Towards Efficient Point-Cloud Object Detection on Autonomous Vehicles

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Weijing Shi

B.S., Microelectronics, Fudan University
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

August 2022

To my parents Binghuan Shi and Wenping Lai
and my wife Lailei Feng

Acknowledgements

I am deeply grateful to all the individuals who have contributed to an exciting journey toward my Ph.D. This dissertation would not have been possible without their help and support.

First and foremost, I would like to thank my advisor, Prof. Ragunathan (Raj) Rajkumar. I am incredibly grateful for meeting and working with Prof. Rajkumar. Working closely under his guidance has made me a better thinker, engineer and researcher. I am thankful for the multiple opportunities he provided to work on diverse projects, ranging from designing computer vision and machine learning algorithms to building the CMU autonomous driving platform. Prof. Rajkumar also gave me multiple opportunities to work with and mentor several other students, which expanded my horizons and led me to become an independent thinker. Prof. Rajkumar's guidance and constructive criticism have also helped me improve my technical writing and presentation skills.

I am grateful to my thesis committee members, Prof. Vijayakumar (Kumar) Bhagavatula, Dr. Gary Overett and Dr. Weiyue Wang, for their time, effort, and input in completing this dissertation. I want to thank Prof. Bhagavatula for his insights and feedback on various aspects of my work. It has been very inspiring interacting with him. I would also like to express my gratitude to Dr. Overett. Collaborating with him has been a great pleasure and led me to computer vision studies. His passion and friendly nature are aspects that I would like to emulate. I would also like to thank Dr. Wang for her guidance during my internship experiences. Her strong work ethic and innovative problem-solving approaches have left a lasting impact.

A special thanks to General Motors (GM) for funding a part of my research. In partic-

ular, I wish to thank Eran Kishon in GM Technical Center Israel. Collaborating with him provided me with an opportunity to solve real-world challenges and significantly broaden my research perspectives.

An important part of my Ph.D. journey has been being a Real-Time and Multimedia Systems Lab (RTML) member. Thanks to all the wonderful people of RTML who shared their time with me: Anand Bhat, Sandeep D'souza, Shunsuke Aoki, Iljoo Baek, Mengwen He, Peter Jan, Swapnil Das, Shounak Sural and Gregory Su. Also, I would like to thank Chelsea Mendenhall, Toni Fox and Bridgette Bernagozzi for their kind support on administrative matters. I am also grateful to John Kozar for his creative mechanical engineering support.

I am privileged to have had a good group of friends in the course of my student life: Miao Yu, Tianlong Yu, Youzhi Bao, Chenlei Fang, Qicheng Huang, Zeye Liu, Rongye Shi, He Xi and Xiaoliang Li. Without these people, I could not have fully enjoyed my time at CMU.

Lastly, my deepest gratitude goes to my family. I would like to thank my parents, Binghuan Shi and Wenping Lai. They are always there for me, love and believe in me. I want to thank my wife, Lailei Feng. She has been a constant source of delight, care and love. My family has always been there to love and support me, and I dedicate this thesis to them.

Abstract

Autonomous vehicles (AVs) must perceive and understand the 3D environment around them. Modern autonomous vehicles use a suite of sensors and a set of machine-learning-based recognition algorithms to try to accomplish this goal. While such a system can achieve high accuracy, its cost is also prohibitively high for many applications. In this thesis, we aim to make the perception system of an AV both less expensive and more accurate. We design a multi-pronged approach that addresses challenges ranging from data representation, sensor configuration and training efficiency to the semantic understanding of road scenarios. We start with the data presentation of lidar point clouds, a primary input to many perception systems, and design a novel graph representation of point clouds to detect objects. Secondly, we study a sensor configuration of sparse lidars and complementary low-cost cameras. We propose a fusion method to combine video streams with sparse lidar point clouds to increase the point-cloud object detection accuracy while reducing the cost. Thirdly, we utilize self-supervised training strategies to use unlabeled data efficiently and create a set of geometric pretext tasks to pre-train the neural network. Finally, we study specific challenging real-world scenarios and implement context-specific solutions to problems raised by the presence of work zones.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Autonomous driving has enormous potential to improve transportation safety and efficiency. One vital prerequisite for autonomous driving is the ability to perceive and understand the 3D environment around the vehicle. Modern autonomous vehicles use a suite of sensors, such as cameras, lidars, radars, and sonars. Those sensors provide input to machine-learning-based recognition algorithms, which generate environmental structure information for other safety-critical tasks such as localization and path planning. Amount these recognition algorithms, 3D object detection is one of the essential tasks for the safety of autonomous driving.

The goal of 3D object detection is to localize and classify the instances of surrounding 3D objects. Many sensors on the autonomous vehicle can provide input to 3D object detection. RGB cameras have a relatively low cost but only provide 2D data capturing an

object's color and texture. Radars are robust to various weather conditions and provide 3D object positions with low spatial resolution. Sonars can also sense object distance using ultra-sound, but their detection range is small and spatial resolution is also low. Unlike these sensors, lidars measure the precise distances between the vehicle and environmental objects using laser beams, generating accurate point clouds. The point clouds provide direct 3D information and have a high spatial resolution. With recent advances in lidar technology, many autonomous vehicles (AVs) have adopted point-cloud-based perception systems.

Object detection has been studied extensively by the robotic and computer vision community for years. Thanks to the progress being made in deep neural networks, point-cloud 3D object detection algorithms have also been improving significantly. However, point-cloud object detection systems are still far from efficient in practice. In a typical algorithm pipeline, a neural network consumes many onboard computational and memory resources due to converting points to dense grids. The neural network also requires increasingly dense point clouds from expensive hardware and needs extensive annotated data for training. The overall point-cloud object detection system has a high cost, which poses challenges to the progress of autonomous driving. The high cost also prevents point-cloud object detection's applicability to many practical domains, including consumer vehicles, warehouses and work zone equipment.

The above challenges motivate us to research methods that make the point-cloud 3D object detection consume fewer resources and remain accurate. Hence, in this dissertation, we provide a suite of solutions to enable a cost-efficient and accurate perception system for autonomous vehicles. As shown in Figure 1.1, we design a multi-pronged approach that

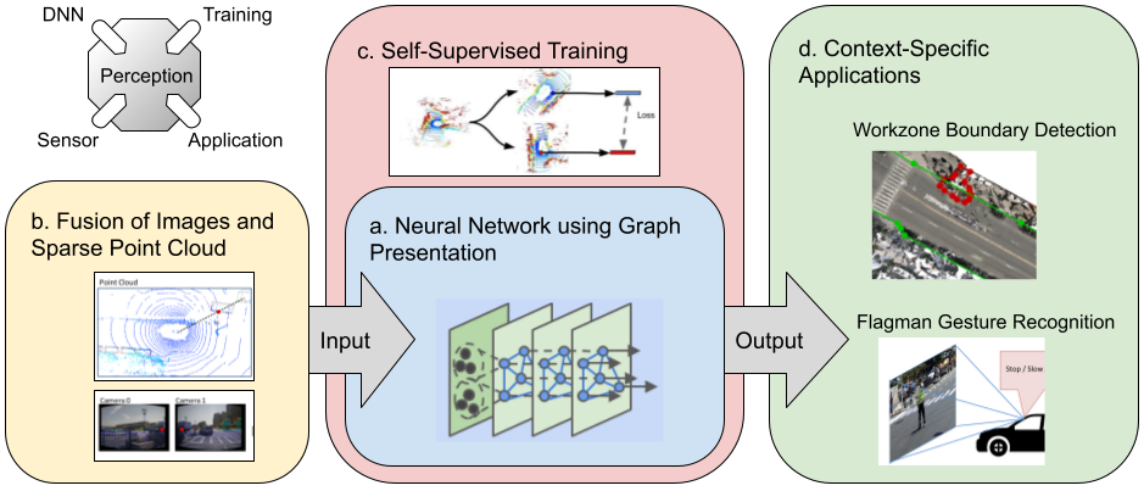

Figure 1.1: We propose a multi-pronged approach to achieve cost-efficient and accurate perception. (a). We first design a novel graph neural network to detect objects from point clouds effectively. (b). We then extend the sensory input and fuse images from cameras with sparse lidar point clouds to increase the point-cloud object detection accuracy. (c). Next, we utilize self-supervised training strategies to pre-train the neural network using unlabeled data. (d). Finally, we study specific challenging work zone scenarios and implement context-specific solutions.

addresses challenges ranging from data representation, sensor configuration and training efficiency to practical solutions to understand work zone scenarios.

Our thesis statement is as follows:

Autonomous vehicles can enhance perception accuracy and efficiency by utilizing a complementary suite of:

- *graph-based compact point-cloud representations,*
- *a sparse fusion of camera images and point clouds,*
- *geometric-concentrated self-supervised training, and*

- *context-specific designs for challenging but important driving scenarios such as work zones.*

1.2 Scope of the Thesis

We now describe the key problem spaces we address in this dissertation and briefly outline the direction of the solutions we propose.

1.2.1 Neural Network for Point Cloud Object Detection

The spatially-sparse nature of point clouds is challenging for designing effective algorithms for object recognition. There is a discrepancy between widely-used neural networks such as convolutional neural networks (CNNs) and point-cloud data representations. The convolution operation requires input data to be evenly spaced on a regular grid such as an image. However, the spatial distribution of 3D points from the lidar is irregular. This discrepancy requires a conversion from point clouds to grids as a preprocessing step [1, 2, 3, 4, 5, 6]. Such conversion from an irregular point cloud to a regular grid induces empty and crowded grid cells. If not appropriately addressed, they may demand high computational and memory resources. Many practical implementations need custom optimizations such as sparse convolution [7, 8] and multi-stage methods [9, 6] to save computational resources.

Alternatively, set-based neural networks [10, 11, 12] allow an unordered set of points as the input. A few studies use this neural network to extract features from a set of points without mapping the point cloud to a grid. However, in order to establish hierarchical

feature aggregation, they need to sample and group points into sets iteratively. Hence, the repeated grouping and sampling on a large point cloud becomes computationally costly. Hybrid approaches [4, 13, 5] often use a grid and a set representation in different stages for a balance of convolution operations and DeepSet [11] operations. Given a good implementation, they have shown both promising speed and accuracy. However, hybrid strategies may still suffer the limitations of both representations. Meanwhile, a novel network architecture that adapts to the point cloud structure natively without conversion can potentially avoid these shortcomings.

Graph neural network (GNN) [14] provides the opportunities to design such a novel network architecture. A GNN operates on a graph by learning the information exchange functions between vertices. If the points are treated as vertices, they need not be converted to another structure. In other words, a graph can represent a point cloud natively. Approaches [15, 16, 17] that use GNN for the semantic segmentation and classification of small-scale point clouds have shown good accuracy. However, few studies have investigated GNN object detection, especially for large outdoor point clouds.

We propose a GNN architecture that detects 3D objects from point clouds. By using a novel graph representation, we encode the irregular point cloud directly without mapping to dense grids. The graph is also constructed once and avoids the repeated conversion to sets. We name our method, Point-GNN. Our approach achieves good accuracy in the KITTI [18] benchmark, demonstrating the effectiveness of using GNN as an alternative point cloud backbone to conventional methods. This backbone can surpass many fusion-based algorithms and remains relatively robust when the density of a point cloud decreases.

1.2.2 Sensor Fusion for Point Cloud Object Detection

Given an effective point cloud neural network, it may still be difficult to infer all object geometries from lidar points alone. The sparse lidar points may not provide enough discriminative information about the object due to occlusion and limited point density. The occlusion arises when the laser path from the lidar to an object's surface is blocked. Furthermore, the density of lidar points is often inversely proportional to the distance between the vehicle and the object. To overcome these limitations and increase detection ranges, many lidar-based 3D detection systems use high-definition lidars and temporal point cloud aggregation [19, 20]. Despite recent advances, a high-definition lidar system is still very expensive in many application domains, and lidar aggregation suffers from misalignment due to vehicle motion.

It is desirable to explore alternative approaches to increase object detection accuracy under the constraint of point cloud density. Recently studies [21, 22] have shown that cameras, which have a relatively low cost, can provide vital semantic information to the point cloud and increase the detection accuracy significantly. Such fusion approaches have demonstrated a cost-efficient way to improve object detection without pursuing an increasingly high-density and expensive point cloud. However, those methods still heavily rely on dense point clouds. They use the image features as decorators to the point clouds and infer the object's 3D shape in the same way as in point cloud object detection.

Instead, we re-design the object detection method by combining multi-view image features with Point-GNN features to infer an object's local 3D structure. We calculate the object's position by matching a point's lidar coordinates and predicted local coordinates. Our approach utilizes the structure information in the image and reduces the reliance on

a point cloud structure. The minimum requirement for detection is only a single lidar point on the object and the target object's image. Our experiments with the nuScenes [20] dataset and Lyft Level5 [23] dataset show a considerable improvement in detection accuracy on sparse point clouds.

1.2.3 Reducing Labels for Point Cloud Object Detection

Besides the challenges of point cloud's data representation and sparsity, data labels are another constraining factor for real-world deployment. Deep neural networks learn to detect objects from a large amount of labeled data. However, such labeled data require extensive manual annotation, which is costly and time-consuming. In many scenarios, the labeled data are not available or are prohibitively expensive to gather. Unlabeled driving data, however, are relatively abundant and inexpensive to collect. Many studies have been carried out on utilizing unlabeled data in the image domain [24, 25, 26] and natural language processing domain [27]. However, how to apply self-supervised learning for point-cloud object detection remains an open question.

PointInfoNCE [28] adapts the *InfoNCE* loss in the contrastive learning for point cloud and shows that pretraining using such loss can help the indoor point cloud object detection. However, such loss supervises the neural network to learn invariant features for point cloud transformations such as rotation. It is unclear whether such features benefit large outdoor point clouds of autonomous driving.

To reduce the labels needed for training in the autonomous driving domain, we build upon the prevalent contrastive learning method and propose a set of geometric pretext tasks to improve the pretraining performance. Our experiments reveal that contrastive

loss alone improves the average precision (AP) but negatively impacts the object heading accuracy. Our geometric pretext tasks benefit both the average precision and heading accuracy. Therefore, our approach provides a feasible direction to reduce the expensive labeling of training data.

1.2.4 Practical Solutions for Real-World Problems

In practice, achieving the goal of full autonomy is also impeded by the need to address several operational challenges. Complex scenarios such as work zones are among such challenges. An autonomous vehicle needs to make safe decisions and facilitate forward progress in the presence of road construction and flagmen, demanding more detailed 3D object recognition such as work zone boundary detection and flagman gesture recognition. In such challenges, pursuing a high-accuracy object detection may not be the most cost-efficient method. Instead, these challenging scenarios require practical solutions.

In the scope of autonomous driving, few studies have looked into the task of work zone detection. Most work zone datasets lack multi-modality sensor configurations, which is common in a modern autonomous vehicle. Also, the work zone detection problem does not have appropriate definitions and evaluation methods. There has also been little discussion on the standard output format of work zone detection algorithms. Therefore, we formulate the work zone detection problem and define a standard output format suitable for a multi-modality sensor configuration. We provide work zone annotations and also baseline implementations for work zone detection.

In a work zone, a flagman may provide temporary traffic control. It's critical that the autonomous vehicle understands and follows the correct traffic gestures. As these tempo-

rary control may not be updated in a database, the autonomous vehicle needs the ability to understand the traffic gestures on-the-fly. Point-cloud object detection can recognize and locate pedestrians. However, the subtle movement of pedestrian gestures is difficult for sparse point clouds to capture. A lidar system capable of a high-resolution human body scan from a distance is still prohibitively expensive. Instead of relying completely on point clouds, the gesture recognition system can use RGB videos for fine-grained recognition after detecting a pedestrian. Many research on video action recognition uses deep neural networks [29, 30, 31] to learn from training data. Nevertheless, there are few studies on traffic gesture recognition due to a lack of traffic gesture datasets. Therefore, we build a video dataset covering common traffic gestures, including the usage of props. We then develop an efficient deep network to use a combined data representation of hand images, pose keypoints, and bounding boxes. Our experiments show that our approach is robust against the variance of flagmen's appearance.

Our algorithms for work zone challenges are low-cost yet accurate solutions to the overall perception system. It also serves as a real-world validation case for our point-cloud object detection framework.

1.3 Organization of The Dissertation

This thesis is structured as follows.

- In Chapter 2, we review the background and relevant prior work.
- In Chapter 3, we start by seeking a more compact data representation of point clouds and a powerful neural network architecture to extract features from them. We encode

the points in a novel graph representation *without* converting them to a grid format and then design a graph neural network for 3D object detection.

- In Chapter 4, we leverage the co-existence of low-cost cameras and lidars to enrich the information in the point clouds. We estimate local object shapes from images and then register them to the lidar points, with the registration requiring only a few points on the object.

- In Chapter 5, we lower the overall system expense by utilizing the relatively abundant and low-cost unlabeled data for training. We adopt self-supervised learning and propose to combine geometric pretext tasks and contrastive loss for pretraining.

- In Chapter 6 and 7, we validate our approach on a real AV and address the practical needs in complex real-world driving scenarios such as work zones. Specifically, we propose efficient methods to recognize work zone boundaries and flagmen gestures.

- Finally, we discuss the remaining questions and future work in Chapter 8.

Chapter 2

Related Work

This chapter presents the background and related work on the following topics: (a) different point cloud data presentations in deep neural networks, (b) multi-modality data for object detection, (c) Self-supervised learning for point clouds, (d) Work zone recognition and (e) Flagman gesture recognition.

2.1 Point Cloud Data Representations

Prior work on point cloud data representations can be grouped into three categories, as shown in Figure 2.1.

2.1.1 Point Cloud in Grids

Many recent studies convert a point cloud to a regular grid to utilize convolutional neural networks. [2] projects a point cloud to a 2D Bird's Eye View (BEV) image and uses

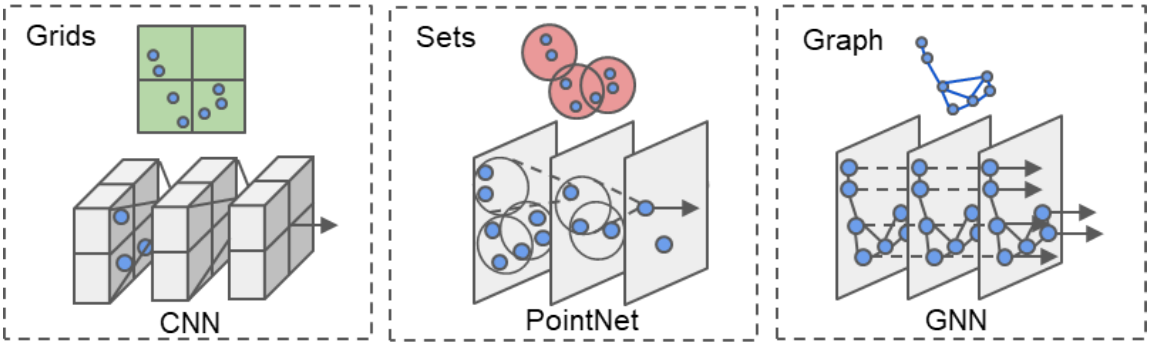

Figure 2.1: Three point cloud representations and their common processing methods.

a 2D CNN for object detection. [1] projects a point cloud to both a BEV image and a Front View (FV) image before applying a 2D CNN on both. Such projection induces a quantization error due to the limited image resolution. Some approaches keep a point cloud in 3D coordinates. [3] represents points in 3D voxels and applies 3D convolution for object detection. When the resolution of the voxels grows, the computation cost of 3D CNN grows cubically, but many voxels are empty due to point sparsity. Optimizations such as the sparse convolution [8] reduce the computation cost. Converting a point cloud to a 2D/3D grid suffers from the mismatch between the irregular distribution of points and the regular structure of the grids.

2.1.2 Point Cloud in Sets

Deep learning techniques on sets such as PointNet [10] and DeepSet [11] show neural networks can extract features from an unordered set of points directly. In such a method, each point is processed by a multi-layer perceptron (MLP) to obtain a point feature vector. Those features are aggregated by an average or max pooling function to form a global fea-

ture vector of the whole set. PointNet++ [12] further proposes the hierarchical aggregation of point features and generates local subsets of points by sampling around some keypoints. The features of those subsets are then again grouped into sets for further feature extraction. Many 3D object detection approaches take advantage of such neural networks to process a point cloud without mapping it to a grid. However, the sampling and grouping of points on a large scale lead to additional computational costs. Most object detection studies only use the neural network on sets as a part of the pipeline. Frustum-PointNet [9] generates object proposals from camera images and uses PointNet++ [12] to separate points that belong to an object from the background and predict a bounding box. PointRCNN [5] uses PointNet++ [12] as a backbone network to generate bounding box proposals directly from a point cloud. Then, it uses a second-stage point network to refine the bounding boxes. Hybrid approaches such as [3, 8, 4, 13] use PointNet [10] to extract features from local point sets and place the features on a regular grid for the convolutional operation. Although they reduce the local irregularity of the point cloud to some degree, they still suffer the mismatch between a regular grid and the overall point cloud structure.

2.1.3 Point Cloud in Graphs

Research on graph neural network [14] seeks to generalize the convolutional neural network to a graph representation. A GNN iteratively updates its vertex features by aggregating features along the edges. Although the aggregation scheme sometimes is similar to that in deep learning on sets, a GNN allows more complex features to be determined along the edges. It typically does not need to sample and group vertices repeatedly. In the computer vision domain, a few approaches represent the point cloud as a graph. [15]

uses a recurrent GNN for the semantic segmentation of RGBD data. [32] partitions a point cloud to simple geometrical shapes and links them into a graph for semantic segmentation. [16, 17] look into classifying a point cloud using a GNN. So far, few investigations have looked into designing a graph neural network for object detection, where an explicit prediction of the object shape is required.

2.2 Sensor Modalities

This section briefly reviews lidar-based, camera-based and fusion-based 3D object detection algorithms.

2.2.1 Lidar-based 3D Object Detection

Many state-of-the-art studies use a lidar point cloud as their primary input. They generally use a neural network as an encoder to extract the structure features from the point cloud and use those features to recognize the underlying object category and shape. Some popular choices of encoders are convolutional neural networks [2, 4], PointNet [10], and graph neural networks [33]. The CNN-based detection algorithms usually convert the point cloud to an image grid or a 3D voxel grid. PIXOR [2] projects the points to multiple Bird's Eye View (BEV) images at different heights and uses a convolutional neural network to extract features in different image positions. The PointPillars [4] approach discretizes the points into 2D evenly spaced grids in BEV and encodes each cell's points. VoxelNet [3] puts points into voxel cells and uses a 3D convolutional neural network to extract voxel features. SECOND [8] uses sparse convolution for further improvement. PointNet [10]

extracts features from an unordered point set. Many detection methods [9, 12] first group the lidar points locally in different spatial areas and then extract the point set's feature by PointNet [10]. PointGNN [33] and CasGNN [34] use graph neural networks to encode the point cloud. They create a neighborhood graph by using points as vertices. The graph neural network then generates features representing each vertex's property.

Regardless of the encoder, lidar-based 3D object detections make the basic assumption that the point cloud is dense enough to extract information about the underlying object shape. However, the sparse lidar points may not provide enough discriminative information about the object due to occlusion and limited point density. The density of lidar points is often inversely proportional to the distance between the vehicle and the object. To overcome these limitations and increase detection ranges, HD lidar systems are often used [19, 18] or temporal aggregation is used to accumulate points from multiple lidar sweeps. Despite recent advances, a high-definition lidar system is still costly in many application domains. Meanwhile, lidar aggregation suffers from misalignment due to vehicle motion and also induces dependencies on other tasks such as vehicle localization. When the density of the point cloud decreases, the detection accuracy unfortunately suffers.

2.2.2 Monocular 3D Object Detection

As a 2D image lacks depth information, monocular 3D object detection faces the challenge of solving an under-constrained inverse problem from 2D to 3D. Many methods [35, 36, 37, 38] leverage a neural network to learn the priors of object shape from data. Mono3D [35] uses a neural network to regress the object's 3D bounding boxes using a candidate image region as input. Deep3DBox [36] also predicts an object's dimension

and orientation from the image using a convolutional neural network. It then finds the object's distance by matching the 3D box's image projection with the 2D object bounding box. M3D-RPN [37] predicts the object position relative to pre-defined anchor boxes and then refines the orientation by matching the 3D box projection to the 2D box. MonoPSR [38] further predicts each object pixel's coordinates from the image and refines the 3D bounding boxes by minimizing those pixels' re-projection errors. Recent work [39] and [40] use the depth-map input to improve the performance of monocular 3D object detection. However, additional labels and training are necessary for a depth estimation network. Monocular 3D object detection, in general, is still not as accurate as lidar-based methods. Nevertheless, they show that we can infer the object category and shapes from the image using the priors in training data.

2.2.3 Lidar-camera Fusion for 3D Object Detection

Lidars and cameras often co-exist on an autonomous vehicle. Many studies combine both sensors' information to achieve a higher object detection accuracy. However, the existing methods often partially or completely rely on a dense structure of point clouds. MV3D [1] first generates 3D object proposals from a bird's eye view map, which is the projected image of a dense point cloud. Frustum PointNet [9] uses an image to generate frustum proposals but relies solely on point cloud features for segmentation and box regression. PointPainting [21] and ImVoteNet [22] associate image segmentation and 2D object detection results with lidar points and then perform point-cloud-based 3D object detection. PseudoLidar+ [41] uses the point cloud to lift the image to 3D. However, it needs an external image depth map estimator.

2.3 Self-supervised Learning

2.3.1 Image Domain

In self-supervised learning, a neural network is pretrained by pretext tasks. Pretext tasks create their own supervision without the need for manual annotation. In the image domain, some common pretext tasks are image reconstruction, geometric transformation recognition and contrastive learning. In image reconstruction tasks, a part of the original image's information, such as color [42, 43], image patch [44] or order [45, 46], is removed and the neural network is trained to reconstruct the missing information. In geometric transformation recognition tasks, the input image is randomly transformed, (e.g. rotated [47]). The neural network is then trained to predict the transformation the input image has gone through. Contrastive learning supervises the neural network to generate similar features when the input image is augmented. At the same time, it also supervises the neural network to generate different features for different input images. Contrastive learning has attracted a lot of research interest because of its simplicity and effectiveness. Recent models such as MoCo [24] and SimCLR[25] have shown that self-supervised pretraining can be competitive with supervised pretraining in the image classification task.

The performance of the pretraining is related to the alignment between the pretext task and the fine-tuning task. A mismatch may reduce the gain of the fine-tuning task from pretraining or even negatively impact the fine-tuning task. Recent studies on image object detection [48, 49] suggest that self-supervised learning designed for image classification may not always improve the downstream object detection task. The discriminative data representation learned for image classification may not work well for localization in object

detection. Therefore, [49] further provide foreground saliency detection as supervision.

2.3.2 Point Cloud Domain

Following the approach in the image domain, self-supervised studies in point cloud design similar pretext tasks. Similar to solving a Jigsaw [45] in images, [50] converts a point cloud to a voxel grid and then permutes the voxel order for the neural network to recover. [51] designs an auto-encoder-decoder task where a parametric GMM generative model replaces the decoder to reconstruct a point cloud. [52] follows the rotation classification task [47] for images and randomly rotates the point cloud for the neural network to recognize. [28] adapts the *InfoNCE* loss in the contrastive learning for point cloud and propose *PointInfoNCE* loss. [28] shows pretraining using *PointInfoNCE* loss can help the downstream indoor point cloud object detection. However, *PointInfoNCE* loss supervises the neural network to learn discriminative features invariant to point cloud transformations such as rotation. The learned features are, therefore, prone to loss of geometric information critical for accurate object bounding box estimation in autonomous driving.

2.4 Work Zone Recognition

Due to road maintenance and repair, a work zone can appear dynamically and may not be included in the map database. For driving safety, an autonomous vehicle must detect the work zone on the fly. However, work zone detection is relatively unexplored due to a lack of standard definitions and data. Existing work zone detection studies focus on detecting the presence or absence of a work zone. Their detection results are relatively

coarse-grained and do not contain the boundary information of the work zone. [53] detects work zones by classifying work zone images using a convolutional neural network. [54] detects traffic signs from the vehicle's camera. It then recognizes work zone signs and uses them as cues for the start and the end of a work zone road segment. [55] also detects traffic signs and traffic cones. It uses the detection results as features to estimate the probability that the vehicle is in a work zone. [56] designs a Bayesian network for construction site detection. The Bayesian network takes detected traffic objects and vehicle states as input and predicts a construction site's probability at different distance bins. None of these studies [53, 54, 55, 56] infers fine-grained geometric attributes such as work zone areas and boundaries. However, knowing the work zone geometry is critical for an autonomous vehicle to navigate safely. [57] studies a more constrained case where temporary lane markings guide the driving through a work zone. [57] tracks the lanes even if both temporary lane markings and original lane markings exist. Therefore, it allows the vehicle to follow the lanes in a work zone. However, such an approach relies on lane marking legislation and cannot handle common work zones with no temporary lane markings.

2.5 Traffic Gesture Recognition

2.5.1 Traffic Gesture Dataset

Although there are many guidelines [58] on common traffic gestures, no strict rules dictate the exact motion that a flagman needs to follow. In practice, a flagman's action varies according to one's body condition and personal style. Moreover, the flagman's gestures also change according to the cultural background in vogue. It is important to capture these

variations in a dataset so that data-driven methods can learn to recognize these varied and complex gestures. Such a dataset can also act as a common benchmark where people can compare different algorithms fairly. However, there are very few existing traffic gesture datasets. Hand gesture datasets such as NVGesture[59] and SHREC'17[60] focus on close-range hand motion outside the context of traffic control, while generic human action datasets such as Deepmind Kinetics [61] and UCF101 [62] do not contain traffic gestures at all. Recently, [63] provides a traffic control gesture dataset where actors wear IMU-based motion capture suits to generate 3D body poses. The poses utilize only the hands without considering the use of common props. As the capture suite is generally not available in practice, it creates a discrepancy between training in the dataset and real-world encounters. We create a dedicated dataset with video sequences of various traffic gestures, facilitating a more realistic evaluation for vision-based traffic gesture recognition.

2.5.2 Action Recognition

Traffic gesture recognition is a sub-domain of action recognition. Many studies use deep learning methods to recognize human actions in videos. Typically, they use neural networks to extract a feature map [29, 30] or a human skeleton representation [31] from each video frame and then process the features via a temporal model to capture the action. For traffic gesture recognition, previous studies often used a skeleton representation. The work in [64] detects up-body keypoints and uses hand-crafted rules for classification. Instead, the study in [63] extracts a skeleton representation using a deep network and then uses a recurrent neural network and a graph neural network for temporal modeling. However, a person's skeleton representation often ignores important subtle information such as finger

movements and does not handle signaling with props. A feature map representation of the input image, on the other hand, contains rich details, but it also includes irrelevant information prone to overfitting. There is little study on accurate traffic gesture recognition using other effective data representations.

Chapter 3

Point-Cloud Object Detection

3.1 Introduction

A point cloud that composes a set of points in space is a widely-used format for 3D sensors such as lidars in robotic perception. Detecting objects accurately from a point cloud is crucial in applications such as autonomous driving. In this chapter, we describe our approach of using a novel graph as a compact representation of a point cloud and design a graph neural network named Point-GNN to detect 3D objects.

Convolutional neural networks that detect objects from images rely on the convolution operation. While the convolution operation is efficient, it requires a regular grid as input. Unlike an image, a point cloud is typically sparse and not spaced evenly on a regular grid. Placing a point cloud on a regular grid generates an uneven number of points in the grid cells. Applying the same convolution operation on such a grid leads to potential information loss in the crowded cells or wasted computation in the empty cells.

Recent breakthroughs in using neural networks [10, 11] allow an unordered set of points as input. Studies take advantage of this type of neural network to extract point cloud features without mapping the point cloud to a grid. However, they typically need to sample and group points iteratively to create a *point set* representation. The repeated grouping and sampling on a large point cloud can be computationally costly. Recent 3D detection approaches [4, 13, 5] often take a hybrid approach to use a grid and a set representation in different stages. Although they show some promising results, such hybrid strategies may suffer the shortcomings of both representations.

Our work differs from previous work by designing a Graph Neural Network (GNN) for object detection. Instead of converting a point cloud to a regular grid, such as an image or a voxel, we use a novel graph representation to preserve the irregularity of a point cloud. We encode the point cloud natively in a graph by using the points as the graph vertices. The edges of the graph connect neighborhood points within a fixed radius, allowing feature information to flow between neighbors. Such a graph representation adapts to the structure of a point cloud directly without the need to make it regular. Unlike the techniques that sample and group the points into sets repeatedly, we construct the graph *once*. A graph neural network reuses the graph edges in every layer.

We name our graph neural network for object detection as Point-GNN. It is a single-stage detection method that uses a point graph as input. Studies [15, 32, 16, 17] have looked into using graph neural networks for the classification and the semantic segmentation of a point cloud. However, little research has looked into using a graph neural network for 3D object detection in a point cloud. Our work demonstrates the feasibility of using a GNN for highly accurate object detection in a point cloud.

Point-GNN takes the point graph as its input. It extracts features of the point cloud by iteratively updating vertex features on the same graph. Point-GNN then outputs the category and bounding boxes of the objects to which each vertex belongs. Point-GNN is a one-stage detection method that detects multiple objects in a single shot. To reduce the translation variance in a graph neural network, we introduce an auto-registration mechanism which allows points to align their coordinates based on their features. We further design a box merging and scoring operation to combine detection results from multiple vertices accurately.

We evaluate Point-GNN on the KITTI benchmark. On the KITTI benchmark, Point-GNN achieves competitive accuracy using the point cloud alone and even surpasses sensor fusion approaches. Our Point-GNN shows the potential of a new type of 3D object detection approach using graph neural networks, and it can serve as a good baseline for future research. We conduct an extensive ablation study on the effectiveness of the components in Point-GNN.

In summary, our contributions are:

- We design a new point-cloud object detection approach using a graph neural network.
- We propose Point-GNN, a graph neural network with an auto-registration mechanism that detects multiple objects in a single shot.
- We achieve competitive 3D object detection accuracy in the KITTI benchmark and analyze the effectiveness of each component in depth.

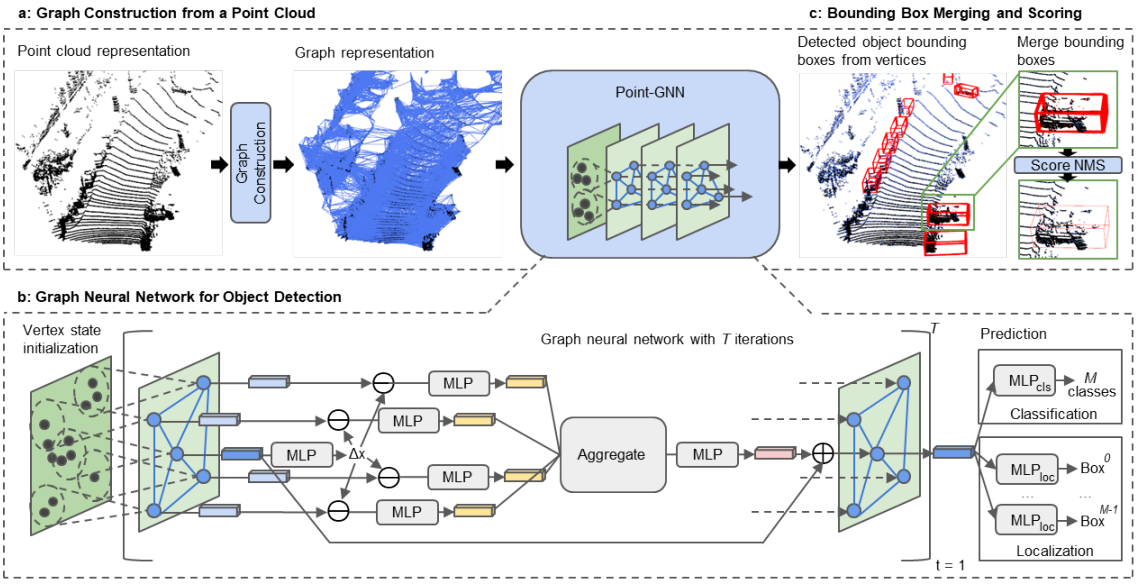

Figure 3.1: The architecture of Point-GNN. It has three main components: (a) graph construction from a point cloud, (b) a graph neural network for object detection, and (c) bounding box merging and scoring.

3.2 Point-GNN Object Detection

In this section, we describe our approach to detect 3D objects from a point cloud. As shown in Figure 3.1, the overall architecture of our method contains three components: (a) graph construction, (b) a GNN of T iterations, and (c) bounding box merging and scoring.

3.2.1 Graph Construction

Formally, we define a point cloud of N points as a set $P = \{p_1, ..., p_N\}$, where $p_i = (x_i, s_i)$ is a point with both 3D coordinates $x_i \in \mathbb{R}^3$ and the state value $s_i \in \mathbb{R}^k$ a k-length vector that represents the point property. The state value s_i can be the reflected laser intensity or the features which encode the surrounding objects. Given a point cloud P, we construct a

graph $G = (P, E)$ by using P as the vertices and connecting a point to its neighbors within a fixed radius r, i.e.

$$E = \{(p_i, p_j) \mid \|x_i - x_j\|_2 < r\} \tag{3.1}$$

The construction of such a graph is the well-known fixed radius near-neighbors search problem. By using a cell list to find point pairs that are within a given cut-off distance, we can efficiently solve the problem with a runtime complexity of $O(cN)$ where c is the max number of neighbors within the radius [65].

In practice, a point cloud commonly comprises tens of thousands of points. Constructing a graph with all the points as vertices imposes a substantial computational burden. Therefore, we use a voxel downsampled point cloud $\hat{P}$ for the graph construction. It must be noted that the voxels here are only used to reduce the density of a point cloud and they are not used as the representation of the point cloud. We still use a graph to present the downsampled point cloud. To preserve the information within the original point cloud, we encode the dense point cloud in the initial state value s_i of the vertex. More specifically, we search the raw points within a r_0 radius of each vertex and use the neural network on sets to extract their features. We follow [4, 3] and embed the lidar reflection intensity and the relative coordinates using an *MLP* and then aggregate them by the *Max* function. We use the resulting features as the initial state value of the vertex. After the graph construction, we process the graph with a GNN, as shown in Figure 3.1b.

3.2.2 Graph Neural Network with Auto-Registration

A typical graph neural network refines the vertex features by aggregating features along the edges. In the $(t+1)^{th}$ iteration, it updates each vertex feature in the form:

$$\begin{aligned} v_i^{t+1} &= g^t(\rho(\{e_{ij}^t \mid (i,j) \in E\}), v_i^t) \\ e_{ij}^t &= f^t(v_i^t, v_j^t) \end{aligned} \tag{3.2}$$

where e^t and v^t are the edge and vertex features from the t^{th} iteration. A function $f^t(.)$ computes the edge feature between two vertices. $\rho(.)$ is a set function which aggregates the edge features for each vertex. $g^t(.)$ takes the aggregated edge features to update the vertex features. The graph neural network then outputs the vertex features or repeats the process in the next iteration.

In the case of object detection, we design the GNN to refine a vertex's state to include information about the object where the vertex belongs. Towards this goal, we re-write Equation 3.2 to refine a vertex's state using its neighbors' states:

$$s_i^{t+1} = g^t(\rho(\{f^t(x_j - x_i, s_j^t) \mid (i,j) \in E\}), s_i^t) \tag{3.3}$$

Note that we use the relative coordinates of the neighbors as input to $f^t(.)$ for the edge feature extraction. The relative coordinates induce translation invariance against the global shift of the point cloud. However, it is still sensitive to translation within the neighborhood area. When a small translation is added to a vertex, the local structure of its neighbors remains similar. But the relative coordinates of the neighbors are all changed, which increases the input variance to $f^t(.)$. To reduce the translation variance, we propose aligning

neighbors' coordinates by their structural features instead of the center vertex coordinates. Because the center vertex already contains some structural features from the previous iteration, we can use it to predict an alignment offset, and propose an *auto-registration* mechanism:

$$\begin{aligned} \Delta x_i^t &= h^t(s_i^t) \\ s_i^{t+1} &= g^t(\rho(\{f(x_j - x_i + \Delta x_i^t, s_j^t)\}, s_i^t) \end{aligned} \tag{3.4}$$

Δx_i^t is the coordination offset for the vertices to register their coordinates. $h^t(.)$ calculates the offset using the center vertex state value from the previous iteration. By setting $h^t(.)$ to output zero, the GNN can disable the offset if necessary. In that case, the GNN returns to Equation 3.3. We analyze the effectiveness of this auto-registration mechanism in Section 3.3.

As shown in Figure 3.1b, we model $f^t(.)$, $g^t(.)$ and $h^t(.)$ using multi-layer perceptrons (*MLP*) and add a residual connection in $g^t(.)$. We choose $\rho(.)$ to be Max for its robustness[10]. A single iteration in the proposed graph network is then given by:

$$\begin{aligned} \Delta x_i^t &= MLP_h^t(s_i^t) \\ e_{ij}^t &= MLP_f^t([x_j - x_i + \Delta x_i^t, s_j^t]) \\ s_i^{t+1} &= MLP_g^t(Max(\{e_{ij} \mid (i,j) \in E\})) + s_i^t \end{aligned} \tag{3.5}$$

where $[,]$ represents the concatenation operation.

Every iteration t uses a different set of MLP^t, which is not shared among iterations. After T iterations of the graph neural network, we use the vertex state value to predict both

the category and the bounding box of the object where the vertex belongs. A classification branch MLP_{cls} computes a multi-class probability. Finally, a localization branch MLP_{loc} computes a bounding box for each class.

3.2.3 Loss

For the object category, the classification branch computes a multi-class probability distribution $\{p_{c_1}, ..., p_{c_M}\}$ for each vertex. M is the total number of object classes, including the *Background* class. If a vertex is within a bounding box of an object, we assign the object class to the vertex. If a vertex is outside any bounding boxes, we assign the background class to it. We use the average cross-entropy loss as the classification loss.

$$l_{cls} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{c_j}^i log(p_{c_j}^i) \tag{3.6}$$

where p_c^i and y_c^i are the predicted probability and the one-hot class label for the i-th vertex, respectively.

For the object bounding box, we predict it in the 7 degree-of-freedom format $b = (x, y, z, l, h, w, \theta)$, where (x, y, z) represent the center position of the bounding box, (l, h, w) represent the box length, height and width respectively, and θ is the yaw angle. We encode the bounding box with the vertex coordinates (x_v, y_v, z_v) as follows:

$$\begin{aligned}
&\delta_x = \frac{x - x_v}{l_m},\ \delta_y = \frac{y - y_v}{h_m},\ \delta_z = \frac{z - z_v}{w_m} \\
&\delta_l = log(\frac{l}{l_m}),\ \delta_h = log(\frac{h}{h_m}),\ \delta_w = log(\frac{w}{w_m}) \\
&\delta_\theta = \frac{\theta - \theta_0}{\theta_m}
\end{aligned} \tag{3.7}$$

where $l_m, h_m, w_m, \theta_0, \theta_m$ are constant scale factors.

The localization branch predicts the encoded bounding box $\delta_b = (\delta_x, \delta_y, \delta_z, \delta_l, \delta_h, \delta_w, \delta_\theta)$ for each class. If a vertex is within a bounding box, we compute the Huber loss [66] between the ground truth and our prediction. If a vertex is outside any bounding boxes or it belongs to a class that we do not need to localize, we set its localization loss as zero. We then average the localization loss of all the vertices:

$$l_{loc} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(v_i \in b_{interest}) \sum_{\delta \in \delta_{b_i}} l_{huber}(\delta - \delta^{gt}) \tag{3.8}$$

To prevent over-fitting, we add $L1$ regularization to each MLP. The total loss is then:

$$l_{total} = \alpha l_{cls} + \beta l_{loc} + \gamma l_{reg} \tag{3.9}$$

where α, β and γ are constant weights to balance each loss.

3.2.4 Box Merging and Scoring

As multiple vertices can be on the same object, the neural network can output multiple bounding boxes of the same object. It is necessary to merge these bounding boxes into one and also assign a confidence score. Non-maximum suppression (NMS) has been widely used for this purpose. The common practice is to select the box with the highest classification score and suppress the other overlapping boxes. However, the classification score does not always reflect the localization quality. Notably, a partially occluded object can have a strong clue indicating the type of the object but lacks enough shape information. The standard NMS can pick an inaccurate bounding box base on the classification score

Algorithm 1: NMS with Box Merging and Scoring

Input: $\mathscr{B} = \{b_1, ..., b_n\}$, $\mathscr{D} = \{d_1, ..., d_n\}$, T_h
$\mathscr{B}$ is the set of detected bounding boxes.
$\mathscr{D}$ is the corresponding set of detection scores.
T_h is an overlapping threshold value.
Green color marks the main modifications.

$\mathscr{M} \leftarrow \{\}$, $\mathscr{Z} \leftarrow \{\}$
while $\mathscr{B} \neq empty$ **do**
  $i \leftarrow argmax\ D$
  $\mathscr{L} \leftarrow \{\}$
  for b_j *in* $\mathscr{B}$ **do**
    if $iou(b_i, b_j) > T_h$ **then**
      $\mathscr{L} \leftarrow \mathscr{L} \cup b_j$
      $\mathscr{B} \leftarrow \mathscr{B} - b_j$, $\mathscr{D} \leftarrow \mathscr{D} - d_j$
    end
  end
  $m \leftarrow median(\mathscr{L})$
  $o \leftarrow occlusion(m)$
  $z \leftarrow (o+1)\sum_{b_k \in \mathscr{L}} IoU(m, b_k) d_k$
  $\mathscr{M} \leftarrow \mathscr{M} \cup m$, $\mathscr{Z} \leftarrow \mathscr{Z} \cup z$
end
return $\mathscr{M}$, $\mathscr{Z}$

alone.

To improve localization accuracy, we propose to calculate the merged box by considering the entire overlapped box cluster. More specifically, we consider the median position and size of the overlapped bounding boxes. We also compute the confidence score as the sum of the classification scores weighted by the Intersection-of-Union (IoU) factor and an occlusion factor. The occlusion factor represents the occupied volume ratio. Given a box b_i, let l_i, w_i, h_i be its length, width and height, and let v_i^l, v_i^w, v_i^h be the unit vectors that indicate their directions respectively. x_j are the coordinates of point p_j. The occlusion

factor o_i is then:

$$o_i = \frac{1}{l_i w_i h_i} \prod_{v \in \{v_i^l, v_i^w, v_i^h\}} \max_{p_j \in b_i} (v^T x_j) - \min_{p_j \in b_i} (v^T x_j) \tag{3.10}$$

We modify standard NMS as shown in Algorithm 1. It returns the merged bounding boxes $\mathscr{M}$ and their confidence score $\mathscr{L}$. We will study its effectiveness in Section 3.3.

3.3 Experiments

3.3.1 Dataset

We evaluate our design using the widely used KITTI object detection benchmark [67]. The KITTI dataset contains 7481 training samples and 7518 testing samples. Each sample provides both the point cloud and the camera image. We only use the point cloud in our approach. Since the dataset only annotates objects that are visible within the image, we process the point cloud only within the field of view of the image. The KITTI benchmark evaluates the average precision (AP) of three types of objects: Car, Pedestrian and Cyclist. Due to the scale difference, we follow the common practice [4, 3, 8, 13] and train one network for the Car and another network for the Pedestrian and Cyclist. For training, we remove samples that do not contain objects of interest.

3.3.2 Implementation Details

We use three iterations ($T = 3$) in our graph neural network. During training, we limit the maximum number of input edges per vertex to 256. During inference, we use all the input edges. All GNN layers perform auto-registration using a two-layer MLP_h of units $(64, 3)$.

The MLP_{cls} is of size $(64, \#(classes))$. For each class, MLP_{loc} is of size $(64, 64, 7)$.

Car: We set (l_m, h_m, w_m) to the median size of Car bounding boxes $(3.88m, 1.5m, 1.63m)$. We treat a side-view car with $\theta \in [-\pi/4, \pi/4]$ and a front-view car $\theta \in [\pi/4, 3\pi/4]$ as two different classes. Therefore, we set $\theta_0 = 0$ and $\theta_0 = \pi/2$ respectively. The scale θ_m is set as $\pi/2$. Together with the *Background* class and *DoNotCare* class, 4 classes are predicted. We construct the graph with $r = 4m$ and $r_0 = 1m$. We set $\hat{P}$ as a downsampled point cloud by a voxel size of 0.8 meters in training and 0.4 meters in inference. MLP_f and MLP_g, are both of sizes $(300, 300)$. For the initial vertex state, we use an *MLP* of $(32, 64, 128, 300)$ for embedding raw points and another *MLP* of $(300, 300)$ after the *Max* aggregation. We set $T_h = 0.01$ in NMS.

Pedestrian and Cyclist. Again, we set (l_m, h_m, w_m) to the median bounding box size. We set $(0.88m, 1.77m, 0.65m)$ for Pedestrian and $(1.76m, 1.75m, 0.6m)$ for Cyclist. Similar to what we did with the Car class, we treat front-view and side-view objects as two different classes. Together with the *Background* class and the *DoNotCare* class, 6 classes are predicted. We build the graph using $r = 1.6m$, and downsample the point cloud by a voxel size of 0.4 meters in training and 0.2 meters in inference. MLP_f and MLP_g are both of sizes $(256, 256)$. For the vertex state initialization, we set $r_0 = 0.4m$. We use a an *MLP* of $(32, 64, 128, 256, 512)$ for embedding and an *MLP* of $(256, 256)$ to process the aggregated feature. We set $T_h = 0.2$ in NMS.

We train the proposed GNN end-to-end with a batch size of 4. The loss weights are $\alpha = 0.1$, $\beta = 10$, $\gamma = 5e-7$. We use stochastic gradient descent (SGD) with a stair-case learning-rate decay. For Car, we use an initial learning rate of 0.125 and a decay rate of 0.1 every $400K$ steps. We trained the network for $1400K$ steps. For Pedestrian and Cyclist,

we use a learning rate of 0.32 and a decay rate of 0.25 every $400K$ steps. We trained it for $1000K$ steps.

3.3.3 Data Augmentation

To prevent overfitting, we perform data augmentation on the training data. Unlike many approaches [8, 4, 5, 13] that use sophisticated techniques to create new ground truth boxes, we choose a simple scheme of global rotation, global flipping, box translation and vertex jitter. During training, we randomly rotate the point cloud by yaw $\Delta\theta \sim \mathcal{N}(0, \pi/8)$ and then flip the x-axis by a probability of 0.5. After that, each box and points within 110% size of the box randomly shift by $(\Delta x \sim \mathcal{N}(0,3), \Delta y = 0, \Delta z \sim \mathcal{N}(0,3))$. We use a 10% larger box to select the points to prevent cutting the object. During the translation, we check and avoid collisions among boxes, or between background points and boxes. During graph construction, we use a random voxel downsample to induce vertex jitter.

3.3.4 Results

We have submitted our results to the KITTI 3D object detection benchmark and the Bird's Eye View (BEV) object detection benchmark. In Table 3.1 and Table 3.2, we compare our results with the existing literature. The KITTI dataset evaluates the Average Precision (AP) on three difficulty levels: Easy, Moderate, and Hard. Our approach achieves the leading results in the Car detection of Easy and Moderate levels and also the Cyclist detection of Moderate and Hard levels. Remarkably, on the Easy level BEV Car detection, we surpass the previous state-of-the-art approach by 3.45. Also, we outperform fusion-based algorithms in all categories except for Pedestrian detection. In Figure 3.2, we provide

Method	Modality	Car			Pedestrian			Cyclist		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
UberATG-ContFuse[68]	Lidar + Image	82.54	66.22	64.04	N/A	N/A	N/A	N/A	N/A	N/A
AVOD-FPN[69]	Lidar + Image	81.94	71.88	66.38	50.80	42.81	40.88	64.00	52.18	46.61
F-PointNet[9]	Lidar + Image	81.20	70.39	62.19	51.21	44.89	40.23	71.96	56.77	50.39
UberATG-MMF[70]	Lidar + Image	86.81	76.75	68.41	N/A	N/A	N/A	N/A	N/A	N/A
VoxelNet[3]	Lidar	81.97	65.46	62.85	**57.86**	**53.42**	**48.87**	67.17	47.65	45.11
SECOND[8]	Lidar	83.13	73.66	66.20	51.07	42.56	37.29	70.51	53.85	53.85
PointPillars[4]	Lidar	79.05	74.99	68.30	52.08	43.53	41.49	75.78	59.07	52.92
PointRCNN[5]	Lidar	85.94	75.76	68.32	49.43	41.78	38.63	73.93	59.60	53.59
STD[13]	Lidar	86.61	77.63	**76.06**	53.08	44.24	41.97	**78.89**	62.53	55.77
Our Point-GNN	Lidar	**88.33**	**79.47**	72.29	51.92	43.77	40.14	78.60	**63.48**	**57.08**

Table 3.1: The Average Precision (AP) comparison of 3D object detection on the KITTI *test* dataset.

Method	Modality	Car			Pedestrian			Cyclist		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
UberATG-ContFuse[68]	Lidar + Image	88.81	85.83	77.33	N/A	N/A	N/A	N/A	N/A	N/A
AVOD-FPN[69]	Lidar + Image	88.53	83.79	77.9	58.75	51.05	47.54	68.06	57.48	50.77
F-PointNet[9]	Lidar + Image	88.70	84 .00	75.33	58.09	50.22	47.20	75.38	61.96	54.68
UberATG-MMF[70]	Lidar + Image	89.49	87.47	79.10	N/A	N/A	N/A	N/A	N/A	N/A
VoxelNet[3]	Lidar	89.60	84.81	78.57	**65.95**	**61.05**	**56.98**	74.41	52.18	50.49
SECOND[8]	Lidar	88.07	79.37	77.95	55.10	46.27	44.76	73.67	56.04	48.78
PointPillars[4]	Lidar	88.35	86.10	79.83	58.66	50.23	47.19	79.14	62.25	56.00
STD[13]	Lidar	89.66	87.76	**86.89**	60.99	51.39	45.89	81.04	65.32	57.85
Our Point-GNN	Lidar	**93.11**	**89.17**	83.9	55.36	47.07	44.61	**81.17**	**67.28**	**59.67**

Table 3.2: The Average Precision (AP) comparison of Bird's Eye View (BEV) object detection on the KITTI *test* dataset.

qualitative detection results on all categories. The results on both the camera image and the point cloud can be visualized. It must be noted that our approach uses *only* the point cloud data. The camera images are purely used for visual inspection since the *test* dataset does not provide ground truth labels. As shown in Figure 3.2, our approach still detects Pedestrian reasonably well despite not achieving the top score. One likely reason why Pedestrian detection is not as good as that for Car and Cyclist is that the vertices are not dense enough to achieve more accurate bounding boxes.

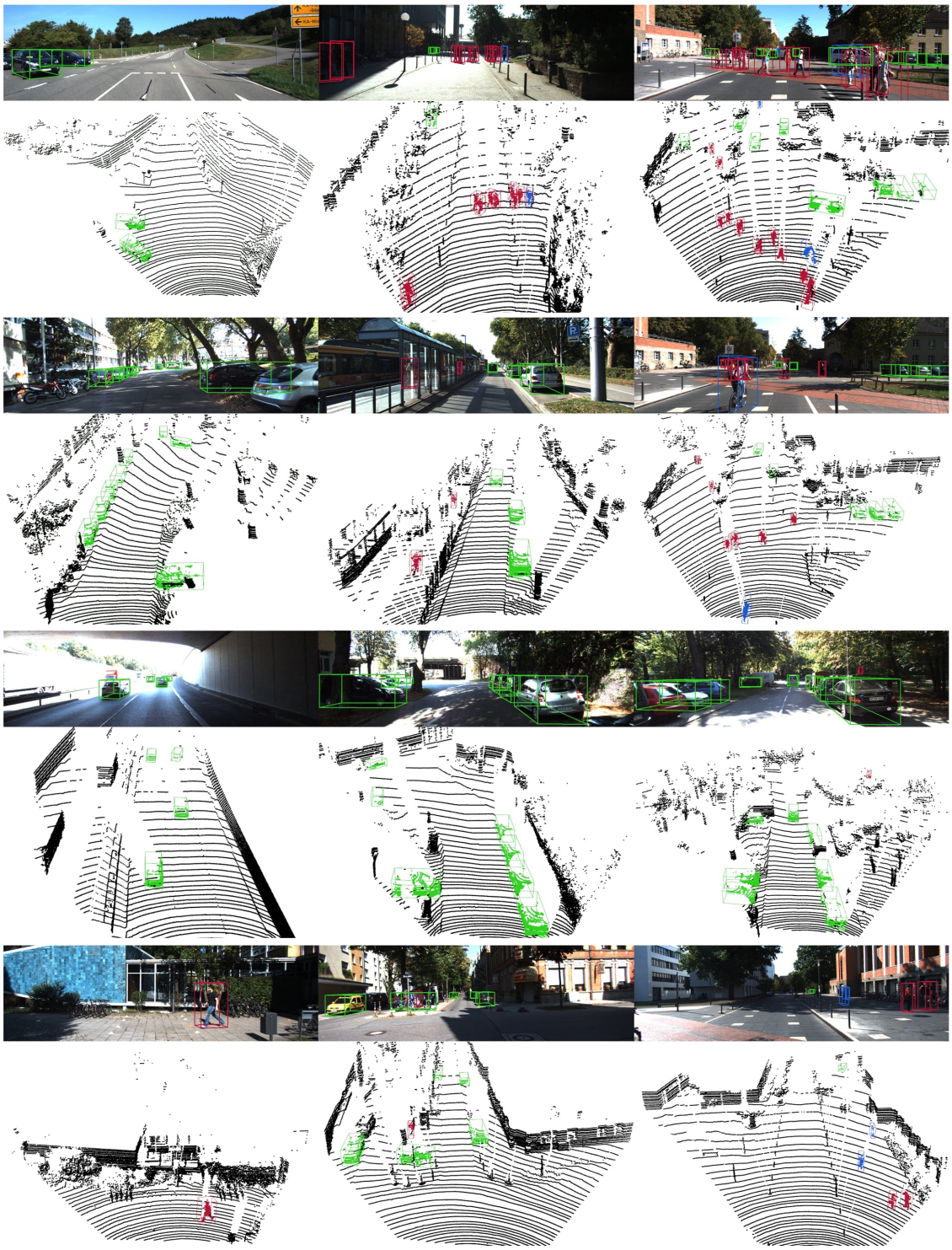

Figure 3.2: Qualitative results on the KITTI *test* dataset using Point-GNN. We show the predicted 3D bounding box of Cars (green), Pedestrians (red) and Cyclists (blue) on both the image and the point cloud. Best viewed in color.

	Box Merge	Box Score	Auto Reg.	BEV AP (Car) Easy	BEV AP (Car) Moderate	BEV AP (Car) Hard	3D AP (Car) Easy	3D AP (Car) Moderate	3D AP (Car) Hard
1	-	-	-	89.11	87.14	86.18	85.46	76.80	74.89
2	-	-	✓	89.03	87.43	86.39	85.58	76.98	75.69
3	✓	-	✓	89.33	87.83	86.63	86.59	77.49	76.35
4	-	✓	✓	89.60	88.02	86.97	87.40	77.90	76.75
5	✓	✓	-	90.03	88.27	87.12	88.16	78.40	77.49
6	✓	✓	✓	89.82	88.31	87.16	87.89	78.34	77.38

Table 3.3: Ablation study on the *val.* split of KITTI data.

3.3.5 Ablation Study

For the ablation study, we follow the standard practice [4, 13, 71] and split the training samples into a training split of 3712 samples and a validation split of 3769 samples. We use the training split to train the network and evaluate its accuracy on the validation split. We follow the same protocol and assess the accuracy by AP[1]. Unless explicitly modified for a controlled experiment, the network configuration and training parameters are the same as those in the previous section. We focus on the detection of Car because of its dominant presence in the dataset.

Box merging and scoring. In Table 3.3, we compare the object detection accuracy with and without box merging and scoring. For the test without box merging, we modify line 11 in Algorithm 1. Instead of taking the *median* bounding box, we directly take the bounding box with the highest classification score as in standard NMS. For the test without box scoring, we modify lines 12 and 13 in Algorithm 1 to set the highest classification score as the box score. For the test without box merging or scoring, we modify lines 11, 12, and 13, which essentially leads to standard NMS. Row 2 of Table 3.3 shows a baseline

[1]The ablation study uses the KITTI 11-recall-position AP.

implementation that uses standard NMS with the auto-registration mechanism. As shown in Row 3 and Row 4 of Table 3.3, both box merging and box scoring outperform the baseline. When combined, as shown in Row 6 of the table, they further outperform the individual accuracy in every category. Similarly, when not using auto-registration, box merging and box scoring (Row 5) also achieve higher accuracy than standard NMS (Row 1). These results demonstrate the effectiveness of the box scoring and merging.

Auto-registration mechanism. Table 3.3 also shows the accuracy improvement from the auto-registration mechanism. As shown in Row 2, by using auto-registration alone, we also surpass the baseline without auto-registration (Row 1) in every category of 3D detection and the moderate and hard categories of BEV detection. The performance in the easy category of BEV detection decreases slightly but remains close. Combining the auto-registration mechanism with box merging and scoring (Row 6), we achieve higher accuracy than using the auto-registration alone (Row 2). However, the combination of all three modules (Row 6) does not outperform box merging and score (Row 5). We hypothesize that the regularization likely needs to be tuned after adding the auto-registration branch.

We further investigate the auto-registration mechanism by visualizing the offset Δx in Equation 3.4. We extract Δx from different GNN iterations and add them to the vertex position. Figure 3.3 shows vertices that output detection results and their positions with added offsets. More examples are shown in Figure 3.4. We observe that the vertex positions with added offsets move towards the center of vehicles. We see such behaviors regardless of the original vertex position. In other words, when the GNN gets deeper, the relative coordinates of the neighbor vertices depend less on the center vertex position but

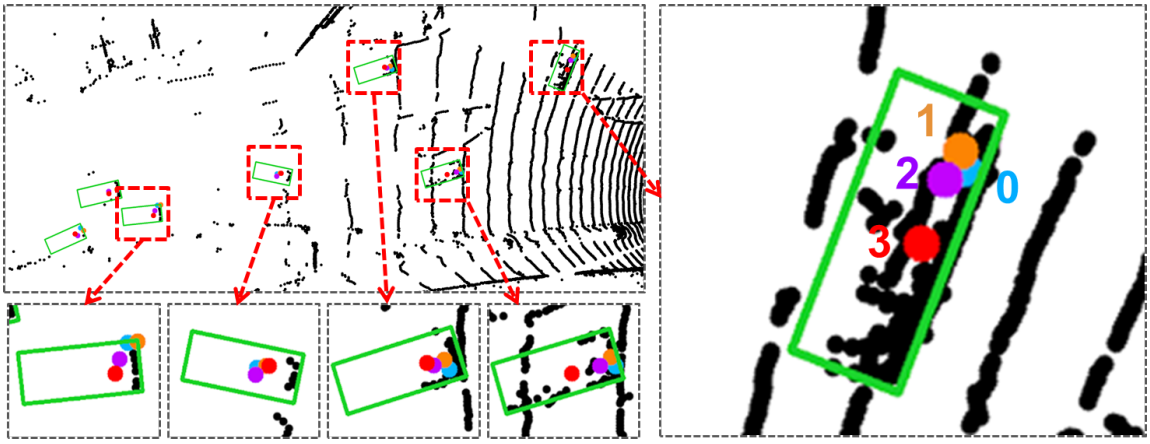

Figure 3.3: An detailed example from the *val.* split showing the vertex locations with added offsets. The blue dot indicates the original position of the vertices. The orange, purple, and red dots indicate the original position with added offsets from the first, the second, and the third graph neural network iterations. Best viewed in color.

more on the property of the point cloud. The offset Δx cancels the translation of the center vertex, and thus reduces the sensitivity to the vertex translation. These qualitative results demonstrate that Equation 3.4 helps to reduce the translation variance of vertex positions.

Point-GNN iterations. Our Point-GNN refines the vertex states iteratively. In Table 3.4, we study the impact of the number of iterations on the detection accuracy. We train Point-GNNs with $T = 1$, $T = 2$, and compare them with $T = 3$, which is the configuration in Section 3.3.4. Additionally, we train a detector using the initial vertex state directly without any Point-GNN iteration. As shown in Table 3.4, the initial vertex state alone achieves the lowest accuracy since it only has a small receptive field around the vertex. Without Point-GNN iterations, the local information cannot flow along the graph edges, and therefore its receptive field cannot expand. Even with a single Point-GNN iteration $T = 1$, the accuracy improves significantly. $T = 2$ has higher accuracy than $T = 3$, which

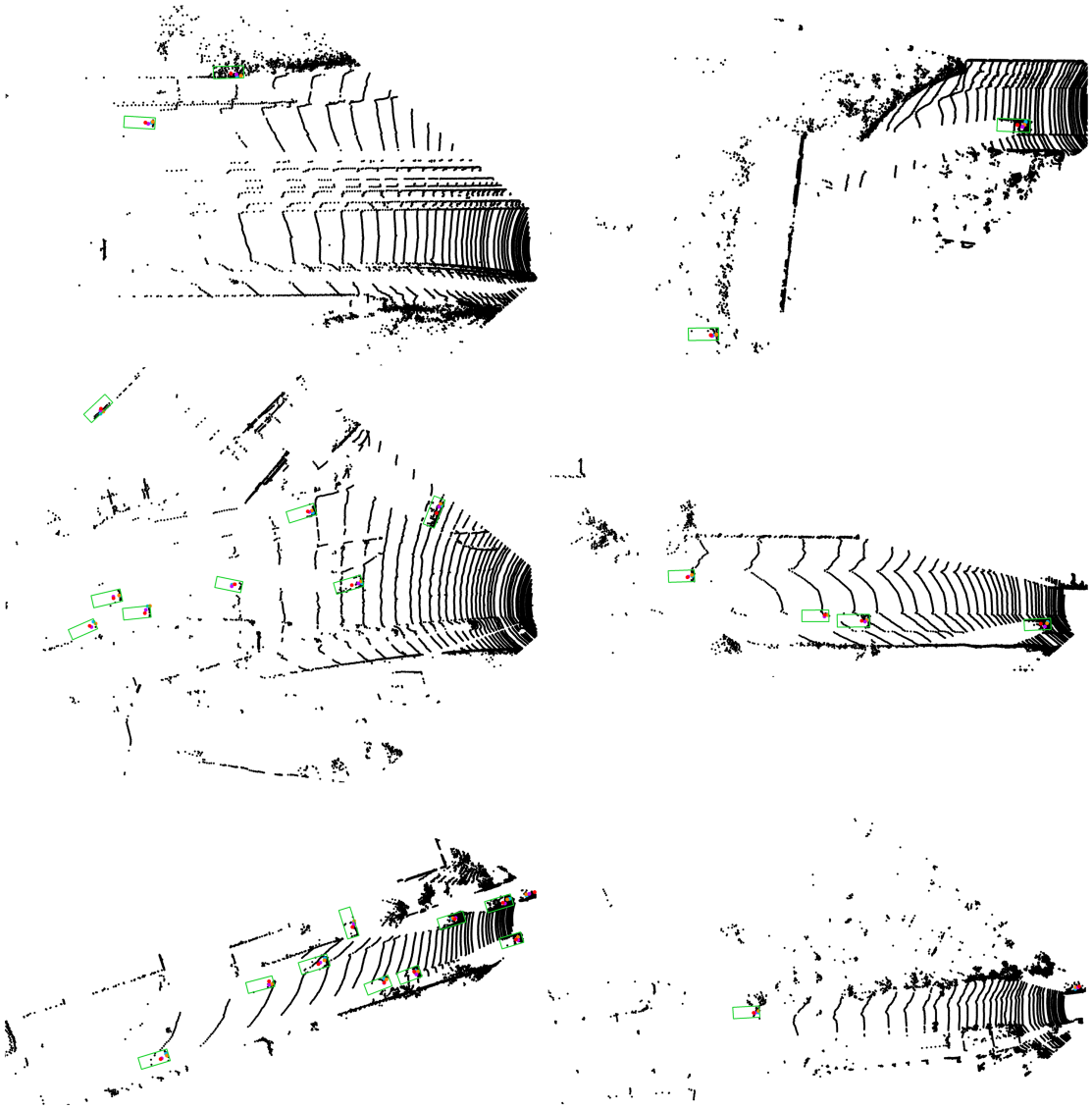

Figure 3.4: Examples from the *val.* split showing the vertex locations with added offsets.

Number of iterations	BEV AP (Car)			3D AP (Car)		
	Easy	Moderate	Hard	Easy	Moderate	Hard
T = 0	87.24	77.39	75.84	73.90	64.42	59.91
T = 1	89.83	87.67	86.30	88.00	77.89	76.14
T = 2	90.00	88.37	87.22	88.34	78.51	77.67
T = 3	89.82	88.31	87.16	87.89	78.34	77.38

Table 3.4: Average precision on the KITTI *val.* split using different numbers of Point-GNN iterations.

is likely due to the training difficulty when the neural network goes deeper.

Running-time analysis. The speed of the detection algorithm is important for real-time applications such as autonomous driving. However, multiple factors affect the running time, including algorithm architecture, code optimization and hardware resource. Furthermore, optimizing the implementation is not the focus of this work. However, a breakdown of the current inference time helps with future optimization. Our implementation is written in Python and uses Tensorflow 1.15 for GPU computation. We measure the inference time on a desktop with Xeon E5-1630 CPU and GTX 1070 GPU. The average processing time for one sample in the validation split is 643ms. Reading the dataset and running the calibration takes 11.0% time (70ms). Creating the graph representation consumes 18.9% time (121ms). The inference of the GNN takes 56.4% time (363ms). Box merging and scoring take 13.1% time (84ms). We also measure the running time for models with and without auto-registration or box merging and scoring. Table 3.5 shows the results. The running time results indicate that both auto-registration and box merging and score do not largely increase the running time. See our code for implementation details.

Robustness on Lidar sparsity. The KITTI dataset collects point cloud data using a 64-scanning-line lidar. Such a high-density lidar usually leads to a high cost. Therefore, it

Box Merging and Scoring	Auto Registration	Running Time					
		Read Input	Graph Construction	GNN	Box Decoding	NMS	Total
-	-	48 ms	123 ms	361 ms	4 ms	48 ms	584 ms
-	✓	48 ms	123 ms	362 ms	4 ms	50 ms	587 ms
✓	-	71 ms	125 ms	362 ms	4 ms	83 ms	645 ms
✓	✓	70 ms	121 ms	363 ms	5 ms	84 ms	643 ms

Table 3.5: The average running time for one sample in the KITTI *validation* split. The inference time is measured on a desktop with Xeon E5-1630 CPU and GTX 1070 GPU using a Python implementation

Number of scanning lines	BEV AP (Car)			3D AP (Car)		
	Easy	Moderate	Hard	Easy	Moderate	Hard
64	89.82	88.31	87.16	87.89	78.34	77.38
32	89.62	79.84	78.77	85.31	69.02	67.68
16	86.56	61.69	60.57	66.67	50.23	48.29
8	49.72	34.05	32.88	26.88	21.00	19.53

Table 3.6: Average precision on downsampled KITTI *val.* split.

is of interest to investigate the object detection performance in a less dense point cloud. To mimic a lidar system with fewer scanning lines, we downsample the scanning lines in the KITTI validation dataset. Because KITTI gives the point cloud without the scanning line information, we use k-means to cluster the elevation angles of points into 64 clusters, where each cluster represents a lidar scanning line. We then downsample the point cloud to 32, 16, 8 scanning lines by skipping scanning lines in between. Our test results on the downsampled KITTI validation split are shown in Table 3.6. The accuracy for the moderate and hard levels drops fast with downsampled data, while the detection for the easy level data maintains a reasonable accuracy until it is downsampled to 8 scanning lines. This is because that the easy-level objects are mostly close to the lidar, and thus have a dense point cloud even if the number of scanning lines drops.

3.4 Summary

In this chapter, we have presented a graph neural network, named Point-GNN, to detect 3D objects from a graph representation of the point cloud. By using a novel graph representation, we encode the irregular point cloud compactly without mapping the sparse point cloud to dense grids. Point-GNN also reuses graph edges to aggregate features and avoids sampling or grouping repeatedly. Our Point-GNN achieves good accuracy in both the 3D and Bird's Eye View object detection of the KITTI benchmark, demonstrating its effectiveness as a point cloud backbone for object detection. It can even surpass many fusion-based algorithms and remains relatively robust when the density of a point cloud decreases. In the next chapter, we aim to increase object detection accuracy under the constraint of point cloud density. We describe our fusion method which combines the inputs from cameras with our Point-GNN.

Chapter 4

Fusion of Point Cloud and Images

4.1 Introduction

Many autonomous vehicles have lidars as primary sensors, which provide precise point clouds. Lidar-based 3D detection algorithms expect a point cloud to contain the geometric characteristics of objects. Therefore, they benefit from high point density. However, acquiring a dense point cloud is not trivial. It either requires a high-resolution lidar system or the temporal aggregation of multiple lidar sweeps. A high-resolution lidar system is still too expensive in many application domains including consumer vehicles, warehouses and work zone equipment. Also, lidar aggregation suffers from registration errors due to vehicle movement and increases latencies. Even within a dense point cloud, a sparse region still inevitably exists due to sensing distance and occlusions. Therefore, it is desirable to explore alternative approaches to increase object detection accuracy under the constraint of point cloud density.

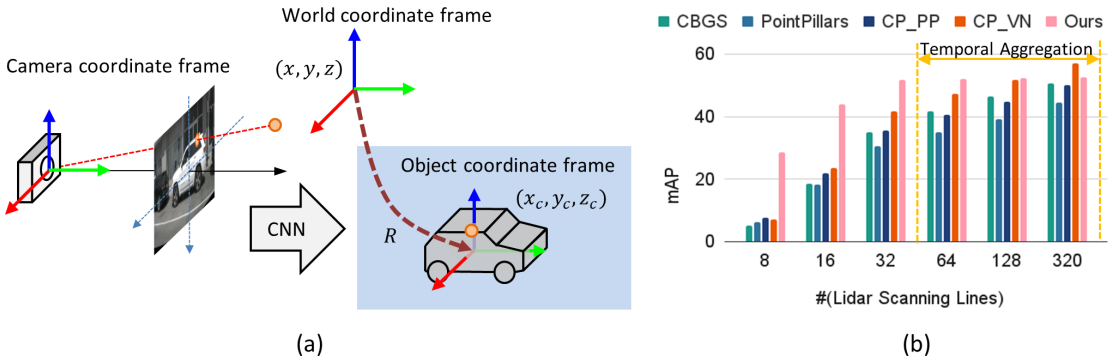

Figure 4.1: (a). Our method uses a lidar point and the image to estimate the point's local coordinates in the object frame. By registering a point's local coordinates with its world coordinates, we locate the object in the world coordinates. (b). Our method detects objects with higher mean average precision (mAP) in sparse point clouds.

In this chapter, we describe our method to boost object detection accuracy on a sparse point cloud using cameras as a complementary source. Unlike lidars, cameras generate images with dense pixel value at a relatively low cost. In addition, an RGB image contains rich color and texture information, which is of great value to inferring object structure. However, as an image does not contain direct depth measurement, monocular 3D object detection often has difficulty recovering the object's position. Previous sensor fusion methods combine image features and point cloud features to achieve higher detection accuracy. For instance, [21, 22] add image semantic labels to the point cloud, [1, 9] combine object proposals from both the camera and the lidar, and [41] utilizes depth estimation from images. While achieving remarkable results, those methods still rely on dense point clouds. Meanwhile, how to use an image jointly with a sparse point cloud to achieve high detection accuracy remains an open question. Our approach addresses this question explicitly.

As shown in Figure 4.1, we leverage image features to infer an object's local 3D struc-

ture. We project a lidar point to the images and use a neural network to aggregate image features from multiple cameras. The neural network predicts every point's local 3D coordinates in the object frame. By matching a point's lidar coordinates and predicted local coordinates, we calculate the object's position. Our minimum requirement for detection is only a single lidar point on the object and the target object's image. We use a 3D pooling method to combine neighbor point features (if neighbors exist) and refine the detection results. Our experiment with the large nuScenes [20] dataset and the Lyft Level5 [23] dataset show that the proposed method largely improves the detection accuracy on sparse point cloud regions.

In summary, our contributions are:

- We propose a pipeline to extract the local geometry of an object from images which considers the placement of multiple cameras on the AV.
- We design a registration method for matching an object's local geometry with lidar points to estimate the object's global coordinates.
- We provide extensive experiments on both nuScenes and Level5 datasets, demonstrating the effectiveness of our method on sparse point clouds.

4.2 Our Fusion Method

As illustrated in Figure 4.2, our 3D detection framework takes images from multiple cameras and a sparse point cloud as input. We use a convolutional neural network to extract image features from the camera images and re-project the image features to the lidar points.

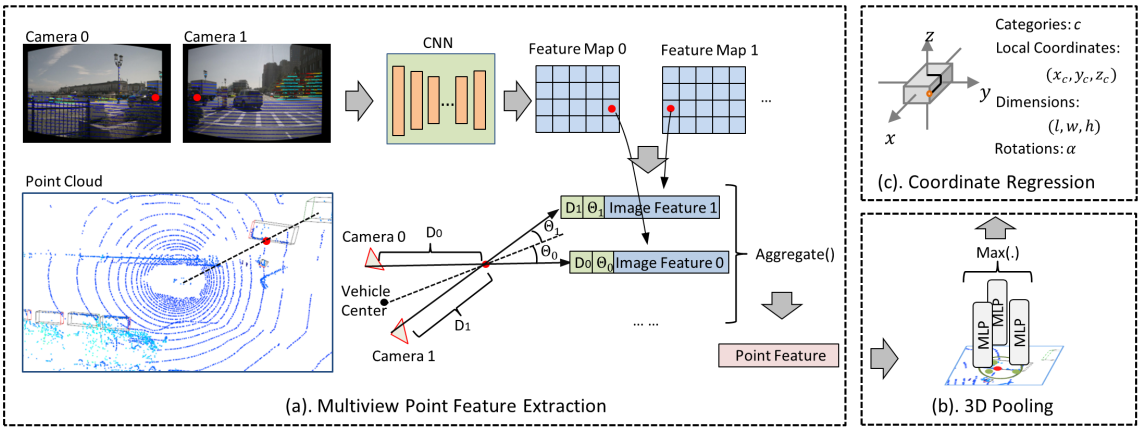

Figure 4.2: The architecture of our proposed approach. First, (a) point features from multiple image views are extracted and aggregated. Then, (b) a 3D pooling operation combines spatially close point features. The combined features are used to predict (c) local coordinates, which are later registered to lidar coordinates

We aggregate the image features from different camera views and pool the features from neighborhood points. The pooled features predict each lidar point's coordinates in its local object coordinate system, and the transformation between the local and lidar coordinates gives the detected object's location and orientation. We describe the details below.

4.2.1 Multiview Point Feature Extraction

We first extract image features by a convolutional neural network backbone. Given K synchronized images $\{I_0, I_1, ..., I_{K-1}\}$ from different cameras on the vehicle, a weight-sharing CNN computes a feature map F for each image:

$$F_k = CNN(I_k) \tag{4.1}$$

Given a point cloud $\{p_i = (x_i, y_i, z_i)\}^N$, we gather each point's image feature vector from the feature maps within which the point is visible:

$$F_{p_i}^k = F_k[u_{p_i}^k, v_{p_i}^k] \tag{4.2}$$

where, $(u_{p_i}^k, v_{p_i}^k)$ is p_i's projected position in the feature map F_k. Note that p_i can be visible in multiple cameras and it can have multiple image features.

In general, camera centers do not align. Therefore, each camera image observes the 3D points from slightly different viewing angles. To capture these variations, we add the viewing angle Θ as a part of the point's image features. We calculate Θ with regard to the ray from the vehicle center to the point, as shown in Figure 4.2(a). Given the vehicle center position (e_x, e_y), camera center position (c_x^k, c_y^k) and the point (x_i, y_i), we compute p_i's viewing angle $\Theta_{p_i}^k$ from camera k as follows:

$$\Theta_{p_i}^k = \text{atan2}(\mathbf{v_i^e} \times \mathbf{v_i^k}, \mathbf{v_i^e} \cdot \mathbf{v_i^k}) \tag{4.3}$$

where,

$$\mathbf{v_i^e} = [x_i - e_x, y_i - e_y] \tag{4.4}$$

$$\mathbf{v_i^k} = [x_i - c_x^k, y_i - c_y^k] \tag{4.5}$$

Due to the misalignment between the lidar center and camera center, artifacts may exist where occluded lidar points are still visible in the image. A point's image feature vector may encode the foreground image instead of the actual object behind it. Also,

multiple lidar points may project to the same image area. To mitigate errors caused by such ambiguity between points and image features, we compute the distance $d^k_{p_i}$ between point p_i and the k^{th} camera center. Intuitively, this distance acts as a query within the network to fetch different spatial features. As will be shown in Section 4.3.5, the distance value improves the detection accuracy significantly.

Overall, the feature vector for p_i from camera k is as follows:

$$y^k_{p_i} = [d^k_{p_i}, \Theta^k_{p_i}, F^k_{p_i}] \tag{4.6}$$

We then combine the feature vectors from multiple cameras by aggregation:

$$y_{p_i} = \max(\{MLP(y^k_{p_i})\}) \tag{4.7}$$

4.2.2 3D Pooling

The aggregated point feature y_{p_i} only uses the images and a single lidar point. Since y_{p_i} does not rely on any other points in the point cloud, y_{p_i} is robust when the point cloud is sparse. However, y_{p_i} by itself does not contain potential geometric information from the point cloud, which does affect the detection accuracy. To utilize the point cloud structure, we adapt a 3D pooling module to combine the point cloud features within a local radius r. The pooling module predicts a residue in the point feature:

$$\delta y_{p_i} = \max_{||p_i - p_j||_2^2 < r} (\{MLP([y_{p_j}, p_j - p_i + MLP(y_{p_i})])\}) \tag{4.8}$$

$$y'_{p_i} = y_{p_i} + \delta y_{p_i} \tag{4.9}$$

If the point cloud contains points from a different time instant due to temporal aggregation, the images and those points are asynchronous. The asynchronous points do not have corresponding image features. Therefore, we treat them differently in the 3D pooling module. We pool those points' timestamps t and lidar intensities instead of image features:

$$y^t_{p_i} = \max_{||p_i - p_j||_2^2 < r} (\{MLP([t_j - t_i, a_j, p_j - p_i])\}) \tag{4.10}$$

We then concatenate the $y^t_{p_i}$ to the point image features:

$$y'_{p_i} = [y_{p_i} + \delta y_{p_i}, y^t_{p_i}] \tag{4.11}$$

Note that both the pooling of the neighborhood point image features and asynchronous points allow the neighborhood to be empty. Therefore, the pooling module works naturally with a sparse point cloud.

4.2.3 Local Object Coordinate Regression

Given the point feature y'_{p_i}, we aim to predict p_i's category and its coordinates within the local object frame. We first use a classification header to predict the point's category. Similar to the MultiBin [36] approach, we also split each object category into multiple bins based on the angle α between the object orientation and the ray from a point to the vehicle center. These bins do not overlap, and we enforce a categorical classification for the bins. We use independent classification headers for different object categories. As a lidar point can belong to multiple classes, we use multi-label classification across different classes. For instance, a point on a driver belongs to both the person and vehicle.

Besides the classification header, we use a localization header to predict the object's dimensions $[l, w, h]$ and, more importantly, the point's coordinates within the local object coordinate system $[x_c, y_c, z_c]$. As illustrated in Figure 4.2(c), the local object coordinates are defined relative to the object center and are invariant to the translation and rotation of the object. Our prediction of the local coordinates avoids solving the object's location directly from the monocular image. The localization header also predicts the observation angle α.

The local object coordinate frame and the global coordinate frame are connected by a rigid transformation, which comprises a rotation $\mathbf{R}$ and a translation $\mathbf{t}$. Locating an object's position is equivalent to finding $\mathbf{R}$ and $\mathbf{t}$. For each lidar point $p_i = (x_i, y_i, z_i)$ of an object, the network predicts its observation angle α and local coordinate $\mathbf{q} = (x_c, y_c, z_c)$. A direct solution from the network prediction is given by:

$$\mathbf{t} = \mathbf{p} - \mathbf{R}\mathbf{q} \tag{4.12}$$

$$\mathbf{R} = \begin{bmatrix} \cos(\beta+\alpha) & -\sin(\beta+\alpha) & 0 \\ \sin(\beta+\alpha) & \cos(\beta+\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.13}$$

$$\beta = \arctan(y_i, x_i) \tag{4.14}$$

Loss Definition. For a total number of C classes and K_i bins for class i, the classification header generates probabilities $(s_{(0,0)}, ..., s_{(0,K_0)}), ..., (s_{(C-1,0)}, ..., s_{(C-1,K_0)})$, where, $(s_{(i,0)}, ..., s_{(i,K_i)})$ are the categorical probabilities for each bin in class i and $s_{(i,0)}$ is the probability for the negative. We use softmax to generate $(s_{(i,0)}, ..., s_{(i,K_i)})$, and the classification

loss L_c for N lidar points is defined as:

$$L_c = -\frac{1}{C}\frac{1}{N}\sum_{j=0}^{N-1}\sum_{i=0}^{C-1}\sum_{k=0}^{K_i} y^j_{(i,k)} \log(s^j_{(i,k)}) \tag{4.15}$$

where, $y_{(i,k)}$ is the one-hot label for class i.

The localization header predicts the encoded form of the object dimension $[l,w,h]$, the object local coordinates $[x_c,y_c,z_c]$, and the observation angle $\hat{\theta}$:

$$\begin{aligned} &x'_c = x_c/t_l && y'_c = y_c/t_w && z'_c = z_c/t_h \\ &l' = \log(l/t_l) && w' = \log(w/t_w) && h' = \log(h/t_h) \\ &\hat{\theta}' = \log(\hat{\theta}/t_\theta) \end{aligned} \tag{4.16}$$

t_l,t_w,t_h,t_θ are scaling constants. For M points that belong to an object of interest, the localization loss L_{loc} is defined as:

$$L_{loc} = \frac{1}{M}\sum_{i=0}^{M-1} SmoothL1(b^i, b^i_{gt}) \tag{4.17}$$

where, $b = [x'_c, y'_c, z'_c, l', w', h', \hat{\theta}]$ and b_{gt} is the ground truth.

To prevent overfitting, we also add L1 regularization on the headers. The overall loss is a weighted combination with constant λ_1 and λ_2:

$$L = L_{cls} + \lambda_1 L_{loc} + \lambda_2 L_{reg} \tag{4.18}$$

4.3 Experimental Evaluation

4.3.1 Dataset

We evaluate our 3D object detection using the large-scale autonomous driving datasets nuScenes [20] and the Lyft Level 5 [23]. Both the nuScenes and the Level 5 datasets include six camera views covering the vehicle's entire surroundings, while other popular datasets such as the KITTI [18] and the Waymo Open [19] datasets contain only partial camera coverage. The nuScenses and the Level 5 datasets also register their point clouds globally, which enable direct experiments with temporal point cloud aggregations and are appropriate for use in our evaluations.

4.3.2 Implementation Details

We pre-process the images using a homography transformation to rotate the cameras to align all cameras. Images are then warped to a cylindrical view using the same focal length. We use random image flipping, color jittering, and region cutout as the image augmentation method. We also randomly jitter the autonomous vehicle's center coordinates. To balance the distribution of objects at different distances, we assign a weight to each lidar point such that the sum of the point weights on each object of the same category is the same.

Figure 4.3 shows the detailed network architecture. We group images of six cameras and feed them to a Resnet-FPN network. The network comprises a ResNet-50 [72] backbone and additionally upsampling layers to construct a feature pyramid network [73]. Each upsampling layer is a $2\times$ nearest-neighbor interpolation followed by a single convolutional

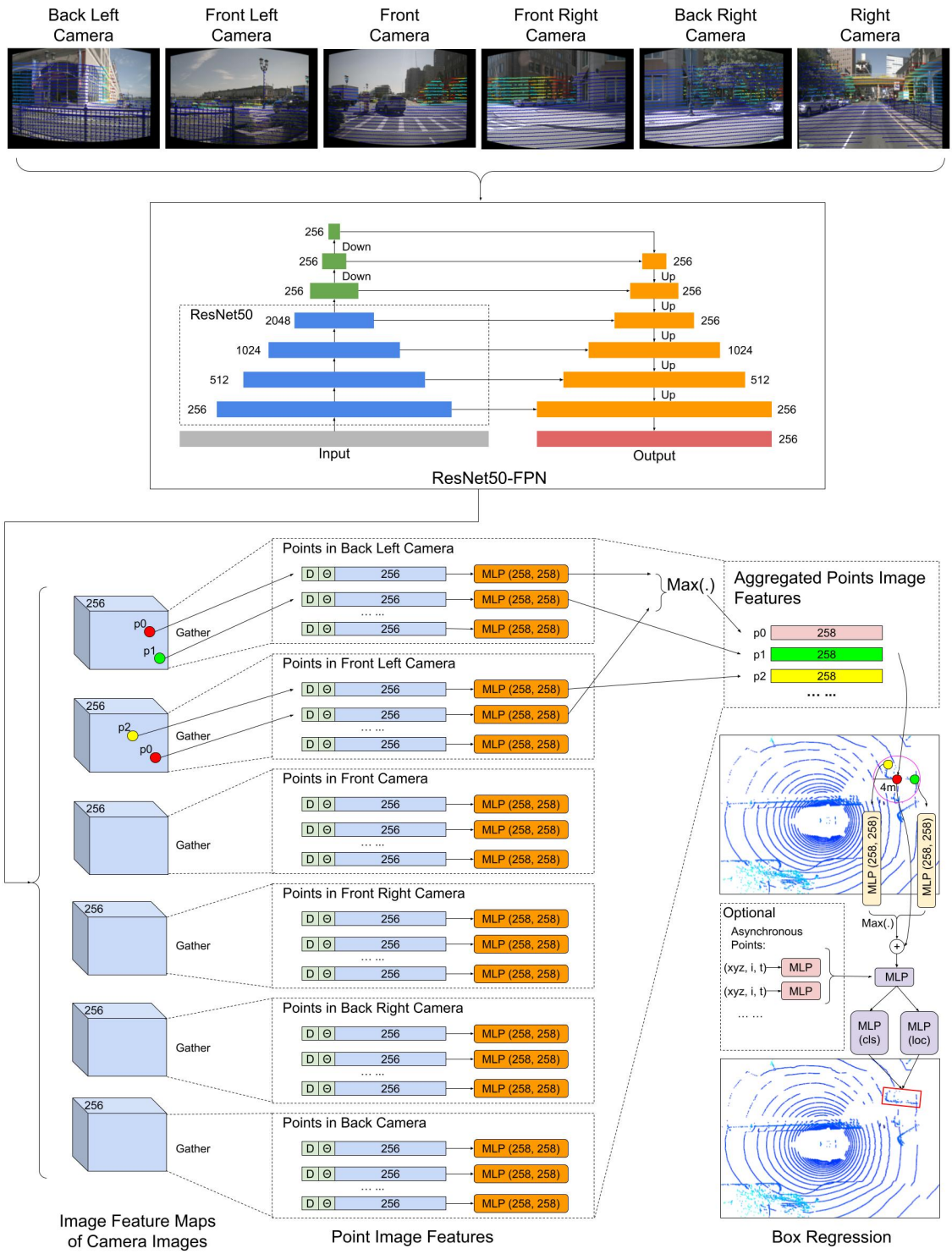

Figure 4.3: The detailed model configuration of the proposed fusion model.

layer and a residual connection. We take the feature level with the highest resolution as output, which is $\frac{1}{4}$ of the input size and have a depth of 256. As the input image size is 1024×512, the output image feature map size is 256×128.

We project the input point cloud to the image feature maps and filter out points that are outside the images. We then gather the image feature vector at each point's corresponding position, resulting in an image feature vector of 256 for each point. We concatenate the image feature vector with the corresponding point's distance to the camera center and the relative yaw angle as in Equation 4.3. The concatenated feature vector has a length of 258. An MLP of [258, 258] units process the concatenated feature vector. We then re-project the results to each lidar point. If a lidar point is visible in multiple images, i.e., it has multiple image feature vectors from multiple cameras, we aggregate multiple image vectors by taking their maximum.

As the aggregated point features alone do not take advantage of point cloud geometrics, we pool the aggregated point features within a radius of 4 meters. To reduce the computational cost, we first select a set of keypoints by downsampling the point cloud using a voxel size of 0.5 meters. Next, we search a 4-meter radius around each keypoint for neighborhood points. We only use a maximum of 16 neighborhood points and randomly drop points if there are more neighbors. The neighbors' point features are concatenated with their relative position to the keypoints and fed to another MLP of [258, 258], followed by a maximum aggregation. The aggregated features are added to the keypoint's point feature, following Equation 4.8. Optional features from temporally aggregated lidar points can be further concatenated to the keypoint's point feature as shown in Equation 4.10.

The feature vector of the keypoints is then processed by the classification headers and localization headers. For each object category, the classification header is an MLP of [128, 128, K] units, where K is the number of orientation bins of that category. The localization header is an MLP of [128, 128, 7] for each bin. Note that the localization header regresses a fine-grain orientation within each bin. The nuScenes dataset has ten object categories for 3D object detection. As mentioned in Chapter 4.2.3, we split the object of the same category into multiple bins using the object's orientation. We split all objects into four bins $\theta \in [0, \frac{\pi}{2})$, $[\frac{\pi}{2}, \pi)$, $[\pi, \frac{3\pi}{2})$, and $[\frac{3\pi}{2}, 2\pi)$, except two categories, barriers and traffic cones. For barriers we only predict the orientation in $[0, \frac{\pi}{2}), [\frac{\pi}{2}, \pi)$ because it is symmetric. For traffic cones, we do not predict their orientation and set $\theta = 0$.

We trained the model end to end with a batch size of 2 using the Adam Optimizer [74] with $\gamma_1 = 0.9$ and $\gamma_2 = 0.999$. We use cosine learning rate decay with 8K warmup steps. The initial learning rate is 10^{-4}, and it decays to 0 in 256K iterations. For all models except the model with aggregated lidar points, we train for 256K steps. Our implementation uses Python and the Tensorflow 2.0 library. Each training takes around three days on a PC with two 1080Ti GPUs. We reduced the training steps to 200K for the model with lidar aggregation because of the increased computational cost.

4.3.3 Results on the nuScenes Dataset

We trained our model on the nuScenes training set and tested it with different numbers of lidar scanning lines on the validation set. Table 4.1 and Figure 4.1 present the results. A single sweep of lidar in nuScenes contains 32 lines. For experiments with 8 and 16 lidar scanning lines, we skip lidar scan lines appropriately. For experiments with more than 32

# Scan Lines	# Sweeps	Method	mAP ↑	car	truck	bus	trailer	cons.	ped.	moto.	bicycle	cone	barrier	mAOE↓
320	10	PointPillars [4]	44.6	81.1	50.0	63.4	35.2	11.8	72.2	29.1	6.3	47.0	49.9	0.32
		CBGS [75]	50.6	81.5	51.6	66.7	**37.4**	14.6	77.8	41.7	17.6	57.2	59.3	**0.27**
		CP-PP [76]	50.2	84.0	53.5	64.3	31.9	12.3	78.9	43.9	18.2	54.8	60.3	0.39
		CP-VN [76]	**57.1**	**85.2**	56.3	**67.0**	35.6	16.3	**84.7**	56.7	35.8	67.2	**65.9**	0.31
		Ours	52.6	80.3	45.3	61.0	23.1	**16.6**	77.1	**57.0**	**42.1**	**68.5**	55.0	0.45
256	8	PointPillars [4]	43.5	80.5	49.7	62.4	34.4	10.8	71.3	27.3	4.6	45.4	48.7	0.32
		CBGS [75]	50.2	81.6	51.3	**67.0**	**37.3**	13.9	77.5	42.4	15.8	56.8	58.7	**0.27**
		CP-PP [76]	49.4	83.5	52.5	63.8	31.7	11.4	78.0	42.0	16.8	54.3	59.7	0.40
		CP-VN[76]	**56.1**	**84.7**	**55.6**	66.3	34.6	15.6	**84.1**	53.8	33.8	66.6	**65.6**	0.32
		Ours	52.6	80.3	45.3	60.7	23.1	**16.7**	77.1	**56.7**	**42.6**	**68.6**	55.3	0.45
128	4	PointPillars [4]	39.3	76.8	44.8	59.1	31.7	7.2	60.8	24.9	1.3	40.9	45.2	0.38
		CBGS [75]	46.57	79.3	47.9	64.6	35.1	10.6	72.8	37.2	10.1	52.7	55.5	**0.31**
		CP-PP [76]	44.8	80.4	48.5	60.2	29.9	7.4	71.0	34.7	8.0	50.2	57.6	0.44
		CP-VN [76]	51.7	**82.2**	**52.1**	**63.4**	**33.5**	10.9	**80.4**	44.9	21.9	63.4	**63.9**	0.36
		Ours	**52.3**	80.0	45.2	61.2	23.4	**16.9**	75.9	**55.5**	**41.8**	**68.2**	54.9	0.4768
64	2	PointPillars [4]	34.9	72.5	39.8	54.9	27.5	3.7	51.2	19.5	0.1	35.9	43.5	0.51
		CBGS [75]	41.7	75.6	43.3	60.4	**29.3**	6.0	64.6	28.8	4.4	46.5	51.9	**0.41**
		CP-PP [76]	40.5	76.9	43.5	56.2	27.1	4.7	63.7	29.9	3.3	44.0	55.3	0.55
		CP-VN [76]	47.2	78.6	**47.3**	59.9	29.2	6.9	**75.3**	39.9	15.3	58.2	**60.9**	0.47
		Ours	**52.0**	**79.6**	44.9	**61.2**	23.4	**16.4**	74.7	**54.7**	**41.6**	**68.9**	55.0	0.51
32	1	PointPillars [4]	30.5	68.3	35.1	50.5	22.6	2.3	42.4	13.9	0.0	30.5	39.5	0.63
		CBGS [75]	35.1	70.5	38.2	55.0	20.8	2.8	55.6	19.9	1.3	38.9	48.0	**0.53**
		CP-PP [76]	35.6	72.9	37.3	51.8	21.10	2.2	56.7	23.0	0.7	37.9	52.2	0.62
		CP-VN [76]	41.6	74.7	41.9	53.4	**24.2**	3.7	69.4	31.1	8.8	51.7	**57.6**	0.55
		Ours	**51.7**	**79.1**	**44.7**	**61.3**	23.2	**16.5**	**74.4**	**53.5**	**41.1**	**68.3**	54.9	0.54
16	1	PointPillars [4]	18.2	48.4	20.4	34.2	10.7	0.4	23.3	2.4	0.0	19.2	23.1	0.72
		CBGS [75]	18.6	48.9	20.0	31.0	5.5	0.2	26.3	5.5	0.0	21.3	27.4	0.63
		CP-PP [76]	22.0	52.9	21.9	34.1	10.2	0.5	33.3	4.9	0.0	24.9	37.3	0.72
		CP-VN [76]	23.6	53.3	24.4	31.4	7.5	0.2	40.0	8.2	0.7	32.6	37.2	0.63
		Ours	**44.0**	**63.0**	**39.2**	**55.5**	**21.40**	**13.8**	**62.7**	**45.7**	**30.5**	**59.9**	**48.6**	**0.55**
8	1	PointPillars [4]	6.4	22.9	6.9	11.3	0.4	0.0	5.2	0.0	0.0	7.9	9.0	0.83
		CBGS [75]	5.2	21.0	5.0	10.7	0.0	0.0	2.3	0.0	0.0	6.5	6.4	0.91
		CP-PP [76]	7.7	25.9	6.6	9.9	1.0	0.0	8.1	0.0	0.0	10.3	14.7	0.85
		CP-VN [76]	7.2	25.0	7.1	9.6	0.0	0.0	8.5	0.0	0.0	11.6	10.2	0.84
		Ours	**28.7**	**42.3**	**28.1**	**33.9**	**11.7**	**9.9**	**44.3**	**23.0**	**15.7**	**40.3**	**31.7**	**0.62**

Table 4.1: The Average Precision (AP) and Average Orientation Error (AOE) comparison on nuScenes validation dataset with different numbers of lidar scanning lines.

	Modality	**mAP**
PointPillars[4]	Lidar	40.1
PointPainting [21]	Lidar+Images	46.4
CenterPoint[76]	Lidar	58.0
MonoDis[77]	Images	30.4
Ours	Lidar+Images	50.8

Table 4.2: Average precision on nuScenes test dataset. We test the model accuracy with ten lidar sweeps, the common practice among the state-of-the-art approaches.

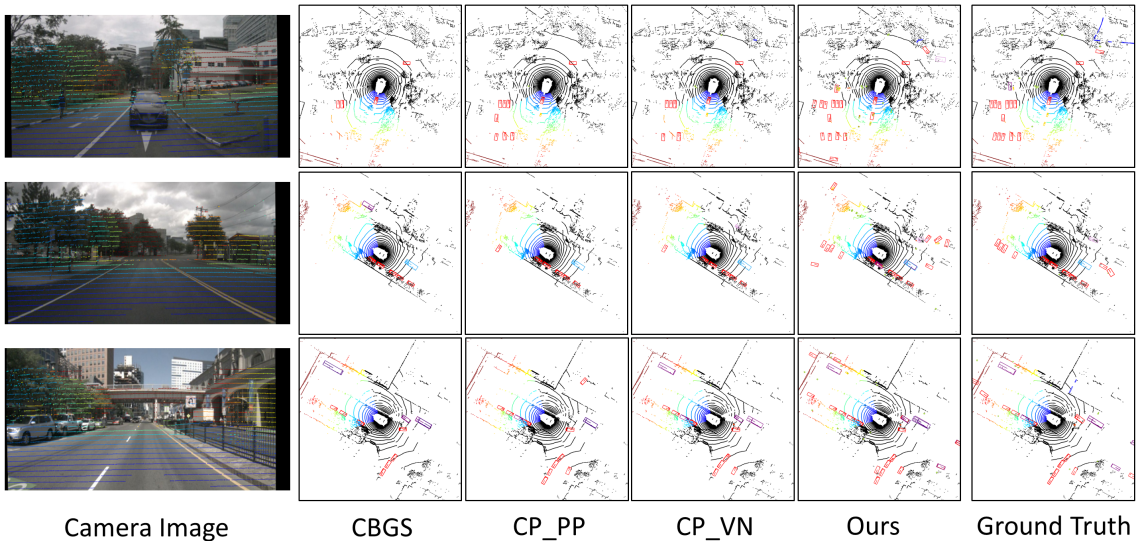

Figure 4.4: The qualitative comparison of methods on the nuScenes validation dataset (a single lidar sweep of 32 lines). We visualize images from one camera and highlight the points in the images. Our method detects objects in sparse point regions and increases the detection range.

lines, we use the official devkit of nuScenes to aggregate multiple lidar sweeps over time. In Table 4.1, we compare the nuScenes mean Average Precision (mAP) and Average Orientation Error (AOE) with state-of-the-art methods. Our method leads in accuracy with a large margin in the case of a single lidar sweep (8 lines, 16 lines and 32 lines), demonstrating that our method achieves our stated goal. In the case of multiple lidar sweeps, our method remains comparable to the others, despite a relatively simple structure to utilize point cloud features. We also report the accuracy on the test dataset in Table 4.2. As the nuScenes test server forbids frequency upload, we only test the model accuracy with ten lidar sweeps, the common practice among the state-of-the-art approaches. We emphasize that our focus is on the performance of a sparse point cloud. Figure 4.4 shows the qualitative comparison of different methods using a single lidar sweep (32 scan lines). Our method detects objects from more sparse point cloud regions and increases the detection range. In Figure 4.5, we show more qualitative results of our model on different point cloud sparsity.

4.3.4 Results on the Level 5 Dataset

We trained our model on the Level 5 training split and evaluated the model on the validation split [78]. The Level 5 dataset is collected using two different sensor configurations. Around 80% of the samples contain point clouds from a 40-beam lidar and camera images of resolution 1224×1024. The remaining samples contain point clouds from a 64-beam lidar and camera images of 1920×1080. To remove the discrepancy, we remove the 64-beam samples in both our training split and validation split. Unlike the nuScenes dataset, the Level 5 dataset does not provide lidar scan line indices. Therefore, we use a clustering

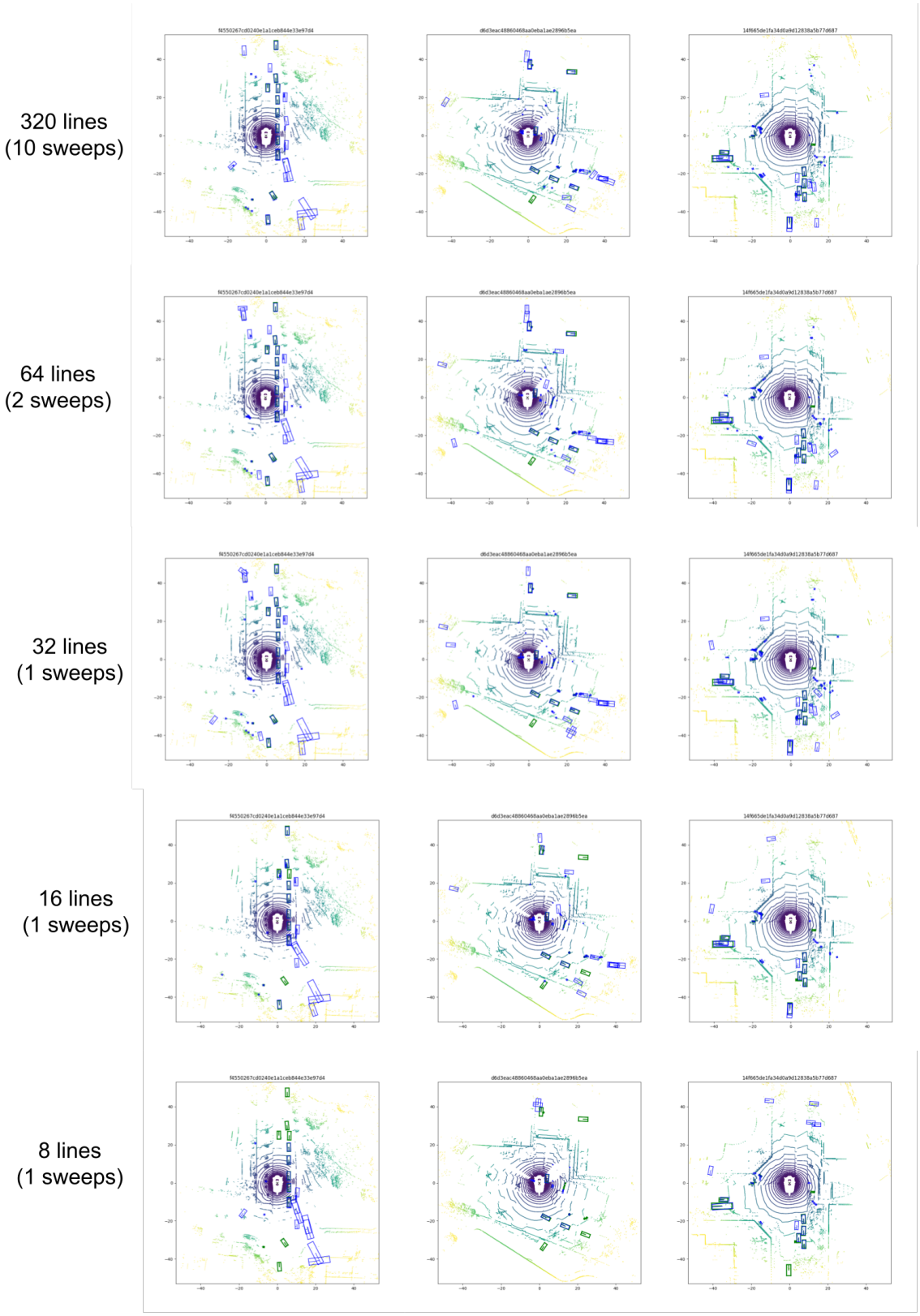

Figure 4.5: Qualitative results on different lidar densities. The green box indicates the ground truth. The blue box indicates the predictions.

	Car mAP@0.5IoU	
	40 scan lines	20 scan lines
PointPillars [4]	76.3	68.6
Ours	79.1	69.6

Table 4.3: Average precision on the Lyft Level 5 validation split.

Distance	Multiview	Image Feature Pooling	Temporal point Pooling	mAP
				34.0
✓				43.5
✓	✓			43.8
✓	✓	✓		51.69
✓	✓	✓	✓	52.6

Table 4.4: Ablation study on the nuScenes validation dataset.

algorithm to assign scan line indices. We then simulate a more sparse 20-beam lidar by skipping lidar scan lines. Also, the Level 5 dataset is unbalanced with the majority of annotations being cars. Therefore, we focus on the mean average precision (mAP) of cars. The results are shown in Table 4.3. Our method achieves 3% and 1% more mAPs than the PointPillar baseline when using a single lidar sweep of 40 scan lines and 20 scan lines, respectively.

4.3.5 Ablation Study

In this section, we study the effect of each component in the proposed algorithm using only the nuScenes dataset due to its broader attributes. Table 4.4 shows the overall results. Unless described otherwise, the training configurations for all the models in the ablation study are the same.

Distance as a feature. Row 1 of Table 4.4 shows a baseline implementation that uses the image features from a single camera to classify lidar points and regress the local co-

ordinates. Row 2 of Table 4.4 shows the detection accuracy of a model that uses both the image features and the distance between the point and the camera. The distance feature significantly improves the detection accuracy. Intuitively, the image features contain information about the object's scale and can be used to predict the distance range of the object. By comparing the input distance, the network can learn to classify the occluded points as negatives.

Multiview features. In Row 3 of Table 4.4, we show the detection accuracy by combining image features from multiple cameras. Compared to Row 2, which randomly picks image features from one camera, combining multiple images has a small improvement in accuracy. We hypothesize that the overlapped camera fields of view allow most of the objects to be fully observed already. Therefore the multi-view improvement is not very significant.

3D pooling Row 4 and Row5 of Table 4.4 shows the effectiveness of 3D pooling. In Row 4, we test the model accuracy with a single lidar sweep. As all the points are from the same timestamp, this test removes the impact of pooling features from asynchronous points and only shows the improvement from pooling points with image features using Equation 4.8. In Row 5, we test the model with ten lidar sweeps. The improvement from Row 5 to Row 4 shows that our proposed algorithm can also utilize dense point cloud features [1].

[1]When a point cloud is denser, complex point cloud backbones can be used to better utilize the point structures

4.4 Summary

In this chapter, we described our camera-lidar fusion method to boost object detection on a point cloud. Our proposed method can utilize a point cloud with low point density, which can be obtained from relatively inexpensive lidars. This, in turn, will enable broader use of lidar in cost-sensitive application domains such as consumer vehicles, warehouse robots and work zones. Our method estimates a lidar point's local object coordinates using multi-view image features and locates the object by registering local object coordinates to lidar coordinates. Our experiments with the nuScenes and the Level 5 datasets showed that our proposed method achieves leading performance when combining lidars with low-cost cameras. In the next chapter, we start to tackle another perspective of overall system cost by reducing the amount of labeled data for training.

Chapter 5

Self-Supervised Learning for Object Detection

5.1 Introduction

Modern perception systems often utilize neural networks and tackle complicated driving scenarios in a data-driven manner. It often utilizes a large amount of labeled data for training. However, such labeled data require manual annotation, which is costly and time-consuming. In many scenarios, the labeled data are not available or are prohibitively expensive to gather, bringing challenges to autonomous driving development. Unlabeled driving data, however, are relatively abundant and low-cost, attracting many research interests.

Many studies have been done on utilizing unlabeled data. Recently, self-supervised learning has achieved promising results in natural language processing [27] and image

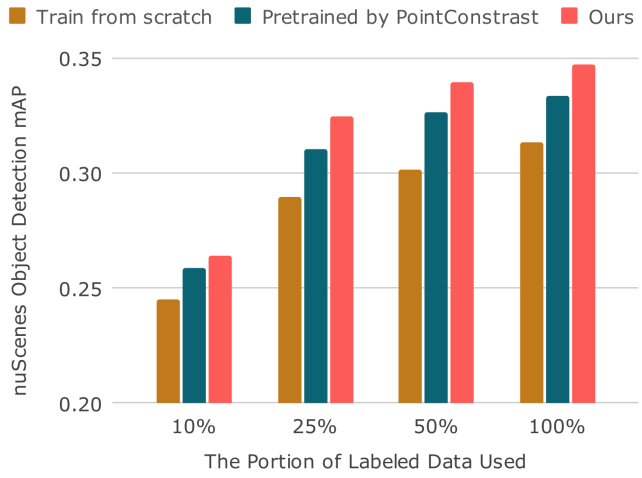

Figure 5.1: Our self-supervised pretraining improves the object detection performance for various amounts of labeled data.

classification [24, 25, 26]. One common paradigm of self-supervised learning is unsupervised pretraining followed by supervised fine-tuning. During the unsupervised pretraining phase, a neural network is trained by pretext tasks using a large amount of unlabeled data. These pretext tasks create their own supervision without manual annotations and allow the neural network to learn proper data representation. After unsupervised pretraining, the neural network is fine-tuned for downstream tasks using a small amount of labeled data.

In the point cloud domain, many studies follow the self-supervised learning approaches for image classification. Pretext tasks such as point cloud assembling [50], orientation estimation [52], and contrastive learning [28] have been used to improve point cloud classification and segmentation. However, few studies have explored outdoor point cloud object detection, a crucial component of autonomous driving. Moreover, recent findings [49, 48] in the image domain show that 2D object detection may not benefit from self-supervised

pretraining designed for classification. Therefore, it remains an open question regarding how self-supervised learning can be effectively utilized for point cloud object detection.

In this chapter, we investigate self-supervised learning for outdoor 3D object detection. We build upon the prevalent contrastive learning method and propose a set of geometric pretext tasks to improve the pretraining performance. We then conduct experiments in the nuScenes autonomous driving dataset with various amounts of labeled data. Our experiments reveal three insights: (1) pretraining with contrastive loss alone improves the average precision (AP) but negatively impacts the object heading accuracy, (2) combining contrastive and geometric pretext tasks benefits both the average precision and heading accuracy, and (3) the improvement by self-supervised pretraining remains even with an increased amount of labeled data and training steps.

In summary, our contributions are:

- We evaluate the prevalent contrastive loss and reveal its limitation on heading accuracy in point cloud 3D object detection.
- We propose a combination of geometric pretext tasks and contrastive loss to improve self-supervised learning for point cloud object detection.
- We conduct extensive experiments for different pretraining methods using various amounts of labeled data from the nuScenes autonomous driving dataset.

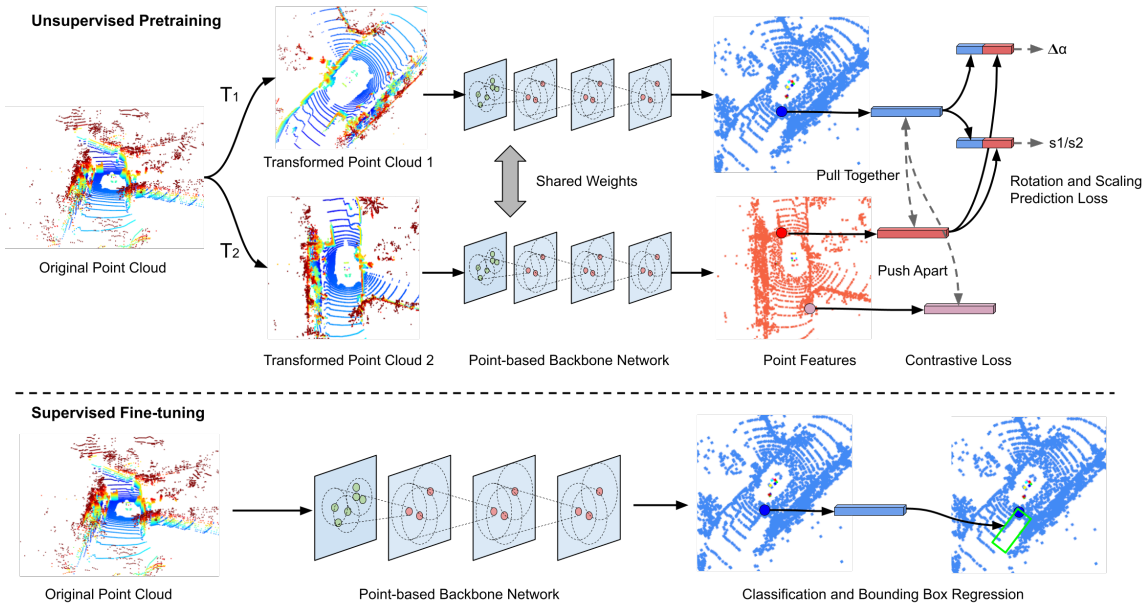

Figure 5.2: The overall architecture of our proposed self-supervised learning. Two sets of random rotation, translation and scaling parameters transform the input point cloud into two different views during the unsupervised pretraining. A contrastive loss pulls the point features from the same point cloud position close to each other while pushing the features from different point positions away from each other. Meanwhile, the point features from the same point cloud position are concatenated and used to predict the difference on scales and observation angles. When fine-tuning for object detection, the point features are used to predict the object type and the bounding box to that the point belongs.

5.2 Our Self-Supervised Learning Method

5.2.1 Overall Architecture

Figure 5.2 shows the overall structure of our self-supervised pretraining. We randomly sample two sets of transformation parameters during the unsupervised pretraining phase, including random rotation around the height axis, random translation, and random scaling. We use these two sets of transformation parameters to augment the input point cloud and create two transformed point clouds. A point-based neural network backbone encodes the point features for each transformed point cloud. We then apply a contrastive loss which pulls together the features of the same point in two transformed point clouds while pushing apart the features of different points. Such contrastive loss supervises the neural network to learn discriminative features that are invariant to transformations. However, the point features must still capture the point cloud geometries to generate accurate 3D bounding boxes. Therefore, we also add two pretext tasks: observation angle difference recognition and relative scaling recognition. As discussed later in the experiments, these two pretext tasks can further benefit the object detection task and avoid degrading box heading accuracy caused by contrastive loss.

After the unsupervised pretraining phase, we fine-tune the backbone neural network for object detection. We use the extracted point features to classify the object category and bounding box that each point belongs to. Overlapping bounding boxes are removed by standard non-maximum suppression.

5.2.2 Point-based Backbone Network

We use a lightweight point-based backbone network similar to PointGNN [33]. The network first samples a set of keypoints by voxel downsampling and aggregates each keypoint's neighborhood points following a PointNet-like [10] structure. Given a keypoint p_i, its aggregated feature vector y_{p_i} is computed as follows:

$$y_{p_i} = \max_{||p_i - p_j||_2^2 < r} (\{f([t_j - t_i, a_j, R_{p_i}(p_j - p_i)])\}) \tag{5.1}$$

where p_j is a lidar point within a distance radius r of p_i, a_j is the laser reflection intensity of p_j, t_j and t_i are the timestamps for p_j and p_i respectively, R_{p_i} is a rotation matrix that removes the yaw angle of p_i, and $f(.)$ is an MLP layer.

The features of keypoints are then iteratively refined by a residue function. In the t^{th} iteration, the features are computed as follows:

$$y_{p_i}^{t+1} = y_{p_i}^{t} + \delta y_{p_i}^{t+1} \tag{5.2}$$

$$\delta y_{p_i}^{t+1} = \max_{||p_i - p_j||_2^2 < r`} (\{f^t([y_{p_j}^t, R_{p_i}(p_j - p_i + g^t(y_{p_i}^t))])\}) \tag{5.3}$$

where p_j is also a keypoint, $y_{p_j}^t$ is p_j's features computed in the previous iterations, and $g^t(.)$ is another MLP layer to compute a coordinate residue.

5.2.3 Contrastive Learning

Contrastive learning aims to pull together the features of the same point in the different transformed point clouds and push apart the features of different points. Such a learning

goal enables the network to extract robust discriminative features invariant to transformations. Contrastive learning on point cloud [28] shows promising results for point cloud classification and segmentation. We use the *PointInfoNCE* loss proposed in PointContrast [28]:

$$L_c = -\Sigma_i \log \frac{\exp(y^1_{p_i} \cdot y^2_{p_i}/\tau)}{\Sigma_{i \neq j} \exp(y^1_{p_i} \cdot y^2_{p_j}/\tau)} \tag{5.4}$$

where $y^1_{p_i}$ and $y^2_{p_i}$ are point features extracted from two transformed point clouds and τ is a temperature constant.

5.2.4 Geometric Prediction Tasks

The *PointInfoNCE* loss supervises the network to learn features invariant to transformations such as point cloud rotation and scaling. Ideally, the neural network should output the same feature vectors regardless of point cloud transformations. In other words, the geometric information such as point cloud rotation and scale may diminish from the features. However, one of the critical tasks for an object detection algorithm is to locate the object and generate accurate bounding boxes for use in path planning. The neural network must keep the geometric attributes of the point cloud to generate the correct bounding boxes. For example, the features from the neural network need to contain point cloud rotation information so that the bounding box heading can be correctly computed. Therefore, we propose a set of geometric prediction tasks to pretrain the network jointly with the contrastive loss.

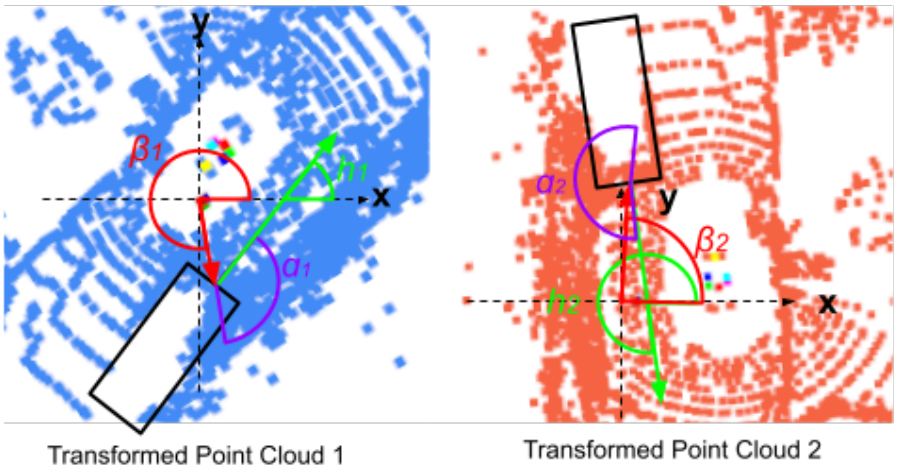

Figure 5.3: Our detection algorithm predicts the object observation angle α, which is the angle difference between object heading h and the yaw angle of the ray from the origin to the point β. Note that the two point clouds have different orientations and origins because of the different transformations that we apply.

5.2.5 Predict Observation Angle Difference

As shown in Figure 5.3, our object detection algorithm predicts the observation angle for the object heading. We define the observation angle as the yaw angle difference between the object heading h and the ray connecting the origin (the lidar center when no random translation) and a lidar point β:

$$\alpha = h - \beta \tag{5.5}$$

In the unsupervised setting, we do not know the object heading h. Thus, we cannot get an observation angle to supervise the network directly. However, we can compute the difference between observation angles in two of the transformed point clouds:

$$\Delta\alpha = (h_1 - \beta_1) - (h_2 - \beta_2) = (h_1 - h2) - (\beta_1 - \beta_2) \tag{5.6}$$

where $h_1 - h2$ equals the difference of two random rotations we apply to the point cloud and $(\beta_1 - \beta_2)$ can be computed using transformed point coordinates. Therefore, $\Delta\alpha$ can be computed without labels.

We then add a header for the network and supervise it to recognize the observation angle difference. We concatenate the feature vectors of the same point in two transformed point clouds and then apply a symmetric loss function for this pretext task:

$$L_r = Huber(MLP_r([y^1_{p_i}, y^2_{p_i}]), \Delta\alpha_{p_i}) + Huber(MLP_r([y^2_{p_i}, y^1_{p_i}]), -\Delta\alpha_{p_i}) \quad (5.7)$$

5.2.6 Predict Relative Scales

The ideal bounding boxes fit tightly to the object contour and adapt their sizes to the object's point cloud. Similar to the observation angle difference prediction, we can compute the relative scales between two transformed point clouds without knowing the actual object dimensions. Given object dimensions $dims$ and two random scaling factors s_1 and s_2, the relative object scale in two transformed point clouds are:

$$\Delta s = \frac{s_1 dims}{s_2 dims} = \frac{s_1}{s_2} \quad (5.8)$$

Therefore, we also concatenate feature vectors of the same point in two transformed point clouds and add a header for scale prediction. We define a loss function for this pretext task as:

$$L_s = Huber(MLP_s([y^1_{p_i}, y^2_{p_i}]), \log(\frac{s_1}{s_2})) + Huber(MLP_s([y^2_{p_i}, y^1_{p_i}]), \log(\frac{s_2}{s_1})) \quad (5.9)$$

The total loss function of the network in pretraining is then:

$$L = L_c + L_r + L_s + L_{reg} \tag{5.10}$$

where L_{reg} is the network's regularization loss.

5.2.7 Dataset and Training Hyperparameter

We use the nuScenes [20] autonomous driving dataset in our experiments. The nuScenes dataset contains sensor data from different driving scenes. Each scene contains a sequence of samples. In total, the nuScenes dataset provides a *train* split of 28130 samples from 700 scenes and a *val* split of 6019 samples from 150 scenes. Each sample contains records from a suite of sensors, including a 32-line lidar, radar, IMU, and six cameras. For our point cloud recognition experiments, we focus on the lidar data.

To study the impact of self-supervised learning with different amounts of labeled data, we create a series of random splits which contains $[35, 70, 175, 350]$ scenes from the *train* split. They correspond to $[5\%, 10\%, 25\%, 50\%]$ of the total *train* scenes and have $[1413, 2817, 7029, 14047]$ samples, respectively. In our self-supervised representation learning, we treat all the samples in *train* as unlabeled data for training. For the downstream 3D object detection task, we use one of the splits as the labeled data to train the network. We then use the completely labeled *val* split for evaluations.

For all the neural network training in our experiments, we use the Adam [74] optimizer and cosine learning rate decay with a warmup [79] of 8K steps. We use an initial learning rate of 10^{-3} for all the training. For the unsupervised pretraining, we use 1M training

Labels	Method	car	truck	bus	trailer	cons.	ped.	moto.	bicycle	cone	barrier
10 %	Train from scratch	0.6475	0.2012	0.2465	**0.1282**	**0.0324**	0.5772	0.121	0.0029	0.2171	0.2757
	Pretrained by PointConstrast	0.6407	0.201	0.267	0.0961	0.0189	0.6498	**0.1664**	**0.0175**	**0.256**	0.2722
	Ours	**0.6547**	**0.2127**	**0.284**	0.1274	0.0308	**0.651**	0.1501	0.0067	0.2386	**0.2851**
25 %	Train from scratch	0.6684	0.2225	0.3462	0.1349	0.0625	0.5898	0.2458	0.015	0.3097	0.3002
	Pretrained by PointConstrast	0.6733	0.2305	0.357	0.1441	0.0613	0.6889	0.2781	0.0209	0.3377	**0.3078**
	Ours	**0.6847**	**0.2663**	**0.3815**	**0.1624**	**0.0648**	**0.7026**	**0.2917**	**0.0591**	**0.336**	0.2966
50 %	Train from scratch	0.6822	0.2389	0.3365	0.1687	0.0581	0.6409	0.2283	0.0462	0.3058	0.309
	Pretrained by PointConstrast	0.6746	0.2367	0.3618	0.1576	0.0673	0.7173	0.2824	0.0585	0.3749	**0.3307**
	Ours	**0.6886**	**0.2738**	**0.3726**	**0.1781**	**0.0751**	**0.7231**	**0.2839**	**0.0832**	**0.3875**	0.3292
100%	Train from scratch	0.6846	0.2457	0.3681	0.16	0.0839	0.6518	0.2305	0.042	0.3607	0.3028
	Pretrained by PointConstrast	0.6789	0.2451	**0.3966**	0.1495	0.0838	0.7162	0.278	0.0712	0.4016	0.3109
	Ours	**0.6934**	**0.2791**	0.3782	**0.179**	**0.0852**	**0.732**	**0.3031**	**0.0976**	**0.4037**	**0.3178**

Table 5.1: The Mean Average Precision (mAP) of models trained by different portions of labeled. The results are conducted on the validation set of the nuScenes dataset.

steps. For the fine-tuning, we use 32K training steps unless we explicitly change them for controlled experiments. To improve training stability, we apply a gradient clip [80] when the total gradient norm exceeds 1.0. We use a batch size of 2 for all training.

5.3 Experimental Results

5.3.1 The Effects of Different Pretext Tasks

We conduct experiments on object detection with various amounts of training labels. We compare training performance from scratch, pretraining using contrastive loss and pretraining with a combination of contrastive loss and geometric pretext tasks. Figure 5.1 shows the mean Average Precision (mAP) (the higher, the better), and Figure 5.4 shows the mean Average Orientation Error (mAOE) for the object heading estimation, where lower the value, better the performance. Figure 5.7 shows the qualitative results of object detection that are fine-tuned by 10% of training data from different pretraining methods. In Table 5.1, we show the test results (mAP) for each object type. The results are con-

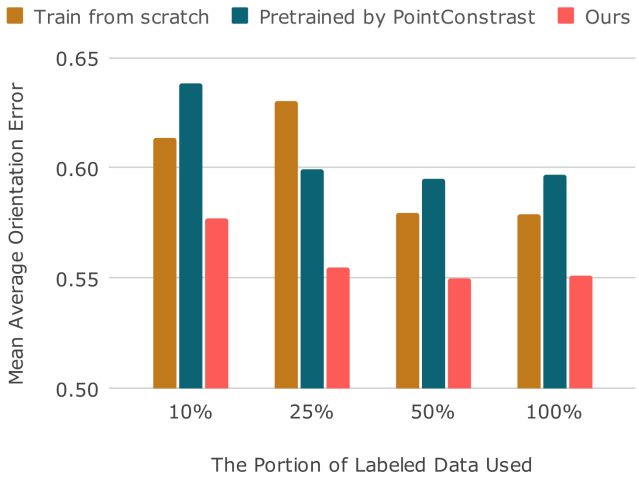

Figure 5.4: The mean Average Orientation Error (mAOE) of model initialized by random weights, pretraining with contrastive loss alone and our combination of contrastive loss and geometric pretext tasks. The lower, the better.

ducted using the nuScenes validation split. In general, it can be observed that our pretraining method achieves better accuracy than training from scratch and pretraining using contrastive loss.

5.3.2 Discussion

Pretraining benefits average precision. As can be observed in Figure 5.1, both pretraining with contrastive loss and our proposed tasks help the detection algorithm to achieve a higher mAP than training from scratch. The benefits remain when we increase the amount of labeled training data.

Pretraining with contrastive loss alone may hurt bounding box orientation estimation. Although pretraining with contrastive loss helps with general average precision,

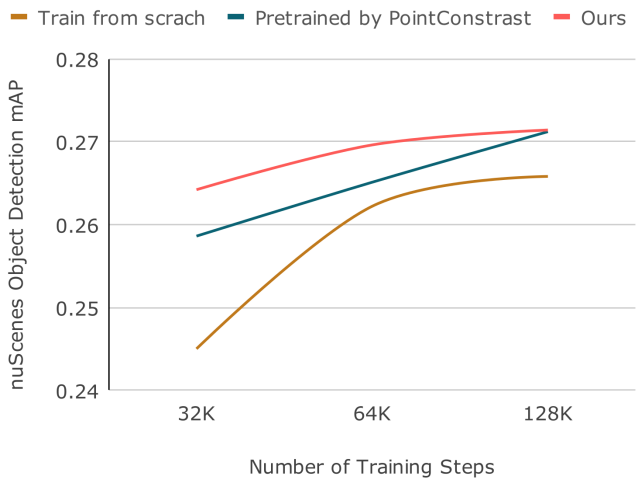

Figure 5.5: The mean Average Precision of models when fine-tuned with different numbers of training steps. The higher, the better.

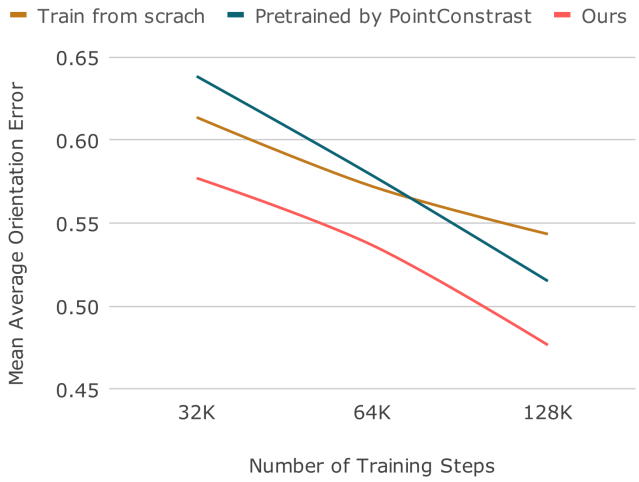

Figure 5.6: The mean Average Orientation Error (mAOE) of models when fine-tuned with different numbers of training steps. The lower, the better.

it hurts the bounding box orientation estimation. Figure 5.4 shows that when pretraining with contrastive loss alone, the heading error can be larger than that of trained from scratch. The transformation invariant features that contrastive loss learned may miss essential geometric cues. These invariant features may be less sensitive to the object orientation change. For autonomous driving, where object localization accuracy is critical, pretraining using contrastive loss alone is hence not a good strategy.

Our proposed pretraining method benefits both average precision and heading estimation. Our proposed combination of contrastive loss and geometric tasks achieves higher average precision and lower orientation errors than contrastive-only pretraining and training from scratch. Noticeably, our pretext tasks of predicting observation angle and relative scale overcome the degradation of object heading estimation by contrastive loss.

5.3.3 The Effects of Training Steps

To study the effects of pretraining with different training steps, we perform fine-tuning with $2\times$ (64K) and $4\times$ (128K) steps using 10% of the labeled data. The results are shown in Figure 5.5 and 5.6. We can observe that, with more training steps, the performance of object detection generally increases. Our proposed pretraining method achieves similar mAP as the contrastive learning method when we run $4\times$ training steps. However, our gain on object orientation estimation remains significant in both $2\times$ and $4\times$ training steps.

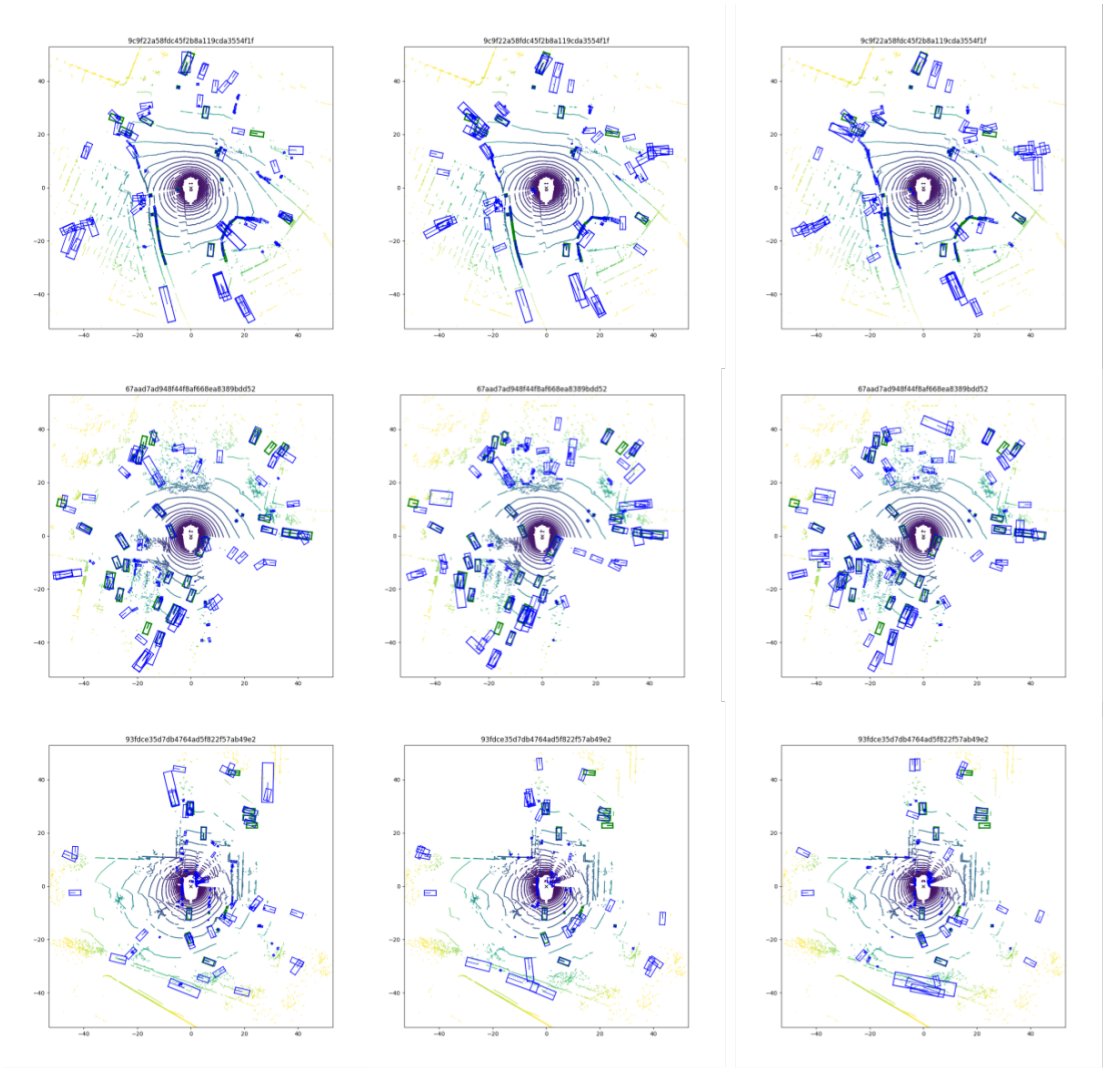

Figure 5.7: Qualitative results of object detection using 10% of training data.

5.4 Summary

In this chapter, we studied the problem of utilizing unlabeled data to improve point cloud object detection for use in autonomous vehicles. We used the realistic nuScenes autonomous driving dataset to conduct experiments on self-supervised learning methods. The prevalent contrastive loss for object detection from a point cloud has core limitations. We proposed a combination of contrastive loss and a set of geometric recognition tasks, which prevents the performance degradation of contrastive loss. Our experiments using various amounts of labeled data and training steps showed that our proposed method can utilize unlabeled data to improve the mean average precision and object heading estimation in object detection. Our results demonstrate the potential to use relatively abundant and cheap unlabeled data to improve the perception system for autonomous driving while keeping the overall system cost relatively low. In the next chapter, we look at important real-world driving scenarios and describe efficient solutions to address perception challenges in work zones.

Chapter 6

Challenging Scenarios: Work Zone Detection

6.1 Introduction

Autonomous vehicles have enormous potential to improve transportation efficiency and driving safety. One fundamental task of an autonomous vehicle is to understand the drivable area around it. An autonomous vehicle commonly has access to a map database that contains road geometries and lane information. However, road maintenance and repair needs often create work zones which the map database may not record accurately in advance. Such a work zone can occur quickly due to emergencies and can also change dynamically as road work proceeds. Therefore, an autonomous vehicle cannot solely rely on a map database, and it needs to detect work zones in real-time. In this chapter, we describe our solutions to detect work zones.

Many studies [55, 53, 56] focus on detecting the presence of a work zone without inferring its fine-grained contour. The awareness of a work zone enables the vehicle to warn drivers of dangerous driving conditions. However, for a vehicle to navigate through the work zone fully autonomously, the work zone boundary information is indispensable. Detection of a work zone boundary, however, is non-trivial. The boundary often comprises loosely-placed items such as traffic cones, barrels, channelizers, barriers and traffic signs. These items can be sparse and may not form a tight enclosure of the work zone. If agnostic to the work zone, a drivable area detection algorithm may find a pathway in the space between these items, leading to a dangerous route into the work zone. It is, therefore, important for a detection algorithm to couple the recognized work zone with its boundaries.

The task of work zone detection is relatively unexplored. One challenge is the lack of proper data. Conventional driving datasets [53] are mainly based on camera videos without capturing the multi-modality sensor configuration in a modern autonomous vehicle. Also, these datasets lack the ground truth geometries of captured work zones, leading to insufficient evaluation. Recently, many additional autonomous driving datasets [20, 19, 18] have been released. Yet, they too do not provide annotations for work zones. Another challenge of studying the work zone detection problem is the lack of appropriate definitions and evaluation methods. There has also been little discussion on the algorithm's output format.

In this chapter, we first formulate the work zone detection problem from a practical perspective. We define a standard output format that is suitable for a multi-modality sensor configuration. We then provide work zone annotations for the autonomous driving dataset

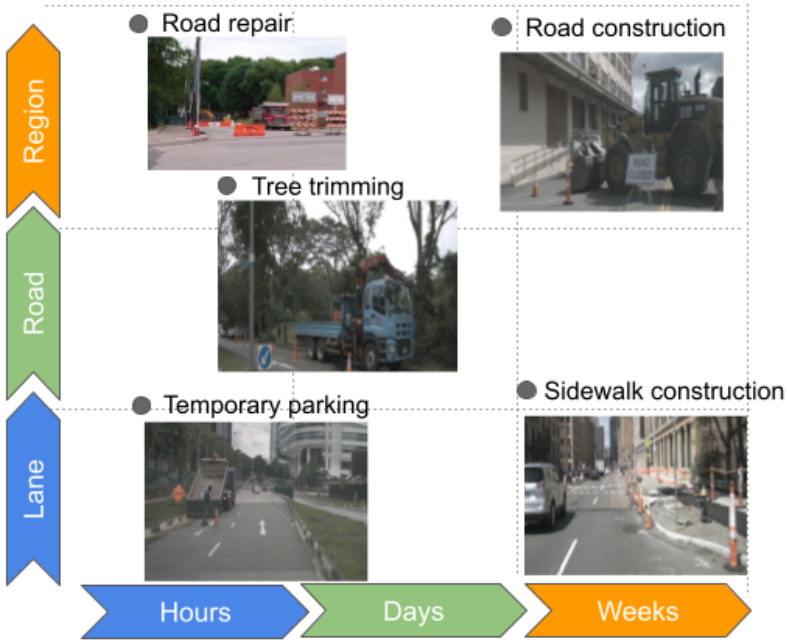

Figure 6.1: Examples of diverse work zones. Different work zones have varied spatial and temporal scales, and they may not be updated in a map database.

nuScenes [20], enabling a realistic evaluation. Next, we propose a detection pipeline and baseline implementations using images, a lidar point cloud and their fusion.

In summary, our contributions are:

- We propose a definition of the work zone detection problem and suggest practical evaluation metrics.
- We add a set of work zone annotations to the popular autonomous driving dataset, nuScenes.
- We propose a work zone detection pipeline using multi-modality sensor configurations and three baselines using images, a lidar point cloud and their fusion.

6.2 Our Work Zone Detection

In this section, we first provide a taxonomy on the different aspects of a work zone. We then formulate the work zone detection problem, which takes into consideration multi-modality sensor configurations. Finally, we propose a set of metrics to evaluate the performance of a work zone detection algorithm.

6.2.1 Work Zone Taxonomy

A *work zone* is an area that is temporarily closed for regular vehicular traffic due to roadwork. For direct use by an autonomous vehicle, as shown in Figure 6.2, we categorize a work zone by its information source, its semantics and its environmental attributes. Typically, a work zone's information comes from the vehicle's onboard perception or from

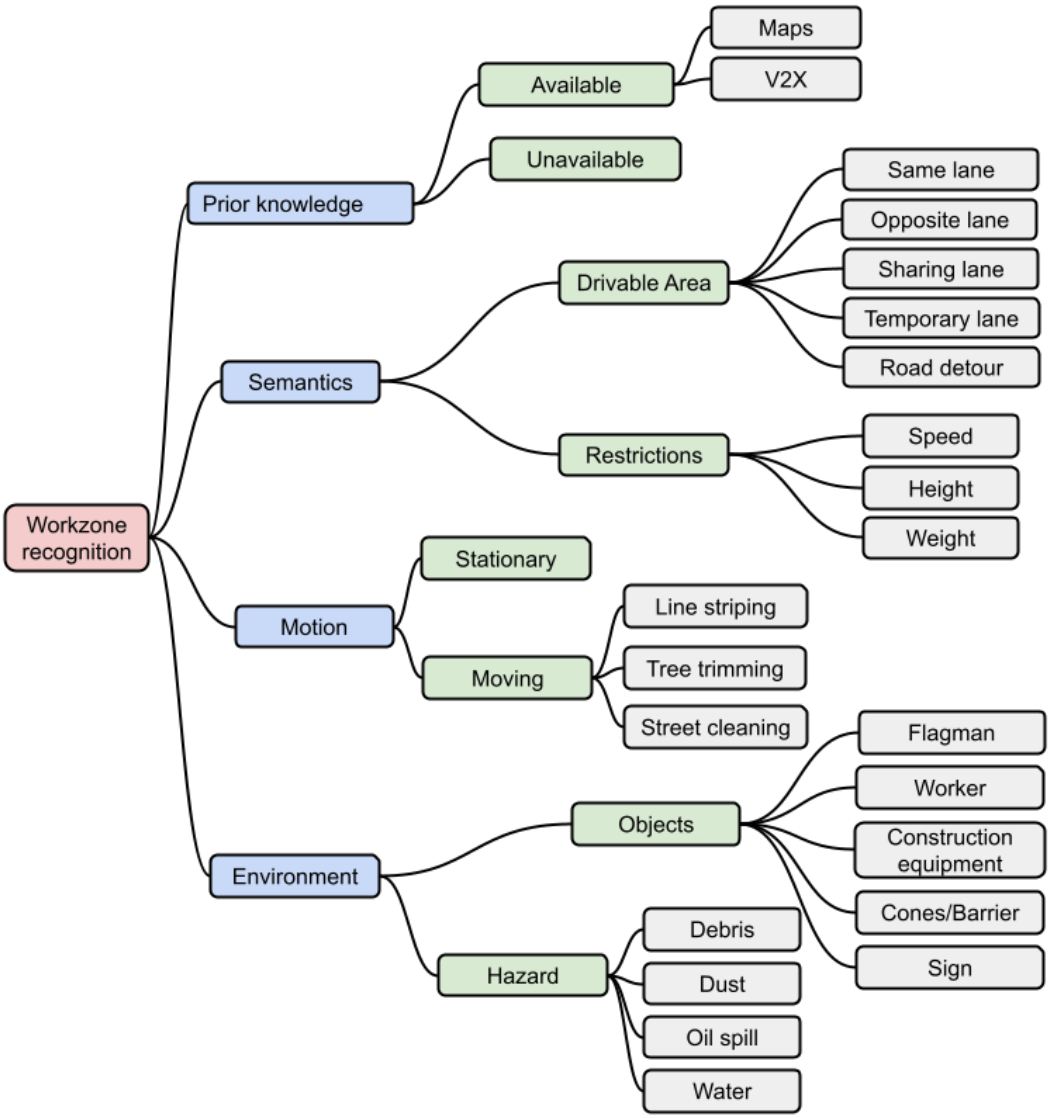

Figure 6.2: A taxonomy of work zones.

prior knowledge. Prior knowledge is the information known about a work zone before a vehicle encounters it. As most modern autonomous vehicles have access to a map database, they may be able to query work zone information from the map. The map may record the work zone information, especially when a work zone lasts for an extended period. Another source for the prior knowledge of a work zone is the vehicle's connection to other vehicles (V2V) or the infrastructure (V2I). These connections can transmit work zone information in real time. However, there is no guarantee that the prior knowledge exists or is correct. If a work zone appears on short notice or has changed recently, the prior knowledge can be misleading. Therefore, the vehicle's onboard perception needs to correct any mistakes and detect the work zone without any priors.

A work zone has varied semantic implications to traffic. First, a work zone changes the original drivable area. Depending on the area that a work zone occupies, traffic can go through the same lanes alongside the work zone, traverse the adjacent lanes, or even share or borrow lanes in the opposite driving direction. In the extreme case where a work zone occupies the entire road, a temporary lane via a road shoulder or a detour to another road may be needed. Secondly, a work zone may also impose additional restrictions on permitted vehicles based on physical dimensions and speeds. For instance, a work zone may have a potentially reduced speed limit. An autonomous vehicle must recognize a work zone's semantics and intelligently plan its driving path. Although the behavior and path planning in a work zone are also challenging and critical, the accurate detection of a work zone serves as an essential prerequisite and is the focus of this chapter.

Work zones have different durations. Although many work zones exist for a relatively long period and can be perceived as stationary by vehicles, some road work creates a

moving work zone. For example, when construction workers are stripping or repainting the lane markers, they often move the work zone along as they progress. Similarly, other road activities like street cleaning and tree trimming may also create a moving work zone. In general, a moving work zone has a more demanding requirement on work zone detection than a stationary one. An autonomous vehicle also needs to be aware of a work zone's moving speed and plan its path safely.

A work zone may also bring different attributes to the driving environment. Specifically, a work zone may bring a new set of objects onto the road, such as flagmen, workers and construction equipment. An autonomous vehicle needs to be aware of these objects and understand the work zone-related behaviors. For example, workers may enter or exit a work zone on a highway. Meanwhile, the work zone may also cause hazardous driving conditions, for example, water and oil spills. Sometimes, accidents or construction activities may leave debris on the road or cause dust which limits visibility. Recognizing all work-zone-related hazards is out of the current scope of our work, and we only focus on detecting the work zone in this chapter.

6.2.2 Problem Formulation

We define the work zone detection task as finding polygonal representations of work zones in the vehicle's Bird's Eye View (BEV) plane. This representation is useful for multiple reasons. First, a polygon is a compact format to approximate any closed contour of the work zone and it is directly applicable to any existing path-planning algorithms that navigate around polygonal obstacles. In general, a work zone can be presented by a single polygon. In the case of multiple disjoint work zones, multiple polygons are needed. Sec-

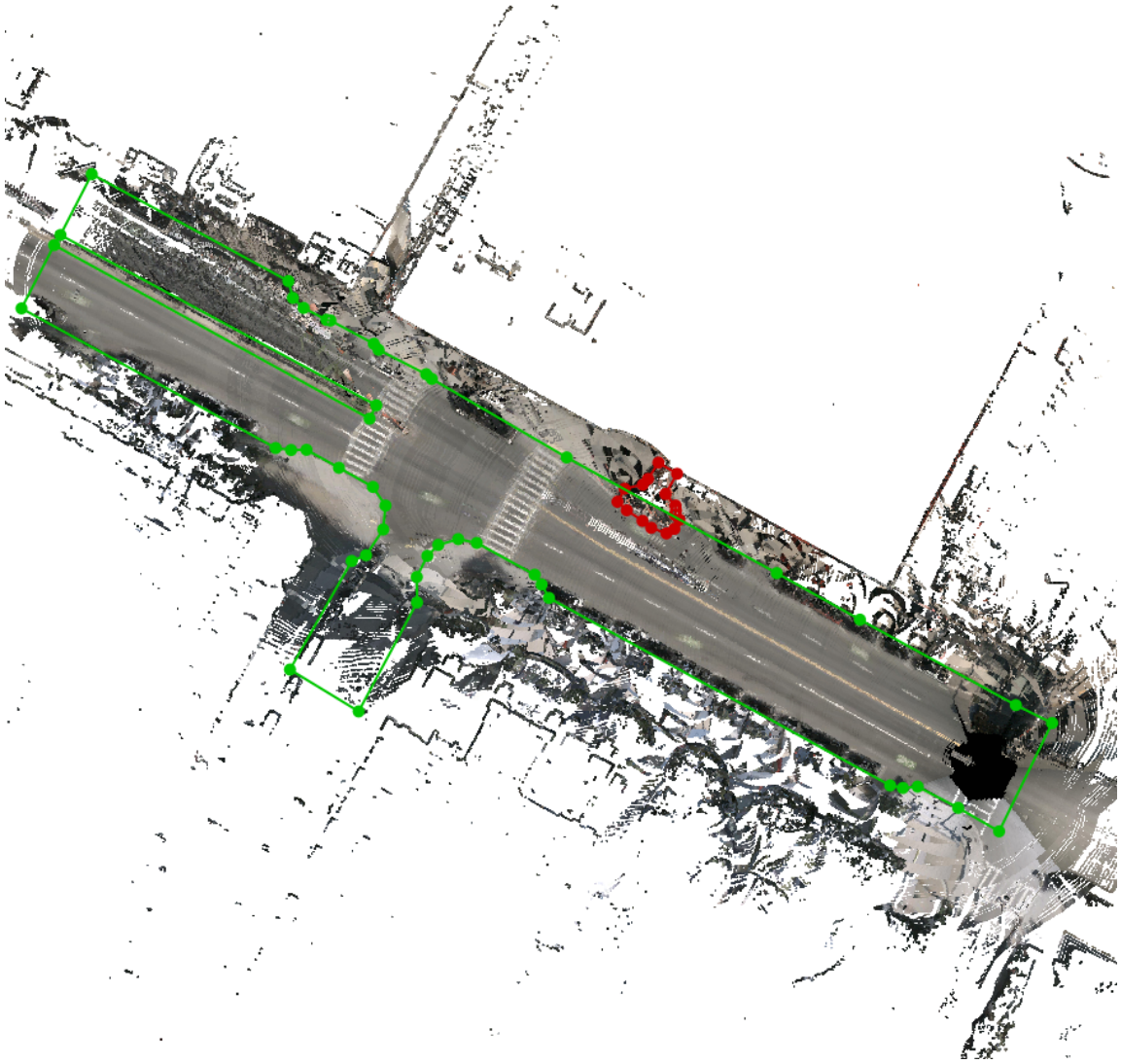

Figure 6.3: An annotation example. The red polygon is the ground truth for a work zone. The green polygon is the road region. In our evaluation of work zone detection algorithms, we focus on the overlap between the work zone and the road region.

ondly, the BEV coordinates are agnostic to the sensor configurations and enable comparison between algorithms with different modalities. Formally, we define the expected output of the detection algorithm as a list of polygons where each polygon is a sequence of points:

$$Det = [P_0, P_1, ...] \tag{6.1}$$

$$P = [p_0, p_1, ..., p_{n-1}, p_n] \tag{6.2}$$

where, $p_i = (x_i, y_i)$ is a point on the BEV plane and $p_0 \equiv p_n$. For any i and j, the line segment (p_i, p_{i+1}) does not intersect with the line segment (p_j, p_{j+1}). We define the BEV plane as a 2D plane parallel to the flat ground, centered at the position of the autonomous vehicle. The input to the detection algorithm depends on the sensor configuration and it can be from a single sensor, multiple sensors, or sensor types.

6.2.3 Evaluation Metric

The ideal detection result of a work zone should be a polygon that encloses the entire work zone area. However, for autonomous driving purposes, we pay more attention to the drivable area. As shown in Figure 6.3, we annotate both the work zone region and the road region. The goal of our evaluation metrics is to estimate the work zone detection accuracy within only the road region. The road region also comprises polygons, and we take the intersection between the road region and the detection results (or ground truth annotations) before evaluation. We propose three types of metrics for evaluation: *IoU*, *Recall and Precision* and *maxDist*.

Intersection over Union at a Distance (IoU@d). IoU measures the overlap between two

regions. Given two regions S_A and S_B, their IoU is calculated as:

$$IoU(S_A, S_B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|} \tag{6.3}$$

where $|.|$ represents the area of the region. For two completely overlapping regions, the IoU score is 1. For two completely disjoint regions, the IoU score is 0. The IoU is a measure of the alignment between the detected region and the ground truth. However, it does not directly consider the distance factor. For autonomous driving, the detection accuracy at different distances has varied impacts on driving safety. First, a long detection range allows sufficient time and space for safe motion planning. Secondly, detection errors within a short range may be catastrophic. To evaluate the work zone detection accuracy at different distances, we measure the IoU at multiple distance thresholds. Given a distance threshold d_{th}, we draw a circle S_r of radius d_{th} in the BEV plane around the vehicle center. We then calculate the IoU score between the detected work zone S_{det} and ground truth S_{gt} as follows:

$$IoU@d_{th}(S_{det}, S_{gt}) = \frac{|(S_{det} \cap S_{gt}) \cap S_r|}{|(S_{det} \cup S_{gt}) \cap S_r|} \tag{6.4}$$

We use a set of thresholds $d_{th} = \{10m, 20m, 50m\}$ in our experiments [1]. When there is no work zone area within the distance threshold, the IoU is invalid. We take the average of valid IoU values as the final score.

Recall and Precision. The IoU score measures the overlap between the detection and its ground truth. However, it does not directly reflect whether an algorithm detects the work zone or not. Also, when there is no work zone area, the IoU does not reflect any

[1] On highways, larger distances will be necessary.

false positive detections. To measure the detection rate and detection accuracy, we use the well-known metrics of recall and precision. We first associate the detected work zone regions and ground truth annotations by their IoU. If the IoU between a detected work zone polygon and a ground truth polygon reaches (say) 0.1, we consider that the ground truth work zone has been detected. If a work zone polygon does not match any ground truth polygons, we consider this detection to be false. We compute the recall as the ratio between the number of detected work zones D_T and the number of total ground truth work zones D_{Gt}. Similarly, we compute the detection precision as the ratio between the number of correctly detected work zones D_T and the number of work zones the algorithm outputs including the false detections D_F.

$$Recall = \frac{D_T}{D_{Gt}} \tag{6.5}$$

$$Precision = \frac{D_T}{D_T + D_F} \tag{6.6}$$

Note that the recall metric is valid only when the ground truth work zone exists. Similarly, precision is valid only when the algorithm generates detections. We filter out all the other invalid cases and take the average value of valid recall and precision values.

Maximum Detection Distance (*maxDist*). For safety, we also want to know the maximum distance at which an algorithm detects a work zone. We record the detection results when the vehicle drives around the same work zone and compute the distance between the detected work zone and the vehicle. After completing a driving sequence, we take the maximum detection distance as the *maxDist* value for this sequence. We then take the average value of *maxDist* across all testing sequences.

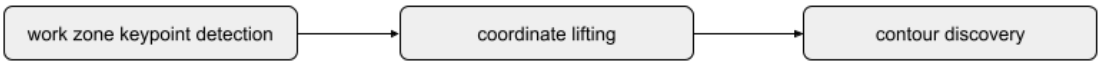

Figure 6.4: WZDetector: a general pipeline for work zone detection.

6.2.4 Our Work Zone Detection Pipeline

We design a general pipeline for work zone detection that allows inputs from different sensing modalities. As shown in Figure 6.4, our proposed work zone detection pipeline consists of three stages: a work zone keypoints detection module, a coordinate lifting module and a contour discovery module. A work zone is morphable without a fixed shape, raising challenges to detecting many work zone in a straightforward manner. However, some features that characterize a work zone remain invariant. For example, work zone objects such as traffic cones, barrels, channelizers, poles or barriers are present in most work zones. The first step of our pipeline is to find some keypoints from those common objects that indicate the existence of a work zone. The keypoints detection module outputs the keypoint position in the sensor domain that may not be directly usable by the motion planner. Therefore, the coordinate lifting module converts the keypoints from the sensor domain to the vehicle's BEV plane, regardless of the sensor modality. Finally, the contour discovery module infers a work zone area and finds its contour using the keypoints in the BEV plane. Our baseline implementations use three different sensor modalities to detect keypoints and lift them to the BEV plane. Once the keypoints are in the BEV plane, we use the same contour discovery module to generate the work zone polygons.

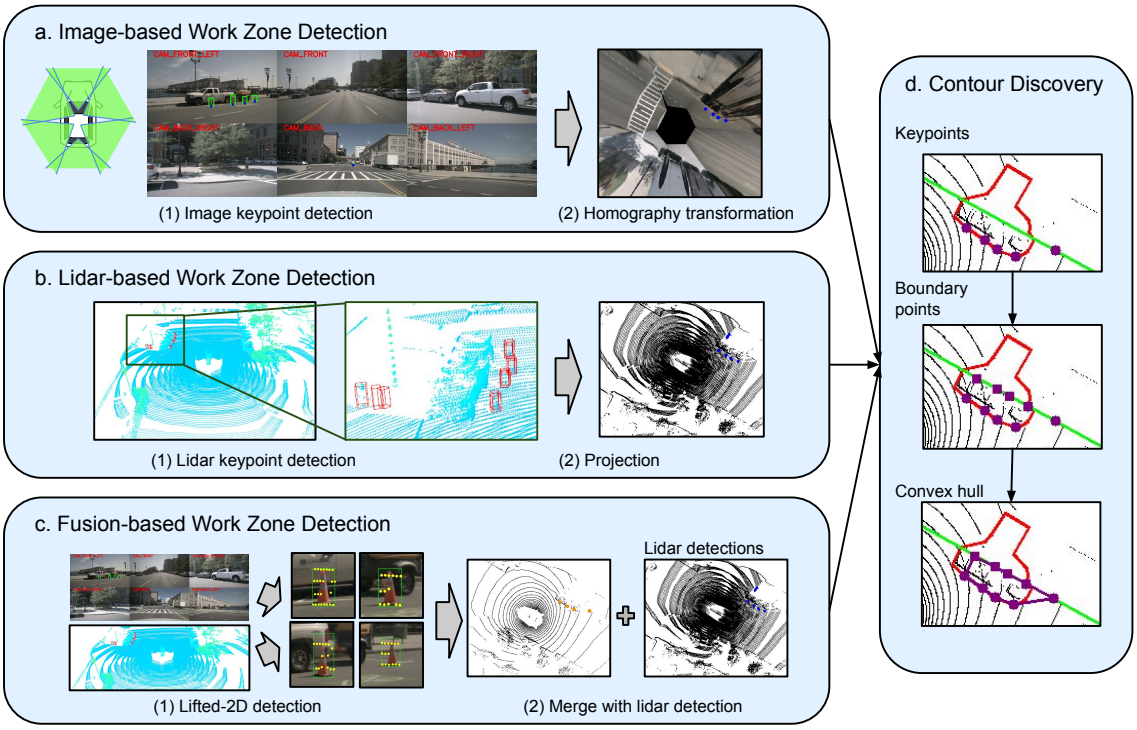

Figure 6.5: Architectures of three baseline implementations using images, a point cloud and a combination of both, respectively.

6.3 Implementation

Cameras and lidars are primary sensors in an autonomous vehicle. This section describes our three baseline implementations using camera images, a lidar point cloud, and a combination of them.

6.3.1 Image-based Work Zone Detection

Image-based work zone detection uses only RGB cameras. As shown in Figure6.5a(1), we first detect work zone objects from all the available camera images. We trained a neural

network model, EfficientDet-D0[81], to detect traffic cones (including barrels and poles) and barriers. The network uses the 512×512 input images and reaches 0.33 mAP on the nuScenes validation set. We trained the network from a checkpoint that is pre-trained in the MS-COCO dataset [82] using the nuScenes training set. The training lasts for 200K steps with a batch size of 16. After object detection, we select the bottom middle of each bounding box as the keypoint and lift its coordinates from the image frame to the vehicle's BEV plane.

We use the classic homography transformation to project the image plane to the BEV plane which is also known as Inverse Perspective Mapping [83]. Although deep-learning-based BEV generation has made remarkable progress, IPM is still widely used and serves as a classic baseline. Assuming that the local road surface is relatively flat and it approximates the BEV plane, we can convert an image point (u_x, u_y) on the road to its coordinates in the BEV plane (x, y) as follows:

$$[x, y, 1]^T = h^{-1}[u_x, u_y, 1]^T \tag{6.7}$$

$$h = k[r_{cam}(0)^T, r_{cam}(1)^T, -t_{cam}^T t_{cam}] \tag{6.8}$$

where, r_{cam} and t_{cam} are the camera rotation matrix and translation vector respectively in the BEV frame. We use the homography transformation to project the keypoints from all available cameras to the BEV plane as shown in Figure6.5a(2).

We then find the work zone area by clustering the keypoints and find the work zone contour. We first run a density-based clustering algorithm DBSCAN [84] to separate keypoints into point groups $\{\mathscr{G}_0, ..., \mathscr{G}_{k-1}\}$, where each group represents a work zone region. A

Algorithm 2: Work Zone Contour Generation

Input: $\mathscr{S} = \{p_0, ..., p_{n-1}\}$, r_p, r_b, $\mathscr{R}$
$\mathscr{S}$ is a detected keypoint set of size n.
r_p is a density threshold to cluster keypoints.
r_b is a distance threshold to search road boundaries.
$\mathscr{R}$ is a set of polygons representing the road map.

$\mathscr{D} \leftarrow \{\}$ $\{\mathscr{G}_0, ..., \mathscr{G}_{k-1}\} \leftarrow$ DBSCAN($\mathscr{S}$, r_p)
for $\mathscr{G}_i$ *in* $\{\mathscr{G}_0, ..., \mathscr{G}_{k-1}\}$ **do**
    $\mathscr{Z}_i \leftarrow \{\}$
    for p_j *in* $\mathscr{G}_i$ **do**
        $\mathscr{Z}_i \leftarrow \mathscr{Z}_i \cup p_j$
        if p_j *in* $\mathscr{R}$ **then**
            $n_j \leftarrow$ nearest(p_j, $\mathscr{R}.boundary$)
            $d \leftarrow |p_j - n_j|_2^2$
            if $d > r_b$ **then**
                $\mathscr{Z}_i \leftarrow \mathscr{Z}_i \cup n_j$
            end
        end
    end
    $\mathscr{P} \leftarrow$ convex_hull($\mathscr{Z}_i$)
    $\mathscr{D} \leftarrow \mathscr{D} \cup \mathscr{P}$
end
return $\mathscr{D}$

work zone may not be entirely observable from the vehicle's perspective and the detected keypoints may only form a partial boundary for the work zone. We close the boundary by searching for the nearest point on the road boundary to a keypoint and add it to the keypoint group $\mathscr{G}$ if its distance to the keypoint is less than a threshold r_b. We then use the convex hull that covers the entire keypoint group and outputs the polygon contour of the convex hull. Algorithm 2 shows the detailed flow.

6.3.2 Lidar-based Work Zone Detection

Unlike the image-based approach, the lidar-based approach directly works on a 3D point cloud from the lidar, allowing object detection in the 3D domain. We use the state-of-the-art lidar object detection algorithm, CenterPoint [76], to detect traffic objects. The detection model is trained on the nuScenes training set and reaches 0.57 mAP in the validation set using ten lidar sweeps. We take the center positions of the detected traffic objects as the keypoints. Because these keypoints are in 3D coordinates already, we project them to the vehicle's BEV plane directly. We generate the work zone contour using the same method as in Algorithm 2.

6.3.3 Fusion-based Work Zone Detection

The lidar-based work zone detection approach relies on the point cloud. However, the point cloud is naturally sparse in regions far from the autonomous vehicle. When the point cloud is sparse, the lidar object detection algorithm's accuracy decreases significantly, limiting the work zone detection range. An image, on the contrary, is dense by nature. An image-based object detector may therefore be able to find tiny objects that are far away from the camera. However, as the image lacks depth information, the lifting from the image's coordinates to 3D is error-prone. Therefore, the work zone contour may not match the ground truth. Our fusion approach, therefore, aims to take advantage of both camera and lidar sensors. Here, we describe our implementation that combines camera images and a lidar point cloud to increase the work zone detection range while preserving the detection accuracy.

We now lift the 2D detections from image to 3D using a point cloud instead of the

homography transformation in Section 6.3.1. As shown in Figure 6.5(c), we project a point cloud to the image and select the points that lay in the bounding boxes of the detected objects. Based on the assumption that the points within a bounding box are likely to be on the object, we use these points to infer the object's 3D coordinates. We take the median position of the lidar points within a bounding box as a keypoint from the object. Unlike the lidar-based detection algorithm, we can find a keypoint with a single lidar point on the object and thus increase the detection range. We name these detections as *lifted-2D detections*. The lifted-2D detections also induce noisy keypoints as the lidar points within a bounding box may not be on the object. For example, the lidar points can be from the background of the object. To take advantage of high-accuracy lidar object detection in the near-range, we set a distance threshold d_{th} to select lidar detections and lifted-2D detections. We use only the lidar detection algorithm to detect accurate keypoints within a distance less than $d_{th} = 20m$. In the region beyond d_{th}, we combine the lifted-2D detections with the lidar detections. In this way, our fusion approach preserves lidar-based work zone detection accuracy while significantly increasing the detection range. Finally, we generate the work zone contour using the same method as in Algorithm 2.

6.4 Experimental Results

6.4.1 Dataset

We use the nuScenes [20] autonomous driving dataset to evaluate our implementations. The nuScenes dataset contains synchronized camera and lidar sensing data in different driving scenarios. The dataset's annotation, however, lacks work zone information. We

Range	Method	mIoU	Recall	Precison	*maxDist*
10m	**Image**	0.50	80.72%	73.46%	8.12
	Lidar	0.65	86.06%	86.16%	8.37
	Fusion	0.64	86.06%	86.03%	8.37
20m	**Image**	0.46	80.24%	72.62%	16.17
	Lidar	0.58	83.43%	86.86%	16.19
	Fusion	0.58	85.26%	85.53%	16.96
50m	**Image**	0.30	60.93%	53.44%	27.63
	Lidar	0.31	59.10%	77.64%	22.57
	Fusion	0.36	64.66%	70.26%	26.96

Table 6.1: Experiment results within a distance of 10 meters, 20 meters and 50 meters

provide annotations to the nuScenes dataset for work zone regions to study and evaluate work zone detection algorithms. We identified 757 work zone samples from 19 driving sequences in the nuScenes validation and test set. We manually annotated both the work zone area and the road area using polygons.

6.4.2 Results

We test the baseline implementations described in the previous section. We compare the IoU, recall, precision, and the *maxDist* in three distance ranges: 10 meters, 20 meters and 50 meters. The results are listed in Table 6.1.

All of our three methods perform well at short distances. Within 10 and 20 meters, the lidar-based method outperforms the image-based method in all metrics. However, in the range of 50 meters, the image-based method achieves a higher *maxDist* score, which means the image-based method can find a work zone at a longer distance although its detected boundary is not accurate on average.

Figure 6.6 shows the qualitative comparison of our three sensing modalities. The green

polygons indicate the road region and the red polygons are the ground truth annotations of work zones. Other colors indicate distinctly detected work zones. The cycle points are detected keypoints and the square points are the nearest road boundary points. The lines are the polygon contours of detected work zones. We assign a uniform color for the keypoints and contours that belong to the same work zone.

The fusion-based approach combines the image keypoints with the lidar keypoints. In the distance range of 10 and 20 meters, the fusion-based detection achieves similar accuracy as the lidar detections. However, in the evaluation at a range of 50 meters, the fusion results have better IoU than lidar results, indicating a better match between the detected contour and the ground truth. Also, the fusion-based method has a longer *maxDist* which is close to the image-based method. These results show that a multi-modality fusion method can take advantage of individual sensors and surpass any single-modality methods.

6.5 Summary

In this chapter, we studied the problem of work zone detection for use in autonomous vehicles. We provided a taxonomy of work zones and formulated the goal of work zone detection to provide a set of polygonal representations of the work zone area within the vehicle's Bird's Eye View. We also proposed a set of evaluation metrics, including *IoU*, *Recall and Precision*, and *maxDist*. To evaluate the performance of work zone detection, we annotated the popular autonomous driving dataset, nuScenes. Furthermore, we designed a general work zone detection pipeline and evaluated three implementations using images, a point cloud and a combination of both. Our experimental results show that

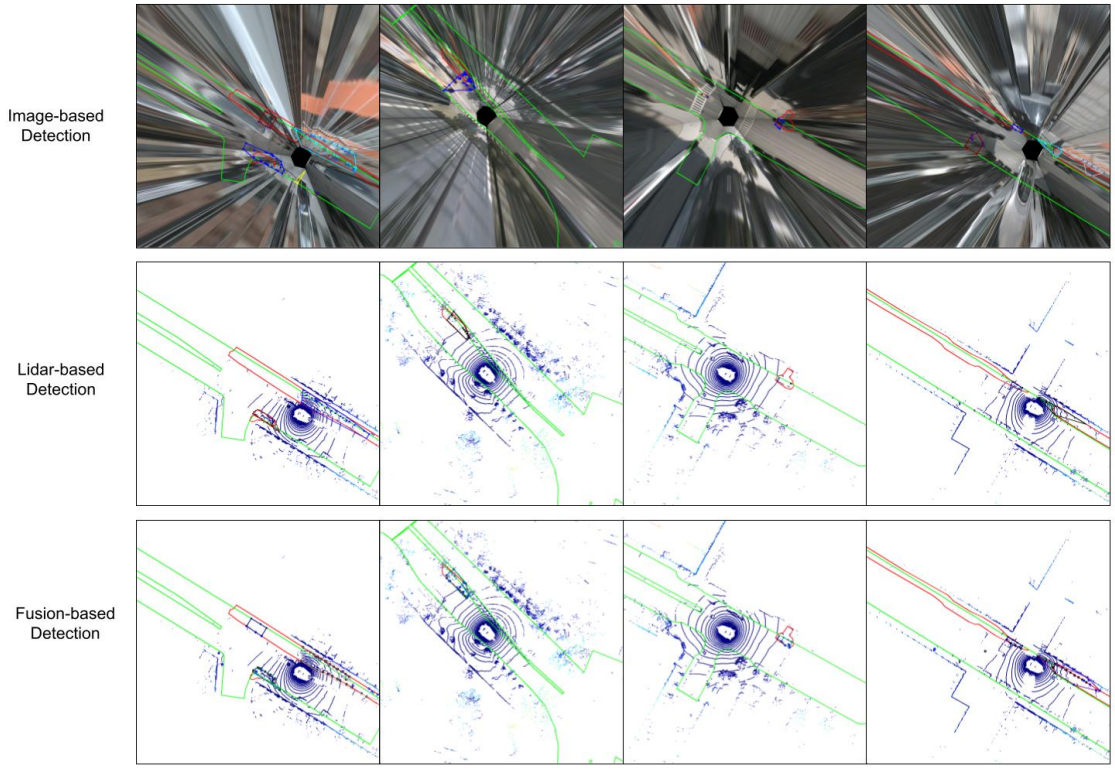

Figure 6.6: Qualitative comparison of three implementations using images, a point cloud and a combination of both, respectively. The green polygons indicate the road region and the red polygons are the ground truth annotations of work zones. Other colored polygons are the detection results.

the image-based approach has the largest detection range while the lidar-based approach has the best detection accuracy. The fusion-based approach takes advantage of both and achieves a balanced detection range and accuracy. Our results serve as good baselines for further study. So far, we detect a work zone as a static area. In practice, a work zone can be more complex where flagmen may provide temporal traffic control. In the next chapter, we discuss our solution for flagman recognition.

Chapter 7

Challenging Scenarios: Flagman Recognition

7.1 Introduction

For an autonomous vehicle to navigate safely, it must follow the rules of the road and abide by traffic control signals. These control signals are typically from road infrastructure such as traffic lights and lane markers. However, in the presence of a work zone, a flagman may provide temporary traffic control instead. A flagman uses hand gestures, often in conjunction with a prop such as a sign paddle, to guide vehicles safely through the zone. A work zone can also occur on short notice due to traffic emergencies or road maintenance, making it difficult for a map database to be updated on time. Therefore, an autonomous vehicle needs to recognize these traffic gestures on-the-fly.

Understanding traffic gestures is a challenging task for computers. Although a point-

cloud object detection system can generally detect pedestrians, it is challenging to recognize the gestures using point clouds. The point clouds are sparse and may not capture the subtle movement of the human body. Instead of utilizing an expensive lidar system capable of a high-resolution scan of the human body from a distance, a gesture recognition system can use low-cost cameras for fine-grained recognition after receiving the general position of a pedestrian. However, it is still challenging to recognize gestures from camera videos. A flagman might wear different uniforms according to the weather condition. When a good samaritan acts as a flagman, one may not wear a uniform at all. Besides clothing, flagmen naturally have different body sizes and skin colors. As there is no strict standard, traffic gestures are often flexible, and flagmen have their individual styles. Sometimes, these actions can be as subtle as finger movements, or in rarer cases emulate dance moves.

Moreover, a flagman commonly uses props together with body poses. For example, a flagman may carry a reflective sign paddle or a light stick to make the signal more noticeable at night. Therefore, the recognition system must perceive a large variety of human body poses and their interactions with miscellaneous objects. Furthermore, the recognition system must run in real-time with computationally constrained resources on the vehicle.

Many studies on traffic gesture recognition capture the body pose using wearable motion sensors such as an IMU [63]. Such a capture system is not available for all flagmen and is not realistic in typical driving scenarios. Alternatively, a camera has a low cost and is available in most autonomous vehicles. Recent research on video action recognition uses deep neural networks [29, 30, 31] to learn from training data. Yet, there are only a few video datasets[63, 85] for traffic gesture recognition, which contain relatively simple

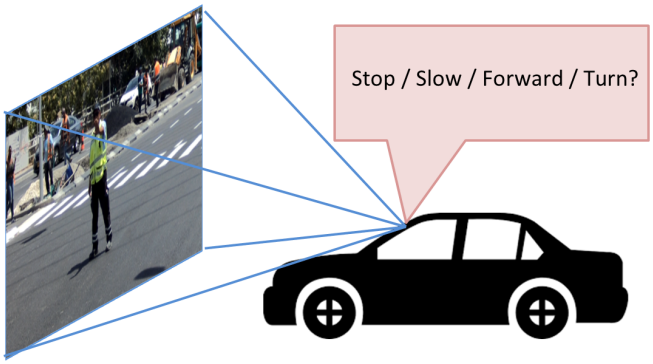

Figure 7.1: An autonomous vehicle needs to make safe decisions and facilitate forward progress in the presence of road construction workers and flagmen.

traffic control poses and lack the use of props.

In this chapter, we first provide a taxonomy that categorizes traffic gestures based on the flagman's appearance, signal semantics, and environmental condition. We then build a video dataset covering common traffic gestures, including the usage of props. After manual annotation on the video frames, we develop an efficient deep network to use a combined data representation of hand images, pose keypoints and bounding boxes. Our experiments show that this mixture of representations effectively captures the meaning of gesture signals while remaining robust against the variance of flagmen's appearance. We also provide an extensive ablation study on the impact of each component.

7.2 Taxonomy

The large variety of human gestures represents significant complexity. This section categorizes traffic gestures according to their semantics, flagman appearance and environmental

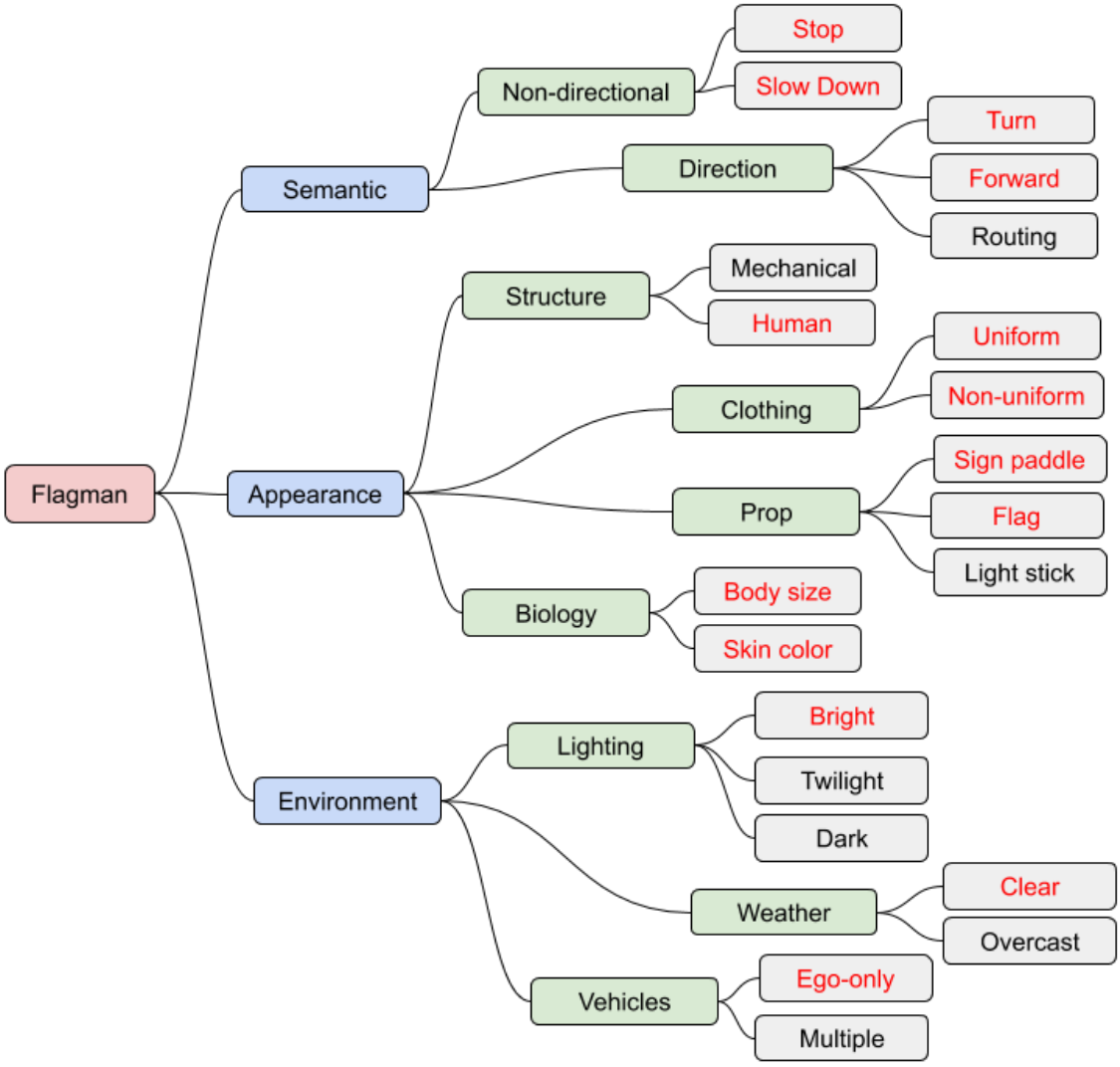

Figure 7.2: A taxonomy of flagman traffic gestures. Red color indicates the attributes that our dataset covers.

context, as shown in Figure 7.2.

7.2.1 Semantic Classification

Traffic gestures are used to provide instructions to vehicles traversing the work zone. We categorize gestures as non-directional and directional, depending on whether they offer instructions to a vehicle's moving direction. Non-directional gestures control the vehicle's moving state. Common examples are the *Stop* and *Slow Down* gestures. The *Stop* gesture requires the vehicle to stop, while the *Slow Down* gesture instructs the vehicle to move slowly. Directional gestures, in contrast, guide the vehicle's route. For example, when a road is closed temporarily, a flagman usually points to the direction in which the vehicle should *Turn*. However, the vehicle should not take the direction literally since the indicated absolute direction may be approximate. In that case, the vehicle may need to incorporate map and drivable region information to choose the safe route. Other than the *Turn* signal, *Forward* is also a common directional gesture that indicates the vehicle to move towards the flagman's direction. The directional gesture can be a complex sequence when it conveys a route to a specific destination, such as an empty spot in a parking lot.

Both non-directional and directional gestures override the traffic guidance from existing infrastructure such as lane markers and sometimes even traffic lights. A flagman may give a *Stop* signal even if the traffic light is green. One may also indicate a direction, following which the vehicle needs to go off-road to bypass the work zone. An autonomous vehicle must make the correct interpretation and combine both the map and traffic gesture recognition for its operation.

7.2.2 Flagman Appearance

Typically, the entity that makes the traffic gestures is a human. However, mechanical flagmen have started to appear as well. Some resemble a human beings, while others are akin to a portable barrier gate operator. As there is no universal standard for the design of a mechanical flagman, detecting one is complicated. Mechanical flagmen are out of our current scope of work, and we focus on human flagmen.

The common variances on the flagman are the clothing, biometrics and the use of props. For their own safety, most flagmen or traffic police wear uniforms typically made of reflective material. However, in the case of a good samaritan acting as a flagman, he/she may not have a uniform at all. The uniform or the lack of one may impact the visibility of gestures as well as the motion range. A flagman's biometrics also contribute to the changes in gestures. As different persons have different body sizes and styles, the exact position of each body part may differ during sign motion. Different skin colors also induce potential changes in camera images.

Moreover, a flagman often uses dedicated props such as sign paddles, flags, light sticks and flashlights. Those props are indispensable components of the gestures and should be considered. Therefore, the desired recognition algorithm also needs to detect these props and body poses regardless of the clothing and biometrics.

7.2.3 Environmental Context

The gesture semantics and flagman appearances are closely related to the environmental conditions. The flagman's motion is more visible on a bright and clear day. In difficult viewing conditions such as bad weather and low light, the camera system might not detect

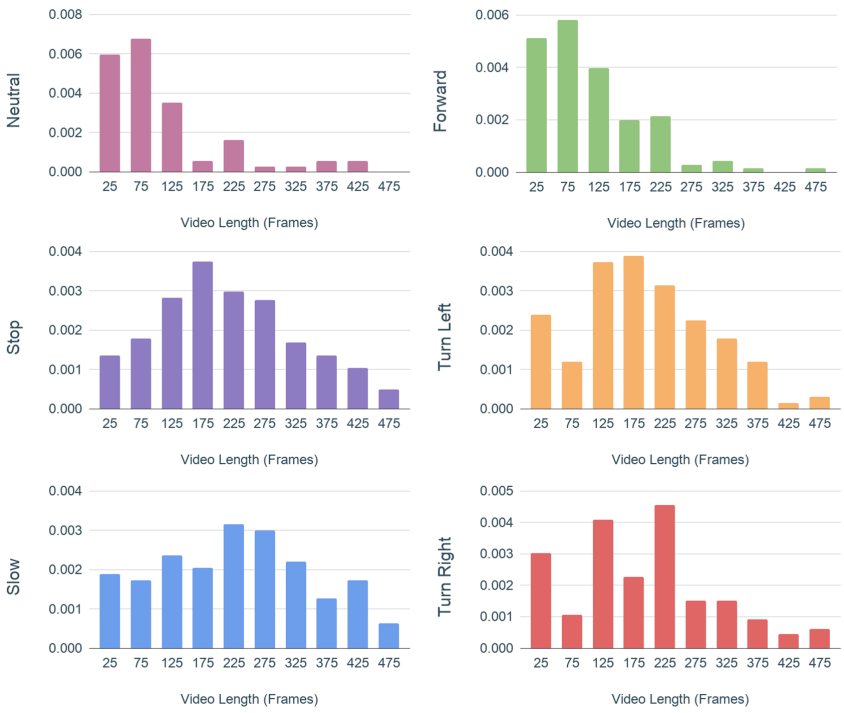

Figure 7.3: The distributions of video length for each category.

or recognize the flagman. The existence of other vehicles is another important environmental factor for gesture recognition. When there are multiple vehicles, the autonomous vehicle must determine whether the flagman is signaling to it or some other vehicles or pedestrians. This requirement is particularly challenging when vehicles merge from multiple lanes. The desired recognition system must take cues from a flagman's standing position and other vehicles' locations to determine the intended recipients of gestures.

Figure 7.4: Examples from our traffic gestures dataset.

Category	Attribute	Videos	Frames
Neutral	-	74	7682
Stop	No object	93	12486
	Paddle	178	45871
	Flag	79	22755
	Paddle and flag	43	16244
Slow	No object	45	8309
	Paddle	47	11737
	Paddle and flag	42	13190
Forward	No object	143	16725
Turn left	No object	111	21042
	Flag	29	8094
Turn right	No object	108	20048
	Flag	27	6961

Table 7.1: Statistics of our TGR dataset

7.3 Our TGR Dataset

To train and evaluate a recognition algorithm, we build a traffic gesture dataset that contains frequent traffic gestures. Our dataset contains video sequences from the view of the vehicle. The videos are recorded inside a studio with bright lighting and a green screen in the background. Actors with different ethnicities, races, gender and body sizes perform six categories of gestures: *Stop*, *Slow*, *Forward*, *Turn Left*, *Turn Right*, and also non-traffic *Neutral*. Each category comprises gestures using only hands and gestures using a sign paddle or a flag. All videos are recorded using a webcam that connects to a Raspberry Pi. The videos are 1080p in raw form. We center the video frames around the stage and crop the image to remove the background outside the green screen. After post-processing, all videos have a resolution of 800×800 pixels and a frame rate of 30 fps. We also instruct the actors on the type of gestures to perform and act naturally according to their personal

preferences to keep our dataset diverse. Table 7.1 shows the number of videos and frames for each category, and Figure 7.4 shows examples from the dataset.

The dataset has 1019 videos in total, divided into a training set of 830 videos and a test set of 189 videos. The actors in the test set are different from the actors in the training set to avoid overfitting. Each video clip contains one continuous traffic gesture. To achieve low latency, we annotate the start of the action at the first frame from which the gesture is recognizable. Similarly, we annotate the end frame as soon as the gesture stops. The video clips have diverse lengths, and their distributions are shown in Figure 7.3.

7.4 Architecture

To address the challenge of diverse flagman appearances and the use of props, we develop a deep-learning-based framework using an efficient representation of traffic gestures. It comprises pose keypoints, hand images and bounding boxes of props. We extract features from this representation and use a Long Short-Term Memory ([86]) network for temporal modeling, as shown in Figure 7.5. In this section, we describe each component in detail.

7.4.1 Pose Keypoints

An image contains rich information on the appearance of the flagman and the surroundings. However, extracting the image features relevant to traffic gestures itself is non-trivial. Direct feature extraction using convolutional neural networks (CNN) may contain unnecessary image information and is prone to overfitting. The network often needs a large amount of training data to improve its generalization, which is expensive to collect and

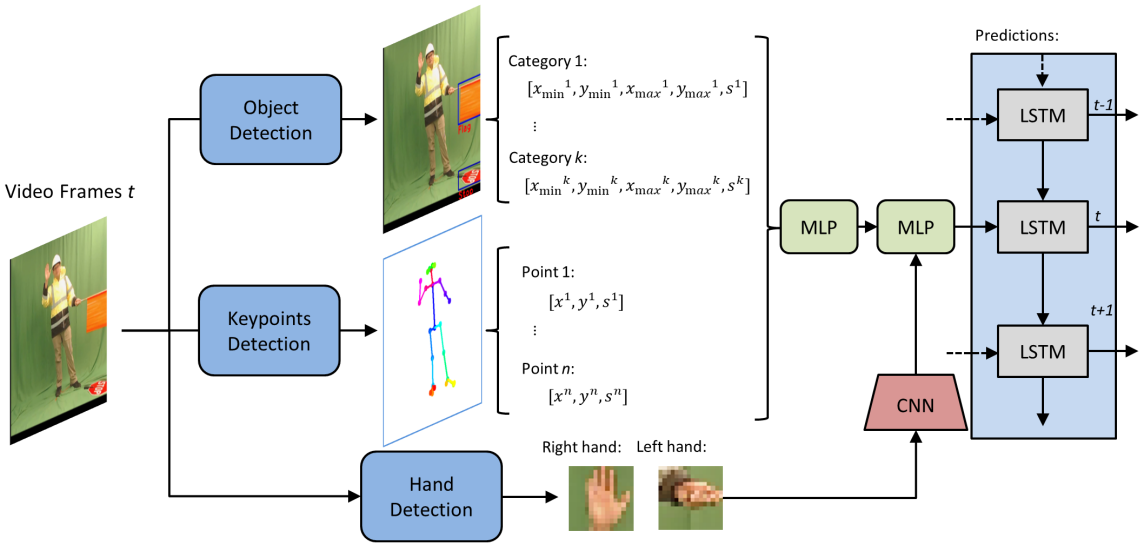

Figure 7.5: Overall Architecture of our recognition system. We create a dataset with common traffic gestures in conjunction with props such as flags. We use a mixture data representation comprise of human skeleton keypoints, hand images, and object bounding boxes. This mixture representation is processed by a neural network to predict the meaning the gesture.

annotate. Instead, we reduce the irrelevant information by extracting keypoints to represent the human pose. We center the coordinates of keypoints and concatenate them into a pose feature vector:

$$f_{pose} = [x_1 - x_c, y_1 - y_c, s_1, ...] \tag{7.1}$$

where, (x_i, y_i) are the coordinates of i^{th} keypoint, s_i is its detection score, and (x_c, y_c) is the skeleton center point. In practice, we choose the neck keypoint as the center point.

7.4.2 Hand Images

The keypoints capture the body pose and aggressively remove the flagman's exterior appearance. However, the keypoint representation also ignores the hand pose and finger movements, which carry valuable gesture information. For instance, the front of the palm often indicates a *Stop* or a *Slow* signal, while the back of the palm often appears in a *Forward* gesture. Moreover, finger movements are common when a flagman indicates a direction. To capture the hand's cues, we crop the hand images and use a light-weight CNN to encode the hand. We use the wrist keypoint (x_w, y_w) and the elbow keypoint (x_e, y_e) to locate the hand position (x_h, y_h):

$$(x_h, y_h) = ((1+\alpha)x_w - \alpha x_e), (1+\alpha)y_w - \alpha y_e)) \tag{7.2}$$

where, α is a constant ratio between the length of the palm and the forearm. We empirically set α to 0.4.

After locating the hand image regions, two weight-sharing CNNs encode the left-hand

image I_{left} and right-hand image I_{right} into hand-feature vectors:

$$(f_{left}, f_{right}) = (CNN(I_{left}), CNN(I_{right})) \tag{7.3}$$

If the wrist or elbow keypoint is missing or the hand position is out of camera view, we create an empty image as the hand image.

7.4.3 Object Bounding Boxes

We also use a real-time object detection algorithm to detect common flagman props such as a stop sign paddle, a slow sign paddle, or a flag. We concatenate the detection results into an object feature vector.

$$f_{object} = [x^1_{min}, y^1_{min}, x^1_{max}, y^1_{max}, s^1, ...] \tag{7.4}$$

where, $[x^i_{min}, y^i_{min}, x^i_{max}, y^i_{max}]$ is the bounding box coordinates of i^{th} object category and s^i is the detection score. Similar to the keypoints, box coordinates are also centered with regard to the skeleton center point. If no object is detected for an object category, we use zeros as the bounding box coordinates and detection score.

7.4.4 Gesture Prediction

Finally, we concatenate the feature vectors of the pose keypoints, the hand images and the object bounding boxes to form the complete feature representation.

$$f_{gesture} = [f_{pose}, f_{left}, f_{right}, f_{object}] \tag{7.5}$$

We pass $f_{gesture}$ to an LSTM to model the temporal motion. An MLP takes the LSTM output and predicts the probability P_m^t for the m^{th} gesture category at time t. We use cross-entropy as the loss function. The overall loss value is given by:

$$Loss = \frac{1}{M}\frac{1}{T}\Sigma_{m=1,t=1}^{M,T}(-l_t log(P_m^t)) + L_{reg} \tag{7.6}$$

where, T is the length of the sliding window and L_{reg} is the regularization loss on network weights.

7.5 Experimental Results

In this section, we conduct experiments on our recognition framework and study the effect of each of its components.

7.5.1 Settings

For keypoint detection, we use the OpenPose [87] toolbox. It detects 25 keypoints in total and generates a f_{pose} of length 75. We use MobileNet-SSD[88] to detect three types of objects: a stop sign paddle, a slow sign paddle or a flag. Following Equation 7.4, f_{object} has a length of 15. We then pass the concatenation of f_{pose} and f_{object} to an MLP of (256, 256, 256) units to compute a joint feature $f_{pose+object}$.

The dimension of hand-image regions is 40% of the height between the neck and hip keypoints. We further resize the hand images to 32×32 pixels. We then use MobileNet[88] to extract f_{left} and f_{right} from the hand images. Afterward, we concatenate f_{left}, f_{right}, and $f_{pose+object}$. We pass the concatenation to another MLP of (512, 512) units for fusion. An

LSTM of 256 units processes the fused features. An MLP classification header of (256, 256, 6) takes the LSTM output and predicts the gesture category. We use a batch size of 16 and a sliding time window of $T = 50$ frames. The states of LSTM pass across batches. We train the networks using SGD with a momentum of 0.9. The initial learning rate is 10^{-4} and decreases to 10^{-5} after 192 epochs. In total, this keypoint model is trained with 200 epochs on the training set.

For comparison, we also train two additional models using the image and the keypoint heatmaps as input, respectively. The image model uses MobileNet as the backbone network to extract image features from a raw input image of [224, 224, 3] and pass the feature vector to the LSTM. Similar to the keypoint model, we also align images from different frames of the same video in the neck keypoints. The image model consumes more GPU memory than the keypoints model. Therefore, the image model is trained with a reduced batch size of 2. Other training configurations are the same as the keypoint model.

The heatmap model uses keypoint heatmaps instead of keypoint coordinates as input. The same OpenPose model generates the heatmaps. But, unlike the keypoint coordinates, each heatmap is an image of a keypoint's detection score, as shown in Figure 7.6. We add three extra heatmaps for the object detection. Again, the heatmaps from different frames of the same video are aligned in the neck keypoints. The heatmap model also uses MobileNet as the backbone network. However, the network input layer is modified to accept an input size [224, 224, 28]. Similar to the image model, the heatmap model uses a batch size of 2. Other training configurations are the same as the keypoint model.

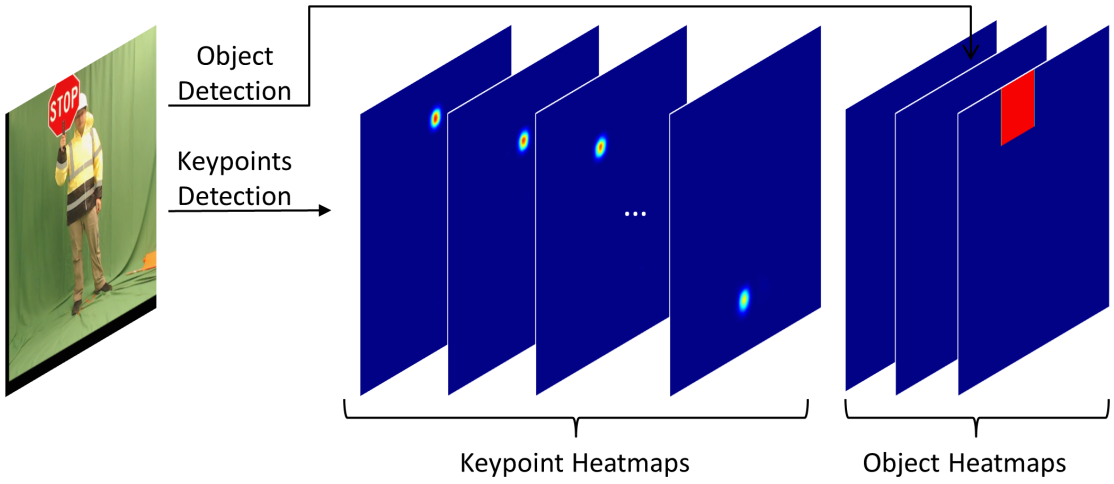

Figure 7.6: A heatmap model for comparison. Each keypoint heatmap is an image of a keypoint's detection score and an object heatmap contains the filled bounding box of a detected object in a category.

7.5.2 Data Augmentation

We apply the same set of data augmentation methods to all models. We randomly flip the image's x-axis and rescale the image in a range of [90%, 110%]. We also randomly adjust the image brightness, contrast, saturation and hue. Note that images from the same video clip share identical augmentations.

7.5.3 Results

We compare the prediction accuracy, recall, precision, and F1-score of different models. The main experiment results are shown in Table 7.2. The accuracy is the rate for each frame's prediction to match the ground truth annotation. The recall, precision and F1-score are averaged across all categories. As can be observed, the proposed representation using

Table 7.2: Experiment results

	Hand Image	LSTM	Accuracy	Recall	Precision	F1-score
Image	×		79.89%	70.05%	76.01%	0.7184
	×	×	84.97%	75.85%	82.63%	0.7829
Heatmap			80.13%	71.02%	77.48%	0.7305
		×	85.62%	79.51%	83.06%	0.8061
Keypoints Only			66.42%	48.09%	45.98%	0.4625
		×	69.75%	56.45%	66.43%	0.5439
Keypoints + Bounding Boxes			76.93%	66.46%	74.63%	0.6849
		×	81.23%	72.26%	77.58%	0.7381
	×		82.01%	74.86%	79.17%	0.7657
	×	×	**89.42%**	**85.83%**	**87.22%**	**0.8620**

keypoints, bounding boxes and hand images achieves the highest metrics. It surpasses the accuracy of the image model and the heatmap model by more than 4% in the accuracy. Figure 7.7 shows the confusion matrix of prediction. In Table 7.2, we also compare the effect of using the LSTM. It can be observed that using the LSTM consistently improves the prediction accuracy, regardless of the input representations. Figure 7.11 shows qualitative results for the model that combines keypoints, bounding boxes and hand images. Our model performs well in most cases. Some common failures are caused by the ambiguity between turning and forward gestures, blurred hand motions and incorrect object detection.

We test the end-to-end running speed of our proposed model on a desktop with an E5-1630 CPU and a GTX 1070 GPU. The proposed model using keypoints, bounding boxes and hand images achieves an average speed of 11.89 frames per second on the validation dataset.

We also compare the robustness of different models by testing their accuracy on varied image brightness, colors and rotations. Figures 7.8, 7.9 and 7.10 show the results. To test the impact of brightness, we change the test image's absolute pixel value by -40%,-20%,

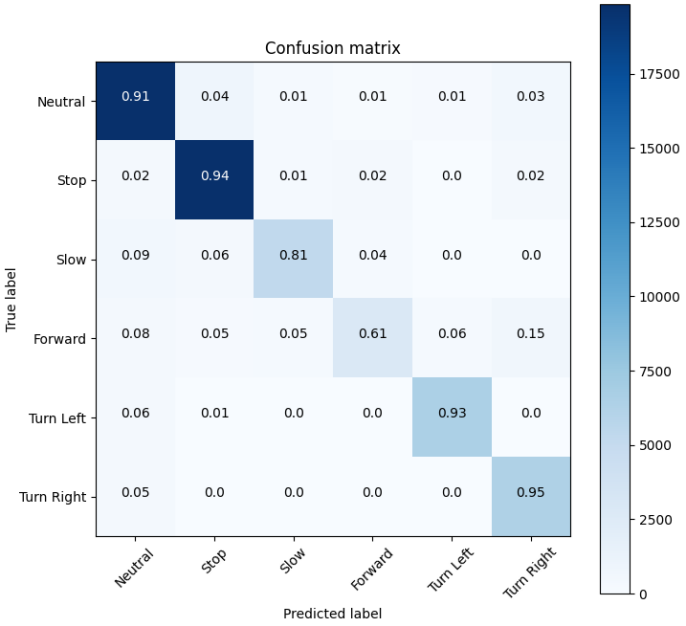

Figure 7.7: The confusion matrix of prediction on the test dataset.

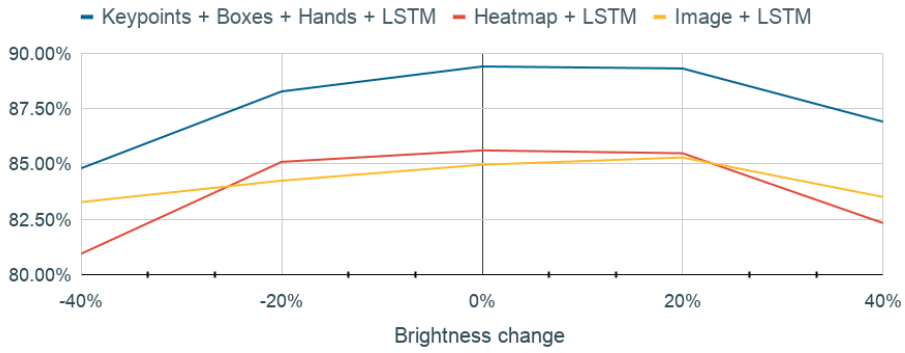

Figure 7.8: Test accuracy with varying image brightness.

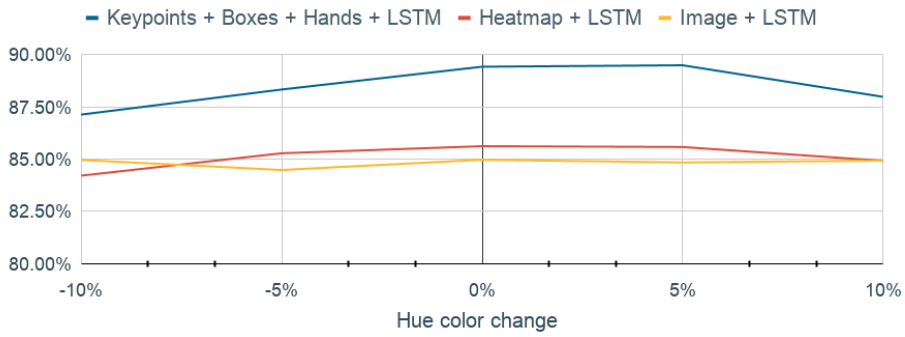

Figure 7.9: Test accuracy with varying image hues.

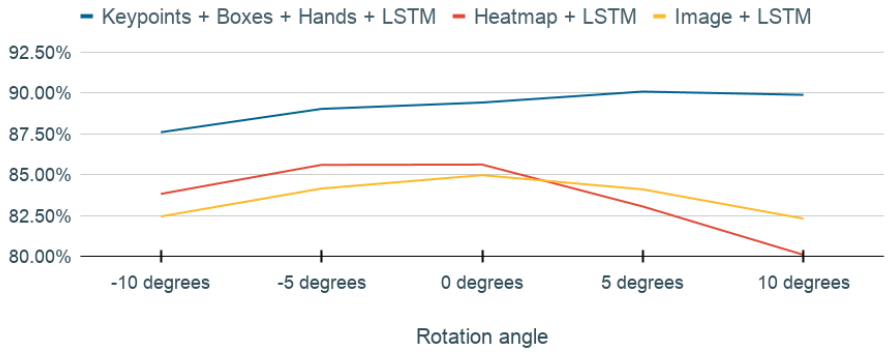

Figure 7.10: Test accuracy with image rotations.

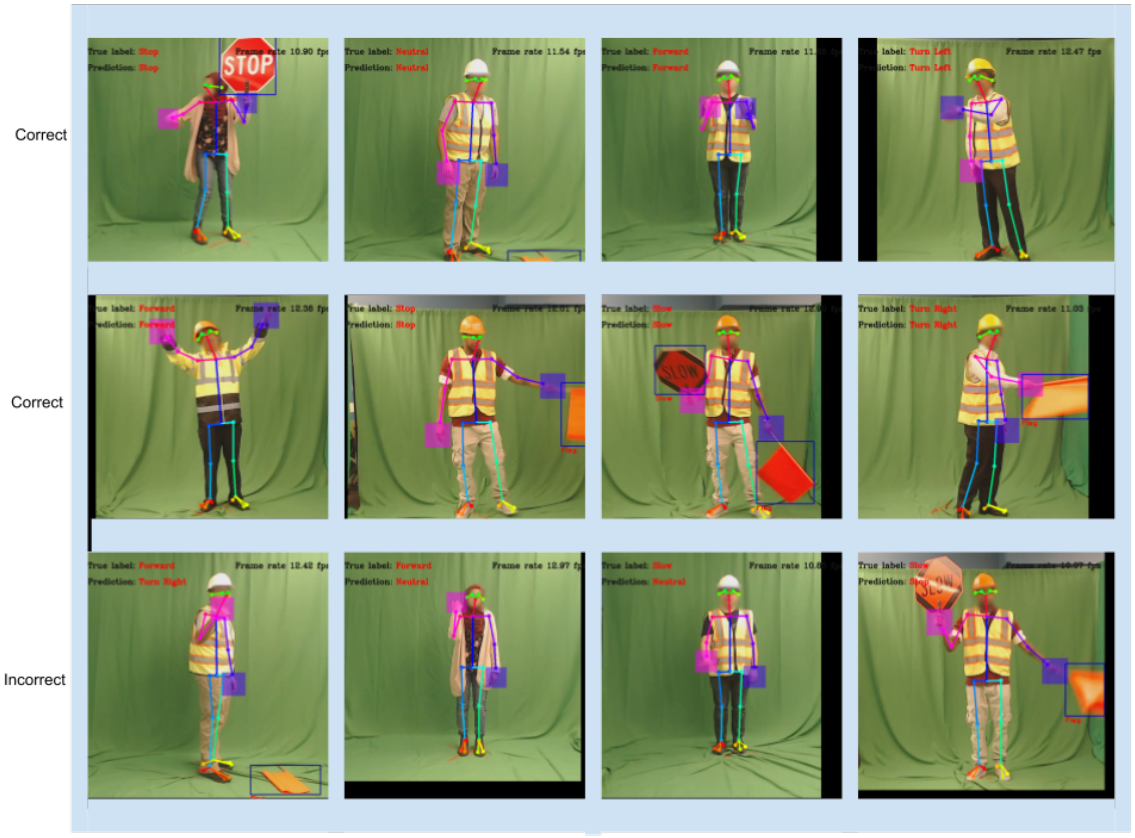

Figure 7.11: Qualitative results from the proposed model using keypoints, bounding boxes and hand images. The first and second rows are examples of correct predictions. The third row shows some examples of failure cases.

20% and 40%. For color variation, we extract the Hue value of the test image and adjust it by -10%, -5%, 5% and 10%. Finally, we rotate the test images -10, -5, 5 and 10 degrees clockwise around their image centers. As shown in Figures 7.8, 7.9 and 7.10, the proposed model using keypoints, bounding boxes and hand images achieves higher accuracy than the basic image model and the heatmap model across all input variations, demonstrating the robustness of our proposed method.

7.6 Summary

In this chapter, we studied the challenging problem of flagman gesture recognition. We presented a taxonomy of challenges related to varied flagman traffic gestures. We built a dataset that covers a range of common flagman gestures, including the use of props. We then proposed a recognition technique that uses keypoints, hand images, and object bounding boxes to represent gestures. Our experiments show that our proposed representation achieves a higher detection accuracy than approaches using raw images or heatmaps. Our method is a feasible downstream solution to recognize gestures after object detection.

Chapter 8

Conclusions and Future Work

In this chapter, we present our conclusions, summarize our research contributions and discuss future work.

8.1 Conclusions

This thesis targets a cost-efficient and accurate perception system for autonomous vehicles. We presented multiple methods to accomplish the goal. We first designed a graph neural network named Point-GNN to detect 3D objects from a compact graph representation of the point cloud. We encode the irregular point cloud directly without densifying them in a grid, unlike conventional methods. Our Point-GNN uses graph edges to aggregate features directly between points. Its competitive accuracy in the KITTI benchmark's 3D and Bird's Eye View object detection demonstrates its effectiveness as a point cloud backbone for object detection.

Secondly, we leveraged the co-existence of low-cost cameras to boost object detec-

tion accuracy on a point cloud. We use multi-view image features to estimate a lidar point's local object coordinates and register them to lidar coordinates. Experiments using the nuScenes and Level 5 dataset showed that our fusion method achieves leading performance when combining lidars with low-cost cameras. It will enable broader use of lidar in cost-sensitive application domains such as consumer vehicles, warehouse robots and work zones.

Thirdly, we reduce the overall system expense by using abundant unlabeled data for training. Our self-supervised learning method combines geometric pretext tasks and contrastive loss for pretraining. Results on the nuScenes dataset demonstrated that we can leverage cheap unlabeled data to improve detection accuracy.

Finally, we proposed solutions for challenges in complex real-world driving scenarios such as work zones. We provided a taxonomy of work zones and formulated the goal of work zone detection. We also proposed a set of evaluation metrics and annotated the popular autonomous driving dataset, nuScenes. Our experimental results showed that our fusion-based approach utilizes both modalities and achieves a balanced detection range and accuracy. We then designed a gesture recognition system for more complex scenarios where a flagman is present. We also presented a taxonomy of challenges related to flagman traffic gestures and built a dataset that covers a range of common flagman gestures. Our recognition algorithm uses keypoints, hand images, and object bounding boxes to achieve a higher detection accuracy than approaches using raw images or heatmaps.

Together, our graph-based point-cloud neural network, a fusion of camera images and a sparse point cloud, a self-supervised training method, and solutions for work zones enhance perception accuracy and cost-efficiency in AVs.

8.2 Research Contributions

As a part of this dissertation, we made the following contributions:

1. Graph-based compact point-cloud [33].
    (a) A point-cloud object detection pipeline using a graph representation.
    (b) A graph neural network (GNN) that is capable of accurate object detection.
2. A sparse fusion of camera images and point clouds [89].
    (a) A fusion method that utilizes multi-camera images to boost object detection using a sparse point cloud.
3. Geometry-centric self-supervised training [90].
    (a) Evaluation of the prevalent contrastive loss and its limitation in point cloud 3D object detection.
    (b) A pretraining method that combines geometric pretext tasks and contrastive loss to improve self-supervised learning for point cloud object detection.
4. Context-specific approaches for work zones [91, 92].
    (a) A definition of the work zone detection problem and practical evaluation metrics.
    (b) A set of work zone annotations to the autonomous driving dataset.
    (c) A work zone detection pipeline using multi-modality sensor configurations.

(d) A taxonomy that categorizes traffic gestures based on the flagman's appearance,signal semantics, and environmental conditions.

(e) A video dataset covering common traffic gestures, including the usage of props.

(f) An deep network to recognize gestures using a combined data representation of hand images, pose keypoints and bounding boxes.

(g) A detection and recognition pipeline utilizing both point-cloud object detection and camera-based gestures recognition.

8.3 Future Work

8.3.1 Open Questions for an Effective GNN

Graph neural network provides an elegant way to introduce inductive bias into a deep learning model. Relational assumptions [93] can be built into a graph structure that enables a neural network to reason on top of them. Moreover, relational reasoning is the core of many robotic perception topics such as object structure understanding, multi-agent Interactions, coordinates association, etc. Therefore, a graph neural network has an enormous potential to be a powerful tool to address many challenges in robotics. One tip of the iceberg discussed in this thesis is to use a graph to encode point cloud structure and detection objects. There are many benefits of our approach. For one thing, a graph directly connects relevant points and omits the computation wasted on unmeasured space. A new sensor modality can also be included by adding new edges in the graph model. However, our current Point-GNN approach has its limitation on running speed, and many state-of-

the-art approaches still prefer convolutional neural networks. This section discusses the core question: how can a GNN model be more effective?

8.3.2 Graph Optimization

A graph neural network operates on a graph. In Point-GNN, we use a neighborhood graph that connects the lidar points within a distance radius. The neighborhood graph encodes our assumption that points from the same object should be spatially close to each other and reasoned together. Although the assumption is simple and generally correct, the resulting graph is not simple in terms of edges. The number of edges significantly increases when the density of points increases. As the graph neural network generates features for each edge, the computational and memory cost increases proportionally to the number of edges. To keep the neural network light-weighted, the graph needs to be optimized. Some options are k-nearest neighborhood graph, distance radius reduction, graph trimming, and metric-based dynamic graph.

K-nearest neighborhood graph. A straightforward alternative to the radius-neighborhood graph is the K-nearest neighborhood graph which connects a point to K nearest neighbors instead of all the neighbors within a radius. A K-nearest neighborhood graph of N points contains at most kN edges, and thus the computational complexity of edge features is O(N). This linear scaling of the number of points is desirable. However, a K-nearest neighborhood graph depends on the point cloud density. The average length graph edge is short in the point cloud region, where the high point density is. While in the point cloud region, where the points are sparse, the average length of the graph edge is long. Given a m-hop neighbor on the graph, its physical distance highly depends on the point cloud

density. For instance, after the same number of graph iterations, a point far away from the lidar may receive features from meters away. In contrast, an object near the lidar only receives features from centimeters away. This mismatch in the distance requires significantly more graph neural network iterations so that every point can have information from enough physical distance regions. Moreover, it also requires the network to have a good mechanism not to forget local features after many iterations.

Distance radius scaling. In a radius neighborhood graph, the number of neighbors decreases when the distance radius decrease. Therefore, another way to reduce the total number of edges is to reduce the distance radius. Assume a total of N points that are uniformly distributed in 3D space with a volume of V, the average number of points within a radius r is $O(\frac{N}{V}r^3)$. The total number of edges in the radius neighborhood graph is then $O(\frac{N^2}{V}r^3)$. If we scale r as $cN^{\frac{1}{3}}$, we can also achieve $O(N)$ linear scaling. Unfortunately, if the r is too small, we get a disjoined graph in the region where the points are sparse. One possible solution is to combine the radius neighborhood graph with K-nearest neighborhood graph.

Graph edge trimming. We could also build a radius neighborhood graph with a large radius to ensure graph connectivity first and then trim the unnecessary edges. We can set a maximum number of edges per point in the simplest form and drop the edges when they max out. This technique is already implemented in Point-GNN's training phase. The limitation is that the dropping process is random. The challenge of designing a better trimming policy is that it is difficult to determine which edges are important. An edge can be vital to pass features in some graph neural network iterations while remaining redundant for other iterations. Although data-driven methods such as [94, 95] can evolve

the graph structure to achieve better task-specific accuracy, they require time-consuming optimization steps. Also, these methods need to be modified to reduce the number of graph edges while keeping reasonable detection accuracy. However, it is still an open question on how to accumulate the optimization step. A possible direction is to perform incremental optimization when processing a temporal sequence of point clouds. Because a point cloud may not change considerably in a short period, the optimization can reuse the graph from previous timestamps to reduce the computational cost. The KITTI [18] dataset only provides independent point cloud samples, and it is not a good dataset to test such a method. However, recent large datasets such as nuScenes [20] and Waymo Open [19] provide sequential point cloud records, which enable new opportunities.

Dynamic Graph. Different from a graph constructed statically from the initial point cloud, a dynamic graph constructs itself on the fly. A dynamic graph can be potentially more cost-effective by utilizing deep neural network features. DGCNN [17] shows that a KNN graph can be constructed using point features in each graph neural network iteration. Similarly, IDGL [95] also propose to update the graph iteratively by mixing it with a neighborhood graph from feature space. These studies have shown promising improvement in task accuracy. However, they do not focus on reducing the number of graph edges. Moreover, some dynamic construction process involves the calculation of pair-wise feature distance, which has the complexity of $O(n^2)$. If not designed properly, the overhead of computing a new graph may overweight the reduction in graph edges. A potential approach is to constrain the dynamic changes to a subset of the graph. In the subset, the number of points can be reduced to m, and the complexity of pair-wise matching can be controlled to $O(m^2)$. Similarly, the dynamic changes can also be limited to the local graph

area.

8.3.3 Limitations of Set Functions

The core of a graph neural network is the aggregation of features from connected vertices. The aggregation is a *Set* function that takes a permutation-invariant set of vertices as input and maps their value to an output. It is challenging to find a set function manually that maps the input set to the desired output, such as semantic labels. Fortunately, deep learning on *Set* allows approximation of desired set function using neural networks in a data-driven manner. However, the open question is how complex a neural network should be to aggregate the features well. In other words, It is still unclear what is a cost-efficient way to aggregate features on a graph.

Compared to the wide range of applications using deep learning on set, the theoretical studies are limited [11, 10, 96]. Some pioneering studies provide some insights into designing an effective GNN.

Sum Decomposition. DeepSet [11] provides a decomposition of set functions via summation:

$$f(X) = \rho(\sum_{x \in X} \Theta(x)) \tag{8.1}$$

where $X = x_1, \ldots, x_n$ is a set. DeepSet proposes to use neural networks to approximate $\rho(.)$ and $\Theta(.)$. It also proves that when x is countable, any permutation-invariant function $f()$ can be decomposed using $\Theta(.) \in \mathbb{R}$. This indicates we can compute just one real number feature value per element in the countable set and sum them to get any permutation-invariant set output. However, the universal approximation theorem of neural networks is

based on a continuous domain instead of a countable domain.

The authors of [96] prove that for a set of size $\leq M$, all continuous permutation-invariant functions can be represented if and only if $\Theta(.) \in \mathbb{R}^K$ and $K \geq M$. This means we need at least a real number feature vector of size M for each element in the set in order to get any set output in the continuous domain. And the feature extraction function $\Theta(.)$ can be approximated by a neural network under the universal approximation theorem. The output length of the neural network should be at least M.

Note that the proof in [96] only indicates the existence of $\Theta(.)$, and it does not guarantee we can find a good approximation of $\Theta(.)$ during training. Also, when the edge features are less than M-dim, we may still get good enough aggregation results.

Max Decomposition. A similar approach is also proposed in PointNet [10], which replace the *sum* operation with *max*:

$$f(X) = \rho(\max_{x \in X} \Theta(x)) \tag{8.2}$$

The authors of [96] also showed that for a set of size N and $\Theta(.) \in \mathbb{R}^K, K < N$, some continuous permutation invariant function $f(X)$ can not be max-decomposed. This is consistent with sum-decomposition and indicates the hidden dimension K needs to increase when the size of a set increases.

When the hidden dimension K is small, PointNet [10] showed that the output value of $max(.)$ function relies on a set of critical points in the set. Other points in the set do not contribute to the output value. This may provide additional robustness. In our preliminary experiments of Point-GNN, we observe that max-decomposition often achieves better results than sum-decomposition.

Attention Mechanism. Similar to sum-decomposition, the attention mechanism com-

putes a weighted sum.

$$f(X) = \rho(\sum_{x \in X} att(x)\Theta(x)) \tag{8.3}$$

where $att(x)$ is the attention weights:

$$\begin{aligned} att(x) &\in [0,1] \\ \sum_{x \in X} att(x) &= 1 \end{aligned} \tag{8.4}$$

$att(.)$ adjusts the weights of each input element. In [97], $att(.)$ is computed as the normalized inner-product between the source vertex' feature vector and the destination vertex's feature vector. Although there is a lack of theoretical discussion, the representation power of the attention mechanism has been demonstrated in the recent development of transformers [98, 99]. It will be an interesting topic to use the attention mechanism as the aggregation function in Point-GNN instead of the current max-decomposition method.

8.3.4 Multi-scale Graph Feature

To detect objects of different scales, point cloud features of different scales are necessary. In a GNN, the receptive field of vertices increases with the number of GNN iterations. With more GNN iterations, a vertex can receive information from further away from neighbors via edges. Generally, point cloud features of large-scale need more GNN iterations than small-scale ones. A large object may need more GNN iterations, while a few GNN iterations may be sufficient for small objects. In the current Point-GNN, we handle the different sizes of objects construction multiple graphs using different radius values.

For large objects such as cars, we set a $4-meter$ radius to connect points. For smaller objects such as pedestrians and cyclists, we set a radius to $1.6meters$. There is no shared computation between two graphs, leading to double the computational cost. For a more cost-effective solution, we can compute features of different scales in the same graph.

In CNN-based object detection, sequential convolution layers of different spatial resolutions fuse and handle objects of different sizes [100, 73]. We can follow the same trend and construct a sequence of graphs that have different lengths of edges. Additional edges can be added between the vertices of different graphs to share their features. This architecture can potentially reduce the overall inference time for the detection of multi-size objects and also be beneficial to other tasks such as semantic segmentation [101], which requires multi-scale features.

8.3.5 Runtime Acceleration

Multiple factors affect the execution time of GNNs, including the neural network architecture, code optimizations, and computing hardware. We have discussed possible directions to improve the neural network architecture in the previous sections. However, there are also challenges to the acceleration of GNNs in the current computing environment. The most notable difference between a CNN and a GNN is that the GNN has a more irregular computational and memory pattern. In a CNN, the convolution operations take a fixed size of the input region and perform the same number of matrix multiplications. These matrix multiplications can be executed in parallel and have similar finishing times. However, vertices in a GNN have irregular connections to a varied number of vertices in varied memory locations. Different vertices need to aggregate different numbers of edge features, making

them difficult to finish simultaneously. It is possible to make the computation pattern more regular by padding dummy edges so that every vertex has the same maximum number of edges. However, these dummy edges increase the overall computational cost. In our Point-GNN implementation, we use a scatter-gather method to scatter vertices features to each edge first and then gather the computed edge features back to the vertices. Although our approach does not need the computation for dummy edges, it still has an irregular memory pattern, and the gathering for maximum value is not optimized to use GPUs. We plan to use GPU implementation for a parallel gathering of maximum values and reduce the memory footprint by placing edges sharing the same vertices closer in memory for future optimization.

8.3.6 Open Questions for a Camera-Lidar Fusion

Calibration Error. Sensor calibration is the prerequisite of sensor fusion. The calibration process computes parameters regard to the sensing process and adjusts the data as necessary. There are two types of parameters: intrinsic and extrinsic parameters. The intrinsic parameters are the internal parameters belong each sensor. They can typically be parameters that each sensor can be individually calibrated. The extrinsic parameters are the position of each sensor. In the fusion of camera-lidar, the errors of both intrinsic and extrinsic parameters can induce a mismatch between the point clouds and image pixels. In other words, the projected point clouds in the image panel may not overlap with the actual objects. It is an open question of how to design a fusion algorithm that is robust to such misalignment between point clouds and images. One potential solution is also to apply machine learning techniques to learn and fine-tune the calibration errors on-the-fly.

Occlusion. Even with a perfect calibration between lidars and cameras, there are still challenges if the centers of lidars and cameras are not alignment. Due to the misalignment between the lidar center and camera center, some lidar points are occluded by the image's forefront object. The lidar points may be at places that a camera can not observe. If we project the lidar points to the image plane using the camera-lidar calibration, the points look like they are from the forefront object.s Adding each point's depth as an input feature to the network reduces the error caused by such occlusion.

8.3.7 Open Questions for Self-supervised Learning

The goal of self-supervised training is to utilize unlabeled data. The pretraining performance is related to the alignment between the pretext task and the fine-tuning task. If the pretraining task is not well-aligned with the perception task of interest, the results cannot be beneficial or even damaging to the final performance. As argued in [48], a pretraining of a different task may limit the final task accuracy in the end. Our experiments found that some geometrical prediction tasks can benefit from pretraining. However, given enough training time, the benefits from pretraining also decrease. Although searching for the right pretext task for point cloud perception is still of great interest, it may not be the best approach. Alternative methods such as self-training, semi-supervised learning, or even a strong augmentation policy may also achieve good results.

8.3.8 Open Questions for Work Zone Recognition

Non-convex Boundary Detection. Our work zone boundary detection baselines utilize a convex hull algorithm to generate a convex frontier of work zones. Although the algorithm

is simple and fast, a convex contour may not approximate the real boundary well. For fine-grained curve recognition, we may need more general line representations such as polylines or B-splines. Similar to that in the field of lane detection or pose recognition, we can predict lines using deep neural networks. For example, we may predict a vector field that connects the points on the work zone boundary.

Detection without Traffic Objects. A work zone is usually morphable and surrounded by traffic objects such as traffic cones and barrels. Our baseline algorithms take advantage of this feature for boundary recognition. However, in the case of a temporal work zone without traffic objects, for example, a crash site, our baselines may not work. In theory, we can utilize image segmentation methods and learn the boundary from the data. Nevertheless, the enormous possibilities of work zones are difficult to be fully captured. It is an open question on how to detect work zones with limited data.

8.3.9 Open Questions for Traffic Gesture Recognition

Realistic Dataset. Our flagmen's gestures dataset facilitates the study of traffic gesture recognition. However, our gesture dataset only covers a relatively small portion of the complex real-world gestures. It has three notable limitations. Firstly, the gestures are recorded from a static position in front of the flagmen. However, a real flagman may stay off the road for safety. Therefore, the vehicle may observe the gesture sideway. The vehicle also moves, which changes the observational angle and size consistently. The motion of vehicles may also induce artifacts such as motion blurs, which our static motion dataset does not capture. Secondly, the gestures are recorded with an indoor green screen instead of a real-world street view. Although this is a desired feature to insert virtual background

and enable data augmentation, it may miss realistic outdoor details such as natural lighting. The results from the validation set may not fully reflect that of real-world tests. Thirdly, our current flagman gesture dataset only contains a single flagman. We assume that only one flagman can perform traffic gestures. This assumption limits our dataset for application to more complex scenarios where multiple agents are guiding the traffic. To build a more realistic dataset than our current one, we need to set up data collection in an outdoor test ground, ideally on real streets. The camera needs to be mounted on a moving platform. Furthermore, multiple flagmen may be present. Such data collection is challenging from both a cost and a safety perspective. Alternatively, we can leverage the advance in 3D simulation tools to build virtual flagmen and street views. How to make the simulation videos identical to street recordings is still actively being researched.

Recognition Distance. The prerequisite of gesture recognition is the observation and detection of human bodies. When a flagman is far away from the vehicle's camera, the person's image may be very small. The fast and accurate detection of tiny objects has been an active research area. In general, detecting tiny objects requires high-resolution cameras and much computational power to search through the images. The recognition of subtle gestures such as finger moving requires detecting even smaller objects. Other than brute force, a possible solution is to use thermal cameras for fast human detection and then zoom in using RGB cameras.

Multi-instruction to Multi-vehicle (MI2MV) Problem. It is common for a flagman to have different signals to different vehicles. For example, a flagman may allow vehicles in one lane to proceed while keeping the other vehicles stopped. This Multi-instruction to Multi-vehicle (MI2MV) problem adds one new dimension to the traffic gesture recognition

problem: the vehicle needs to know whether it is the recipient of one or more instructions while recognizing instructions. MI2MV is a challenging problem where the vehicle needs to consider the road map and the existence of other vehicles. These complex scenarios are difficult to collect from the real world. It is also yet unclear how to automatically generate simulated data.

Bibliography

[1] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6526–6534, July 2017.

[2] B. Yang, W. Luo, and R. Urtasun. Pixor: Real-time 3d object detection from point clouds. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, June 2018.

[3] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, June 2018.

[4] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[5] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[6] Pei Sun, Weiyue Wang, Yuning Chai, Gamaleldin Elsayed, Alex Bewley, Xiao Zhang, Cristian Sminchisescu, and Dragomir Anguelov. Rsn: Range sparse net for efficient, accurate lidar 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5725–5734, June 2021.

[7] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. *CVPR*, 2018.

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

[9] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 918–927, June 2018.

[10] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, July 2017.

[11] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3391–3401. Curran Associates, Inc., 2017.

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

[13] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Std: Sparse-to-dense 3d object detector for point cloud. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[14] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *arXiv e-prints*, page arXiv:1901.00596, Jan 2019.

[15] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun. 3d graph neural networks for rgbd semantic segmentation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5209–5218, Oct 2017.

[16] Yin Bi, Aaron Chadha, Alhabib Abbas, Eirina Bourtsoulatze, and Yiannis Andreopoulos. Graph-based object classification for neuromorphic vision sensing. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[17] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019.

[18] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[19] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset, 2019.

[20] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[21] Sourabh Vora, Alex H. Lang, Bassam Helou, and Oscar Beijbom. Pointpainting: Sequential fusion for 3d object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[22] Charles R. Qi, Xinlei Chen, Or Litany, and Leonidas J. Guibas. Imvotenet: Boosting 3d object detection in point clouds with image votes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[23] Level 5 dataset https://level-5.global/data/, Jul 2021.

[24] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[25] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. Big self-supervised models are strong semi-supervised learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22243–22255. Curran Associates, Inc., 2020.

[26] Chunyuan Li, Jianwei Yang, Pengchuan Zhang, Mei Gao, Bin Xiao, Xiyang Dai, Lu Yuan, and Jianfeng Gao. Efficient self-supervised vision transformers for representation learning. *arXiv preprint arXiv:2106.09785*, 2021.

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[28] Saining Xie, Jiatao Gu, Demi Guo, R. Charles Qi, J. Leonidas Guibas, and Or Litany. Pointcontrast: Unsupervised pre-training for 3d point cloud understanding. *european conference on computer vision*, pages 574–591, 2020.

[29] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):677–691, 2017.

[30] K. Hara, H. Kataoka, and Y. Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6546–6555, 2018.

[31] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 7444–7452. AAAI Press, 2018.

[32] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[33] Weijing Shi and Ragunathan (Raj) Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[34] Ao Luo, Xin Li, Fan Yang, Zhicheng Jiao, Hong Cheng, and Siwei Lyu. Cascade graph neural networks for rgb-d salient object detection. In *In 16th European Conference on Computer Vision (ECCV)*, 2020.

[35] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2147–2156, 2016.

[36] A. Mousavian, D. Anguelov, J. Flynn, and J. Košecká. 3d bounding box estimation using deep learning and geometry. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5632–5640, 2017.

[37] Garrick Brazil and Xiaoming Liu. M3d-rpn: Monocular 3d region proposal network for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, Seoul, South Korea, 2019.

[38] Jason Ku, Alex D. Pon, and Steven L. Waslander. Monocular 3d object detection leveraging accurate proposals and shape reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[39] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *CVPR*, 2019.

[40] Mingyu Ding, Yuqi Huo, Hongwei Yi, Zhe Wang, Jianping Shi, Zhiwu Lu, and Ping Luo. Learning depth-guided convolutions for monocular 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11672–11681, 2020.

[41] Yurong You, Yan Wang, Wei-Lun Chao, Divyansh Garg, Geoff Pleiss, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving. In *ICLR*, 2020.

[42] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. In *ECCV*, 2016.

[43] Richard Zhang, Phillip Isola, and Alexei A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 645–654, 2017.

[44] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2536–2544, 2016.

[45] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *ECCV*, 2016.

[46] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1422–1430, 2015.

[47] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *CoRR*, abs/1803.07728, 2018.

[48] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin Dogus Cubuk, and Quoc Le. Rethinking pre-training and self-training. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[49] Nanxuan Zhao, Zhirong Wu, Rynson W.H. Lau, and Stephen Lin. Distilling localization for self-supervised representation learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):10990–10998, May 2021.

[50] Jonathan Sauder and Bjarne Sievers. Self-supervised deep learning on point clouds by reconstructing space. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[51] Benjamin Eckart, Wentao Yuan, Chao Liu, and Jan Kautz. Self-supervised learning on 3d point clouds by learning discrete generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8248–8257, June 2021.

[52] Omid Poursaeed, Tianxing Jiang, Han Qiao, Nayun Xu, and Vladimir Kim. Self-supervised learning of point clouds via orientation estimation. pages 1018–1028, 11 2020.

[53] F. Abodo, R. Rittmuller, B. Sumner, and A. Berthaume. Detecting work zones in shrp 2 nds videos using deep learning based computer vision. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 679–686, 2018.

[54] Y. Seo, J. Lee, W. Zhang, and D. Wettergreen. Recognition of highway workzones for reliable autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):708–718, 2015.

[55] B. Mathibela, M. A. Osborne, I. Posner, and P. Newman. Can priors be trusted? learning to anticipate roadworks. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 927–932, 2012.

[56] P. Kunz and M. Schreier. Automated detection of construction sites on motorways. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1378–1385, 2017.

[57] R. Graf, A. Wimmer, and K. C. J. Dietmayer. Probabilistic estimation of temporary lanes at road work zones. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 716–721, 2012.

[58] Surabhi Gupta, Maria Vasardani, and Stephan Winter. Conventionalized gestures for the interaction of people in traffic with autonomous vehicles. In *Proceedings of the 9th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '16, page 55–60, New York, NY, USA, 2016. Association for Computing Machinery.

[59] P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4207–4215, 2016.

[60] Quentin De Smedt, Hazem Wannous, Jean-Philippe Vandeborre, Joris Guerry, Bertrand Le Saux, and David Filliat. SHREC'17 Track: 3D Hand Gesture Recognition Using a Depth and Skeletal Dataset. In I. Pratikakis, F. Dupont, and M. Ovsjanikov, editors, *3DOR - 10th Eurographics Workshop on 3D Object Retrieval*, pages 1–6, Lyon, France, April 2017.

[61] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The Kinetics Human Action Video Dataset. *arXiv e-prints*, page arXiv:1705.06950, May 2017.

[62] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. *arXiv e-prints*, page arXiv:1212.0402, December 2012.

[63] Julian Wiederer, Arij Bouazizi, Ulrich Kressel, and Vasileios Belagiannis. Traffic control gesture recognition for autonomous vehicles. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)October 25-29, 2020, Las Vegas, NV, USA (Virtual)*, 2020.

[64] F. Guo, Z. Cai, and J. Tang. Chinese traffic police gesture recognition in complex scene. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1505–1511, 2011.

[65] Jon L. Bentley, Donald F. Stanat, and E.Hollins Williams. The complexity of finding fixed-radius near neighbors. *Information Processing Letters*, 6(6):209 – 212, 1977.

[66] Peter J. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1):73–101, 03 1964.

[67] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[68] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[69] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander. Joint 3d proposal generation and object detection from view aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, Oct 2018.

[70] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[71] Yilun Chen, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Fast point r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[72] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[73] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[74] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[75] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced Grouping and Sampling for Point Cloud 3D Object Detection. *arXiv e-prints*, page arXiv:1908.09492, Aug 2019.

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

[77] A. Simonelli, S. Rota Bulo, L. Porzi, M. Lopez Antequera, and P. Kontschieder. Disentangling monocular 3d object detection: From single to multi-class recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020.

[78] MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. https://github.com/open-mmlab/mmdetection3d, 2020.

[79] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[80] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page III–1310–III–1318. JMLR.org, 2013.

[81] M. Tan, R. Pang, and Q. V. Le. Efficientdet: Scalable and efficient object detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, 2020.

[82] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[83] Hanspeter Mallot, Heinrich Bülthoff, J.J. Little, and S Bohrer. Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological cybernetics*, 64:177–85, 02 1991.

[84] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[85] Chunyong Ma, Yu Zhang, Anni Wang, Yuan Wang, and Ge Chen. Traffic command gesture recognition for virtual urban scenes based on a spatiotemporal convolution neural network. *ISPRS International Journal of Geo-Information*, 7(1):37, 2018.

[86] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[87] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[88] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv e-prints*, page arXiv:1704.04861, April 2017.

[89] Weijing Shi and Ragunathan Raj Rajkumar. Boosting sparse point cloud object detection via image fusion. In *2021 IEEE Third International Conference on Cognitive Machine Intelligence (CogMI)*, pages 214–220, 2021.

[90] Weijing Shi and Ragunathan Raj Rajkumar. Self-supervised pretraining for point cloud object detection in autonomous driving. In *2022 IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2022.

[91] Weijing Shi and Ragunathan Raj Rajkumar. Work zone detection for autonomous vehicles. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 1585–1591, 2021.

[92] Weijing Shi, Ragunathan Rajkumar, and Eran Kishon. Opportunities and challenges for flagman recognition in autonomous vehicles. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 1470–1477, 2021.

[93] Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam

Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018.

[94] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[95] Yu Chen, Lingfei Wu, and Mohammed Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems*, 33, 2020.

[96] Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A. Osborne. On the limitations of representing functions on sets. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6487–6494. PMLR, 09–15 Jun 2019.

[97] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[98] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need.

In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[99] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[100] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing.

[101] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834—848, April 2018.