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

[30] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin 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*, pages 2446–2454, 2020.

[31] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[33] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems*, volume 1, pages 10–15607. Rome, Italy, 2015.

[34] Yu Wang, Sijia Chen, Li Huang, Runzhou Ge, Yihan Hu, Zhuangzhuang Ding, and Jie Liao. 1st place solutions for waymo open dataset challenges–2d and 3d tracking. *arXiv preprint arXiv:2006.15506*, 2020.

[35] Yue Wang, Alireza Fathi, Abhijit Kundu, David Ross, Caroline Pantofaru, Tom Funkhouser, and Justin Solomon. Pillar-based object detection for autonomous driving. *arXiv preprint arXiv:2007.10323*, 2020.

[36] Xinshuo Weng and Kris Kitani. A baseline for 3d multi-object tracking. *arXiv preprint arXiv:1907.03961*, 2019.

[37] Xinshuo Weng, Yongxin Wang, Yunze Man, and Kris M Kitani. Gnn3dmot: Graph neural network for 3d multi-object tracking with 2d-3d multi-feature learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6499–6508, 2020.

[38] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[39] Jinlin Wu, Yang Yang, Hao Liu, Shengcai Liao, Zhen Lei, and Stan Z Li. Unsupervised graph association for person re-identification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8321–8330, 2019.

[40] Jiarui Xu, Yue Cao, Zheng Zhang, and Han Hu. Spatial-temporal relation networks for multi-object tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3988–3998, 2019.

[41] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018.

[42] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[43] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3dssd: Point-based 3d single stage object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11040–11048, 2020.

[44] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Std: Sparse-to-dense 3d object detector for point cloud. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1951–1960, 2019.

[45] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3d object detection and tracking. *arXiv preprint arXiv:2006.11275*, 2020.

[46] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[47] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

[48] Yin Zhou, Pei Sun, Yu Zhang, Dragomir Anguelov, Jiyang Gao, Tom Ouyang, James Guo, Jiquan Ngiam, and Vijay Vasudevan. End-to-end multi-view fusion for 3d object detection in lidar point clouds. In *Conference on Robot Learning*, pages 923–932. PMLR, 2020.

[49] Yin Zhou and Oncel 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*, pages 4490–4499, 2018.

Appendices

A. Votenet for Lidar Outdoor Scenes

A.1 Data preparation and Training

VoteNet consists of two modules: the proposal module which consumes the raw point cloud and produces the virtual voting points; and the object proposal and classification module that operates on the voting points to propose and classify objects. Up to now, VoteNet has only been tuned and tested on RGB-D data (ScanNet and SUN RGB-D). in order to adapt the model for Waymo, we implemented a custom input pipeline to optimally preprocess and feed the Waymo point cloud data into the VoteNet network. Following the same train/validation/test split from Waymo paper [30], and following the same training procedure mentioned in [22], For each scene we randomly subsample 20k points. The scenes are furthermore augmented on-the-fly with random flips in horizontal plane, random uniform rotations along up-axis in $\pm 30^\circ$ range, and random uniform scaling of $\pm 10\%$. In addition to three Euclidean coordinates for each point, we also include up to two additional features: the provided laser reflection intensities, and a height estimate for each point. The latter is estimated as a 1% percentile of all point positions along the up-axis. We perform training on the dynamic classes of the Waymo dataset: car, pedestrian, and cyclist. We choose one size template per class, and 12 bins for the heading angle. We train with a batch size of 16 and incorporate a scheduled learning policy with a starting LR of 0.001 and LR-decays by 0.1 at 60, 90, and 120 epochs, as well as an exponential BN momentum decay. We use same loss function as in SUN RGB-D case.

Points clouds from outdoor LiDAR scans are quite different from point clouds from RGB-D indoor scans. For example, the typical scales of the Waymo scenes are significantly larger than those of indoor scans, with depth fields reaching beyond 70 meters along forward-axis. the LiDAR point cloud has strongly varying density and is generally more sparse than the point clouds produced from the RGB-D imagery. Point features extracted from sparse regions may generalize poorly to dense regions, and vice versa

to further adapt VoteNet to the distinct characteristics of the outdoor LiDAR scenes. we adapt the receptive field radii and increase the number of clusters for feature aggregation. to capture fine details of point cloud and, at the same time, mitigate the corruption of local patterns due to sampling deficiency in sparse cloud regions, we enhance the set-abstraction modules of the backbone network with multi-scale grouping (MSG) layers [25], which concatenate features at different scales before feeding them into the feature aggregation layer.

Among all the experiments to find the best backbone network parameters, we ended up following the configuration in [28] by configuring four MSG-based SA layers used to subsample the inputs to 4096, 1024, 256 and 64 points, and two FP layers to upsample the points back to 1024 points, each with additional 512 deep features. Tables A.1 and A.1 illustrates the difference between the two backbones.

A.2 Results on 360° Point cloud

When working the 360 point cloud, the Average Precision results from different VoteNet architectures were very poor. after training for nearly 120 epochs, mAP over the test set was nearly 0.15 only. the problem lies in the different points densities between outdoor LiDAR point clouds and indoor RGB-D point cloud. there are a lot of objects which have only 2 10 points per bounding box. after the initial downsampling from 120k to 20k points for processing the further downsampling done by Set abstraction modules, the final proposal points are actually lie outside most of the ground truth bounding boxes as shown in Figure 5. Thus, the network was not able to learn the correct voting. nearly all of the detected objects in this experiment were objects near to

Module (Input)	Output dimensions	clusters K	Receptive field radius	MLP layer dimensions
SA1 (PC)	(4096, 3 + 96)	2048	0.2	64/64/128
SA2 (SA1)	(1024, 3 + 256)	1024	0.4	128/128/256
SA3 (SA2)	(512, 3 + 512)	512	0.8	128/128/256
SA4 (SA3)	(64, 3 + 512)	256	1.2	128/128/256
FC1 (SA3, SA4)	(512, 3 + 512)	-	-	512/512
FC2 (SA2, SA3)	(1024, 3 + 512)	-	-	512/512

Table 5. Original Architecture of VPN in [22]

Module (Input)	Output dimensions	clusters K	MSG Radii	MLP layers
SA1 (PC)	(4096, 3 + 96)	2048	0.1, 0.5	16/16/32, 32/32/64
SA2 (SA1)	(1024, 3 + 256)	1024	0.1, 0.5	64/64/128, 64/96/128
SA3 (SA2)	(512, 3 + 512)	512	0.1, 0.5	128/196/256, 128/196/256
SA4 (SA3)	(64, 3 + 512)	256	0.1, 0.5	256/256/512, 256/256/512
FC1 (SA3, SA4)	(512, 3 + 512)	-	-	512/512
FC2 (SA2, SA3)	(1024, 3 + 512)	-	-	512/512

Table 6. Enhanced architecture of VPN with two MSG radii of receptive fields $r_{1,2}$ (meters), and MLP parameters for each MSG group implemented [28]

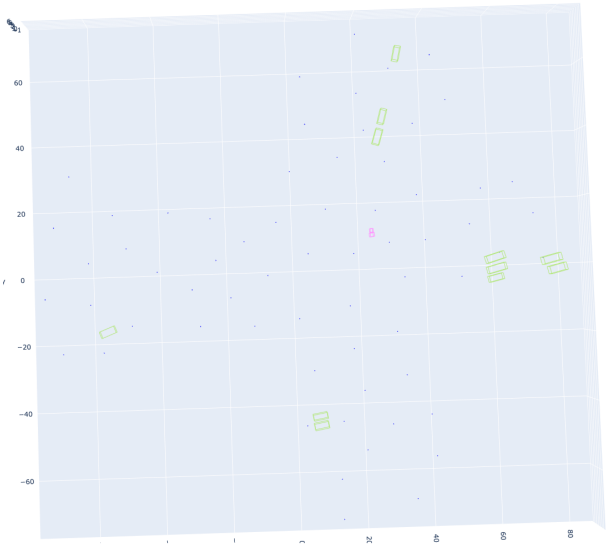

Figure 5. Top view of final 3D space. Blue Points are the final proposal points from VoteNet. Green and Magenta boxes are Ground truth bounding boxes. During training, the final proposal are all lie outside the ground truth bounding boxes. Hence, the voting module cannot learn the correct modules. (Best viewed on screen)

the sensor with ¿= 1000 points, all objects with fewer points were missed.

A.3 Results on Front Camera Frustum Point Cloud

To verify the results of the previous section, we did an extensive exploration in the literature about the experiments involves raw point processing approaches [24, 25, 22, 23, 28] for outdoor datasets (Waymo, KITTI, A2D2). interesting work in [28] in foreground background segmentation caught our attention. as a first step, we reproduced the work in [28] on KITTI dataset. Then, Before reproducing the same experiment on Waymo citesun2020scalability, we implemented a specific data processing pipeline to make Waymo point cloud identical to KITTI. As KITTI includes annotations only for the objects in teh front camera frustum. we first project the 360-degree LiDAR data onto the image plane and extract the point cloud that lie inside the result- ing frustum. This step reduces the average number on points per scene from ∼170k to 16,384. For each scene we randomly subsample 16,384 points, and for scenes with fewer points, we randomly copy the points up to a total of 16,384.

Reducing the size of the point cloud make a tremendous boost in the (AP) results of VoteNet when we re-executed the experiments on this data setup. Due to the small initial input size, the ratio between points inside bounding boxes to the points outside bounding boxes are very large when compared in case of using the 360 point cloud. many of the final proposal points are found inside the ground truth bounding boxes. Thus the voting module was able to learn the correct voting. Figure 6 shows the final results of running two VoteNet architectures on the reduced Waymo point clouds.

B. Sparse Convolutions Libraries

As discussed in section 1, one of the the main modules in the Detection Network is the convolutional Middle Layers which takes the Voxel feature as input and outputs the middle representation which will be further processed using separate head for regressing different outputs. In our

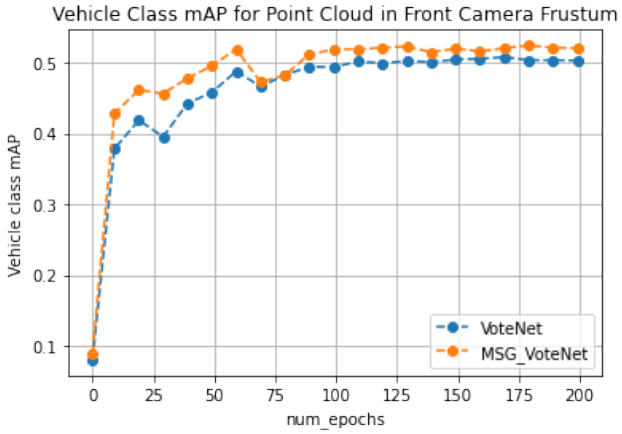

Figure 6. mAP on Waymo Validation set for Original VoteNet architecture and tuned VoteNet architecture

# parameters	Forward + Loss Time		Backward Time	
2695312	690 ms	770 ms	600 ms	700 ms
2694352	200ms	300ms	300ms	400ms

Table 7. Comparison between different sparse convolution libraries

implementation, we relied on sparse convolutions instead of 3D convolutions due to its high computational time and high memory consumption. we compared two different open source libraries for sparse convolutions: SpConv [1] and MinkowskiEngine [2]. By trying to implement the same backbone design in the two libraries, MinkowskiEngine showed a huge advantage in terms of computational time. Table 7 shows a comparison between both libraries.

[1] https://github.com/traveller59/spconv
[2] https://github.com/NVIDIA/MinkowskiEngine