Department of Informatics
Technical University of Munich

Master's Thesis in informatics

Real-time LiDAR-based 3D Object Detection on the Providentia++ Test Stretch Using a Single-Stage Architecture

Echtzeit LiDAR basierte 3D Objekterkennung auf der Providentia++ Teststrecke unter Einbeziehung einer Single-Stage Architektur

Supervisor Prof. Dr.-Ing. habil. Alois C. Knoll

Advisor M.Sc. Walter Zimmer

Author Jialong Wu

Date December 15, 2021 in Garching

Disclaimer

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, December 15, 2021

(Jialong Wu)

Abstract

Huge progress has been made in the field of autonomous driving in recent years. However, due to the limitation of perception range and occlusions, relying on the perception of vehicles themselves is not enough. The Providentia++ project aims to build an Intelligent Infrastructure System, which can provide additional information about the real-time traffic situation and therefore improve the reliability of autonomous driving.

As part of the Providentia++ project, this thesis focuses on 3D real-time object detection using infrastructure-mounted LiDARs. We propose SE-ProPillars, a single-stage LiDAR-only 3D object detector. We train and evaluate our model on the KITTI dataset. Experiments show that our model outperforms the baseline (ProPillars) by $(0.64\%, 0.73\%, 2.67\%)$ 3D mAP and $(2.19\%, 0.08\%, 2.23\%)$ BEV mAP on easy, moderate and hard difficulties respectively, while the inference speed is accelerated from 32 ms to 22 ms. We also customize the IPS300+ dataset to fit our scenario, train on it, and perform transfer learning on our dataset — proFusion. Furthermore, the sparsity of LiDAR-captured point clouds has always been a problem for downstream tasks. We design a point cloud registration algorithm for merging scans from multiple infrastructure-mounted LiDARs. We also study neural network-based point cloud upsampling methods to address the sparsity issue in point clouds. We generate a high-resolution point cloud dataset — proSynth with 1000 data samples using the CARLA simulator. We train our point cloud super-resolution model on the synthetic dataset and also test the model on real-world low-resolution data. Finally, we implement two different types of real-time web-based visualization tools for point cloud data.

Contents

Chapter 1

Introduction

Autonomous driving has been depicted in science fiction movies and novels for several decades. One of the most famous autonomous vehicles is named Herbie in the movie *The Love Bug* made in 1968. Decades later, with the development of deep learning, the fantasy of humans is gradually becoming reality.

In this chapter, we give a brief introduction to the topic and structure of this thesis. The motivation of the Providentia++ project will be introduced in Section 1.1. In Section 1.2 we discuss the problems and challenges that this thesis is trying to solve. Our main contributions will be summarized in Section 1.3. Section 1.4 shows the overall structure of this thesis.

1.1 Motivation

In recent years, more and more applications in the field of autonomous driving have emerged. In closed scenarios, such as ports or logistics centers, autonomous vehicles are revolutionizing the efficiency of freight. Since these closed scenarios have lower speeds, limited ranges, and fewer security risks, autonomous vehicles are being put into use earlier. However, in open scenarios, autonomous driving is facing much more challenges, e.g. vehicle speeds, sensor ranges, occlusions, more complex environments and traffic, etc. Aside from the technical challenges, there are even doubts from ethical aspects. As a consequence, at present open scenarios of automatic driving on the market mostly appear in the form of driver assistance systems.

A Ferrari cannot run fast on the muddy ground. Autonomous driving is hard to achieve without the assistance of infrastructure on the road. Krämmer et al. [32] pointed out that Intelligent Transportation Systems (ITS) can help to deal with the aforementioned challenges and improve the safety of autonomous driving.

Providentia++ is an ITS research project aiming to build a digital twin of the real-time traffic situation on the Providentia++ test stretch: along the Highway A9 extending into the urban area. It uses sensors installed on gantry bridges to provide additional information about the current traffic to vehicles. This additional information contains the status of all road users

with their position, velocity, type, size, etc. With this help, autonomous driving systems can make safer and more efficient motion plans. Because of higher perspectives, sensors mounted on the infrastructure have wider coverage than vehicle-mounted sensors and suffer less from occlusions. Therefore, the reliability and safety of autonomous driving will be improved. Figure 1.1 shows one of our sensor stations with multiple sensors mounted on the gantry bridge. Data captured by those sensors is processed by the data fusion unit which equips powerful hardware to keep computing and processing in real-time.

Figure 1.1: Picture of a Providentia sensor station [32].

1.2 Purpose of Research

In an Intelligent Transportation System, reliability, real-time performance, robustness, and security are the most important concerns. We mainly work on improving the first two elements.

First of all, vehicle perception is one of the key tasks in Providentia++. 3D object detection using LiDAR sensors and point clouds has become a hot topic in recent years. We research existing detectors and make further improvements on the precision to improve the reliability of the system. At the same time, we keep the inference speed higher than the real-time requirement of 25 frame per second (FPS).

In the Providentia++ test stretch, there are multiple LiDARs mounted on each sensor station. The scanning results of them should be combined to provide the traffic situation of the entire area. In this thesis, we design a point cloud registration algorithm suitable for our scenario. It can merge infrastructure-mounted LiDAR scans in real-time.

Point clouds captured by LiDARs suffer from the sparsity issue, which may lead to few points projected on a vehicle, especially over longer distances. Therefore, it largely increases the difficulty of detecting vehicles and undermines reliability. Registration of multiple point clouds

can raise the number of points to some extent. Another way of increasing the density is to use a LiDAR with a higher resolution. For example, an Ouster OS1-128 has twice the number of channels as OS1-64 and can capture twice as many points. However, a 128-channel LiDAR is more expensive than a 64-channel one. An alternative method is point cloud super-resolution: training a neural network that takes 64-channel point clouds as input and generates 128-channel point clouds as output. In this thesis, we create a synthetic dataset — proSynth using the CARLA simulator [18]. It contains 128-channel point clouds captured by virtual LiDARs in the simulated Providentia++ test stretch map. We train our point cloud super-resolution model on the dataset and test it on real-world data.

In order to monitor the real-time traffic situation, we implement two web-based visualization tools. One is a 3D visualizer with a 3D navigation function. The other one is a 2D projection-based visualizer which requires lower bandwidth.

To train a deep learning-based detector, huge amounts of training data are necessary. Unfortunately, since our LiDAR sensors are mounted higher, existing 3D datasets recorded by vehicle-mounted sensors have significant differences from ours. Consequently, detector models trained on existing datasets have poor performance in the perception of Providentia++. The labeling work of our dataset — proFusion is still ongoing. To solve the problem of insufficient data, we train our detector on the IPS300+ dataset, which is also captured by infrastructure-mounted LiDARs, and fine-tune on proFusion. To speed up the labeling work, a new automatic labeling function is added to our labeling tool — ProAnno that is based on 3D BAT [89]. It is trained on a small piece of data and can make predictions as references to reduce human labor.

1.3 Contribution

This thesis is part of the Providentia++ project. We focus on LiDAR sensor-related applications. In summary, we made the following contributions:

- We take ProPillars [87] as our baseline and propose a single-stage LiDAR-only 3D object detector — SE-ProPillars. In the experiment on the KITTI dataset, compared to the baseline, our model improves by $(0.64, 0.73, 2.67)$ % 3D mAP and $(2.19, 0.08, 2.23)$ % BEV mAP on easy, moderate, and hard difficulties respectively, while the inference time is reduced from 32 ms to 22 ms.
- We train our model on IPS300+ and perform transfer learning on the first batch of data from proFusion.
- We design a point cloud registration algorithm for LiDARs installed on infrastructure.
- We create a synthetic dataset — proSynth with 1000 128-channel point cloud frames using the CARLA simulator and train a point cloud super-resolution model on it.
- We implement two web-based visualization tools for real-time LiDAR data in 3D and 2D view respectively.
- We add an automatic labeling function to our labeling tool ProAnno [89].

1.4 Structure of the Thesis

In Chapter 1, we introduce the background and motivation of the Providentia++ project and the purpose of our research. Our main contributions are also summarized.

Chapter 2 provides a comprehensive literature review on the previous work related to our research: existing 3D autonomous driving datasets; state-of-the-art 3D LiDAR-only object detectors; popular point cloud registration methods and their pros and cons; neural network-based point cloud upsampling methods.

In Chapter 3, we present the neural network structure of our detector SE-ProPillars. Everything about our registration algorithm and super-resolution model will also be introduced in this chapter, as well as our visualization tools and the auto label function.

In Chapter 4, we describe implementation details of SE-ProPillars, as well as the training process and hyperparameters on the KITTI, IPS300+, and proFusion datasets.

In Chapter 5, experimental results of our detector on these three datasets will be given, as well as some visualized detection results.

Chapter 6 summarizes this thesis, and Chapter 7 gives some potential directions for future work.

Chapter 2

Related Work

This chapter introduces the previous work that is related to our study. Section 2.1 introduces popular autonomous driving datasets and our proFusion dataset. Section 2.2 gives a comprehensive literature review on existing state-of-the-art 3D detectors. In section 2.3, we review point cloud registration methods. Section 2.4 presents deep learning-based point cloud upsampling methods in recent years.

2.1 3D Autonomous Driving Datasets

Deep learning relies greatly on the training dataset and its quality. Autonomous driving datasets usually have multi-modal data sources, e.g. LiDARs, cameras, radars, GPS/IMU, etc. These multi-modal data facilitate different types of tasks, such as 3D LiDAR-only detection, 2D camera-only detection, LiDAR-camera fusion detection, tracking, SLAM, segmentation, etc. For object detection tasks, datasets provide annotations in the form of bounding boxes. Since we focus on LiDAR-only object detection, we introduce these popular datasets mainly from a LiDAR perspective.

KITTI. The most famous autonomous driving dataset is KITTI [22] built in 2012. The data is recorded by vehicle-mounted sensors, including four cameras, a Velodyne HDL-64E LiDAR, and a combined GPS+IMU inertial navigation system. The point clouds scanned by Velodyne contain 3D coordinate and intensity information. Each scan has around 120K points. The dataset has 15.4K point cloud frames in total with 80K labeled objects. The object annotations have 8 classes: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person (sitting)', 'Cyclist', 'Tram', and 'Misc' (e.g., Trailers, Segways). According to the size, visibility, occlusion and truncation level of labeled objects, they are divided into three levels of difficulty: 1) Easy: objects are fully visible, with bounding box height above 40 pixels, truncation level below 15%; 2) Moderate: objects are partly occluded, with bounding box height above 25 pixels, truncation level below 30%; 3) Hard: objects are difficult to see, with bounding box height above 25 pixels, truncation level below 50%. Precision in different classes and difficulties is usually measured separately when evaluation.

nuScenes. The nuScenes [9] dataset is also a popular dataset, published in 2019. The data

is also recorded by vehicle-mounted sensors: six cameras, a 32-channel Velodyne LiDAR, five radars, and a GPS+IMU system. It contains 40K labeled point cloud frames (around 34K points per frame) and 1.4M 3D bounding boxes. For the object detection task, the nuScenes annotated 10 classes: 'Car', 'Pedestrian', 'Bus', 'Barrier', 'Traffic Cone', 'Truck', 'Trailer', 'Motorcycle', 'construction vehicles', and 'Bicycle'. An additional feature of the nuScenes is that it also records data in different lighting conditions (day and night) and different weather (sun, rain, and clouds).

Waymo. Waymo [60] is one of the largest dataset in the field of autonomous driving, released in 2019. It contains 230K labeled point cloud frames (around 177K points per frame) and 12M 3D bounding boxes. Sensors used to record the data are five 64-channel LiDARs, 5 cameras mounted in different parts of the ego vehicle. Data is recorded under different lighting conditions (day, night, and dawn) but no weather changes. Objects have four classes: 'Vehicle', 'Pedestrian', 'Cyclist', and 'Sign'. In addition to the coordinate and intensity value in the first two datasets, the scanned point clouds in the Waymo also have an elongation feature.

proFusion. proFusion is the dataset of the Providentia++ project but still in progress. As part of the Providentia++ project, data is collected by sensors mounted on the gantry bridges on the Highway A9 and the extended urban area. In total, eight LiDARs and forty-two cameras will be used for data collection. LiDARs include Ouster OS1-64 LiDARs (main) and Valeo SCALA2 LiDARs. The OS1-64 LiDARs have 64 channels and can capture up to 131K points per frame. Moreover, it also provides ambient and range information in addition to coordinates and intensity. Figure 2.1 and 2.2 show an example of a point cloud frame in proFusion in different views.

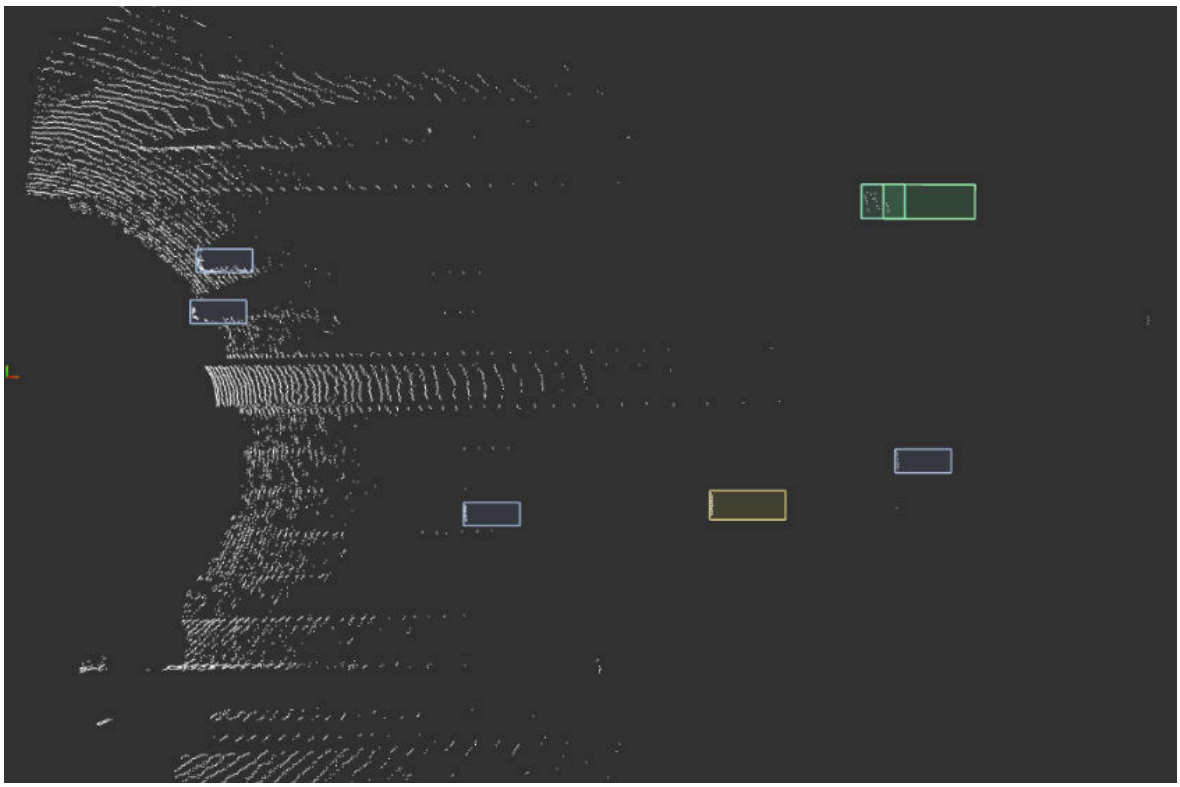

Figure 2.1: Point cloud frame with annotations in bird's eye view.

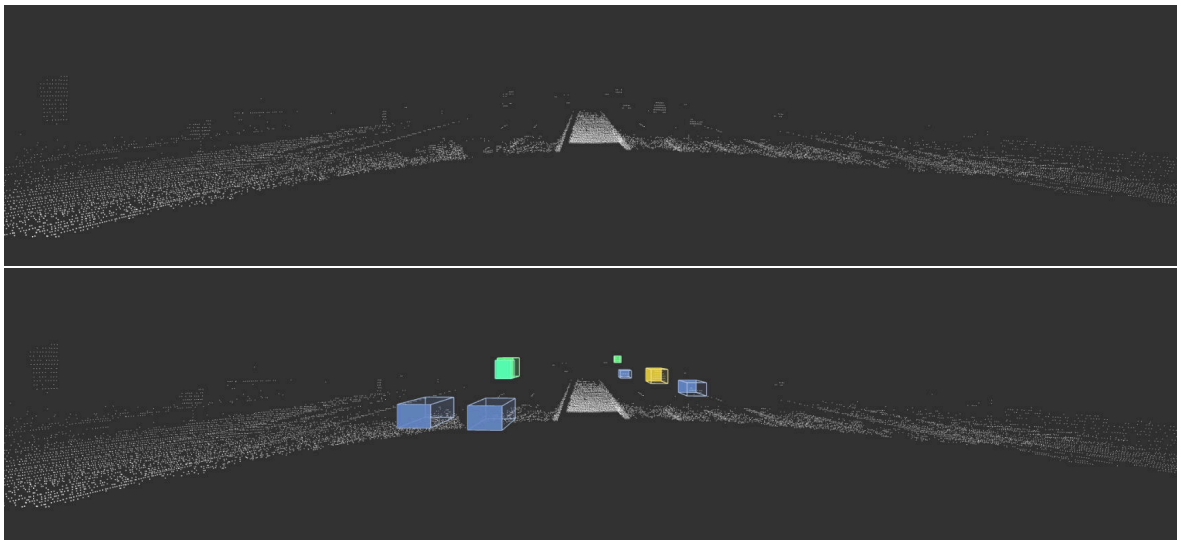

Figure 2.2: Point cloud frame in 3D front view.

2.2 3D LiDAR-only Object Detection

3D object detection is a very active field in the past five years. Detectors evolve very quickly and various methods have been proposed. To refine the scope of the thesis, we focus on LiDAR-only detectors which take point clouds as input and generate 3D bounding boxes and classifications as output.

According to the detection process, existing 3D detectors can be divided into two types: single-stage detectors and two-stage detectors. A typical 3D single-stage detector consists of 1) a feature encoder module to encode the point cloud data into different representations of features, 2) a middle feature extractor module for further feature extraction, and 3) a region proposal network (RPN) as the detection head to directly generate the final predictions. Compared with the single-stage detector, two-stage detectors usually have an additional bounding box refinement process, where detectors focus only on the regions of interest (ROI) generated in the first stage. Because of the additional refinement process, most of the two-stage detectors have a higher precision but lower speed than single-stage detectors. However, this pattern is gradually being broken in recent methods, as researchers are trying to address the limitations of both approaches. Table 2.1 gives a summary of the most representative detectors in recent years.

According to the form of feature representation, detectors can be divided into three main categories: point-based, voxel-based, and hybrid methods. In this section, we make a comprehensive literature review of these three categories respectively.

2.2.1 Point-based Methods

Point-based detectors take point clouds as input, and in the whole process of the neural network, features always maintain the form of point-wise features, either by a subset of points or derived virtual points. Point-based methods are more intuitive but have to deal

with a huge amount of point-wise features, which leads to a relatively lower inference speed.

Point Feature Encoder

In order to efficiently process point clouds, the neural network must be able to deal with three main properties of point clouds [48]: 1) Unordered: a point cloud is a set of unordered data. The neural network must be invariant to permutations of the point order in the input data; 2) Transformable: rotation and transformation of a point cloud do not change its semantic. The neural network must be invariant to geometric transformations, e.g. rotations and translations; 3) Locally relevant: Points in a point cloud are not isolated. Instead, they and their neighboring points form a meaningful subset. The neural network needs to be able to aggregate the feature from nearby points. The first two properties are quite challenging to deep neural networks. Earlier neural networks require a conversion of representation before feeding the point cloud.

The idea of directly extracting point-wise features from unordered point clouds stemmed from PointNet [48]. To deal with the unordered property, Qi et al. [48] point out that symmetry functions are invariant to permutations. For symmetry functions, such as addition and max function, changing the order of arguments will not change the function value. The authors further prove that the composition of multi-layer perceptron (implemented by 1x1 convolution with batchnorm and ReLU) and max pooling layers is a symmetry function and a universal approximator. Therefore this kind of composed structure can be used to extract point-wise features, while the max pooling can aggregate the global information from each unordered point, known as the unit PointNet. To deal with the transformable property, Qi et al. [48] use T-net, a lightweight neural network, to predict an affine transformation matrix and apply the transformation directly to the points or point-wise features. In terms of the locally relevant property, in PointNet, the authors simply concatenate the global feature from max pooling back to point-wise features.

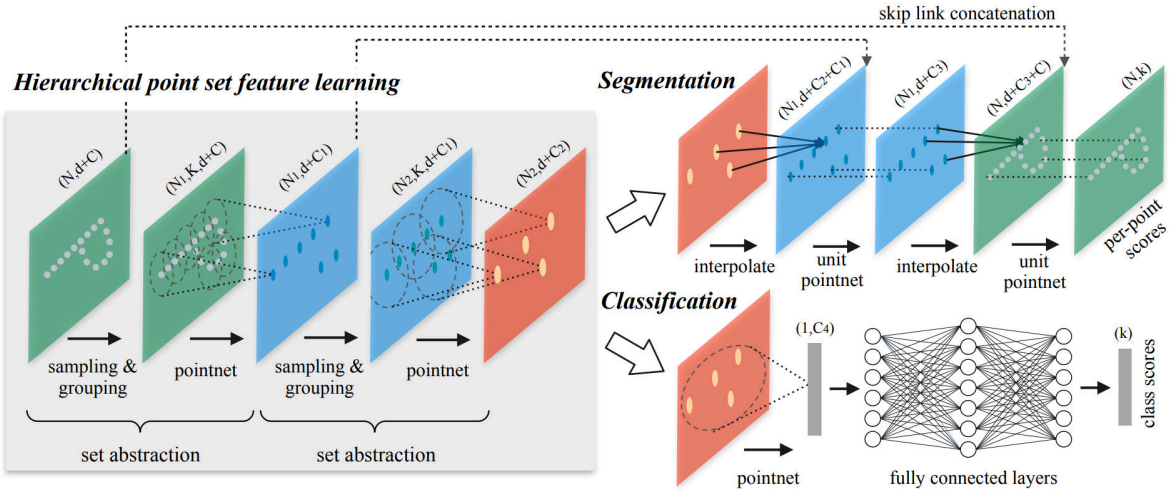

Figure 2.3: Architecture of PointNet++ [47].

In PointNet++ [47], a better way of extracting local structures is proposed as shown in Figure 2.3. The MLPs in PointNet are replaced by hierarchical set abstraction (SA) levels. A set abstraction level consists of a sampling layer, a grouping layer, and a PointNet layer. In the sampling layer, the iterative farthest point sampling (FPS) algorithm is applied to the input points and it outputs a subset of point features. For each subsampled point, the

grouping layer uses the ball query or K nearest neighbor (KNN) to find k neighboring points and aggregates their features and then outputs a set of grouped points. Then in the PointNet layer, the features encoding local information that are extracted from each group as the final output of a SA level. For the classification task, SA levels can be used as a feature extractor followed by fully connected layers for final class predictions. For tasks that require higher point resolution, e.g. segmentation where features of all points are necessary, the authors propose the feature propagation (FP) level to propagate features from subsampled points to the full point set. In a feature propagation level, a larger point set's coordinates are obtained by skip links from the corresponding SA level, see Figure 2.3. To interpolate the feature of one point in the the larger set, we find its k nearest neighbors in the smaller input set and then aggregate their point-wise features by inverse distance weighted average. These aggregated features are then concatenated with the feature of the point from skip links, followed by a PointNet layer refining the concatenated features. The same number of FP levels restores point-wise features to the original number, forming an encoder-decoder structure.

PointNet and PointNet++ are efficient point feature encoder. Almost all point-based methods take their variants as the feature encoder backbone, and many of the voxel-based methods also use their paradigm.

Two-stage Point-based Detection

PointRCNN [56] is one of the earliest point-based 3D object detector. Shi et al. [56] observe that 3D objects are usually separated without overlapping. Therefore, points inside a ground truth bounding box can be seen as foreground points of an object. The authors utilize this segmentation strategy for point-based detection: firstly using a PointNet++ backbone (with FP levels) to extract point-wise features from the raw point cloud, and doing a foreground segmentation based on learned features; then for each foreground point generating one 3D proposal by a bin-based scheme; after that, a point cloud ROI pooling is carried out to pool all the point-wise features learned before and extract 3D regional features; finally, a canonical transformation-based bounding box refinement process is applied to generate final outputs.

PC-RGNN [80] takes the first stage from PointRCNN, namely the backbone, foreground segmentation, and bin-based 3D proposal generation module. After that, Zhang et al. [80] propose a point cloud completion (PC) module to address the sparsity and occlusion issues in point clouds. It can enhance the sparse and partial point clouds inside 3D proposals to dense and entire point clouds. The PC module uses farthest point sampling and graph convolutional network (GCN) layers [65] to encode point-wise features of vertices and uses coarse-to-fine fully connected layers to decode the features. Furthermore, a GCN-based discriminator and adversarial Loss are also added to this module for better completion. GCNs can capture extra geometry information from the edge connection of points and therefore facilitate the completion. Finally, the authors use four attention-based multi-scale graph neural network (AMS-GNN) layers to encode completed object point clouds, and a global attention layer to predict refined bounding boxes.

Single-stage Point-based Detection

Point-GNN [59] also uses the graph neural network (GNN), but differently, it is a single-stage detector and the graph is constructed on the raw point cloud instead of points inside

a proposal. Points are vertices, and edges connect points with a distance less than a fixed radius. Initial features of vertices are extracted by a unit PointNet. The constructed graph is then processed by a message-passing framework-based GNN. The GNN iteratively extracts and aggregates vertex and edge features, and outputs a set of bounding box candidates. Then a non-maximum suppression (NMS) based box merging and scoring operation is applied to those candidates and output final predictions.

3DSSD [73] is the fastest point-based detector by far. The authors believe that the FP levels in the backbone and bounding box refinement process are limiting the speed of point-based methods. Nevertheless, removing FP levels will make the backbone unable to upsample and lose valuable points. To address this issue, the authors introduce feature farthest-point-sampling (F-FPS), which computes the feature distance for sampling, instead of Euclidean distance in traditional distance farthest-point-sampling (D-FPS). In the SA levels of 3DSSD, F-FPS and D-FPS each sample half of the points. This sampling strategy is named fusion sampling: F-FPS captures foreground points for bounding box prediction, while D-FPS captures background points for reliable classification. It can capture most of the useful points, and therefore FP levels are safely deleted. After the backbone, Yang et al. [73] proposes a candidate generation module: for those points that sampled from F-FPS, they are shifted to be closer to the object center by a MLP for better prediction. The shifting is supervised by the central location of the corresponding object, derived points called candidate points. Then point-wise features for these candidate points are aggregated and extracted from neighboring points sampled by both F-FPS and D-FPS. At last, the final predictions are generated by a simple anchor-free detection Head. 3DSSD achieves real-time performance by these improvements and opens a new chapter for point-based methods in the future.

2.2.2 Voxel-based Methods

In order to avoid processing huge amounts of points and reducing computational cost, voxel-based methods divide the point cloud into equally spaced 3D voxels. Then they extract point-wise features inside each voxel and aggregate them as the voxel feature. All subsequent operations take voxels as their unit. Voxelization not only reduces the number of features but also makes 3D convolution on point cloud data become possible.

Single-stage Voxel-based Detection

VoxelNet [88] is considered as the pioneer of voxel-based 3D detectors. The authors propose a classic end-to-end architecture, including a voxel feature encoder, middle convolutional layers, and a RPN detection head. In the feature learning network of VoxelNet, the whole 3D space is divided into non-overlapping fixed-size voxels, and each voxel contains a different number of points. Then a subsampling is applied on those voxels that contain more than threshold T points, in order to reduce computational cost and decrease the imbalance in the number of points. The points in the subsampled voxels are then fed into a set of stacked voxel feature encoding (VFE) layers, and the output is treated as the feature of the voxel. A VFE layer is similar to a unit PointNet, including MLPs (implemented by fully connected layers), max pooling, and concatenation. 3D convolutional middle layers operate on these encoded voxel features. During the convolution, the height of the voxel feature map shrinks to only 2 and is reshaped to 2D in the end. After 2D convolutional layers in the RPN, the

neural network outputs the final prediction. One problem in VoxelNet is, that more than 90% of voxels are empty, and traditional 3D convolutional operations on sparse tensors bring too much unnecessary computational cost. Therefore, VoxelNet can only run at 4 frames per second (FPS). SECOND [69] propose to use sparse convolutional middle extractor [24, 40], which greatly speeds up the inference time. Yan et al. [69] also propose a sine-error loss and a direction classifier to address the problem of confusing 0 and π radians in the yaw rotation value prediction.

PointPillars [33] is the fastest 3D object detector by far. It runs at 42 FPS and can be accelerated up to 62 FPS by TensorRT [46]. In PointPillars, the point cloud is divided into pillars (vertical columns), which are special voxels without partition in the z-direction. Similarly, the pillar feature net (PFN) uses unit PointNet to extract pillar features. The feature map of pillars can be treated as a pseudo-image, and therefore the 3D convolutional middle layers are replaced by 2D convolution. The 2D convolutional layers form a hierarchical structure for better feature learning. A SSD detection head and losses similar to SECOND is used for prediction and training. TANet [42] introduces a Triple Attention (TA) module to enhance the feature extraction, including channel-wise, point-wise, and voxel-wise attention. Adding the attention mechanism makes the model perform well in objects that are hard to detect, such as Pedestrians. The authors also propose a Coarse Regression (CR) module a Fine Regression (FR) module for 3D box estimation, in order to utilize both semantic and spatial information. The output feature map and intermediate feature maps of the CR module are provided to the FR module by sampling and concatenation. The output coarse prediction from the CR module is used as the new anchor boxes, while the CR module itself uses original anchor boxes.

SA-SSD [81] adds a detachable auxiliary network to the sparse convolutional middle layers to provide additional point-level supervision. The auxiliary network takes intermediate feature maps from the sparse convolution, converts them back to point-wise by computing the center of the corresponding voxel, and applies FP levels. The final feature map of the auxiliary network is used to predict a point-wise foreground segmentation and a center estimation task. The auxiliary network is optimized jointly with final prediction losses during training, but removed in testing and it brings no additional cost in inference time. SA-SSD also proposes a part-sensitive warping (PS-Warp) operation as an extra detection head. It can alleviate the misalignment between the predicted bounding boxes and classification confidence maps, since they are generated by two different convolutional layers in the detection head.

CIA-SSD [81] also notices the misalignment issue. Zheng et al. [81] design an IoU-aware confidence rectification module, using an additional convolution layer in the detection head to make IoU predictions. The predicted IoU value is used to rectify the classification score. CIA-SSD also proposes a Spatial-Semantic Feature Aggregation (SSFA) module to capture both spatial and semantic features. In addition, an improved distance-variant IoU-weighted NMS (DI-NMS) is used for bounding box filtering. SE-SSD [82] is the current state-of-the-art 3D detector. The model architecture is similar to CIA-SSD with only an additional Orientation-aware Distance-IoU loss (OD-IoU) in the detection head for better supervising the bounding box regression. The main contributions are the novel shape-aware data augmentation method and the self-ensembling teacher & student post-training architecture. ProPillars [87] is the baseline of this thesis. Zhou et al. [87] uses the triple attention mechanism in TANet, and adds an attentive addition module to the 2D middle convolution of PointPillars, followed by the detection head and PS-Warp extra head in SA-SSD. More details about CIA-SSD, SE-SSD, and ProPillars will be introduced in Section 3.1.

Two-stage Voxel-based Detection

Part-A^2 [57] uses a structure like U-Net [51] in its first stage. The downsampling process is followed by a RPN head similar to SECOND for 3D proposals, while the upsampling features are used for foreground segmentation and point-wise intra-object part location prediction (eight corners). Unlike the pooling approach in PointRCNN, in the second stage of Part-A^2, points in each 3D proposal are divided into smaller sub-voxels once again and pooling the features inside each sub-voxels. This method is named RoI-aware point cloud pooling. Then the segmentation score and part locations are aggregated for refinement.

Voxel R-CNN [15] observes that the point-voxel feature interaction in the second stage takes almost half of the total runtime. To speed up two-stage detectors, Deng et al. [15] propose a new voxel RoI pooling operation and an accelerated unit PointNet. In the voxel RoI pooling layer, a 3D proposal is also divided into smaller sub-voxels, but the center is treat as the grid point of the corresponding sub-voxel. Then, features from neighboring sub-voxels are aggregated to the grid point by a newly designed grid voxel query, instead of the ball query in traditional RoI pooling. These improvements make Voxel R-CNN reach 25 FPS in the inference speed.

2.2.3 Hybrid Methods

Hybrid methods aim to take advantage of both point-based and voxel-based methods. Point-based methods have a higher spatial resolution but involve higher computational cost, while voxel-based methods can efficiently use CNN layers for feature extraction but lose local point-wise information. Hybrid methods try to strike a balance between them.

HVPR [45] is a single-stage detector. It has two feature encoder streams extracting point-wise and voxel-wise features. Extracted features are integrated together and scattered into a pseudo image as hybrid features. An attentive convolutional middle module is performed on the hybrid feature map, followed by a single-stage detection head.

STD [72] is a two-stage detector that uses PointNet to extract point-wise feature. Yang et al. [72] design a point-based proposal generation module with spherical anchors to achieve high recall. Then a PointsPool module voxelizes each proposal, followed by a VFE layer. In the box refinement module, CNNs are applied on those voxels for final prediction. PV-RCNN [58] uses the 3D sparse convolution for voxel feature extraction. A Voxel Set Abstraction (Voxel-SA) module is added to each convolutional layer to encode voxel features into a small set of key points, which are sampled by farthest point sampling. Key point features are then re-weighted by foreground segmentation score. Finally, they are used to enhance the ROI grid points for refinement. $H^2$3D R-CNN [16] extracts point-wise features from multi-view projection. It projects the point cloud to bird's eye view (BEV) under Cartesian coordinates and perspective view (PV) under the cylindrical coordinate, separately. BEV feature and PV feature are concatenated together for proposal generation in BEV and fused together by the Bilaterally Guided Multi-View Fusion (BGMVF) module as point-wise hollow-3D (H3D) features. Then voxelization is performed on the 3D space, and point-wise H3D features are aggregated as voxel-wise H3D features for the refinement process. $H^2$3D R-CNN argues that, in the Voxel RoI Pooling [15], the voxel query with a different query range may result in

the same grid points, which may undermine the significance of multi-scale grouping. Moreover, the computational cost grows cubically with the voxel number and query range. Deng et al. [16] therefore propose an improved version of Hierarchical Voxel RoI Pooling (HV RoI Pooling). It divides the sub-voxels in different granularity, coarse sub-voxels with large query range and fine sub-voxels with smaller query range. Coarse and fine ROI features are concatenated for final prediction. As a two-stage detector, $H^2$3D R-CNN achieves an extraordinary inference speed with decent accuracy.

	Method	Reference	Modality	3D Cars			Time(ms)	Hardware
				Easy	Mod	Hard		
two-stage	PointRCNN[56]	CVPR 2019	points	86.96	75.64	70.70	100	TITAN XP
	Fast PointRCNN[12]	ICCV 2019	points	85.29	77.40	70.24	65	TESLA P40
	STD[72]	ICCV 2019	hybrid	87.95	79.71	75.09	80	TITAN V
	Part-A^2[57]	TPAMI 2020	voxels	87.81	78.49	73.51	80	TESLA V100
	PV-RCNN[58]	CVPR 2020	hybrid	90.25	81.43	76.82	80	GTX 1080 Ti
	PC-RGNN[80]	AAAI 2021	graphs	87.94	81.38	76.88	100	GTX 1080 Ti
	P2V-RCNN[34]	IEEE 2021	hybrid	88.34	81.45	77.20	100	RTX 3090
	Voxel R-CNN[15]	AAAI 2021	voxels	**90.90**	81.62	77.06	40	GTX 2080 Ti
	$H^2$3D R-CNN[16]	IEEE 2021	multiview	90.43	81.55	77.22	**27**	RTX 2080 Ti
	BtcDet[16]	AAAI 2022	voxels	90.64	**82.86**	**78.09**	90	GTX 1080 Ti
single-stage	VoxelNet[88]	CVPR 2018	voxels	77.82	64.17	57.51	220	TITAN X
	SECOND[69]	Sensors 2018	voxels	87.44	79.46	73.97	50	GTX 1080 Ti
	PointPillars[33]	CVPR 2019	pillars	82.58	74.31	68.99	**24**	GTX 1080 Ti
	TANet[42]	AAAI 2020	voxels	84.39	75.94	68.82	35	TITAN V
	HotSpotNet[10]	ECCV 2020	voxels	87.60	78.31	73.34	40	TITAN V100
	Point-GNN[59]	CVPR 2020	graphs	88.33	79.47	72.29	643	GTX 1070
	3DSSD[73]	CVPR 2020	points	88.36	79.57	74.55	38	TITAN V
	SA-SSD[26]	CVPR 2020	voxels	88.75	79.79	74.16	40	GTX 2080 Ti
	HVPR[45]	CVPR 2021	hybrid	86.38	77.92	73.04	36	GTX 2080 Ti
	CIA-SSD[81]	AAAI 2021	voxels	89.59	80.28	72.87	30	TITAN XP
	SE-SSD[82]	CVPR 2021	voxels	**91.49**	**82.54**	**77.15**	30	TITAN XP

Table 2.1: Comparison of LiDAR-only 3D object detectors on the KITTI test set for car detection. The performance is evaluated by mean average precision (mAP) with an IoU threshold of 0.7 in three difficulty levels separately. The inference time is measured in milliseconds on different hardware.

2.3 Point Cloud Registration

Point clouds captured by LiDARs or depth cameras provide an accurate measurement of the real world. In contrast to images, the scale of captured objects does not vary with distance. However, these sensors are limited by occlusion, their sensor range and field of view. In order to model the whole scene, multiple sensors can be used to capture point clouds from different views. Point cloud registration is the process of computing a transformation matrix and transforming the source and target point cloud into the same coordinate system. With point cloud registration, we can merge the point clouds from multiple sensors into one and build a complete point cloud, an example shown in Figure 2.4.

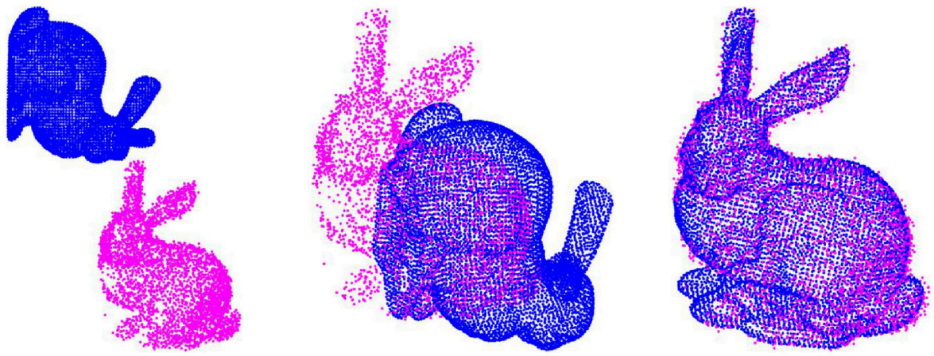

Figure 2.4: Example of point cloud registration [70]. Two point clouds are combined as one.

Methods of point cloud registration can be divided into two categories: traditional optimization-based methods and emerging learning-based methods. Optimization-based methods treat point cloud registration as a non-convex optimization problem. Typically, they have two stages: the correspondence searching and transformation estimating [29]. Correspondences are two points that have the same position but from two different point clouds. Learning-based methods use a neural network to extract features from the source and target point cloud, and the output could be directly the transformation in an end-to-end pattern, or the predicted correspondences for optimization-based transformation estimating in the next step. Learning-based methods are more robust but require a huge amount of training data. Optimization-based methods have higher computational efficiency and are free of training data, but may fall into a local optimum.

2.3.1 Optimization-based Methods

Iterative Closest Point (ICP) [5] is the most classic point cloud registration algorithm. The first step of ICP is finding correspondences by matching the closest point between the source and target point cloud. According to the correspondences and their center of mass, ICP computes a rotation and a translation matrix by performing singular value decomposition (SVD) on the correlation matrix. Then the computed transformation is applied to the source point cloud. After that, computing the root mean square error (RMSE) between correspondences, if the error is larger than the maximum matching threshold, the algorithm iteratively repeats these steps until convergence.

Chen and Medioni [11] argue that the traditional point-to-point objective function depending

on pairs of correspondences is unreliable and inefficient, because correspondences are simply computed by closest points. They propose to use the point-to-plane error: by computing the distance between a point and the tangent plane of its correspondence point. That reduces the number of iterations and increases the accuracy. Brenner et al. [8] propose a more efficient plane-to-plane error function, which treats point clouds as point-sampled 2D manifolds and uses local surface normal to represent points in both source and target point cloud.

The closest point matching process is the most time-consuming part in standard ICP. The K-D tree algorithm [4] is often applied to speed it up. For further acceleration, Rusinkiewicz and Levoy [52] propose to use a sampled subset in the correspondence searching, instead of using the whole point cloud. LM-ICP [20] proposes to use the Levenberg–Marquardt (LM) algorithm for optimizing the RMSE function. The LM algorithm adaptively varies the parameter updates between the gradient descent and the Gauss-Newton method [21]. By adding the LM algorithm, LM-ICP has stronger robustness without significant loss of speed. LM-ICP also uses the chamfer distance transform to replace the K-D tree for closest point searching, which reduces the influence of the initial position of point clouds on its registration result.

Generalized-ICP [54] provides a probabilistic view on the objective function of ICP. The transformation is regarded as a parameter in the distribution of the residual between correspondences. In this probabilistic framework, point-to-point ICP is a special case of point-to-plane ICP, and plane-to-plane ICP is a generalized application. Sparse ICP [7] uses sparsity-inducing norms to further improve the robustness and the performance on outliers and incomplete data.

One limitation of ICP algorithms is, they can only process point clouds that are relatively close to each other. The aforementioned ICP variants manage to alleviate the influence of the initial position and are more robust. Methods in recent years are trying to jump out of ICPs and break the limits. 4PCS [1] use the 4-points sets method to build geometric features for correspondence searching, and migrate RANSAC algorithm [19] to the field of 3D point cloud registration. CPD [44] is based on Gaussian mixture model (GMM). It treats the source point cloud as the centroids of GMM, the target as the data points, and then uses centroids to fit the data by maximizing the likelihood. The optimization is efficiently done by the expectation-maximization (EM) algorithm. Fast Global Registration (FGR) [85] adds Geman-McClure penalty to the cost function and applies graduated non-convexity (GNC) for optimization. Go-ICP [71] is the first global ICP variant. It uses the branch-and-bound algorithm to search in the entire 3D motion space. The distance transform algorithm is used for finding the closest points. TEASER [70] is a certifiable registration algorithm. It can provide checkable conditions to verify whether the result is optimal. The authors use a truncated least squares (TLS) cost function, which can reduce the impact of outliers. They also propose a graph theory-based framework to decouple scale, rotation, and translation estimation, so that they can be solved in cascade.

2.3.2 Learning-based Methods

In the field of learning-based point cloud registration, there are three widely used datasets: ModelNet40 [66] contains 13K 3D CAD models for objects in 40 categories; 3DMatch Dataset integrates data from multiple datasets and contains 200K RGB-D images; with the develop-

ment of learning-based methods, KITTI [22] is also used in recent methods for testing the efficiency on large point clouds.

Earlier learning-based methods use a neural network to extract features only for correspondence searching, and then use a traditional algorithm, e.g. SVD or RANSAC, to compute the transformation matrix. 3DMatch [79] is one of the earliest learning-based methods. It takes the idea of voxelization and applies 3D convolution on the divided 3D region. The source and target point cloud are fed into a siamese neural network for feature extraction. Voxels with similar features are regarded as correspondences. Unlike ICPs, learning-based methods directly generate the final correspondence pairs, without iteration. Then a RANSAC algorithm is applied for computing the transformation matrix. OctNet [50] uses octrees to deal with the sparsity of point cloud and to reduce the cost of convolutional operations. OctNet hierarchically divides the 3D space into a set of octrees, where branches with more data points have deeper depth. 3DSmoothNet [23] uses the smoothed density value (SDV) voxelization for convolutional layers. SDV is acquired by firstly aligning the point clouds using an estimated local reference frame (LRF), then applying Gaussian smoothing on original voxels. Since PointNet [48] was proposed, point-based registration methods become more and more popular. PPFNet [13] uses PointNet to extract point-wise features and match similar points as correspondences. DCP [64] applies a graph convolutional neural network for point-wise feature extraction. SiamesePointNet [84] proposes a novel siamese network architecture and performs point convolution operation (PointCNN) [36] on point clouds.

In the past three years, some end-to-end learning-based registration methods were also proposed. They directly output the final transformation, instead of only providing correspondences for further optimization-based transformation estimation. Deng et al. [14] treats the registration as a regression problem. They design an autoencoder structure for point-wise feature extraction and correspondences searching, and then they use a downstream fully connected network to learn correspondence-specific features and output final transformation. DeepVCP [43] uses PointNet++ [47] to subsample key points from the point cloud and extract features. A Deep Feature Embedding module is used for aggregating local features from neighboring points, and a Corresponding Point Generation module is responsible for generating correspondences. DeepGMR [78] is a neural network implementation of the Gaussian mixture model. It trains a neural network to do the fitting step in GMM and uses two following differentiable parameter-free blocks to compute a closed-form of GMM parameters and the transformation.

2.4 Point Cloud Upsampling

Point clouds suffer from the sparsity issue, which leads to limited points projected on objects that we are interested in. Especially when an object is at a longer distance from the sensor, there might be, for example only five points on that object. Insufficient points lead to insufficient features. That increases the difficulty of computer vision tasks, e.g. object detection, segmentation, etc. Increasing the number of points becomes a problem demanding prompt solution. Point cloud upsampling using a neural network is one of the ways to increase the density of point clouds.

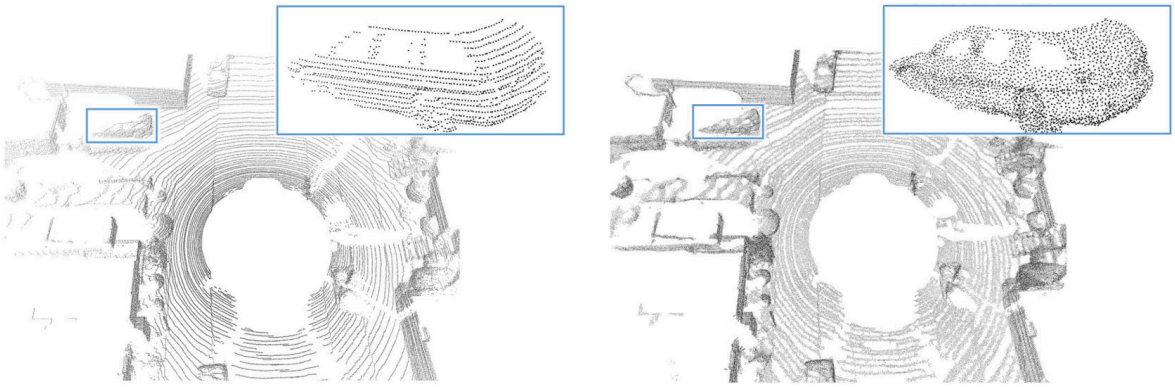

Figure 2.5: Example of point cloud upsampling [35]. Left: raw point cloud. Right: upsampled point cloud.

PointNet [48] and PointNet++ [47] facilitates point-level feature extraction and therefore makes the point-level task — point cloud upsampling become possible. PU-Net [77] firstly randomly samples a set of points and groups neighboring points as surface patches. Patches are fed into a PointNet++ based module for point feature extraction and aggregation. Patches and feature aggregation can capture local geometry patterns. Then PU-Net uses an efficient feature expansion operation based on the sub-pixel convolution layers to expand extracted point-wise features to r times larger, where r is the upsampling rate. After that expanded features are fed into fully connected layers, and the output is reconstructed coordinates of upsampled points.

MPU [75] proposes a progressive cascading of patch-based upsampling networks. It extracts features on different levels of detail and in multiple steps, with intra and inter skip connections. PU-GAN [35] utilize the generative adversarial network (GAN) structure. In the generator, it expands the features to $r+2$ times and then uses farthest sampling to sample them to the upsampling rate. The self-attention unit is applied in both generator and discriminator, to better aggregate the whole context of the point cloud. PU-GCN [49] designs a graph convolutional network for upsampling. The NodeShuffle module is proposed to better encode the local information between neighboring points. It applies a shuffle operation to features extracted from the GCN, and the $N \times rC$ dimension feature is rearranged to $rN \times C$ for upsampling. The previous methods are trained on a specific upsampling factor, and changing factors requires retraining the neural network. Differently, Meta-PU [74] can deal with arbitrary scale factors. In the backbone, the PointCNN [36] and residual graph convolutional (RGC) blocks are used for feature extraction. Then the meta-subnet takes the upsampling rate as input and outputs the graph convolutional weights tensor for the designed meta-RGC blocks. In this way, the neural network can adapt to different factors.

Inspired by the image super-resolution problem, Shan et al. [55] propose to increase the density of point clouds by extending the number of channels. More specifically, given a point cloud captured by a 32-channel or 16-channel LiDAR, the neural network outputs a 64-channel point cloud. By this method, the density of the point cloud gets a relatively regular increase, more details in Section 3.3.

Chapter 3

Solution Approach

In this chapter, we propose our solutions to those problems mentioned in Section 1.2. In Section 3.1 we present the architecture of our 3D object detector. In Section 3.2, we introduce the point cloud registration algorithm we used for infrastructure LiDARs. Section 3.3 presents the data simulation environment and our point cloud super-resolution model. In Section 3.4 we describe the two visualization tools that we implemented. In Section 3.5 we show the auto label function we added to the labeling tool ProAnno.

3.1 Proposed 3D Object Detector

In this section, we present SE-ProPillars, a LiDAR-only single-stage pillar-based 3D object detector. ProPillars [87] is an improved detector based on PointPillars [33], with the additional triple attention mechanism [42] and PS-Warp detection head [26]. We take ProPillars as our baseline and make further improvements. We greatly increase the inference speed of ProPillars, and at the same time, the accuracy also gets appreciable improvements.

3.1.1 Network Overview

The overview architecture of SE-ProPillars is shown in Figure 3.1. The input to the detector is raw point clouds. The 3D space is divided into pillars (vertical columns). The stacked triple attention module extracts features from the raw point cloud using the triple attention mechanism, including channel-wise, point-wise, and voxel-wise attention. That enhances the learned features. Then, the pillar feature net turns point-wise features into pillar features and scatters the pillar features into a pseudo image. The attentive hierarchical middle layers perform 2D convolution operations on the pseudo image. Hierarchical feature maps are concatenated with attentive addition. Finally, the multi-task head is used for the final prediction. The multi-task head makes an additional IoU prediction to alleviate the misalignment between the localization accuracy and classification confidence.

In comparison to ProPillars, we keep the stacked triple attention module and the pillar back-

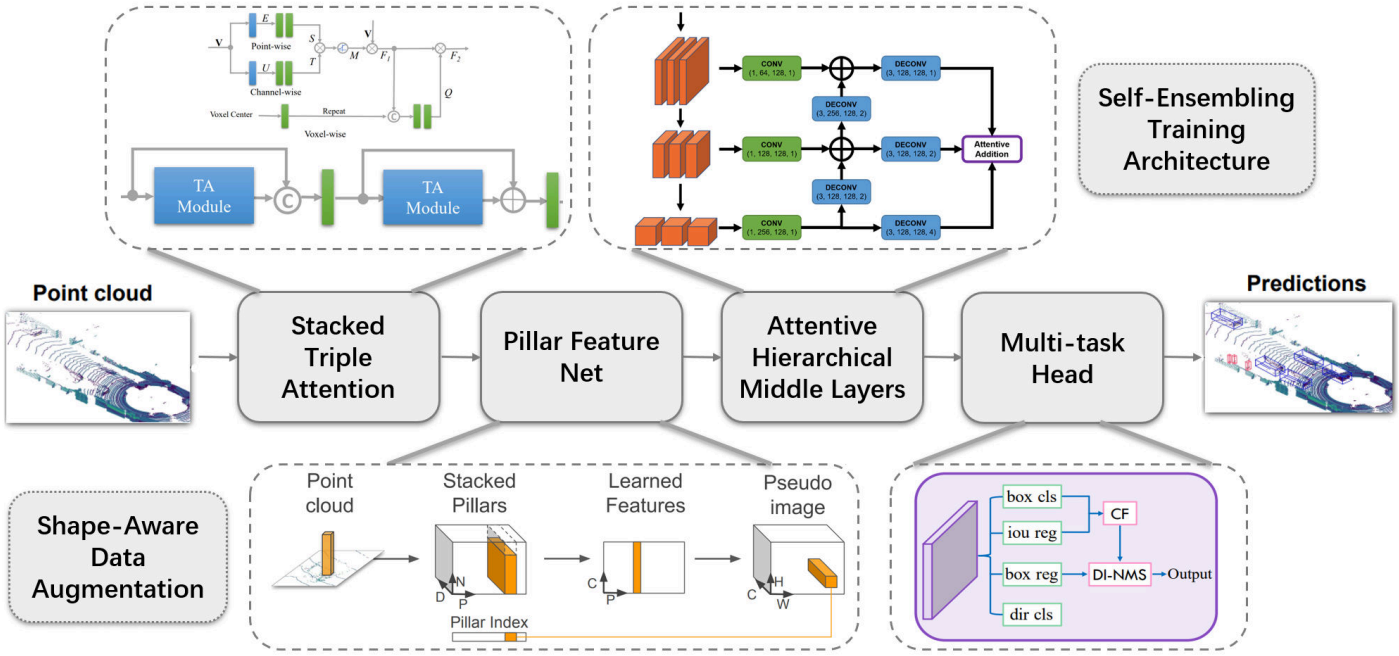

Figure 3.1: Overview architecture of SE-ProPillars.

bone, because they ensure good feature extraction power and high computational efficiency respectively, but we replace the heavy detection head with the lightweight multi-task head for solving the misalignment problem. The new detection head greatly reduces the inference time. In addition, we introduce two training techniques in the training process: the shape-aware data augmentation module and the self-ensembling teacher & student training framework. These two tricks lead to a substantial increase in the precision without any additional cost in the inference time.

3.1.2 Voxelization

Before feeding into the neural network, we first divide the raw point cloud into pillars, a special voxel that has no splitting in the vertical direction. The advantages of using pillars instead of voxels are: firstly, a pillar-based backbone is faster than a voxel-based backbone due to fewer grids; secondly, the pillar representation eliminates the demand for time-consuming 3D convolutional middle layers, so that we could use 2D operations instead; finally, we do not need to manually tune the hyperparameter — the bin size in the z-direction.

If a pillar contains more points than a threshold, then we subsample the points to the threshold using farthest point sampling. If a pillar contains fewer points than the threshold, then we pad it with zeros to make the dimensions consistent. Due to the sparsity issue of point clouds, most of the pillars are empty. We record the coordinate of non-empty pillars according to the pillars' center and their index. If a pillar is empty, we ignore them during the feature extraction, until we scatter all pillars back to a pseudo image for 2D convolution.

3.1.3 Stacked Triple Attention

TANet [42] proposes the stacked triple attention (TA) module for more powerful feature learning on hard to detected objects and for better dealing with noise points. This method can be applied on both voxel and pillar-divided point clouds, while we prefer the speed and convenience that pillar-based backbone brings.

The attention mechanism in this module follows the Squeeze-and-Excitation pattern [28]. Specifically, if we want to apply channel-wise attention to an input tensor with shape ($H \times W \times C$), we firstly use a global pooling operation to pool the tensor to shape ($1 \times 1 \times C$), called squeeze operation. Then we applied two fully connected layers to the squeezed tensor — attention score, called excitation operation. Between the two FC layers, the feature dimension is reduced and then recovered with a reduction ratio, which forms a bottleneck structure. After that, we perform a sigmoid function to get the attention scale. Finally we element-wisely multiply the ($1 \times 1 \times C$) scale to the original ($H \times W \times C$) feature.

The input to the module is a ($P \times N \times C$) tensor, where P is the number of non-empty pillars, N is the maximum number of points in each pillar, C is the dimension of the input point-wise feature. At the beginning, $C = 9$, including $(x, y, z, r, x_c, y_c, z_c, x_p, y_p)$, where x, y, z are the coordinates of the point, r is the intensity, x_c, y_c, z_c are the distance to the arithmetic mean of all points inside the pillar, x_p, y_p are the location of the pillar from the pillars' center.

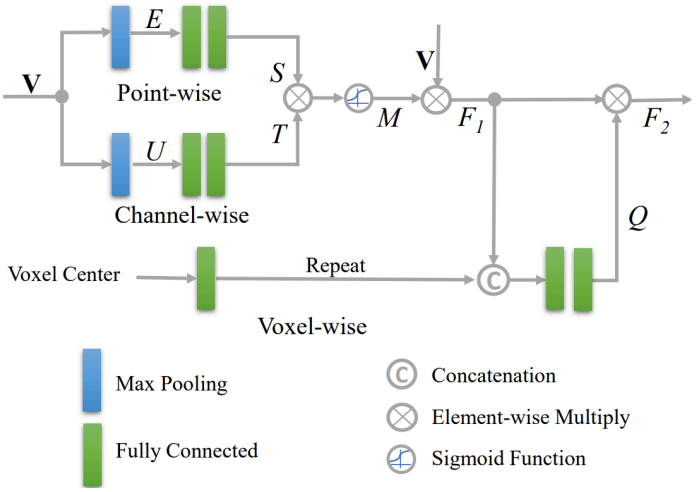

Figure 3.2: Structure of triple attention module [42].

The structure of the triple attention module is shown in Figure 3.2. The TA module extract features inside each pillar, using point-wise, channel-wise, and voxel-wise attention. Input V is fed to the module. In the upper branch point-wise attention, we follow the Squeeze-and-Excitation pattern, firstly performing max pooling to aggregate point-wise features across the channel-wise dimensions, and then we compute the point-wise attention score S using two fully connected layers (a ReLU function and bottleneck reduction in between). Similarly, the middle branch channel-wise attention aggregates channel-wise features across their point-wise dimensions, got channel-wise attention score T. Then S and T are combined by element-wise multiplication, followed with a sigmoid function to get the attention scale matrix M, $M = \sigma(S \times T)$. M is then multiplied with input V, got feature tensor $F1$. In the bottom branch voxel-wise attention, the C-dim channel-wise feature in $F1$ is enlarged by the voxel

center (arithmetic mean of all points inside the pillar) to $C+3$-dim for better voxel-awareness. Then the enlarged $F1$ is fed into two fully connected layers. The two FC layers respectively compress the point-wise and the channel-wise dimensions to 1, to get the voxel-wise attention score. Finally, a sigmoid function generates the voxel-wise attention scale Q, multiplied with the original $F1$ to generate the final output of the TA module $F2$.

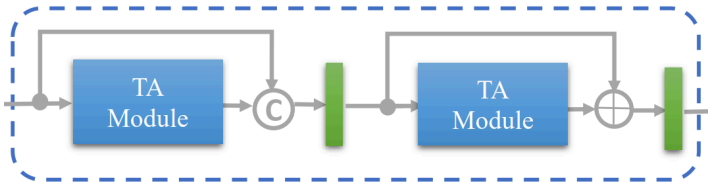

Figure 3.3: Structure of stacked triple attention module [42].

In order to further exploit the multi-level feature attention, two triple attention modules are stacked with a structure similar to the skip connection in ResNet [27], called stacked TA module, shown in Figure 3.3. The first TA module takes raw point cloud 9-dim features as input, while the second one works on the extracted high dimensional features. For each TA module, the input is concatenated or summed to the output to fuse more feature information. Each TA module is followed by a fully connected layer. They are used for increasing the feature dimension, because inside TA modules, the attention mechanism only re-weights features but does not increase dimensions.

3.1.4 Pillar Feature Net

The pillar feature net (PFN) is proposed in PointPillars [33]. It takes pillars as input, extracts pillar features, and scatters pillars back to a pseudo image for 2D convolution operations in the middle layers. In our case, we have already encoded high-dimensional features in the stacked TA module. Therefore, we use the PFN for further feature extraction.

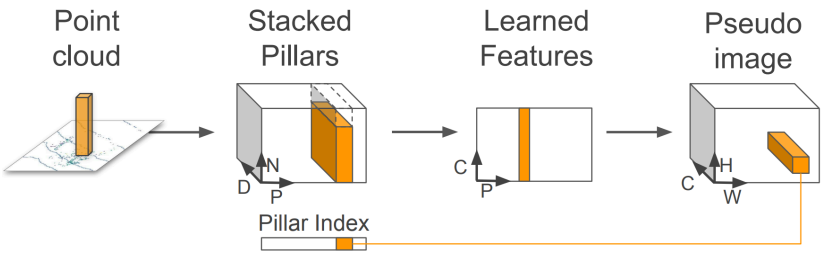

Figure 3.4: Structure of pillar feature net [33].

The structure of the PFN is shown in Figure 3.4. The point-wise pillar-organized features from the stacked TA module with shape $(P \times N \times C)$ are fed to a set of PFN layers. Each PFN layer is a simplified unit PointNet [48], which consists of a linear layer, batchnorm, ReLU, and max pooling. The max-pooled features are concatenated back to the ReLU's output, to keep the point-wise feature dimension inside each pillar, until the last PFN layer. While the last PFN layer makes the final max pooling and outputs a $(P \times C)$ feature as the pillar feature. Pillar features are then scattered back to the original pillar location, forming a $(C \times H \times W)$ pseudo image, where H and W are the height and width of the pillar grid. Here the location of empty pillars is padded with zeros.

3.1.5 Attentive Hierarchical Middle Layers

Traditional voxel-based methods have partitions in the vertical direction. Therefore, they need a sparse 3D convolutional middle layer. In our case, the 2D convolutional operation is satisfactory. The computational cost saved by using 2D operations allows for adding more advanced structures.

Zhou et al. [87] propose the attentive hierarchical middle (AHM) layers to perform 2D convolution on the pseudo image from the pillar feature net. Figure 3.5 depicts the structure of AHM layers. In the first stage, the spatial resolution of the pseudo image is gradually downsampled by three groups of convolution. Each group contains three convolutional layers, where the first one has a stride of two for downsampling, the two subsequent layers only for feature extraction. After downsampling, deconvolution operations are applied to recover the spatial resolution. Deconvolutional layers pointing up recover the size of feature maps with stride 2 and element-wise add them to upper branches. Deconvolutional layers pointing right together make all three branches have the same size (half of the original feature map). Then three branches' feature maps are combined by the attentive addition, to fuse both spatial and semantic features.

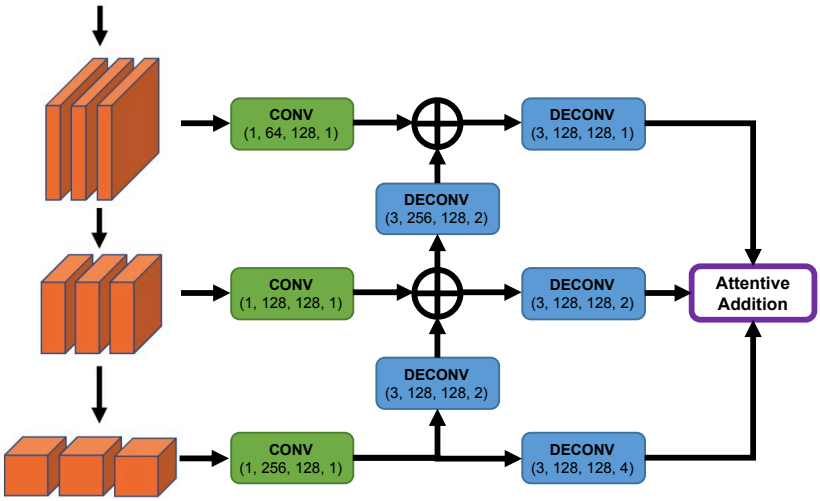

Figure 3.5: Structure of attentive hierarchical middle layers.

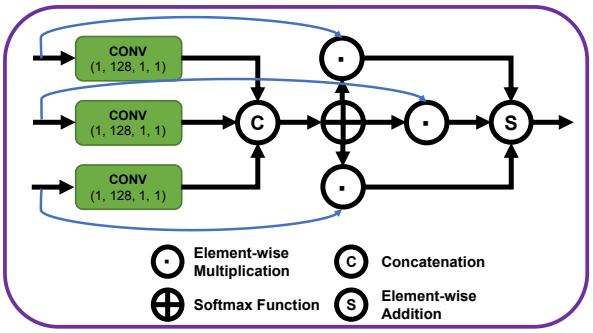

Figure 3.6: Structure of attentive addition.

The attentive addition is shown in Figure 3.6. It uses the plain attention mechanism. Feature maps from three branches pass through convolution and are channel-wise concatenated together as attention scores. The softmax function generates the attention distribution, and

feature maps are multiplied with the corresponding distribution weight. The element-wise addition gives the final attention output, a $(C \times H/2 \times W/2)$ feature map.

3.1.6 Multi-task Head

The multi-task head makes the final prediction based on the feature map from attentive hierarchical middle layers. Four convolutional layers operate on the feature map separately. Figure 3.7 shows the brief structure. The "box cls" convolution generates the box classification confidence score. The "box reg" convolution generates the bounding box regression result: $(x, y, z, l, w, h, \theta)$, where θ is the yaw rotation. The "dir cls" generates the direction classification to solve the problem that the sine-error loss [69] cannot distinguish flipped boxes. These three are the traditional 3D detection predictions.

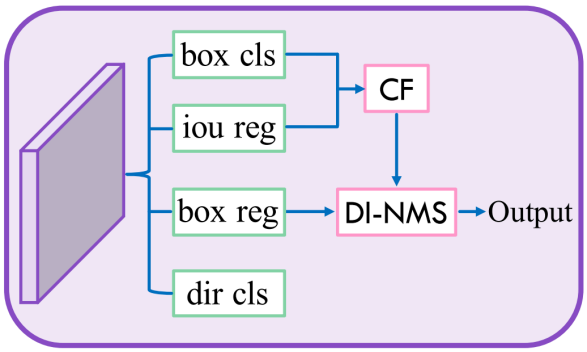

Figure 3.7: Structure of multi-task head [81].

The "iou reg" generates an IoU prediction, which is a predicted IoU value between the ground truth bounding box and predicted bounding box. It is introduced in CIA-SSD [81], for dealing with the misalignment between the predicted bounding boxes and corresponding classification confidence maps. The misalignment is mainly because these two predictions are from different convolutional layers. Compared to the PS-Warp extra head, using only one convolutional layer for the confidence correction is simpler and brings less computational cost. Based on this IoU prediction, we use the confidence function (CF) to correct the confidence map and use the distance-variant IoU-weighted NMS (DI-NMS) module to post-process the predicted bounding boxes.

From experiments on the KITTI dataset, Zheng et al. [81] find that the predicted IoU varies in a larger range when the real IoU is low (calculated IoU between ground truth and anchor-based predicted bounding box), shown on the left side of Figure 3.8. That is because when the anchor is far away from the ground truths, the learned feature contains insufficient information, and therefore the IoU prediction has high uncertainty. When we apply the exponential function on the predicted IoUs, the variation range shrinks, shown on the right side of Figure 3.8.

In order to suppress the uncertainties of low-IoU predictions, and further augment the discrimination between low-IoU and high-IoU predictions, Zheng et al. [81] introduce the rectification item g: $g = i^{\beta}$, where i is the predicted IoU, β is a hyperparameter that controls the degree of suppression. After the rectification, the predicted IoU can better reflect the real IoU. Therefore, it can be used to deal with the misalignment between the localization

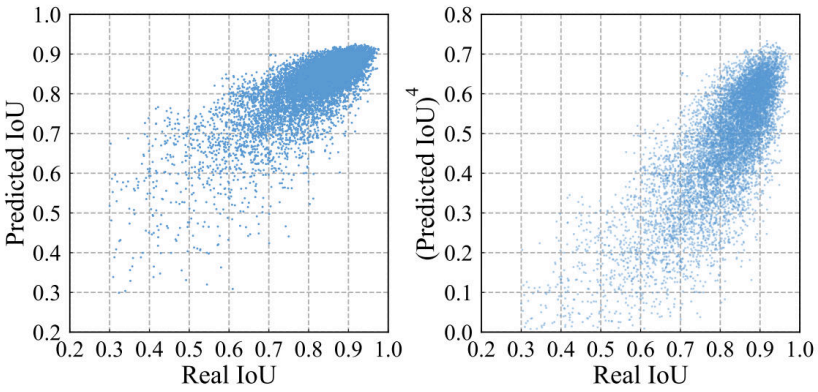

Figure 3.8: Experimental results on the KITTI dataset [81]. Left: real IoUs vs. predicted IoUs, high predicted IoUs are often related to high real IoUs, but predicted IoUs vary a lot when real IoUs are low. Right: real IoUs vs. (predicted IoUs)$^{\beta}$ with β = 4, the variation range of predicted IoUs are suppressed by the exponential function.

accuracy and classification confidence, i.e., for a predicted bounding box with low predicted IoU, we should be less confident. The confidence function is formulated as:

$$f : s_{rectified} = c \cdot g = c \cdot i^{\beta}, \tag{3.1}$$

where c is the classification score of the predicted bounding box. Note that, this rectification is only applied in the test time. During training, we use plain classification score, and when the IoU branch updates, we detach the predicted bounding boxes from the computational graph to avoid the IoU prediction loss backpropagate to the box branch.

The distance-variant IoU-weighted NMS is designed for dealing with long-distance predictions, to better align far bounding boxes with ground truths, and reduce false-positive predictions. The procedure is as follows:

1. We calculate the L_2 distance between the BEV center of each pair of anchor and predicted box, and use this distance offset to further refine the rectified confidence:

$$s_{refined} = s_{rectified} \cdot (1 - softmax(L_2 dist)).$$

2. Similar to the plain NMS, given a collection of generated boxes L, we use the refined confidence score $s_{refined}$ for selecting the candidate box $c^{'}$ with the highest confidence, and we compute a set of auxiliary boxes $L^{'}$ which overlap with $c^{'}$ and have an IoU greater than the IoU threshold.

3. We compute a cumulative sum term of products between the predicted IoU and the real IoU of candidate and auxiliary boxes:

$$cnt = \sum_{l_i \in L^{'}} iou_i \cdot IoU(l_i, c^{'}),$$

where l_i is an auxiliary box in $L^{'}$, iou_i is the predicted IoU for l_i, $IoU()$ is computing the real IoU between l_i and $c^{'}$. This term is used for filtering out false-positive predictions in Step 5.

4. We use a Gaussian weighted average to obtain a smooth bounding box from the auxiliary boxes:

$$box = \frac{\sum_{l_i \in L'} iou_i \cdot l_i \cdot e^{-(1-IoU(l_i,c'))^2/\sigma^2}}{\sum_{l_i \in L'} iou_i \cdot e^{-(1-IoU(l_i,c'))^2/\sigma^2}}, \sigma \propto \|l_i, [0,1]\|_2,$$

where σ is a depth factor positively correlated with the BEV distance between the box center and viewpoint.

5. If the cnt term is larger than the cnt threshold, then we get a final output bounding box and we remove L' from L, like naive NMS.

6. Repeat until L is empty.

The depth factor σ in the Gaussian weighted average works as: If the predicted box is near to the origin of perspective, then we give higher weights to those box predictions with high IoU. That is because near objects are easier to detect and we are more confident. If the predicted box is far, then we give all auxiliary boxes relatively uniform weights, to get a more smooth final box. By this factor, we could better utilize distant auxiliary boxes which are often oscillated around.

3.1.7 Shape-Aware Data Augmentation

Data augmentation is proved to be an efficient way to improve the neural network model. It can better exploit the training dataset and help the model to be more generalized. SE-SSD [82] designs the shape-aware data augmentation module, shown in Figure 3.9. This module aims to mimic the common challenges in 3D point cloud object detection, e.g. sparsity, partial occlusion, different shapes of objects in the same class, etc.

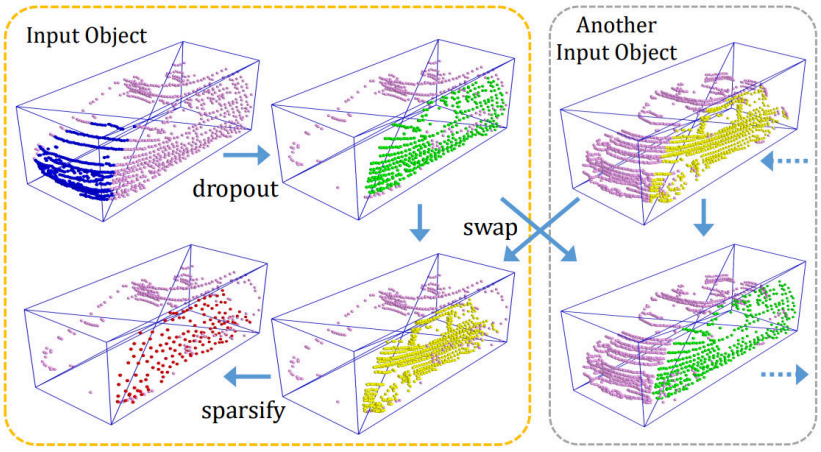

Figure 3.9: Illustration of the shape-aware data augmentation module [82]. Including random dropout in the blue subset, random swap between green and yellow subsets, random sparsifying from the yellow to the red subset.

The ground truth box is divided into six pyramidal subsets. Then we independently augment each subset using three operations: 1) random dropout: randomly remove all points in a subset, to mimic partial occlusions; 2) random swap: randomly take a corresponding subset from another object and swap them, to mimic different shapes of objects in the same class;

3) random sparsifying: randomly subsample points in a subset, to mimic the sparsity of point clouds. Some traditional global operations are also applied before the shape-aware augmentation, e.g. rotation, flipping, scaling, etc. These traditional augmentation methods are called global transformations in the next section. After augmentation, the model's capacity to detect difficult objects is significantly improved.

3.1.8 Self-Ensembling Training Framework

Besides the aforementioned modules, we also introduce the self-ensembling training framework [82] to do a post-training: we firstly train the model without self-ensembling, and then we take the pre-trained model as the teacher model to train a student model that has the same network structure. The predictions from the teacher model can be used as soft supervision. Combined with the hard supervision from the ground truth, we could provide more information to the student model. The framework of the self-ensembling training is shown in Figure 3.10.

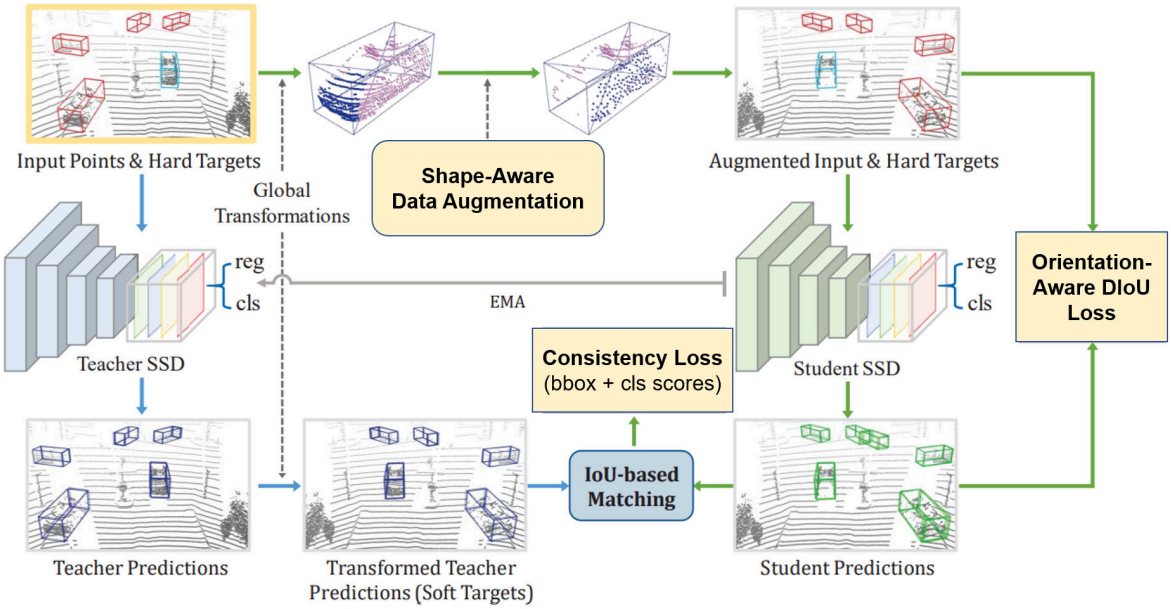

Figure 3.10: Framework of self-ensembling training [82]. Blue arrow shows the process of generating soft targets. Green arrow shows the generation of student predictions. OD-IoU loss replaces the traditional regression loss. With the consistency loss and other losses in the multi-task head, they together supervise the student model.

More specifically, the teacher model and the student model are initialized by the same pre-trained parameters. During training, we firstly feed the raw point cloud to the teacher model and get teacher predictions. Then we apply global transformations to the teacher predictions as soft targets. For the hard targets (ground truths), we apply the same global transformations and additionally the shape-aware data augmentation. After that we feed the augmented point cloud to the student model and get student predictions. The orientation-aware distance-IoU (OD-IoU) loss is introduced in the hard supervision, to better align the box centers and orientations between the student predictions and hard targets. Finally, we use an IoU-based matching to match student and teacher predictions. We use the consistency loss to provide soft supervision to the student model. During the post-training, the parameters of the teacher are updated based on the parameters of the student, using the exponential moving average (EMA) strategy.

Consistency Loss

Before computing the consistency loss, we use an IoU-based matching to match teacher and student box pairs. The teacher and student predictions are firstly be filtered by their confidence with a threshold τ_c. Then we calculate the IoU between every pair of remaining teach and student predictions, and filter those pairs with IoUs below threshold τ_I. By these filtering methods, we avoid mismatched soft targets and only keep pairs with higher IoUs. After that we match each student box with the teacher box which has the highest IoU with it. Compared with hard targets, soft targets are often closer to the student predictions, because they are actually based on similar features. Therefore, soft targets can help the student to fine-tune its predictions.

We use the soft targets to supervise both the bounding box regression and classification from the teacher and student. Therefore the consistency loss consists of two parts, one for minimizing the difference between the bounding boxes from student-teacher pairs, the other one for minimizing the difference between the classification scores from student-teacher pairs.

Adapted from the Smooth-L_1 loss, the consistency loss for bounding boxes is formulated as:

$$\begin{aligned} \mathcal{L}^c_{box} &= \frac{1}{N'}\sum_{i=1}^{N}\mathbb{I}(IoU_i > \tau_I)\sum_e \frac{1}{7}\mathcal{L}^c_{\delta_e} \\ \text{and } \delta_e &= \begin{cases} |e_s - e_t| & \text{if } e \in \{x,y,z,w,l,h\} \\ |sin(e_s - e_t)| & \text{if } e \in \{\theta\} \end{cases} \end{aligned} \tag{3.2}$$

where N is the total box pairs after τ_c filtering, $N^{'}$ is the remaining pairs after τ_I, $\{x,y,z,w,l,h,\theta\}$ are the 7 dimensions of a bounding box, e_s represents a certain dimension predicted by the student model, e_t is from the teacher, δ_e is the L_1 distance or absolute sine-error, $\mathcal{L}^c_{\delta_e}$ represents the Smooth-L_1 loss of δ_e, and IoU_i denotes the highest IoU that the i-th student box can acquire from all teacher boxes, $\mathbb{I}(IoU_i > \tau_I)$ performs the τ_I filtering.

The consistency loss for classification scores is formulated as:

$$\begin{aligned} \mathcal{L}^c_{cls} &= \frac{1}{N'}\sum_{i=1}^{N}\mathbb{I}(IoU_i > \tau_I)\mathcal{L}^c_{\delta_c} \\ \text{and } \delta_c &= |\sigma(c_s) - \sigma(c_t)| \end{aligned} \tag{3.3}$$

where δ_c is the L_1 distance, $\sigma(c_s)$ and $\sigma(c_s)$ are classification scores from the student and teacher, $\mathcal{L}^c_{\delta_c}$ represents the Smooth-L_1 loss of δ_c.

The consistency loss is the sum of two parts:

$$\mathcal{L}_{consist} = \mathcal{L}^c_{box} + \mathcal{L}^c_{cls} \tag{3.4}$$

Orientation-Aware Distance-IoU Loss

The smooth-L_1 loss [41] is a common choice in the traditional bounding box regression. L_p losses treat all dimensions of the bounding box as independent. However, these dimensions

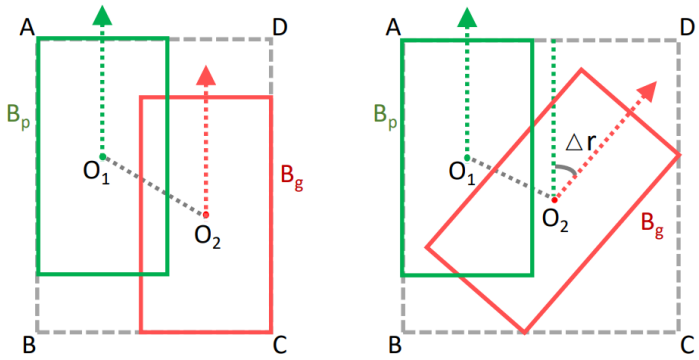

Figure 3.11: Illustration of OD-IoU loss [82]. Left: axis-aligned bounding boxes. Right: non-axis-aligned bounding boxes in BEV. In OD-IoU loss, the central point distance $|O_1O_2|$, minimum enclosing box's diagonal $|AC|$ and orientation difference $\triangle r$ are considered.

have a degree of correlation. The family of IoU loss can capture this correlation by using the IoU between ground truth and predicted bounding boxes as supervision [76].

The D-IoU loss considers the central point distance and diagonal length of the minimum enclosing box [83]. In this way, the loss achieves faster convergence and could handle non-overlapping boxes:

$$\mathcal{L}_{D-IoU} = 1 - IoU + \frac{c^2}{d^2}, \tag{3.5}$$

where c is the distance $|O_1O_2|$ between the central points of two bounding boxes, shown in Figure 3.11, d is the diagonal length $|AC|$ of the smallest cuboid that encloses two boxes, IoU is the IoU between two boxes.

The orientation-aware distance-IoU loss is designed as a fresh supervision signal for our hard supervision, aiming to better deal with non-axis-aligned boxes. Compared with the plain D-IoU loss, the OD-IoU loss also takes orientation into account:

$$\mathcal{L}_{OD-IoU} = 1 - IoU + \frac{c^2}{d^2} + \gamma(1 - |\cos(\triangle r)|), \tag{3.6}$$

where $\triangle r$ is the orientation difference between two boxes in bird's eye view, γ is a hyperparameter — weight factor. With this constraint on the orientation, $\triangle r$ will tend to 0, π or $-\pi$, providing preferable rotation regression.

It should be noted that, here the OD-IoU loss is a loss for supervising the bounding box regression in training, while the IoU prediction mentioned in Section 3.1.6 is a predicted IoU value by a convolutional layer for rectification. The latter one is trained by the smooth-L_1 loss and only applied in inference.

Overall Loss

The overall loss for training the student model consists of:

$$\mathcal{L}_{student} = \mathcal{L}^s_{cls} + \omega_1 \mathcal{L}^s_{OD-IoU} + \omega_2 \mathcal{L}^s_{dir} + \lambda \mathcal{L}_{iou} + \mu_t \mathcal{L}_{consist}, \tag{3.7}$$

where $\mathcal{L}^s_{cls}$ is the focal loss [39] for box classification, $\mathcal{L}^s_{OD-IoU}$ is the OD-IoU loss for bounding box regression, $\mathcal{L}^s_{dir}$ is the cross-entropy loss for direction classification, $\mathcal{L}_{iou}$ is the smooth-L_1 loss for IoU prediction, $\mathcal{L}_{consist}$ is the consistency loss, ω_1, ω_2, λ and μ_t are weights of losses.

3.2 Proposed Point Cloud Registration Algorithm

In the urban area of the Providentia++ test stretch — the intersection of Schleißheimer Strasse and Zeppelinstrasse in Garching-Hochbrück (S110), there are two Ouster OS1-64 and three Valeo Scala2 LiDARs. Two OS1-64 are mounted side by side (about 13 m apart) on a gantry bridge as the main LiDARs. To merge point cloud scans from these two OS1-64, we propose our point cloud registration algorithm for infrastructure-mounted LiDARs.

3.2.1 Problem Definition

$L1$ and $L2$ are two time synchronized LiDARs. One is treated as the source LiDAR and the other is the target LiDAR. Each scan captures N points. The two point cloud scans of them at timestamp t are $A = \{a_i\}_{i=1,\dots,N}$ and $B = \{b_i\}_{i=1,\dots,N}$. Our goal is to put these two point clouds into the same coordinate system. More specifically, we want to transform the source point cloud to the target point cloud's coordinate system. To achieve this, we need to find a set of n correspondence pairs $S = \{s_i\}_{i=1,\dots,n}$ and $T = \{t_i\}_{i=1,\dots,n}$, where s_i corresponds with t_i. Then we estimate the rigid transformation $\mathbf{T}$, including rotation $\mathbf{R}$ and translation $\mathbf{t}$, by minimizing the RMSE between correspondences.

3.2.2 Motivation

To achieve real-time point cloud registration on OS1-64's 130K point clouds, optimization-based registration methods are more desirable because more computationally efficient. Another major advantage is optimization-based methods are free of training data. However, the problem of trapping in local optimum is quite challenging for optimization-based methods in dynamic scenes and large relative distance. In our scenario, a naive observation of infrastructure-mounted LiDARs is that their positions are static for the most of time, with slight oscillation caused by wind or large vehicles passing by. Inspired by [31], we could provide an initial transformation matrix to help registration algorithms better overcome local optimum. The initial transformation can be acquired by a global registration that doesn't require an initial position, or by accurate Real-time Kinematic (RTK) positioning.

The designed registration algorithm is divided into two stages: the initial registration and the continuous registration. The initial registration is a global registration based on geometric feature-based correspondence searching and RANSAC-based transformation estimation. With this initial transformation, the following continuous registration gets a good initial position and therefore it is less likely to be trapped in local optimum. The continuous registration is done by point-to-point ICP.

3.2.3 Initial Registration

The initial registration has no prior transformation as a reference, so traditional ICP algorithms may not perform well. Therefore, we take a geometric feature-based correspondence

searching. Fast Point Feature Histograms (FPFH) [53] is a hand-crafted geometric feature descriptor. To extract the FPFH feature for point p:

1. A set of k neighbor points $\mathbf{P}^k$ within radius r are selected, forming a set of $\langle p, p_j \rangle$ point pairs. For each pair p and p_j, we compute their estimated normal vector n and n_j of the surface formed by a point and its neighbourhood [3].

2. Using p as the origin, we build a Darboux frame ($u = n$, $v = (p_j - p) \times u$, $w = u \times v$), and calculate the angular variant of n and each n_j:

$$\begin{aligned} \alpha_j &= v \cdot n_j \\ \phi_j &= (u \cdot (p_j - p)) / \left\| p_j - p \right\| \\ \theta_j &= \arctan(w \cdot n_j, u \cdot n_j) \end{aligned}$$

   These three kinds of angular information of all pairs are collected as $\langle \alpha, \phi, \theta \rangle$ and constitute the hand-crafted geometric feature.

3. The distribution of each angular information are then categorized using three 11-bin histograms separately, called Simplified Point Feature Histogram (SPFH).

4. Then we re-determine k neighbors of p and compute the final FPFH by weighted addition of neighbors' SPFH:

$$FPFH(P) = SPFH(P) + \frac{1}{K} \sum_{i=1}^{k} \frac{1}{\omega_k} \cdot SPFH(p_k),$$

   where ω_k is the distance between p and neighbor p_k.

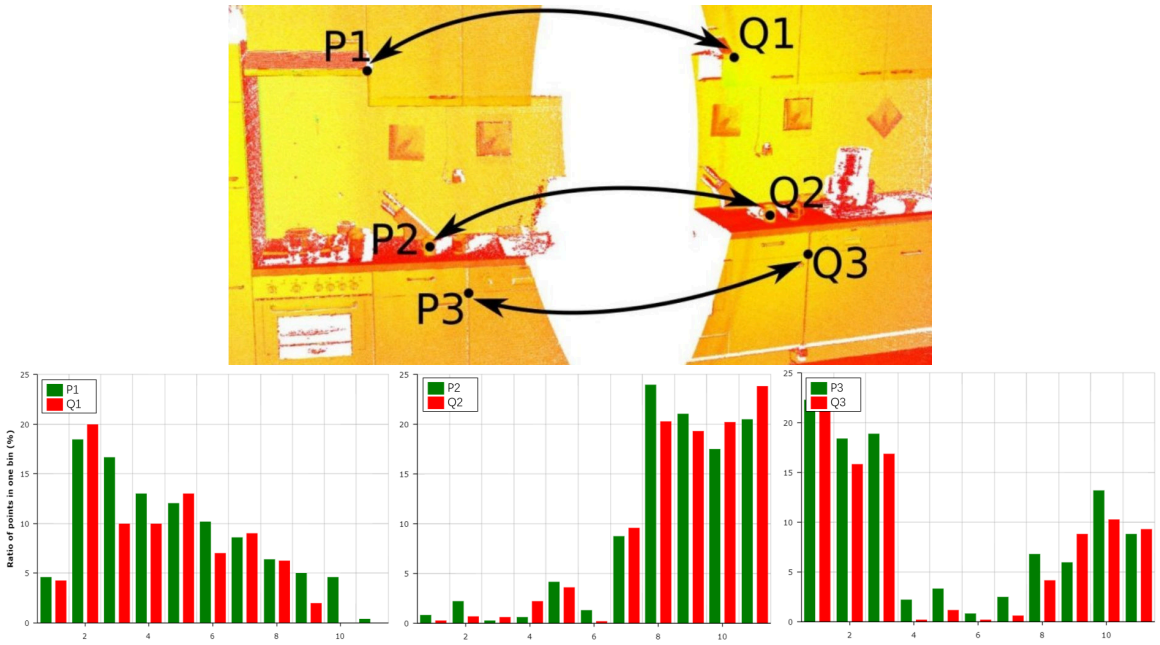

Figure 3.12: Illustrate one of the three kinds of 11-bin angular information in Fast Point Feature Histograms. The x axis represents 11 bins of the α angle, while y axis represents the ratio of neighbor points that fall into the bin. Green: points P1, P2, P3. Red: point Q1, Q2, Q3. ϕ and θ information is not displayed.

The resulting FPFH feature can be formulated as a 33-dimensional vector. Figure 3.12 shows an example of the 11-bin α variant's histogram. Three pairs of similar points in two point clouds are depicted. Points in a similar local geometric structure have similar histogram distribution. A nearest neighbor query in the FPFH feature space can return points with similar local geometric structures. Therefore, FPFH features can be used for the correspondence searching task.

Algorithm 1 Initial Registration

Input: Source point cloud: $A = \{a_i\}_{i=1,\dots,N}$; Target point cloud: $B = \{b_i\}_{i=1,\dots,N}$;
Init voxel size: v; ▷ For downsampling and determining radii.
Init budget: b; ▷ For running multiple times and selecting the best result.
RANSAC convergence criteria: confidence: c, maximum iteration: I_{max};
Output: Initial transformation: T_{init}

1: **while** $b > 0$ **do**
2: **for** $P \in \{A, B\}$ **do** ▷ FPFH feature extraction
3: $P_{down} \leftarrow VoxelDownSample(P, v)$
4: $r_{normal} \leftarrow v * 2$ ▷ Radius for k-NN
5: $P_{normal} \leftarrow EsimateNormals(P_{down}, KNNSearch(r_{normal}))$
6: $r_{feature} \leftarrow v * 5$
7: $P_{FPFH} \leftarrow ComputeFPFHFeature(P_{down}, P_{normal}, KNNSearch(r_{feature}))$
8: **end for**
9: **for** $iter < I_{max}$ **do** ▷ RANSAC iteration
10: $\{s_i\}_{i=1,\dots,n} \leftarrow RandomSample\_n\_Points(A)$
11: $\{t_i\}_{i=1,\dots,n} \leftarrow FPFHFeatureQuery(\{s_i\}, A_{FPFH}, B_{FPFH})$
12: $T_i \leftarrow SVDBasedTransformEstimate(\{s_i\}, \{t_i\})$
13: $pass \leftarrow CorrespondenceMatchCheckers(\{s_i\}, \{t_i\}, T_i)$
14: **if** pass **then**
15: $\{T_i\} \leftarrow \{T_i\} \cup T_i$
16: **if** $ConvergenceCheck(T_i)$ **then**
17: **break**
18: **end if**
19: **end if**
20: **end for**
21: $\{T_{temp}\} \leftarrow \{T_{temp}\} \cup BestCriteria(\{T_i\})$
22: $b \leftarrow b - 1$
23: $A, B \leftarrow GetNextFrames()$
24: **end while**
25: $T_{init} \leftarrow BestCriteria(\{T_{temp}\})$
26: **return** T_{init}

Algorithm 1 outlines the procedure of the initial registration: 1) Init budget controls running initial registration multiple times and select the best transformation, in case that some difficult frame; 2) In Steps 2-8, we compute the FPFH features for two input point clouds, including voxel-based downsampling to reduce the number of points, estimating normals for downsampled point clouds with k-d tree-based k-nearest neighbors algorithm (K-NN), and FPFH featuring extraction as mentioned before; 3) In Steps 9-19, the RANSAC iteration [19] is performed. We randomly sample n points from the source point cloud. A FPFH feature-based query is responsible for searching sampled points' correspondences in the target point cloud. Then we perform a SVD-based point-to-point transformation estimation on the correlation matrix of correspondences. After that, two inexpensive pruning methods are applied to

earlier reject false matches. One is the edge length checker: checking whether edges drawn by points from source and target correspondences have similar lengths. The other one is the distance checker: checking whether transformed points are close to each other. Estimated transformations that passed two checkers are then collected and computed for the RMSE, fitness, and convergence criteria; 4) The best transformation in all passed transformations will be selected as the current RANSAC result; 5) The best transformation with lowest RMSE and highest fitness score in budget $\{T_{temp}\}$ will be used as the initial transformation.

The RMSE is computed among correspondences, while the fitness is the percentage of the correspondences in the target point cloud. The convergence criteria includes the maximum number of iterations I_{max} and the confidence c to determine whether the found transformation is eligible for early termination: if $num\_iteration \geq \log(1-c)/\log(1-fitness^n)$, then the RANSAC iteration could terminate.

In order to provide a more precise initial transformation, we use an ICP-based method to refine the temporal transformation T_{temp} after each FPFH feature-based registration (not shown in Algorithm 1). This ICP-based method is the same as the continuous registration.

3.2.4 Continuous Registration

After computing the initial transformation, we get a nice initial guess of the relative position of source and target point clouds. We use a point-to-point ICP algorithm to register the subsequent point cloud frames [5].

Algorithm 2 Continuous Registration

Input: Source point cloud: $A = \{a_i\}_{i=1,\dots,N}$; Target point cloud: $B = \{b_i\}_{i=1,\dots,N}$;
Cont voxel size: v;
Initial transformation: T_{init};
ICP convergence criteria: RMSE: e, fitness: f, maximum iteration: I_{max};
Output: Continuous transformation: T_{cont}

$A_{down} \leftarrow VoxelDownSample(A, v)$
$B_{down} \leftarrow VoxelDownSample(B, v)$
$r_{neighbor} \leftarrow v * 0.4$
$T_{temp} \leftarrow T_{init}$
for $iter < I_{max}$ **do** ▷ ICP iteration
  $A' \leftarrow Transform(A_{down}, T_{temp})$
  $\{s_i\}_{i=1,\dots,n}, \{t_i\}_{i=1,\dots,n} \leftarrow NNSearch(A', B_{down}, r_{neighbor})$
  $T_{temp} \leftarrow SVDBasedTransformEstimate(\{s_i\}, \{t_i\})$
  $e_{temp} = ComputeRMSE(T_{temp})$
  $f_{temp} = ComputeFitness(T_{temp})$
  if $e_{temp} < e$ **and** $f_{temp} > f$ **then**
    break
  end if
end for
$T_{cont} \leftarrow T_{temp}$
return T_{cont}

Algorithm 2 shows the process of the continuous registration: 1) Similar voxel-based down-

sampling performs on the source and target point clouds; 2) In the ICP iteration Step 5-14, we iteratively transform the source point cloud to the target's coordinate with the latest T_{temp}. After each transformation, for points in A', we find their nearest neighbors from B_{down} within distance $r_{neighbor}$, to get correspondence sets s_i and t_i. We apply a SVD based point-to-point transformation estimation on correspondences and compute RMSE and fitness scores. If the temporal transformation passes the convergence criteria, then break, otherwise iterate until it reaches the maximum number of ICP iterations; 3) The final temporal transformation after iterations will be returned as the continuous transformation matrix.

Continuous registration is applied on each subsequent frame. One OS1-64 LiDAR emits 1,310,720 points per second, while the two registered LiDARs emit 2,621,440 points per second (PPS). They are denser and cover a larger region. The intensity information is also preserved, although they are not used in computing the transformation matrix. This solves the point cloud sparsity problem to some extent and facilitates subsequent tasks.

3.2.5 Evaluation of Point Cloud Registration Algorithm

Our two OS1-64 LiDARs at the Providentia++ test stretch run at 10 Hz to get the highest horizontal resolution of 2048 points. Therefore, the total callback time of our registration algorithm should not be slower than 100 ms. We test our algorithm with an Intel Core i7-9750H CPU. Table 3.1 shows some statistical results under different voxel sizes. Through experimental observations, an initial voxel size of 1 or 2 meters could give a stable initial transformation. Initial sizes larger than 2 meters may cause oscillation in continuous registration or local optimum problem. Increasing the continuous voxel size could reduce the registration time and total callback time, but the RMSE will increase and be more likely to oscillate if voxels are larger than 4 meters. We count the average callback time only for the continuous registration. All callback times are below 100 ms even with a weak CPU. For continuous registrations with voxels larger than 2 meters, the callback time is below 50 ms.

	Size(m)	Reg. time(ms)	RMSE	Fitness(%)	Callback(ms)
Init.	1	118.94	**0.20522**	22.2	63.6
Cont.	1	31.4	0.26742	32.8	
Init.	1	129.42	0.20814	20.5	53.3
Cont.	2	18.74	0.52228	40.5	
Init.	2	**45.94**	0.3999	28.4	63.6
Cont.	1	33.06	0.26646	32.6	
Init.	2	46.22	0.40046	29.1	51.8
Cont.	2	18.36	0.52164	40.5	
Init.	2	47.62	0.40588	25.6	48
Cont.	3	15.46	0.7599	42.7	
Init.	2	47.62	0.40438	27.0	**46.1**
Cont.	4	**13.08**	0.98882	**47.3**	

Table 3.1: Performance of initial and continuous registration under different voxel sizes. Only continuous registration callback times are counted.

We take the initial voxel size and continuous voxel size both as 2 meters, because it has both speed and accuracy. Our code is based on the Open3D library [86]. Figure 3.14 shows some results of point cloud scans before and after registration.

Figure 3.13: Up: unregistered point clouds. Down: corresponding registered point clouds.

Figure 3.14: Point cloud registration results. Up: unregistered point clouds. Down: corresponding registered point clouds.

3.3 Proposed Point Cloud Upsampling Method

The point cloud registration algorithm in Section 3.3 provides us denser point clouds. However, this only works when two LiDARs are installed at one sensor station. In sensor stations on the Highway A9 (S40, S50, and S60), we only have one OS1-64 as the main LiDAR. It is possible to merge Ouster LiDARs and other mounted Valeo LiDARs, but the OS1-64 is a rotating LiDAR while Valeo LiDARs are solid-state and run at a different frequency. Merging them is more difficult and provides us with heterogeneous point clouds.

Another intuitive way of increasing the density of point clouds is to increase the resolution of the LiDAR. Figure 3.15 compares the density of the 64-channel Ouster LiDAR and the 128-channel one. They have the same vertical field of view (FOV) but the latter one has a double number of channels. It can capture double points, but it is also more expensive. In this section, we present our point cloud upsampling method — a neural network-based point cloud super-resolution model. It takes 64-channel LiDAR scans as input and outputs 128-channel scans.

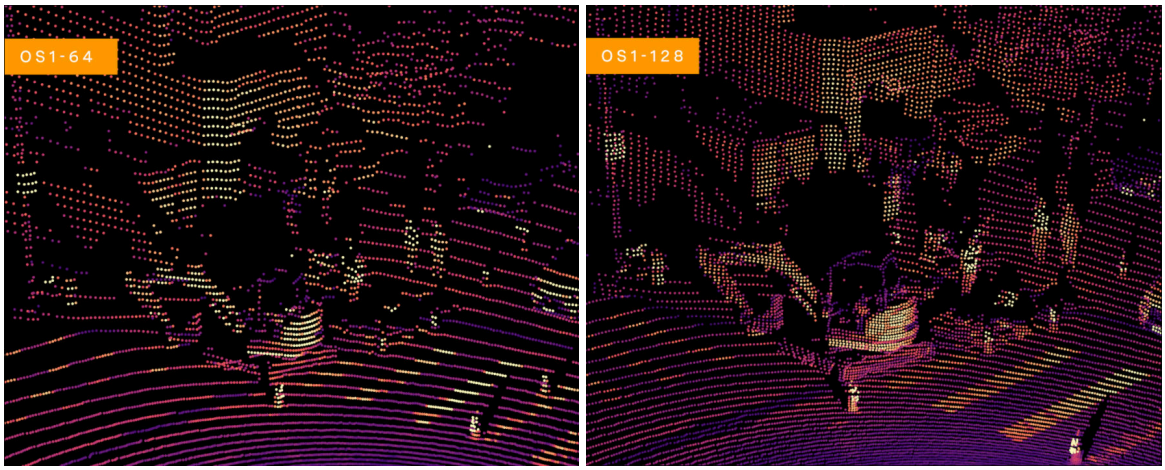

Figure 3.15: Comparison of OS1-64 and OS1-128.

3.3.1 Generation of Synthetic Dataset proSynth

To train such a neural network, we need a dataset of 128-channel point clouds as the training data. However, we have no real OS1-128 LiDAR available to create such a dataset. Therefore, we use the virtual LiDAR in the CARLA simulator [18] to generate a synthetic 128-channel LiDAR dataset proSynth. In the Providentia++ project, a simulated map of the test stretch has been built based on CARLA as the digital twin of the real-world test stretch. Figure 3.16 shows the simulated environment in CARLA v0.9.12.

We set two virtual LiDARs to the position corresponding to the real-world sensor station S50 and S110. The coordinates of them in the simulated map are $[x = -25, y = 1470, z = 493.14]$ and $[x = -691.3, y = 623.6, z = 490]$ separately. We spawn virtual vehicles using the spawning script provided by CARLA to simulate the traffic. Spawning points are defined in the Providentia++ map.

Figure 3.16: Simulated Providentia++ test stretch.

The OS1 LiDARs have following parameters:

Parameter	OS1-64	OS1-128
Vertical Resolution	64 channels	128 channels
Horizontal Resolution	512, 1024, or 2048	512, 1024, or 2048
Range	120 m	120 m
Vertical FOV	45^o ($\pm 22.5^o$)	45^o ($\pm 22.5^o$)
Vertical Angular Resolution	0.35^o - 2.8^o	0.35^o
Precision	±0.7 – 5 cm	±0.7 – 5 cm
Points Per Second	1,310,720	2,621,440
Rotation Rate	10 or 20 Hz	10 or 20 Hz

Table 3.2: Parameters of OS1-64 and OS1-128.

We set the horizontal resolution of our OS1-64 LiDAR to 2048 and the rotation rate to 10 Hz. The resolution of it is 64×2048. Our plan is to generate 128-channel point cloud scans with resolution 128×2048. To simulate 128-channel LiDAR scans, we configure the virtual LiDAR (Blueprint: sensor.LiDAR.ray_cast) as follows:

Blueprint attribute	Value
channels	128 channels
range	120 m
vertical_fov	45^o ($\pm 22.5^o$)
horizontal_fov	360^o
points_per_second	5,242,880
rotation_frequency	20 Hz
dropoff_general_rate	0.1
noise_stddev	0.1

Table 3.3: Parameters of the virtual LiDAR in CARLA.

We double the number of points per second and the frequency. Because we find that in the current version of the CARLA simulator, if we set the frequency to 10 Hz, only half the region of the point cloud will be dumped. Another flaw of the current CARLA is the simulated intensity is obtained by the distance to the LiDAR and a drop rate, instead of the surface material of the object. We wish an update on this in future releases of CARLA, to reduce the reality gap. We save one frame in every five frames to increase the randomness in the traffic.

In total, we capture 1000 point cloud scans in .ply format, with 500 in S50 and 500 in S110. Figure 3.17 shows two examples of generated point clouds.

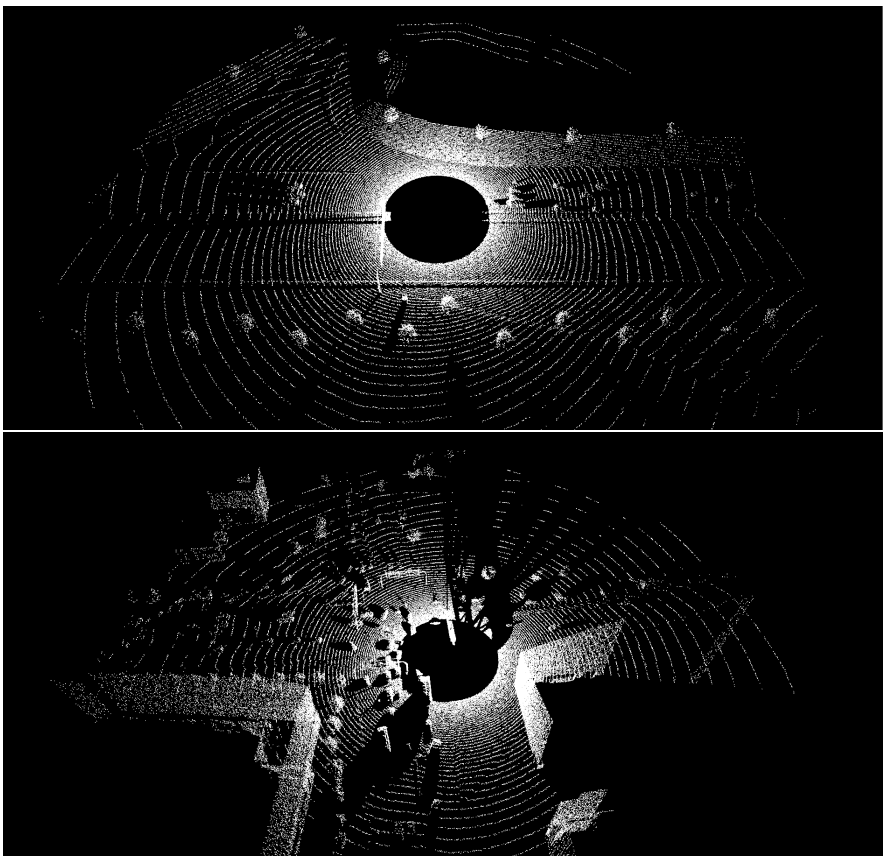

Figure 3.17: Simulated 128-channel point cloud scans.

3.3.2 Channel-wise Point Cloud Super-resolution

Similar to image super-resolution, Shan et al. [55] propose a channel-wise point cloud super-resolution method. It converts input low-resolution 3D point cloud to a low-resolution 2D range image and uses the 2D convolutional neural network to perform on the range image. The output is a high-resolution range image, and it can be converted back to a 3D high-resolution point cloud. The process is shown in Figure 3.18. The authors focus on 16 to 64 and 32 to 64 channel upsampling. Based on this work, we make further improvements and train the model on our proSynth dataset. Our improvements are mainly the model structure and data augmentation.

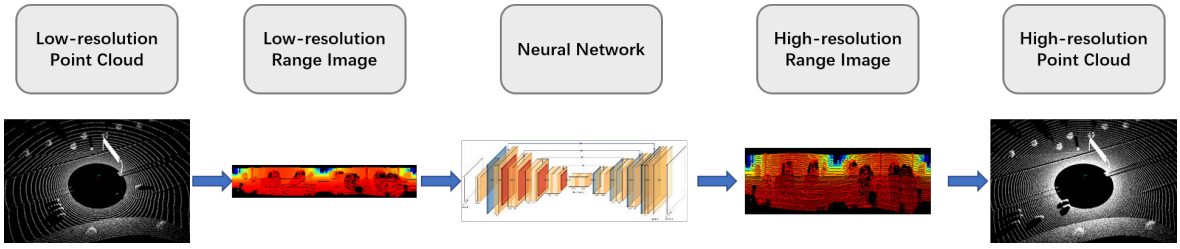

Figure 3.18: Process of channel-wise point cloud super-resolution.

Neural Network Architecture

We enlarge the input to a concatenation of the range image and intensity image. In [55], the input and output to the model are ($H \times W \times 1$) dimensional range images, converted from the 3D point cloud. That means the output contains only range information and the intensity information in the raw point cloud is discarded. The intensity is an important feature for further tasks, and we want to keep it. Therefore, the input to our model is a ($64 \times 2048 \times 2$) dimensional range-intensity map. Our neural network makes both range and intensity predictions. Since super-resolution is formulated as a regression problem, the input is normalized by the maximum range and maximum intensity before being fed into the neural network. We use the L_1 loss to train the model. Experiments show L_2 loss converges slowly after normalization.

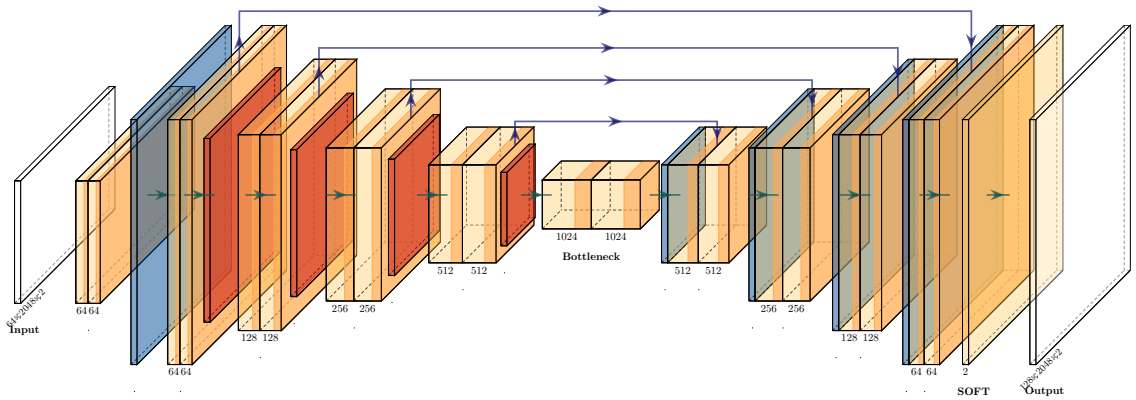

Figure 3.19: Architecture of point cloud super-resolution model. White layer: input and output. Double yellow layer: convolution block. Blue layer: transposed convolution. Red layer: average pooling. Single yellow layer: 1×1 convolution. Purple arrow: skip connection. The diagram is created by PlotNeuralNet [30].

In the original implementation, the authors design a model similar to UNet [51], except at the beginning, one or two transposed convolutional layers (depends on the upsampling factor) are applied directly on the low-resolution input. We argue that directly applying transposed convolution on the input puts too much burden on the transposed convolutional layer, which leads to information loss. Therefore, we add a convolution block before the transposed convolution to provide a better feature map for the transposed convolutional operation.

The architecture of our model is shown in Figure 3.19. A UNet-like structure follows the transposed convolution. The encoder part captures high-level semantic features. Each convolution block contains two 3×3 same padding convolutional layers, batch normalization, and ReLU activation function. A 2×2 average pooling layer does the downsampling, cuts the spatial resolution to half, followed by a dropout layer. The number of feature channels gradually increases from 64 to 1024 till the bottleneck. The decoder part recovers spatial information and propagates semantic features. A transposed convolutional layer gradually upsamples the feature map, followed by a convolution block and dropout. A final 1×1 convolutional layer generates the output ($128 \times 2048 \times 2$) range-intensity map. Skip connections are added between the encoder and decoder, which concatenates the upsampled feature maps and feature maps from the convolution block in the encoder. It helps to better preserve spatial information.

Data Augmentation

The ground truth in our proSynth dataset is 128-channel point clouds. We convert them to $(128 \times 2048 \times 2)$ range-intensity maps. When training the neural network, we take one in every two channels in the ground truth as the input, i.e. $(64 \times 2048 \times 2)$. In the original paper, the authors only use one kind of data augmentation — Gaussian noise, applied on the range images in the training set before extracting the input from it. That makes the training set have more noise than the test set. However, that causes a critical flaw — the input is always exactly part of the ground truth. That makes the model easy to overfit, and also may cause learning to take shortcuts: the neural network makes no feature learning, but only propagates the 64-channel oracle to the output. The shortcuts harm the optimization process: the parameters update towards oracle propagation and get stuck in a local optimum.

To facilitate feature learning and reduce the generalization gap, we add five different data augmentation methods to the training pipeline: 1) global random flip: randomly flip the y coordinates of the 128-channel 3D ground truth (flip left and right); 2) global random rotation: randomly rotate the ground truth by $[-45, 45]$ degrees; 3) global scaling: randomly scale the ground truth by $[0.95, 1.05]$ times; 4) global 3D Gaussian noise: add Gaussian noise to each 3D point in the ground truth, before extracting the input; 5) local 2D Gaussian noise: add noise to the extracted input range image, after converting the ground truth.

The first four data augmentation methods improve the generalization ability, while the last one facilitates feature learning. The 3D global noise could have any direction. The 2D local noise on range images has only two directions. We also tried to randomly move the starting channel up by one, when extracting the input. However, it raises the difficulty too much. There's no feature implied starting from channel 0 or channel 1. That causes ambiguity to the neural network, therefore abandoned.

3.3.3 Evaluation of Point Cloud Super-resolution Model

We split the proSynth dataset into 640 training samples, 160 validation samples, and 200 testing samples. We use L_1 loss to train our model for 80 epochs. The batch size is set to 8. The probabilities of applying the first four data augmentation methods are all 20%, while the local 2D Gaussian noise is always added. There's no metric of precision in the point cloud super-resolution task. Here we only report the plain L_1 loss (mean absolute error) is 0.0494 and the L_2 loss (mean squared error) is 0.0345 on the test set. The inference time is 59 ms on a Nvidia GeForce RTX 3090 GPU.

To refine the final output, Shan et al. [55] propose two methods. One is the Monte-Carlo dropout: run the model multiple times on each testing frame and compute the mean, also denoise using the standard deviation. Since we have the real-time requirement in Providentia++, we are not allowed to be slower than our LiDARs. Running the inference multiple times doesn't fit our scenario. The other refinement is we could copy the oracle 64-channel input back to the raw output. This is a postprocessing operation and does no harm to the training. In image super-resolution, this operation may cause inconsistent clarity, but in sparse point clouds, it's a free gift. Therefore, we adopt this refinement process. The final output is a combination of the 64-channel input and 64 predicted channels.

The super-resolution results on the test set of our proSynth dataset is shown in Figure 3.28.

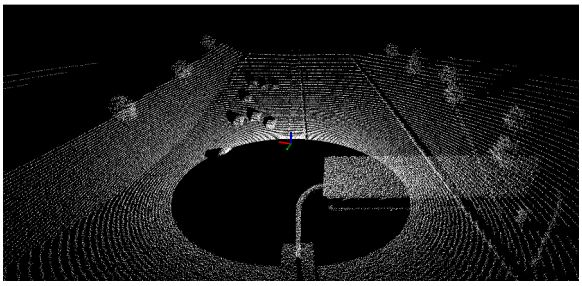

Figure 3.20: 128-channel ground truth at S50.

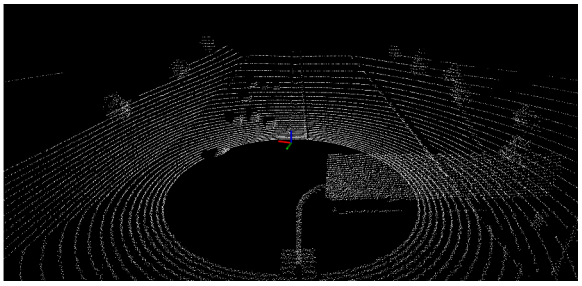

Figure 3.21: 64-channel input.

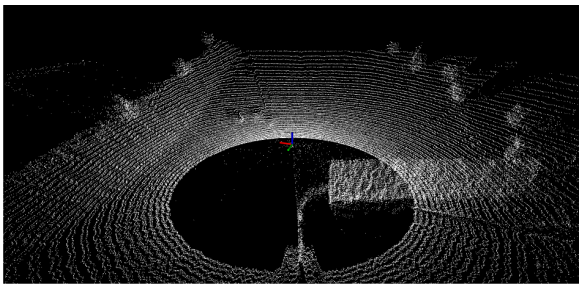

Figure 3.22: Raw output.

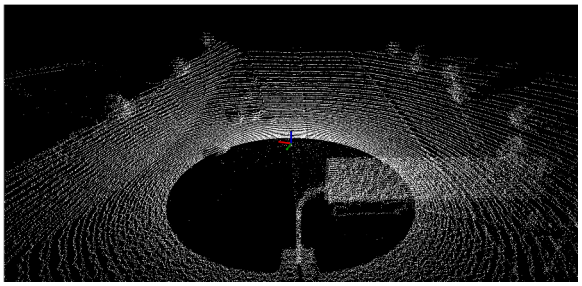

Figure 3.23: Output after postprocessing.

Figure 3.24: 128-channel ground truth at S110.

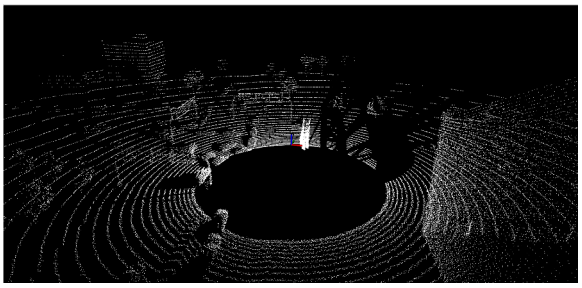

Figure 3.25: 64-channel input.

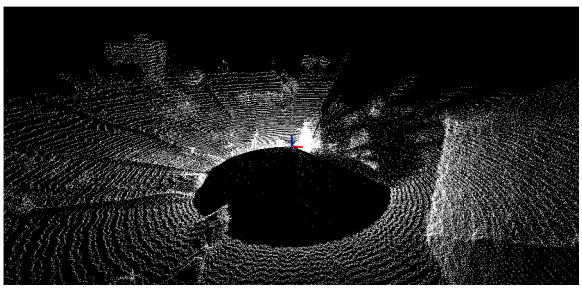

Figure 3.26: Raw output.

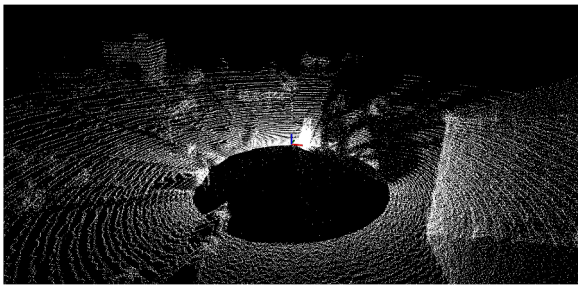

Figure 3.27: Output after postprocessing.

Figure 3.28: Super-resolution results on simulated S50 and S110.

We also test our model on the real world OS1-64 LiDAR, shown in Figure 3.37. However, because of the reality gap between the CARLA Simulator and the real world, our model doesn't perform well on real-world data. Despite these flaws, real-world experiments show the effect of our modification to the model structure and our data augmentation: the original implementation [55] only learns to propagate some of the 64-channel input, while the other 64 channels are all random abnormal points spreading in the space (Figure 3.32, 3.36). On the contrary, our model makes a full 128-channel prediction. Our model output is more regular and contains fewer random points. The model performs a little bit better at S50 because we have a larger reality gap at simulated S110.

Figure 3.29: 64-channel real-world LiDAR scan at S50.

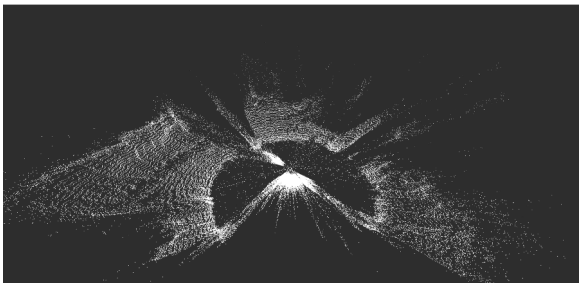

Figure 3.30: 128-channel raw output.

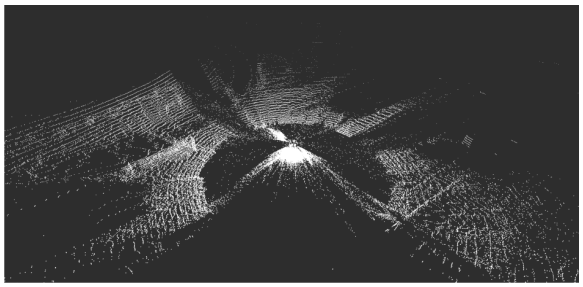

Figure 3.31: 128-channel output after postprocessing.

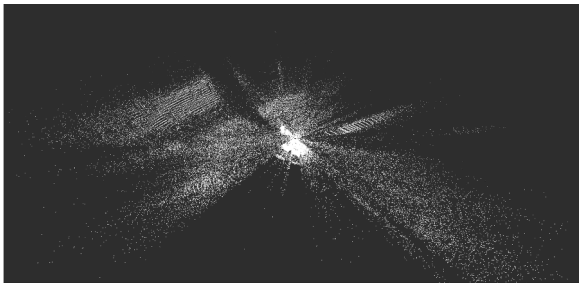

Figure 3.32: Output from the original implementation without modification and data augmentation.

Figure 3.33: 64-channel real-world LiDAR scan at S110.

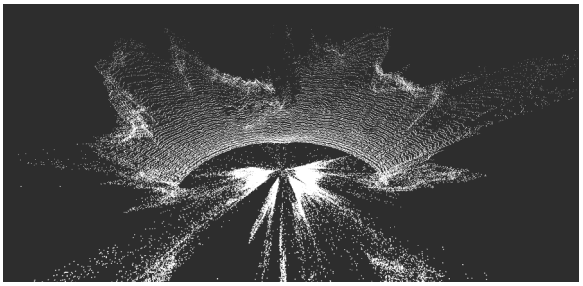

Figure 3.34: 128-channel raw output.

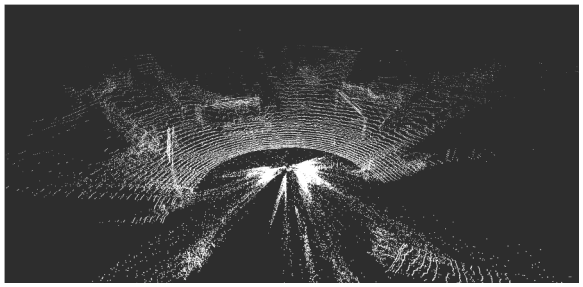

Figure 3.35: 128-channel output after postprocessing.

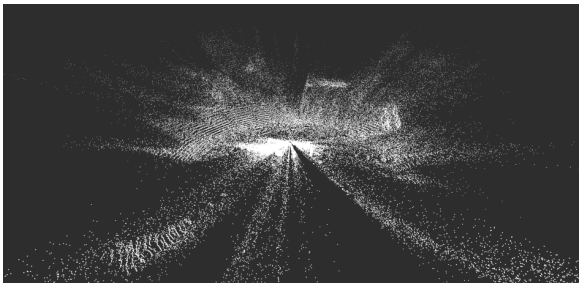

Figure 3.36: Output from the original implementation without modification and data augmentation.

Figure 3.37: Super-resolution results at real-world S50 and S110. In the S110 examples, we show a case of a bus passing by. Outputs are not from the same frame because it's a ROS node test. The perspective of each figure is also not the same.

3.4 Proposed Web-based Point Cloud Visualizer

Visualization is an important task in a traffic digital twin like the Providentia++ project. To visualize point cloud scans from LiDARs and better monitor the real-time traffic situation, we implement two different web-based visualization tools. One is the 3D visualizer with a 3D perspective. The other one is the projection-based 2D visualizer which is bandwidth saving.

3.4.1 3D Visualization Tool

Our 3D visualizer is built on Uber's XVIZ [61] protocol and streetscape.gl [62] toolkit. XVIZ is a real-time data transfer protocol, based on Node.js. It supports the transfer of a variety of information, including geometry, point clouds, images, 3D and 2D shapes, etc. It defines different types of information streams and can be used for building a backend server to stream data from autonomous systems. Streetscape.gl is a frontend visualization toolkit based on React and Uber's WebGL-powered visualization frameworks. Our work is mainly on the backend, subscribing to ROS topics from sensors and streaming data to the web frontend.

Although the protocol is powerful, the authors didn't provide live streaming APIs or examples. Simply cloning the repository, we could only stream files on disks. To speed up the development and also match our technique stack. We take a C++ implementation of XVIZ by Xu [67]. It has better support for building a live data streaming backend. This C++ version XVIZ is used to build CarlaViz [68], a web-based visualizer for CARLA Simulator modified from streetscape.gl. We like CarlaViz's black theme, so we directly used it as the frontend without any modification.

In our streaming server, we provide options for subscribing to point cloud topics, image topics, and 3D bounding box topics. Messages from different topics are further synchronized by the ROS message_filters. We push received messages into the corresponding message queue in the message_filters::Synchronizer thread. The server class — LiveSession (inherited from XVIZBaseSession) keeps fetching messages from the queue at a streaming rate (e.g. 10 Hz) we set.

Figure 3.38 shows an example at S110, with merged point clouds and bounding boxes. Figure 3.39 shows an example with point clouds, bounding boxes, and images. Since the LiDAR-camera synchronization in the Providentia++ project is not finished yet, we use the range image as an example, shown in the top left corner. Figure 3.40 displays the visualization result of a 25 Hz Valeo LiDAR. All point cloud ROS topics are from recorded ROS bags.

3.4.2 2D Visualization Tool

Although the 3D perspective is more intuitive, transforming point clouds with 131K points at 10 Hz takes around 9 to 10 MiB/s of bandwidth. In each sensor station, we have a lot of programs running and a lot of information to be transferred. Therefore, we are looking forward to a more bandwidth-efficient solution for point cloud visualization. We decide to project point clouds onto images. In this way, the bandwidth usage is very low.

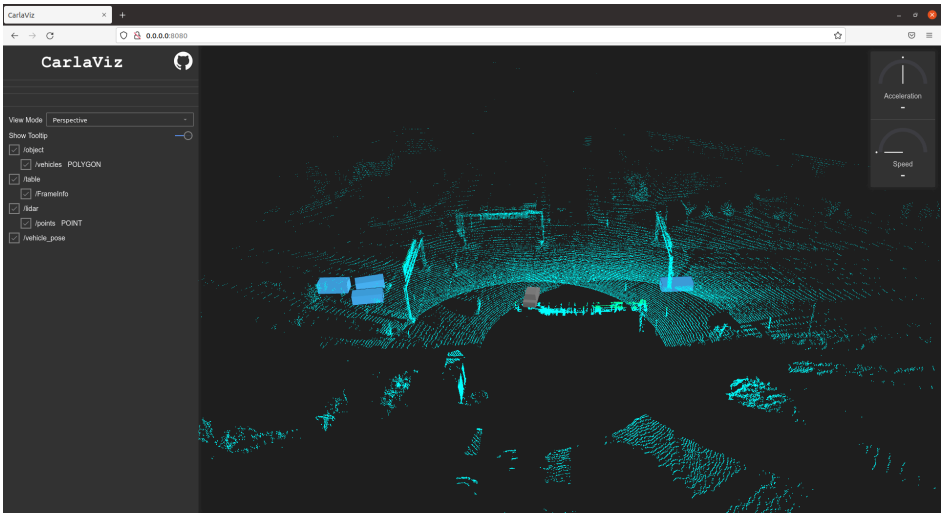

Figure 3.38: Web-based visualization of OS1-64 LiDARs at S110. The point clouds are and registered by the algorithm in Section 3.2. The bounding box topic is published by our 3D detector ROS node.

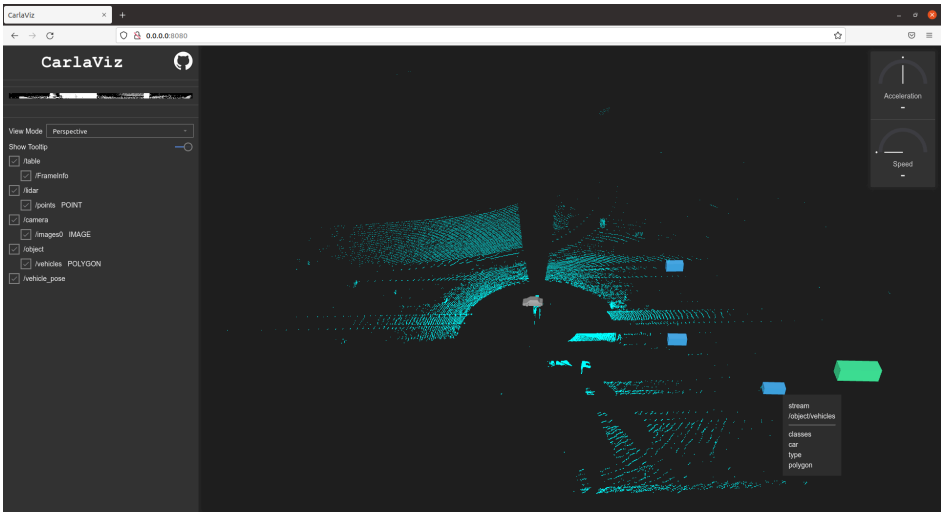

Figure 3.39: Web-based visualization of a single OS1-64 at S50.

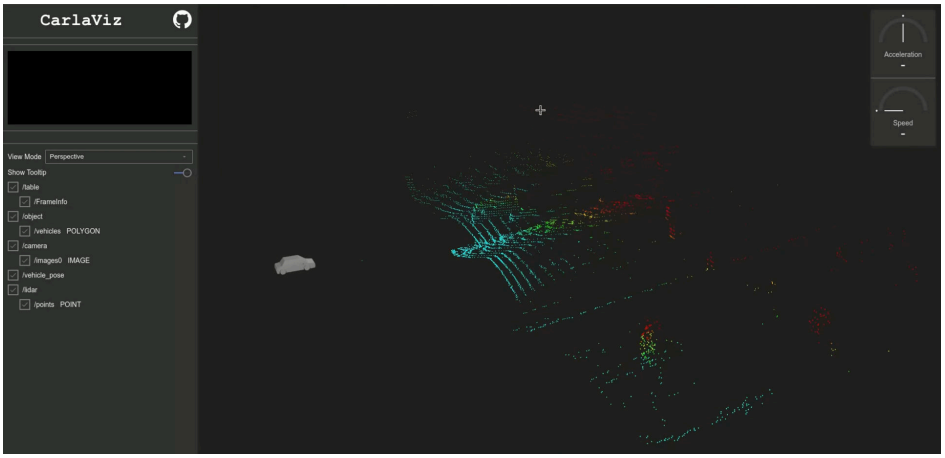

Figure 3.40: Web-based visualization of a Valeo LiDAR at S50.

We borrow some utility code from [2]. This repository is a projection-based visualizer for results of the KITTI dataset, but it's offline and only for files on disks. We greatly speed it up. The total callback time drops from 250 ms to 37 ms. We reduced the projection time by the matrix chain multiplication trick and re-implement the point painting function. The modified visualization tool could publish the projected point cloud by ROS topic, network socket, and FFmpeg streaming (pix_fmt bgr24). We support projected front-view point clouds, BEV point clouds, and painting the projected points onto camera images. For the point clouds only mode, the output image is 1280×512. For the BEV mode, it's 1000×1000. For the image painting mode, it depends on the image size. All these parameters can be modified. Figure 3.41 to 3.44 demonstrate our 2D visualization tool. In Figure 3.43, we show our image-painting streaming by topics from a KITTI rosbag, because the LiDAR-camera calibration of the Providentia++ project is not finished.

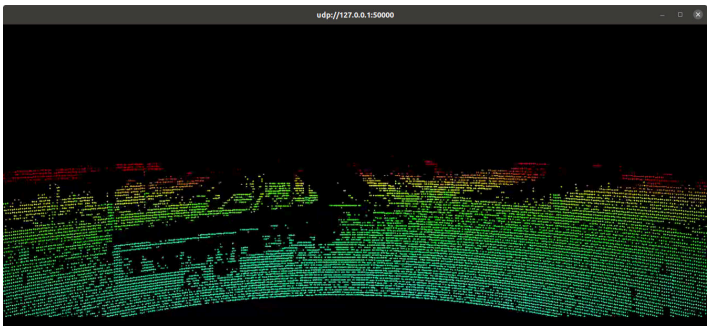

Figure 3.41: FFmpeg streaming of the projected OS1-64 scan at S110.

Figure 3.42: Socket streaming of the BEV OS1-64 scan at S110.

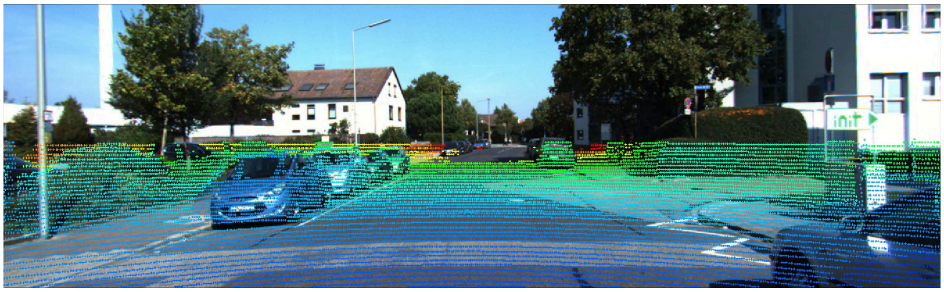

Figure 3.43: Example of the image-painting streaming as a ROS topic. Raw point clouds and images published by a KITTI rosbag.

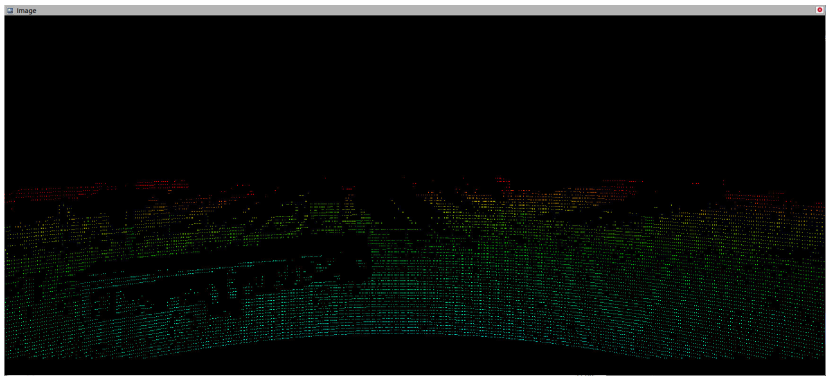

Figure 3.44: ROS image topic of the projected OS1-64 scan at S110.

3.5 Proposed Auto Labeling Function to ProAnno

ProAnno (based on 3D BAT [89]) is the labeling tool that we used in labeling the dataset of Providentia++ project. In order to reduce the effort of manual labeling, we add an auto labeling function to ProAnno. We use a 3D object detector to generate predicted labels. Here we take ProPillars [87] as the model, since this auto labeling function is implemented at the beginning of this thesis.

ProAnno is based on JavaScript. However, ONNX [17] doesn't support those complicated operations in a 3D detector. Therefore, we cannot really embed the function inside ProAnno's code. We implement the function as an extension module. We provide two different modes for the auto labeling. One is the offline "Auto Label All", where the detector will go through all point cloud files and save its predictions into the corresponding annotation files. The other model is the online "Auto Label Server". We implement a detector server based on Flask [25]. We first start the server in the backend and then click the "Auto label" button in ProAnno to transfer the current point cloud frame and receive the predictions back, shown in Figure 3.45.

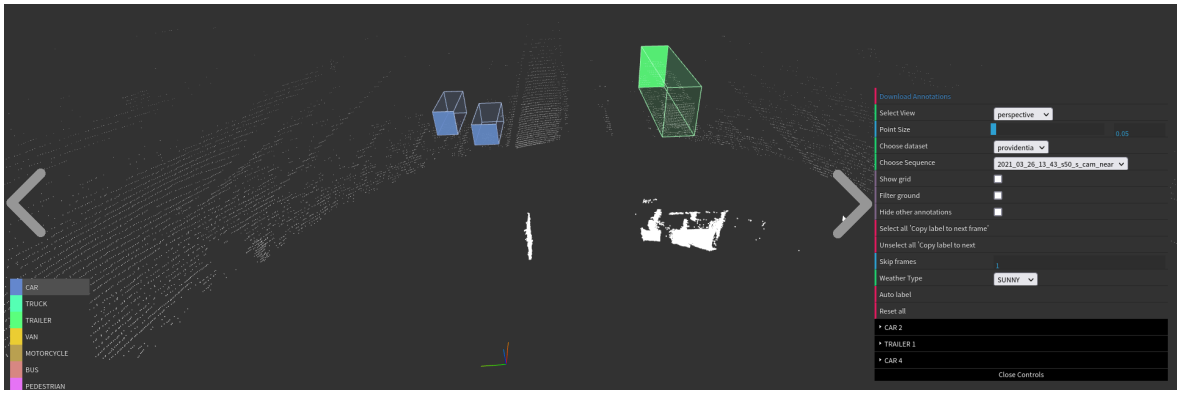

Figure 3.45: Auto label button in the ProAnno.

Chapter 4

Experimental details

In this chapter, we introduce the details in our proposed detector SE-ProPillars using KITTI as the main scenario in Section 4.1. We also describe our training setting for the IPS300+ [63] dataset in Section 4.2, as well as the transfer learning on the proFusion dataset in Section 4.3.

4.1 Implementation Details of SE-ProPillars

In the self-ensembling training architecture of SE-ProPillars, we need a teacher model to provide predictions as the soft target. Therefore, we have to pre-train a model with the same model structure but without the self-ensembling training. We firstly introduce the details of the model structure using the KITTI dataset as our scenario, while details of the pre-training and post-training are presented in Section 4.1.2 and 4.1.3

4.1.1 Network Details

Voxelization

In the KITTI dataset, we follow the convention of the detection range as $[0, 70.4]$ m for the x-axis, $[-40, 40]$ m for the y-axis, and $[-3, 1]$ m for the z-axis (no rotation, roads lied on the diagonal). Therefore, our range for the voxelization is set to $[0, -40, -3, 70.4, 40, 1]$ m and the voxel (pillar) size is $[0.2, 0.2, 4]$ m. We limit the maximum number of voxels up to 10,000 voxels and maximum points inside each voxel to 40 points. For voxels that contain more than 40 points, we use the farthest sampling to subsample the voxel.

Stacked Triple Attention

The stacked triple attention module does the first stage feature extraction. It works on point-wise features but organized inside pillars. The input is voxelized points with enlarged 9-dim

features, i.e. a $(P \times N \times 9)$ tensor, where P is the maximum number of voxels, up to 10,000 but varies with the input; $N = 40$ is the maximum points in a pillar, padded with zeros for insufficient pillars. The output is a $(P \times 40 \times 64)$ tensor.

Before and after each TA module, the feature dimension is not increased. In the excitation operation of point-wise and channel-wise attention, the bottleneck structure uses a reduction ratio of 8. Taking the point-wise attention as an example: The squeeze operation squeezes the $(P \times N \times C)$ input to $(P \times N \times 1)$ using max pooling; Then the first FC layer of the excitation operation reduces the point-wise dimension by 8 times, i.e. $(P \times N/8 \times 1)$; After ReLU, the second FC layer recovers it back to $(P \times N \times 1)$ as the point-wise attention score; The attention scale is computed together with the channel-wise attention score, after a similar Squeeze-and-Excitation process in the channel-wise attention. The voxel-wise attention has no bottleneck structure. The feature dimension is increased by a FC layer after the first TA module, from 9 to 64, while the second TA module and the followed FC layer keep this dimension.

Pillar Feature Net

The pillar feature net is responsible for the second stage feature extraction. The input is the $(P \times 40 \times 64)$ tensor from the stacked triple attention module. In principle, we could apply multiple PFN layers, but considering the speed, we only use one PFN layer. It generates a $(P \times 64)$ tensor as the final pillar feature, using the final max pooling because it is the last layer. The pillar feature is then scattered back to a $(64 \times 400 \times 352)$ pseudo image. The location of empty pillars is padded with zeros.

Attentive Hierarchical Middle Layers

In the AHM module, 2D convolutional layers perform on the $(64 \times 400 \times 352)$ pseudo image. Each of the three downsampling convolutional groups contains three convolutional layers (orange cubes in the first column of Figure 3.5). We denote a convolutional layer with (K, C_{in}, C_{out}, S), where K is the kernel size, C_{in} and C_{out} are input and output feature dimensions, S is the stride. From top to bottom, layers in these three downsampling groups have the following parameters: $[(3, 64, 64, 2), (3, 64, 64, 1), (3, 64, 64, 1)]$, $[(3, 64, 128, 2), (3, 128, 128, 1), (3, 128, 128, 1)]$, and $[(3, 128, 256, 2), (3, 256, 256, 1), (3, 256, 256, 1)]$. Parameters of other layers are marked in the diagram. The output feature map after the attentive addition has shape $(128 \times 200 \times 176)$. After that, we apply four same padding $(3, 128, 128, 1)$ and one $(1, 128, 128, 1)$ convolutional layers for further feature extraction.

Multi-task Head

The multi-task head takes the $(128 \times 200 \times 176)$ feature map as input. It applies four different convolutional layers separately on the feature map: a $(1, 128, 2, 1)$ convolutional layer for the bounding box classification with output shape $(200, 176, 2)$; a $(1, 128, 14, 1)$ layer for the bounding box regression with output shape $(200, 176, 14)$; a $(1, 128, 4, 1)$ layer for the direction classification with output shape $(200, 176, 4)$; a $(1, 128, 2, 1)$ layer for the IoU prediction with output shape $(200, 176, 2)$. Here, the dimensions of the output shapes are doubled: we generate two predictions from one cell on the (200×176) feature maps, for those two anchors assigned at each cell of the feature map (x-axis aligned and y-axis aligned). Output

from each layer is then decoded as the raw prediction. For the hyperparameter β in the confidence function, we use the same empirical setting of $\beta = 4$ from the original CIA-SSD. The depth factor in the distance-variant IoU-weighted NMS is set to $\sigma = \{0.0009, 0.009, 0.1, 1\}$ for BEV distances $[0, 20)$ m, $[20, 40)$ m, $[40, 60)$ m, and $[60, 70.4]$ m, separately. The threshold for the cumulative sum term cnt in DI-NMS is set to 2.6. After the DI-NMS, the model outputs the final prediction. For multi-class detection, we use multiple multi-task heads to make predictions separately on the shared feature map from the AHM module. Each multi-task head is responsible for one class. In principle, we could use one multi-task head to predict multiple similar classes, e.g. in nuScenes: motorcycle and bicycle, but in the KITTI dataset, there are no similar classes like this.

4.1.2 Pre-training on KITTI Dataset

Dataset

The KITTI dataset contains 7,481 training samples and 7,518 test samples. We follow the convention of splitting the training set, with 3,712 frames for training and 3,769 frames for validation. We evaluate our model on KITTI with single-class detection (Car category) and also multi-class detection (Car, Pedestrian, and Cyclist), see Chapter 5 for the results.

Data Augmentation

In the data augmentation, we use both the traditional augmentation methods and the shape-aware data augmentation module. We take the mix-up data augmentation from SECOND [69]: generate a database of all ground truth; then randomly sample ground truth and add them to the current point cloud scene; a collision test is performed to avoid unreasonable cases. In this way, the number of objects is increased. For the Car category, we sample up to 15 objects. For the Pedestrian and Cyclist class, the maximum number is 10. Traditional methods also include 1) object level: randomly rotate ground truth boxes by a factor sampled uniformly between $[-\pi/2, \pi/2]$; randomly add $\mathcal{N}(0, 0.5)$ Gaussian noise to points in ground truth boxes; and 2) point cloud level: randomly flip the y-axis of whole point cloud with probability 0.5; randomly rotate the whole point cloud with a uniformly sampled factor $[-\pi/4, \pi/4]$; randomly scale the whole point cloud, scaling factor drawn from the uniform distribution $[0.95, 1.05]$.

In addition, the shape-aware data augmentation uses the following parameters: for the Car class: the random dropout rate is 0.25, random swap rate 0.1 and only swap pyramids containing more than 50 points, random sparsifying rate 0.05 with threshold 50 points; for the Pedestrian and Cyclist class: the random dropout rate is 0.2, random swap rate 0.1 with threshold 10 points, random sparsifying rate 0.1 with threshold 25 points. In the pre-training, we test the result with or without shape-aware data augmentation. In the post-training, the shape-aware data augmentation is compulsory for providing more noisy samples to the student model. We apply random flip, rotation, and scaling to the teacher predictions as mentioned in Section 3.1.8.

Anchor

We basically follow the common setting of anchor assignment in the KITTI dataset. The anchor range is the same as the detection range. Since the feature map size is (200×176), the anchor strides at the x and y-axis are 0.4 m. We assign all anchor centers at the average z-coordinate level of each category, Car: -1 m, Pedestrian: -0.6 m, Cyclist: -0.6 m. At the center of each anchor cell, we assign two anchors with yaw angle 0 degree and 90 degree, in order to match ground truth with different orientations. The Car class has an anchor size $[1.6, 3.9, 1.56]$ m, matched IoU threshold 0.6, unmatched threshold 0.45; Pedestrian: anchor size $[0.6, 0.8, 1.73]$ m, matched threshold 0.4, unmatched threshold 0.2; and Cyclist: anchor size $[0.6, 1.76, 1.73]$ m, matched threshold 0.4, unmatched threshold 0.2. For the Car class, the number of positive samples in each frame varies between 60 to 90, and for Pedestrians and Cyclists, positive samples are around 10. We sample 512 positive and negative samples in total in training.

Loss

In the pre-training, the model is supervised by: the bounding box classification loss $\mathcal{L}_{cls}$ using the focal loss, bounding box regression loss L_{box} using the smooth-L_1 loss, direction classification loss $\mathcal{L}_{dir}$ using cross-entropy loss, and the IoU prediction loss $\mathcal{L}_{iou}$ using the smooth-L_1 loss. We don't directly use the real IoU value as the regression target in the IoU prediction loss. Instead, we encode the real IoU value to be in the range of $[-1, 1]$ by: $iou_t = 2 \cdot (iou_{real} - 0.5)$, and take this re-centered value as the IoU regression target. The overall loss in the pre-training is:

$$\mathcal{L}_{pre} = \mathcal{L}_{cls} + \omega_1 \mathcal{L}_{box} + \omega_2 \mathcal{L}_{dir} + \lambda \mathcal{L}_{iou}, \tag{4.1}$$

where we set $\omega_1 = 2.0$, $\omega_2 = 0.2$ and $\lambda = 1.0$.

Optimization Hyperparameters

We pre-train our model for 80 epochs with a batch size of 2 for a fair competition with our baseline ProPillars [87]. We use the Adam optimizer with weight decay 0.01. The gradient is clipped using L_2 norm with threshold 35. Our maximum learning rate is set to 0.003.

4.1.3 Self-ensembling Training on KITTI Dataset

After the pre-training, we get a relatively good model used as the initial teacher model in the self-ensembling training. The student uses the same model structure and anchor assignment as the teacher. We post-train our model for 60 epochs with a batch size of 4, because the self-ensembling training takes much longer than the pre-training. Other optimization hyperparameters are consistent with the pre-training.

The parameters of the teacher model are updated by the student model with the exponential moving average (EMA) decay weight of 0.999. The confidence threshold τ_c in the IoU-based matching before the consistency loss is set to $\tau_c = 0.3$, and the IoU filtering threshold

$\tau_I = 0.7$. The loss for the bounding box regression in the post-training is replaced by the orientation-aware distance-IoU Loss. The hyperparameter γ in the OD-IoU loss (Equation 3.6) is set to $\gamma = 1.25$. The total loss is (same as Equation 3.7):

$$\mathcal{L}_{student} = \mathcal{L}^s_{cls} + \omega_1 \mathcal{L}^s_{OD-IoU} + \omega_2 \mathcal{L}^s_{dir} + \lambda \mathcal{L}_{iou} + \mu_t \mathcal{L}_{consist}, \tag{4.2}$$

where, μ_t is gradually increased from 0 to 1 in the first 15 epochs with a rule $\mu_t = e^{-5(1-x)^2}$; $\omega_1 = 2.0$, $\omega_2 = 0.2$ and $\lambda = 1.0$ not changed.

4.2 Training on IPS300+ Dataset

IPS300+ [63] is a newly published dataset in year 2021. Data is captured by infrastructure-mounted sensors, including four cameras and two 80-channel LiDARs. Two sets of sensors are installed on opposite sides of an intersection. The labeled data contains 6 classes: 'Car', 'Cyclist', 'Pedestrian', 'Tricycle', 'Bus', and 'Truck'. The first batch of released data contains 623 registered point cloud frames in total (around $2 \times 144K$ points per frame). The maximum scanning region is 150 m. The annotation frequency is 5 Hz. Although fewer frames than other datasets, IPS300+ is captured in a busy intersection at the evening rush hour. The average number of objects in each frame is much higher than other datasets, a comparison shown in Table 4.1. The number of annotations in the first release is already above 160K. The authors not only publish the merged point cloud frames from the two LiDARs, but also frames captured by each single LiDAR. Examples of point cloud frames with labels are shown in Figure 4.1. In the merged point cloud, areas unrelated to roads are cropped out.

Dataset	KITTI	nuScenes	Waymo	proFusion	IPS300+
avg. vehicles	4.1	20.0	26.5	3.2	263.0
avg. pedestrian	0.8	7.0	12.2	0	56.8

Table 4.1: Average object per frame in different autonomous driving dataset [63].

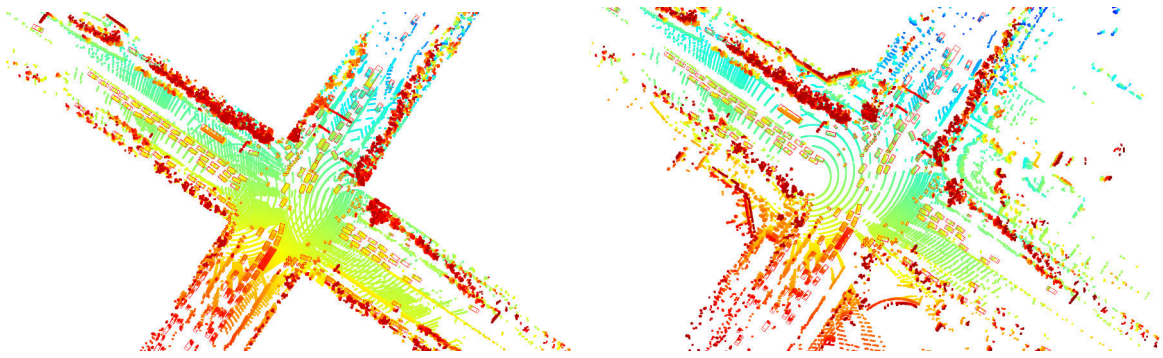

Figure 4.1: Example of IPS300+ dataset frame. Left: merged point cloud. Right: point cloud from a single LiDAR.

IPS300+ is similar to our scenario in the Providentia++ project, where LiDARs are mounted on the infrastructure above the ground. Our goal is to train on the IPS300+ dataset and use transfer learning to fine-tune on our data, because the labeling work of the Providentia++ dataset is still ongoing.

The authors of IPS300+ use a much larger detection region of $[-100, 100]$ m for the x-axis, $[-100, 100]$ m for the y-axis and $[-10, 5]$ m for the z-axis. However, our LiDARs in the Providentia++ project don't have a 360° field of view because of the occlusion by the gantry sign behind. Moreover, our LiDARs cannot cover 150 m like LiDARs in IPS300+. In order to make it fits better to our scenario, we use the center of the intersection as the origin and split the point clouds into 4 parts. Each part includes a $[0, -40, -8, 70.4, 40, -2]$ m region. We also rotate the point cloud and bounding box labels. The height of the point cloud is kept. We do this preprocessing to both registered point clouds and single LiDAR-captured point clouds. For single LiDAR-captured frames, we filter out those labels that don't contain enough points (threshold: 5). After the split, the registered split-IPS300+ dataset includes 2,492 frames, while the single-LiDAR split-IPS300+ dataset includes 4,984 frames. We divided the dataset into 60 percent as the training set, and 40 percent as the validation and test set. Some example frames are shown in Figure 4.2.

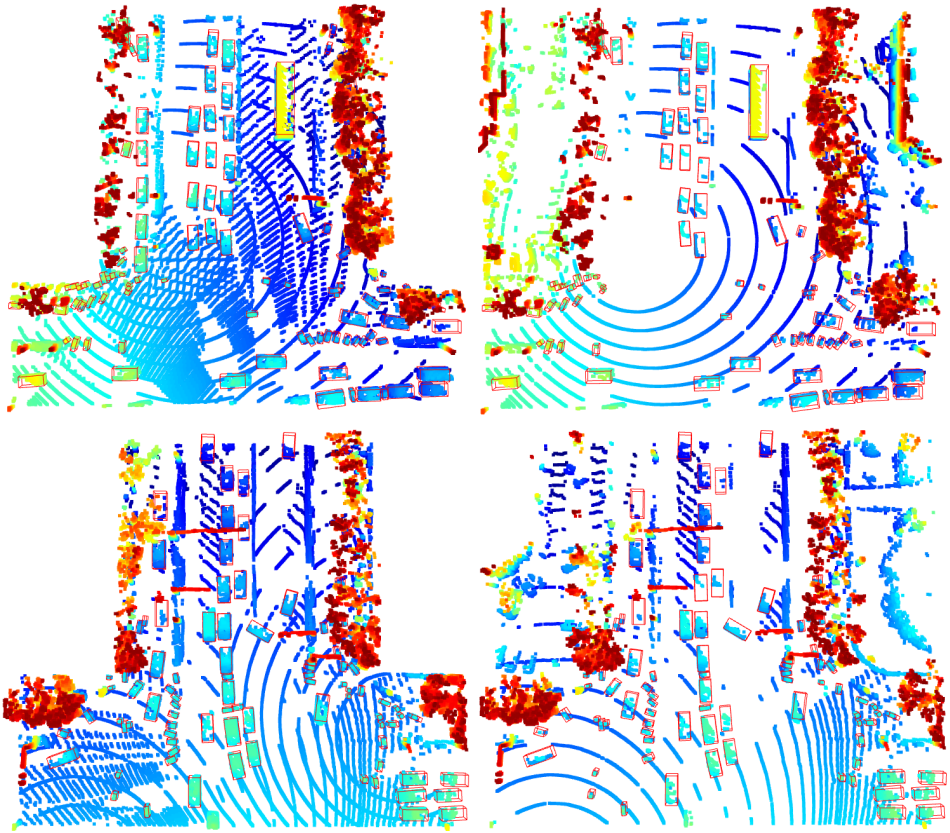

Figure 4.2: Split-IPS300+ dataset frame. Left: registered split-IPS300+. Right: single-LiDAR split-IPS300+.

We train only on the Car category because other categories either have too few samples or don't match our data categories. The distribution of dimensions of the Car class is shown in Figure 4.3. The average height is 1.62 m, average width 2.0 m, and average length 4.6 m.

The authors [63] did not release too many details of their experiment. Hence, we define our own hyperparameters. According to the size distribution, we use an anchor size of $[1.65, 4.65, 2.1]$ m, a little bit larger than the average. Because the distribution is right-skewed, we want to match more positive samples for training. During training, the number of positive samples is around 280 for registered split-IPS300+, and around 200 for single-LiDAR split-IPS300+. We increase the total sample number to 1024, to get a reasonable ratio of positive and negative samples. Experiments show using a total sample number as

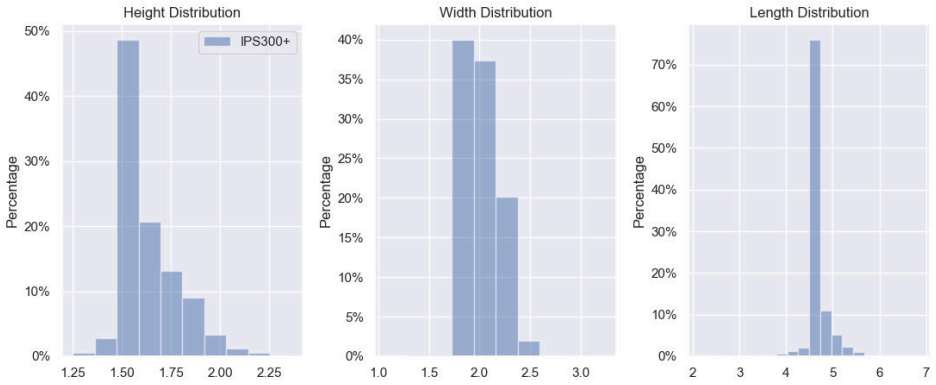

Figure 4.3: Bounding box size distribution of Car in IPS300+.

512 leads to a slow convergence problem, while as 2048 no obvious effect is observed. The z-axis anchor center is set to -5 m. Our voxelization range is $[0, -40, -8, 70.4, 40, -2]$ and the voxel (pillar) size is $[0.2, 0.2, 6]$ m. Because there are more objects per frame, we increase the maximum number of voxels to 20000, aiming to not miss any useful voxel. We use a batch size of 2 and train for 100 epochs, because the IPS300+ dataset is much more difficult than KITTI, and therefore we want more iterations. Other hyperparameters stay the same.

4.3 Transfer Learning on proFusion Dataset

The proFusion R0 (release 0) dataset is the first batch of data from the Providentia++ project's dataset. It is captured by a single OS1-64 LiDAR on the Highway A9 sensor station S50. It contains 351 labeled frames including three classes: 'Car', 'Van', and 'Trailer'. Examples of proFusion frames are shown in Figure 2.1 and 2.2. Since the number of frames is not enough for training a detector, we apply transfer learning: training a model on the IPS300+ dataset and fine-tuning on our proFusion dataset.

To transfer the feature learned from IPS300+, we have to make these two datasets as similar as possible. First of all, we shift the origin of point clouds in proFusion down 1.7 m. The level of the ground is shifted from -7.5 m to -5.8 m. In this way, the height of the origin is consistent in these two datasets. Secondly, we use the model trained on single-LiDAR split-IPS300+, because the main difference between proFusion and IPS300+ is the density of the point cloud, and the registered dataset exacerbates the differences between them. The size of bounding boxes is a key prior knowledge. To solve this problem, Zhou et al. [87] take KITTI as the base dataset and use the statistical normalization method to enlarge or shrink the ground truth boxes and associated points to the same size of objects in proFusion. However, after reviewing the proFusion dataset, we found that, in most of the labels, the height of the objects is labeled larger than they really are. That means the statistical normalization is compromising to inaccurate annotations. The reason why objects are labeled larger is because of the sparsity of OS1-64. Most of the time, only a small part of a vehicle is captured, e.g. only the bumper of a car. Therefore the size is not identifiable. When our group label the data, we mislabeled the size. That leads to an unreasonable average height of the car class — 1.83 m.

KITTI is already an 8-year-old dataset. The size of vehicles has changed in these years. That

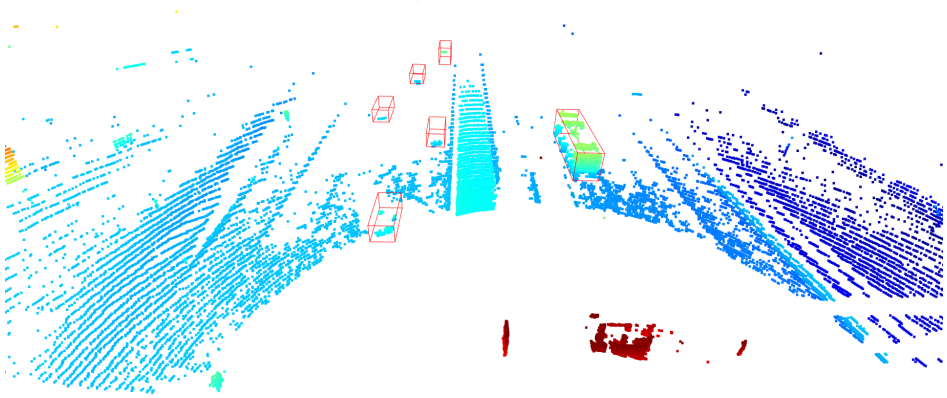

Figure 4.4: Example of corrected proFusion data frame. Illegible cars are relabeled with the average size of cars in IPS300+.

is why Zhou et al. [87] use the statistical normalization. Since both IPS300+ and proFusion are collected this year, and contemporary vehicles are similar in size. We relabel proFusion using the average size of cars in IPS300+ as the default value, i.e., height: 1.62 m, width: 2.0 m, and length: 4.6 m. This label correction fits most of the cars, except several SUVs have an identifiable higher height, where we label them with their real height. The distribution of car size after the label correction is shown in Figure 4.5.

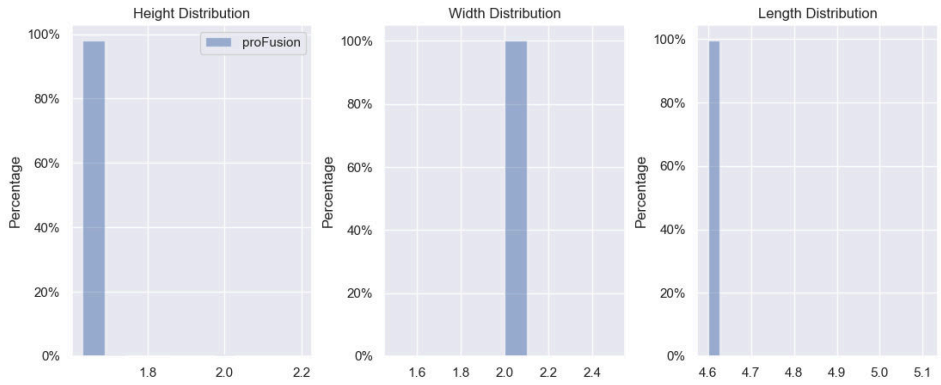

Figure 4.5: Bounding box size distribution of Car in proFusion.

We divide the 351 frames into 210 frames for the training set and 141 frames for validation and testing. We fine-tune on the Car class for 100 epochs with batch size 2. Training for 100 epochs is because we want to provide enough iterations. There are fewer frames than other datasets. Other hyperparameters remain the same as when we train on the single-LiDAR split-IPS300+ dataset.

Chapter 5

Evaluation

This chapter presents the experimental results of the SE-ProPillars on the KITTI dataset (Section 5.1), IPS300+ dataset (Section 5.2) and the transfer learning result on proFusion (Section 5.3). We show the effect of modules in our model in the form of the ablation study using KITTI as the main scenario. Our experiment is performed on a Nvidia GeForce RTX 3090 GPU.

5.1 Detector Performance on KITTI Dataset

We evaluate our detector SE-ProPillars on the KITTI validation set using the conventional metric of KITTI, i.e. the BEV and 3D mAP result under 0.7 IoU threshold with 40 recall positions. All three levels of difficulty are taken into account. We follow the NMS parameters in [82], NMS IoU threshold 0.3, confidence score threshold 0.3. We evaluate the performance on both single-class detection — Car category, and multi-class detection — Car, Pedestrian, and Cyclist. In the single-class detection, we enable the similar type, which means we train on both Car and Van categories, but treat them all as Cars. ProPillars [87] is used as our baseline.

5.1.1 Result of Single-class Pre-training

In the pre-training, we switch the codebase from ProPillars' SA-SSD to SE-SSD. We change the SSD detection head and PS-Warp extra head of ProPillars to the multi-task head. After applying the confidence function (CF), we call this model 'naive SE-ProPillars' (no training tricks). We also apply the distance-variant IoU-weighted NMS (DI-NMS) and the shape-aware data augmentation (SA-DA) module in the pre-training. Results shown in Table 5.1.

After enabled the multi-task head but without applying the CF, although the precision drops, the inference speed gets a large increase. The runtime is reduced from 32 ms to 19.4 ms. The CF in the multi-task head and the PS-Warp extra head are trying to solve the same misalignment problem between the predicted bounding boxes and corresponding classification

confidence maps, but the multi-task head is more lightweight. After applying the confidence function, an increase shows in all metrics, except the BEV easy mode keeps the same level. After adding the DI-NMS module, the precision becomes competitive with ProPillars considering the balance of precision and speed, although a relatively large gap in the 3D easy mode. The shape-aware data augmentation leads to a further increase in precision. Our model after the pre-training outperforms the baseline in almost all metrics, except a tiny 0.1% gap in the 3D easy mode. Our speed is much faster than the baseline. There is a small latency in the runtime after enabling the shape-aware data augmentation module, because as the precision increases, the computation of the DI-NMS increases.

Method	3D mAP			BEV mAP			Time(ms)
	E	**M**	**H**	**E**	**M**	**H**	
ProPillars	**92.46**	80.53	75.50	94.25	89.80	84.98	32.0
naive without CF	89.64	79.59	74.43	93.88	89.29	86.29	**19.4**
naive	89.71	79.95	75.03	93.85	89.55	86.75	20.1
naive+DI-NMS	90.25	80.48	75.60	94.18	89.84	85.11	21.4
naive+DI-NMS+SA-DA	92.38	**80.93**	**76.05**	**96.30**	**89.85**	**87.21**	22.0

Table 5.1: 3D object detection results after the pre-training of SE-ProPillars. We report the 3D and BEV mAP of the Car and Van category (similar type enabled) on the KITTI validation set under 0.7 IoU threshold with 40 recall positions. E, M, H denote easy, moderate, and hard mode separately.

5.1.2 Result of Single-class Post-training

We use the pre-trained model as the initial teacher model in the self-ensembling training architecture. Results shown in Table 5.2. In the post-training, the consistency loss is enabled to provide supervision for the student model, as well as the replacement from the traditional smooth-L_1 loss to the orientation-aware distance-IoU loss for the bounding box regression. The SA-DA module is compulsory for providing more noisy samples to the student model. After the 60 epochs of post-training, all metrics gain further improvement. Our model outperforms the baseline by $(0.64\%, 0.73\%, 2.67\%)$ 3D mAP and $(2.19\%, 0.08\%, 2.23\%)$ BEV mAP on easy, moderate, and hard difficulties respectively. At the same time, we have a much lower inference time of 22 ms, compared to our baseline of 32 ms. The inference time includes 5.7 ms for data preprocessing, 16.3 ms for network forwarding and post-processing (NMS). Our disk are relatively slow, which leads to a higher data loading time. The self-ensembling architecture is a training technique and therefore has no additional cost in the inference time. Some visualized results are shown in Figure 5.1 and 5.2.

Method	3D mAP			BEV mAP			Time(ms)
	E	**M**	**H**	**E**	**M**	**H**	
ProPillars	92.46	80.53	75.50	94.25	89.80	84.98	32.0
naive+DI-NMS+SA-DA	92.38	80.93	76.05	96.30	89.85	87.21	22.0
SE-ProPillars	**93.10**	**81.26**	**78.17**	**96.44**	**89.88**	**87.21**	**22.0**

Table 5.2: 3D object detection results after the post-training of SE-ProPillars. We report the 3D and BEV mAP of the Car and Van category (similar type enabled) on the KITTI validation set under 0.7 IoU threshold with 40 recall positions. E, M, H denote easy, moderate, and hard mode separately.

5.1.3 Result of Multi-class Detection

We use three multi-task heads to predict Car, Pedestrian, and Cyclist separately. As a convention, the IoU thresholds for Car, Pedestrian and Cyclist are 0.7, 0.5 and 0.5 respectively, similar type disabled. Our baseline did not report the result on multi-class detection. Here we only report our ablation study results of 3D mAP, shown in Table 5.3.

The detection of Pedestrian and Cyclist in KITTI is a well-known harder task. Moreover, our model is a pillar-based detector. Our voxel size is set to $[0.2, 0.2, 4]$ m, which is coarser than other detectors in the x and y-axis, not to mention the non-partition z-axis. Coarse voxels could reduce the inference time but are not good at detecting smaller objects. Because of these reasons, the precision of the Pedestrian and Cyclist is low. Since multiple detection heads are sharing the same backbone and middle layers, the shared part is solving a harder feature extraction problem. That makes the model to not focus on a single class and therefore leads to a precision drop in the Car class.

Method	Cars			Pedestrians			Cyclists		
	E	**M**	**H**	**E**	**M**	**H**	**E**	**M**	**H**
naive w/o CF	85.56	74.13	70.85	36.21	31.00	27.73	78.97	60.05	56.10
naive	86.54	74.69	71.47	37.36	32.07	28.10	83.29	63.28	59.07
naive+SA-DA	86.04	74.13	69.28	**47.41**	40.60	37.17	**85.99**	65.08	60.75
SE-ProPillars	**89.01**	**77.07**	**72.26**	46.91	**41.09**	**37.44**	82.66	**65.94**	**61.48**

Table 5.3: 3D multi-class detection results of SE-ProPillars. We report the 3D mAP of the Car, Pedestrian and Cyclist category on the KITTI validation set under 0.7, 0.5 and 0.5 IoU threshold separately, with 40 recall positions. E, M, H denote easy, moderate, and hard mode separately. The DI-NMS is removed from the SE-ProPillars in the multi-class detection.

After using the confidence function to correct the confidence map, the precision gets an increase. The shape-aware data augmentation brings further improvement and makea the model pay more attention to the other two classes. Experiment shows that the DI-NMS will cause a precision drop in less accurate classes, e.g. Pedestrians, while the precision for more accurate classes gets improved, e.g. Cars. The unsatisfactory effect is shown in Table 5.4. The reason is that, these imprecise classes make imprecise bounding box predictions during training, and the predicted IoU is learned to make low-value predictions. Therefore, the cumulative sum term cnt cannot reach the threshold in Step 3. Even if we reduce the threshold, the refined confidence score $s_{refined}$ is very low after multiplying the distance offset term. They will be filtered out. Consequently, we remove this module from the SE-ProPillars for the multi-class detection.

Method	Cars			Pedestrians			Cyclists		
	E	**M**	**H**	**E**	**M**	**H**	**E**	**M**	**H**
DI-NMS	88.19	74.92	71.86	27.03	20.72	18.64	80.11	57.73	53.16

Table 5.4: Unsatisfactory effect after applying the DI-NMS on the naive SE-ProPillars with SA-DA.

After the self-ensembling training, the precision of the Car class grows largely. However, the improvements on Pedestrian and Cyclist are not obvious. The reason is pretty straightforward: The self-ensembling training requires a relatively good teacher model. These less precise classes cannot provide high-quality predictions to the student model. The inference time of our SE-ProPillars (without DI-NMS) on the multi-class detection is 25 ms.

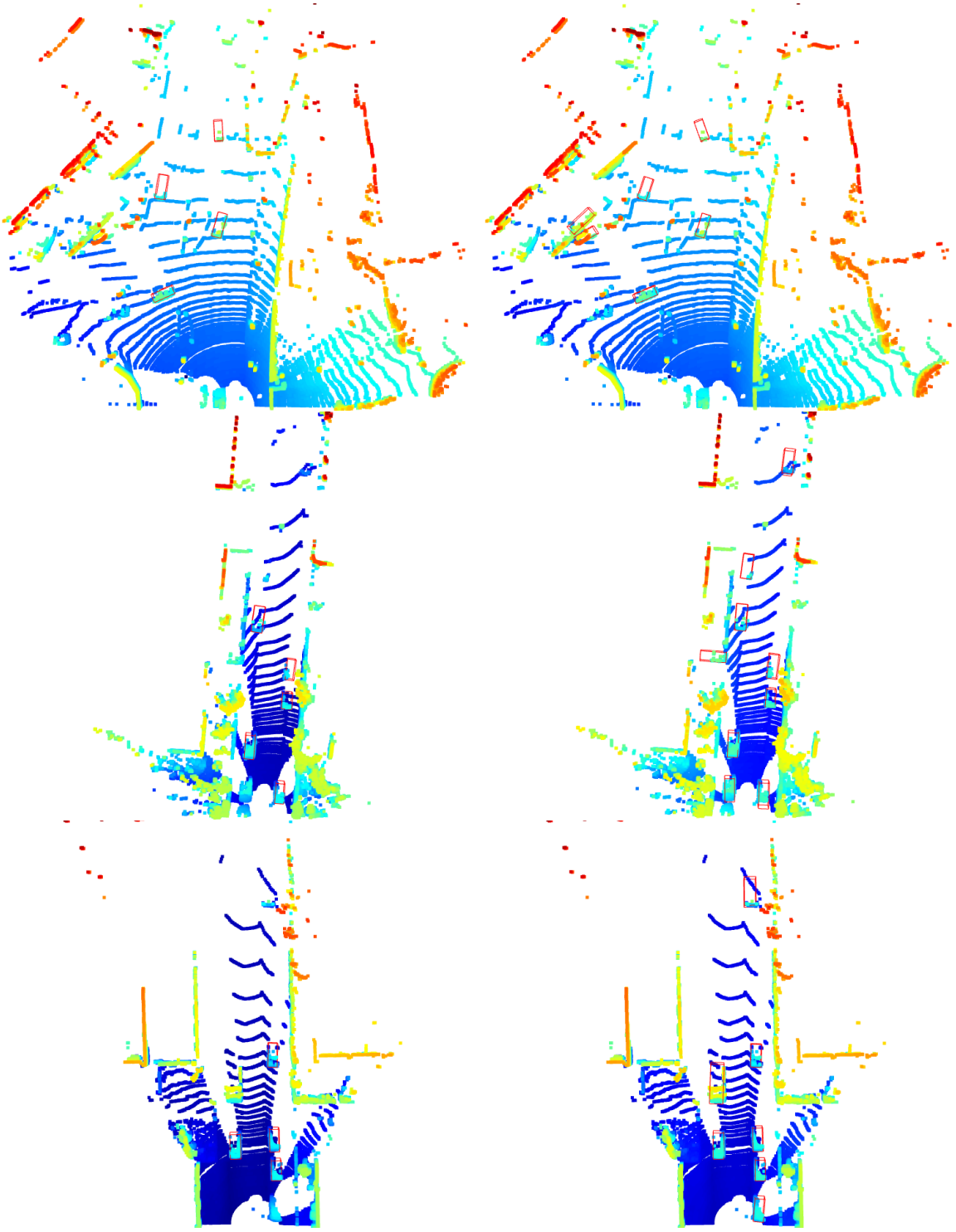

Figure 5.1: BEV visualization of single-class detection results on KITTI. Left: ground truth bounding boxes. Right: predicted bounding boxes. In KITTI, some ground truth boxes are filtered out because they are smaller than 25 pixels in height in the corresponding captured image, or completely out of the image. Some of these filtered objects are still detected by our SE-ProPillars.

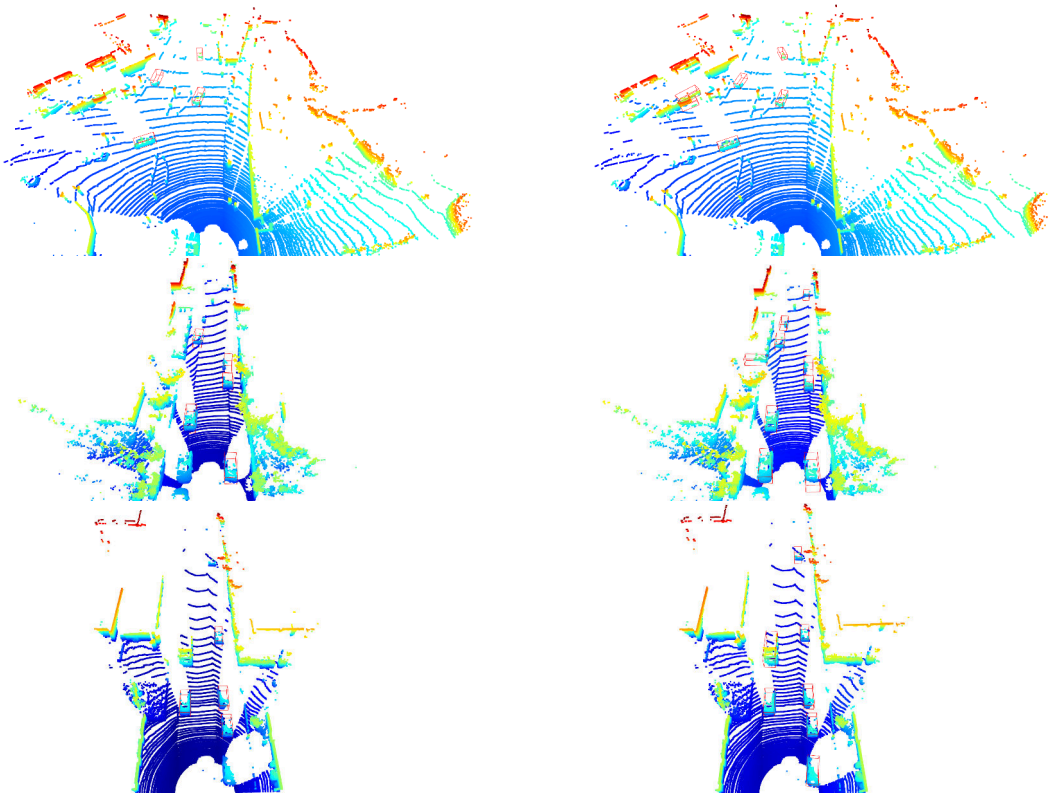

Figure 5.2: 3D front view visualization of single-class detection results on KITTI. Left: ground truth bounding boxes. Right: predicted bounding boxes. In the example at the bottom, a larger vehicle is detected as a Car.

5.2 Result on IPS300+ Dataset

The goal of training on the IPS300+ dataset is to transfer the learned feature to our proFusion dataset. We evaluate on the Car class using the BEV and 3D mAP result under 0.7 and 0.5 IoU threshold with 40 recall positions. To reduce the gap between IPS300+ and proFusion, we transfer only the model trained on the single-LiDAR split-IPS300+, because the registered one is much denser. The single-LiDAR split-IPS300+ has 2,990 frames as the training set and 1,994 frames as the validation and test set. We also report the result on the registered split-IPS300+ for comparing with the result reported by Wang et al. [63]. The registered split-IPS300+ has 1,500 training samples and 992 testing and validation samples. We take the confidence threshold parameter as 0.1 like in nuScenes [9], but we increase the NMS IoU threshold to 0.5. In IPS300+ there are more objects per frame and the density of point clouds is higher. Hence, we don't want to remove predicted boxes that are close to each other but are from two different cars .

In the IPS300+ dataset, the detection of Cars is a more difficult problem than in KITTI. The LiDARs are mounted higher, which makes the perception of height harder, because the distribution of z-coordinates has a large range of 6 m, the center (-5 m) shifts more from zero, and there are no positive z-coordinates. In addition, the number of objects per frame is much higher. It is difficult to detect them all. As a result, the authors report only a 68.9% 3D mAP within a distance of 70 meters, using the large detection region mentioned before.

From the experiment results of multi-class detection on the KITTI dataset, we find that the DI-NMS does not perform well on imprecise classes. In addition, the self-ensembling post-training also does not lead to a substantial improvement on imprecise classes. Considering these factors, we decide to remove both the DI-NMS and the self-ensembling training architecture. We use only the naive SE-ProPillars with the SA-DA module as our model in training on split-IPS300+.

5.2.1 Result on Registered split-IPS300+

In the IPS300+ dataset, almost all Cars are labeled with occlusion level 1.0 (partly occluded), therefore the authors report the precision on this occlusion level. The original result is trained on a $[-100, -100, 100, 100]$ m region in the x and y-axis. No rotation made, therefore the maximum detection distance is up to 150 meters in the diagonal. Their precision is reported from 10 m to 150 m with an interval of 10 m. We compare our result on the registered split-IPS300+ with the original result at a 70 m distance.

Metric	70m 3D mAP		70m BEV mAP	
IoU threshold	**0.7**	**0.5**	**0.7**	**0.5**
IPS300+ original	68.90	97.82	95.53	98.16
Registered split-IPS300+	63.58	87.52	78.34	89.42

Table 5.5: 3D object detection results of naive SE-ProPillars with SA-DA. We report the 3D and BEV mAP of Car on the registered split-IPS300+ test set under 0.7 and 0.5 IoU threshold, with 40 recall positions.

As shown in Table 5.5, our result is lower than the result on the original IPS300+. Firstly, the split of the dataset makes the input point cloud have a different perspective and density. In one direction of the intersection, more points are captured, while the other one contains fewer points. This is easily to see from the size of the split point cloud files. Different perspectives and densities force the model to deal with heterogeneous input. Compared to feeding stable input with the same perspective, the split-IPS300+ is more difficult. Moreover, the original result is trained on a larger region. That means the model can learn from objects at a further distance. That helps the model to deal with sparse objects within 70 meters. Most importantly, after splitting the dataset, some vehicles are truncated at the border, as shown in Figure 4.2. All these factors lead to a lower precision in both 0.7 and 0.5 IoU threshold. Some visualized detection results shown in Figure 5.3.

5.2.2 Result on Single-LiDAR split-IPS300+

We use the same setting, namely naive SE-ProPillars with SA-DA, to train on the single-LiDAR split-IPS300+, results shown in Table 5.6.

The single-LiDAR dataset is sparser than the registered one, and frames from two single LiDARs installed at different positions cause more heterogeneity in the input. As a result, the precision is lower than the same model on the registered split-IPS300+. In spite of this, the single-LiDAR dataset is more similar to our proFusion dataset. In the registered split-IPS300+, in addition to the density, some vehicles are captured by two LiDARs from two

Metric	3D mAP		BEV mAP	
IoU threshold	**0.7**	**0.5**	**0.7**	**0.5**
Registered split-IPS300+	62.19	87.52	78.34	89.42
Single-LiDAR split-IPS300+	54.91	83.76	74.57	85.97

Table 5.6: 3D object detection results of naive SE-ProPillars with SA-DA. We report the 3D and BEV mAP of Car on the single-LiDAR split-IPS300+ test set under 0.7 and 0.5 IoU threshold, with 40 recall positions.

sides, which is not a usual case in our single LiDAR dataset. The model trained on the single-LiDAR split-IPS300+ is used in the transfer learning on proFusion. Some visualized detection results on split-IPS300+ are shown in Figure 5.4.

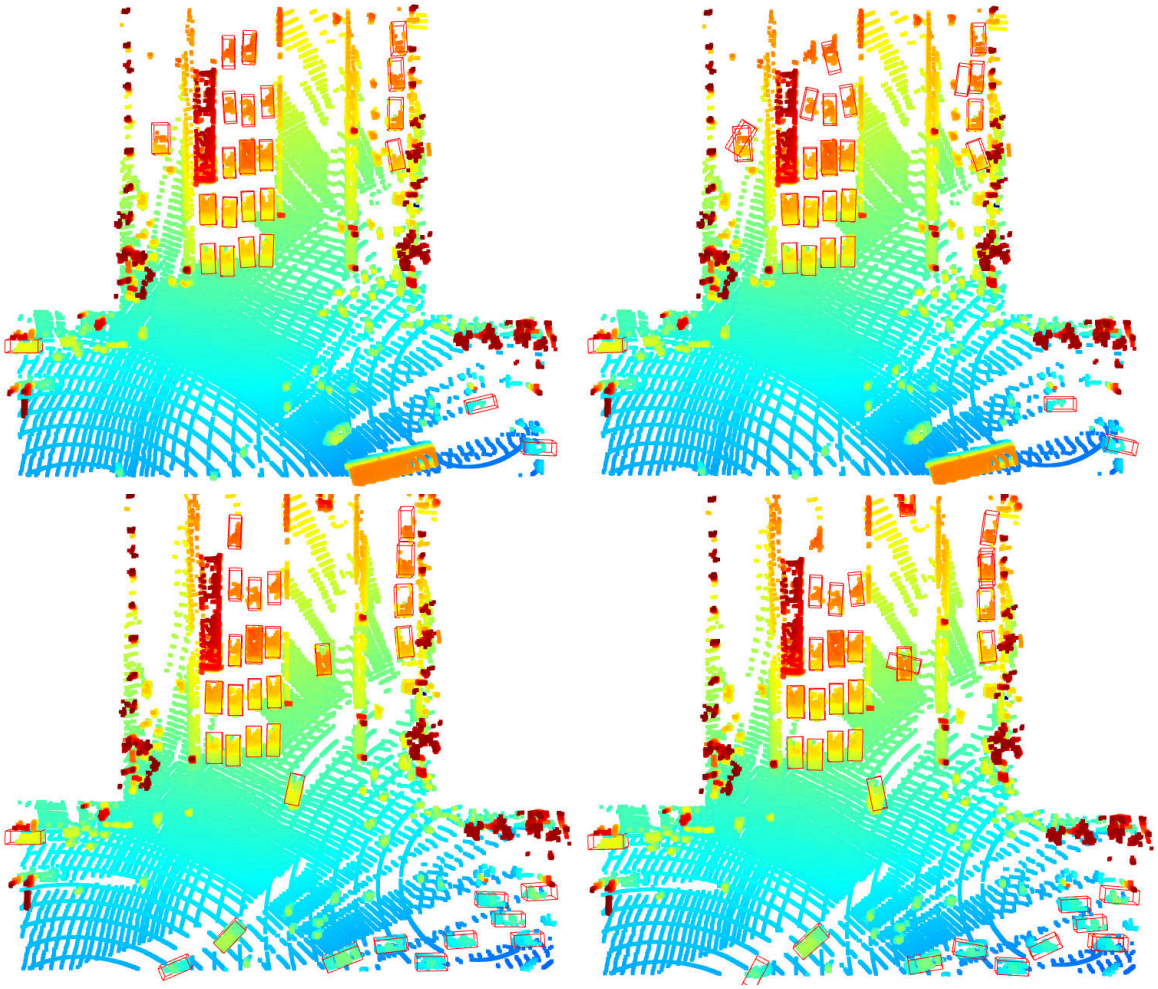

Figure 5.3: Visualization of detection results on registered split-IPS300+. Left: ground truth bounding boxes. Right: predicted bounding boxes.

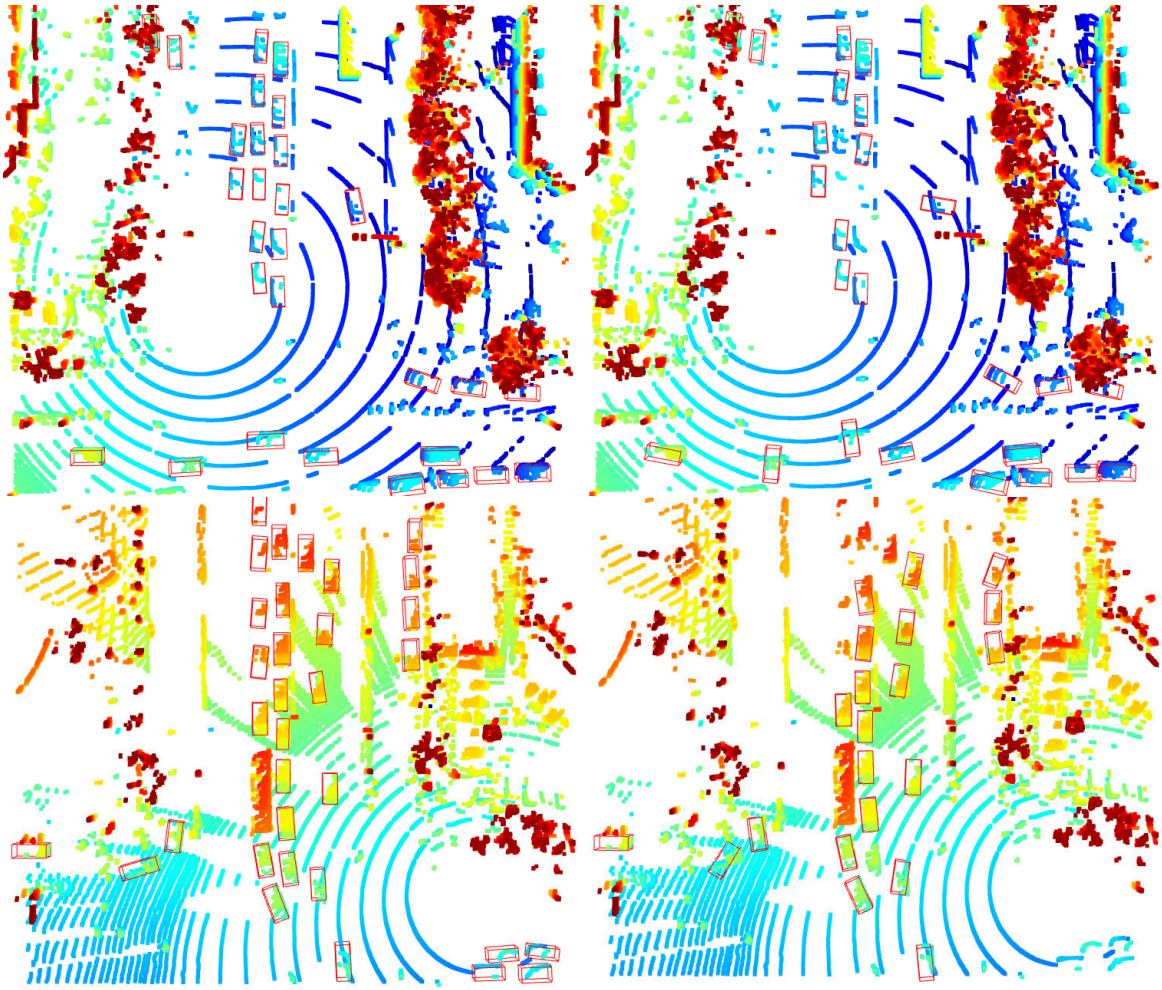

Figure 5.4: Visualization of detection results on single-LiDAR split-IPS300+. Left: ground truth bounding boxes. Right: predicted bounding boxes.

5.3 Result of Transfer Learning on proFusion

We use the naive SE-ProPillars with SA-DA module trained on single-LiDAR split-IPS300+ as our model, fine-tune on the 210 training samples for 100 epochs, and test on the 140 testing samples. We report our result using 3D and BEV under IoU threshold 0.5 and 0.25 like Zhou et al. [87] did. The confidence score is limited with threshold 0.1, but we again decrease the NMS IoU threshold to 0.2, exactly the same setting as the convention in nuScenes. Since our proFusion is captured on the Highway A9, the distance between cars is further and there are no parked cars. We also test the case of using 0.1 as the NMS threshold. The result is shown in Table 5.7.

Zhou et al. [87] report only the BEV mAP under 0.25 IoU threshold. Note that, their model is transferred from the KITTI dataset, and they enable the similar type like we train on KITTI in Section 5.1. As mentioned before, in IPS300+ there is no Van class. We cannot use the same similar type trick. We also use a different default object size and different way to divide the dataset. Therefore this comparison is just a simple reference, not a fair competition.

Metric	3D mAP		BEV mAP	
IoU threshold	**0.5**	**0.25**	**0.5**	**0.25**
ProPillars	N/A	N/A	N/A	47.56
Ours NMS IoU 0.1	30.13	50.09	40.21	51.53
Ours NMS IoU 0.2	31.03	48.88	39.91	49.68

Table 5.7: 3D object detection results of naive SE-ProPillars with SA-DA. We report the 3D and BEV mAP of Car on the proFusion test set under 0.7 and 0.5 IoU threshold, with 40 recall positions.

Even though we take the single-LiDAR split-IPS300+, shift the origin and relabel our proFusion dataset with a better default size, there are still two problems that cannot be resolved — the sparsity of point clouds and the insufficient data samples in proFusion. A comparison of the sparsity between single-LiDAR split-IPS300+ and proFusion is shown in Figure 5.5. Due to these two limitations, the result of the transfer learning is unsatisfactory.

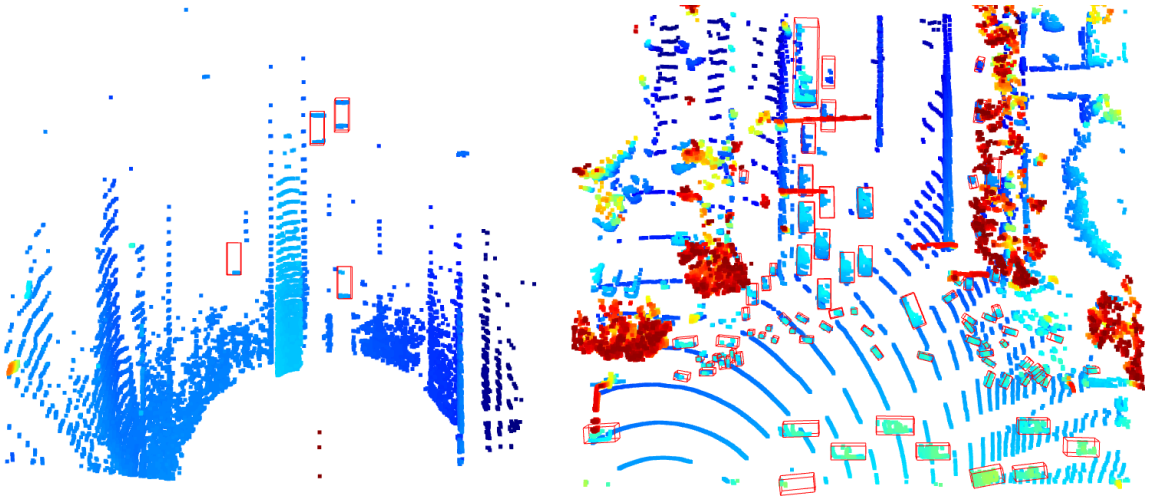

Figure 5.5: Comparison of sparsity between proFusion and IPS300+ Left: frame from proFusion. Right: frame from single-LiDAR split-IPS300+.

We define all hyperparameters by ourselves in the fine-tuning. Due to the lack of data samples, we cannot divide another test set, but we do verify the training curve to make sure we are not overfitting. The precision on the validation set didn't decrease during the 100 epochs of training. After the 100 epochs training, we also test with 50 and 80 epochs, their final precision is lower than 100 epochs. Some visualized detection results are shown in Figure 5.6.

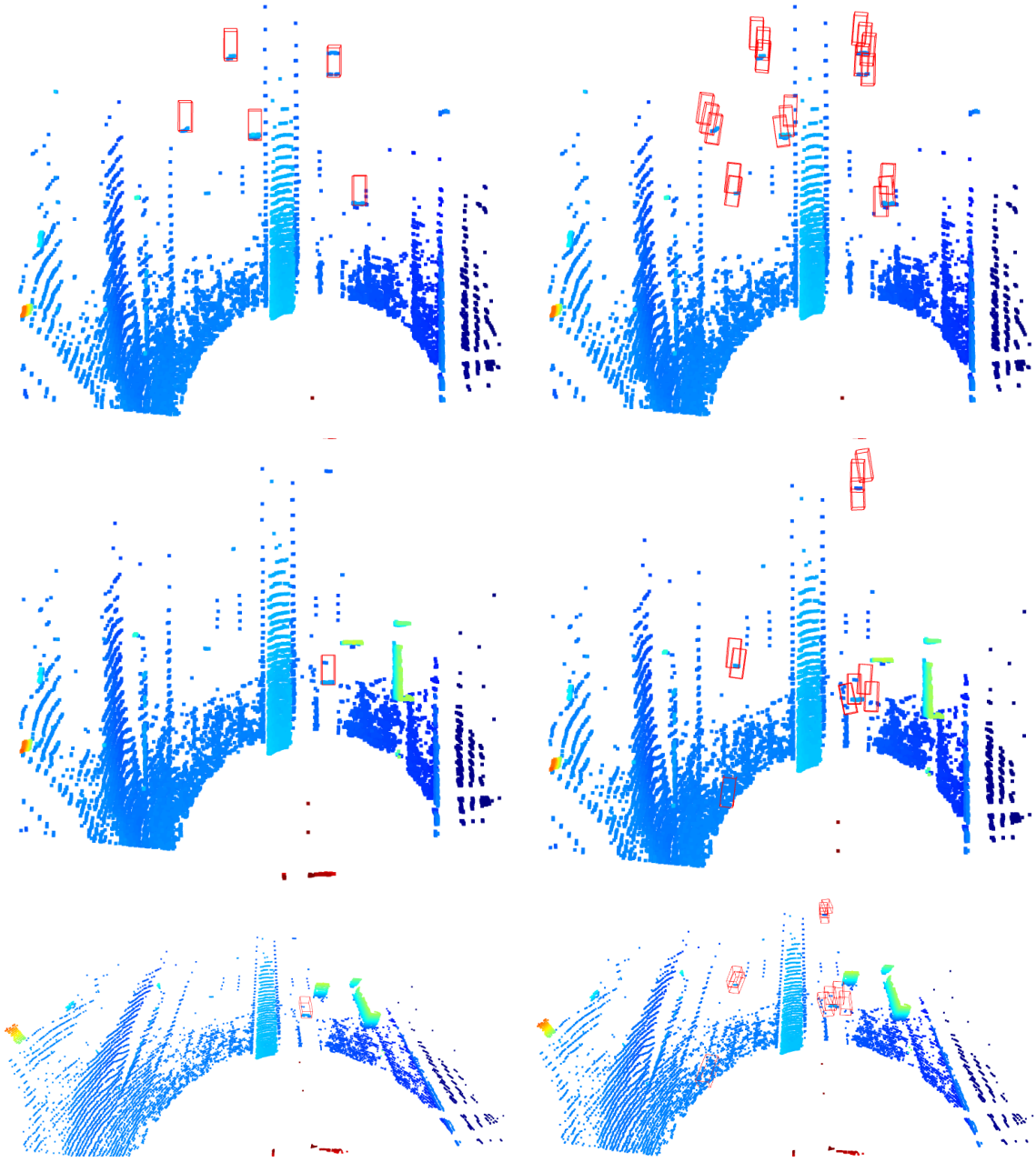

Figure 5.6: Visualization of detection results on proFusion. Left: ground truth bounding boxes. Right: predicted bounding boxes. The four pictures below are from the same frame: two bounding boxes are filtered out because they contain fewer points than the threshold (5 points), but they are perceived by our model.

Chapter 6

Discussion & Conclusion

In conclusion, we mainly focus on LiDAR-only 3D object detection in this thesis. According to the needs of the Providentia++ project, we also work on some small tasks that are related to LiDAR sensors and point clouds: point cloud registration, super-resolution, visualization, and the simple auto labeling function.

In the object detection task, we use ProPillars [87] as our baseline. We don't add too many modifications to the neural network structure, because the pillar-based backbone is quite efficient and also very popular in academia and in the industry. We only replace the detection head with a more lightweight multi-task head. We add two training techniques to our baseline: the shape-aware data augmentation module and the self-ensembling training architecture. Experiments on KITTI show that these two modules bring an increase to all evaluation metrics of precision. Our SE-ProPillars achieves faster speed and higher accuracy. However, experiments show the DI-NMS and self-ensembling training does not perform well on less precise classes in the multi-class detection. They are good at refining a relatively good model, but not good at training an underfitted model. In an ablation study: self-ensembling architecture enabled and the OD-IoU loss disabled, we find that the precision didn't improve as we thought it would. This ablation study is also not reported in the SE-SSD paper. We think this is because we didn't provide a 'fresh' supervision of regression, and therefore it leads to a little bit of overfitting in the post-training.

We also attempt to use transfer learning to improve the model performance on our proFusion dataset, which has a limited number of samples. Although results on IPS300+ and proFusion are both unsatisfactory, the topic of infrastructure-mounted LiDAR object detection deserves more attention. According to the authors of IPS300+, there will be more batches of data including different scenarios. We hope there will be more research on this dataset in the future.

In the point cloud registration task, we design an initial registration and continuous registration method for LiDARs installed on infrastructure. An alternative way is to use the Real-time Kinematic (RTK) positioning device to measure the relative position of LiDARs. Then, we could use this relative position as the initial transformation matrix to make it more reliable.

In the point cloud upsampling method, we generate a synthetic dataset — proSynth and train

our super-resolution model on it. We use data augmentation methods to increase the ability of generalization. If we could access a 128-channel LiDAR for several hours and capture a real point cloud dataset, the performance of our model on real-world data will be improved considerably.

We also implement two point cloud visualization tools for point cloud scans and add a simple auto labeling function to the labeling tool of the Providentia++ project.

Providentia++ is an interesting and prospective project. It looks forward to the future of autonomous driving, and it is continuously adopting new technologies to move toward that future. I believe the Providentia++ project will be a success. I am honored to do my master thesis as part of this project.

Chapter 7

Future Work

First of all, pillar-based detectors are desirable, considering the balance of speed and accuracy. That is why many applications in the industry use PointPillars and its variants as the detector. Since our model has a high inference speed, we have the potential to sacrifice some speed for accuracy. $H^2$3D R-CNN [16] is a two-stage detector, but it achieves an extraordinary inference speed of 27 ms. It applies a bird's eye view projection and a cylindrical coordinate-based projection on the raw point cloud, while our detector is essentially a bird's eye view projection to pillars and the pseudo image. Therefore, we could extend our pillar backbone to a multi-view backbone with the cylindrical coordinate projection for better feature extraction and perception. In SA-Det3D [6], a full self-attention module (FSA) and a deformable self-attention module (DSA) are proposed and tested on different types of backbone networks. For pillar-based networks, they are in parallel with the 2D convolutional middle layers. The resulting context feature from FSA and DSA and the feature from the middle layers are then concatenated before the detection head. These two modules may also lead to further improvement in precision. In addition, the speed of SE-ProPillars allows us to add an additional second stage module like LiDAR R-CNN [37]. Li et al. [37] claim that their additional second stage module can increase 2 to 4 % of precision with only 4.5 ms additional cost, and the module is universal to multiple types of single-stage detectors.

Training techniques are proved to be efficient in this thesis. We have the shape-aware data augmentation module, the self-ensembling training architecture, and the OD-IoU loss. These training techniques bring no cost to the inference time and can be used for free. One of our early experiments is to add the detachable auxiliary network [26] to CIA-SSD. The improvement was not obvious, but these training techniques are worth trying in all appropriate models. Although they make the training time become much longer. In the self-ensembling training of the multi-class model on KITTI, it takes around 26 hours to train for 60 epochs!

Considering the sparsity in the proFusion dataset, a two-stage detector with a point cloud completion process on 3D proposals, e.g. SIENet [38], may perform better than single-stage detectors. Moreover, a multi-modal method using LiDARs and cameras can use the image information to fight with the sparsity. Multi-modal models could also help to detect smaller objects like pedestrians. In the subsequent data batches of proFusion, we are planing to include the intersection in the urban area. There will be pedestrians and cyclists there.

As to the point cloud registration task, our code is based on the Open3D library. In Dec 2021,

the latest 0.14.0 version of Open3D was released, and it now supports G-ICP. We could use G-ICP for a faster and more robust continuous registration.

For the point cloud upsampling task, we are using a channel-wise upsampling pattern, while point-wise upsampling may also be desirable. Moreover, the collection of the dataset for point cloud super-resolution needs no manual label at all. Therefore, it could be designed as a self-supervised online machine learning topic: we could keep collecting high-resolution data and keep training with new data. An application could be, for example, we have both high and low-resolution LiDARs, or we ask to get a loan or sample of a high-resolution LiDAR from a LiDAR manufacturer or distributor. Then we could train on data collected by the high-resolution one to make the low-resolution lidar provide upsampled point cloud scans.

Auto labeling basically has no requirement on the inference time, and only precision is important. Therefore, we should jump out of the single-stage detector and try more accurate methods, for example, utilizing temporal information and making predictions based on multiple previous and later frames; embedding a point cloud completion sub-module; or combining traditional algorithms like removing the ground and matching remaining parts with point cloud registration templates, if the RMSE is lower than a threshold, then a label is outputted.

Acknowledgement

Upon the completion of this thesis, I would like to express my appreciation to those who have offered me invaluable help during my study.

First, the completion of my thesis would not have been possible without the guidance of my supervisor, Walter Zimmer. He is always patient, and he could always give me useful suggestions. Many thanks to my student colleagues, Xingcheng Zhou, Maximilian Fortkord, and Xavier Diaz, for their help and inspiration to my thesis. Thanks also to Markus Weber, I learned a lot from the experience at Filics.

Second, I very much appreciate my friends. To my roommate Junjie Ming: thank you for finding me an apartment and helping me in daily life. To Jigao Luo: you are my mentor at TUM. Every time I have a problem, you are the first person I turn to for advice. To Zhaowei Zheng: I'm glad to share the scenery of the cliffs of Santorini with you. To my friends since high school, Zhiyuan Li, Ruidong Huang, Tianhao Fu, Jiaqi Wang, Yuxuan Wang, Xianhao Jin, Hechen Yu, Han Ding, and Sitong Li: I really miss the time we spent together. I can't wait to return to China and get drunk with you guys. Special thanks to my best friend Jingwei Duan, she has always been there during every struggle and all my successes.

Last, I'm extremely grateful to my parents and all my family members, for their love and concern. Without their support and encouragement, I couldn't achieve anything.

List of Figures

List of Tables

Bibliography

[1] Aiger, Dror, Mitra, Niloy J, and Cohen-Or, Daniel. “4-points congruent sets for robust pairwise surface registration”. In: *ACM SIGGRAPH 2008 papers*. 2008, pp. 1–10.

[2] Bae, Changsub. *KITTI_Tutorial*. https://github.com/windowsub0406/KITTI_Tutorial. 2018.

[3] Bae, Kwang-Ho. “Automated registration of unorganised point clouds from terrestrial laser scanners”. PhD thesis. Curtin University, 2006.

[4] Bentley, Jon Louis. “Multidimensional binary search trees used for associative searching”. In: *Communications of the ACM* 18.9 (1975), pp. 509–517.

[5] Besl, Paul J and McKay, Neil D. “Method for registration of 3-D shapes”. In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–606.

[6] Bhattacharyya, Prarthana, Huang, Chengjie, and Czarnecki, Krzysztof. “SA-Det3D: Self-Attention Based Context-Aware 3D Object Detection”. In: *arXiv:2101.02672* (2021).

[7] Bouaziz, Sofien, Tagliasacchi, Andrea, and Pauly, Mark. “Sparse iterative closest point”. In: *Computer graphics forum*. Vol. 32. 5. Wiley Online Library. 2013, pp. 113–123.

[8] Brenner, C, Dold, C, and Ripperda, N. “Coarse orientation of terrestrial laser scans in urban environments”. In: *ISPRS journal of photogrammetry and remote sensing* 63.1 (2008), pp. 4–18.

[9] Caesar, Holger et al. “nuscenes: A multimodal dataset for autonomous driving”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[10] Chen, Qi et al. “Object as hotspots: An anchor-free 3d object detection approach via firing of hotspots”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 68–84.

[11] Chen, Yang and Medioni, Gérard. “Object modelling by registration of multiple range images”. In: *Image and vision computing* 10.3 (1992), pp. 145–155.

[12] Chen, Yilun et al. “Fast point r-cnn”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9775–9784.

[13] Deng, Haowen, Birdal, Tolga, and Ilic, Slobodan. “Ppfnet: Global context aware local features for robust 3d point matching”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 195–205.

[14] Deng, Haowen, Birdal, Tolga, and Ilic, Slobodan. “3d local features for direct pairwise registration”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3244–3253.

[15] Deng, Jiajun et al. “Voxel R-CNN: Towards High Performance Voxel-based 3D Object Detection”. In: *arXiv preprint arXiv:2012.15712* (2020).

[16] Deng, Jiajun et al. “From Multi-View to Hollow-3D: Hallucinated Hollow-3D R-CNN for 3D Object Detection”. In: *IEEE Transactions on Circuits and Systems for Video Technology* (2021).

[17] developers, ONNX Runtime. *ONNX Runtime*. https://onnxruntime.ai/. Version: x.y.z. 2021.

[18] Dosovitskiy, Alexey et al. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[19] Fischler, Martin A and Bolles, Robert C. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[20] Fitzgibbon, Andrew W. “Robust registration of 2D and 3D point sets”. In: *Image and vision computing* 21.13-14 (2003), pp. 1145–1153.

[21] Gavin, Henri P. “The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems”. In: *Department of Civil and Environmental Engineering, Duke University* (2019), pp. 1–19.

[22] Geiger, Andreas et al. “Vision meets robotics: The kitti dataset”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[23] Gojcic, Zan et al. “The perfect match: 3d point cloud matching with smoothed densities”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5545–5554.

[24] Graham, Benjamin, Engelcke, Martin, and Van Der Maaten, Laurens. “3d semantic segmentation with submanifold sparse convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 9224–9232.

[25] Grinberg, Miguel. *Flask web development: developing web applications with python*. " O’Reilly Media, Inc.", 2018.

[26] He, Chenhang et al. “Structure aware single-stage 3d object detection from point cloud”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11873–11882.

[27] He, Kaiming et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[28] Hu, Jie, Shen, Li, and Sun, Gang. “Squeeze-and-excitation networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141.

[29] Huang, Xiaoshui et al. “A comprehensive survey on point cloud registration”. In: (2021).

[30] Iqbal, Haris. *PlotNeuralNet*. https://github.com/HarisIqbal88/PlotNeuralNet. 2020.

[31] Kloeker, Laurent, Kotulla, Christian, and Eckstein, Lutz. “Real-Time Point Cloud Fusion of Multi-LiDAR Infrastructure Sensor Setups with Unknown Spatial Location and Orientation”. In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2020, pp. 1–8.

[32] Krämmer, Annkathrin et al. “Providentia-a large scale sensing system for the assistance of autonomous vehicles”. In: *Robotics: Science and Systems (RSS), Workshop on Scene and Situation Understanding for Autonomous Driving*. 2019.

[33] Lang, Alex H et al. “Pointpillars: Fast encoders for object detection from point clouds”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12697–12705.

[34] Li, Jiale et al. “P2V-RCNN: Point to Voxel Feature Learning for 3D Object Detection From Point Clouds”. In: *IEEE Access* 9 (2021), pp. 98249–98260.

[35] Li, Ruihui et al. “Pu-gan: a point cloud upsampling adversarial network”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 7203–7212.

[36] Li, Yangyan et al. “Pointcnn: Convolution on x-transformed points”. In: *Advances in neural information processing systems* 31 (2018), pp. 820–830.

[37] Li, Zhichao, Wang, Feng, and Wang, Naiyan. “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.

[38] Li, Ziyu et al. “SIENet: Spatial Information Enhancement Network for 3D Object Detection from Point Cloud”. In: *arXiv preprint arXiv:2103.15396* (2021).

[39] Lin, Tsung-Yi et al. “Focal loss for dense object detection”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[40] Liu, Baoyuan et al. “Sparse convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 806–814.

[41] Liu, Wei et al. “Ssd: Single shot multibox detector”. In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[42] Liu, Zhe et al. “Tanet: Robust 3d object detection from point clouds with triple attention”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07. 2020, pp. 11677–11684.

[43] Lu, Weixin et al. “Deepvcp: An end-to-end deep neural network for point cloud registration”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 12–21.

[44] Myronenko, Andriy and Song, Xubo. “Point set registration: Coherent point drift”. In: *IEEE transactions on pattern analysis and machine intelligence* 32.12 (2010), pp. 2262–2275.

[45] Noh, Jongyoun, Lee, Sanghoon, and Ham, Bumsub. “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. 14605–14614.

[46] NVIDIA. *TensorRT*. 2018. URL: https://developer.nvidia.com/tensorrt (visited on 2021).

[47] Qi, Charles R et al. “PointNet++ deep hierarchical feature learning on point sets in a metric space”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 5105–5114.

[48] Qi, Charles R. et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[49] Qian, Guocheng et al. “Pu-gcn: Point cloud upsampling using graph convolutional networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 11683–11692.

[50] Riegler, Gernot, Osman Ulusoy, Ali, and Geiger, Andreas. “Octnet: Learning deep 3d representations at high resolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3577–3586.

[51] Ronneberger, Olaf, Fischer, Philipp, and Brox, Thomas. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[52] Rusinkiewicz, Szymon and Levoy, Marc. “Efficient variants of the ICP algorithm”. In: *Proceedings third international conference on 3-D digital imaging and modeling*. IEEE. 2001, pp. 145–152.

[53] Rusu, Radu Bogdan, Blodow, Nico, and Beetz, Michael. “Fast point feature histograms (FPFH) for 3D registration”. In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 3212–3217.

[54] Segal, Aleksandr, Haehnel, Dirk, and Thrun, Sebastian. “Generalized-icp.” In: *Robotics: science and systems*. Vol. 2. 4. Seattle, WA. 2009, p. 435.

[55] Shan, Tixiao et al. “Simulation-based lidar super-resolution for ground vehicles”. In: *Robotics and Autonomous Systems* 134 (2020), p. 103647.

[56] Shi, Shaoshuai, Wang, Xiaogang, and Li, Hongsheng. “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.

[57] Shi, Shaoshuai et al. “From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network”. In: *IEEE transactions on pattern analysis and machine intelligence* (2020).

[58] Shi, Shaoshuai et al. “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. 10529–10538.

[59] Shi, Weijing and Rajkumar, Raj. “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.

[60] Sun, Pei 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.

[61] Uber. *XVIZ*. https://github.com/uber/xviz. 2019.

[62] Uber. *streetscape.gl*. https://github.com/uber/streetscape.gl. 2021.

[63] Wang, Huanan et al. “IPS300+: a Challenging Multimodal Dataset for Intersection Perception System”. In: *arXiv preprint arXiv:2106.02781* (2021).

[64] Wang, Yue and Solomon, Justin M. “Deep closest point: Learning representations for point cloud registration”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3523–3532.

[65] Wang, Yue et al. “Dynamic graph cnn for learning on point clouds”. In: *Acm Transactions On Graphics (tog)* 38.5 (2019), pp. 1–12.

[66] Wu, Zhirong et al. “3d shapenets: A deep representation for volumetric shapes”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920.

[67] Xu, Minjun. *XVIZ In C++*. https://github.com/mjxu96/xviz. 2020.

[68] Xu, Minjun. *CarlaViz*. https://github.com/mjxu96/carlaviz. 2021.

[69] Yan, Yan, Mao, Yuxing, and Li, Bo. “Second: Sparsely embedded convolutional detection”. In: *Sensors* 18.10 (2018), p. 3337.

[70] Yang, Heng, Shi, Jingnan, and Carlone, Luca. “Teaser: Fast and certifiable point cloud registration”. In: *IEEE Transactions on Robotics* 37.2 (2020), pp. 314–333.

[71] Yang, Jiaolong et al. “Go-ICP: A globally optimal solution to 3D ICP point-set registration”. In: *IEEE transactions on pattern analysis and machine intelligence* 38.11 (2015), pp. 2241–2254.

[72] Yang, Zetong et al. “Std: Sparse-to-dense 3d object detector for point cloud”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1951–1960.

[73] Yang, Zetong et al. “3dssd: Point-based 3d single stage object detector”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11040–11048.

[74] Ye, Shuquan et al. “Meta-PU: An Arbitrary-Scale Upsampling Network for Point Cloud”. In: *IEEE Transactions on Visualization and Computer Graphics* (2021).

[75] Yifan, Wang et al. “Patch-based progressive 3d point set upsampling”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5958–5967.

[76] Yu, Jiahui et al. “Unitbox: An advanced object detection network”. In: *Proceedings of the 24th ACM international conference on Multimedia*. 2016, pp. 516–520.

[77] Yu, Lequan et al. “Pu-net: Point cloud upsampling network”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2790–2799.

[78] Yuan, Wentao et al. “Deepgmr: Learning latent gaussian mixture models for registration”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 733–750.

[79] Zeng, Andy et al. “3dmatch: Learning local geometric descriptors from rgb-d reconstructions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1802–1811.

[80] Zhang, Yanan, Huang, Di, and Wang, Yunhong. “PC-RGNN: Point Cloud Completion and Graph Neural Network for 3D Object Detection”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 4. 2021, pp. 3430–3437.

[81] Zheng, Wu et al. “CIA-SSD: Confident IoU-Aware Single-Stage Object Detector From Point Cloud”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 4. 2021, pp. 3555–3562.

[82] Zheng, Wu et al. “SE-SSD: Self-Ensembling Single-Stage Object Detector From Point Cloud”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 14494–14503.

[83] Zheng, Zhaohui et al. “Distance-IoU loss: Faster and better learning for bounding box regression”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07. 2020, pp. 12993–13000.

[84] Zhou, J et al. “Siamesepointnet: A siamese point network architecture for learning 3d shape descriptor”. In: *Computer Graphics Forum*. Vol. 39. 1. Wiley Online Library. 2020, pp. 309–321.

[85] Zhou, Qian-Yi, Park, Jaesik, and Koltun, Vladlen. “Fast global registration”. In: *European conference on computer vision*. Springer. 2016, pp. 766–782.

[86] Zhou, Qian-Yi, Park, Jaesik, and Koltun, Vladlen. “Open3D: A Modern Library for 3D Data Processing”. In: *arXiv:1801.09847* (2018).

[87] Zhou, Xingcheng et al. “Real-Time LiDAR-Based 3D Object Detection on the Highway”. unpublished thesis. MA thesis. Technische Universität München, 2021.

[88] Zhou, Yin and Tuzel, Oncel. “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.

[89] Zimmer, Walter, Rangesh, Akshay, and Trivedi, Mohan. “3d bat: A semi-automatic, web-based 3d annotation toolbox for full-surround, multi-modal data streams”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 1816–1821.