GROUP 08: Report Project 2 Problem 1

Yannick Neuffer
ETH Zürich

Thorbjörn Höllwarth
ETH Zürich

Lukas Nüesch
ETH Zürich

1.

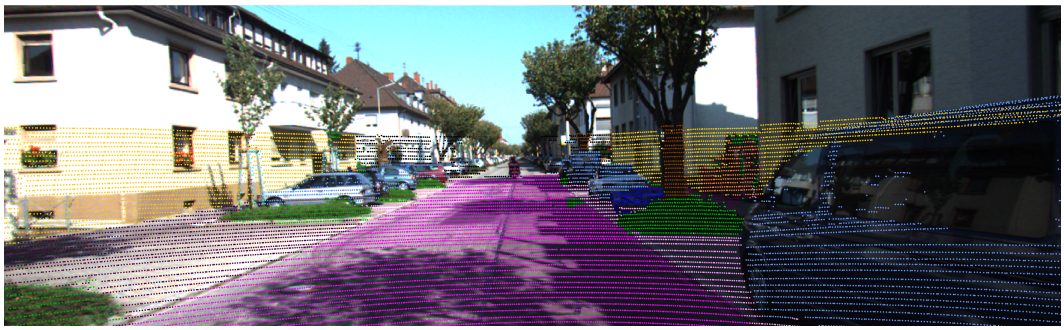


Figure 1. LiDAR point cloud projected onto the Cam 2 image

2.

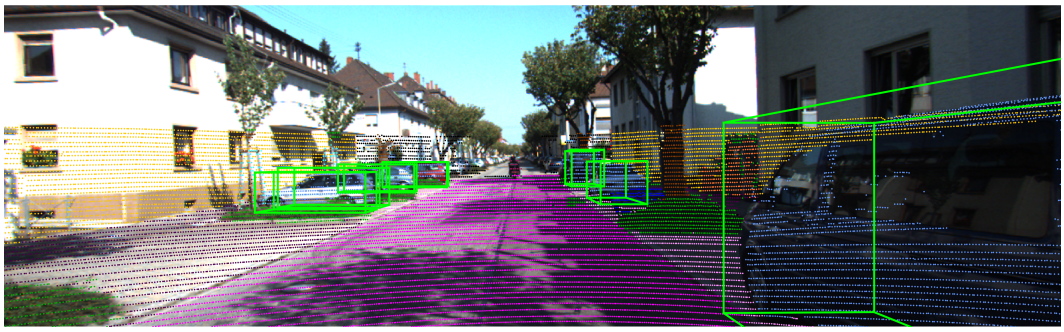


Figure 2. LiDAR point cloud and 3D bounding boxes of all given vehicles projected onto the Cam 2 image

3.

It appears as though the network failed to identify four cars, two of them next to each other and very close to the camera on the left, and two at the back of the field of view, one on each side of the road

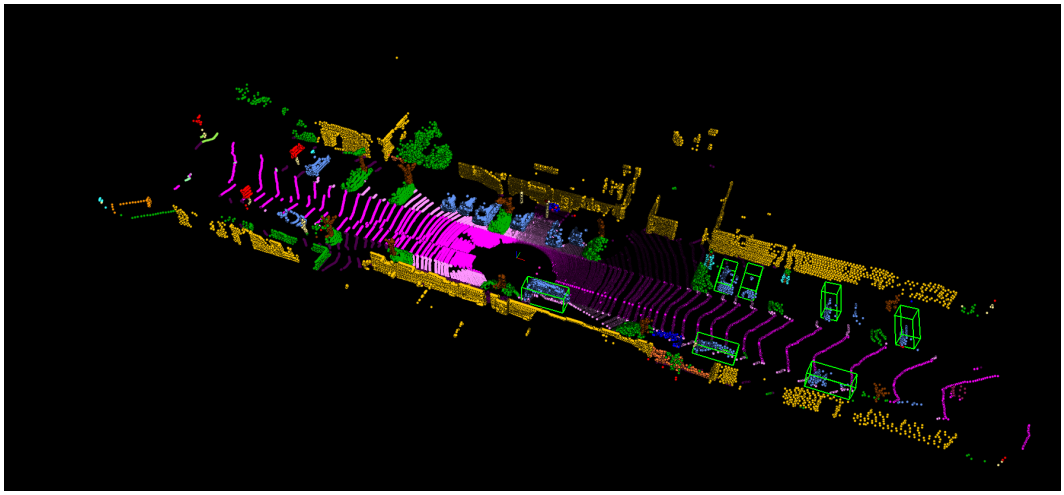


Figure 3. Colored 3D Overview of the data.p scene with bounding boxes

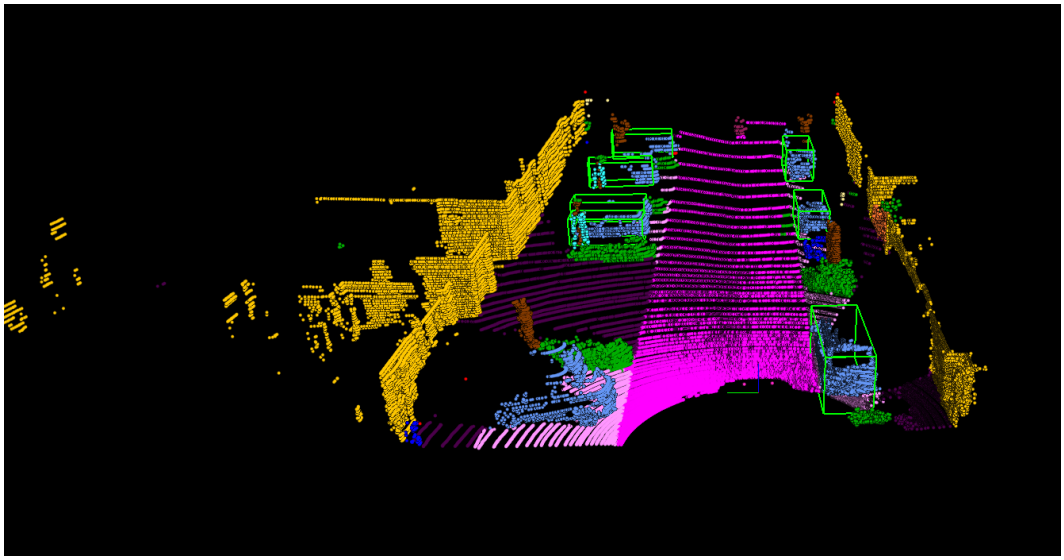


Figure 4. FOV filtered point cloud

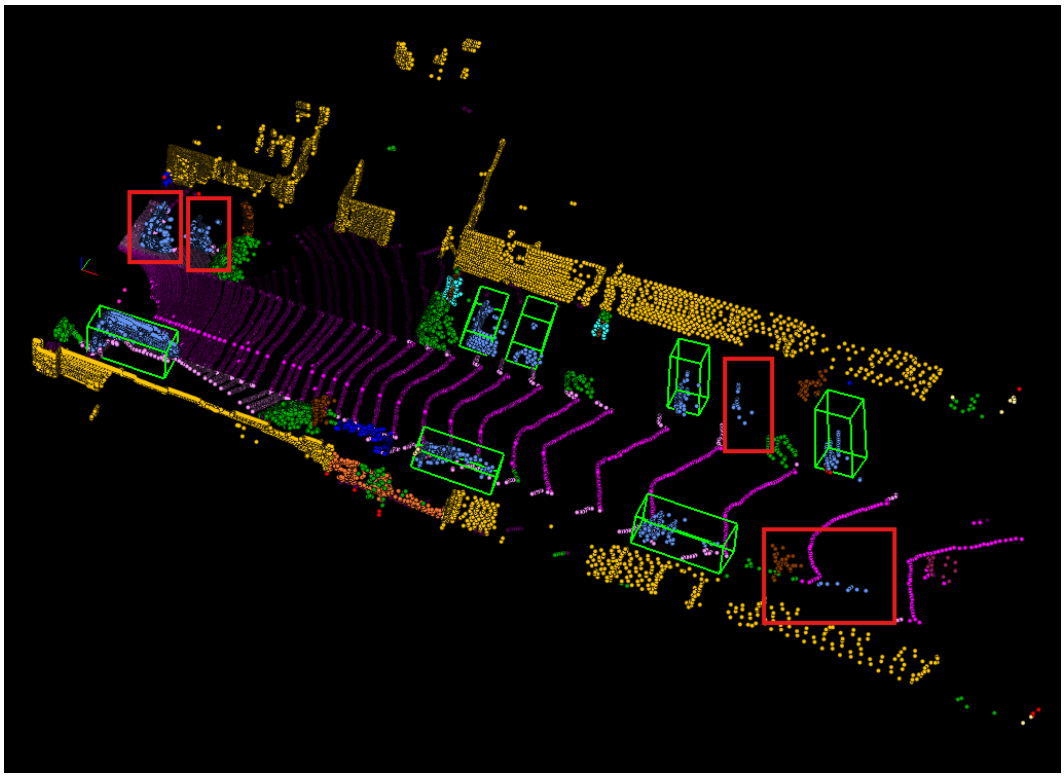


Figure 5. FOV filtered point cloud from side angle, with missed clusters of points associated to the vehicle class marked in red

Computer Vision and Artificial Intelligence for Autonomous Cars
GROUP 08: Report Project 2 Problem 2

Yannick Neuffer
ETH Zürich

Thorbjörn Höllwarth
ETH Zürich

Lukas Nüesch
ETH Zürich

2. Problem 2

As a first note: We pass all provided test cases.

2.1.

`label2corners`: We define a bounding box according to h, w, l and then rotate and shift it accordingly.

`get_iou`: After computing the corners for the pred and target we compute the volume for both boxes aswell as their intersection. The volume is simply h * w * l. The intersection is calculated using shapely. We first compute the intersection in the x-z plane and then multiply the vertical overlap.

Questions:
For t=0.5 the average recall for the validation set is 80.1. Recall is a good metric because in the first stage we mainly care about the true positives. The false positives can be filtered out in later stages.

2.2.

In this part, we process the coarse detection results from the stage-1 network by enlarging the first-stage bounding boxes, pooling the xyz points, and pooling the features. We define the following helper functions:

- `getEnlargedBoxes()`: Expands a bounding box by a given delta in all directions. The center gets moved by the delta too, to remain at the center of the bottom plane of the enlarged bounding box.
- `getBoxMinBoundBatch()`: Calculates the minimum corner of a bounding box.
- `getBoxMaxBoundBatch()`: Calculates the maximum corner of a bounding box.

Per box, using the minimum and maximum bounds, we create a mask to pool the xyz points and features within the enlarged bounding box:

Code Snippet 1. Filtering Points Within Enlarged Bounding Box.

```
mask_enlarged = (
    (xyz[:, 0] >= enlarged_min_bounds[i,
        0]) & (xyz[:, 0] <=
       enlarged_max_bounds[i, 0]) &
    (xyz[:, 1] >= enlarged_min_bounds[i,
        1]) & (xyz[:, 1] <=
       enlarged_max_bounds[i, 1]) &
    (xyz[:, 2] >= enlarged_min_bounds[i,
        2]) & (xyz[:, 2] <=
       enlarged_max_bounds[i, 2])
)

points_in_enlarged_box = xyz[
    mask_enlarged]
features_in_enlarged_box = feat[
    mask_enlarged]
```

Next, we filter the bounding boxes based on the number of points within the enlarged boxes:

- If a bounding box contains more than 512 points, we randomly sample 512 points.
- If it contains fewer than 512 points, we randomly repeat some points until the total equals 512.
- Bounding boxes with no points are discarded.

The results of three different bounding boxes with their associated xyz points can be seen in Figure 1.

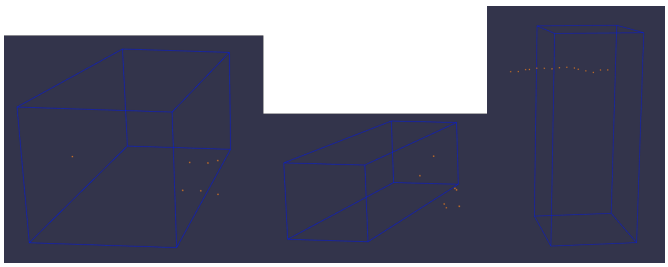


Figure 1. Three different original bounding boxes with their xyz points contained in their enlarged bounding box. Frame=0.

2.3.

`sample_proposals`: Samples 64 proposals for training or evaluation.
Steps:
1. IoU Calculation: Assigns each prediction to the ground truth with the highest IoU using `calculate_iou_and_assign_targets`.
2. Categorization: Classifies proposals into foreground, hard background, and easy background using IoU thresholds.
3. Sampling: Samples up to 32 foreground proposals, with the remaining as balanced background proposals.
4. Output: Returns 64 target boxes, 3D points, features, and IoUs.

`calculate_iou_and_assign_targets`: Computes IoU matrix and assigns each prediction the target with the highest IoU.

Code Snippet 2. IoU Calculation and Target Assignment.

```
iou_matrix = get_iou(pred, target)
assigned_targets = target[np.argmax(
    iou_matrix, axis=1)]
```

`categorize_proposals`: Categorizes proposals into foreground (`IoU` $\geq$ t_fg_lb), hard background (t_bg_hard_lb $\leq$ IoU $<$ t_bg_up), and easy background (`IoU` $<$ t_bg_hard_lb).

`sample_indices`: Randomly samples indices to ensure class balance.

`pad_samples`: Pads sampled indices to maintain the total sample count.

Questions:
(1) The sampling scheme is required because it ensures a balanced representation of foreground and background samples. This eliminates bias towards any particular class and prevents overfitting.
(2) This would result in an imbalanced dataset, and the model would struggle with challenging background samples.
(3) The model would struggle to learn the basic distinction between foreground and background.
(4) This would mean we also train with samples in the critical region between 0.45 and 0.55, which could confuse the model and introduce unwanted noise in the labels.
(5) This ensures that each ground truth box has at least one proposal and prevents missing detections. Otherwise we wouldn't learn from the samples that have no proposals.

This is the visualization of frame 0:

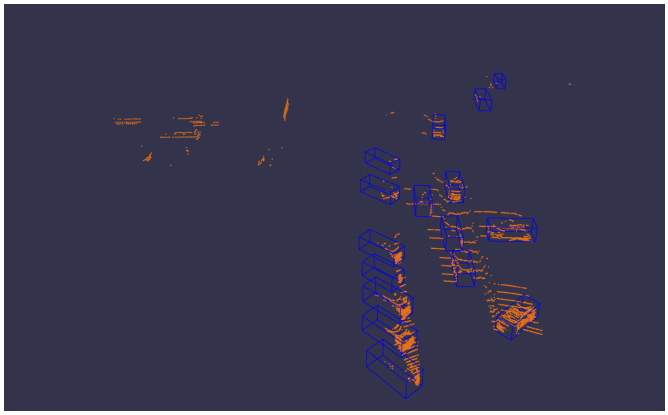


Figure 2. Visualization of frame 0 with all proposals only showing the pooled points.

2.4.

`RegressionLoss`: Computes the regression loss using only positive samples with IoU ≥ 0.55. The loss is computed over object location, size, and rotation parameters using Smooth L1 Loss. Only positive samples contribute to the loss.

Steps:
1. Positive Sample Mask: Identifies positive samples using IoU $\geq$ `positive_reg_lb`.
2. Filtering: Filters out negative samples to compute the regression loss only on positive samples.
3. Loss Computation: Computes the Smooth L1 Loss for location, size (weighted by 3), and rotation.
4. Output: Returns the computed regression loss.

Code Snippet 3. Regression Loss Computation.

```
positive_mask = (iou >= self.config['
    positive_reg_lb'])

# Filter positive samples
pred_pos = pred[positive_mask]
target_pos = target[positive_mask]

location_loss = self.loss(pred_pos[:,
    0:3], target_pos[:, 0:3]).mean()
size_loss = self.loss(pred_pos[:, 3:6],
    target_pos[:, 3:6]).mean()
rotation_loss = self.loss(pred_pos[:,
    6:], target_pos[:, 6:]).mean()

reg_loss = location_loss + 3 * size_loss
     + rotation_loss
```

`ClassificationLoss`: Computes the binary cross-entropy classification loss using positive samples with IoU

≥ 0.6 and negative samples with IoU ≤ 0.45. All other proposals are ignored to avoid noisy labels.

Steps:
1. Positive and Negative Masks: Identifies positive samples (IoU $\geq$ `positive_cls_lb`) and negative samples (IoU $\leq$ `negative_cls_ub`).
2. Filtering: Filters out samples that fall outside these thresholds.
3. Target Assignment: Assigns a target label of 1 for positive samples and 0 for negative samples.
4. Loss Computation: Computes binary cross-entropy loss between predicted scores and assigned targets.
5. Output: Returns the computed classification loss.

Code Snippet 4. Classification Loss Computation.

```
positive_mask = iou >= self.config['
    positive_cls_lb']
negative_mask = iou <= self.config['
    negative_cls_ub']

# Combine masks
valid_mask = positive_mask |
    negative_mask

# Assign labels
targets = torch.zeros_like(iou, device=
    pred.device)
targets[positive_mask] = 1.0

pred_valid = pred[valid_mask]
target_valid = targets[valid_mask]

loss = self.loss(pred_valid.view(-1),
    target_valid.view(-1))
```

2.5.

`nms`: Applies NMS to reduce overlapping bounding box predictions using a threshold of 0.1. The IoU is calculated only on the BEV.

Steps:
1. Sorting: Sorts predicted bounding boxes by descending confidence scores.
2. Selection: Iteratively selects the bounding box with the highest score and removes overlapping boxes with IoU above the threshold.
3. BEV IoU Computation: Computes IoU only in BEV by flattening the y coordinate and setting height to 1.0.
4. Output: Returns bounding boxes and their confidence scores after NMS.

Code Snippet 5. NMS Implementation.

```
idxs = np.argsort(score)[::-1]
pred = pred[idxs]
score = score[idxs]

s_f = []
c_f = []

while len(pred) > 0:
    curr_box = pred[0].reshape(1, -1)
    curr_score = np.array([[score[0]]])

    s_f.append(curr_box)
    c_f.append(curr_score)

    if len(pred) == 1:
        break

    # Compute BEV IoU
    pred_bev = pred.copy()
    pred_bev[:, 1] = 0.0  # Flatten y-
        coordinate
    pred_bev[:, 3] = 1.0  # Standardize
        height

    main_box_bev = pred_bev[0].reshape
        (1, -1)
    rest_bev = pred_bev[1:]

    ious = get_iou(main_box_bev,
        rest_bev).squeeze(0)
    mask = ious <= threshold

    # Remove overlapping boxes
    pred = pred[1:][mask]
    score = score[1:][mask]

s_f = np.vstack(s_f)
c_f = np.vstack(c_f)
return s_f, c_f
```

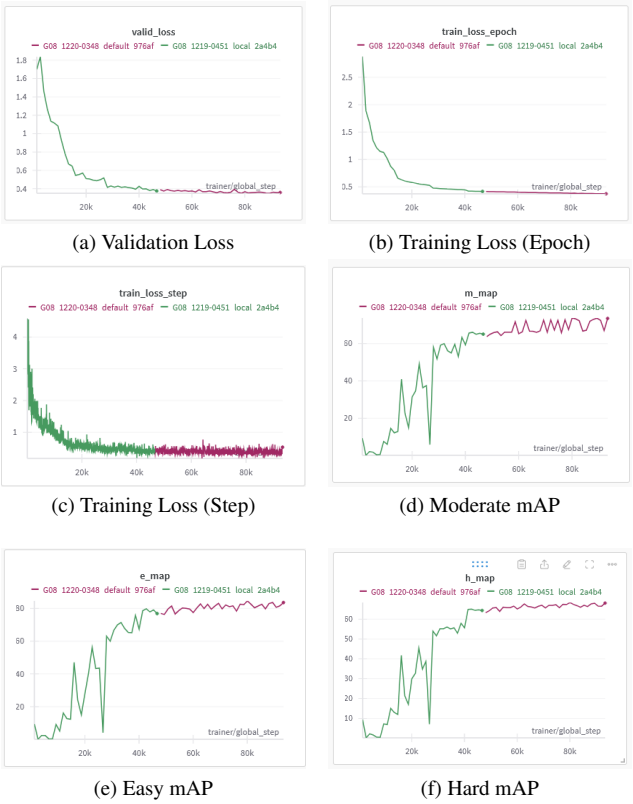
Questions:
The IoU could also be calculated in 3D but BEV is more practical. This is because the BEV is computationally more efficient than precise 3D IoU. But since most objects in self-driving scenarios are on the same plane, this simplification makes sense. If we were to use actual 3D IoU, we would get more precise overlap measurements, which can be beneficial if the driving scene has varying elevation.

2.6.

For time constraint reasons we did all our training locally on a RTX 3080 (after epoch 35 we started using a checkpoint). We increased the epochs to 70 since our steps per epoch were roughly half of what it would be on the cluster.

The CodaLab submission achieved following scores:
Moderate mAP: 63.95 (Rank 17)
Easy mAP: 79.31 (Rank 10)
Hard mAP: 63.84 (Rank 13)

First we show the loss and precision curves:



(a) Validation Loss

(b) Training Loss (Epoch)

(c) Training Loss (Step)

(d) Moderate mAP

(e) Easy mAP

(f) Hard mAP

Figure 3. Loss and precision curves.

Here we visualize the 3rd epoch, 35th epoch and 70th epoch:

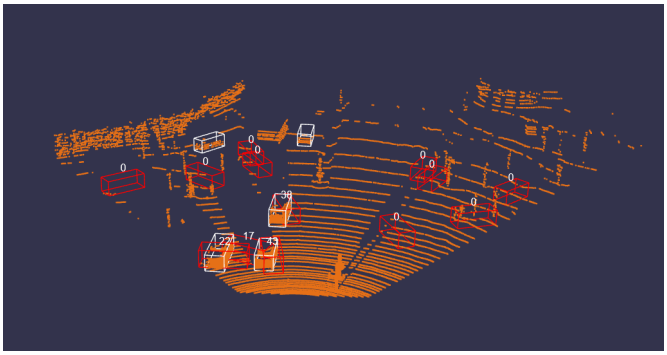


Figure 4. Visualization at Epoch 3.

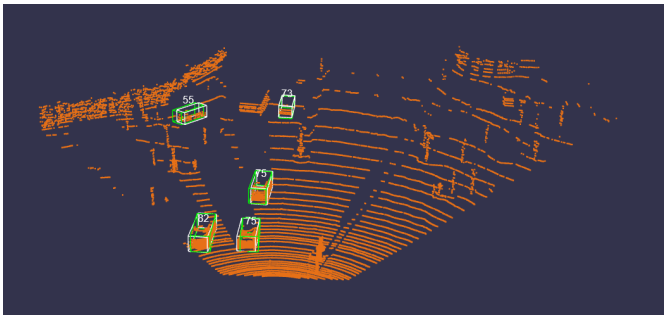


Figure 5. Visualization at Epoch 35.

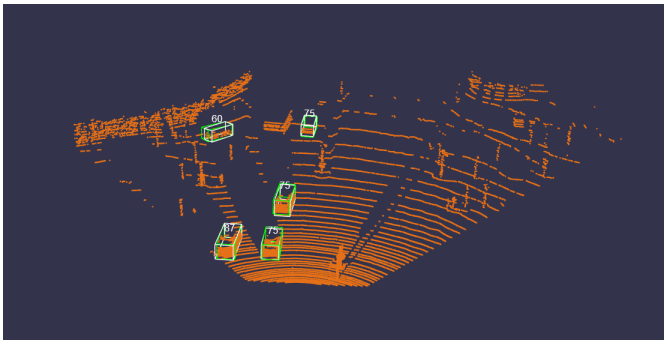


Figure 6. Visualization at Epoch 69.

GROUP 08: Report Project 2 - Problem 3

Yannick Neuffer
ETH Zürich

Thorbjörn Höllwarth
ETH Zürich

Lukas Nüesch
ETH Zürich

Abstract

In this work, we propose an enhancement to the baseline 3D object detection pipeline using Graph Neural Networks (GNNs) combined with attention mechanisms. By introducing relational modeling and dynamic feature weighting, our approach addresses the limitations of the baseline's local feature aggregation and aims to improve its ability to understand complex spatial structures in point clouds. This integration aims to enhance the model's ability to detect and classify objects in challenging scenarios.

1. Introduction

Object detection in 3D point clouds is a crucial task for autonomous driving systems. While the baseline method in Problem 2 employs PointNet++-like operations for feature extraction, it lacks the ability to explicitly model relationships between points. This shortcoming limits its capacity to capture complex spatial dependencies, particularly in cluttered or occluded scenes.

To address these limitations, we propose a novel integration of Graph Neural Networks (GNNs) and attention mechanisms into the detection pipeline. Our contributions can be summarized as follows:

- We introduce GNN layers to model the relational structure of the point cloud, enabling the network to aggregate features from neighboring points effectively.
- We integrate an attention mechanism to dynamically weigh the importance of features, allowing the model to focus on critical regions within the point cloud.
- We extend the baseline by introducing a novel node-level dropout mechanism during training, ensuring better generalization and robustness.

Our enhancements address the limitations of the baseline method, providing a robust framework for 3D object detection in autonomous driving scenarios.

2. Related Work

The field of 3D object detection has seen significant advancements with the advent of neural network architectures tailored for point cloud data. Our work draws inspiration from two key areas:

PointNet and PointNet++ PointNet [1] pioneered the use of neural networks for processing unordered point clouds by applying shared multi-layer perceptrons (MLPs) to each point independently. PointNet++ [2] extended this approach by introducing hierarchical feature learning with local neighborhoods, enabling better understanding of local geometric structures. However, these methods treat points independently within each neighborhood, lacking explicit relational modeling.

Graph Neural Networks GNNs [3] have emerged as a powerful tool for processing graph-structured data. Point-GNN [4] demonstrated the effectiveness of GNNs in 3D object detection by constructing graphs over point clouds and aggregating features from neighboring nodes. Our work adopts a similar approach but combines GNNs with attention mechanisms to enhance feature aggregation dynamically.

Attention Mechanisms Attention mechanisms [5] have revolutionized sequence modeling by allowing networks to focus on the most relevant parts of the input. Recent works [6] have explored their application in 3D point clouds, showing improved performance in tasks like segmentation and detection. Our integration of attention mechanisms with GNNs provides an additional layer of feature refinement.

Our proposed method combines the strengths of these approaches, introducing a relational understanding of point clouds and dynamic feature weighting to improve detection performance.

3. Method

The proposed method enhances the baseline 3D object detection pipeline by incorporating GNN layers, attention

mechanisms, and a novel node dropout strategy. We describe the key components of our approach in detail below.

3.1. Graph Construction

To incorporate GNNs, we represent the point cloud as a graph where each point is a node, and edges connect a point to its k nearest neighbors based on Euclidean distance. This graph is encoded using an adjacency matrix $A \in \mathbb{R}^{N \times N}$, where A_{ij} represents the connection strength between nodes i and j. The adjacency matrix is computed in the preprocessing step.

3.2. Graph Neural Networks

GNN layers are added to the model after the set abstraction layers. Each GNN layer updates the feature representation of a node by aggregating features from its neighbors:

$$H^{(l+1)} = \sigma\left(AH^{(l)}W^{(l)}\right), \tag{1}$$

where $H^{(l)}$ and $H^{(l+1)}$ are the input and output feature matrices of layer l, A is the adjacency matrix, $W^{(l)}$ is a learnable weight matrix, and σ is an activation function (e.g., ReLU).

3.3. Attention Mechanism

To further refine feature aggregation, we incorporate an attention mechanism. The attention layer assigns a weight to each feature vector based on its relevance, computed as:

$$\alpha_i = \text{softmax}\left(W_2 \cdot \text{ReLU}(W_1 \cdot H_i)\right), \tag{2}$$

where W_1 and W_2 are learnable weight matrices. The weighted features are then aggregated to produce the final representation for each point.

3.4. Node Dropout Strategy

To improve generalization and robustness, we introduce a node-level dropout mechanism during training. This strategy randomly drops nodes (i.e., points in the point cloud) during graph convolution operations, preventing the model from over-relying on specific nodes. This dropout is implemented as follows:

$$H_i^{(l)} = \begin{cases} 0 & \text{if dropped,} \\ H_i^{(l)} & \text{otherwise,} \end{cases} \tag{3}$$

where each node has a fixed probability of being dropped during training.

4. Experiments and Results

To evaluate the proposed enhancements, we outline a series of experiments and metrics:

4.1. Evaluation Metrics

We utilize the following metrics to compare the performance of our method against the baseline:

- Mean Average Precision (mAP): To measure the detection accuracy across different IoU thresholds.
- Recall: To quantify the proportion of correctly detected objects.
- Inference Time: To assess the computational efficiency of the enhanced model.

4.2. Proposed Experiments

Comparison with Baseline We evaluate the overall performance of the proposed architecture against the baseline model from Problem 2, using the same dataset and evaluation setup.

Ablation Studies To isolate the contribution of each module, we conduct the following ablation studies:

- Without GNN Layers: To quantify the impact of relational modeling.
- Without Attention Mechanism: To evaluate the role of dynamic feature weighting.
- Without Node Dropout: To assess the effect of regularization on generalization.

Visualization We leverage the visualization tools from Problem 1 to illustrate the impact of attention mechanisms and GNN layers on feature aggregation, highlighting key regions in the point cloud.

4.3. Results

Unfortunately, due to the long training times and several crashed runs, we were not able to complete the proposed tests in time for the deadline.

5. References

References

[1] Qi, C. R., Su, H., Mo, K., Guibas, L. J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In CVPR, 2017. 1

[2] Qi, C. R., Yi, L., Su, H., Guibas, L. J. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In NeurIPS, 2017. 1

[3] Kipf, T. N., Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In ICLR, 2017. 1

[4] Shi, W., Rajkumar, R. Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud. In CVPR, 2020. 1

[5] Vaswani, A., Shazeer, N., Parmar, N., et al. Attention Is All You Need. In NeurIPS, 2017. 1

[6] Yang, Z., Wang, Y., Liang, F., et al. Modeling Point Clouds with Self-Attention and Gumbel Subset Sampling. In CVPR, 2019.

1