CSC 490 Module 1 Part A

Juan Deng
Yu Hau Chen

Due: March 5, 2022

Part A: Learning the Fundamentals

2.1. LiDAR Voxelization

Part 1. Implement the forward method of the Voxelizer class

See Voxelizer.py

Part 2. Visualize the result of Voxelizer.forward on one LiDAR point cloud for different resolutions step $\in \{0.25, 0.50, 1.0, 2.0\}$. How do the visualizations vary as you sweep different resolutions? What is the trade-off between memory consumption and the fidelity of the voxel grid?

Figure 1 visualizes the result of Voxelizer.forward on one LiDAR point cloud for different resolutions step. We observe that as the resolution step grows larger, the visualization become lighter, and extracts less information from the point cloud. Therefore, we conclude that as memory consumption increases, the fidelity of the voxel grid increases.

2.2 Model Training Inference

Part 1. Implement the create_heatmap function

See loss_target.py

Part 2. Implement the build_target_tensor_for_label method of DetectionLossTargetBuilder

See loss_target.py

Part 3. Implement the function heatmap_weighted_mse_loss

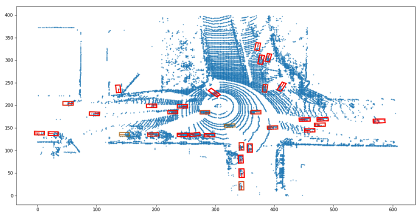
See loss_function.py

Part 4. Implement the inference method of DetectionModel

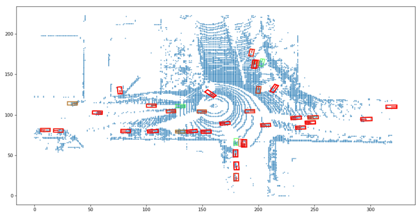
See model.py

Part 5. Try overfitting to a single training sample using the default hyperparameters. Your detector should be able to do this and (almost) perfectly detect every vehicle in that one scene. Include a visualization of the resulting detections after overfitting in your write-up

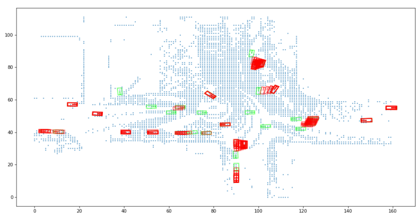
Figure 2 showcases the visualization of the resulting detections after overfitting. We observe that the detector is able to perfectly detect every vehicles.



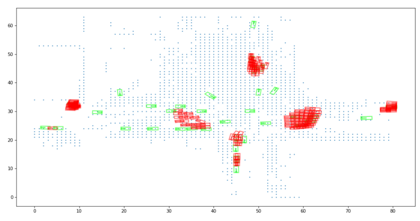
(a) Step 0.25



(b) Step 0.50



(c) Step 1.0



(d) Step 2.0

Figure 1: Voxelized LiDAR with Different Steps

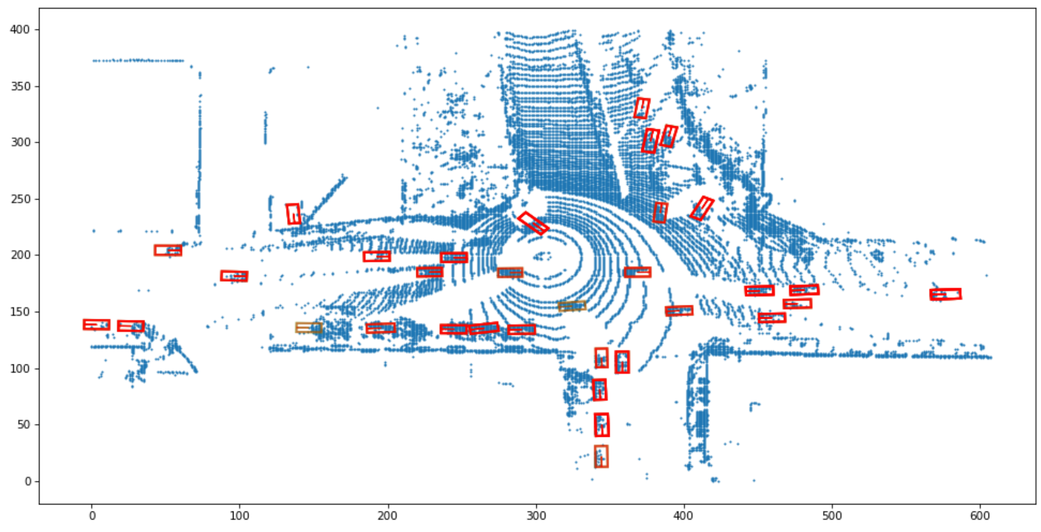
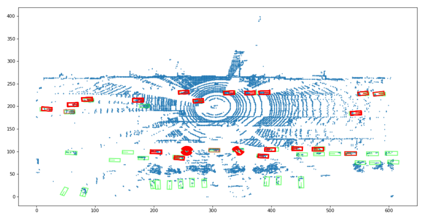
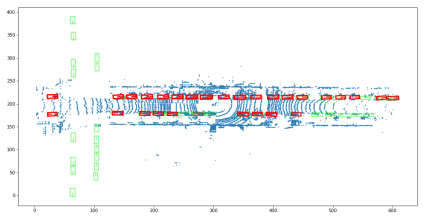


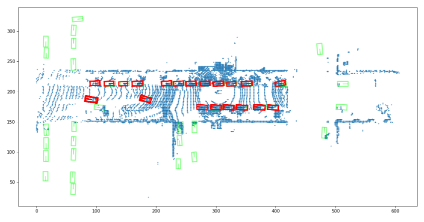
Figure 2: Overfitted Detections



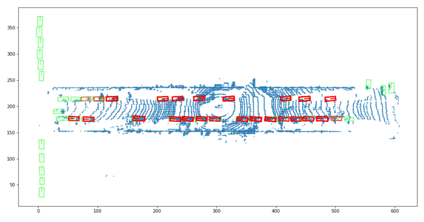
(a) Example 1



(b) Example 2



(c) Example 3



(d) Example 4

Figure 3: Model's Detections on Validation Dataset Examples

Part 6. After you've successfully overfitted your detector to a single training sample, it's time to train it on the full dataset. A word of warning — this could take up to an hour using the default hyperparameters. When training is complete, test your model on the held-out validation dataset. Include several visualizations of your model's detections on the validation dataset in your write-up

Figure 3 showcases some examples of model's detections on the validation dataset. We observe that the model is able to detect most of the vehicles that are close to the road.

2.3 Evaluation

Part 1. Implement the function compute_precision_recall_curve

See average_precision.py

Part 2. Implement the function compute_area_under_curve

See average_precision.py

Part 3. Implement the function compute_average_precision using Parts 1 and 2. Using the provided starter code, evaluate your trained detection model. Plot the PR curve for thresholds $\tau \in \{2.0, 4.0, 8.0, 16.0\}$ and report their corresponding AP. How does the PR curve and AP change as τ varies?

Figure 4 showcases the PR Curve and AP for different thresholds τ. We observe that the PR curve becomes closer to the upper right corner as τ increases, we also observe that AP increases as τ increases.

Part 4 Explain how the ground truth for the metrics is computed. Describe in detail the transformations applied to go from the raw data (the Pandaset files) to the model targets and the inputs to the compute_average_precision function. Discuss the implications of the transforms and how they influence what the model learns. Are there learning targets that we use during training that we could not realistically expect the model to predict at inference time?

First, we used the score of each detection as its own score threshold. That is, we assume that each detection score is larger than or equal to the score threshold. Then we begin to loop over the lists of frames. The first step is to generate the euclidean distance tensor between each detection and each label. Then, we

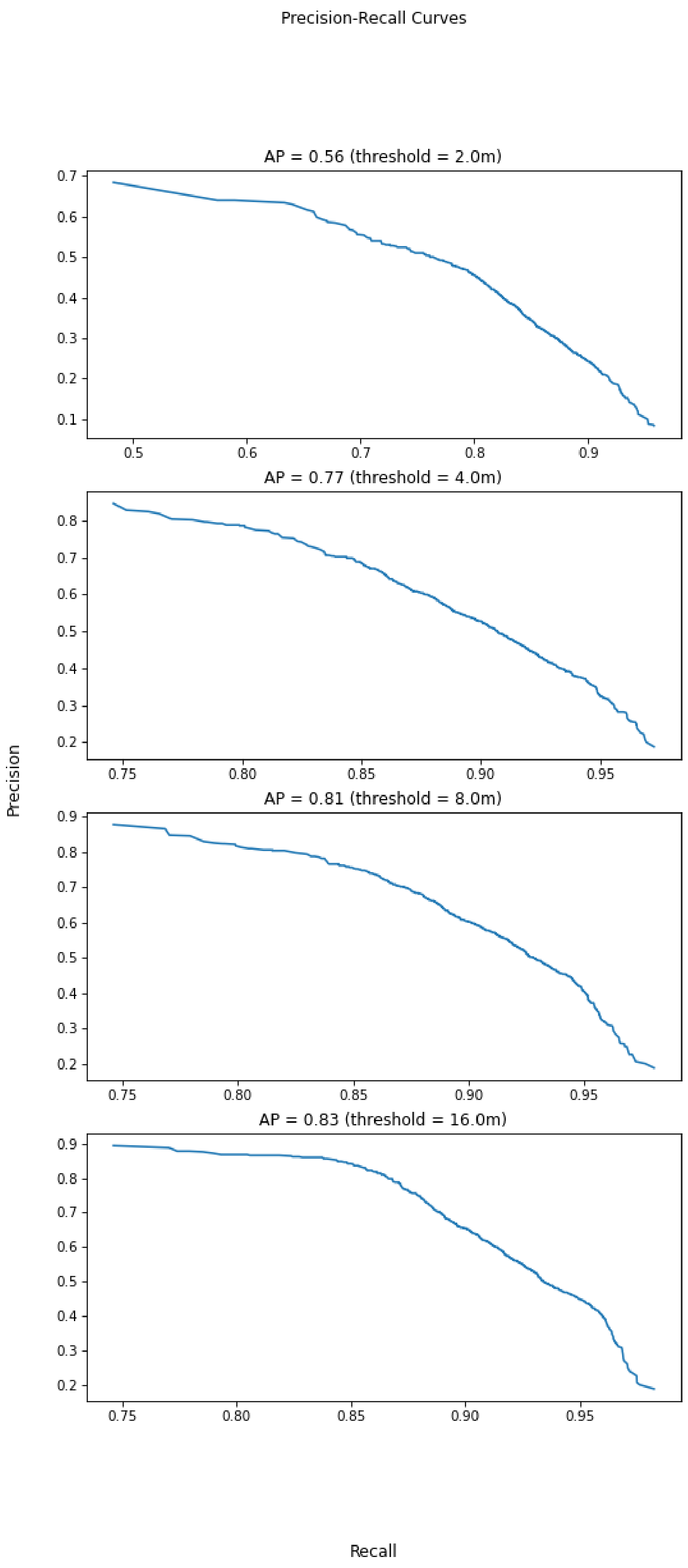


Figure 4: PR Curve

loop through the euclidean distance tensor for each detection to check if its euclidean distance is smaller than or equal to the threshold and filter our euclidean distance tensor to get the index of labels satisfying this threshold. Then, we look at the whole column of each satisfied labels to check if the euclidean distance for this detection is the largest value among all under-threshold euclidean distance in the same labels. We use torch.any and torch.all to get the results of TP and FP. For FN calculation, we used the index tensor generated from the previous step to find the TP labels. We generate two recording tensors: one for for detection (TP + FP) and one for label (FN). We updated the associated value in these two tensors for each frame iteration and compute precision and recall.

Since the number of detection is greatly larger than the number of label, which means a large amount of our detection are useless. If we filter these useless detection using TP and FP, our running time will be largely reduced. We could not expect predicted bounding box at inference time since they are imaginary rectangles.