Learning 3D Representations from Data

by

Yue Wang

B.Eng., Zhejiang University (2015)
M.S., University of California, San Diego (2016)
S.M., Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author .....
Department of Electrical Engineering and Computer Science
May 13, 2022

Certified by .....
Justin M. Solomon
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Learning 3D Representations from Data

by

Yue Wang

Submitted to the Department of Electrical Engineering and Computer Science
on May 13, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Deep learning has achieved tremendous progress and success in processing images and natural languages. Deep models enable human-level perception, photorealistic image generation, and conversational language understanding. Despite significant progress, existing deep models still fail to meet the demands of robotics. There are several factors leading to this gap. First, existing computer vision algorithms have been primarily targeted to 2D images. These algorithms are extremely good at recognizing objects in an image, but they fail to reason about 3D geometry. Second, the current success in the 2D domain is mainly due to the advance in convolutional neural networks (CNNs). However, CNNs do not generalize to arbitrary data modalities such as point clouds. Finally, 3D annotations are scarce and hard to obtain. Annotating 3D data usually requires more human effort, which hinders supervised learning from 3D data. Therefore, learning 3D representations from data remains challenging and demands further study.

This thesis investigates how to learn representations from 3D data efficiently and effectively. This thesis aims to design 3D learning algorithms that understand geometry with minimal supervision. First, we proposed a general point cloud network , termed Dynamic Graph Convolutional Neural Networks (DGCNN), to learn a latent structure from sensory inputs. The induced structure will improve feature learning from point clouds. Unlike prior works that focus on global features, DGCNN views local geometry as the key to point cloud feature learning. Second, we study using DGCNN to enable high-level semantic reasoning tasks such as shape segmentation and 3D object detection. To that end, we propose a multi-view based object detection model that learns complementary features by projecting point clouds to virtual views. In addition, our follow-up work Object DGCNN leverages DGCNN to model object relations and empowers a post-processing free object detection pipeline with state-of-the-art performances on multiple benchmarks. Third, we generalize these point cloud models to tackle low-level motion estimation problems such as point cloud registration. The proposed Deep Closest Point architecture combines a traditional optimization pipeline with deep learning. Moreover, Partial Registration Network (PRNet) uses shape registration as a proxy task to enable self-supervised learning

from point clouds in a subsequent work. Finally, this thesis allows for a critical application – scene understanding for autonomous driving. These studies collectively facilitate 3D deep learning in a broad range of scenarios in visual computing.

Thesis Supervisor: Justin M. Solomon
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

"Getting an education from MIT is like taking a drink from a fire hose." My five years at MIT were an incredible journey. I feel incredibly fortunate to become a member of the MIT family, which I have dreamed of since childhood. This is the most rewarding and splendid experience of my life. I met the best advisor, made life-long friends, pushed the research boundary of my field a bit, and experienced completely different cultures all here at MIT. I still remember when I accepted the offer from MIT, I questioned whether I would regret pursuing a PhD rather than following other paths. Now I can firmly say that regret didn't happen. This is the most correct choice I've made in my life. I thank MIT for offering me such a great opportunity.

My deepest thanks go to my advisor Prof. Justin Solomon. I can't express how much I feel grateful to him. Justin supported me throughout the whole journey with his kindness, patience, and enthusiasm for everything that happens in our group. I started with zero experience in research and still remember how I felt puzzled and stressed at the beginning of my PhD study. Justin taught me how to read a paper, formulate a research problem, and write a paper from scratch. Mostly importantly, Justin never pushed me to finish a project but instead encouraged me to focus on the problems that interested me. I can never think of a better advisor than Justin. Without him, I wouldn't be able to love research. Justin is also my role model in life. He deeply cares about the members of our group and the people around us. He always influences us with his attitude to life and his endeavor to work. I sincerely thank Justin for holding me up for these five years. This journey wouldn't be so meaningful without his guidance.

I want to express my sincere gratitude to Prof. Michael Bronstein. Michael is another important person in my PhD study. His mentorship significantly shaped my research direction and laid a solid foundation for my whole PhD. My first project, DGCNN, which is my most significant work, wouldn't be possible without Michael's guidance. I will never forget how those fruitful discussions in each week's one-on-one meeting motivated me to stick to geometric deep learning. In addition to research

discussions, Michael also cares about my career development. I remember Michael shared the news of the Nvidia fellowship and encouraged me to apply even though the deadline was approaching. Moreover, Michael connected me with many other researchers and taught me how to collaborate with others. I look forward to many more interactions with Michael.

I am also very grateful to Prof. Phillip Isola. I have never seen such a great person as Phil. Phil is the most gentle and affable professor I've ever known. Every moment I talk with Phil, I feel like I am bathing in a spring breeze. I am highly jealous of my PhD fellows who are students of Phil. Phil is also a visionary researcher. He always provides us with new perspectives on the research problems we study. Phil's research style thoroughly influences me. I wish through years of learning, I could have just one inch of Phil's breadth and depth of knowledge and become a person like Phil in all aspects. Thank you, Phil!

I would also like to thank Prof. Leonidas Guibas, who is also a dissertation committee member. Leo is a pioneer in many fields of computer science. Leo's research ranges from algorithms to computer graphics and vision. I don't know how a professor can be so successful in so many fields. Although I did not interact directly with Leo much, his research impacts every corner of my papers. My research started based on the PointNet paper from Leo's lab. This paper completely changed my view of point cloud processing with deep learning. I thank Leo for pioneering our research field.

My PhD study wouldn't have been so much fun without being around people from the Geometric Data Processing group. I was fortunate to begin my PhD with Paul Zhang, David Palmer, and Dima Smirnov in the same year. I also feel extremely lucky to have interactions with many intelligent lab mates (in approximately chronological order): Mieke Moran, Anne Gehre, Mikhail Bessmeltsev, Sebastian Claici, Yu Wang, Edward Chien, Eva Agapaki, Lawrence Stewart, Richard Barnes, Hugo Lavenant, Francesca Matrone, Nicolas Girard, Florian Cyril Stutz, Daryl DeFord, Charlie Frogner, Kevin Shao, Aude Genevay, Mikhail Yurochkin, Kristjan Greenewald, Mazdak Abulnaga, Lingxiao Li, Rickard Brüel Gabrielsson, James Xiu, Leti-

cia Mattos Da Silva, Oded Stein, Tal Shnitzer, Artem Lukoianov, and Zoë Marschner. I am also thankful to everyone in the Vision Graphics Neighborhood. I enjoyed every small chat we had in our neighborhood.

My research couldn't be going so smoothly without my great collaborators, including (in no particular order): Yongbin Sun, Ziwei Liu, Sanjay Sarma, Josh Tenenbaum, Dilip Krishnan, Alireza Fathi, Hang Zhao, David Ross, Abhijit Kundu, Caroline Pantofaru, Tom Funkhouser, Jiajun Wu, Jiahui Fu, Lu Mi, Tianxing He, Core Francisco Park, Hao Wang, Chris Choy, Benjamin Eckart, Yilun Wang, Vitor Guizilini, Xiangru Huang, Tianyuan Zhang, Rares Ambrus, and Adrien Gaidon.

Besides those already mentioned, I made many life-long friends who shared pleasure and distress with me through this journey: Sihan Liu, Yonglong Tian, Macheng Shen, Hank Yang, Jun Yin, Weishun Zhong, Xiuming Zhang, Zhoutong Zhang, Lei Xu, Ge Liu, Shaoxiong Wang, Bai Liu, Rui Guo, Hanrui Wang, Pingchuan Ma. Thanks to all of you for making my MIT life colorful.

Finally, I want to thank my parents Aihua Han and Jiaomin Wang for their unswerving love and support. They have always been supporting me soundlessly and countlessly throughout my life. Although they never went to college, they deeply understand how important education is. They encouraged me to seize every opportunity to pursue higher education. I would have joined industry and not bothered to apply to MIT without their inspiration. They are proud of my every triumphant moment, and they share my distress when facing challenges. They always manage to cheer me up when bad things happen and calm me down when I am on cloud nine. As the single child in my family, I wish I could keep gone back home more and stayed longer with my parents during my PhD. I am deeply indebted to my parents. Thank you Ma-ma and Ba-ba! I love you and will show you that we have made the right calls.

To my beloved mom and dad.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Deep learning has demonstrated considerable success embedding images and more general 2D representations into compact feature spaces for downstream tasks like recognition [49, 62, 48, 61], segmentation [107], and generation [54, 251]. This success is largely attributed to the reviving of convolutional neural networks (CNNs) [62], which enforces appropriate inductive biases for grid-like data such as images. In addition to plain CNN models, many variant architectures and training algorithms have been proposed to improve the efficiency and generalization of current deep learning models. All these progress in 2D deep learning jointly enable applications in medical imaging, visual search, and smart transportation.

Despite this progress, learning from 3D data is still the missing piece for embodied agents to perceive their surrounding environments. For example, autonomous driving cars rely on LiDAR sensors to capture point clouds and performs scene understanding to recognize objects and pedestrians. 2D deep learning methods, although good at identifying objects from images, fail to process this non-grid-like data and yield poor performance in 3D scene understanding. To bridge the gap between deep learning and 3D scene understanding, this thesis focuses on learning 3D representations from data with minimal supervision. It investigates how to incorporate insights from geometry processing, computer vision, and machine learning into algorithmic/modeling designs.

Also, works included in this thesis facilitate robotic applications such as autonomous driving and virtual reality.

There are three significant challenges in learning-based 3D understanding. First, 3D data is entirely different from 2D data. 2D data such as images is typically parameterized on 2D grids, with fixed orders and structures. 3D data, however, is usually represented sparsely. For example, point clouds are parameterized by orderless sets, which break the fundamental assumption required by CNNs. Identical shapes represented by points in different orders are transformed into different feature representations by CNNs. This construction requires CNNs to learn in a permutation space, hampering their generalization to unseen data. Second, 3D scene understanding is inherently more challenging than 2D recognition. These tasks require predicting more attributes of the geometry of surrounding environments. Third, 3D data is hard to annotate, although it is easy to acquire. As long as autonomous vehicles drive on the road, they can capture unlimited point clouds. Nevertheless, annotating these point clouds with semantic labels demands substantial human effort.

To tackle these mutually-reinforced challenges, I study three major topics in this thesis. First, we develop a general architecture, Dynamic Graph CNNs (DGCNN) [215], to learn features from point clouds. This is the first attempt to combine graph-based operations with deep learning to process point clouds. DGCNN takes point clouds as input and reconstructs latent graphs to enable feature learning. The latent graphs are approximated by the K nearest neighbor (KNN) algorithm. On the KNN graphs, features are defined on edges. Then, shared multi-layer perceptrons (MLPs) are employed to transform edge features to a high dimension space, followed by permutation-invariant functions to bring features from edges to points. After obtaining per-point features, we construct new KNN graphs based on the latest features. DGCNNs alternate between feature learning and graph reconstruction. Moreover, DGCNN generalizes prior works including PointNet [141], PointNet++ [143], and Transformer [200]. DGCNNs shed new light on point cloud processing with graph deep learning.

Second, we use these point cloud networks to solve perception problems. For example, DGCNN enables a broad range of vision applications, including both high-

level semantic reasoning tasks and low-level motion estimation tasks. DGCNN is immediately applicable to shape classification and part segmentation in the high-level regime. Also, DGCNN is used to model object relations in 3D object detection. Bounding box proposals in a scene form a set of points that DGCNN can directly process. DGCNN provides each bounding box proposal with richer contextual information. This architecture is called Object DGCNNs [214]. Beyond point cloud object detection, this architecture is adapted to address 3D object detection from multi-view images. Learning 3D attributes from multi-view images is more challenging because depth information is missing. In my work DETR3D [211], we leverage a similar idea to Object DGCNN to formulate 3D object detection as a sparse set prediction problem. These methods do not count on post-processing steps while maintaining state-of-the-art performance on multiple benchmarks.

In addition to high-level semantic reasoning tasks, DGCNN helps low-level motion estimation tasks. Point cloud registration is a long-standing problem in computer vision, computer graphics, and robotics. This problem is usually tackled by optimization-based approaches that involve hand-crafting features and estimating correspondences. These algorithms do not have learnable parameters and do not learn any prior from data. To equip these algorithms with deep learning, we propose a pipeline termed Deep Closest Point (DCP) [212]. DCP uses DGCNN to encode point clouds into feature space. In feature space, point correspondences are estimated based on feature similarities. Then, DCP solves a rigid motion problem to align point clouds. DCP is fully end-to-end and can learn matching priors from data directly.

To improve the efficiency of feature learning from point clouds, we introduce a pillar-based feature extractor [209]. This model projects point clouds onto multiple 2D planes. In each 2D plane, view-dependent features are learned with CNNs. After that, features are transformed back to 3D space and aggregated. This model can be plugged into any point cloud learning pipeline with minimal modification.

Finally, we study how to leverage geometry as supervision to alleviate the data problem. This thesis presents a pipeline called Partial Registration Network (PR-

Net) [213], which followed the DCP architecture. PRNet solves general partial-to-partial registration problems. Most importantly, PRNet enables self-supervised learning by transferring the low-level features learned from point cloud registration to high-level semantic reasoning tasks. PRNet significantly reduces the amount of data annotations required by semantic reasoning tasks.

1.2 Thesis Structure

This thesis investigates each of the directions mentioned earlier and their associated methods. In Chapter 2, I present Dynamic Graph CNNs [215], which lays the foundation of my research. In Chapter 3, 4, and 5, I present methods that are designed to tackle high-level semantic reasoning tasks. These chapters include a Pillar-based object detector [209], Object DGCNN [214], and DETR3D [211]. These methods tackle 3D object detection from different perspectives. In Chapter 6, we introduce the method Deep Closest Point [212], which combines deep learning with Iterative Closest Point to solve low-level motion estimation problems. We show that this learning-based method significantly outperforms its non-learning counterparts in various settings. Chapter 7 introduces a partial registration network [213]. It tackles general partial-to-partial point cloud registration problems. In addition, PRNet enables self-supervised learning from point clouds. Finally, Chapter 8 summarizes contributions made by this thesis and suggests several opportunities for future inquiry.

Chapter 2

Dynamic Graph CNNs

Point clouds, or scattered collections of points in 2D or 3D, are arguably the simplest shape representation; they also comprise the output of 3D sensing technologies including LiDAR scanners and stereo reconstruction. With the advent of fast 3D point cloud acquisition, recent pipelines for graphics and vision often process point clouds directly, bypassing expensive mesh reconstruction or denoising due to efficiency considerations or instability of these techniques in the presence of noise. A few of the many recent applications of point cloud processing and analysis include indoor navigation [254], self-driving vehicles [140], robotics [156], and shape synthesis and modeling [53]. This chapter primarily addresses the representations of point clouds – how to extract features from raw point clouds with deep learning architectures.

2.1 Introduction

Modern applications demand *high-level* processing of point clouds. Rather than identifying salient geometric features like corners and edges, recent algorithms search for semantic cues and affordances. These features do not fit cleanly into the frameworks of computational or differential geometry and typically require learning-based approaches that derive relevant information through statistical analysis of labeled or unlabeled datasets.

In this chapter, we primarily consider point cloud classification and segmentation,

two model tasks in the point cloud processing world. Traditional methods for solving these problems employ handcrafted features to capture geometric properties of point clouds [111, 154, 155]. More recently, the success of deep neural networks for image processing has motivated a data-driven approach to learning features on point clouds. Deep point cloud processing and analysis methods are developing rapidly and outperform traditional approaches in various tasks [27].

Adaptation of deep learning to point cloud data, however, is far from straightforward. Most critically, standard deep neural network models take as input data with regular structure, while point clouds are fundamentally irregular: Point positions are continuously distributed in the space, and any permutation of their ordering does not change the spatial distribution. One common approach to process point cloud data using deep learning models is to first convert raw point cloud data into a volumetric representation, namely a 3D grid [118, 218]. This approach, however, usually introduces quantization artifacts and excessive memory usage, making it difficult to go to capture high-resolution or fine-grained features.

State-of-the-art deep neural networks are designed specifically to handle the irregularity of point clouds, directly manipulating raw point cloud data rather than passing to an intermediate regular representation. This approach was pioneered by *PointNet* [141], which achieves permutation invariance of points by operating on each point independently and subsequently applying a symmetric function to accumulate features. Various extensions of PointNet consider neighborhoods of points rather than acting on each independently [143, 162]; these allow the network to exploit local features, improving upon performance of the basic model. These techniques largely treat points independently at local scale to maintain permutation invariance. This independence, however, neglects the geometric relationships among points, presenting a fundamental limitation that leads to local features missing.

To address these drawbacks, we propose a novel simple operation, called EdgeConv, which captures local geometric structure while maintaining permutation invariance. Instead of generating points' features directly from their embeddings, EdgeConv generates *edge features* that describe the relationships between a point and its neigh-

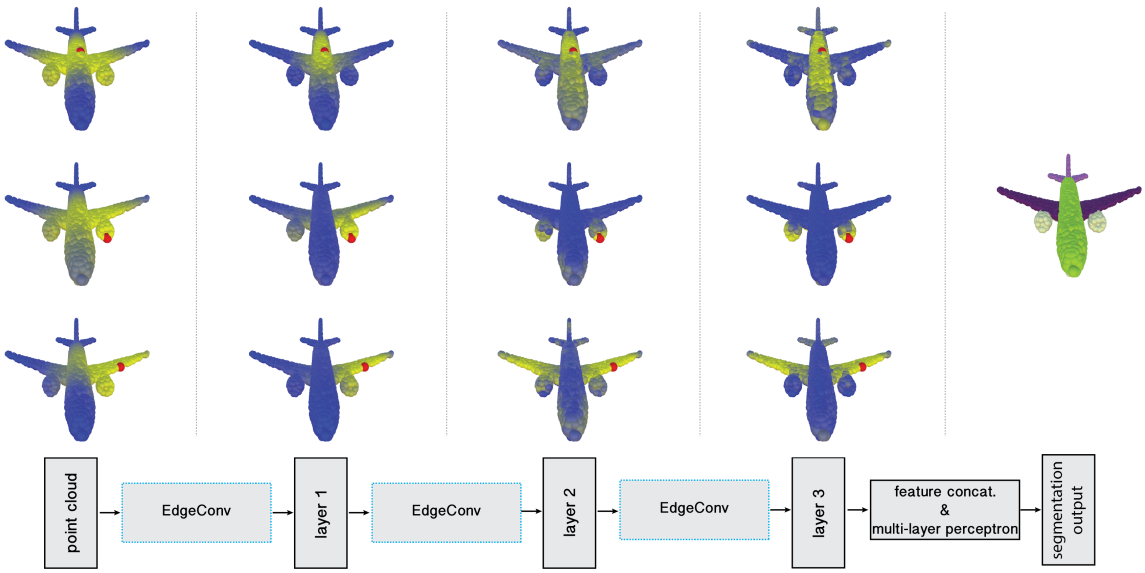


Figure 2-1: **Point cloud segmentation using the proposed neural network.** Bottom: schematic neural network architecture. Top: Structure of the feature spaces produced at different layers of the network, visualized as the distance from the red point to all the rest of the points (shown left-to-right are the input and layers 1-3; rightmost figure shows the resulting segmentation). Observe how the feature space structure in deeper layers captures semantically similar structures such as wings, fuselage, or turbines, despite a large distance between them in the original input space.

bors. EdgeConv is designed to be invariant to the ordering of neighbors, and thus permutation invariant.

EdgeConv is easy to implement and integrate into existing deep learning models to improve their performance. In our experiments, we integrate EdgeConv into the basic version of *PointNet* without using any feature transformation. We show performance improvement by a large margin; the resulting network achieves state-of-the-art performance on several datasets, most notably *ModelNet40* and *S3DIS* for classification and segmentation.

Key Contributions. We summarize the key contributions of our work as follows:

- We present a novel operation for point clouds, EdgeConv, to better capture local geometric features of point clouds while still maintaining permutation invariance.

- We show the model can learn to semantically group points by dynamically updating the graph.
- We demonstrate that EdgeConv can be integrated into multiple existing pipelines for point cloud processing.
- We present extensive analysis and testing of EdgeConv and show that it achieves state-of-the-art performance on benchmark datasets.

2.2 Related Work

In this section, we introduce related works in learning for geometry processing. First, we talk about approaches that use hand-craft features in 2.2.1. Then, we discuss learning-based methods in 2.2.2. Next, we describe view-based methods in 2.2.3 and volumetric methods in 2.2.4 in detail. Also, we present a very popular work – PointNet and its follow-ups in 2.2.5. Finally, we give a overview of a recent research focus – geometric deep learning in 2.2.6.

2.2.1 Hand-Crafted Features

Various tasks in geometric data processing and analysis — including segmentation, classification, and matching — require some notion of local similarity between shapes. Traditionally, this similarity is established by constructing feature descriptors that capture local geometric structure. Countless papers in computer vision and graphics propose local feature descriptors for point clouds suitable for different problems and data structures. A comprehensive overview of hand-designed point features is out of the scope of this chapter, but we refer the reader to [198, 57, 14] for comprehensive discussion.

Broadly speaking, one can distinguish between *extrinsic* and *intrinsic* descriptors. Extrinsic descriptors usually are derived from the coordinates of the shape in 3D space and includes classical methods like shape context [11], spin images [75], integral features [114], distance-based descriptors [99], point feature histograms [155, 154],

and normal histograms [194], to name a few. Intrinsic descriptors treat the 3D shape as a manifold whose metric structure is discretized as a mesh or graph; quantities expressed in terms of the metric are by definition intrinsic and invariant to isometric deformation. Representatives of this class include spectral descriptors such as global point signatures [153], the heat and wave kernel signatures [180, 5], and variants [19]. Most recently, several approaches wrap machine learning schemes around standard descriptors [57, 161].

2.2.2 Learned Features

In computer vision, approaches relying on 'hand-crafted' features have reached a plateau in performance on challenging image analysis problems like image recognition. A breakthrough came with the use of convolutional neural networks (CNNs) [90, 83], leading to an overwhelming trend to abandon hand-crafted features in favor of models that learn task-specific features from data.

A basic CNN architecture is the *deep neural network*, which interleaves convolutional and pooling layers to aggregate local information in images. This success of deep learning for images suggests the value of adapting related insight to geometric data like point clouds. Unlike images, however, geometric data usually are not on an underlying grid, requiring new definitions for building blocks like convolution and pooling.

Existing 3D deep learning methods can be split into two classes. View-based and volumetric representations exemplify techniques that try to "place" geometric data onto a grid and apply existing deep learning algorithms to the adapted structure. Other methods replace the standard building blocks of deep neural architectures with special operations suitable for unstructured geometric data [117, 16, 123, 141, 143]. We provide details about the closest techniques to ours below.

2.2.3 View-based Methods

View-based techniques represent a 3D object as a collection of 2D views, to which standard CNNs used in image analysis can be applied. Typically, a CNN is applied to each view and then the resulting features are aggregated by a view pooling procedure [178]. View-based approaches are also good match for applications where the input comes from a 3D sensor and represented as a range image [216], in which case a single view can be used.

2.2.4 Volumetric Methods

Voxelization is a straightforward way to convert unstructured geometric data to a regular 3D grid over which standard CNN operations can be applied [118, 218]. These volumetric representations are often wasteful, since voxelization produces a sparsely-occupied 3D grid. Time and space complexity considerations limit the resolution of the volumetric grids, yielding quantization artifacts. Recent space partition methods like k-d trees [82] or octrees [189] remedy some resolution issues but still rely on subdivision of a bounding volume rather than local geometric structure. Finally, Qi et al. [142] studied a combination of view-based and volumetric approaches for 3D shape classification.

2.2.5 PointNets

PointNets [141] comprise a special class of architectures for point sets like 3D point clouds. The key ingredient is a symmetric function applied to 3D coordinates in a manner invariant to permutation. While they achieve impressive performance on point cloud analysis tasks, PointNets treat each point individually, essentially learning a mapping from 3D to the latent features without leveraging local geometric structure. Furthermore, the learned mapping is sensitive to the global transformation of the point cloud; to cope with this issue, PointNet employs a complex and computationally expensive spatial transformer network [73] to learn 3D alignment.

Local information is important for feature learning in two ways. First, as for

handcrafted descriptors, local features usually account for geometric relationships among neighboring points to be robust to various transformations. Second, local information is critical to the succcess of image-based deep convolutional architectures. Follow-up work proposed an improved PointNet++ architecture exploiting geometric features in local point sets and hierarchically aggregating them for inference [143]. A similar approach is proposed in [162], where initial point features are obtained from a point kernel correlation layer and then aggregated among nearby points. Benefiting from local structure, PointNet++ achieves state-of-the-art results on several point cloud analysis benchmarks. PointNet++, however, still treats individual points in local point sets independently and does not consider relationships between point pairs.

2.2.6 Geometric Deep Learning

PointNets exemplify a broad class of deep learning architectures on non-Euclidean structured data termed *geometric deep learning* [18]. These methods date back to early methods to construct neural networks on graphs [159]. More recently, [20] proposed a generalization of convolution for graphs via the Laplacian operator [165]. This foundational approach had a number of drawbacks including the computational complexity of Laplacian eigendecomposition, the large number of parameters to express the convolutional filters, and a lack of spatial localization. These issues are alleviated in follow-up work using polynomial [36, 81] or rational [92] spectral filters that avoid the Laplacian eigendecomposition and also guarantee localization.

Spectral graph CNN models are notable for isometry invariance and hence have applied to non-rigid shape analysis [15]. A key difficulty, however, is that the Laplacian eigenbasis is domain-dependent; thus, a filter learned on one shape may not generalize to others. Spectral transformer networks address this problem to some extent [231].

An alternative definition of non-Euclidean convolution employs spatial rather than spectral filters. The *Geodesic CNN (GCNN)* is a deep CNN on meshes generalizing the notion of patches using local intrinsic parameterization [117]. Its key advantage over

spectral approaches is better generalization. Follow-up work proposed different local charting techniques using anisotropic diffusion [16] or Gaussian mixture models [201, 123]. [101] incorporate a differentiable functional map [129] layer into a geometric deep neural network, allowing to do intrinsic structured prediction of correspondence between nonrigid shapes.

The last class of geometric deep learning approaches attempt to pull back a convolution operation by embedding the shape into a domain with shift-invariant structure such as the sphere [169], torus [116], or plane [44].

2.3 Method

We propose an approach inspired by PointNet and convolution operations. Instead of working on individual points like PointNet, however, we exploit local geometric structures by constructing a local neighborhood graph and applying convolution-like operations on the edges connecting neighboring pairs of points, in the spirit of graph neural networks. We show in the following that such an operation, dubbed *edge convolution* (EdgeConv), has the properties of lying between translation-invariant and non-locality.

Differently from graph CNNs, the graph is not fixed but rather is dynamically updated after each layer of the network. That is, the k-nearest neighbors of a point changes from layer to layer of the network and is computed from the sequence of embeddings. Proximity in feature space differs from proximity in the input, leading to nonlocal diffusion of information throughout the point cloud.

2.3.1 Edge Convolution

Consider a F-dimensional point cloud with n points, denoted by $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^F$. In the simplest setting of $F = 3$, each point contains 3D coordinates $\mathbf{x}_i = (x_i, y_i, z_i)$; it is also possible to include additional coordinates representing color, surface normal, and so on. In a deep neural network architecture, each subsequent layer operates on the output of the previous layer, so more generally the dimension

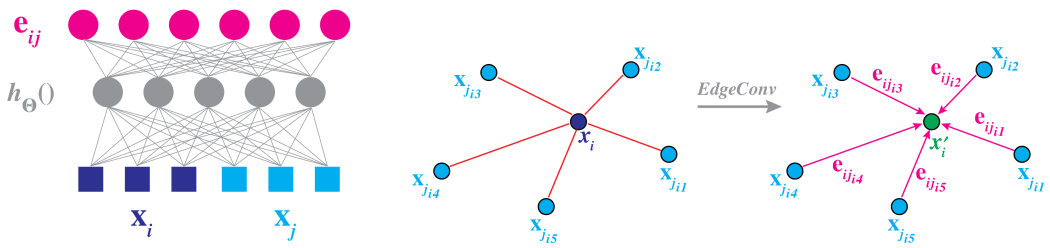


Figure 2-2: **Left**: An example of computing an edge feature, $\mathbf{e}_{ij}$, from a point pair, $\mathbf{x}_i$ and $\mathbf{x}_j$. In this example, $h_{\mathbf{\Theta}}()$ is instantiated using a fully connected layer, and the learnable parameters are its associated weights. **Right**: Visualize the EdgeConv operation. The output of EdgeConv is calculated by aggregating the edge features associated with all the edges emanating from each connected vertex.

F represents the feature dimensionality of a given layer.

We further assume to be given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the local structure of the point cloud, where $\mathcal{V} = \{1, \ldots, n\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ are the *vertices* and *edges*, respectively. In the simplest case, we construct $\mathcal{G}$ as the k-nearest neighbor (k-NN) graph in $\mathbb{R}^F$, containing directed edges of the form $(i, j_{i1}), \ldots, (i, j_{ik})$ such that points $\mathbf{x}_{j_{i1}}, \ldots, \mathbf{x}_{j_{ik}}$ are the closest to $\mathbf{x}_i$. We define *edge features* as $\mathbf{e}_{ij} = h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j)$, where $h_{\mathbf{\Theta}} : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^{F'}$ is some parametric non-linear function parameterized by the set of learnable parameters $\mathbf{\Theta}$.

Finally, we define the EdgeConv operation by applying a channel-wise symmetric aggregation operation $\square$ (e.g., $\sum$ or max) on the edge features associated with all the edges emanating from each vertex. The output of EdgeConv at the i-th vertex is thus given by

$$\mathbf{x}'_i = \square_{j:(i,j)\in\mathcal{E}} h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j). \tag{2.1}$$

Making analogy to the classical convolution operation in images, we can regard $\mathbf{x}_i$ as the central pixel and $\{\mathbf{x}_j : (i, j) \in \mathcal{E}\}$ as a patch around it (see Figure 2-2). Overall, given an F-dimensional point cloud with n points, EdgeConv produces an F'-dimensional point cloud with the same number of points.

Choice of h and $\square$. The choice of the edge function and the aggregation operation has a crucial influence on the properties of the resulting EdgeConv operation.

First, note that in the setting when $\mathbf{x}_1, \ldots, \mathbf{x}_n$ represent image pixels layed out on a regular grid and the graph has a local connectivity representing patches of fixed size around each pixel, the choice $h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j) = \theta_j \mathbf{x}_j$ as the edge function and sum as the aggregation operation yields the classical Euclidean convolution,

$$\mathbf{x}'_i = \sum_{j:(i,j)\in\mathcal{E}} \theta_j \mathbf{x}_j,$$

where the parameters $\mathbf{\Theta} = (\theta_i, \ldots, \theta_k)$ act as the weights of the filter.

The second possible choice of h is $h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j) = h_{\mathbf{\Theta}}(\mathbf{x}_i)$, encoding only global shape information oblivious of the local neighborhood structure. This type of operation is used in *PointNet*, which can thus be regarded as a particular choice of our EdgeConv.

A third option is $h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j) = h_{\mathbf{\Theta}}(\mathbf{x}_j - \mathbf{x}_i)$. Note that such a choice encodes only local information, essentially treating the shape as a collection of small patches and losing the global shape structure.

Finally, the fourth option, which we adopt in this paper, is an asymmetric edge function of the form $h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j) = h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i)$. Such a function combines both the global shape structure (captured by the coordinates of the patch centers $\mathbf{x}_i$) and local neighborhood information (captured by $\mathbf{x}_j - \mathbf{x}_i$).

2.3.2 Dynamic Graph CNNs

Similarly to classical CNNs used in computer vision, the EdgeConv operation can be applied multiple times, possibly interleaved with pooling depending on the task at hand. When applied without pooling, multiple applications of EdgeConv produce effectively larger support ('receptive field') of the filter. We denote by $\mathbf{X}^{(l)} = \{\mathbf{x}_1^{(l)}, \ldots, \mathbf{x}_n^{(l)}\} \subseteq \mathbb{R}^{F_l}$ the output of the l-th layer; $\mathbf{X}^{(0)} = \mathbf{X}$ is the input point cloud.

Dynamic graph update. Our experiments suggest that it is possible and actually beneficial to *recompute the graph* using nearest neighbors in the features space produces by each layer. This is a crucial distinction of our method from graph CNNs working on a fixed input graph. Such a dynamic graph update is the reason for the

name of our architecture, the *Dynamic Graph CNN (DGCNN)*. At each layer we thus have a different graph $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$, where the l-th layer edges are of the form $(i, j_{i1}), \ldots, (i, j_{ik_l})$ such that $\mathbf{x}_{j_{i1}}^{(l)}, \ldots, x_{j_{ik_l}}^{(l)}$ are the k_l points closest to $\mathbf{x}_i^{(l)}$. The F_{l+1}-dimensional output of the $(l+1)$-st layer is produced by applying EdgeConv to the F_l-dimensional output of the l-th layer,

$$\mathbf{x}_i^{(l+1)} = \square_{j:(i,j)\in\mathcal{E}^{(l)}} h_{\mathbf{\Theta}}^{(l)}(\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)}), \tag{2.2}$$

where $h^{(l)} : \mathbb{R}^{F_l} \times \mathbb{R}^{F_l} \to \mathbb{R}^{F_{l+1}}$.

2.3.3 Implementation Details

We consider particular instances of Dynamic Graph CNNs for two prototypical tasks in point cloud analysis: classification and segmentation. The respective architectures, depicted in Figure 2-3 (top and bottom branches), have a similar structure to PointNet. Both architectures share a spatial transformer component, computing a global shape transformation. The classification network includes two EdgeConv layers, followed by a pooling operation and three fully-connected layers producing classification output scores. The segmentation network uses a sequence of three EdgeConv layers, followed by three fully-connected layers producing, for each point, segmentation output scores. For each EdgeConv block, we use a shared edge function $h^{(l)}(\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)}) = h(\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)} - \mathbf{x}_i^{(l)})$ across all layers; the function is implemented as a multi-layer perceptron (MLP) and $\square$ = max aggregation operation.

In our classification architecture, the graph is constructed using $k = 20$ nearest neighbors, while in our segmentation architecture, $k = 30$.

2.3.4 Relations to existing methods

Our DGCNN is related to two classes of approaches, PointNets and graph CNNs, which we show to be particular settings of our method.

PointNet is a special case of our method with $k = 1$, which results in a graph with

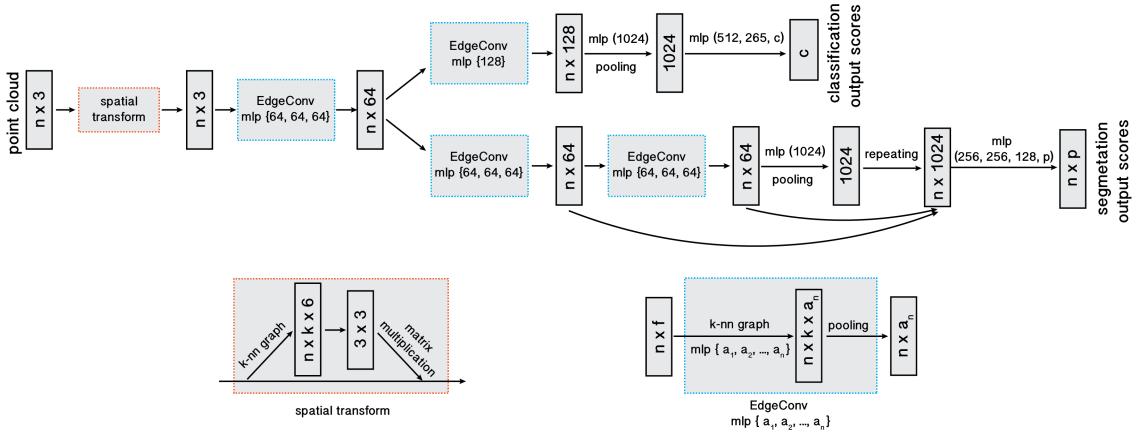


Figure 2-3: **Model architectures:** The model architectures used for classification (top branch) and segmentation (bottom branch). The classification model takes as input n points, calculates an edge feature set of size k for each point at an EdgeConv layer, and aggregates features within each set to compute EdgeConv responses for corresponding points. The output features of the last EdgeConv layer are aggregated globally to form an $1D$ global descriptor, which is used to generate classification scores for c classes. The segmentation model extends the classification model by concatenating the $1D$ global descriptor and all the EdgeConv outputs (serving as local descriptors) for each point. It outputs per-point classification scores for p semantic labels. For illustration purposes, two arrowed arcs are plotted to represent feature concatenation. **Point cloud transform block:** The point cloud transform block is designed to align an input point set to a canonical space by applying an estimated 3×3 matrix. To estimate the 3×3 matrix, a tensor concatenating the coordinates of each point and the coordinate differences between its k neighboring points is used. **EdgeConv block:** The EdgeConv block takes as input a tensor of shape $n \times f$, computes edge features for each point by applying a multi-layer perceptron (mlp) with the number of layer neurons defined as $\{a_1, a_2, ..., a_n\}$, and generates a tensor of shape $n \times a_n$ after pooling among neighboring edge features.

an empty edge set $\mathcal{E} = \varnothing$. The edge function used in PointNet is $h(\mathbf{x}_i, \mathbf{x}_j) = h(\mathbf{x}_i)$, which considers only the global geometry but discards the local one. The aggregation operation used in PointNet is $\square = \max$ (or $\sum$, because the aggregation function only works on a single node).

PointNet++ tries to account for local point cloud structure by applying PointNet in a local manner. In terms of our notation, PointNet++ first constructs the graph according to the Euclidean distances between the points, and in each layer, applies a graph coarsening operation. For each layer, a certain number of points are selected by using farthest point sampling (FPS) algorithm. Only the selected points are preserved

while others are directly discarded after this layer and in this way, the graph becomes smaller after the operation applied on each layer. Different from ours, PointNet++ computes pairwise distances using point input coordinates. The edge function used by PointNet++ is also $h(\mathbf{x}_i, \mathbf{x}_j) = h(\mathbf{x}_i)$, and the aggregation operation is also a max.

Among graph CNNs, MoNet [123], ECC [168], and Graph Attention Networks [201] are the most related approaches. The common denominator of these methods is the notion of a local patch on a graph, in which a convolution-type operation can be defined.[1] Specifically, [123] use the graph structure to compute a local "pseudo-coordinate system" $\mathbf{u}$ in which the neighborhood vertices are represented; the convolution is then defined as an M-component Gaussian mixture in these coordinates:

$$\mathbf{x}'_i = \sum_{m=1}^{M} w_m \sum_{j:(i,j)\in\mathcal{E}} g_{\boldsymbol{\Theta}_m}(\mathbf{u}(\mathbf{x}_i, \mathbf{x}_j))\mathbf{x}_j, \tag{2.3}$$

where g denotes the Gaussian kernel, $\{\boldsymbol{\Theta}_1, \dots, \boldsymbol{\Theta}_M\}$ encode the learnable parameters of the Gaussians (mean and covariance), and $\{w_1, \dots, w_M\}$ are the learnable filter coefficients. We can easily observe that (2.3) is an instance of our more general EdgeConv operation (2.1), with a particular choice of the edge function

$$h_{w_1,\boldsymbol{\Theta}_1,\dots,w_M,\boldsymbol{\Theta}_M}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{m=1}^{M} w_m g_{\boldsymbol{\Theta}_m}(\mathbf{u}(\mathbf{x}_i, \mathbf{x}_j))\mathbf{x}_j$$

and summation as the aggregation operation.

A crucial difference between EdgeConv and MoNet and other graph CNN methods is that the latter assume a given fixed graph on which the convolution-like operations are applied, while we *dynamically update the graph* for each layer output. This way, our model not only learns how to extract local geometric features, but also how to group points in a point cloud. Figure 2-4 shows the distance in different feature spaces, exemplifying that the distances in deeper layers carry semantic information over long distances.

[1] The methods of [168] and [201] can be considered as instances of [123], with the difference that the weights are constructed employing features from the adjacent nodes instead of the graph structure.

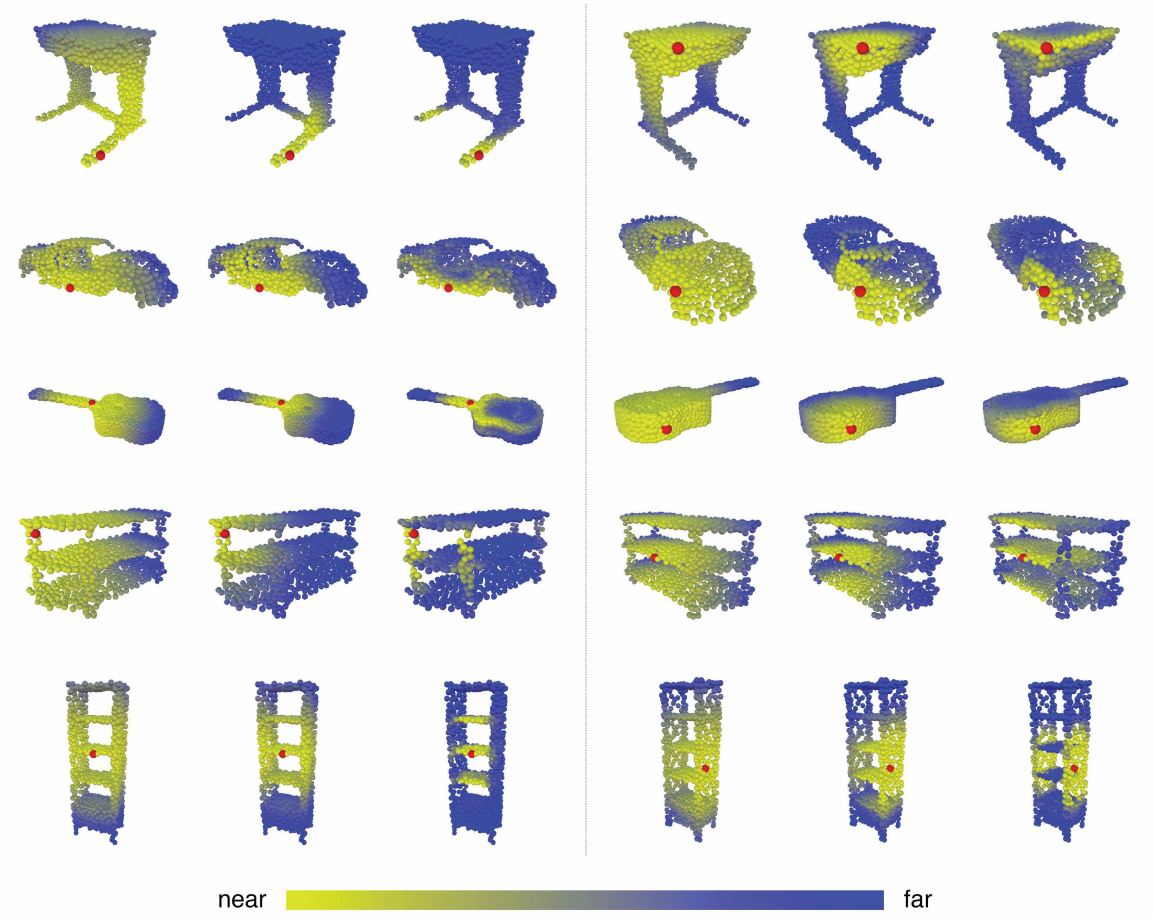


Figure 2-4: **Structure of the feature spaces** produced at different stages of our shape classification neural network architecture, visualized as the distance between the red point to the rest of the points. For each set, **Left:** Euclidean distance in the input $\mathbb{R}^3$ space; **Middle:** Distance after the point cloud transform stage, amounting to a global transformation of the shape; **Right:** Distance in the feature space of the last layer. Observe how in the feature space of deeper layers semantically similar structures such as shelves of a bookshelf or legs of a table are brought close together, although they are distant in the original space.

2.4 Experiments

In this section, we evaluate the models constructed using EdgeConv for different tasks: classification, part segmentation, and semantic segmentation. We also visualize experimental results to illustrate key differences from previous work.

2.4.1 Classification

Data. We evaluate our model on the ModelNet40 [218] classification task, consisting in predicting the category of a previously unseen shape. The dataset contains 12,311 meshed CAD models from 40 categories. 9,843 models are used for training and 2,468 models are for testing. We follow verbatim the experimental settings of [141]. For each model, 1,024 points are uniformly sampled from the mesh faces and normalized to the unit sphere. Only the (x, y, z) coordinates of the sampled points are used and the original meshes are discarded. During the training procedure, we augment the data as in [143] by randomly rotating and scaling objects and perturbing the object and point locations.

Architecture. The network architecture used for the classification task is shown in Figure 2-3 (top branch). We use a local-aware spatial transformer network to align the point cloud. It has two shared fully-connected layers $(64, 128)$[2] to construct one EdgeConv layer and after which, one shared fully-connected layer (1024) is used to transform the pointwise features to higher-dimensional space. After the global max pooling, two fully-connected layers $(512, 256)$ are used to compute the transformation matrix.

We use two EdgeConv layers to extract geometric features. The first EdgeConv layer uses three shared fully-connected layers (64, 64, 64) while the second EdgeConv layer uses a shared fully-connected layer (128). Shortcut connections are included to extract multi-scale features and one shared fully-connected layer (1024) to aggregate multi-scale features. The number k of nearest neighbors is 20. Then, a global max pooling is used to get the point cloud global feature, after which two multi-layer perceptrons $(512, 256)$ are used to transform the global feature. Dropout with keep probability of 0.5 is used in the last two fully-connected layers. All layers include ReLU and batch normalization.

[2]Here, we use $(64, 128)$ to denote that the two fully-connected layers have 64 filters and 128 filters, respectively; we use the same notation for the remainder of our discussion.

Training. We use the same training strategy as [141]. We use Adam [80] with learning rate 0.001 that is divided by 2 every 20 epochs. The decay rate for batch normalization is initially 0.5 and 0.99 finally. The batch size is 32 and the momentum is 0.9.

Results. Table 2.1 shows the results of the classification task. Our model achieves the best results on this dataset. Our baseline without transformer network and using fixed graph is 0.5% better than PointNet++. An advanced version including a local-aware network and dynamical graph recomputation achieves best results on this dataset.

	Mean Class Accuracy	Overall Accuracy
3DShapeNets [218]	77.3	84.7
VoxNet [118]	83.0	85.9
Subvolume [142]	86.0	89.2
ECC [168]	83.2	87.4
PointNet [141]	86.0	89.2
PointNet++ [143]	-	90.7
Ours (baseline)	88.8	91.2
Ours	**90.2**	**92.2**

Table 2.1: Classification results on ModelNet40.

2.4.2 Model Complexity

We use the ModelNet40 [218] classification experiment to compare the complexity of our model to previous state-of-the-art. Table 2.2 shows that our model achieves the best tradeoff between the model complexity (number of parameters), computational complexity (measured as forward pass time), and achieved classification accuracy.

Our baseline model outperforms the previous state-of-the-art PointNet++ by 0.5% accuracy, at the same time being 5 times faster compared to PointNet++. Our baseline model does not use a spatial transformer and uses the fixed k-NN graph. A more advanced version of our model including a spatial transformer block and dynamically graph computation outperforms PointNet++ by 1.5% while having comparable number of parameters and computational complexity.

	Model size(MB)	Forward time(ms)	Accuracy(%)
PointNet (Baseline)	9.4	11.6	87.1
PointNet	40	25.3	89.2
PointNet++	12	163.2	90.7
Ours (Baseline)	11	29.7	91.2
Ours	21	94.6	92.2

Table 2.2: Complexity, forward time and accuracy of different models

2.4.3 More Experiments on ModelNet40

We also experiment with various settings of our model on the ModelNet40 [218] dataset. In particular, we analyze the effectiveness of local-aware transformer network, different distance metrics, and explicit usage of $\mathbf{x}_i - \mathbf{x}_j$.

Table 2.3 shows the results. "Centralization" denotes using concatenation of $\mathbf{x}_i$ and $\mathbf{x}_i - \mathbf{x}_j$ as the edge features rather than concatenating $\mathbf{x}_i$ and $\mathbf{x}_j$. "Spatial Transformer" denotes the local-aware transformer network while "Dynamic graph recomputation" denotes we reconstruct the graph rather than using a fixed graph. By dynamically updating graph, there is about 0.2%$\sim$0.3% improvement and Figure 2-4 also verifies our hypothesis that the model can extract semantics. In the later layers, certain patterns occur for recognition tasks. Explicitly centralizing each patch by using the concatenation of $\mathbf{x}_i$ and $\mathbf{x}_i - \mathbf{x}_j$ makes the operator more robust to translation, leading to about 0.2%$\sim$0.3% improvement for overall accuracy. The local-aware transformer makes the model invariant to rigid transformation and leads to approximately 0.7%

improvement.

We also experiment with different numbers k of nearest neighbors as shown in Table 2.4. While we do not exhaustively experiment with all possible k, we find with large k that the performance degenerates. This confirms our hypothesis that with large k the Euclidean distance fails to approximate geodesic distance, destroying the geometry of each patch.

We further evaluate the robustness of our model (trained on 1,024 points with $k = 20$) to point cloud density. We simulate the environment that random input points drops out during testing. Figure 2-5 shows that even half of points is dropped, the model still achieves reasonable results. With fewer than 512 points, however, performance degenerates dramatically.

CENT	DYN	XFORM	Mean Class Accuracy(%)	Overall Accuracy(%)
x			88.8	91.2
x	x		88.8	91.5
x		x	89.6	91.9
	x	x	89.8	91.9
x	x	x	90.2	92.2

Table 2.3: Effectiveness of different components. CENT denotes centralization, DYN denotes dynamical graph recomputation, and XFORM denotes the use of a spatial transformer.

2.4.4 Part Segmentation

Data We extend our EdgeConv model architectures for part segmentation task on ShapeNet part dataset [230]. For this task, each point from a point cloud set is classified into one of a few predefined part category labels. The dataset contains 16,881 3D shapes from 16 object categories, annotated with 50 parts in total. 2,048 points are sampled from each training shape, and most sampled point sets are labeled with less than six parts. We follow the official train/validation/test split scheme as in [27] in our experiment.

Number of nearest neighbors (k)	Mean Class Accuracy(%)	Overall Accuracy(%)
5	88.0	90.5
10	88.8	91.4
20	90.2	92.2
40	89.2	91.7

Table 2.4: Results of our model with different numbers of nearest neighbors.

Architecture The network architecture is illustrated in Figure 2-3 (bottom branch). The same spatial transformer network is used for segmentation task. Nine shared fully-connected layers are used to construct three EdgeConv layers; each EdgeConv layer has three fully-connected layers $(64, 64, 64)$. A shared fully-connected layer (1024) is used to aggregate information from the previous layers. Shortcut connections are used to include all the EdgeConv outputs as local feature descriptors. At last, three shared fully-connected layers $(256, 256, 128)$ are used to transform the pointwise features. Batch-norm, dropout, and ReLU are included in the similar fashion to our classification network.

Training The same training setting as in our classification task is adopted, except k is changed from 20 to 30 due to the increase of point density. A distributed training scheme is further implemented on two NVIDIA TITAN X GPUs to maintain the training batch size.

Results We use Intersection-over-Union (IoU) on points to evaluate our model and compare with other benchmarks. We follow the same evaluation scheme as PointNet: IoU of a shape is computed by averaging the IoUs of different parts occurring in that shape; IoU of a category is obtained by averaging the IoUs of all the shapes belonging to that category. The mean IoU (mIoU) is finally calculated by averaging the IoUs of all the testing shapes. We compare our results with PointNet [141], PointNet++ [143], Kd-Net [82], and LocalFeatureNet [162]. The evaluation results are shown in Table 2.5. We also visually compare the results of our model and PointNet in Figure 2-12.

	MEAN	AREO	BAG	CAP	CAR	CHAIR	EAR PHONE	GUITAR	KNIFE	LAMP	LAPTOP	MOTOR	MUG	PISTOL	ROCKET	SKATE BOARD	TABLE	WINNING CATEGORIES
# SHAPES		2690	76	55	898	3758	69	787	392	1547	451	202	184	283	66	152	5271	
POINTNET	83.7	83.4	78.7	82.5	74.9	89.6	73.0	**91.5**	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6	1
POINTNET++	**85.1**	82.4	79.0	**87.7**	77.3	90.8	71.8	91.0	85.9	**83.7**	95.3	**71.6**	**94.1**	81.3	58.7	**76.4**	82.6	5
KD-NET	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3	0
LOCALFEATURENET	84.3	**86.1**	73.0	54.9	**77.4**	88.8	55.0	90.6	86.5	75.2	**96.1**	57.3	91.7	**83.1**	53.9	72.5	**83.8**	5
OURS	**85.1**	84.2	**83.7**	84.4	77.1	**90.9**	**78.5**	**91.5**	**87.3**	82.9	96.0	67.8	93.3	82.6	**59.7**	75.5	82.0	**6**

Table 2.5: Part segmentation results on ShapeNet part dataset. Metric is mIoU(%) on points.

Intra-cloud distances. We next explore the relationships between different point clouds captured using our features. As shown in Figure 2-11, we take one red point from a source point cloud and compute its distance in feature space to points in other point clouds from the same category. An interesting finding is that although points are from different sources, they are close to each other if they are from semantically similar parts. We evaluate on the features after the third layer of our segmentation model for this experiment.

Segmentation on partial data. Our model is robust to partial data. We simulate the environment that part of the shape is dropped from one of six sides (top, bottom, right, left, front and back) with different percentages. The results are shown in Figure 2-13. On the left, the mean IoU versus "keep ratio" is shown. On the right, the results for an airplane model are visualized.

2.4.5 Indoor Scene Segmentation

Data. We evaluate our model on Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [3] for a semantic scene segmentation task. This dataset includes 3D scan point clouds for 6 indoor areas including 272 rooms in total. Each point belongs to one of 13 semantic categories—e.g. board, bookcase, chair, ceiling, and beam—plus clutter. We follow the same setting as in [141], where each room is split into blocks with area $1m \times 1m$, and each point is represented as a 9D vector (XYZ, RGB, and normalized spatial coordinates). 4,096 points are sampled for each block during training process, and all points are used for testing. We also use the same 6-fold cross

validation over the 6 areas, and the average evaluation results are reported.

The model used for this task is similar to part segmentation model, except that a probability distribution over semantic object classes is generated for each input point. We compare our model with both PointNet [141] and PointNet baseline, where additional point features (local point density, local curvature and normal) are used to construct handcrafted features and then fed to an MLP classifier. We further compare our work with [40], who present network architectures to enlarge the receptive field over the 3D scene. Two different approaches are proposed in their work: MS+CU for multi-scale block features with consolidation units; G+RCU for the grid-blocks with recurrent consolidation Units. We report evaluation results in Table 2.6, and visually compare the results of PointNet and our model in Figure 2-15.

	Mean IoU	overall accuracy
PointNet (Baseline) [141]	20.1	53.2
PointNet [141]	47.6	78.5
MS + CU(2) [40]	47.8	79.2
G + RCU [40]	49.7	81.1
Ours	**56.1**	**84.1**

Table 2.6: 3D semantic segmentation results on S3DIS. MS+CU for multi-scale block features with consolidation units; G+RCU for the grid-blocks with recurrent consolidation Units.

2.4.6 Surface Normal Prediction

We can also adapt our segmentation model to predict surface normals from point clouds.

Data. We still use the ModelNet40 dataset. The surface normals are sampled directly from CAD models. The normal of one point is represented by (n_x, n_y, n_z). We use 9,843 models for training and 2,468 models for testing.

Architecture. We change the last layer of our segmentation model to output 3 continuous values; mean squared error (MSE) is used as the training loss.

Results. Qualitative results are shown in Figure 2-14, compared with ground truth. Our model faithfully captures orientation even in the presence of fairly sharp features.

2.5 Conclusion

In this work we propose a new operator for learning on point cloud and show its performance on various tasks. The success of our technique verifies our hypothesis that local geometric features are crucial to 3D recognition and segmentation tasks, even after introducing machinery from deep learning. Furthermore, we show our model can be easily modified for various tasks like normal prediction while continuing to achieve reasonable results.

While our architectures easily can be incorporated as-is into existing pipelines for point cloud-based graphics, learning, and vision, our experiments also indicate several avenues for future research and extension. Primarily, the success of our model suggests that intrinsic features can be equally valuable if not more than simply point coordinates; developing a practical and theoretically-justified framework for balancing intrinsic and extrinsic considerations in a learning pipeline will require insight from theory and practice in geometry processing. Another possible extension is to design a non-shared transformer network that works on each local patch differently, adding flexibility to our model. Finally, we will consider applications of our techniques to more abstract point clouds coming from applications like document retrieval rather than 3D geometry; beyond broadening the applicability of our technique, these experiments will provide insight into the role of geometry in abstract data processing.

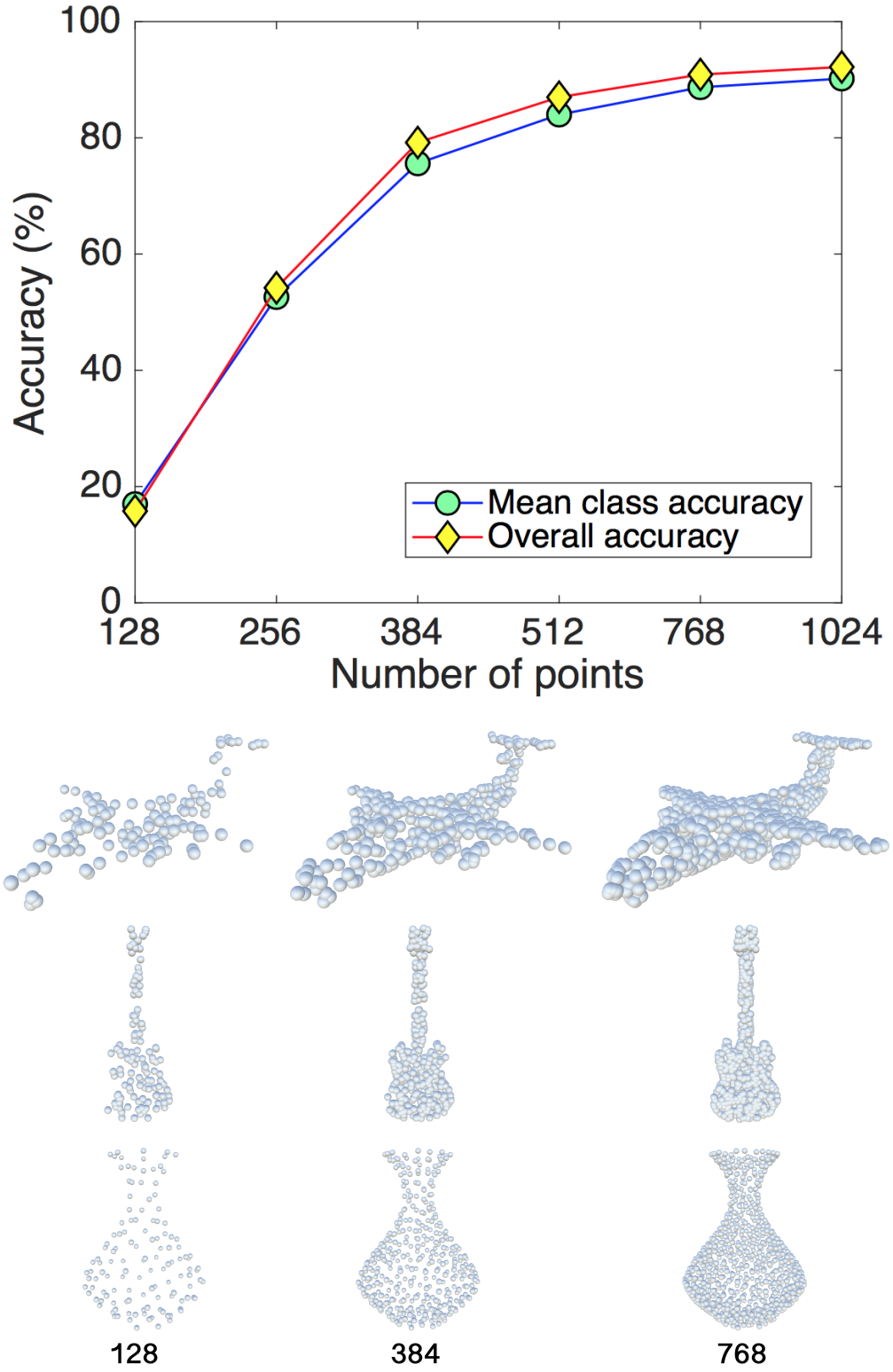


Figure 2-5: **Up:** Results of our model tested with random input dropout. The model is trained with number of points being 1024 and k being 20. **Bottom:** Point clouds with different number of points. The numbers of points are shown below the bottom row.

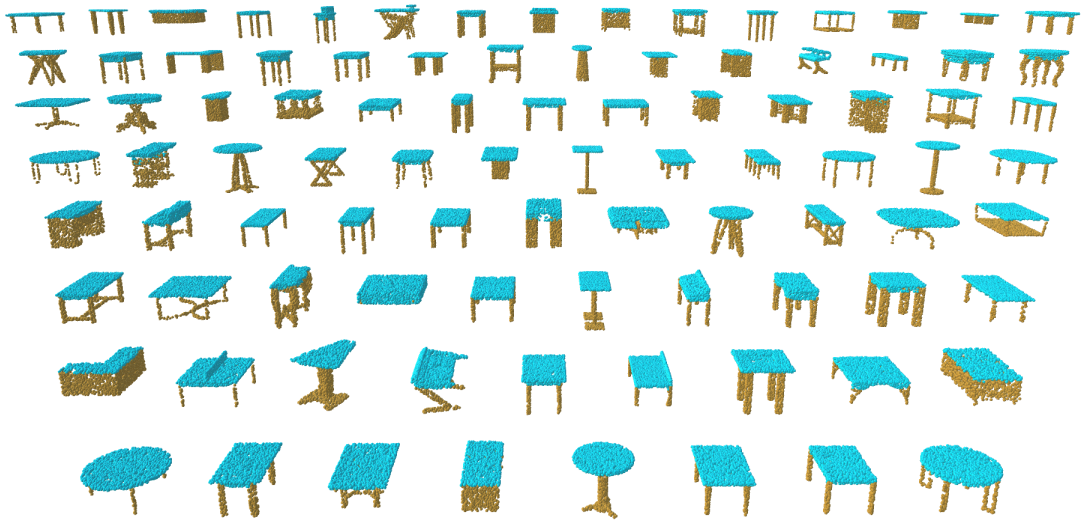


Figure 2-6: Our part segmentation testing results for tables.

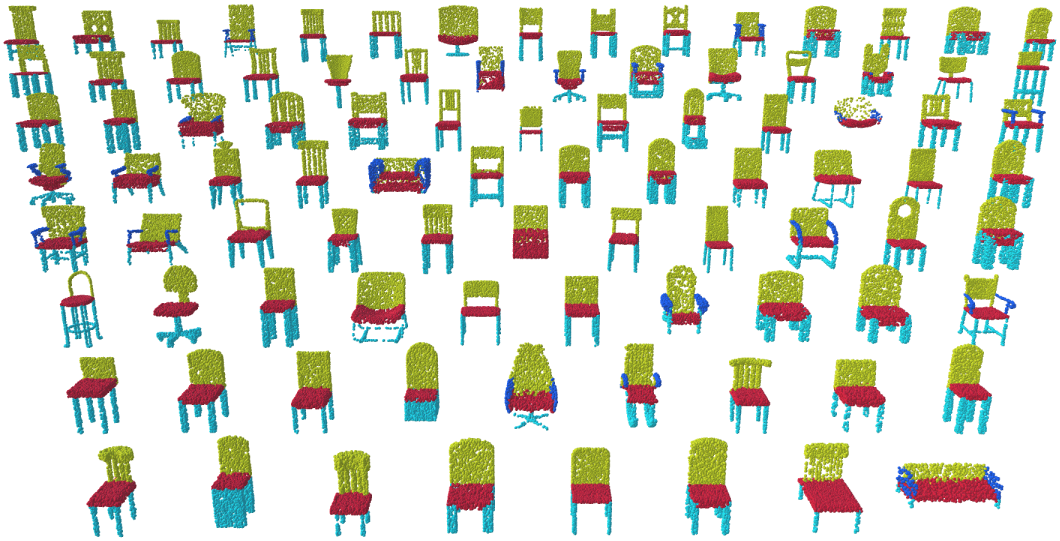


Figure 2-7: Our part segmentation testing results for chairs.

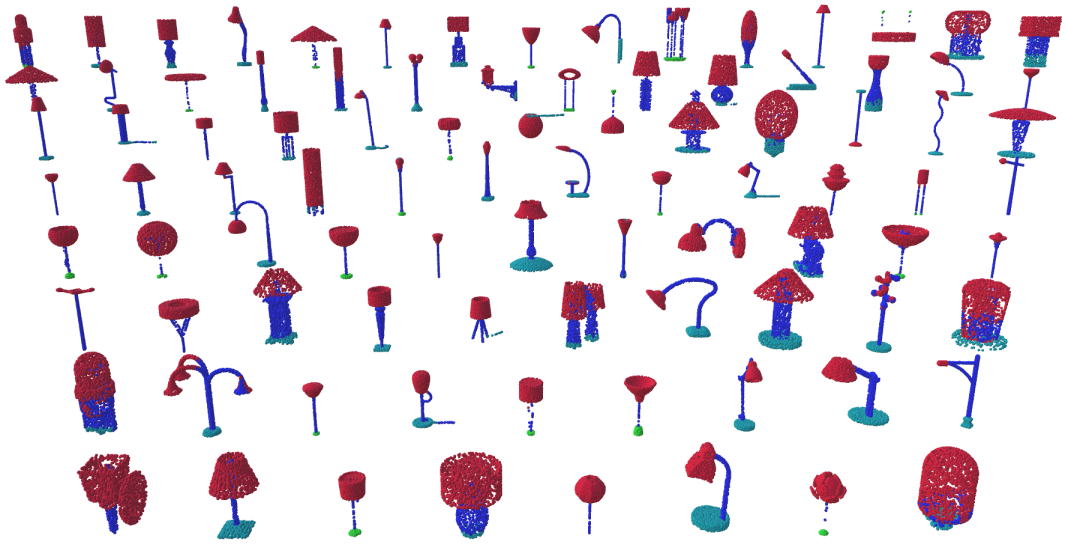


Figure 2-8: Our part segmentation testing results for lamps.

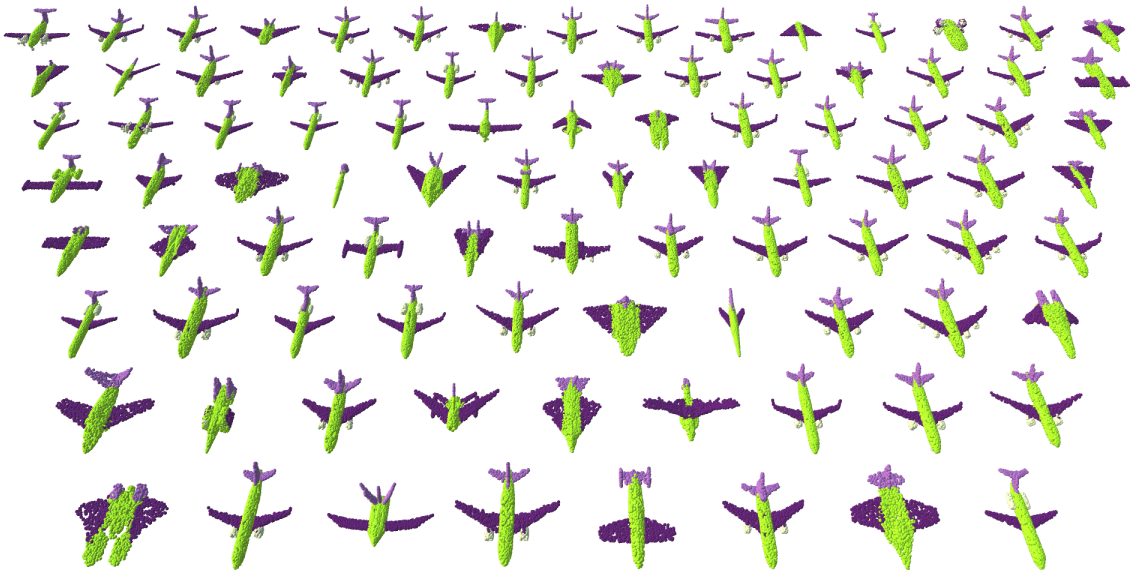


Figure 2-9: Our part segmentation testing results for aircrafts.

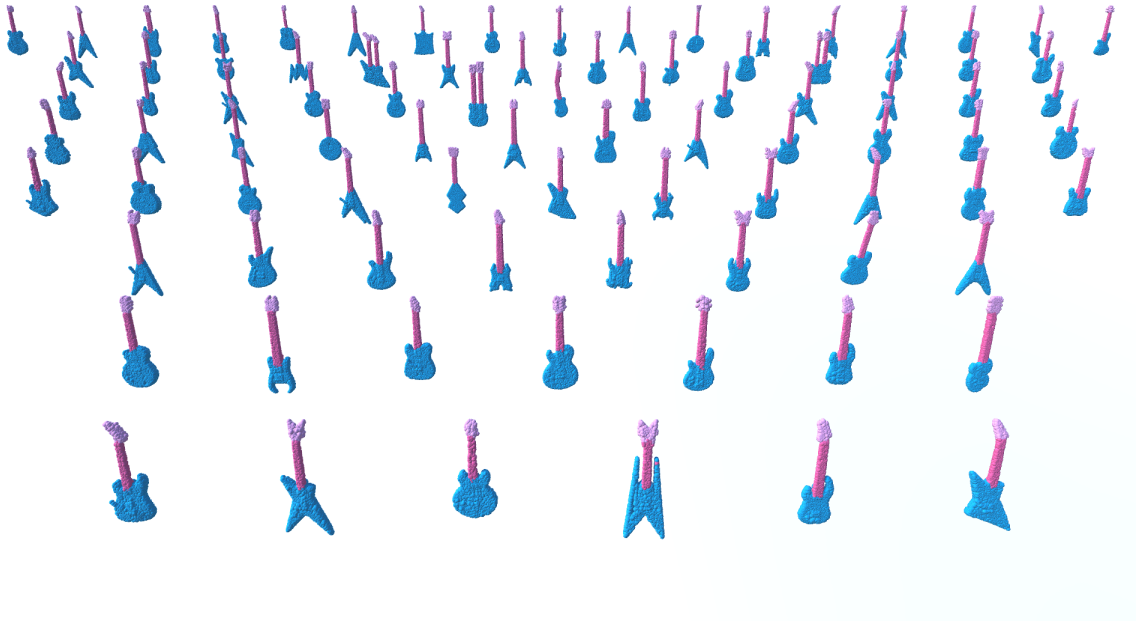


Figure 2-10: Our part segmentation testing results for guitars.

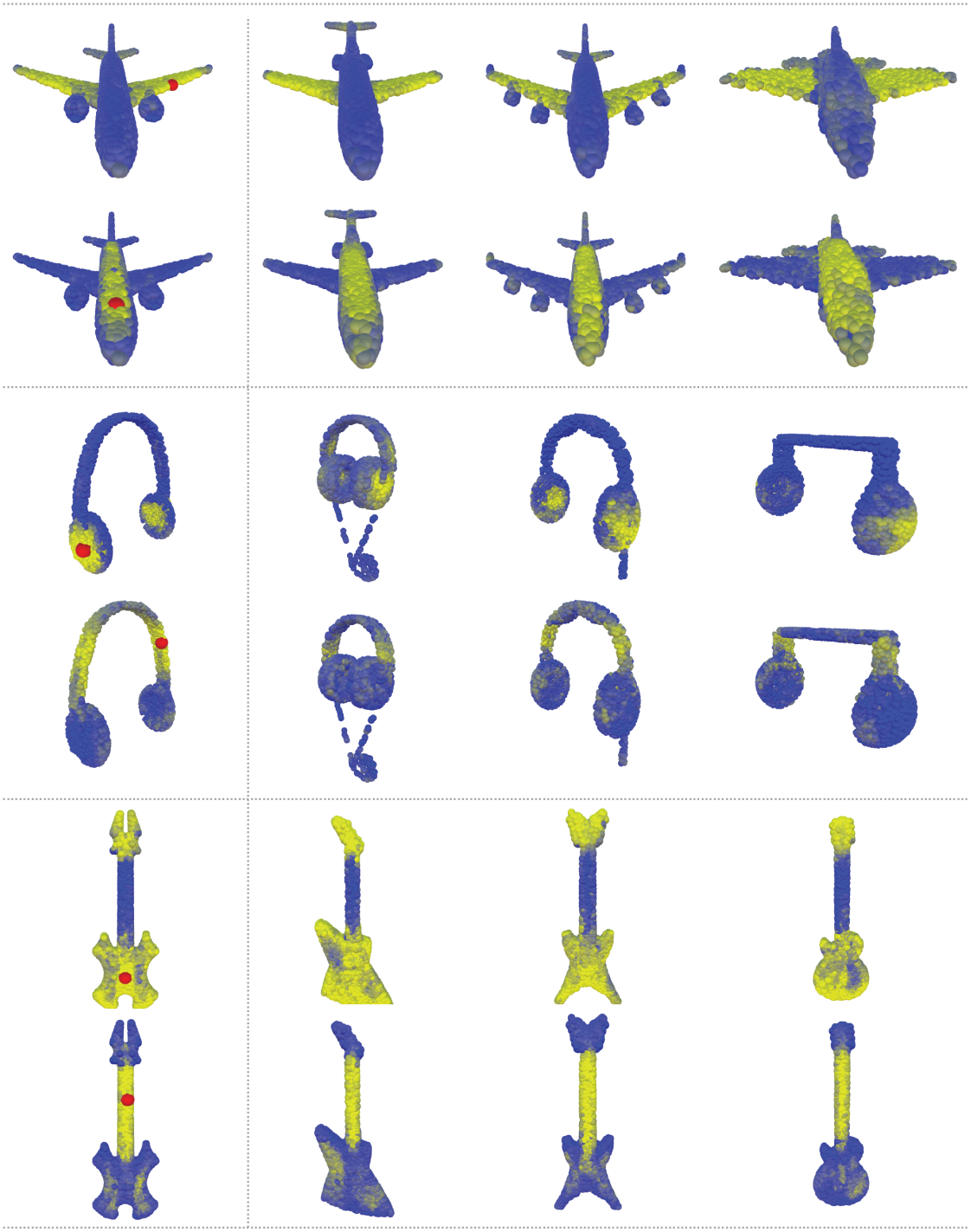
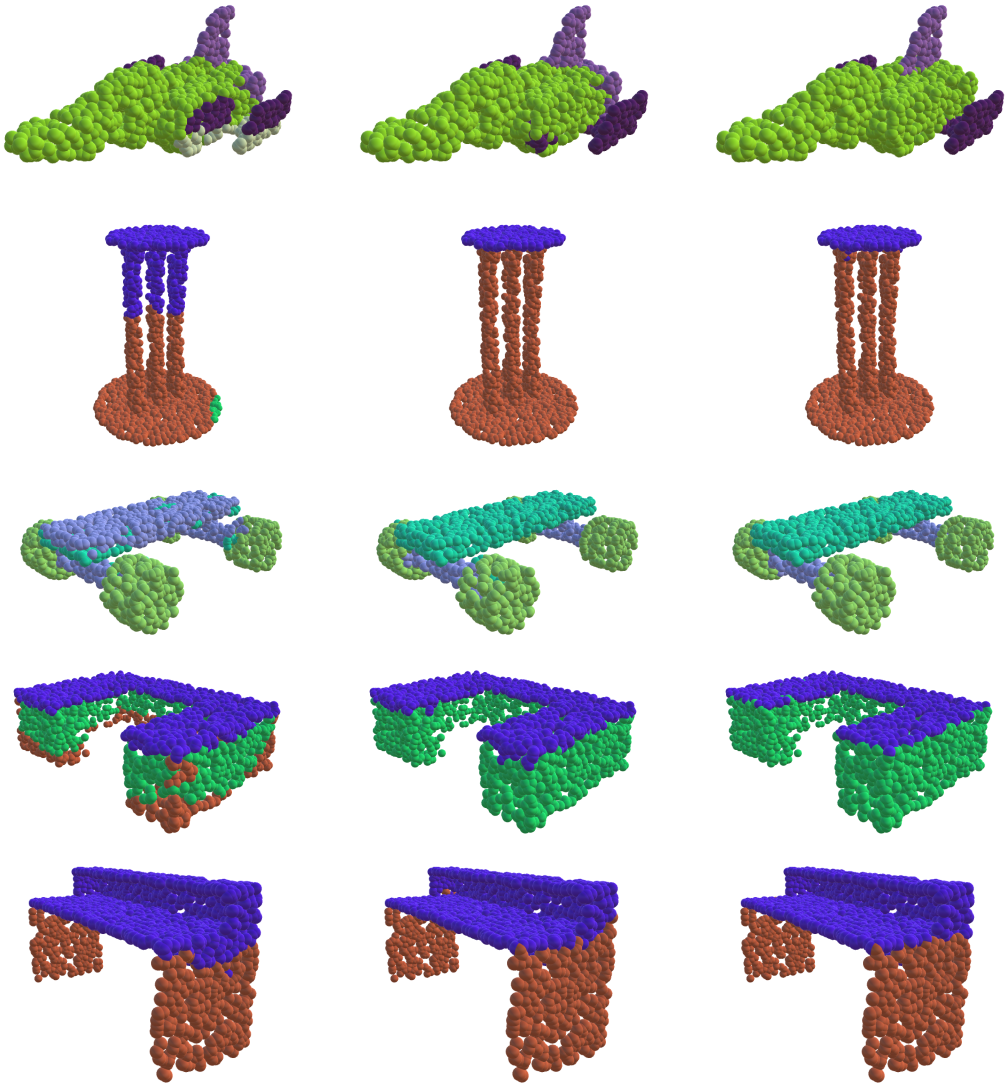


Figure 2-11: Visualization of the Euclidean distance (yellow: near, blue: far) between source points (red points in the left column) and multiple point clouds from the same category in the feature space after the third EdgeConv layer. Notice source points not only capture semantically similar structures in the point clouds that they belong to, but also capture semantically similar structures in other point clouds from the same category.



PointNet Ours Ground truth

Figure 2-12: Compare part segmentation results. For each set, from left to right: PointNet, ours and ground truth.

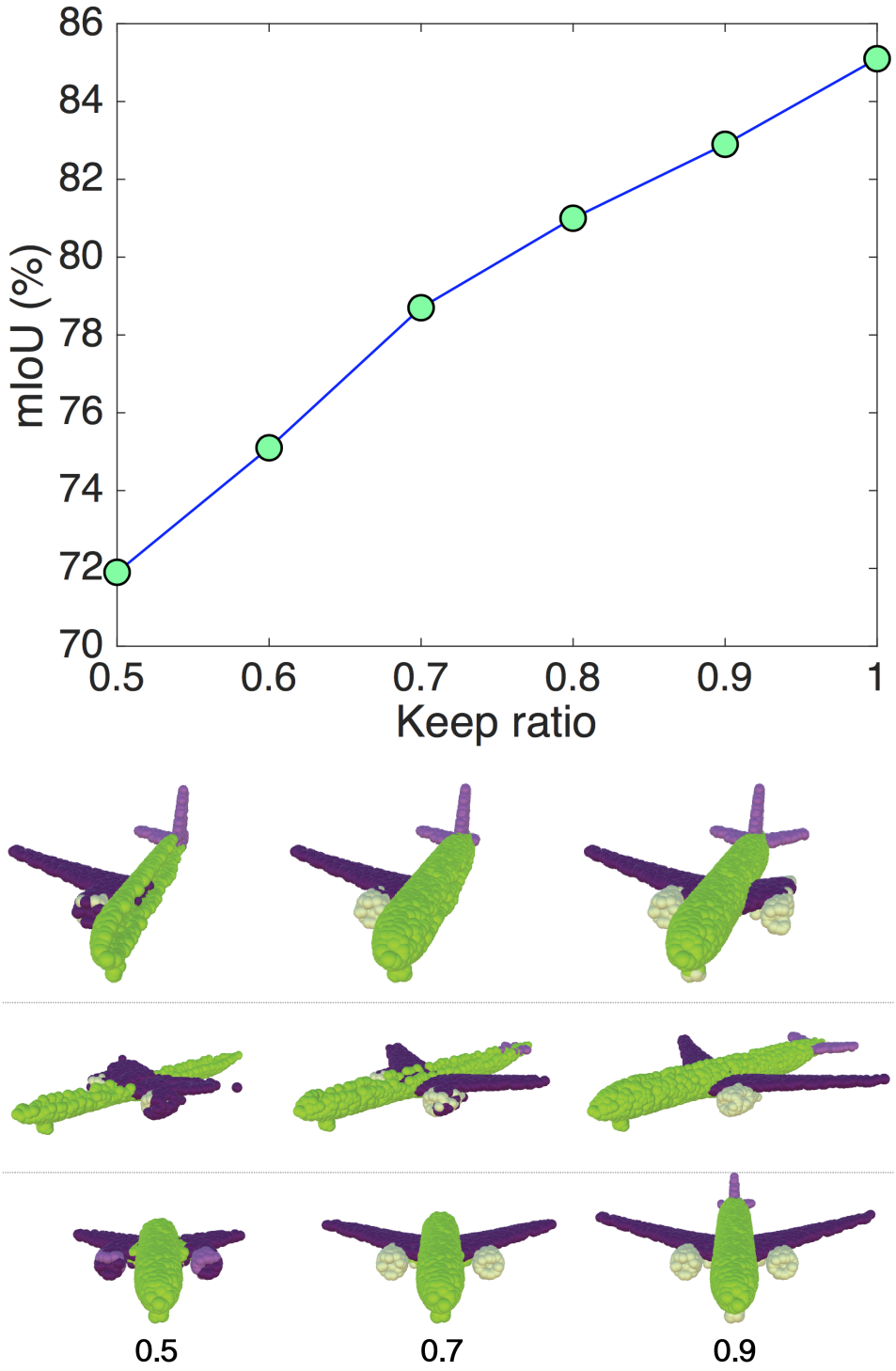


Figure 2-13: **Up:** The mean IoU (%) improves when the ratio of kept points increases. Points are dropped from one of six sides (top, bottom, left, right, front and back) randomly during evaluation process. **Bottom:** Part segmentation results on partial data. Points on each row are dropped from the same side. The keep ratio is shown below the bottom row. Note that the segmentation results of turbines are improved when more points are included.

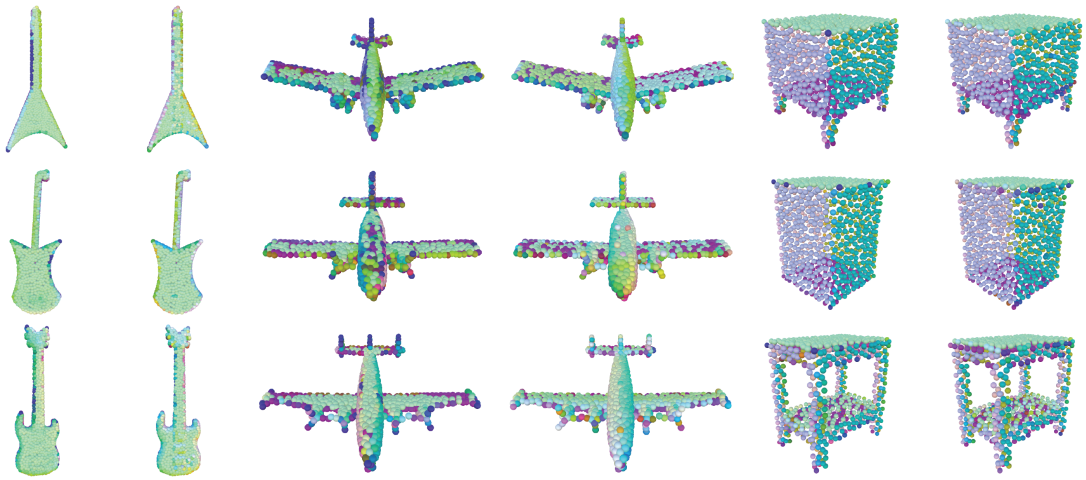


Figure 2-14: Surface normal estimation results. The colors shown in the figure are RGB-coded surface normals, meaning XYZ components of surface normal vectors are put into RGB color channels. For each pair: our prediction (left) and ground truth (right).

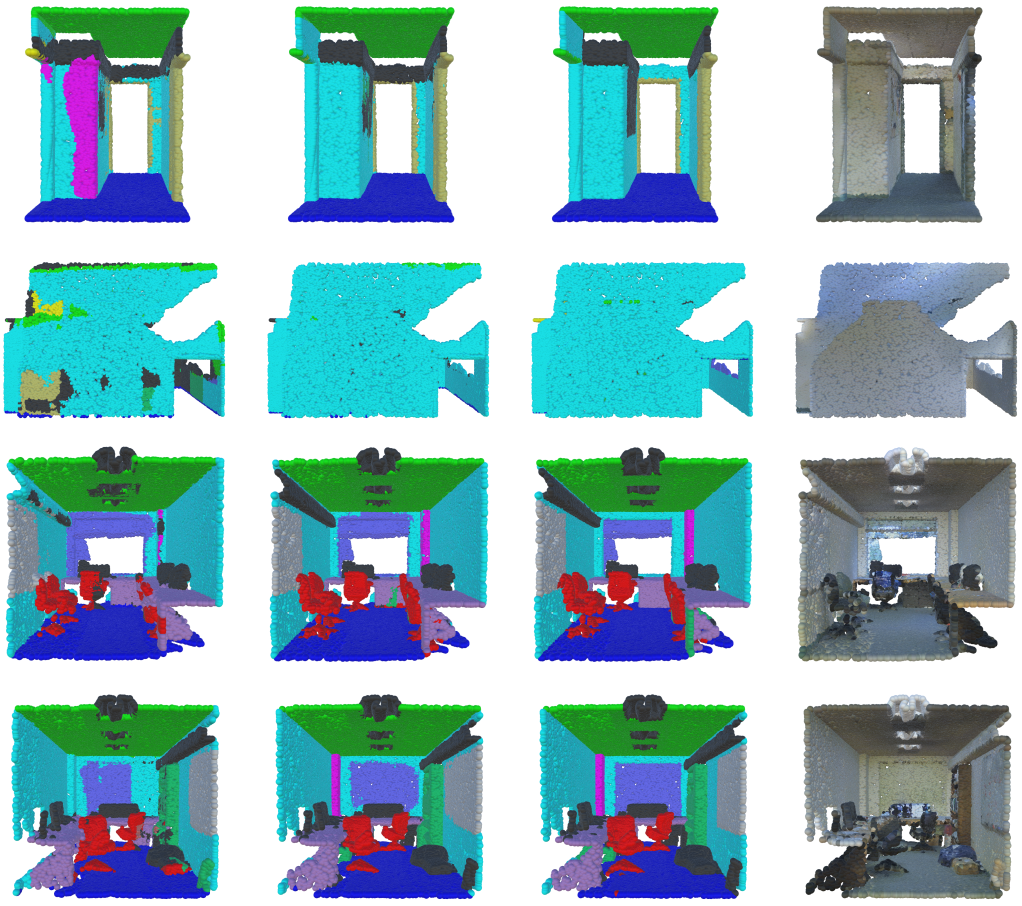


Figure 2-15: Semantic segmentation results. From left to right: PointNet, ours, ground truth and point cloud with original color. Notice our model outputs smoother segmentation results, for example, wall (cyan) in top two rows, chairs (red) and columns (magenta) in bottom two rows.

Chapter 3

Pillar-based Object Detection

3D object detection is a central component of perception systems for autonomous driving, used to identify pedestrians, vehicles, obstacles, and other key features of the environment around a car. Ongoing development of object detection methods for vision, graphics, and other domain areas has led to steady improvement in the performance and reliability of these systems as they transition from research to production. This chapter uses 3D object detection as a showcase to demonstrate how our methods tackle point cloud feature learning problems. The point cloud networks developed in this chapter can be immediately applicable to other high-level semantic reasoning tasks.

3.1 Introduction

Most 3D object detection algorithms project points to a single prescribed view for feature learning. These views—e.g., the "bird's eye" view of the environment around the car—are not necessarily optimal for distinguishing objects of interest. After computing features, these methods typically make *anchor-based* predictions of the locations and poses of objects in the scene. Anchors provide useful position and pose priors, but they lead to learning algorithms with many hyperparameters and potentially unstable training.

Two popular architectures typifying this approach are PointPillars [88] and multi-

view fusion (MVF) [247], which achieve top efficiency and performance on recent 3D object detection benchmarks. These methods use learned representations built from birds-eye view pillars above the ground plane. MVF also benefits from complementary information provided by a spherical view. These methods, however, predict parameters of a bounding box per anchor. Hyperparameters of anchors need to be tuned case-by-case for different tasks/datasets, reducing practicality. Moreover, anchors are sparsely distributed in the scene, leading to a significant class imbalance. An anchor is assigned as positive when its intersection-over-union (IoU) with a ground-truth box reaches above prescribed threshold; the number of positive anchors is less than 0.1% of all anchors in a typical point cloud.

As an alternative, we introduce a fully pillar-based (anchor-free) object detection model for autonomous driving. In principle, our method is an intuitive extension of PointPillars [88] and MVF [247] that uses pillar representations in multi-view feature learning and in pose estimation. In contrast to past works, we find that predicting box parameters per anchor is neither necessary nor effective for 3D object detection in autonomous driving. A critical new component of our model is a per-pillar prediction network, removing the necessity of anchor assignment. For each birds-eye view pillar, the model directly predicts the position and pose of the best possible box. This component improves performance and is significantly simpler than current state-of-the-art 3D object detection pipelines.

In addition to introducing this pillar-based object detection approach, we also propose ways to address other problems with previous methods. For example, we find the spherical projection in MVF causes unnecessary distortion of scenes and can actually degrade detection performance. So, we complement the conventional birds-eye view with a new cylindrical view, which does not suffer from perspective distortions. We also observe that current methods for pillar-to-point projection suffer from spatial aliasing, which we improve with bilinear interpolation.

To investigate the performance of our method, we train and test the model on the Waymo Open Dataset [181]. Compared to the top performers on this dataset [127, 247, 88], we show significant improvements by 6.87 3D mAP and 6.71 2D mAP for

vehicle detection. We provide ablation studies to analyze the contribution of each proposed module in §3.4 and show that each outperforms its counterpart by a large margin.

Contributions. We summarize our key contributions as follows:

- We present a fully pillar-based model for high-quality 3D object detection. The model achieves state-of-the-art results on the most challenging autonomous driving dataset.
- We design an pillar-based box prediction paradigm for object detection, which is simpler and stronger than its anchor-based and/or point-based counterparts.
- We analyze the multi-view feature learning module and find a cylindrical view is the best complementary view to a birds-eye view.
- We use bilinear interpolation in pillar-to-point projection to avoid quantization errors.
- We release our code to facilitate reproducibility and future research.

3.2 Related Work

Methods for object detection are highly successful in 2D visual recognition [49, 48, 148, 61, 102, 147]. They generally involve two aspects: backbone networks and detection heads. The input image is passed through a backbone network to learn latent features, while the detection heads make predictions of bounding boxes based on the features. In 3D, due to the sparsity of the data, many special considerations are taken to improve both efficiency and performance. Below, we discuss related works on general object detection, as well as more general methods relevant to learning on point clouds.

2D object detection. RCNN [49] pioneers the modern two-stage approach to object detection; more recent models often follow a similar template. RCNN uses a sim-

ple selective search to find regions of interest (region proposals) and subsequently applies a convolutional neural network (CNN) to bottom-up region proposals to regress bounding box parameters.

RCNN can be inefficient because it applies a CNN to each region proposal, or image patch. Fast RCNN [48] addresses this problem by sharing features for region proposals from the same image: it passes the image in a one-shot fashion through the CNN, and then region features are cropped and resized from the shared feature map. Faster RCNN [148] further improves speed and performance by replacing the selective search with region proposal networks (RPN), whose features can be shared.

Mask RCNN [61] is built on top of Faster RCNN. In addition to box prediction, it adds another pathway for mask prediction, enabling enables object detection, semantic segmentation, and instance segmentation using a single pipeline. Rather than using ROIPool [48] to resize feature patch to a fixed size grid, Mask RCNN proposes using bilinear interpolation (ROIAlign) to avoid quantization error. Beyond significant structural changes in the general two-stage object detection models, extensions using machinery from image processing and shape registration include: exploiting multi-scale information using feature pyramids [97], iterative refinement of box prediction [24], and using deformable convolutions [35] to get an adaptive receptive field. Recent works [250, 193, 246] also show anchor-free methods achieve comparable results to existing two-stage object detection models in 2D.

In addition to two-stage object detection, many works aim to design real-time object detection models via one-stage algorithms. These methods densely place anchors that define position priors and size priors in the image and then associate each anchor with the ground-truth using an intersection-over-union (IoU) threshold. The networks classify each anchor and regress parameters of anchors; non-maximum suppression (NMS) removes redundant predictions. SSD [102] and YOLO [146, 147] are representative examples of this approach. RetinaNet [98] is built on the observation that the extreme foreground-background class imbalance encountered during training causes one-stage detectors trailed the accuracy of two-stage detectors. It proposes a focal loss to amplify a sparse set of hard examples and to prevent easy negatives

from overwhelming the detector during training. Similar to image object detection, we also find the imbalance issue causes instability in 3D object detection. In contrast to RetinaNet, however, we replace anchors with pillar-centric predictions to alleviate imbalance.

Learning from point clouds. Point clouds provide a natural representation of 3D shapes and scenes. Due to irregularity and symmetry under reordering, however, defining convolution-like operations on point clouds is difficult.

PointNet [141] exemplifies a broad class of deep learning architectures that operate on raw point clouds. It uses a shared multi-layer perceptron (MLP) to lift points to high-demensional space and then aggregates features of points using symmetric set function. PointNet++ [143] exploits local context by building hierarchical abstraction of point clouds. DGCNN [215] uses graph neural networks (GCN) on the k-nearest neighbor graphs to learn geometric features. KPConv [190] defines a set of kernel points to perform deformable convolutions, providing more flexibility than fixed grid convolutions. PCNN [4] defines extension and restriction operations, mapping point cloud functions to volumetric functions and vice versa. SPLATNet [177] renders point clouds to lattice grid and perform lattice convolutions.

FlowNet3D [104] and MeteorNet [105] adopt these methods and learn pointwise flows on dynamical point clouds. In addition to high-level point cloud recognition, recent works [212, 213, 51, 158] tackle low-level registration problems using point cloud networks and show significant improvements over traditional optimization-based methods. These point-based approaches, however, are constrained by the number of points in the point clouds and cannot scale to large-scale settings such as autonomous driving. To that end, sparse 3D convolutions [55] have been proposed to apply 3D convolutions sparsely only on areas where points reside. Minkowski ConvNet [32] generalizes the definition of high-dimensional sparse convolution and improves 3D temporal perception.

3D object detection. The community has seen rising interest in 3D object detection thanks to the popularity of autonomous driving. VoxelNet [248] proposes a generic one-stage model for 3D object detection. It voxelizes the whole point cloud and uses dense 3D convolutions to perform feature learning. To address the efficiency issue, PIXOR [225] and PointPillars [88] both organize in vertical columns (pillars); a PointNet is used to transform features from points to pillars. MVF [247] learns to use complementary information from birds-eye view pillars and perspective view pillars. Complex-YOLO [166] extends YOLO to 3D scenarios and achieves real-time 3D perception; PointRCNN [164], on the other hand, adopts a RCNN-style detection pipeline. Rather than working in 3D, LaserNet [121] performs convolutions in raw range scans. Beyond point clouds only, recent works [84, 31, 221] combine point clouds with camera images to utilize additional information. Frustum-PointNet [140] leverages 2D object detectors to form a frustum crop of points and then uses a PointNet to aggregate features from points in frustum. [96] designs an end-to-end learnable architecture that exploits continuous convolutions to have better fused feature maps in every level. In addition to visual inputs, [224] shows that High-Definition (HD) maps provide strong priors that can boost the performance of 3D object detectors. [95] argues multi-tasking training can help the network to learn better representations than single-tasking. Beyond supervised learning, [217] investigates how to learn a perception model for unknown classes.

3.3 Method

In this section, we detail our approach to object pillar-based detection. We establish preliminaries about the pillar-point projection, PointPillars, and MVF in §3.3.1 and summarize our model in §3.3.2. Next, we discuss three critical new components of our model: cylindrical view projection (§3.3.3), a pillar-based prediction paradigm (§3.3.4), and a pillar-to-point projection module with bilinear interpolation (§3.3.5). Finally, we introduce the loss function in §3.3.6. For ease of comparison to previous work, we use the same notation as MVF [247].

3.3.1 Preliminaries

We consider a three-dimensional point cloud with N points $P = \{p_i\}_{i=0}^{N-1} \subseteq \mathbb{R}^3$ with K-dimensional features $F = \{f_i\}_{i=0}^{N-1} \subseteq \mathbb{R}^K$. We define two functions $F_V(p_i)$ and $F_P(v_j)$, where $F_V(p_i)$ returns the index j of p_i's corresponding pillar v_j and $F_P(v_j)$ gives the set of points in v_j. When projecting features from points to pillars, multiple points can potentially fall into the same pillar. To aggregate features from points in a pillar, a PointNet [141] (denoted as PN) is used to aggregate features from points to get pillar-wise features, where

$$f_j^{\text{pillar}} = \text{PN}(\{f_i | \forall p_i \in F_P(v_j)\}). \tag{3.1}$$

Then, pillar-wise features are further transformed through an additional convolutional neural network (CNN), notated $\phi^{\text{pillar}} = \Phi(f^{\text{pillar}})$ where Φ denotes the CNN. To retrieve point-wise features from pillars, the pillar-to-point feature projection is given by

$$f_i^{\text{point}} = f_j^{\text{pillar}} \quad \text{and} \quad \phi_i^{\text{point}} = \phi_j^{\text{pillar}}, \quad \text{where} \quad j = F_V(p_i). \tag{3.2}$$

While PointPillars only considers birds-eye view pillars and makes predictions based on the birds-eye feature map, MVF also incorporates spherical pillars. Given a point $p_i = (x_i, y_i, z_i)$, its spherical coordinates $(\varphi_i, \theta_i, d_i)$ are defined via

$$\varphi_i = \arctan \frac{y_i}{x_i} \qquad \theta_i = \arccos \frac{z_i}{d_i} \qquad d_i = \sqrt{x_i^2 + y_i^2 + z_i^2}. \tag{3.3}$$

We can denote the established point-pillar transformations as $(F_V^{\text{bev}}(p_i), F_P^{\text{bev}}(v_j))$ and $(F_V^{\text{spv}}(p_i), F_P^{\text{spv}}(v_j))$ for the birds-eye view and the spherical view, respectively. In MVF, pillar-wise features are learned independently in two views; then the point-wise features are gathered from those views using Eq. 3.2. Next, the fused point-wise features are projected to birds-eye view again and embedded through a CNN as in PointPillars.

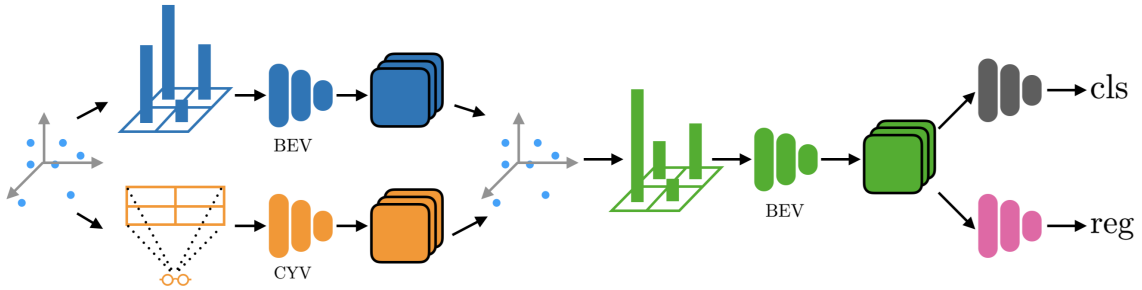
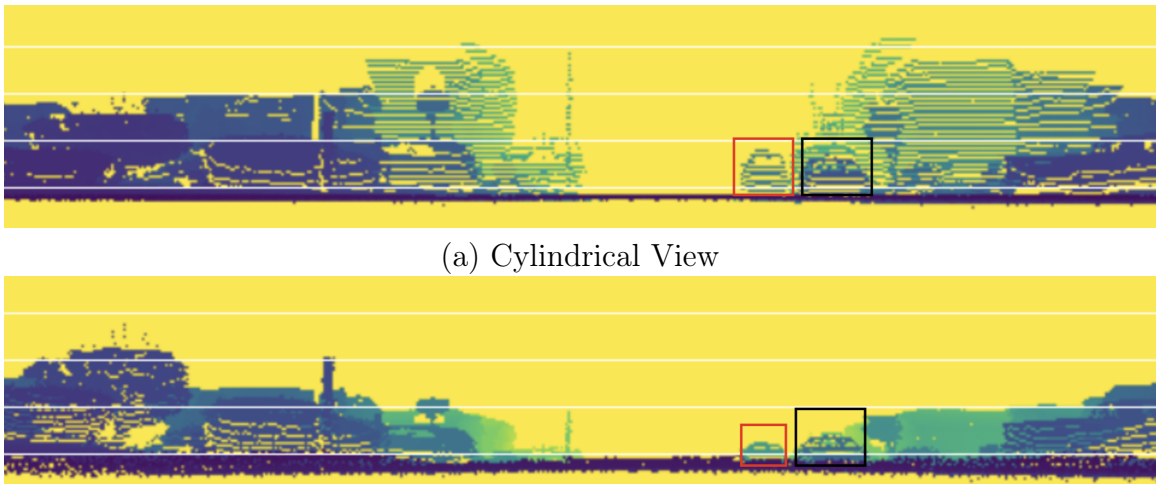


Figure 3-1: Overall architecture of the proposed model: a point cloud is projected to BEV and CYV respectively; then, view-specific feature learning is done in each view; third, features from multiple views are aggregated; next, point-wise features are projected to BEV again for further embedding; finally, in BEV, a classification network and a regression network make predictions per pillar. BEV: birds-eye view; CYV: cylindrical view; cls: per pillar classification target; reg: per pillar regression target.

The final detection head for both PointPillars and MVF is an anchor-based module. Anchors, parameterized by $(x^a, y^a, z^a, l^a, w^a, h^a, \theta^a)$, are densely placed in each cell of the final feature map. During pre-processing, an anchor is marked as "positive" if its intersection-over-union (IoU) with a ground-truth box is above a prescribed positive threshold, and "negative" if its IoU is below a negative threshold; otherwise, the anchor is excluded from the final loss computation.

3.3.2 Overall architecture

An overview of our proposed model is shown in Figure 3-1. The input point cloud is passed through the birds-eye view network and the cylindrical view network individually. Then, features from different views are aggregated in the same way with MVF. Next, features are projected back to birds-eye view and passed through additional convolutional layers. Finally, a classification network and a regression network make the final predictions per birds-eye view pillar. We *do not* use anchors in any stage. We describe each module in detail below.



(a) Cylindrical View

(b) Spherical View

Figure 3-2: Comparison of (a) cylindrical view projection and (b) spherical view projection. We label two example cars in these views. Objects in spherical view are distorted (in Z-axis) and no longer in physical scale.

3.3.3 Cylindrical view

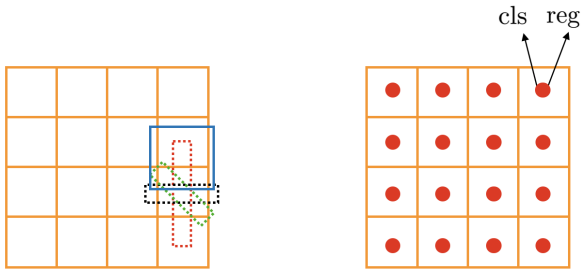
In this section, we formulate the cylindrical view projection. The cylindrical coordinates (ρ_i, φ_i, z_i) of a point p_i are given by the following:

$$\rho_i = \sqrt{x_i^2 + y_i^2} \qquad \varphi_i = \arctan \frac{y_i}{x_i} \qquad z_i = z_i. \tag{3.4}$$

Cylindrical pillars are generated by grouping points that have the same φ and z coordinates. Although it is closely related to the spherical view, the cylindrical view does not introduce distortion in the Z-axis. We show an example in Figure 3-2, where cars are clearly visible in the cylindrical view but not distinguishable in the spherical view. In addition, objects in spherical view are no longer in their physical scales—e.g., distant cars become small.

3.3.4 Pillar-based prediction

The pillar-based prediction module consists of two networks: a classification network and a regression network. They both take the final pillar features ϕ^{pillar} from birds-eye



(a) Prediction per anchor (b) Prediction per pillar

Figure 3-3: Differences between prediction per anchor and prediction per pillar. (a) Multiple anchors with different sizes and rotations are densely placed in each cell. Anchor-based models predict parameters of bounding box for the positive anchor. For ease of visualization, we only show three anchors. Grid (in orange): birds-eye view pillar; dashed box (in red): a positive match; dashed box (in black): a negative match; dashed box (in green): invalid anchors because their IoUs are above negative threshold and below positive threshold. (b) For each pillar (center), we predict whether it is within a box and the box parameters. Dots (in red): pillar center.

view. The prediction targets are given by

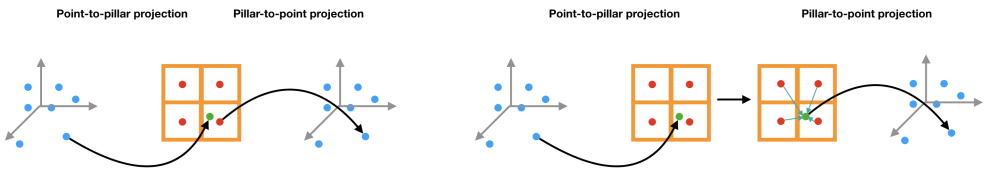
$$\mathrm{p} = f_{\mathrm{cls}}(\phi^{\mathrm{pillar}}) \quad \text{and} \quad (\Delta_x, \Delta_y, \Delta_z, \Delta_l, \Delta_w, \Delta_h, \theta^p) = f_{\mathrm{reg}}(\phi^{\mathrm{pillar}}), \tag{3.5}$$

where p denotes the probability of whether a pillar is a positive match to a ground-truth box and $(\Delta_x, \Delta_y, \Delta_z, \Delta_l, \Delta_w, \Delta_h, \theta^p)$ are the regression targets for position, size, and heading angle of the bounding box.

The differences between anchor-based method and pillar-based method are explained in Figure 3-3. Rather than associating a pillar with an anchor and predicting the targets with reference to the anchor, the model (on the right) directly makes a prediction per pillar.

3.3.5 Bilinear interpolation

The pillar-to-point projection used in PointPillars [88] and MVF [247] can be thought of as a version of nearest neighbor interpolation, however, which often introduces quantization errors. Rather than performing nearest neighbor interpolation, we propose using bilinear interpolation to learn spatially-consistent features. We describe



(a) Nearest neighbor interpolation (b) Bilinear interpolation

Figure 3-4: Comparison between nearest neighbor interpolation and bilinear interpolation in pillar-to-point projection. Rectangles (in orange): birds-eye view pillars; dots (in blue): points in 3D Cartesian coordinates; dots (in green): points projected to pillar frame; dots (in red): centers of pillars.

the formulation of nearest neighbor interpolation and bilinear interpolation in the context of pillar-to-point projection below.

As shown in Figure 3-4 (a), we denote the center of a pillar v_j as c_j where c_j is defined by its 2D or 3D coordinates. Then, the point-to-pillar mapping function is given by

$$F_V(p_i) = j, \quad \text{where } \|p_i - c_j\| \leq \|p_i - c_k\| \quad \forall k \tag{3.6}$$

and $\|\cdot\|$ denotes the $\mathcal{L}_2$ norm. When querying the features for a point p_i from a collection pillars, we determine the corresponding pillar v_j by checking F_V and copy the features of v_j to p_i—that is $\phi_i^{\text{point}} = \phi_j^{\text{pillar}}$.

This operation, though straightforward, leads to undesired spatial misalignment: if two points p_i and p_j with different spatial locations reside in the same pillar, their pillar-to-point features are the same. To address this issue, we propose using bilinear interpolation for the pillar-to-point projection. As shown in Figure 3-4 (b), the bilinear interpolation provides consistent spatial mapping between points and pillars.

3.3.6 Loss function

We use the same loss function as in SECOND [223], PointPillars [88], and MVF [247]. The loss function consists of two terms: a pillar classification loss and a pillar regression loss. The ground-truth bounding box is parametrized as $(x^g, y^g, z^g, l^g, w^g, h^g, \theta^g)$; the center of pillar is (x^p, y^p, z^p); and the prediction targets for the bounding box are

$(\Delta_x, \Delta_y, \Delta_z, \Delta_l, \Delta_w, \Delta_h, \theta^p)$ as in §3.3.4. Then, the regression loss is:

$$\begin{aligned} L_{\text{reg}} = \text{SmoothL1}(\theta^p - \theta^g) + \sum_{r \in \{x,y,z\}} \text{SmoothL1}(r^p - r^g - \Delta_r) \\ + \sum_{r \in \{l,w,h\}} \text{SmoothL1}(\log(r^g) - \Delta_r) \end{aligned} \tag{3.7}$$

where

$$\text{SmoothL1}(d) = \begin{cases} 0.5 \cdot d^2 \cdot \sigma^2, & \text{if} \quad |d| < \frac{1}{\sigma^2} \\ |d| - \frac{1}{2\sigma^2}, & \text{otherwise.} \end{cases} \tag{3.8}$$

We take $\sigma = 3.0$. For pillar classification, we adopt the focal loss [98]:

$$L_{\text{cls}} = -\alpha(1 - \text{p})^\gamma \log \text{p}. \tag{3.9}$$

We use $\alpha = 0.25$ and $\gamma = 2$, as recommended by [98].

3.4 Experiments

Our experiments are divided into four parts. First, we demonstrate performance of our model for vehicle and pedestrian detection on the Waymo Open Dataset [181] in §3.4.1. Then, we compare anchor-, point-, and pillar-based detection heads in §3.4.2. We compare different combinations of views in §3.4.3. Finally, we test the effects of bilinear interpolation in §3.4.4.

Dataset. The *Waymo Open Dataset* [181] is the largest publicly-available 3D object detection dataset for autonomous driving. The dataset provides 1000 sequences total; each sequence contains roughly 200 frames. The training set consists of 798 sequences with 158,361 frames, containing 4.81M vehicle and 2.22M pedestrian boxes. The validation set consists of 202 sequences with 40,077 frames, containing 1.25M vehicle and 539K pedestrian boxes. The detection range is set to $[-75.2, 75.2]$ meters (m) horizontally and $[-3, 3]$ m vertically.

Metrics. For our experiments, we adopt the official evaluation protocols from the Waymo Open Dataset. In particular, we employ the 3D and BEV mean average precision (mAP) metrics. The orientation-aware IoU threshold is 0.7 for vehicles and 0.5 for pedestrians. We also break down the metrics according to the distances between the origin and ground-truth boxes: 0m–30m, 30m–50m, and 50m–infinity (Inf). The dataset is split based on the number of points in each box: LEVEL_1 denotes boxes that have more than 5 points while LEVEL_2 denotes boxes that have 1-5 points. Following StarNet [127], MVF [247], and PointPillars [88] as reimplemented in [181], we evaluate our models on LEVEL_1 boxes.

Implementation details. Our model consists of three parts: a multi-view feature learning network; a birds-eye view PointPillar [88] backbone; and a per-pillar prediction network. In the multi-view feature learning network, we project point features to both birds-eye view pillars and cylindrical pillars. For each view, we apply three ResNet [62] layers with strides $[1, 2, 2]$, which gradually downsamples the input feature to $1/1$, $1/2$, and $1/4$ of the original feature map. Then, we project the pillar-wise features to points using bilinear interpolation and concatenate features from both views and from a parallel PointNet with one fully-connected layer. Then, we transform the per-point features to birds-eye pillars and use a PointPillars [88] backbone with three blocks to further improve the representations. The three blocks have $[4, 6, 6]$ convolutional layers, with dimensions $[128, 128, 256]$. Finally, for each pillar, the model predicts the categorical label using a classification head and 7 DoF parameters of its closest box using a regression head. The classification head and regression head both have four convolutional layers with 128 hidden dimensions. We use BatchNorm [70] and ReLU [125] after every convolutional layer.

Training. We use the Adam [80] optimizer to train the model. The learning rate is initially 3×10^{-4} and then linearly increased to 3×10^{-3} in the first 5 epochs. Finally, the learning rate is decreased to 3×10^{-6} using cosine scheduling [109]. We train the model for 75 epochs in 64 TPU cores.

Inference. The input point clouds pass through the whole model once to get the initial predictions. Then, we use non-maximum suppression (NMS) [48] to remove redundant bounding boxes. The oriented IoU threshold of NMS is 0.7 for vehicle and 0.2 for pedestrian. We keep the top 200 boxes for metric computation. The size of our model is on a par with MVF; the model runs at 15 frames per second (FPS) on a Tesla V100.

3.4.1 Results compared to state-of-the-art

We compare the proposed method to top-performing methods on the Waymo Open Dataset. StarNet [127] is a purely point-based method with a small receptive field, which performs well for small objects such as pedestrians but poorly for large objects such as vehicles. LaserNet [121] operates on range images, which is similar to our cylindrical view feature learning. Although PointPillars [88] does not evaluate on this dataset, MVF [247] and the Waymo Open Dataset [181] both re-implement the PointPillars. So we adopt the results from MVF [247] and [181]. The re-implementation from [181] uses larger feature map resolution in the first PointPillars block; therefore, it outperforms the re-implementation from MVF [247].

MVF [247] extends PointPillars [88] with the same backbone networks and multi-view feature learning. We use the same backbone networks with PointPillars [88] and MVF [247].

As shown in Table 3.1 and Table 3.2, we achieve significantly better results for both pedestrians and vehicles. Especially for distant vehicles (30m–Inf), the improvements are more significant. This is inline with our hypothesis: in distant areas, anchors are less possible to match to a ground-truth box; therefore, the imbalance problem is more serious. Also, to verify the improvements are *not* due to differences in training protocol, we re-implement PointPillars; using our training protocol, it achieves 54.25 3D mAP and 70.59 2d mAP, which are worse than the re-implementations in [247] and [181] . Therefore, we can conclude the improvements are due to the three new components added by our proposed model.

Method	BEV mAP (IoU=0.7)				3D mAP (IoU=0.7)			
	Overall	0 - 30m	30 - 50m	50m - Inf	Overall	0 - 30m	30 - 50m	50m - Inf
StarNet [127]	-	-	-	-	53.7	-	-	-
LaserNet [121]	71.57	92.94	74.92	48.87	55.1	84.9	53.11	23.92
PointPillars¶ [88]	80.4	92.0	77.6	62.7	62.2	81.8	55.7	31.2
PointPillars‡ [88]	70.59	86.63	69.34	49.3	54.25	76.31	48.08	24.21
PointPillars† [88]	75.57	92.1	74.06	55.47	56.62	81.01	51.75	27.94
MVF [247]	80.4	93.59	79.21	63.09	62.93	86.3	60.2	36.02
Ours	**87.11**	**95.78**	**84.74**	**72.12**	**69.8**	**88.53**	**66.5**	**42.93**
Improvements	**+6.71**	**+2.19**	**+5.53**	**+9.03**	**+6.87**	**+2.23**	**+6.3**	**+6.91**

Table 3.1: Results on vehicle. ¶: re-implemented by [181], the feature map in the first PointPillars block is two times as big as in others; ‡: our re-implementation; †: re-implemented by [247].

Method	BEV mAP (IoU=0.5)				3D mAP (IoU=0.5)			
	Overall	0 - 30m	30 - 50m	50m - Inf	Overall	0 - 30m	30 - 50m	50m - Inf
StarNet [127]	-	-	-	-	66.8	-	-	-
LaserNet [121]	70.01	78.24	69.47	52.68	63.4	73.47	61.55	42.69
PointPillars¶ [88]	68.7	75.0	66.6	58.7	60.0	68.9	57.6	46.0
PointPillars† [88]	68.57	75.02	67.11	53.86	59.25	67.99	57.01	41.29
MVF [247]	74.38	80.01	72.98	62.51	65.33	72.51	63.35	50.62
Ours	**78.53**	**83.56**	**78.7**	**65.86**	**72.51**	**79.34**	**72.14**	**56.77**
Improvements	**+4.15**	**+3.55**	**+5.72**	**+3.35**	**+5.71**	**+6.83**	**+8.77**	**+6.15**

Table 3.2: Results on pedestrian. ¶: re-implemented by [181]. †: re-implemented by [247].

3.4.2 Comparing anchor-based, point-based, and pillar-based prediction

In this experiment, we compare to alternative means of making predictions: predicting box parameters per anchor or per point. For these three detection head choices, we use the same overall architecture with experiments in §3.4.1. We conduct this ablation study on vehicle detection.

Anchor-based model. We use the parameters and matching strategy from PointPillars [247] and MVF [247]. Each class anchor is described by a width, length, height, and center position and is applied at two orientations: 0° and 90°. Anchors are matched to ground-truth boxes using the 2D IoU with the following rules: a positive match is either the highest with a ground truth box, or above the positive match

Method	BEV mAP (IoU=0.7)				3D mAP (IoU=0.7)			
	Overall	0 - 30m	30 - 50m	50m - Inf	Overall	0 - 30m	30 - 50m	50m - Inf
Anchor-based	78.84	91.91	74.99	59.59	59.78	82.69	53.38	31.02
Point-based	79.77	92.35	76.58	60.00	60.6	83.66	55.48	30.95
Pillar-based	**87.11**	**95.78**	**84.74**	**72.12**	**69.8**	**88.53**	**66.5**	**42.93**

Table 3.3: Comparison of making prediction per anchor, per point, or per pillar.

threshold (0.6); while a negative match is below the negative threshold (0.45). All other anchors are ignored in the box parameter prediction. The model is to predict whether a anchor is positive or negative, and width, length, height, heading angle, and center position of the bounding box.

Point-based model. The per-pillar features are projected to points using bilinear interpolation. Then, we assign each point to its surrounding box with the following rules: if a point is inside a bounding box, we assign it as a foreground point; otherwise it is a background point. The model is asked to predict the binary label whether a point is a foreground point or a background point. For positive points, the model also predicts the width, length, height, heading angle, and center offsets (with reference to point positions) of their associated bounding boxes. Conceptually, this point-based model is an instantiation of VoteNet [139] applied to this autonomous driving scenario. The key difference is: the VoteNet [139] uses a PointNet++ [143] backbone while we use a PointPillars [247] backbone.

Pillar-based model. Since we use the same architecture, we take the results from §3.4.1. As Table 3.3 shows, anchor-based prediction performs the worst while point-based prediction is slightly better. Our pillar-based prediction is top performing among these three choices. The pillar-based prediction model achieves the best balance between coarse prediction (per anchor) and fine-grained prediction (per point).

View	Coordinates	Range
3D Cartesian	(x, y, z)	(-75.2, 75.2)m, (-75.2, 75.2)m, (-3, 3)m
BEV	(x, y, z)	(-75.2, 75.2)m, (-75.2, 75.2)m, (-3, 3)m
SPV	$(\arctan(\frac{y}{x}), \arccos(\frac{z}{\sqrt{x^2+y^2+z^2}}), \sqrt{x^2+y^2+z^2})$	(0, 2π), (0.485π, 0.55π), (0, 107)m,
XZ view	(x, y, z)	(-75.2, 75.2)m , (-75.2, 75.2)m, (-3, 3)m
CYV	$(\sqrt{x^2+y^2}, \arctan(\frac{y}{x}), z)$	(0, 107)m, (0, 2π), (-3, 3)m

Table 3.4: View projection

3.4.3 View combinations

In this section, we test different view projections in multi-view feature learning: birds-eye view (BEV), spherical view (SPV), XZ view, cylindrical view (CYV), and their combinations. First, we define the vehicle frame: the X-axis is positive forwards, the Y-axis is positive to the left, and the Z-axis is positive upwards. Then, we can write the coordinates of a point $p = (x, y, z)$ in different views; the range of each view is given in Table 3.4. The pillars in the corresponding view are generated by projecting points from 3D to 2D using the coordinate transformation. One exception is in XZ view, in which we use separate pillars for positive part and negative part for Y-axis to avoid undesired occlusions.

We show results of different view projections and their combinations in Table 3.5 for vehicle detection. When using a single view, the cylindrical view achieves significantly better results than the alternatives, especially in the long-range detection case (50m–Inf). When combining with the birds-eye view, the cylindrical view still outperforms others in all metrics. The spherical view, albeit similar to cylindrical view, introduces distortion in Z-axis, degrading performance relative to the cylindrical view. On the other hand, the XZ view does not distort the Z-axis, but occlusions in Y-axis prevent it from achieving as strong results as the cylindrical view. We also test with additional view combinations (such as using birds-eye view, spherical view, and cylindrical view) and do not observe any improvements over combining just the birds-eye view and the cylindrical view.

Method	BEV mAP (IoU=0.7)				3D mAP (IoU=0.7)			
	Overall	0 - 30m	30 - 50m	50m - Inf	Overall	0 - 30m	30 - 50m	50m - Inf
BEV	81.58	92.69	78.64	63.52	61.86	83.61	56.91	33.53
SPV	81.58	93.7	78.43	63.2	62.08	83.31	56.59	34.05
XZ	81.49	94.03	78.04	62.32	61.67	84.64	55.01	32.06
CYV	83.43	95.21	81.49	66.77	64.77	87.09	60.91	37.99
BEV + SPV	85.09	95.19	82.01	69.13	66.31	86.56	61.15	39.36
BEV + XZ	82.45	94.1	79.19	63.91	62.76	85.08	56.8	33.36
BEV + CYV	**87.11**	**95.78**	**84.74**	**72.12**	**69.8**	**88.53**	**66.5**	**42.93**

Table 3.5: Ablation on view combinations.

Method	BEV mAP (IoU=0.7)				3D mAP (IoU=0.7)			
	Overall	0 - 30m	30 - 50m	50m - Inf	Overall	0 - 30m	30 - 50m	50m - Inf
Nearest neighbor	84.67	94.42	79.2	65.77	64.76	85.55	59.21	35.63
Bilinear	**87.11**	**95.78**	**84.74**	**72.12**	**69.8**	**88.53**	**66.5**	**42.93**

Table 3.6: Comparing bilinear interpolation and nearest neighbor projection.

3.4.4 Bilinear interpolation or nearest neighbor interpolation?

In this section, we compare bilinear interpolation to nearest neighbor interpolation in pillar-to-point projection (for vechile detection). The architectures remain the same for both alternatives except the way we project multi-view features from pillars to points: In nearest neighbor interpolation, for each query point, we sample its closest pillar center and copy the pillar features to it, while in bilinear interpolation, we sample its four pillar neighbors and take a weighted average of the corresponding pillar features. Table 3.6 shows bilinear interpolation systematically outperforms its counterpart in all metrics. This observation is consistent with the comparison of ROIAlign [61] and ROIPool [148] in 2D.

3.5 Discussion

Our model achieves state-of-the-art results on the largest publicly-available 3D object detection dataset. The success of our model suggests many designs from 2D object detection/visual recognition are *not* directly applicable to 3D scenarios. In addition, we find that learning features in correct views is import to the performance of the

model.

Our experiments also suggest several avenues for future work. For example, rather than hand-designing a view projection as we do in §3.3.3, learning an optimal view transformation from data may provide further performance improvements. Learning features using 3D sparse convolutions rather than 2D convolutions could improve performance as well. Also, following post-processing free object detection models designed for images, adding a set-to-set module will further simplifies the current pipeline. This is studied in the next two chapters, which aim to build streamlined architectures for 3D object detection from point clouds and from images.

Finally, we hope to find more applications of the proposed model beyond object detection. For example, we could incorporate instance segmentation, which may help with fine-grained 3D recognition and robotic manipulation.

Chapter 4

Object DGCNN

Methods for 3D object detection have progressed rapidly, yielding deployable autonomous driving perception systems. Following common practice in 2D vision, 3D object detection often employs complex training and testing pipelines including many post-processing operations to achieve superior performance. These operations are typically non-parallelizable and inefficient even with modern deep learning frameworks, implying a steep trade-off between between efficiency and effectiveness.

Modern methods usually employ two stages [164, 163], including a region proposal network [148] that can introduce significant training overhead. Subsequent efforts simplify this pipeline for 3D object detection. PointPillars [88] introduces a one-stage anchor-based design, simplfying training. PillarOD [209] and CenterPoint [233] improve the one-stage model by making per-pillar predictions, that is, one prediction per point on the ground plane. They assign ground-truth bounding boxes to multiple outputs while training to ease optimization. However, they predict redundant boxes, which can overlap in the same positions; extra boxes are eliminated *a posteriori* using non-maximum suppression (NMS). As suggested in previous chapters, it remains elusive to remove hand-designed components like NMS in training and testing.

4.1 Introduction

We introduce Object DGCNN, a streamlined architecture for 3D object detection from point clouds. Like DETR for 2D object detection [25], we predict a set of bounding boxes from the raw data, enabling an NMS-free pipeline that achieves real-time performance. A critical new component is to treat each object query as a point in a set whose embedding is learned using DGCNN [215]. Compared to the self-attention module [200] in DETR, DGCNN leverages a *sparse* set of object relations, which reflects the real object distribution in the scene. In contrast to PointPillars [88], PillarOD [209], and CenterPoint [233], our method *does not* require post-processing.

We also provide a knowledge distillation approach customized to 3D object detection. Existing methods typically distill dense feature maps from a teacher model to a student model, whose training objective does not necessarily capture 3D object detection performance [210]. In contrast, we propose set-to-set distillation training that aligns the outputs of the teacher and the student in a permutation-invariant fashion. This process is enabled by the unified Object DGCNN architecture. In addition to obtaining better performance, through this process our model can benefit from privileged information (e.g., dense point clouds) only available at training time.

Contributions. We summarize our key contributions as follows:

- We propose a post-processing-free 3D object detection model achieving state-of-the-art performance. To our knowledge, this is the first NMS-free 3D object detector.
- We generalize DGCNN to model objects as a point set. The DGCNN module outperforms its self-attention counterpart thanks to its sparse structure.
- We propose a set-to-set distillation method for 3D object detection. In our construction, knowledge distillation on object detection simply penalizes differences between the outputs of the teacher model and the student model.
- We show our model can use privileged information (such as dense point clouds) that is naturally available at training time to improve the model performance

at inference time.

- We release our code to promote reproducibility and future research.

4.2 Related Work

2D object detection. Object recognition research has been transitioning from models with hand-crafted components to models with limited post-processing. One-stage detectors [102, 147, 98] remove the complicated region proposal networks in two-stage objectors [148, 61], yielding more efficient training and testing. Anchor-free methods [246, 89] further simplify the one-stage pipeline by shifting from per-anchor prediction to per-pixel prediction. However, these methods still make dense predictions and rely on NMS to reduce redundancy. To alleviate this issue, DETR [25] formulates object detection as a set-to-set prediction problem. It introduces a set-to-set loss that implicitly penalizes redundant boxes, removing the necessity of post-processing. To accelerate convergence, Deformable DETR [253] proposes deformable self-attention and streamlines the optimization process. Our method also formulates 3D object detection as set prediction, but with a customized design for 3D.

3D object detection. VoxelNet [248] generalizes one-stage object detection to 3D. It uses 3D dense convolutions to learn representations on voxelized point clouds, which is too inefficient to capture fine-grained features. To address that, PIXOR [225] and PointPillars [88] project points to a birds-eye view (BEV) and operate on 2D feature maps; PointNet [141] aggregates features within each BEV pixel. We use a variant of PointPillars [88] for 3D detection (§4.4). These methods are efficient but drop information along the vertical axis. To accompany the BEV projection, MVF [247] adds a spherical projection. Pillar-OD [209] and CenterPoint [233] use pillar-centric object detection, making predictions per BEV pixel (pillar) rather than per anchor. These anchor-free methods simplify 3D object detection while maintaining efficiency. Beyond SSD-style [102] one-stage models, Complex-YOLO [166] extends YOLO to 3D for real-time perception. PointRCNN [164] employs a two-stage architecture for high-quality detection. To improve representations of two-stage models, PVRCNN [163]

proposes a point-voxel feature set abstraction layer to leverage the flexible receptive fields of PointNet-based networks. Unlike works on point clouds, LaserNet [121] operates on raw range scans with comparable performance. [84, 31, 221] combine point clouds with camera images. Frustum-PointNet [140] leverages 2D object detectors to form a frustum crop of points and then uses PointNet to aggregate features. [96] describes an end-to-end learnable architecture that exploits continuous convolutions to fuse feature maps. VoteNet [139, 138] generalizes Hough voting [66] for 3D object detection in point clouds. DOPS [126] extends VoteNet and predicts 3D object shapes. In addition to visual input, [224] shows that high-definition (HD) maps can boost performance of 3D object detectors. [95] argues that multi-tasking can learn better representations than single-tasking. Beyond supervised learning, [217] learns a perception model for unknown classes.

DGCNN. DGCNN [215] pioneered learning point cloud representations via dynamic graphs. It models point clouds as connected graphs, which are dynamically built using k-nearest neighbors in the latent space. DGCNN learns per-point features through message passing. However, it operates on point clouds for single object recognition and semantic segmentation. One of our key contributions is to generalize DGCNN to model scene-level object relations for 3D detection.

Knowledge distillation (KD). KD compresses knowledge from an ensemble of models into a single smaller model [22]. [63] generalizes this idea and combines it with deep learning. KD transfers knowledge from a teacher model to a student model by minimizing a loss, in which the target is the distribution of class probabilities induced by the teacher. [150, 235, 195, 133, 232, 192, 128, 59, 29, 191, 47, 33] improve knowledge distillation for classification. Beyond image classification, KD has been extended to improve object detection. [28] leverages FitNets for object detection, addressing obstacles such as class imbalance, loss instability, and feature distribution mismatch. [93] distills between region proposals, accelerating training with added instability. To address this issue, [205] uses fine-grained representation imitation using object masks. [103] uses KD to tackle a continual learning problem.

Privileged information. [199] introduces the framework of learning with priv-

ileged information in the context of support vector machines (SVMs), wherein additional information is accessible during training but not testing. [108] unifies KD and learning using privileged information theoretically. [179] identifies practical applications, e.g., transferring knowledge from localized data to non-localized data, from high resolution to low resolution, from color images to edge images, and from regular images to distorted images. To mediate uncertainty and improve training efficiency, [87] makes the variance of Dropout [175] a function of privileged information. We extend these methods to 3D data, in which privileged information consists of dense point clouds aggregated from LiDAR sequences.

4.3 Overview

Our target application of object detection differs from the recognition and segmentation tasks considered for DGCNN. Our point clouds typically contain too many points to apply DGCNN and its peers directly to the entire scene. Moreover, the size of our output set, a small set of bounding boxes, differs from the size of our input set, a huge set of points in $\mathbb{R}^3$.

Following state-of-the-art in large-scale object detection, our pipeline learns a grid-based intermediate representation to capture local features (§4.4). We test two standard learning-based methods for collecting local point cloud features on a birds-eye view (BEV) grid. While in principle it might be possible to avoid grids entirely in our pipeline, this BEV representation is far more efficient and—as observed in previous work—is sufficient to find objects reliably in autonomous driving, where there is likely only one object above any given grid cell on the ground plane.

Our main architecture contribution is the Object DGCNN pipeline (§4.5), which transitions from this BEV grid of features to a *set* of object bounding boxes. Object DGCNN draws inspiration from the DGCNN architecture; its layers alternate between local feature transformations and k-nearest neighbor aggregation to capture relationships between objects. Unlike conventional DGCNN, however, Object DGCNN incorporates features from the BEV grid in each of its layers; each layer

incorporates several queries into the BEV to refine object position estimates. The output of Object DGCNN is a *set* of objects in the scene. We use a permutation-invariant loss (4.10) to measure divergence from the ground truth set of objects.

The pipeline above does not require hand-designed post-processing like NMS; our output boxes are usable directly for object detection. Beyond simplifying the object detection pipeline, this allows us to propose object detection-specific distillation procedures (§4.7.3) that further improve performance. These use one network to train another, e.g., to train a network operating on sparse point clouds to output features that imitate those learned by a network trained on denser point clouds.

4.4 Local Features

We begin with a point cloud $\mathbf{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_N\} \subset \mathbb{R}^3$ with per-point features $\mathcal{F} = \{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_i, \ldots, \boldsymbol{f}_N\} \subset \mathbb{R}^K$, ground-truth bounding boxes $\mathcal{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_j, \ldots, \boldsymbol{b}_M\} \subset \mathbb{R}^9$, and categorical labels $\mathcal{C} = \{c_j, \ldots, c_j, \ldots, c_M\} \subset \mathbb{Z}$. Each $\boldsymbol{b}_j$ contains position, size, heading angle, and velocity in the birds-eye view (BEV); our architecture aims to predict these boxes and their labels from the point cloud and its features.

As an initial step, modern 3D object detection models scatter points into either BEV pillars or 3D voxels and then use convolutional neural networks to extract features on a grid. This strategy accelerates object detection for large point clouds. We test two neural network architectures for BEV feature extraction, detailed below.

PointPillars [88] maps sparse point clouds onto a dense BEV pillar map on which 2D convolutions can be applied. Suppose $F_P(i)$ returns the points in pillar i, that is, the set of points in a vertical column above point i on the ground. When collecting features from points to pillars, multiple points can fall into the same pillar. In this case, PointNet [141] (PN) is used to obtain pillar-wise features:

$$\boldsymbol{f}_i^{\text{pillar}} = \text{PN}(\{\boldsymbol{f}_j | \boldsymbol{x}_j \in F_P(\boldsymbol{p}_i)\}), \tag{4.1}$$

where $\boldsymbol{f}_i^{\text{pillar}}$ is the feature of pillar $\boldsymbol{p}_i$. We set the features for empty pillars to $\mathbf{0}$.

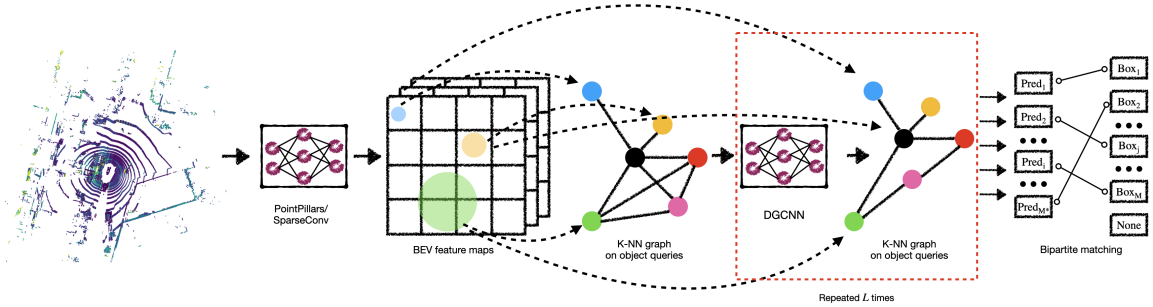


Figure 4-1: Overview. Point cloud features are learned in BEV, followed by L DGCNNs to model object relations. We predict a set of bounding boxes and compute loss in a one-to-one manner.

This results in a dense 2D grid $\mathcal{F}^{\text{pillar}} \subset \mathbb{R}^{H^{\text{p}} \times W^{\text{p}} \times C^{\text{p}}}$, where $H^{\text{p}}, W^{\text{p}}$ and C^{p} are the height, width, number of channels of this 2D pillar map, respectively. Multiple stacked convolutional layers further embed the pillar features to the final feature map $\mathcal{F}^{\text{d}} \subset \mathbb{R}^{H^{\text{d}} \times W^{\text{d}} \times C^{\text{d}}}$.

An alternative BEV embedding is SparseConv [55]. If $F_V(i)$ returns the set of points in voxel i, SparseConv collects point-wise features into voxel-wise features by

$$\boldsymbol{f}_i^{\text{voxel}} = \text{PN}(\{\boldsymbol{f}_j | \boldsymbol{x}_j \in F_V(i)\}), \tag{4.2}$$

where $\boldsymbol{f}_i^{\text{voxel}}$ contains the features of voxel i. In contrast to PointPillars, SparseConv conducts 3D sparse convolutions to refine the voxel-wise features. Finally, we compress these sparse voxels to a BEV 2D grid by filling empty voxels with zeros and averaging along the z-axis. For ease of notation, we also denote the resulting 2D grid $\mathcal{F}^{\text{d}} \subset \mathbb{R}^{H^{\text{d}} \times W^{\text{d}} \times C^{\text{d}}}$.

4.5 Object DGCNN

After obtaining the BEV features $\mathcal{F}^{\text{d}}$ using one of the architectures above, we predict a set of bounding boxes as well as a label for each box. The key difference between our architecture and most recent 3D object detection methods is that ours produces a *set* of bounding boxes rather than a box per grid cell followed by NMS, as in [209, 233]. Hence, we need to transition from a grid of per-pillar features to an unordered set of

objects; we detail our approach below. We address two key issues: prediction of the bounding boxes and evaluation of the loss.

Desiderata. Object DGCNN uses a DGCNN-inspired architecture but incorporates grid-based BEV features, built on the philosophy that local features (§4.4) are reasonable to store on a dense grid, but object predictions are better modeled using sets. Hence, we require a new architecture and set-to-set loss that encourage bounding box diversity.

Object DGCNN uses L layers that follow a series of set-based computations to produce bounding box predictions from the BEV feature maps. Each layer employs the following steps (Figure 4-1):

- predict a set of query points and attention weights;
- collect BEV features from keypoints determined by the queries; and
- model object-object interactions via DGCNN.

Each layer results in a more refined set of bounding box predictions, one per query. At the end of these layers, we match the prediction set with the ground-truth set in a one-to-one fashion and evaluate a set-to-set object detection loss.

Single layer. Inspired by DETR [25], each layer $\ell \in \{0, \ldots, L-1\}$ of Object DGCNN operates on a set of *object queries* $\mathbb{Q}_\ell = \{\boldsymbol{q}_{\ell 1}, \ldots, \boldsymbol{q}_{\ell M^*}\} \subset \mathbb{R}^Q$, producing a new set $\mathbb{Q}_{\ell+1}$. Although queries are fully learnable, our intuition is that they represent progressively refined object positions.

The initial set of object queries $\mathbb{Q}_0$ is learned jointly with the neural network weights, yielding a dataset-specific prior. Beyond this fixed initial set, below we detail how to incorporate scene information to obtain $\mathbb{Q}_{\ell+1}$ from $\mathbb{Q}_\ell$ using an approach inspired by DGCNN [215] and deformable self-attention [253]. For notational convenience, we drop the ℓ subscript.

Starting from each query $\boldsymbol{q}_i$ (or, without the index dropped, $\boldsymbol{q}_{\ell i}$), we decode a reference point $\boldsymbol{p}_i \in \mathbb{R}^2$, a set of offsets $\{\boldsymbol{\delta}_{i0}, \ldots, \boldsymbol{\delta}_{iK}\} \subset \mathbb{R}^2$, and a set of attention

weights $\{w_{i0}, \ldots, w_{iK}\} \subset \mathbb{R}$:

$$\begin{aligned} \boldsymbol{p}_i &= \Phi_{\text{ref}}(\boldsymbol{q}_i), \qquad \{\boldsymbol{\delta}_i^0, \ldots, \boldsymbol{\delta}_i^k, \ldots \boldsymbol{\delta}_i^K\} = \Phi_{\text{neighbor}}(\boldsymbol{q}_i), \\ \{w_i^0, \ldots, w_i^k, \ldots w_i^K\} &= \Phi_{\text{atten}}(\boldsymbol{q}_i), \end{aligned} \tag{4.3}$$

where Φ_{ref}, Φ_{neighbor}, and Φ_{atten} are shared neural networks among the queries. We think of $\boldsymbol{p}_i$ as a hypothesis for the center of the i-th object; the $\boldsymbol{\delta}$'s represent the positions of K informative points relative to the position of the object that determine its geometry.

Next, we collect a BEV feature $\boldsymbol{f}_{ik}$ associated to each neighbor point $\boldsymbol{p}_{ik} = \boldsymbol{p}_i + \boldsymbol{\delta}_{ik}$:

$$\boldsymbol{f}_{ik} = f_{\text{bilinear}}(\mathcal{F}^d, \boldsymbol{p}_i + \boldsymbol{\delta}_{ik}), \tag{4.4}$$

where f_{bilinear} bilinearly interpolates the BEV feature map $\mathcal{F}^d$. Note this step is the interaction between our set-based architecture manipulating query points $\boldsymbol{q}_i$ and the grid-based feature map $\mathcal{F}^d$. We then aggregate a single object query feature $\boldsymbol{f}_i^o$ from the $\boldsymbol{f}_{ik}$s:

$$\boldsymbol{f}_i^o = \sum_k \frac{e^{w_{ik}}}{\sum_k e^{w_{ik}}} \boldsymbol{f}_{ik}. \tag{4.5}$$

This generates scene-aware features; each object query "attends" to a certain area in the scene.

In the current layer ℓ, the queries have not yet interacted with each other. To incorporate neighborhood information in object detection estimates, we use DGCNN-style operations to model a sparse set of relations. We construct a graph between the queries using a nearest neighbor search in feature space. In particular, we connect each query feature $\boldsymbol{f}_i^o$ to its 16 nearest neighbors as ablated in Table 4.7. Identically to DGCNN, we learn a feature per edge e_{ij} and then aggregate back to the vertices i to produce the new set of object queries. In detail, we write:

$$\boldsymbol{q}_{(\ell+1)i} = \max_{\text{edges } e_{ij}} \Phi_{\text{edge}}(\boldsymbol{f}_i^o, \boldsymbol{f}_j^o), \tag{4.6}$$

where max denotes a channel-wise maximum and Φ_{edge} is a neural network for computing edge features. This completes our layer for computing $\mathbb{Q}_{\ell+1}$ from $\mathbb{Q}_\ell$. Optionally, we repeat this last step multiple times, in effect applying DGCNN to the features $\boldsymbol{f}_i^o$ to get the point set $\mathbb{Q}_{\ell+1}$.

Set-to-set loss. After L Object DGCNN layers as described above, we are left with a set of M^* queries $\mathbb{Q}_L$ used to predict our bounding boxes. For each query $\boldsymbol{q}_{Li}$, we use a classification network to predict a categorical label $\hat{c}_i$ and a regression network to predict bounding box parameters $\hat{\boldsymbol{b}}_i$. Our final task is to assign the predictions to the ground-truth boxes and compute a set-to-set loss.

Most object detection models minimize a loss $\mathcal{L}_{\text{od}}$ given by

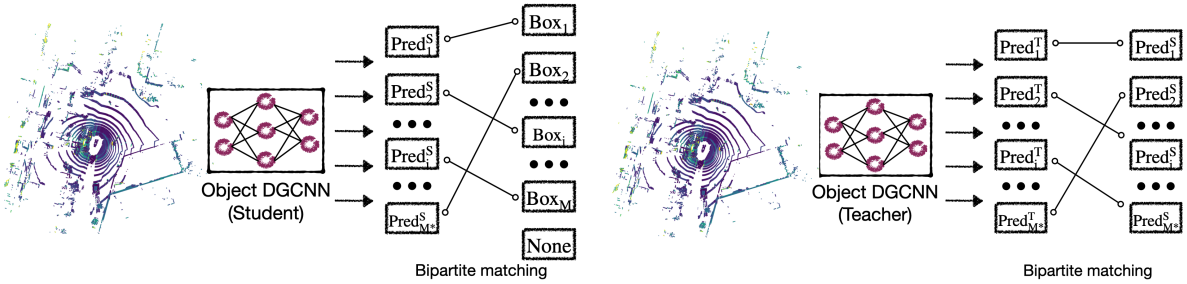
$$\mathcal{L}_{\text{od}} = \sum_{j=1}^{\hat{M}} -\log \hat{p}_{\hat{\sigma}(j)}(\hat{c}_j) + 1_{\{c_{\hat{\sigma}(j)} \neq \varnothing\}} \mathcal{L}_{\text{box}}(\hat{\boldsymbol{b}}_j, \boldsymbol{b}_{\hat{\sigma}(j)}), \tag{4.7}$$

where $\hat{M} = H^{\text{d}} * W^{\text{d}}$, $\hat{\sigma}(*)$ returns the corresponding index of the ground-truth bounding box, $\hat{p}_{\hat{\sigma}(j)}(c_j)$ is the probability of class $c_{\hat{\sigma}(j)}$ for the prediction with index $\sigma(j)$, $\varnothing$ denotes an invalid box, and $\mathcal{L}_{\text{box}}$ is typically the $\mathcal{L}_1$ distance. Different matchings $\hat{\sigma}$ yield different optimization landscapes and hence different prediction models. Pillar-OD [209] and CenterPoint [233] employ a simple $\hat{\sigma}$ to determine the ground-truth box used to evaluate the box predicted at BEV pixel j:

$$\hat{\sigma}_{\text{overlap}}(j) = \begin{cases} j', & \text{if } \boldsymbol{b}_{j'} \text{ overlaps with BEV pixel } j; \\ \varnothing, & \text{otherwise.} \end{cases} \tag{4.8}$$

This strategy can assign a box to multiple nearby BEV pixels. This one-to-many assignment provides dense supervision for the object detector and eases optimization. Since the training objective encourages each BEV pixel to predict the same surrounding box, however, redundant boxes are inevitable. So, NMS is usually required to remove redundant boxes at inference time.

Rather than performing dense predictions in the BEV, we make per-query predictions. Typically, M^* is much larger than the number of ground-truth boxes M. To



(a) Student network (b) Teacher network (fixed)

Figure 4-2: The set-to-set distillation pipeline. The student network is trained with the ground-truth supervision as well as with the supervision from a fixed teacher network.

account for this difference, we pad the set of ground-truth boxes with $\varnothing$s (no object) up to M^*. Following [25], we use an objective built on an optimal matching between these two sets. We define the optimal bipartite matching as

$$\sigma^* = \arg\min_{\sigma \in \mathcal{P}} \sum_{j=1}^{M} -1_{\{c_j \neq \varnothing\}} \hat{p}_{\sigma(j)}(c_j) + 1_{\{c_j = \varnothing\}} \mathcal{L}_{\text{box}}(\boldsymbol{b}_j, \hat{\boldsymbol{b}}_{\sigma(j)}), \tag{4.9}$$

where $\mathcal{P}$ denotes the set of permutations, $\hat{p}_{\sigma(j)}(c_j)$ is the probability of class c_j for the prediction with index $\sigma(j)$, and $\mathcal{L}_{\text{box}}$ is the $\mathcal{L}_1$ loss for bounding box parameters. We use the Hungarian algorithm [86] to solve this assignment problem, as in [176, 25]. Our final set-to-set loss adapts (4.7):

$$\mathcal{L}_{\text{sup}} = \sum_{j=1}^{N} -\log \hat{p}_{\sigma^*(j)}(c_j) + 1_{\{c_j \neq \varnothing\}} \mathcal{L}_{\text{box}}(\boldsymbol{b}_j, \hat{\boldsymbol{b}}_{\sigma^*(j)}). \tag{4.10}$$

4.6 Distillation

Object DGCNN enables a new set-to-set knowledge distillation (KD) pipeline. KD usually involves a teacher model $\mathcal{T}$ and a student model $\mathcal{S}$. The common practice is to align the outputs of the student with those of the teacher using $\mathcal{L}_2$ distance or KL divergence. In past 3D object detection methods, since final performance heavily relies on NMS and the predictions are post-processed to be a smaller set, distilling the teacher to the student is neither efficient nor effective. Since our set-

based detection model is NMS-free, we can easily distill the information between models with homogeneous detection heads (per-query object detection head in our case). First, we train a teacher $\mathcal{T}$ using the method above with the loss in (4.10). Then, we train a student $\mathcal{S}$ with supervision given by $\mathcal{T}$ and the ground-truth. The class label and box parameters predicted by the teacher for each object query are $c_j^{\mathcal{T}}$ and $\boldsymbol{b}_j^{\mathcal{T}}$, respectively. The corresponding student outputs are $c_j^{\mathcal{S}}$ and $\boldsymbol{b}_j^{\mathcal{S}}$. We find an optimal matching between the output set of the teacher and that of the student:

$$\sigma_d^* = \underset{\sigma_d \in \mathcal{P}}{\arg\min} \sum_j^N - \log p_{\sigma_d(j)}(c_j^{\mathcal{T}}) + \mathcal{L}_{\text{box}}(\boldsymbol{b}_j^{\mathcal{T}}, \boldsymbol{b}_{\sigma_d(j)}^{\mathcal{S}}). \tag{4.11}$$

Then, the optimal matching's KD loss is given by

$$\mathcal{L}_{\text{distill}} = \sum_j^N - \log \hat{p}_{\sigma_d^*(j)}(c_j^{\mathcal{T}}) + \mathcal{L}_{\text{box}}(\boldsymbol{b}_j^{\mathcal{T}}, \boldsymbol{b}_{\sigma_d^*(j)}^{\mathcal{S}}). \tag{4.12}$$

So the overall loss during KD is $\mathcal{L} = \alpha\mathcal{L}_{\text{sup}} + \beta\mathcal{L}_{\text{distill}}$, where α and β balance the supervised loss and distillation loss. In practice, we use $\alpha = \beta = 1$.

4.7 Experiments

We present our experiments in four parts. We introduce the dataset, metrics, implementation, and optimization details in §4.7.1. Then, we demonstrate performance on the nuScenes dataset [23] in §4.7.2. We present knowledge distillation results in §4.7.3. Finally, we provide ablation studies in §4.7.4.

4.7.1 Training & testing procedures

Dataset. We experiment on the nuScenes dataset [23]. nuScenes provides rich annotations and diverse scenes. It has 1K short sequences captured in Boston and Singapore with 700, 150, and 150 sequences for training, validation, and testing, respectively. Each sequence is $\sim$20s and contains 400 frames. This dataset provides annotation every 0.5s, leading to 28K, 6K, 6K annotated frames for training, valida-

Table 4.1: Comparisons to recent works. Our method is robust to whether to use NMS. ∗: implementations with the same PointPillars backbone. ‡: implementations with the same SparseConv backbone.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	NMS
PointPillars [88]	53.3	40.0	-	-	-	-	-	✓
SSN [252]	54.83	41.56	-	-	-	-	-	✓
FreeAnchor [241]	55.3	43.7	-	-	-	-	-	✓
RegNetX-400MF-SECFPN [145]	55.2	41.2	-	-	-	-	-	✓
Pillar-OD [209]	56.84	44.41	-	-	-	-	-	✓
CenterPoint (pillar) [233] ∗	59.56	47.48	**31.27**	**25.81**	33.78	32.25	20.20	✓
CenterPoint (pillar) [233] ∗	55.08	40.27	35.14	26.44	36.75	32.66	19.55	
CenterPoint (voxel) [233] ‡	**64.19**	**54.99**	**29.83**	**25.71**	32.56	26.08	**18.89**	✓
CenterPoint (voxel) [233] ‡	57.00	45.32	31.66	27.14	40.47	37.23	20.14	
Ours (pillar) ∗	**62.97**	**53.31**	34.62	26.56	**31.61**	**26.02**	**18.71**	✓
Ours (pillar) ∗	62.80	53.20	34.62	26.56	31.62	26.07	19.10	
Ours (voxel) ‡	**66.10**	58.73	33.31	**26.32**	**28.80**	**25.11**	19.08	✓
Ours (voxel) ‡	**66.04**	58.62	33.33	26.34	28.80	**25.11**	**19.06**	

Table 4.2: Comparisons of different distillation approaches.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
Baseline (without distillation)	62.80	53.20	34.62	26.56	31.62	26.02	19.10
Feature distill (voxel→pillar)	62.84	53.29	34.55	26.78	31.61	26.11	19.02
Pseudo labels (voxel→pillar)	63.18	53.31	34.58	26.55	31.29	25.99	18.87
Set-to-set distill (voxel→pillar)	63.37	53.89	34.34	26.25	31.01	25.57	18.77

tion, and testing. nuScenes uses 32-beam LiDAR, producing 30K points per frame. Following common practice, we use calibrated vehicle pose information to aggregate every 9 non-key frames to key frames, so each annotated frame has $\sim$300K points. The annotations include 23 classes with a long-tail distribution, of which 10 classes are included in the benchmark.

Metrics. The major metrics are mean average precision (mAP) and the nuScenes detection score (NDS). In addition, we use a set of true positive metrics (TP metrics), which include average translation error (ATE), average scale error (ASE), average orientation error (AOE), average velocity error (AVE), and average attribute error (AAE). These metrics are computed in the physical unit.

Model architecture. Our model consists of three parts: a point-based feature extractor, a DGCNN to encode object queries and to connect the point cloud features to object queries, and a detection head to output the categorical label and bounding

Table 4.3: Comparisons of self-distillation versus baselines.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
Baseline (pillar, without distillation)	62.80	53.20	34.62	26.56	31.62	26.02	19.10
Self-distillation (pillar→pillar)	63.41	53.89	34.21	26.19	31.11	25.67	18.54
Baseline (voxel, without distillation)	66.04	58.62	33.33	26.34	28.80	25.11	19.06
Self-distillation (voxel→voxel)	66.45	59.25	31.17	25.77	30.73	25.72	18.77

Table 4.4: Self-distillation with privileged information.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
Sparse→sparse (pillar)	42.12	38.89	44.01	27.78	64.01	144.01	39.21
Dense→sparse (pillar)	42.79	39.10	43.89	27.77	64.01	143.97	39.11
Sparse→sparse (voxel)	59.55	49.84	31.17	25.77	33.73	32.22	20.22
Dense→sparse (voxel)	59.89	50.12	31.11	25.76	33.70	32.19	20.11

box parameters. We experiment with PointPillars [88] and SparseConv [55] as feature extractors. The three blocks of the PointPillars backbone have $[3, 5, 5]$ convolutional layers, with dimensions $[64, 128, 256]$ and strides $[2, 2, 2]$; the input features are downsampled to 1/2, 1/4, 1/8 of the original feature map. For SparseConv, we use four blocks of $[3, 3, 3, 2]$ 3D sparse convolutional layers, with dimensions $[16, 32, 64, 128]$ and strides $[2, 2, 2, 1]$; the input features are downsampled to 1/2, 1/4, 1/8, 1/8 of the original feature map. For SparseConv, we transform the features into BEV by collapsing the z-axis. Both backbones use two deformable self-attention [253] layers with dimensions $[256, 256]$ to transform the BEV features. Then, we use two DGCNNs to encode the object queries. Each DGCNN [215] contains two EdgeConv layers with dimensions $[256, 256]$, both with 16 nearest neighbors. For each object query, we predict four points in the BEV to obtain and aggregate the BEV features. The final feature for this object query is the weighted sum of features of these four BEV points. The final detection head takes the features of each object query and predicts class label and bounding box parameters w.r.t. the reference point.

Training & inference. We use AdamW [110] to train the model. The weight decay for AdamW is 10^{-2}. Following a cyclic schedule [170], the learning rate is initially 10^{-4} and gradually increased to 10^{-3}, which is finally decreased to 10^{-8}. The

model is initialized with a pre-trained PointPillars network on the same dataset. We train for 20 epochs on 8 RTX 3090 GPUs. During inference, we take the top 100 objects with highest classification scores as the final predictions. We *do not* use any post-processing such as NMS. For evaluation, we use the toolkit provided with the nuScenes dataset.

4.7.2 Object DGCNN

We compare to top-performing methods on the nuScenes dataset in Table 4.1. PointPillars [88] is an anchor-based method with reasonable trade-off between performance and efficiency. FreeAnchor [241] extends PointPillars by learning how to assign anchors to the ground-truth. RegNetX-400MF-SECFPN [145] uses neural architecture search (NAS) to learn a flexible neural network for 3D detection; it is essentially a variant of PointPillars with an enhanced backbone network. Different from anchor-based methods, Pillar-OD [209] makes predictions per pillar, alleviating the class imbalance issue caused by anchors. CenterPoint [233] exploits similar detection heads, with better performance using better training scheduling and data augmentation. For these methods, we use re-implementations in MMDetection3D [34], which match the performances in the original papers.

We mainly compare to CenterPoint with both PointPillars and SparseConv backbones, denoted as "voxel" and "pillar" respectively. Our method outperforms other methods significantly including CenterPoint with NMS. Without NMS, the performance of CenterPoint drops considerably while our method is unaffected by NMS. This finding verifies the DGCNN implicitly models object relations and removes redundant boxes.

4.7.3 Set-to-set distillation

In this section, we present experiments involving our set-to-set distillation pipeline. We conduct three types of distillation. First, we distill a teacher model with a SparseConv backbone to a student model with a PointPillars backbone (denoted

as voxel→pillar). This aligns with the common knowledge distillation setup for classification. We compare to feature-based distillation and pseudo label based methods. The objective of feature-based distillation is to align the middle-level features of the teacher model and the student model while the pseudo label based methods generate pseudo training examples with the pre-trained teacher networks. As Table 4.2 shows, our set-to-set distillation achieves better performance, confirming that distilling the last stage of the object detection model is more effective than distilling feature maps.

Second, we perform self-distillation [47] (denoted as voxel→voxel and pillar→pillar), where the teacher and the student are identical and take the same point clouds as input. As Table 4.3 shows, even when the teacher network and the student network have the same capacity, self-distillation still introduces a performance boost. This finding is consistent with the results in [47].

Finally, we try distillation with privileged information [108], where the teacher gets access to privileged information but the student does not. Following [210], the teacher takes dense point clouds, and the student takes sparse point clouds (denoted as "dense→sparse"). To limit computation time, we train each model over a shorter period of time. The goal is for the student model to learn the same representations as the teacher model without knowing the dense inputs. In Table 4.4, we compare this setup with self-distillation, where the difference is the teacher model and the student model take the same sparse point clouds in self-distillation. The student achieves better performance when the teacher takes dense point clouds. The result suggests that set-to-set knowledge distillation is an effective approach to transfer insight from privileged information.

4.7.4 Ablation

We provide ablation studies on different components of our model to verify assorted design choices. First, we study the improvements of DGCNN over its counterpart, multi-head self-attention [200]. The multi-head self-attention has 8 heads with embedding dimension 256 and LayerNorm [7], following common usage. The DGCNN has two EdgeConv layers with dimensions $[256, 256]$. The number of neighbors K in

Table 4.5: DGCNN versus multi-head self-attention.

Metric \ Method	Multi-head self-attention	DGCNN
NDS	39.89	**41.32**
mAP	36.35	**37.81**

Table 4.6: Models with different # DGCNNs.

Metric \ # layers	1	2	3	4	5	6
NDS	35.91	39.75	41.15	41.26	41.07	**41.32**
mAP	32.32	36.54	37.25	37.75	37.78	**37.81**

EdgeConv is 16. In principle, DGCNN is a sparse version of multi-head self-attention; the sparse structure reduces overhead in back-propagation and leads to sharper "attention maps" as well as faster convergence.

Table 4.5 shows the comparisons: DGCNN consistently outperforms multi-head self-attention. This aligns with our hypothesis: objects are distributed sparsely in the scene, so dense interactions among objects are neither efficient nor effective. Furthermore, we study the effect of number of neighbors in DGCNN. When it is 1, the model reduces to an architecture without object interactions. As we increase the number, it approaches multi-head self-attention. As shown in Table 4.7, the sweet spot is 16, which appears to balance object interactions and sparsity.

We also investigate improvements introduced when more DGCNNs are stacked in Table 4.6. This result suggests it is beneficial to incorporate multiple DGCNNs to model the dynamic object relations.

Moreover, we hypothesize our method produces different distribution of the output scores with respect to overlapping boxes. To verify this hypothesis (Table 4.8, we compute average scores for three types of boxes: filtered boxes after NMS, remaining boxes after NMS, and all boxes. Below we show the results. We also compute the percentage of filtered boxes by NMS in our method and Centerpoint. In our method, 21.9% boxes are removed while in Centerpoint 85.16% boxes are filtered. Hence,

Table 4.7: The number of neighbors in DGCNN.

Metric \ # neigbhors	1	4	8	16	32	64
NDS	40.21	40.45	40.51	**41.32**	40.17	39.80
mAP	36.81	37.15	37.46	**37.81**	37.12	36.70

Table 4.8: The distribution of the output scores with respect to overlapping boxes.

Method	filtered boxes	remaining boxes	all boxes
CenterPoint	0.0764	0.1859	0.0829
Ours	0.1222	0.1711	0.1604

we conclude that our method indeed exhibits a different distribution pattern from Centerpoint.

Finally, we include a complexity comparison between DGCNN and Multi-head self-attention. Table 4.9 shows the results; DGCNN layer is on a par with Multi-head self-attention.

4.8 Conclusion

Object DGCNN is a highly-efficient 3D object detector for point clouds. It is able to learn object interactions via dynamic graphs and is optimized through a set-to-set loss, leading to NMS-free detection. The success of Object DGCNN indicates that many post-processing operations in 3D object detection are likely unnecessary and can be replaced with suitable neural network modules. Moreover, we introduce a set-to-set knowledge distillation pipeline enabled by the Object DGCNN. This new pipeline significantly simplifies knowledge distillation for 3D object detection and may be applicable to other tasks like 3D model compression. Beyond the direct usage of our model, our experiments suggest several future directions to address current limitations. For example, our method is initialized with a pre-trained backbone network. Training the model from scratch remains elusive due to the sparse set-to-set supervision; solving this issue may yield improved generalization as in [60]. Furthermore,

Table 4.9: Complexity comparsion between DGCNN and Multi-head self-attention.

Module	Complexity	# parameters
DGCNN	$O(n^2d)$	262144
Multi-head self-attention	$O(n^2d)$	263168

studying 3D-specific feature extractors will improve the speed and generalizability of 3D object detection. Finally, the large amount of unlabeled data available at training time can serve as another type of privileged information to apply self-supervised learning to 3D domains through set-to-set distillation.

Potential impact. Our method aims to improve the object detection pipeline, which is crucial for the safety of autonomous driving systems. One potential negative impact of our work is that it still lacks theoretical guarantees, similar to many deep learning methods. Future work to improve applicability in this domain might consider challenges of *explainability* and *transparency*.

Chapter 5

DETR3D

3D object detection from visual information is a long-standing challenge for low-cost autonomous driving systems. While object detection from point clouds collected using modalities like LiDAR benefits from information about the 3D structure of visible objects, the camera-based setting is even more ill-posed, since we must generate 3D bounding box predictions solely from the 2D information contained in RGB images.

Existing methods [246, 203] typically build their detection pipelines purely from 2D computations. That is, they predict 3D information like object pose and velocity using an object detection pipeline designed for 2D tasks (e.g., CenterNet [246], FCOS [193]), without considering 3D scene structure or sensor configuration. These methods require several post-processing steps to fuse predictions across cameras and to remove redundant boxes, yielding a steep trade-off between efficiency and effectiveness. As an alternative to these 2D-based methods, some methods incorporate more 3D computations into our object detection pipeline by applying a 3D reconstruction method like [50, 91, 56] to create a pseudo-LiDAR or range input of the scene from camera images. Then, they could apply 3D object detection methods to this data as if it were collected directly from a 3D sensor. This strategy, however, is subject to compounding errors [167]: poorly-estimated depth values have a strongly negative effect on the performance of 3D object detection, which also can exhibit errors of its own.

5.1 Introduction

In this chapter, we propose a more graceful transition between 2D observations and 3D predictions for autonomous driving, which does not rely on a module for dense depth prediction. Our framework, termed Multi-View 3D Detection (DETR3D), addresses this problem in a top-down fashion. We link 2D feature extraction and 3D object prediction via geometric back-projection with camera transformation matrices. Our method starts from a sparse set of object priors, shared across the dataset and learned end-to-end. To gather scene-specific information, we back-project a set of reference points decoded from these object priors to each camera and fetch the corresponding image features extracted by a ResNet backbone [62]. The features collected from the image features of the reference points then interact with each other through a multi-head self-attention layer [200]. After a series of self-attention layers, we read off bounding box parameters from every layer and use a set-to-set loss inspired by DETR [25] to evaluate performance.

Our architecture does not perform point cloud reconstruction or explicit depth prediction from images, making it robust to errors in depth estimation. Moreover, our method does not require any post-processing, such as non-maximum suppression (NMS), improving efficiency and reducing reliance on hand-designed methods for cleaning its output. On the nuScenes dataset, our method (without NMS) is comparable with prior art (with NMS). In the camera overlap regions, our method significantly outperforms others.

Contributions. We summarize our key contributions as follows:

- We present a streamlined 3D object detection model from RGB images. Different from existing works that combine object predictions from the different camera views in a final stage, our method fuses information from all the camera views in each layer of computation. To the best of our knowledge, this is the first attempt to cast multi-camera detection as 3D set-to-set prediction.

- We introduce a module that connects 2D feature extraction and 3D bounding box prediction via backward geometric projection. It does not suffer from

inaccurate depth predictions from a secondary network, and seamlessly uses information from multiple cameras by back-projecting 3D information onto all available frames.

- Similarly to Object DGCNN [214], our method does not require post-processing such as per-image or global NMS, and it is on par with existing NMS-based methods. In the camera overlap regions, our method outperforms others by a substantial margin.

5.2 Related Work

2D object detection. RCNN [49] pioneered object detection using deep learning. It feeds a set of pre-selected object proposals into a convolutional neural network (CNN) and predicts bounding box parameters accordingly. Although this method exhibits surprising performance, it is an order of magnitude slower than others because it performs a ConvNet forward pass for each object proposal. To fix this issue, Fast RCNN [148] introduces a shared learnable CNN to process the entire image at a single forward pass. To further improve performance and speed, Faster RCNN [148] includes a region proposal network (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. Mask RCNN [61] incorporates a mask prediction branch to enable instance segmentation in parallel. These methods typically involve multi-stage refinements and can be slow in practice. Different from these multi-stage methods, SSD [102] and YOLO [146] perform dense predictions in a single shot. Although they are significantly faster than the alternatives above, they still rely on NMS to remove redundant box predictions. These methods predict bounding boxes w.r.t. pre-defined anchors. CenterNet [246] and FCOS [193] change the paradigm by shifting from per-anchor prediction to per-pixel prediction, significantly simplifying the common object detection pipeline.

Set-based object detection. DETR [25] casts object detection as a set-to-set problem. It employs a Transformer [200] to capture feature and object interactions. DETR learns to assign predictions to a set of ground-truth boxes; thus, it

does not require post-processing to filter out redundant boxes. One critical drawback of DETR, however, is that it requires a significant amount of training time. Deformable DETR [253] analyzes DETR's slow convergence and proposes a deformable self-attention module to localize features and accelerate training. Concurrently, [184] attributes the slow convergence of DETR to the set-based loss and the Transformer cross attention mechanism. They propose two variants, TSP-FCOS and TSP-RCNN, to overcome these problematic aspects. SparseRCNN [183] incorporates set prediction into a RCNN-style pipeline; it outperforms multi-stage object detection without NMS. OneNet [182] studies an interesting phenomenon: dense-based object detectors can be made NMS-free after they are equipped with a minimum-cost set loss. For 3D domains, Object DGCNN [214] studies 3D object detection from point clouds. It models 3D object detection as message passing on a dynamic graph, generalizing the DGCNN framework to predict a set of objects. Similar to DETR, Object DGCNN is also NMS-free.

Monocular 3D object detection. An early method for 3D detection from RGB images is Mono3D [30], which uses semantic and shape cues to select from a collection of 3D proposals, using scene constraints and additional priors at training time. [149] uses the birds-eye-view (BEV) for monocular 3D detection, and [124] leverages 2D detections for 3D bounding box regression via the minimization of 2D-3D projection error. The use of 2D detectors as a starting point for 3D computation recently has become a standard approach [78, 85]. Other works also explore advances in differentiable rendering [10] or 3D keypoint detection [9, 106, 246] to enable state-of-the-art 3D object detection performance. All these methods operate in a monocular setting, and extensions to multiple cameras are done by independently processing each frame before merging the outputs in a post-processing stage.

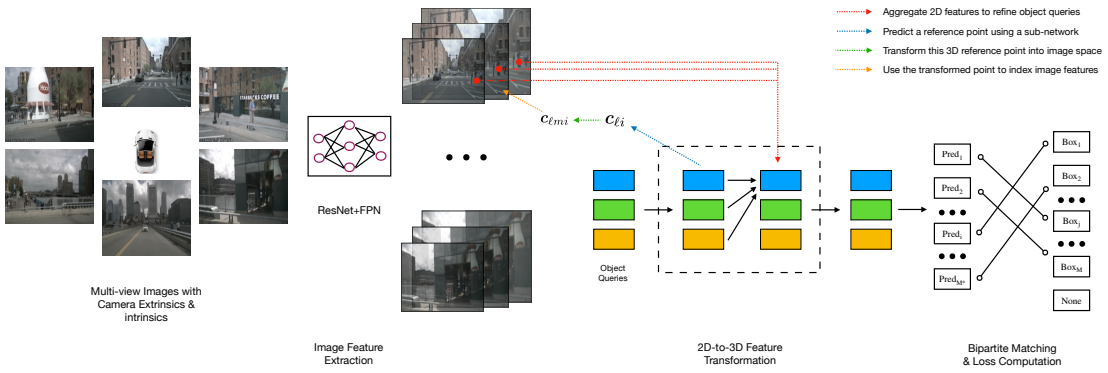


Figure 5-1: Overview of our method. The inputs to the model are a set of multi-view images, which are encoded by a ResNet and a FPN. Then, our model operates on a set of sparse object queries in which each query is decoded to a 3D reference point. 2D features are transformed to refine the object queries by projecting the 3D reference point into the image space. Our model makes per-query predictions and uses a set-to-set loss.

5.3 Multi-view 3D Object Detection

5.3.1 Overview

Our architecture inputs RGB images collected from a set of cameras whose projection matrices (the combination of intrinsics and relative extrinsics) are known, and it outputs a set of 3D bounding box parameters for the objects in the scene. In contrast to past approaches, we build our architecture based on a few high-level desiderata:

- We incorporate 3D information into intermediate computations within our architecture, rather than performing purely 2D computations in the image plane.
- We do not estimate dense 3D scene geometry, avoiding associated reconstruction errors.
- We avoid post-processing steps such as NMS.

We address these desiderata using a new set prediction module, which links 2D feature extraction and 3D box prediction by alternating between 2D and 3D computations. Our model contains three critical components, illustrated in Figure 5-1. First, following common practice in 2D vision, it extracts features from the camera images

using a shared ResNet [62] backbone. Optionally, these features are enhanced by a feature pyramid network (FPN) [97] (§5.3.2). Second, a detection head (§5.3.3)—our main contribution—links the computed 2D features to a set of 3D bounding box predictions in a geometry-aware manner (§5.3.3). Each layer of the detection head starts from a sparse set of object queries, which are learned from the data. Each object query encodes a 3D location, which is projected to the camera planes and used to collect image features via bilinear interpolation. Similarly to DETR [25], we then use multi-head attention [200] to refine the object queries by incorporating object interactions. This layer is repeated multiple times, alternating between feature sampling and object query refinement. Finally, we evaluate a set-to-set loss [176, 25] to train the network (§5.3.4).

5.3.2 Feature Learning

Our model starts with a set of images $\mathcal{I} = \{\mathbf{im}_1, \ldots, \mathbf{im}_K\} \subset \mathbb{R}^{\mathrm{H_{im}} \times \mathrm{W_{im}} \times 3}$ (captured by surrounding cameras), camera matrices $\mathcal{T} = \{T_1, \ldots, T_K\} \subset \mathbb{R}^{3\times4}$, ground-truth bounding boxes $\mathcal{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_j, \ldots, \boldsymbol{b}_M\} \subset \mathbb{R}^9$, and categorical labels $\mathcal{C} = \{c_1, \ldots, c_j, \ldots, c_M\} \subset \mathbb{Z}$. Each $\boldsymbol{b}_j$ contains position, size, heading angle, and velocity in the birds-eye view (BEV); our model aims to predict these boxes and their labels from the these images. We *do not* use point clouds, which are usually captured by high-end LiDAR.

These images are encoded with a ResNet [62] and a FPN [97] into four sets of features $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4$. Each set $\mathcal{F}_k = \{\boldsymbol{f}_{k1}, \ldots, \boldsymbol{f}_{k6}\} \subset \mathbb{R}^{H\times W\times C}$ corresponds to a level of features of the 6 images. These multi-scale features provide rich information to recognize objects of different sizes. Next, we detail our approach to transform these 2D features into 3D using a novel set prediction module.

5.3.3 Detection Head

Existing methods for detecting objects from camera input typically employ a bottom-up approach, which predicts a dense set of bounding boxes per image, filters re-

dundant boxes between the images, and aggregates predictions across cameras in a post-processing step. This paradigm has two crucial drawbacks: dense bounding box prediction requires accurate depth perception, which itself is a challenging problem; and NMS-based redundancy removal and aggregation are non-parallelizable operations that introduce significant inference overhead. We address these issues using a top-down object detection head described below.

Analogously to [214, 253], DETR3D is *iterative*; it uses L layers with set-based computations to produce bounding box estimates from 2D feature maps. Each layer includes the following steps:

- predict a set of bounding box centers associated with object queries;
- project these centers into all the feature maps using the camera transformation matrices;
- sample features via bilinear interpolation and incorporate them into object queries; and
- describe object interactions using multi-head attention.

Motivated by DETR [25], each layer $\ell \in \{0, \dots, L-1\}$ operates on a set of *object queries* $\mathbb{Q}_\ell = \{\boldsymbol{q}_{\ell 1}, \dots, \boldsymbol{q}_{\ell M^*}\} \subset \mathbb{R}^C$, producing a new set $\mathbb{Q}_{\ell+1}$. A reference point $\boldsymbol{c}_{\ell i} \in \mathbb{R}^3$ is decoded from a object query $\boldsymbol{q}_{\ell i}$ as follows:

$$\boldsymbol{c}_{\ell i} = \Phi^{\text{ref}}(\boldsymbol{q}_{\ell i}), \tag{5.1}$$

where Φ^{ref} is a neural network. $\boldsymbol{c}_{\ell i}$ can be thought of a hypothesis for the center of the i-th box. Next, we acquire image features corresponding to $\boldsymbol{c}_{\ell i}$ to refine and predict the final bounding box. Then, $\boldsymbol{c}_{\ell i}$ (or more accurately its homogeneous counterpart $\boldsymbol{c}^*_{\ell i}$) is projected into each one of the images using the camera transformation matrices:

$$\boldsymbol{c}^*_{\ell i} = \boldsymbol{c}_{\ell i} \oplus 1 \qquad\qquad \boldsymbol{c}_{\ell m i} = T_m \boldsymbol{c}^*_{\ell i}, \tag{5.2}$$

where $\oplus$ denotes concatenation, and $\boldsymbol{c}_{\ell mi}$ is the projection of the reference point onto the m-th camera. To remove the effects of the feature map size and gather features across different levels, we normalize $\boldsymbol{c}_{\ell mi}$ to $[-1, 1]$. Next, the images features are collected by

$$\boldsymbol{f}_{\ell kmi} = f^{\text{bilinear}}(\mathcal{F}_{km}, \boldsymbol{c}_{\ell mi}), \tag{5.3}$$

where $\boldsymbol{f}_{\ell kmi}$ is the feature for i-th point from k-th level of m-th camera at ℓ-th layer.

A given reference point is not necessarily visible in all the camera images, so we need some heuristics to filter invalid points. To that end, we define a binary value $\sigma_{\ell kmi}$, which is determined based on whether a reference point is projected outside an image plane. The final feature $\boldsymbol{f}_{\ell i}$ and object query in next layer $\boldsymbol{q}_{(\ell+1)i}$ are given by

$$\boldsymbol{f}_{\ell i} = \frac{1}{\sum_k \sum_m \sigma_{\ell kmi} + \epsilon} \sum_k \sum_m \boldsymbol{f}_{\ell kmi} \sigma_{\ell kmi} \qquad \text{and} \qquad \boldsymbol{q}_{(\ell+1)i} = \boldsymbol{f}_{\ell i} + \boldsymbol{q}_{\ell i}, \tag{5.4}$$

where ϵ is a small number to avoid division by zero. Finally, for each object query $\boldsymbol{q}_{\ell i}$, we predict a bounding box $\hat{\boldsymbol{b}}_{\ell i}$ and its categorical label $\hat{c}_{\ell i}$ with two neural networks Φ_ℓ^{reg} and Φ_ℓ^{cls}:

$$\hat{\boldsymbol{b}}_{\ell i} = \Phi_\ell^{\text{reg}}(\boldsymbol{q}_{\ell i}) \qquad \text{and} \qquad \hat{c}_{\ell i} = \Phi_\ell^{\text{cls}}(\boldsymbol{q}_{\ell i}). \tag{5.5}$$

We compute the loss for the predictions $\hat{\mathcal{B}}_\ell = \{\hat{\boldsymbol{b}}_{\ell 1}, \ldots, \hat{\boldsymbol{b}}_{\ell j}, \ldots, \hat{\boldsymbol{b}}_{\ell M^*}\} \subset \mathbb{R}^9$ and $\hat{\mathcal{C}}_\ell = \{\hat{c}_{\ell 1}, \ldots, \hat{c}_{\ell j}, \ldots, \hat{c}_{\ell M}\} \subset \mathbb{Z}$ from every layer during training. During inference, we only use the outputs from the last layer.

5.3.4 Loss

Following [176, 25], we use a set-to-set loss to measure the discrepancy between the prediction set $(\hat{\mathcal{B}}_\ell, \hat{\mathcal{C}}_\ell)$ and the ground-truth set $(\mathcal{B}, \mathcal{C})$. This loss consists of two parts: a focal loss [98] for the class labels and a L^1 loss for the bounding box parameters. For notational convenience, we drop the ℓ subscript in $\hat{\mathcal{B}}_\ell$ and $\hat{\mathcal{C}}_\ell$. The number of

ground-truth boxes M is typically smaller than the number of predictions M^*, so we pad the set of ground-truth boxes with $\varnothing$s (no object) up to M^* for ease of computation. We establish a correspondence between the ground-truth and the prediction via a bipartite matching problem: $\sigma^* = \arg\min_{\sigma \in \mathcal{P}} \sum_{j=1}^{M} -1_{\{c_j \neq \varnothing\}} \hat{p}_{\sigma(j)}(c_j) + 1_{\{c_j \neq \varnothing\}} \mathcal{L}_{\text{box}}(\boldsymbol{b}_j, \hat{\boldsymbol{b}}_{\sigma(j)})$, where $\mathcal{P}$ denotes the set of permutations, $\hat{p}_{\sigma(j)}(c_j)$ is the probability of class c_j for the prediction with index $\sigma(j)$, and $\mathcal{L}_{\text{box}}$ is the L_1 loss for bounding box parameters. We use the Hungarian algorithm [86] to solve this assignment problem, as in [176, 25], yielding the set-to-set loss $\mathcal{L}_{\text{sup}} = \sum_{j=1}^{N} -\log \hat{p}_{\sigma^*(j)}(c_j) + 1_{\{c_j \neq \varnothing\}} \mathcal{L}_{\text{box}}(\boldsymbol{b}_j, \hat{\boldsymbol{b}}_{\sigma^*(j)})$.

5.4 Experiments

We present our results as follows: first, we detail the dataset, metrics, and implementation in §5.4.1; then we compare our method to existing works in §5.4.2; we benchmark the performance of different models in camera overlap regions in §5.4.3; we compare to a forward prediction model in §5.4.4; and we provide additional analysis and ablations in §5.4.5.

5.4.1 Implementation Details

Dataset. We test our method on the nuScenes dataset [23]. nuScenes consists of 1,000 sequences; each sequence is roughly 20s long, with a sampling rate of 20 frames/second. Each sample contains images from 6 cameras `[front_left, front, front_right, back_left, back, back_right]`. Camera parameters including intrinsics and extrinsics are available. nuScenes provides annotations every 0.5s; in total there are 28k, 6k, and 6k annotated samples for training, validation, and testing, respectively. 10 from the total 23 classes are available to compute the metrics.

Metrics. We follow the official evaluation protocol provided by nuScenes. We evaluate average translation error (ATE), average scale error (ASE), average orientation error (AOE), average velocity error (AVE), and average attribute error (AAE). These metrics are true positive metrics (TP metrics) and computed in the physi-

cal unit. In addition, we measure mean average precision (mAP). To capture all aspects of the detection task, a consolidated scalar metric–the nuScenes Detection Score (NDS) [23]–is defined as NDS $= \frac{1}{10}[5\text{mAP} + \sum_{\text{mTP}\in\mathbb{TP}}(1 - \min(1, \text{mTP}))]$.

Model. Our model consists of a ResNet [62] feature extractor, a FPN, and a DETR3D detection head. We use ResNet101 with deformable convolutions [35] in the 3rd stage and 4th stage. The FPN [97] takes features output by the ResNet and produces 4 feature maps whose sizes are $1/8$, $1/16$, $1/32$, and $1/64$ of the input image sizes. The DETR3D detection head consists of 6 layers, where each layer is a combination of a feature refinement step and a multi-head attention layer. The hidden dimension of the DETR3D detection head is 256. Finally, two sub-networks predict bounding box parameters and a class label per object query; each sub-network consists of two fully connected layers with hidden dimensions 256. We use LayerNorm [7] in the detection head.

Training & inference. We use AdamW [110] to train the whole pipeline. The weight decay is 10^{-4}. We use an initial learning rate 10^{-4}, which is decreased to 10^{-5} and 10^{-6} at 8th and 11th epochs. The model is trained for 24 epochs in total on 8 RTX 3090 GPUs and the per-GPU batch size is 1. The training procedure takes roughly 18 hours. We do not use any post-processing such as NMS during inference. For evaluation, we use the nuScenes evalutation toolkit.

5.4.2 Comparison to Existing Works

We compare to previous state-of-the-art methods CenterNet [246] and FCOS3D [203]. CenterNet is an anchor-free 2D detection method that makes dense predictions in a high resolution feature map. FCOS3D employs a FCOS [193] pipeline to make per-pixel predictions. These methods both turn 3D object detection into a 2D problem, and in doing so ignore scene geometry and sensor configuration. To perform multi-view object detection, these methods have to process each image independently, and use both per-image and global NMS to remove redundant boxes in each view and in the overlap regions respectively. As shown in Table 5.1, our method outperforms these methods even though we do not use any post-processing. Our method performs

Table 5.1: Comparisons to recent works on the validation set. Our method is robust to the usage of NMS. ∗: CenterNet uses a customized backbone DLA [234]. ‡: this model is trained with depth weight 1.0 and initialized from a FCOS3D checkpoint; the checkpoint is trained on the same dataset with depth weight 0.2. §: with test-time augmentation. ¶: with test-time augmentation, more epochs, and model ensemble. For details, see [203]. †: our model is also initialized from a FCOS3D backbone; the detection head is initialized randomly. #: trained with CBGS [249]

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	NMS
CenterNet ∗	0.328	0.306	0.716	0.264	0.609	1.426	0.658	✓
FCOS3D	0.373	0.299	0.785	0.268	0.557	1.396	0.154	✓
FCOS3D ‡	0.393	0.321	0.746	0.265	0.503	1.351	0.160	✓
FCOS3D §	0.402	0.326	0.743	0.259	0.441	1.341	0.163	✓
FCOS3D ¶	0.415	0.343	0.725	0.263	0.422	1.292	0.153	✓
DETR3D (Ours)	0.374	0.303	0.860	0.278	0.437	0.967	0.235	-
DETR3D (Ours) †	0.425	0.346	0.773	0.268	0.383	0.842	0.216	-
DETR3D (Ours) #	0.434	0.349	0.716	0.268	0.379	0.842	0.200	-

Table 5.2: Comparisons to top-performing works on the test set from the leaderboard. #: initialized from a DD3D checkpoint. †: initialized from a backbone pre-trained on extra data.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	NMS
Mono3D	0.429	0.366	0.642	0.252	0.523	1.591	0.119	N/A
DHNet	0.437	0.363	0.667	0.259	0.402	1.589	0.120	N/A
PGD [204]	0.448	0.386	0.626	0.245	0.451	1.509	0.127	✓
DD3D [132] †	0.477	0.418	0.572	0.249	0.368	1.014	0.124	✓
DETR3D (Ours) #	0.479	0.412	0.641	0.255	0.394	0.845	0.133	-

worse than FCOS3D in terms of mATE. We suspect this is because FCOS3D directly predicts bounding box depth, which leads to strong supervision on object translation. Also, FCOS3D uses disentangled heads for different bounding box parameters, which can increase performance.

On the test set (Table 5.2), our method outperforms all existing methods as of 10/13/2021; our method uses the same backbone as DD3D [132] for a fair comparison.

5.4.3 Comparison in Overlap Regions

A great challenge lies in the overlap regions where objects are more likely to be cut off. Our method considers all cameras simultaneously, while FCOS3D predicts

Table 5.3: Comparisons in Overlap Region. ‡: this model is trained with depth weight 1.0 and initialized from a FCOS3D checkpoint; the checkpoint is trained on the same dataset with depth weight 0.2. For details, see [203]. †: our model is also initialized from a FCOS3D backbone; the detection head is initialized randomly.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	NMS
FCOS3D	0.317	0.213	0.841	0.276	0.604	1.122	0.173	✓
FCOS3D‡	0.329	0.229	0.816	0.272	0.571	1.084	0.195	✓
DETR3D (Ours)	0.356	0.231	0.825	0.280	0.400	0.863	0.223	-
DETR3D (Ours) †	0.384	0.268	0.807	0.273	0.453	0.788	0.184	-

bounding boxes per camera individually. To further demonstrate the advantages of fused inference, we calculate the metrics for boxes falling into the camera overlaps. To compute the metrics, we select boxes whose 3D center is visible to multiple cameras. On the validation set, there are 18,147 such boxes, 9.7% of the total. Table 5.3 shows the results; our method outperforms FCOS3D remarkably in terms of NDS scores in this setting. This confirms that our integrated prediction approach is more effective.

5.4.4 Comparison to pseudo-LiDAR Methods

Another way to perform 3D object detection is by generating pseudo-LiDAR point clouds from multi-view images using a depth prediction model. On the nuScenes dataset, there are no publicly available pseudo-LiDAR works for us to make a direct comparison. Hence, we implement a baseline ourselves to verify that our approach is more effective than explicit depth prediction. We use a pre-trained PackNet [56] network to predict dense depth maps from all six cameras and then convert these depth maps into point clouds using the camera transformations. We also experimented with a self-supervised PackNet model with velocity supervision (as in the original paper), but we found that ground-truth depth supervision yielded more realistic point clouds and therefore used a supervised model as baseline. For 3D detection, we employ the recently-proposed CenterPoint architecture [233]. Conceptually, this pipeline is a variant of pseudo-LiDAR [207]. Table 5.4 shows the results; we conclude that this pseudo-LiDAR method underperforms ours significantly even when depth estimates

Table 5.4: Comparisons to pseudo-LiDAR Methods.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	NMS
pseudo-LiDAR	0.160	0.048	-	-	-	-	-	✓
DETR3D (Ours)	0.374	0.303	0.860	0.278	0.437	0.967	0.235	-

Table 5.5: Evaluation on detection results from different layers.

Layer ↑	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
0	0.380	0.302	0.855	0.280	0.435	0.910	0.231
1	0.410	0.335	0.791	0.275	0.408	0.887	0.217
2	0.420	0.343	0.782	0.271	0.395	0.851	0.214
3	0.420	0.346	0.778	0.268	0.390	0.874	0.218
4	0.424	0.346	0.777	0.268	0.389	0.855	0.217
5	0.425	0.346	0.773	0.268	0.383	0.842	0.216

are generated by a state-of-the-art model. One possible explanation is that pseudo-LiDAR object detectors suffer from compounding errors introduced by inaccurate depth prediction, that in turn is known to overfit to training data and generalizes poorly to other distributions [167].

5.4.5 Ablation & Analysis

We visualize object query refinement in Figure 5-2. We visualize bounding boxes decoded from the object queries in each layer. The predicted bounding boxes get closer to the ground-truth as we go into deeper layers in the model. Also, the leftmost figure shows the learned object query priors shared by all data. We also provide quantitative results in Table 5.5, which shows that iterative refinement indeed improves performance significantly. This suggests that iterative refinement is both beneficial and necessary to fully leverage our proposed architecture. Furthermore, we provide ablations on the number of object queries in Table 5.6; increasing the number queries consistently improves the performance until it gets saturated at 900. Finally, Table 5.7 shows the results with different backbones.

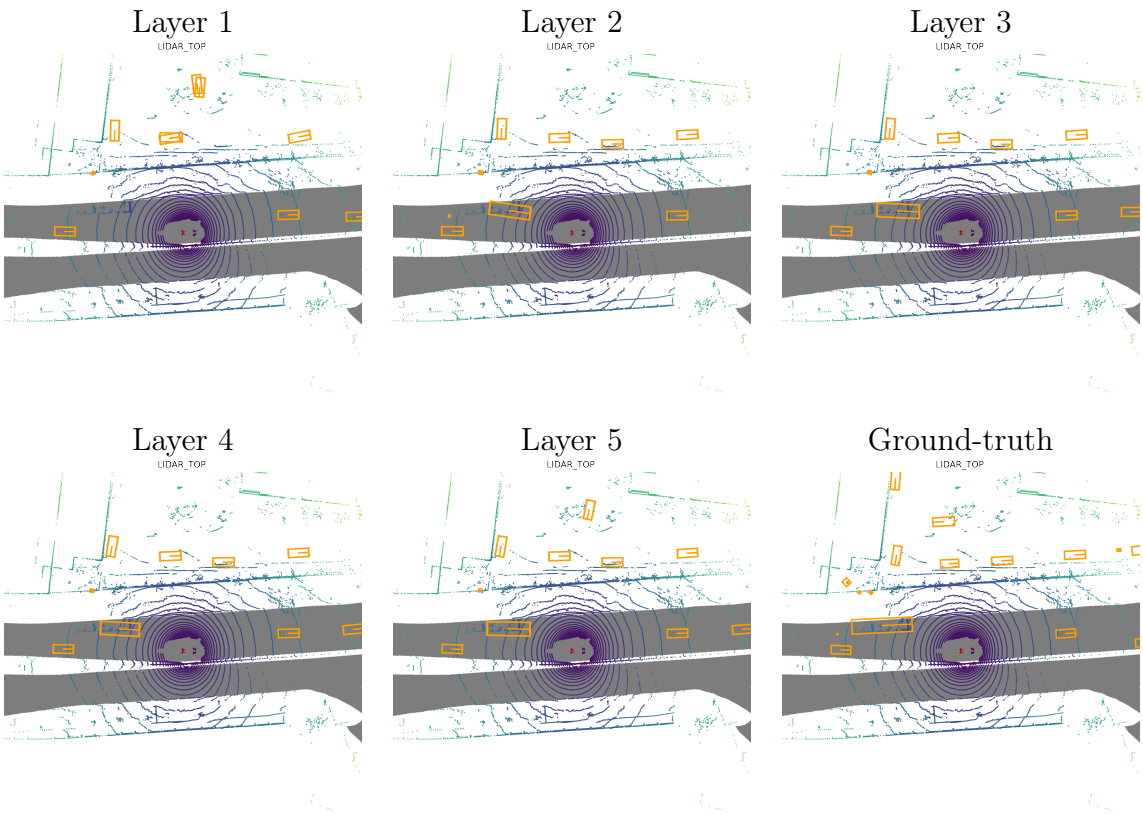


Figure 5-2: Detection results from layer 1 to layer 5 in the DETR3D head. We visualize the bounding boxes in the BEV and overlay the point clouds from `lidar_top`. The predictions get closer to the ground-truth in the deeper layers.

Table 5.6: Results with different number of queries.

# queries	30	100	300	600	900	1200	1500
mAP ↑	0.201	0.313	0.338	0.347	0.346	0.340	0.346
NDS ↑	0.331	0.408	0.415	0.420	0.425	0.415	0.420

Table 5.7: Results with different backbones.

Backbone ↑	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
ResNet50	0.373	0.302	0.811	0.282	0.493	0.979	0.212
ResNet101	0.425	0.346	0.773	0.268	0.383	0.842	0.216
DLA34	0.394	0.312	0.829	0.276	0.450	0.844	0.221

5.5 Conclusion

We propose a new paradigm to address the ill-posed inverse problem of recovering 3D information from 2D images. In this setting, the input signal lacks essential information for models to make effective predictions without priors learned from data. While other methods either operate solely on 2D computations or use additional depth networks to reconstruct the scene, ours operates in 3D space and uses backward projection to retrieve image features as needed. The benefits of our approach are two-fold: (1) it eliminates the need for middle-level representations (e.g., predicted depth maps or point clouds), which can be a source of compounding errors; and (2) it uses information from multiple cameras by projecting the same 3D point onto all available frames.

Beyond the direct application of our work to 3D object detection for autonomous driving, there are several venues that warrant future investigation. For example, single point projection creates a limited receptive field in the retrieved image feature maps, and sampling multiple points for each object query would incorporate more information for object refinement. Furthermore, the new detection head is input-agnostic, and including other modalities such as LiDAR/RADAR would enhance performance and robustness. Finally, generalizing our pipeline to other domains such as indoor navigation and object manipulation would increase its scope of application and reveal additional ways for further improvement.

Chapter 6

Deep Closest Point

Previous chapters address how to learn representations for high-level semantic reasoning tasks such as shape labeling and object detection. In the low-level domain, we aim to answer a similar question – can geometric registration benefit from learning? Geometric registration is a key task in many computational fields, including medical imaging, robotics, autonomous driving, and computational chemistry. In its most basic incarnation, registration involves the prediction of a rigid motion to align one shape to another, potentially obfuscated by noise and partiality.

Many modeling and computational challenges hamper the design of a stable and efficient registration method. Given exact correspondences, singular value decomposition yields the globally optimal alignment; similarly, computing matchings becomes easier given some global alignment information. Given these two observations, most algorithms alternate between these two steps to try to obtain a better result. The resultant iterative optimization algorithms, however, are prone to local optima.

The most popular example, Iterative Closest Point (ICP) [13, 160], alternates between estimating the rigid motion based on a fixed correspondence estimate and updating the correspondences to their closest matches. Although ICP monotonically decreases a certain objective function measuring alignment, due to the non-convexity of the problem, ICP often stalls in suboptimal local minima. Many methods [152, 46, 227] attempt to alleviate this issue by using heuristics to improve the matching or

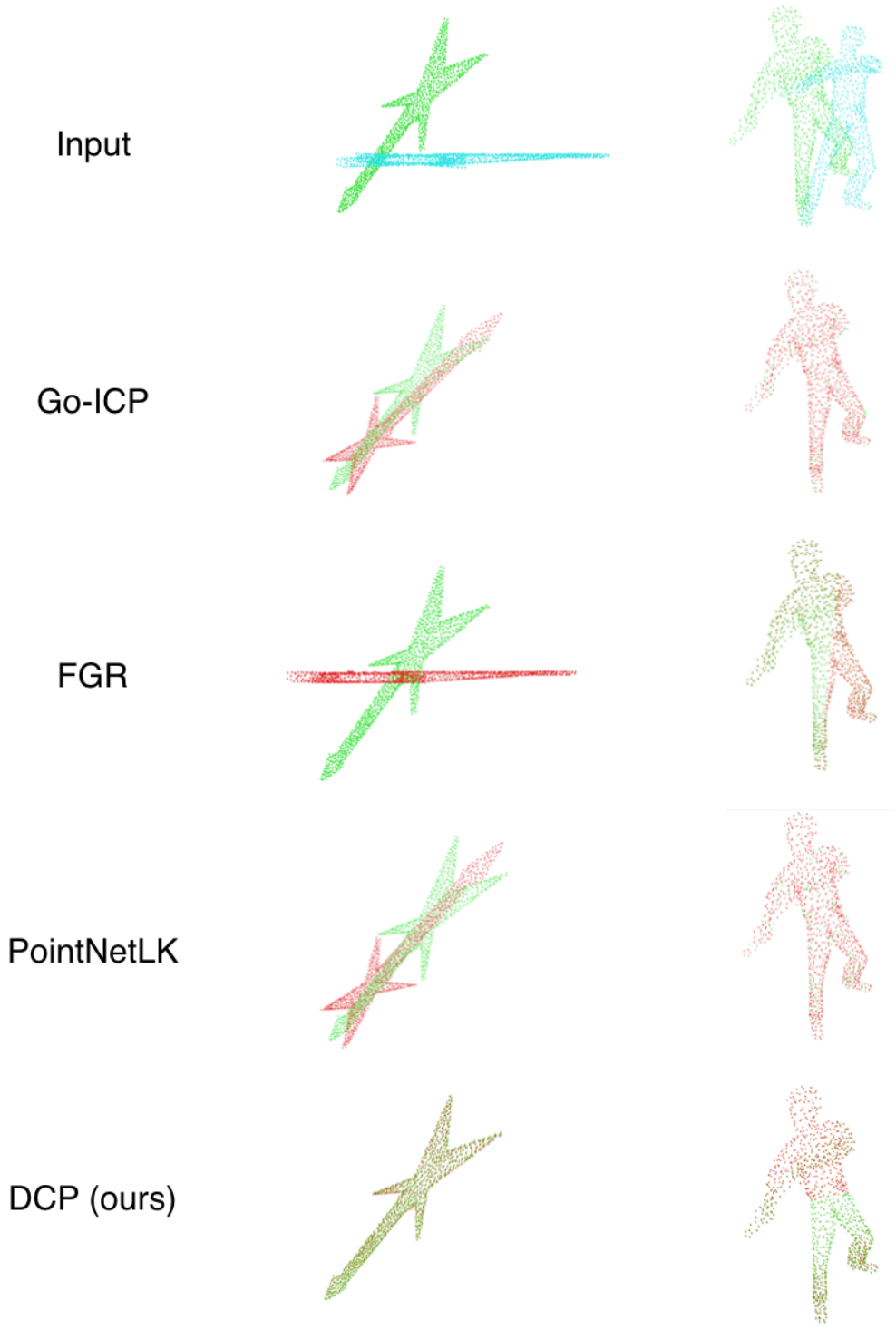


Figure 6-1: **Left**: a moved guitar. **Right**: rotated human. All methods work well with small transformation. However, only our method achieve satisfying alignment for objects with sharp features and large transformation.

by searching larger parts of the motion space SE(3). These algorithms are typically slower than ICP and still do not always provide acceptable output.

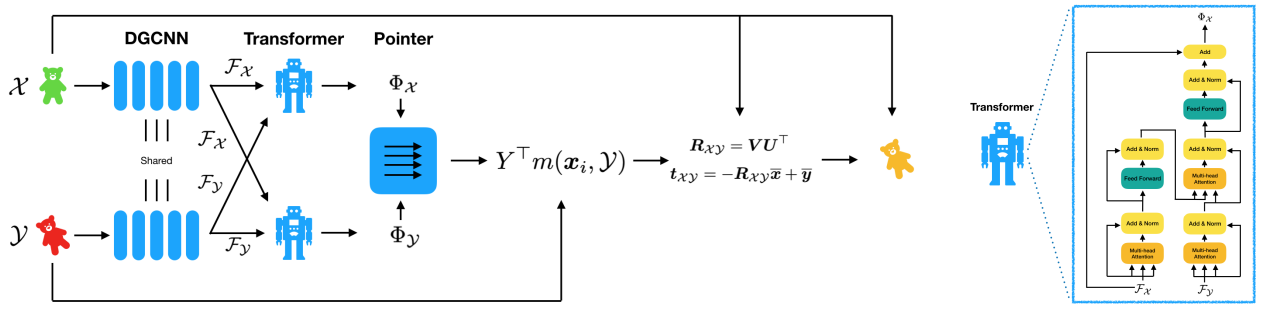
6.1 Introduction

In this work, we revisit ICP from a deep learning perspective, addressing key issues in each part of the ICP pipeline using modern machine learning, computer vision, and natural language processing tools. We call our resulting algorithm *Deep Closest Point (DCP)*, a learning-based method that takes two point clouds and predicts a rigid transformation aligning them.

Our model consists of three parts: (1) We map the input point clouds to permutation and rigid-invariant embeddings that help identify matching pairs of points (we compare PointNet [141] and DGCNN [215] for this step); then, (2) an attention-based module combining pointer network [202, 200] predicts a soft matching between the point clouds; and finally, (3) a differentiable singular value decomposition layer predicts the rigid transformation. We train and test our model end-to-end on ModelNet40 [218] in various settings, showing our model is not only efficient but also outperforms ICP and its extensions, as well as the recently-proposed PointNetLK method [51]. Our learned features generalize to unseen data, suggesting that our model is learning salient geometric features.

Contributions. Our contributions include the following:

- We identify sub-network architectures designed to address difficulties in the classical ICP pipeline.
- We propose a simple architecture to predict a rigid transformation aligning two point clouds.
- We evaluate efficiency and performance in several settings and provide an ablation study to support details of our construction.
- We analyze whether local or global features are more useful for registration.



(a) Network architecture (b) Transformer module

Figure 6-2: Network architecture for DCP, including the Transformer module for DCP-v2.

- We release our code to facilitate reproducibility and future research.

6.2 Related Work

Point cloud registration methods. ICP [13] is the best-known algorithm for solving rigid registration problems; it alternates between finding point cloud correspondences and solving a least-squares problem to update the alignment. ICP variants [152, 160, 17] consider issues with the basic method, like noise, partiality, and sparsity; probabilistic models [2, 64, 69] also can improve resilience to uncertain data. ICP can be viewed as an optimization algorithm searching jointly for a matching and a rigid alignment. Hence, [46] propose using the Levenberg–Marquardt algorithm to optimize the objective directly, which can yield a better solution. For more information, [137, 152] summarize ICP and its variants developed over the last 20 years.

ICP-style methods are prone to local minima due to non-convexity. To find a good optimum, Go-ICP [227] uses a branch-and-bound (BnB) method to search the motion space SE(3). It outperforms local ICP methods when a global solution is desired but is several orders of magnitude slower than other ICP variants despite using local ICP to accelerate the search process. Other methods attempt to identify global optima using Riemannian optimization [151], convex relaxation [115], and mixed-integer programming [72].

Recently, descriptor learning methods have brought significant progress in point cloud registration: 3DMatch [238] proposes learning a local volumetric patch descriptor to establish correspondences; 3DFeatNet [228] takes similar approach to point cloud representation for local regions; PPF-FoldNet [37] uses a folding-based autoencoder to learn a local descriptor; and 3DSmoothNet [52] employs a voxelized smoothed density value (SDV) representation for descriptor learning. The critical difference between our algorithm and these techniques is that we carry out end-to-end registration prediction while the others target descriptor learning. Also, these works rely on keypoint detection and outlier removal using RANSAC. Concurrent work [112] proposes an end-to-end pipeline for point cloud registration. A significant difference is that their methods compute loss for each point sample while ours optimizes the registration objective directly.

Learning on graphs and point sets. A broad class of deep architectures for geometric data termed *geometric deep learning* [18] includes recent methods learning on graphs [219, 243, 45] and point clouds [141, 143, 215, 236].

The *graph neural network* (GNN) is introduced in [159]; similarly, [39] defines convolution on graphs (GCN) for molecular data. [81] uses renormalization to adapt to the graph structure and applies GCN to semi-supervised learning on graphs. MoNet [123] learns a dynamic aggregation function based on the graph structure, generalizing GNN. Finally, graph attention networks (GAT) [201] incorporate multi-head attention into GCN. DGCNN [215] (discussed below) can be regarded as a graph neural network applied to point clouds with dynamic edges.

Another branch of geometric deep learning includes PointNet [141] and other algorithms designed to process point clouds. PointNet can be seen as applying GCN to graphs without edges, mapping points in $\mathbb{R}^3$ to high-dimensional space. PointNet only encodes global features gathered from the point cloud's embedding, impeding application to tasks involving local geometry. To address this issue, PointNet++ [143] applies a shared PointNet to k-nearest neighbor clusters to learn local features. As an alternative, DGCNN [215] explicitly recovers the graph structure in both Euclidean

space and feature space and applies graph neural networks to the result. PCNN [4] uses an extension operator to define convolution on point clouds, while PointCNN [94] applies Euclidean convolution after applying a learned transformation. Finally, SPLATNet [177] encodes point clouds on a lattice and performs bilateral convolution. All these works aim to apply convolution-like operations to point clouds and extract local geometric features.

Sequence-to-sequence learning and pointer networks. Many tasks in natural language processing, including machine translation, language modeling, and question answering, can be formulated as sequence-to-sequence (seq2seq) problems. [186] first uses deep neural networks (DNN) to address seq2seq problems at large scale. Seq2seq, however, often involves predicting discrete tokens corresponding to positions in the input sequence. This problem is difficult because there is an exponential number of possible matchings between input and output positions. Similar problems can be found in optimal transport [171, 135], combinatorial optimization [71], and graph matching [222]. To address this issue, in our registration pipeline we use a related method to Pointer Networks [202], which use attention as a pointer to select from the input sequence. In each output step, a Pointer Network predicts a distribution over positions and uses it as a "soft pointer." The pointer module is fully differentiable, and the whole network can be trained end-to-end.

Non-local approaches. To denoise images, non-local means [21] leverages the simple observation that Gaussian noise can be removed by non-locally weighted averaging all pixels in an image. Recently, non-local neural networks [206] have been proposed to capture long-range dependencies in video understanding; [220] uses the non-local module to denoise feature maps to defend against adversarial attacks. Another instantiation of non-local neural networks, known as *relational networks* [157], has shown effectiveness in visual reasoning [157], meta-learning [185], object detection [67], and reinforcement learning [237]. Its counterpart in natural language processing, attention, is arguably the most fruitful recent advance in this discipline. [200] replaces

recurrent neural networks [76, 65] with a model called the Transformer, consisting of several stacked multi-head attention modules. Transformer-based models [38, 144] outperform other recurrent models by a considerable amount in natural language processing. In our work, we also use a Transformer to learn contextual information of point clouds.

6.3 Problem Statement

In this section, we formulate the rigid alignment problem and discuss the ICP algorithm, highlighting key issues in the ICP pipeline. We use $\mathcal{X}$ and $\mathcal{Y}$ to denote two point clouds, where $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_N\} \subset \mathbb{R}^3$ and $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_j, \ldots, \boldsymbol{y}_M\} \subset \mathbb{R}^3$. For ease of notation, we consider the simplest case, in which $M = N$. The methods we describe here extend easily to the $M \neq N$ case because DGCNN, Transformer, and Softmax treat inputs as unordered sets. None requires $\mathcal{X}$ and $\mathcal{Y}$ to have the same length or a bijective matching.

In the rigid alignment problem, we assume $\mathcal{Y}$ is transformed from $\mathcal{X}$ by an unknown rigid motion. We denote the rigid transformation as $[\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}]$ where $\boldsymbol{R}_{\mathcal{XY}} \in \mathrm{SO}(3)$ and $\boldsymbol{t}_{\mathcal{XY}} \in \mathbb{R}^3$. We want to minimize the mean-squared error $E(\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}})$, which—if $\mathbf{X}$ and $\mathcal{Y}$ are ordered the same way (meaning $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are paired)—can be written

$$E(\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}) = \frac{1}{N} \sum_{i}^{N} \|\boldsymbol{R}_{\mathcal{XY}} \boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{XY}} - \boldsymbol{y}_i\|^2. \tag{6.1}$$

Define centroids of $\mathbf{X}$ and $\mathcal{Y}$ as

$$\overline{\boldsymbol{x}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i \qquad \text{and} \qquad \overline{\boldsymbol{y}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{y}_i. \tag{6.2}$$

Then the cross-covariance matrix $\boldsymbol{H}$ is given by

$$\boldsymbol{H} = \sum_{i=1}^{N} (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{y}_i - \overline{\boldsymbol{y}})^\top. \tag{6.3}$$

We can use the singular value decomposition (SVD) to decompose $\boldsymbol{H} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$. Then, the alignment minimizing $E(\cdot, \cdot)$ in (6.1) is given in closed-form by

$$\boldsymbol{R}_{\mathcal{X}\mathcal{Y}} = \boldsymbol{V}\boldsymbol{U}^\top \qquad \text{and} \qquad \boldsymbol{t}_{\mathcal{X}\mathcal{Y}} = -\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}\overline{\boldsymbol{x}} + \overline{\boldsymbol{y}}. \tag{6.4}$$

Here, we take the convention that $\boldsymbol{U}, \boldsymbol{V} \in \mathrm{SO}(3)$, while $\boldsymbol{S}$ is diagonal but potentially signed; this accounts for orientation-reversing choices of $\boldsymbol{H}$. This classic orthogonal Procrustes problem assumes that the point sets are matched to each other, that is, that $\boldsymbol{x}_i$ should be mapped to $\boldsymbol{y}_i$ in the final alignment for all i. If the correspondence is unknown, however, the objective function E must be revised to account for matching:

$$E(\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}, \boldsymbol{t}_{\mathcal{X}\mathcal{Y}}) = \frac{1}{N}\sum_{i}^{N} \|\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}\boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{X}\mathcal{Y}} - \boldsymbol{y}_{m(x_i)}\|^2. \tag{6.5}$$

Here, a mapping m from each point in $\mathbf{X}$ to its corresponding point in $\mathcal{Y}$ is given by

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \arg\min_{j} \|\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}\boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{X}\mathcal{Y}} - \boldsymbol{y}_j\|. \tag{6.6}$$

Equations (6.5) and (6.6) form a classic *chicken-and-egg* problem. If we know the optimal rigid transformation $[\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}, \boldsymbol{t}_{\mathcal{X}\mathcal{Y}}]$, then the mapping m can be recovered from (6.6); conversely, given the optimal mapping m, the transformation can be computed using (6.4).

ICP iteratively approaches a stationary point of E in (6.5), including the mapping $m(\cdot)$ as one of the variables in the optimization problem. It alternates between two steps: finding the current optimal transformation based on a previous mapping m^{k-1} and finding an optimal mapping m^k based on the current transformation using (6.6), where k denotes the current iteration. The algorithm terminates when a fixed point or stall criterion is reached. This procedure is easy to implement and relatively efficient, but it is extremely prone to local optima; a distant initial alignment yields a poor estimate of the mapping m, quickly leading to a situation where the algorithm gets stuck. Our goal is to use learned embeddings to recover a better matching $m(\cdot)$ and

to use this matching to compute a rigid transformation, as we will detail in the next.

6.4 Deep Closest Point

Having established preliminaries about the rigid alignment problem, we are now equipped to present our Deep Closest Point architecture, illustrated in Figure 6.2. In short, we embed point clouds into high-dimensional space using PointNet [141] or DGCNN [215] (§6.4.1), encode contextual information using an attention-based module (§6.4.2), and finally estimate an alignment using a differentiable SVD layer (§6.4.4).

6.4.1 Initial Features

The first stage of our pipeline embeds the unaligned input point clouds $\mathcal{X}$ and $\mathcal{Y}$ into a common space used to find matching pairs of points between the two clouds. The goal is to find an embedding that quotients out rigid motion while remaining sensitive to relevant features for rigid matching. We evaluate two possible choices of learnable embedding modules, PointNet [141] and DGCNN [215].

Since we use per-point embeddings of the two input point clouds to generate a mapping m and recover the rigid transformation, we seek a feature *per point* in the input point clouds rather than one feature per cloud. For this reason, in these two network architectures, we use the representations generated before the last aggregation function, notated $\mathcal{F}_{\mathcal{X}} = \{\boldsymbol{x}_1^L, \boldsymbol{x}_2^L, ..., \boldsymbol{x}_i^L, ..., \boldsymbol{x}_N^L\}$ and $\mathcal{F}_{\mathcal{Y}} = \{\boldsymbol{y}_1^L, \boldsymbol{y}_2^L, ..., \boldsymbol{y}_i^L, ..., \boldsymbol{y}_N^L\}$, assuming a total of L layers.

In more detail, PointNet takes a set of points, embeds each by a nonlinear function from $\mathbb{R}^3$ into a higher-dimensional space, and optionally outputs a global feature vector for the whole point cloud after applying a channel-wise aggregation function f (e.g., max or $\sum$). Let $\boldsymbol{x}_i^l$ be the embedding of point i in the l-th layer, and let h_θ^l be a nonlinear function in the l-th layer parameterized by a shared multilayer perceptron (MLP). Then, the forward mechanism is given by $\boldsymbol{x}_i^l = h_\theta^l(\boldsymbol{x}_i^{l-1})$.

While PointNet largely extracts information based on the embedding of each point

in the point cloud independently, DGCNN explicitly incorporates local geometry into its representation. In particular, given a set of points $\mathcal{X}$, DGCNN constructs a k-NN graph $\mathcal{G}$, applies a nonlinearity to the values at edge endpoints to obtain edgewise values, and performs vertex-wise aggregation (max or $\sum$) in each layer. The forward mechanism of DGCNN is thus

$$\boldsymbol{x}_i^l = f(\{h_\theta^l(\boldsymbol{x}_i^{l-1}, \boldsymbol{x}_j^{l-1}) \; \forall j \in \mathcal{N}_i\}), \tag{6.7}$$

where $\mathcal{N}_i$ denotes the neighbors of vertex i in graph $\mathcal{G}$. While PointNet features do not incorporate local neighborhood information, we find empirically that DGCNN's local features are critical for high-quality matching in subsequent steps of our pipeline (see §6.6.3).

6.4.2 Attention

Our transition from PointNet to DGCNN is motivated by the observation that the most useful features for rigid alignment are learned jointly from local and global information. We additionally can improve our features for matching by making them *task-specific*, that is, changing the features depending on the particularities of $\mathcal{X}$ and $\mathcal{Y}$ together rather than embedding $\mathcal{X}$ and $\mathcal{Y}$ independently. That is, the task of rigidly aligning, say, organic shapes might require different features than those for aligning mechanical parts with sharp edges. Inspired by the recent success of BERT [38], non-local neural networks [206], and relational networks [157] using attention-based models, we design a module to learn co-contextual information by capturing *self-attention* and *conditional attention*.

Take $\mathcal{F}_\mathcal{X}$ and $\mathcal{F}_\mathcal{Y}$ to be the embeddings generated by the modules in §6.4.1; these embeddings are computed independently of one another. Our attention model learns a function $\phi : \mathbb{R}^{N\times P} \times \mathbb{R}^{N\times P} \to \mathbb{R}^{N\times P}$, where P is embedding dimension, that provides

new embeddings of the point clouds as

$$\begin{aligned} \Phi_{\mathcal{X}} &= \mathcal{F}_{\mathcal{X}} + \phi(\mathcal{F}_{\mathcal{X}}, \mathcal{F}_{\mathcal{Y}}) \\ \Phi_{\mathcal{Y}} &= \mathcal{F}_{\mathcal{Y}} + \phi(\mathcal{F}_{\mathcal{Y}}, \mathcal{F}_{\mathcal{X}}) \end{aligned} \tag{6.8}$$

Notice we treat ϕ as a *residual* term, providing an additive change to $\mathcal{F}_{\mathcal{X}}$ and $\mathcal{F}_{\mathcal{Y}}$ depending on the order of its inputs. The idea here is that the map $\mathcal{F}_{\mathcal{X}} \mapsto \Phi_{\mathcal{X}}$ modifies the features associated to the points in $\mathcal{X}$ in a fashion that is knowledgeable about the structure of $\mathcal{Y}$; the map $\mathcal{F}_{\mathcal{Y}} \mapsto \Phi_{\mathcal{Y}}$ serves a symmetric role. We choose ϕ as an asymmetric function given by a Transformer [200]. The Transformer is a framework to solve sequence-to-sequence problems. It consists of several stacked encoder-decoder layers. The encoder takes one sequence/set ($\mathcal{F}_{\mathcal{X}}$) and encodes it to an embedding space by using a self-attention layer and shared multi-layer perceptron (MLP). The decoder has two parts: The first part takes another sequence/set ($\mathcal{F}_{\mathcal{Y}}$) and encodes it in the same way as the encoder, and the second part relates two embedded sequences/sets using co-attention. Therefore, the output embeddings ($\Phi_{\mathcal{X}}$ and $\Phi_{\mathcal{Y}}$) have contextual information from both sequences/sets ($\mathcal{F}_{\mathcal{X}}$ and $\mathcal{F}_{\mathcal{Y}}$). The matching problem we encounter in rigid alignment is analogous to the sequence-to-sequence problem that inspired its development, other than their use of positional embeddings to describe where words are in a sentence.

6.4.3 Pointer Generation

The most common failure mode of ICP occurs when the matching estimate m^k is far from optimal. When this occurs, the rigid motion subsequently estimated using (6.6) does not significantly improve alignment, leading to a spurious local optimum. As an alternative, our learned embeddings are trained specifically to expose matching pairs of points using a simple procedure explained below. We term this step *pointer generation*, again inspired by terminology in the attention literature introduced in §6.4.2.

To avoid choosing non-differentiable hard assignments, we use a probabilistic ap-

proach that generates a (singly-stochastic) "soft map" from one point cloud into the other. That is, each $\boldsymbol{x}_i \in \mathcal{X}$ is assigned a probability vector over elements of $\mathcal{Y}$ given by

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \text{softmax}(\Phi_{\mathcal{Y}} \Phi_{\boldsymbol{x}_i}^{\top}). \tag{6.9}$$

Here, $\Phi_{\mathcal{Y}} \in \mathbb{R}^{N \times P}$ denotes the embedding of $\mathcal{Y}$ generated by the attention module, and $\Phi_{\boldsymbol{x}_i}$ denotes the i-th row of the matrix $\Phi_{\mathcal{X}}$ from the attention module. We can think of $m(\boldsymbol{x}_i, \mathcal{Y})$ as a *soft pointer* from each $\boldsymbol{x}_i$ into the elements of $\mathcal{Y}$.

6.4.4 SVD Module

The final module in our architecture extracts the rigid motion from the soft matching computed in §6.4.3. We use the soft pointers to generate a matching averaged point in $\mathcal{Y}$ for each point in $\mathcal{X}$:

$$\hat{\boldsymbol{y}}_i = Y^{\top} m(\boldsymbol{x}_i, \mathcal{Y}) \in \mathbb{R}^3. \tag{6.10}$$

Here, we define $Y \in \mathbb{R}^{N \times 3}$ to be a matrix containing the points in Y. Then, $\boldsymbol{R}_{\mathcal{XY}}$ and $\boldsymbol{t}_{\mathcal{XY}}$ are extracted using (6.4) based on the pairing $\boldsymbol{x}_i \mapsto \hat{\boldsymbol{y}}_i$ over all i.

To backpropagate gradients through the networks, we need to differentiate the SVD. [131] describes a standard means of computing this derivative; versions of this calculation are included in PyTorch [134] and TensorFlow [1]. Note we need to solve only 3×3 eigenproblems, small enough to be solved using simple algorithms or even (in principle) a closed-form formula.

6.4.5 Loss

Combined, the modules above map from a pair of point clouds $\mathcal{X}$ and $\mathcal{Y}$ to a rigid motion $[\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}]$ that aligns them to each other. The initial feature module (§6.4.1) and the attention module (§6.4.2) are both parameterized by a set of neural network weights, which must be learned during a training phase. We employ a fairly straightforward strategy for training, measuring the deviation of $[\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}]$ from ground

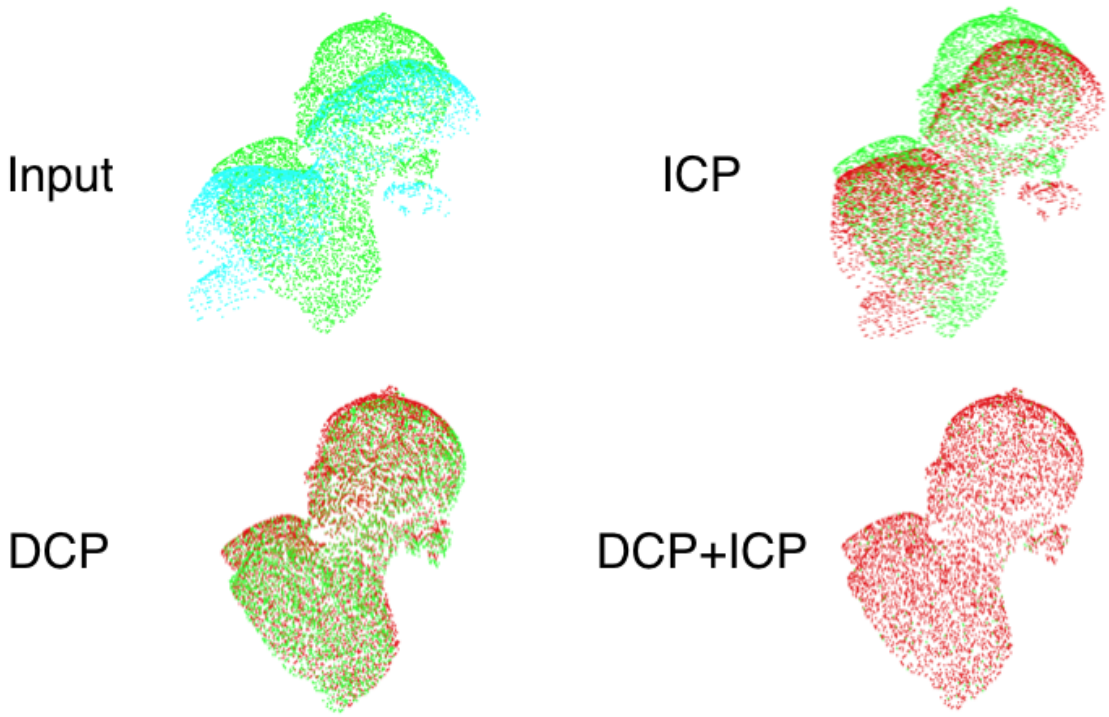


Figure 6-3: **Top left**: input. **Top right**: result of ICP with random initialization. **Bottom left**: initial transformation provided by DCP. **Bottom right**: result of ICP initialized with DCP. Using a good initial transformation provided by DCP, ICP converges to the global optimum.

truth for synthetically-generated pairs of point clouds.

We use the following loss function to measure our model's agreement to the ground-truth rigid motions:

$$\text{Loss} = \|\boldsymbol{R}_{\mathcal{XY}}^{\top}\boldsymbol{R}_{\mathcal{XY}}^{g} - I\|^2 + \|\boldsymbol{t}_{\mathcal{XY}} - \boldsymbol{t}_{\mathcal{XY}}^{g}\|^2 + \lambda\|\theta\|^2. \tag{6.11}$$

Here, g denotes ground-truth. The first two terms define a simple distance on SE(3). The third term denotes Tikhonov regularization of the DCP parameters θ, which serves to reduce the complexity of the network.

6.5 Experiments

We compare our models to ICP, Go-ICP [227], Fast Global Registration (FGR) [244], and the recently-proposed PointNetLK deep learning method [51]. We denote our model without attention (§6.4.2) as **DCP-v1** and the full model with attention as **DCP-v2**. Go-ICP is ported from the authors' released code. For ICP and FGR, we use the implementations in Intel Open3D [245]. For PointNetLK, we adapt the code partially released by the authors. Notice that FGR [244] uses additional geometric features. In all experiments, the proposed method does *not* make an assumption that a good initial pose is given; the test point clouds are generated in the same way as the training point clouds. ICP and its variants are initialized with an identity transformation matrix.

The architecture of DCP is shown in Figure 6.2. We use 5 *EdgeConv* (denoted as DGCNN [215]) layers for both DCP-v1 and DCP-v2. The numbers of filters in each layer are $[64, 64, 128, 256, 512]$. In the Transformer layer, the number of heads in multi-head attention is 4 and the embedding dimension is 1024. We use LayerNorm [7] without Dropout [175]. Adam [80] is used to optimize the network parameters, with an initial learning rate of 0.001. We divide the learning rate by 10 at epochs 75, 150, and 200, training for a total of 250 epochs. DCP-v1 does not use the Transformer module but rather employs identity mappings $\Phi_{\mathcal{X}} = \mathcal{F}_{\mathcal{X}}$ and $\Phi_{\mathcal{Y}} = \mathcal{F}_{\mathcal{Y}}$.

We experiment on the ModelNet40 [218] dataset, which consists of 12,311 meshed CAD models from 40 categories. Of these, we use 9,843 models for training and 2,468 models for testing. We follow the experimental settings of PointNet [141], uniformly sampling 1,024 points from each model's outer surface. As in previous work, points are centered and rescaled to fit in the unit sphere, and no features other than (x, y, z) coordinates appear in the input.

We measure mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE) between ground truth values and predicted values. Ideally, all of these error metrics should be zero if the rigid alignment is perfect. All angular measurements in our results are in units of degrees.

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	894.897339	29.914835	23.544817	0.084643	0.290935	0.248755
Go-ICP [227]	140.477325	11.852313	2.588463	0.000659	0.025665	0.007092
FGR [244]	87.661491	9.362772	1.999290	0.000194	0.013939	0.002839
PointNetLK [51]	227.870331	15.095374	4.225304	0.000487	0.022065	0.005404
DCP-v1 (ours)	6.480572	2.545697	1.505548	**0.000003**	**0.001763**	**0.001451**
DCP-v2 (ours)	**1.307329**	**1.143385**	**0.770573**	**0.000003**	0.001786	**0.001195**

Table 6.1: ModelNet40: Test on unseen point clouds

6.5.1 ModelNet40: Full Dataset Train & Test

In our first experiment, we randomly divide all the point clouds in the ModelNet40 dataset into training and test sets, with no knowledge of the category label; different point clouds are used during training and during testing. During training, we sample a point cloud $\mathcal{X}$. Along each axis, we randomly draw a rigid transformation; the rotation along each axis is uniformly sampled in $[0, 45^\circ]$ and translation is in $[-0.5, 0.5]$. $\mathcal{X}$ and a transformation of $\mathcal{X}$ by the rigid motion are used as input to the network, which is evaluated against the known ground truth using (6.11).

Table 6.1 evaluates the performance of our method and its peers in this experiment (vanilla ICP nearly fails). DCP-v1 already outperforms other methods under all the performance metrics, and DCP-v2 exhibits even stronger performance.

6.5.2 ModelNet40: Category Split

To test the generalizability of different models, we split ModelNet40 evenly by category into training and testing sets. We train DCP and PointNetLK on the first 20 categories, then test them on the held-out categories. ICP, Go-ICP, and FGR are also tested on the held-out categories. As shown in Table 6.2, on unseen categories, FGR behaves more strongly than other methods. DCP-v1 has much worse performance than DCP-v2, supporting our use of the attention module. Although the learned representations are task-dependent, DCP-v2 exhibits smaller error than others except for FGR, including the learning-based method PointNetLK.

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	892.601135	29.876431	23.626110	0.086005	0.293266	0.251916
Go-ICP [227]	192.258636	13.865736	2.914169	0.000491	0.022154	0.006219
FGR [244]	97.002747	9.848997	**1.445460**	0.000182	0.013503	0.002231
PointNetLK [51]	306.323975	17.502113	5.280545	0.000784	0.028007	0.007203
DCP-v1 (ours)	19.201385	4.381938	2.680408	**0.000025**	**0.004950**	**0.003597**
DCP-v2 (ours)	**9.923701**	**3.150191**	2.007210	**0.000025**	0.005039	0.003703

Table 6.2: ModelNet40: Test on unseen categories

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	882.564209	29.707983	23.557217	0.084537	0.290752	0.249092
Go-ICP [227]	131.182495	11.453493	2.534873	0.000531	0.023051	0.004192
FGR [244]	607.694885	24.651468	10.055918	0.011876	0.108977	0.027393
PointNetLK [51]	256.155548	16.004860	4.595617	0.000465	0.021558	0.005652
DCP-v1 (ours)	6.926589	2.631841	1.515879	0.000003	0.001801	0.001697
DCP-v2 (ours)	**1.169384**	**1.081380**	**0.737479**	**0.000002**	**0.001500**	**0.001053**

Table 6.3: ModelNet40: Test on objects with Gaussian noise

6.5.3 ModelNet40: Resilience to Noise

We also experiment with adding noise to each point of the input point clouds. We sample noise independently from $\mathcal{N}(0, 0.01)$, clip the noise to $[-0.05, 0.05]$, and add it to $\mathcal{X}$ during testing. In this experiment, we use the model from §6.5.1 trained on noise-free data from all of ModelNet40.

Table 6.3 shows the results of this experiment. ICP typically converges to a far-away fixed point, and FGR is sensitive to noise. Go-ICP, PointNetLK, and DCP, however, remain robust to noise.

6.5.4 DCP Followed By ICP

Since our experiments involve point clouds whose initial poses are far from aligned, ICP fails nearly every experiment we have presented so far. In large part, this failure is due to the lack of a good initial guess. As an alternative, we can use ICP as a *local* algorithm by initializing ICP with a rigid transformation output from our DCP model. Figure 6-3 shows an example of this two-step procedure; while ICP fails at

# points	ICP	Go-ICP	FGR	PointNetLK	DCP-v1	DCP-v2
512	0.003972	15.012375	0.033297	0.043228	0.003197	0.007932
1024	0.004683	15.405995	0.088199	0.055630	0.003300	0.008295
2048	0.044634	15.766001	0.138076	0.146121	0.040397	0.073697
4096	0.044585	15.984596	0.157124	0.162007	0.039984	0.74263

Table 6.4: Inference time (in seconds)

the global alignment task, with better initialization provided by DCP, it converges to the global optimum. In some sense, this experiment shows how ICP can be an effective way to "polish" the alignment generated by DCP.

6.5.5 Efficiency

We profile the inference time of different methods on a desktop computer with an Intel I7-7700 CPU, an Nvidia GTX 1070 GPU, and 32G memory. Computational time is measured in seconds and is computed by averaging 100 results. As shown in Table 6.4, DCP-v1 is the fastest method among our points of comparison, and DCP-v2 is only slower than vanilla ICP.

6.6 Ablation Study

We conduct several ablation experiments in this section, dissecting DCP and replacing each part with an alternative to understand the value of our construction. All experiments are done in the same setting as the experiments in §6.5.1.

6.6.1 MLP or SVD?

While MLP is in principle a universal approximator, our SVD layer is designed to compute a rigid motion specifically. In this experiment, we examine whether an MLP or a custom-designed layer is better for registration. We compare MLP and SVD with both DCP-v1 and DCP-v2 on ModelNet40. Table 6.5 shows both DCP-v1 and

Metrics	DCP-v1+MLP	DCP-v1+SVD	DCP-v2+MLP	DCP-v2+SVD
MSE($\boldsymbol{R}$)	21.115917	6.480572	9.923701	1.307329
RMSE($\boldsymbol{R}$)	4.595206	2.545697	3.150191	1.143385
MAE($\boldsymbol{R}$)	3.291298	1.505548	2.007210	0.770573
MSE($\boldsymbol{t}$)	0.000861	0.000003	0.000025	0.000003
RMSE($\boldsymbol{t}$)	0.029343	0.001763	0.005039	0.001786
MAE($\boldsymbol{t}$)	0.022501	0.001451	0.003703	0.001195

Table 6.5: Ablation study: MLP or SVD?

Metrics	DCP-v1 (512)	DCP-v1 (1024)	DCP-v2 (512)	DCP-v2 (1024)
MSE($\boldsymbol{R}$)	6.480572	7.291216	1.307329	1.217545
RMSE($\boldsymbol{R}$)	2.545697	2.700225	1.143385	1.103424
MAE($\boldsymbol{R}$)	1.505548	1.616465	0.770573	0.750242
MSE($\boldsymbol{t}$)	0.000003	0.000001	0.000003	0.000003
RMSE($\boldsymbol{t}$)	0.001763	0.001150	0.001786	0.001696
MAE($\boldsymbol{t}$)	0.001451	0.000677	0.001195	0.001170

Table 6.6: Ablation study: Embedding dimension

DCP-v2 perform better with SVD layer than MLP. This supports our motivation to compute rigid transformation using SVD.

6.6.2 Embedding Dimension

[141] remarks that the embedding dimension is an important parameter affecting the accuracy of point cloud deep learning models up to a critical threshold, after which there is an insignificant difference. To verify our choice of dimensionality, we compare models with embeddings into spaces of different dimensions. We test models with DCP-v1 and v2, using DGCNN to embed the point clouds into $\mathbb{R}^{512}$ or $\mathbb{R}^{1024}$. The results in Table 6.6 show that increasing the embedding dimension from 512 to 1024 does marginally help DCP-v2, but for DCP-v1 there is small degeneracy. Our results are consistent with the hypothesis in [143].

Metrics	PN+DCP-v1,	DGCNN+DCP-v1	PN+DCP-v2	DGCNN+DCP-v2
MSE($\boldsymbol{R}$)	17.008427	6.480572	49.863022	1.307329
RMSE($\boldsymbol{R}$)	4.124127	2.545697	7.061375	1.143385
MAE($\boldsymbol{R}$)	2.800184	1.505548	4.485052	0.770573
MSE($\boldsymbol{t}$)	0.000697	0.000003	0.000258	0.000003
RMSE($\boldsymbol{t}$)	0.026409	0.001763	0.016051	0.001786
MAE($\boldsymbol{t}$)	0.01327	0.001451	0.010546	0.001195

Table 6.7: Ablation study: PointNet or DGCNN?

6.6.3 PointNet or DGCNN?

We first try to answer whether the localized features gathered by DGCNN provide value over the coarser features that can be measured using the simpler PointNet model. As discussed in [215], PointNet [141] learns a global descriptor of the whole shape while DGCNN [215] learns local geometric features via constructing the k-NN graph. We replace the DGCNN with PointNet (denoted as PN) and conduct the experiments in §6.5.1 on ModelNet40 [218], using DCP-v1 and DCP-v2. Table 6.7. Models perform consistently better with DGCNN than their counterparts with PointNet.

6.7 Conclusion

In some sense, the key observation in our Deep Closest Point technique is that learned features greatly facilitate rigid alignment algorithms; by incorporating DGCNN [215] and an attention module, our model reliably extracts the correspondences needed to find rigid motions aligning two input point clouds. Our end-to-end trainable model is reliable enough to extract a high-quality alignment in a single pass, which can be improved by iteration or "polishing" via classical ICP.

DCP is immediately applicable to rigid alignment problems as a drop-in replacement for ICP with improved behavior. Beyond its direct usage, our experiments suggest several avenues for future inquiry. One straightforward extension is to see if our learned embeddings transfer to other tasks like classification and segmentation.

We could also train DCP to be applied *iteratively* (or recursively) to refine the alignment, rather than attempting to align in a single pass; insight from reinforcement learning could help refine approaches in this direction, using mean squared error as a reward to learn a policy that controls when to stop iterating.

We are also interested in testing on scenes, which often have up to 300,000 points. Current deep networks, however, can only handle object-level point clouds (each usually has around 500 to 5,000 points); this is a common limitation of recent point cloud learning methods. Testing on scenes, no matter the task, requires designing an efficient scene-level point cloud encoding network, which is a promising but challenging direction for point cloud learning generally. Also, DCP only tackles full-to-full shape registration; we leave partial-to-partial registration as a future direction that is studied in next chapter. Finally, we hope our method can be incorporated into larger pipelines to enable high-accuracy Simultaneous Localization and Mapping (SLAM) or Structure from Motion (SFM).

Chapter 7

Partial Registration Network

Registration is the problem of predicting a rigid motion aligning one point cloud to another. Algorithms for this task have steadily improved, using machinery from vision, graphics, and optimization. These methods, however, are usually orders of magnitude slower than "vanilla" Iterative Closest Point (ICP), and some have hyperparameters that must be tuned case-by-case. The trade-off between efficiency and effectiveness is steep, reducing generalizability and/or practicality.

Recently, PointNetLK [51] and Deep Closest Point (DCP) [212] (presented in the previous chapter) showed that learning-based registration can be faster and more robust than classical methods, even when trained on different datasets. These methods, however, cannot handle partial-to-partial registration, and their one-shot constructions preclude refinement of the predicted alignment.

7.1 Introduction

We introduce the Partial Registration Network (PRNet), a sequential decision-making framework designed to solve a broad class of registration problems. Like ICP, our method is designed to be applied *iteratively*, enabling coarse-to-fine refinement of an initial registration estimate. A critical new component of our framework is a keypoint detection sub-module, which identifies points that match in the input point clouds based on co-contextual information. Partial-to-partial point cloud registration then

boils down to detecting keypoints the two point clouds have in common, matching these keypoints to one another, and solving the Procrustes problem.

Since PRNet is designed to be applied iteratively, we use Gumbel–Softmax [74] with a straight-through gradient estimator to sample keypoint correspondences. This new architecture and learning procedure modulates the sharpness of the matching; distant point clouds given to PRNet can be coarsely matched using a diffuse (fuzzy) matching, while the final refinement iterations prefer sharper maps. Rather than introducing another hyperparameter, PRNet uses a sub-network to predict the temperature [188] of the Gumbel–Softmax correspondence, which can be cast as a simplified version of the actor-critic method. That is, PRNet *learns* to modulate the level of map sharpness each time it is applied.

We train and test PRNet on ModelNet40 and on real data. We visualize the keypoints and correspondences for shapes from the same or different categories. We transfer the learned representations to shape classification using a linear SVM, achieving comparable performance to state-of-the-art supervised methods on ModelNet40.

Contributions. We summarize our key contributions as follows:

- We present the *Partial Registration Network* (PRNet), which enables partial-to-partial point cloud registration using deep networks with state-of-the-art performance.
- We use Gumbel–Softmax with straight-through gradient estimation to obtain a sharp and near-differentiable mapping function.
- We design an *actor-critic closest point* module to modulate the sharpness of the correspondence using an action network and a value network. This module predicts more accurate rigid transformations than differentiable soft correspondence methods with fixed parameters.
- We show registration is a useful proxy task to learn representations for 3D shapes. Our representations can be transferred to other tasks, including keypoint detection, correspondence prediction, and shape classification.

7.2 Related Work

Rigid registration. ICP [13] and variants [152, 160, 17, 137, 119] have been widely used for registration. Recently, probabilistic models [2, 64, 69] have been proposed to handle uncertainty and partiality. Another trend is to improve the optimization: [46] applies Levenberg—Marquardt to the ICP objective, while global methods seek a solution using branch-and-bound [227], Riemannian optimization [151], convex relaxation [115], mixed-integer programming [72], and semidefinite programming [226].

Learning on point clouds and 3D shapes. Deep Sets [236] and PointNet [141] pioneered deep learning on point sets, a challenge problem in learning and vision. These methods take coordinates as input, embed them to high-dimensional space using shared multilayer perceptrons (MLPs), and use a symmetric function (e.g., max or $\sum$) to aggregate features. Follow-up works incorporate local information, including PointNet++ [143], DGCNN [215], PointCNN [94], and PCNN [4]. Another branch of 3D learning designs convolution-like operations for shapes or applies graph convolutional networks (GCNs) [39, 81] to triangle meshes [123, 208], exemplifying architectures on non-Euclidean data termed *geometric deep learning* [18]. Other works, including SPLATNet [177], SplineCNN [45], KPConv [190], and GWCNN [44], transform 3D shapes to regular grids for feature learning.

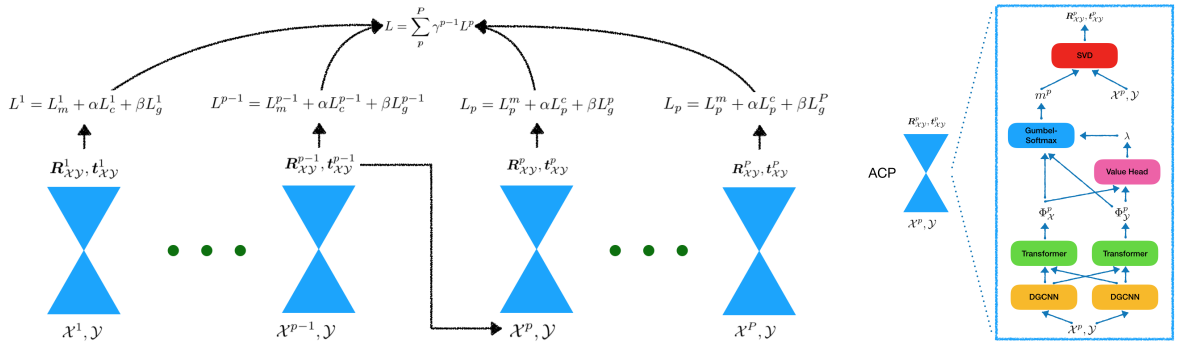
Keypoints and correspondence. Correspondence and registration are dual tasks. Correspondence is the approach while registration is the output, or vice versa. Countless efforts tackle the correspondence problem, either at the point-to-point or part-to-part level. Due to the $O(n^2)$ complexity of point-to-point correspondence matrices and $O(n!)$ possible permutations, most methods (e.g., [113, 68, 100, 79, 173, 172, 174, 43]) compute a sparse set of correspondences and extend them to dense maps, often with bijectivity as an assumption or regularizer. Other efforts use more exotic representations of correspondences. For example, functional maps [129] generalize to mappings between functions on shapes rather than points on shapes, expressing a map as a linear operator in the Laplace–Beltrami eigenbasis. Mathematical methods like functional maps can be made 'deep' using priors learned from data: Deep functional

maps [101, 58] learn descriptors rather than designing them by hand.

For partial-to-partial registration, we cannot compute bijective correspondences, invalidating many past representations. Instead, keypoint detection is more secure. To extract a sparser representation, KeyPointNet [187] uses registration and multi-view consistency as supervision to learn a keypoint detector on 2D images; our method performs keypoint detection on point clouds. In contrast to our model, which learns correspondences from registration, [229] uses correspondence prediction as the training objective to learn how to segment parts. In particular, it utilizes PointNet++ [143] to product point-wise features, generates matching using a correspondence proposal module, and finally trains the pipeline with ground-truth correspondences.

Self-supervised learning. Humans learn knowledge not only from teachers but also by predicting and reasoning about unlabeled information. Inspired by this observation, self-supervised learning usually involves predicting part of an input from another part [239, 240], solving one task using features learned from another task [187] and/or enforcing consistency from different views/modalities [26, 6]. Self-supervised pretraining is an effective way to transfer knowledge learned from massive unlabeled data to tasks where labeled data is limited. For example, BERT [38] surpasses state-of-the-art in natural language processing by learning from contextual information. ImageNet Pretrain [60] commonly provides initialization for vision tasks. Video-audio joint analysis [242, 130, 41] uses modality consistency to learn representations. Our method is also self-supervised, in the sense that no labeled data is needed.

Actor–critic methods. Many recent works can be counted as actor–critic methods, including deep reinforcement learning [122], generative modeling [136], and sequence generation [8]. These methods generally involve two functions: taking actions and estimating values. The predicted values can be used to improve the actions while the values are collected when the models interact with environment. PRNet uses a sub-module (value head) to predict the level of granularity at which we should map two shapes. The value adjusts the temperature of Gumbel–Softmax in the action head.



(a) Network architecture for PRNet (b) ACP

Figure 7-1: Network architecture for PRNet and ACP.

7.3 Method

We establish preliminaries about the rigid alignment problem and related algorithms in §7.3.1; then, we present PRNet in §7.3.2. For ease of comparison to previous work, we use the same notation as chapter 6.

7.3.1 Preliminaries: Registration, ICP, and DCP

Consider two point clouds $\mathcal{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_i, \dots, \boldsymbol{x}_N\} \subset \mathbb{R}^3$ and $\mathcal{Y} = \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_j, \dots, \boldsymbol{y}_M\} \subset \mathbb{R}^3$. The basic task in rigid registration is to find a rotation $\boldsymbol{R}_{\mathcal{XY}}$ and translation $\boldsymbol{t}_{\mathcal{XY}}$ that rigidly align $\mathcal{X}$ to $\mathcal{Y}$. When $M = N$, ICP and its peers approach this task by minimizing the objective function

$$E(\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}, m) = \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{R}_{\mathcal{XY}} \boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{XY}} - \boldsymbol{y}_{m(\boldsymbol{x}_i)}\|^2. \tag{7.1}$$

Here, the rigid transformation is defined by a pair $[\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}]$, where $\boldsymbol{R}_{\mathcal{XY}} \in \mathrm{SO}(3)$ and $\boldsymbol{t}_{\mathcal{XY}} \in \mathbb{R}^3$; m maps from points in $\mathcal{X}$ to points in $\mathcal{Y}$. Assuming m is fixed, the alignment in (7.1) is given in closed-form by

$$\boldsymbol{R}_{\mathcal{XY}} = \boldsymbol{V}\boldsymbol{U}^\top \qquad \text{and} \qquad \boldsymbol{t}_{\mathcal{XY}} = -\boldsymbol{R}_{\mathcal{XY}} \overline{\boldsymbol{x}} + \overline{\boldsymbol{y}}, \tag{7.2}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are obtained using the singular value decomposition (SVD) $\boldsymbol{H} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$, with $\boldsymbol{H} = \sum_{i=1}^{N} (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{y}_{m(\boldsymbol{x}_i)} - \overline{\boldsymbol{y}})^\top$. In this expression, centroids of $\mathcal{X}$ and

$\mathcal{Y}$ are defined as $\overline{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i$ and $\overline{\boldsymbol{y}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{y}_{m(x_i)}$, respectively.

We can understand ICP and the more recent learning-based DCP method [212] as providing different choices of m:

Iterative Closest Point. ICP chooses m to minimize (7.1) with $[\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}]$ fixed, yielding:

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \arg\min_j \ \|\boldsymbol{R}_{\mathcal{XY}}\boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{XY}} - \boldsymbol{y}_j\|_2 \tag{7.3}$$

ICP approaches a fixed point by alternating between (7.2) and (7.3); each step decreases the objective (7.1). Since (7.1) is non-convex, however, there is no guarantee that ICP reaches a global optimum.

Deep Closest Point. DCP uses deep networks to learn m. In this method, $\mathcal{X}$ and $\mathcal{Y}$ are embedded using learned functions $\mathcal{F}_{\mathbf{X}}$ and $\mathcal{F}_{\mathcal{Y}}$ defined by a Siamese DGCNN [215]; these lifted point clouds are optionally contextualized by a Transformer module [200], yielding embeddings $\Phi_{\mathcal{X}}$ and $\Phi_{\mathcal{Y}}$. The mapping m is then

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \text{softmax}(\Phi_{\mathcal{Y}}\Phi_{\boldsymbol{x}_i}^{\top}). \tag{7.4}$$

This formula is applied in one shot followed by (7.2) to obtain the rigid alignment. The loss used to train this pipeline is mean-squared error (MSE) between ground-truth rigid motion from synthetically-rotated point clouds and prediction; the network is trained end-to-end.

7.3.2 Partial Registration Network

DCP is a one-shot algorithm, in that a single pass through the network determines the output for each prediction task. Analogously to ICP, PRNet is designed to be *iterative*; multiple passes of a point cloud through PRNet refine the alignment. The steps of PRNet, illustrated in Figure 7-1, are as follows:

- take as input point clouds $\mathcal{X}$ and $\mathcal{Y}$;

- detect keypoints of $\mathcal{X}$ and $\mathcal{Y}$;
- predict a mapping from keypoints of $\mathcal{X}$ to keypoints of $\mathcal{Y}$;
- predict a rigid transformation $[\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}]$ aligning $\mathcal{X}$ to $\mathcal{Y}$ based on the keypoints and map;
- transform $\mathcal{X}$ using the obtained transformation;
- return to 7.3.2 using the pair $(\boldsymbol{R}_{\mathcal{XY}}\mathbf{X} + \boldsymbol{t}_{\mathcal{XY}}, \mathcal{Y})$ as input.

When predicting a mapping from keypoints in $\mathcal{X}$ to keypoints in $\mathcal{Y}$, PRNet uses Gumbel–Softmax [74] to sample a matching matrix, which is sharper than (7.4) and approximately differentiable. It has a value network to predict a temperature for Gumbel–Softmax, so that the whole framework can be seen as an actor-critic method. We present details of and justifications behind the design below.

Notation. Denote by $\mathcal{X}^p = \{\boldsymbol{x}_1^p, \ldots, \boldsymbol{x}_i^p, \ldots, \boldsymbol{x}_N^p\}$ the rigid motion of $\mathcal{X}$ to align to $\mathcal{Y}$ after p applications of PRNet; $\mathcal{X}^1$ and $\mathcal{Y}^1$ are initial input shapes. We will use $[\boldsymbol{R}_{\mathcal{XY}}^p, \boldsymbol{t}_{\mathcal{XY}}^p]$ to denote the p-th rigid motion predicted by PRNet for the input pair $(\mathcal{X}, \mathcal{Y})$.

Since our training pairs are synthetically generated, before applying PRNet we know the ground-truth $[\boldsymbol{R}_{\mathcal{XY}}^*, \boldsymbol{t}_{\mathcal{XY}}^*]$ aligning $\mathcal{X}$ to $\mathcal{Y}$. From these values, during training we can compute "local" ground-truth $[\boldsymbol{R}_{\mathcal{XY}}^{p*}, \boldsymbol{t}_{\mathcal{XY}}^{p*}]$ on-the-fly, which maps the current $(\mathcal{X}^p, \mathcal{Y})$ to the best alignment:

$$\boldsymbol{R}_{\mathcal{XY}}^{p*} = \boldsymbol{R}_{\mathcal{XY}}^* \boldsymbol{R}_{\mathcal{XY}}^{1\ldots p\top} \qquad \text{and} \qquad \boldsymbol{t}_{\mathcal{XY}}^{p*} = \boldsymbol{t}_{\mathcal{XY}}^* - \boldsymbol{R}_{\mathcal{XY}}^{p*} \boldsymbol{t}_{\mathcal{XY}}^{1\ldots p}, \tag{7.5}$$

where

$$\boldsymbol{R}_{\mathcal{XY}}^{1\ldots p} = \boldsymbol{R}_{\mathcal{XY}}^{p-1} \ldots \boldsymbol{R}_{\mathcal{XY}}^{1} \qquad \text{and} \qquad \boldsymbol{t}_{\mathcal{XY}}^{1\ldots p} = \boldsymbol{R}_{\mathcal{XY}}^{p-1} \boldsymbol{t}_{\mathcal{XY}}^{1\ldots p-1} + \boldsymbol{t}_{\mathcal{XY}}^{p-1}. \tag{7.6}$$

We use m^p to denote the mapping function in p-th step.

Synthesizing the notation above, $\mathcal{X}^p$ is given by

$$\boldsymbol{x}_i^p = \boldsymbol{R}_{\mathcal{XY}}^{p-1} \boldsymbol{x}_i^{p-1} + \boldsymbol{t}_{\mathcal{XY}}^{p-1} \tag{7.7}$$

where

$$\boldsymbol{R}^p_{\mathcal{XY}} = \boldsymbol{V}^p \boldsymbol{U}^{p\top} \qquad \text{and} \qquad \boldsymbol{t}^p_{\mathcal{XY}} = -\boldsymbol{R}^p_{\mathcal{XY}} \overline{\boldsymbol{x}}^p + \overline{\boldsymbol{y}}. \tag{7.8}$$

In this equation, $\boldsymbol{U}^p$ and $\boldsymbol{V}^p$ are computed using (7.2) from $\mathcal{X}^p$, $\mathcal{Y}^p$, and m^p.

Keypoint detection. For partial-to-partial registration, usually $N \neq M$ and only subsets of $\mathcal{X}$ and $\mathcal{Y}$ match to one another. To detect these mutually-shared patches, we design a simple yet efficient keypoint detection module based on the observation that the L^2 norms of features tend to indicate whether a point is important.

Using $\mathcal{X}^p_k$ and $\mathcal{Y}^p_k$ to denote the k keypoints for $\mathcal{X}^p$ and $\mathcal{Y}^p$, we take

$$\begin{aligned} \mathcal{X}^p_k &= \mathcal{X}^p(\text{topk}(\|\Phi^p_{\boldsymbol{x}_1}\|_2, \ldots, \|\Phi^p_{\boldsymbol{x}_i}\|_2, \ldots, \|\Phi^p_{\boldsymbol{x}_N}\|_2)) \\ \mathcal{Y}^p_k &= \mathcal{Y}^p(\text{topk}(\|\Phi^p_{\boldsymbol{y}_1}\|_2, \ldots, \|\Phi^p_{\boldsymbol{y}_i}\|_2, \ldots, \|\Phi^p_{\boldsymbol{y}_M}\|_2)) \end{aligned} \tag{7.9}$$

where $\text{topk}(\cdot)$ extracts the indices of the k largest elements of the given input. Here, Φ denotes embeddings learned by DGCNN and Transformer.

By aligning only the keypoints, we remove irrelevant points from the two input clouds that are not shared in the partial correspondence. In particular, we can now solve the Procrustes problem that matches keypoints of $\mathcal{X}$ and $\mathcal{Y}$. We show in §7.4.3 that although we do not provide explicit supervision, PRNet still learns how to detect keypoints reasonably.

Gumbel–softmax sampler. One key observation in ICP and DCP is that (7.3) usually is not differentiable with respect to the map m but by definition yields a sharp correspondence between the points in $\mathcal{X}$ and the points in $\mathcal{Y}$. In contrast, the smooth function (7.4) in DCP is differentiable, but in exchange for this differentiability the mapping is blurred. We desire the best of both worlds: A potentially sharp mapping function that admits backpropagation.

To that end, we use Gumbel–Softmax [74] to sample a matching matrix. Using a straight-through gradient estimator, this module is approximately differentiable. In particular, the Gumbel–Softmax mapping function is given by

$$m^p(\boldsymbol{x}_i, \mathcal{Y}) = \text{one hot}\left[\arg\max_j \text{softmax}(\Phi^p_{\mathcal{Y}} \Phi^{p\top}_{\boldsymbol{x}_i} + g_{ij})\right], \tag{7.10}$$

where $(g_{i1}, \ldots, g_{ij}, \ldots, g_{iN})$ are i.i.d. samples drawn from Gumbel(0, 1). The map in (7.10) is not differentiable due to the discontinuity of arg max, but the straight-through gradient estimator [12] yields (biased) subgradient estimates with low variance. Following their methodology, on backward evaluation of the computational graph, we use (7.4) to compute $\frac{\partial L}{\partial \Phi_*^p}$, ignoring the one hot operator and the arg max term.

Actor–critic closest point (ACP). The mapping functions (7.4) and (7.10) have fixed "temperatures," that is, there is no control over the sharpness of the mapping matrix m^p. In PRNet, we wish to adapt the sharpness of the map based on the alignment of the two shapes. In particular, for low values of p (the initial iterations of alignment) we may satisfied with high-entropy approximate matchings that obtain a coarse alignment; later during iterative evaluations, we can sharpen the map to align individual pairs of points.

To make this intuition compatible with PRNet's learning-based architecture, we add a parameter λ to (7.10) to yield a generalized Gumbel–Softmax matching matrix:

$$m^p(\boldsymbol{x}_i, \mathcal{Y}) = \text{one hot}\left[\arg\max_j \text{softmax}\left(\frac{\Phi_{\mathcal{Y}}^p \Phi_{\boldsymbol{x}_i}^{p\top} + g_{ij}}{\lambda}\right)\right], \tag{7.11}$$

When λ is large, the map matrix m^p is smoothed out; as $\lambda \to 0$ the map approaches a binary matrix.

It is difficult to choose a single λ that suffices for all $(\mathcal{X}, \mathcal{Y})$ pairs; rather, we wish λ to be chosen adaptively and automatically to extract the best alignment for each pair of point clouds. Hence, we use a small network Θ to predict λ based on global features $\Psi_{\mathcal{X}}^p$ and $\Psi_{\mathcal{Y}}^p$ aggregated from $\Phi_{\mathcal{X}}^p$ and $\Phi_{\mathcal{Y}}^p$ channel-wise by global pooling (averaging). In particular, we take $\lambda = \Theta(\Psi_{\mathcal{X}}^p, \Psi_{\mathcal{Y}}^p)$, where $\Psi_{\mathcal{X}}^p = \text{avg}_i \Phi_{\boldsymbol{x}_i}^p$ and $\Psi_{\mathcal{Y}}^p = \text{avg}_i \Phi_{\boldsymbol{y}_i}^p$. In the parlance of reinforcement learning, this choice can be seen as a simplified version of actor-critic method. $\Phi_{\mathcal{X}}^p$ and $\Phi_{\mathcal{Y}}^p$ are learned jointly with DGCNN [215] and Transformer [200]; then an actor head outputs a rigid motion, where (7.11) uses the λ predicted from a critic head.

Loss function. The final loss L is the summation of several terms L_p, indexed by

the number p of passes through PRNet for the input pair. L_p consists of three terms: a rigid motion loss L_p^m, a cycle consistency loss L_p^c, and a global feature alignment loss L_g. We also introduce a discount factor $\gamma < 1$ to promote alignment within the first few passes through PRNet; during training we pass each input pair through PRNet P times.

Combining the terms above, we have

$$L = \sum_{p=1}^{P} \gamma^{p-1} L_p, \qquad \text{where} \qquad L_p = L_p^m + \alpha L_p^c + \beta L_g^p. \tag{7.12}$$

The rigid motion loss L_p^m is

$$L_p^m = \|\boldsymbol{R}_{\mathcal{XY}}^{p\top} \boldsymbol{R}_{\mathcal{XY}}^{p*} - I\|^2 + \|\boldsymbol{t}_{\mathcal{XY}}^p - \boldsymbol{t}_{\mathcal{XY}}^{p*}\|^2. \tag{7.13}$$

Equation (7.5) gives the "localized" ground truth values for $\boldsymbol{R}_{\mathcal{XY}}^{p*}, \boldsymbol{t}_{\mathcal{XY}}^{p*}$. Denoting the rigid motion from $\mathcal{Y}$ to $\mathcal{X}$ in step p as $[\boldsymbol{R}_{\mathcal{YX}}^p, \boldsymbol{t}_{\mathcal{YX}}^p]$, the cycle consistency loss is

$$L_p^c = \|\boldsymbol{R}_{\mathcal{XY}}^p \boldsymbol{R}_{\mathcal{YX}}^p - I\|^2 + \|\boldsymbol{t}_{\mathcal{XY}}^p - \boldsymbol{t}_{\mathcal{YX}}^p\|^2. \tag{7.14}$$

Our last loss term is a global feature alignment loss, which enforces alignment of global features $\Psi_{\mathcal{X}}^p$ and $\Psi_{\mathcal{Y}}^p$. Mathematically, the global feature alignment loss is

$$L_g^p = \|\Psi_{\mathbf{X}}^p - \Psi_{\mathcal{Y}}^p\|. \tag{7.15}$$

This global feature alignment loss also provides signal for determining λ. When two shapes are close in global feature space, λ should be small, yielding a sharp matching matrix; when two shapes are far from each other, λ increases and the map is blurry.

Model	MSE($\boldsymbol{R}$) ↓	RMSE($\boldsymbol{R}$) ↓	MAE($\boldsymbol{R}$) ↓	R^2($\boldsymbol{R}$) ↑	MSE($\boldsymbol{t}$) ↓	RMSE($\boldsymbol{t}$) ↓	MAE($\boldsymbol{t}$) ↓	R^2($\boldsymbol{t}$) ↑
ICP	1134.552	33.683	25.045	-5.696	0.0856	0.293	0.250	-0.037
Go-ICP [227]	195.985	13.999	3.165	-0.157	0.0011	0.033	0.012	0.987
FGR [244]	126.288	11.238	2.832	0.256	0.0009	0.030	0.008	0.989
PointNetLK [51]	280.044	16.735	7.550	-0.654	0.0020	0.045	0.025	0.975
DCP-v2 [212]	45.005	6.709	4.448	0.732	0.0007	0.027	0.020	0.991
PRNet (Ours)	**10.235**	**3.199257**	**1.454**	**0.939**	**0.0003**	**0.016**	**0.010**	**0.997**

Table 7.1: Test on unseen point clouds

Model	MSE($\boldsymbol{R}$) ↓	RMSE($\boldsymbol{R}$) ↓	MAE($\boldsymbol{R}$) ↓	R^2($\boldsymbol{R}$) ↑	MSE($\boldsymbol{t}$) ↓	RMSE($\boldsymbol{t}$) ↓	MAE($\boldsymbol{t}$) ↓	R^2($\boldsymbol{t}$) ↑
ICP	1217.618	34.894	25.455	-6.253	0.086	0.293	0.251	-0.038
Go-ICP [227]	157.072	12.533	2.940	0.063	0.0009	0.031	0.010	0.989
FGR [244]	98.635	9.932	**1.952**	0.414	0.0014	0.038	0.007	0.983
PointNetLK [51]	526.401	22.943	9.655	-2.137	0.0037	0.061	0.033	0.955
DCP-v2 [212]	95.431	9.769	6.954	0.427	0.0010	0.034	0.025	0.986
PRNet (Ours)	**24.857**	**4.986**	2.329	**0.850**	**0.0004**	**0.021**	**0.015**	**0.995**
PRNet (Ours*)	**15.624**	**3.953**	**1.712**	**0.907**	**0.0003**	**0.017**	**0.011**	**0.996**

Table 7.2: Test on unseen categories. PRNet (Ours*) denotes the model trained on ShapeNetCore and tested on ModelNet40 held-out categories. Others are trained on first 20 ModelNet40 categories and tested on ModelNet40 held-out categories.

7.4 Experiments

Our experiments are divided into four parts. First, we show performance of PRNet on a partial-to-partial registration task on synthetic data in §7.4.1. Then, we show PRNet can generalize to real data in §7.4.2. Third, we visualize the keypoints and correspondences predicted by PRNet in §7.4.3. Finally, we show a linear SVM trained on representations learned by PRNet can achieve comparable results to supervised learning methods in §7.4.4.

7.4.1 Partial-to-Partial Registration on ModelNet40

We evaluate partial-to-partial registration on ModelNet40 [218]. There are 12,311 CAD models spanning 40 object categories, split to 9,843 for training and 2,468 for testing. Point clouds are sampled from the CAD models by farthest-point sampling on the surface. During training, a point cloud with 1024 points $\mathcal{X}$ is sampled. Along each axis, we randomly draw a rigid transformation; the rotation along each axis is sampled in $[0, 45^\circ]$ and translation is in $[-0.5, 0.5]$. We apply the rigid transformation

Model	MSE($\boldsymbol{R}$) ↓	RMSE($\boldsymbol{R}$) ↓	MAE($\boldsymbol{R}$) ↓	R^2($\boldsymbol{R}$) ↑	MSE($\boldsymbol{t}$) ↓	RMSE($\boldsymbol{t}$) ↓	MAE($\boldsymbol{t}$) ↓	R^2($\boldsymbol{t}$) ↑
ICP	1229.670	35.067	25.564	-6.252	0.0860	0.294	0.250	-0.045
Go-ICP [227]	150.320	12.261	2.845	0.112	0.0008	0.028	0.029	0.991
FGR [244]	764.671	27.653	13.794	-3.491	0.0048	0.070	0.039	0.941
PointNetLK [51]	397.575	19.939	9.076	-1.343	0.0032	0.0572	0.032	0.960
DCP-v2 [212]	47.378	6.883	4.534	0.718	0.0008	0.028	0.021	0.991
PRNet (Ours)	**18.691**	**4.323**	**2.051**	**0.889**	**0.0003**	**0.017**	**0.012**	**0.995**

Table 7.3: Test on unseen point clouds with Gaussian noise

to $\mathcal{X}$, leading to $\mathcal{Y}$. We simulate partial scans of $\mathcal{X}$ and $\mathcal{Y}$ by randomly placing a point in space and computing its 768 nearest neighbors in $\mathcal{X}$ and $\mathcal{Y}$ respectively.

We measure mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and coefficient of determination (R^2). Angular measurements are in units of degrees. MSE, RMSE and MAE should be zero while R^2 should be one if the rigid alignment is perfect. We compare our model to ICP, Go-ICP [227], Fast Global Registration (FGR) [244], and DCP [212].

Figure 7-1 shows the architecture of ACP. We use DGCNN with 5 dynamic *EdgeConv* layers and a Transformer to learn co-contextual representations of $\mathcal{X}$ and $\mathcal{Y}$. The number of filters in each layer of DGCNN are $(64, 64, 128, 256, 512)$. In the Transformer, only one encoder and one decoder with 4-head attention are used. The embedding dimension is 1024. We train the network for 100 epochs using Adam [80]. The initial learning rate is 0.001 and is divided by 10 at epochs 30, 60, and 80.

Partial-to-partial registration on unseen objects. We first evaluate on the ModelNet40 train/test split. We train on 9,843 training objects and test on 2,468 testing objects. Table 7.1 shows performance. Our method outperforms its counterparts in all metrics.

Partial-to-partial registration on unseen categories. We follow the same testing protocol as [212] to compare the generalizability of different models. ModelNet40 is split evenly by category into training and testing sets. PRNet and DCP are trained on the first 20 categories, and then all methods are tested on the held-out categories. Table 7.2 shows PRNet behaves more strongly than others. To further test generalizability, we train it on ShapeNetCore dataset [27] and test on ModelNet40

Model	z/z ↑	SO(3)/SO(3) ↑	input size
PointNet [141]	89.2	83.6	2048×3
PointNet++ [143]	89.3	85.0	1024×3
DGCNN [215]	92.2	87.2	1024×3
VoxNet [118]	83.0	73.0	30^3
SubVolSup [142]	88.5	82.7	30^3
SubVolSup MO [142]	89.5	85.0	20×30^3
MVCNN 12x [178]	89.5	77.6	12×224^2
MVCNN 80x [178]	90.2	86.0	80×224^2
RotationNet 20x [77]	**92.4**	80.0	20×224^2
Spherical CNNs [42]	88.9	**86.9**	2×64^2
PRNet (Ours)	**85.2**	**80.5**	1024×3

Table 7.4: ModelNet40: transfer learning

held-out categories. ShapeNetCore has 57,448 objects, and we do the same preprocessing as on ModelNet40. The last row in Table 7.2, denoted as PRNet (Ours*), surprisingly shows PRNet performs much better than when trained on ModelNet40. This supports the intuition that data-driven approaches work better with more data.

Partial-to-partial registration on unseen objects with Gaussian noise. We further test robustness to noise. The same preprocessing is done as in the first experiment, except that noise independently sampled from $\mathcal{N}(0, 0.01)$ and clipped to $[-0.05, 0.05]$ is added to each point. As in Table 7.3, learning-based methods, including DCP and PRNet, are more robust. In particular, PRNet exhibits stronger performance and is even comparable to the noise-free version in Table 7.1.

7.4.2 Partial-to-Partial on Real Data

We test our model on the Stanford Bunny dataset [197]. Since the dataset only has 10 real scans, we fine tune the model used in Table 7.1 for 10 epochs with learning rate 0.0001. For each scan, we generate 100 training examples by randomly transforming the scan in the same way as we do in §7.4.1. This training procedure can be viewed as inference time fine-tuning, in contrast to optimization-based methods that perform

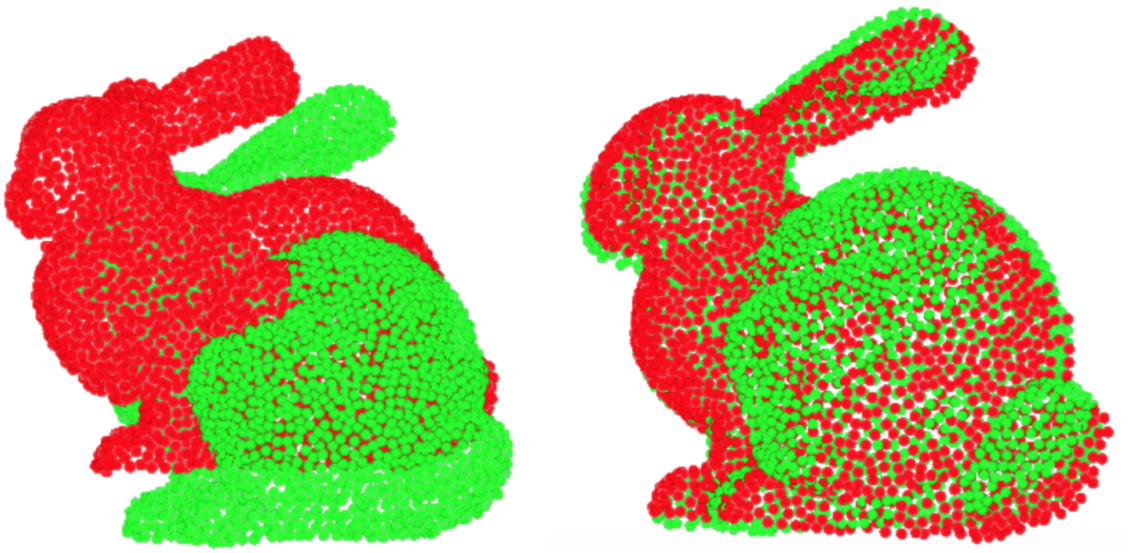


Figure 7-2: Left: Input partial point clouds. Right: Transformed partial point clouds.

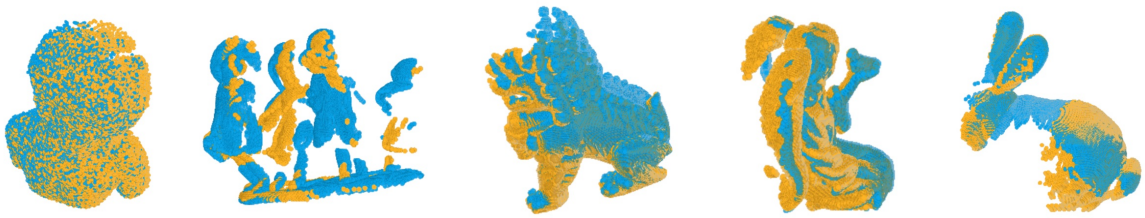
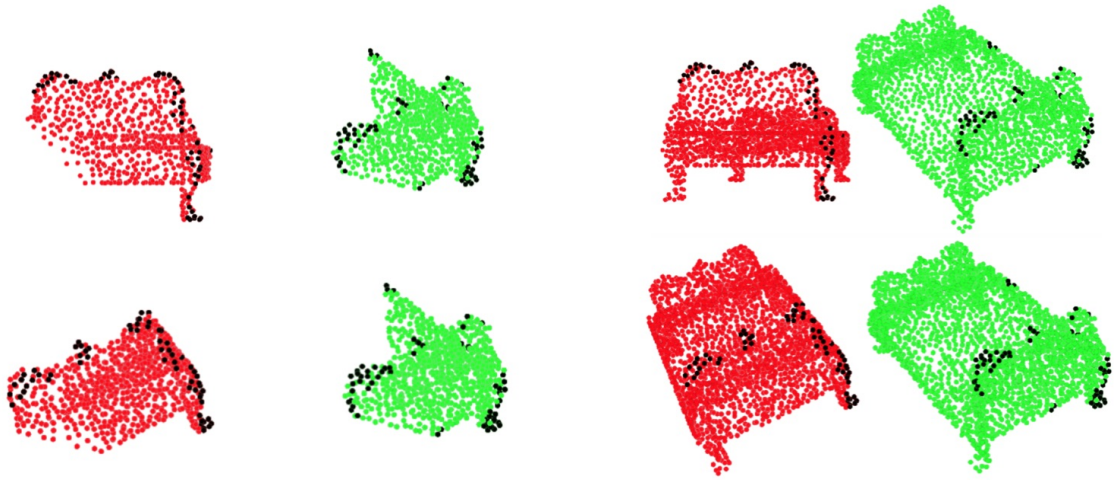


Figure 7-3: More examples on The Stanford 3D Scanning Repository [196].

one-time inference for each test case. Figure 7-2 shows the results. We further test our model on more scans from Stanford 3D Scanning Repository [196] using a similar methodology; Figure 7-3 shows the registration results.

7.4.3 Keypoints and Correspondences

We visualize keypoints on several objects in Figure 7-4 and correspondences in Figure 7-6. The model detects keypoints and correspondences on partially observable objects. We overlay the keypoints on top of the fully observable objects. Also, as shown in Figure 7-5, the keypoints are consistent across different views.



(a) Partial Point Clouds (b) Full Point Clouds

Figure 7-4: Keypoint detection of a pair of beds. Point clouds in red are $\mathcal{X}$ and point clouds in green are $\mathcal{Y}$. Points in black are keypoints detected by PRNet. Point clouds in the first row are in the original pose while point clouds in the second row are transformed using the rigid transformation predicted by PRNet. (a) PRNet takes as input partial point clouds. (b) The obtained rigid transformation is applied to full point clouds for better visualization.

Figure 7-5: Keypoint detection with different partial scans. We show the same pair of $\mathcal{X}$ and $\mathcal{Y}$ with different views. The keypoints are data-dependent and consistent across different views.

7.4.4 Transfer to Classification

Representations learned by PRNet can be transferred to object recognition. We use the model trained on ShapeNetCore in §7.4.1 and train a linear SVM on top of the embeddings predicted by DGCNN. In Table 7.4, we show classification accuracy on ModelNet40. z/z means the model is trained when only azimuthal rotations are

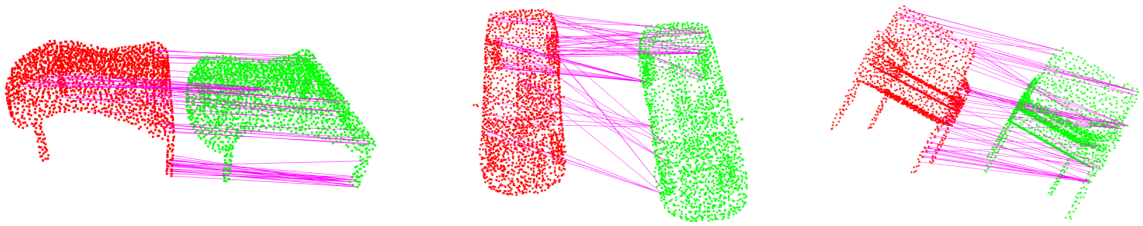


Figure 7-6: Correspondences for pairs of objects.

Method	MAE($\boldsymbol{R}$) ↓	R^2($\boldsymbol{R}$) ↑	MAE($\boldsymbol{t}$) ↓	R^2($\boldsymbol{t}$) ↑
Random sampling	1.689	0.927	0.011	**0.997**
Closeness to other points	2.109	0.861	0.013	0.995
L^2 Norm	**1.454**	**0.939**	**0.010**	**0.997**

(a) Different keypoint detection methods.

Model	MAE($\boldsymbol{R}$) ↓	R^2($\boldsymbol{R}$) ↑	MAE($\boldsymbol{t}$) ↓	R^2($\boldsymbol{t}$) ↑
ICP	25.165	-5.860	0.250	-0.045
Go-ICP	2.336	0.308	0.007	0.994
FGR	2.088	0.393	**0.003**	0.999
PointNetLK	3.478	0.051	0.005	0.994
DCP	2.777	0.887	0.009	0.998
PRNet (Ours)	**0.960**	**0.979**	0.006	**1.000**

(b) Experiments on full point clouds.

Discount Factor λ	MAE($\boldsymbol{R}$) ↓	R^2($\boldsymbol{R}$) ↑	MAE($\boldsymbol{t}$) ↓	R^2($\boldsymbol{t}$) ↑
0.5	1.921	0.917	0.014	0.995
0.7	1.998	0.884	0.014	0.995
0.9	**1.454**	**0.939**	**0.010**	**0.997**
0.99	1.732	0.915	0.012	0.996

(c) Different discount factors (λ).

Different k	MAE($\boldsymbol{R}$) ↓	R^2($\boldsymbol{R}$) ↑	MAE($\boldsymbol{t}$) ↓	R^2($\boldsymbol{t}$) ↑
16	27.843	-14.176	0.136	0.326
32	8.293	-1.848	0.048	0.892
64	3.129	0.563	0.024	0.979
128	2.007	0.879	0.016	0.991
256	1.601	0.932	0.012	0.996
384	1.508	0.934	0.011	**0.997**
512	**1.454**	**0.939**	**0.010**	**0.997**

(d) Different number of keypoints (k).

Data Missing Ratio	MAE($\boldsymbol{R}$) ↓	R^2($\boldsymbol{R}$) ↑	MAE($\boldsymbol{t}$) ↓	R^2($\boldsymbol{t}$) ↑
75%	6.447	0.028	0.042	0.921
50%	3.939	0.623	0.0288	0.969
25%	**1.454**	**0.939**	**0.010**	**0.997**

(e) Data missing ratio.

Data Noise	MAE($\boldsymbol{R}$) ↓	R^2($\boldsymbol{R}$) ↑	MAE($\boldsymbol{t}$) ↓	R^2($\boldsymbol{t}$) ↑
$\mathbb{N}(0, 0.01^2)$	**2.051**	**0.889**	**0.012**	**0.995**
$\mathbb{N}(0, 0.1^2)$	5.013	0.617	0.020	0.991
$\mathbb{N}(0, 0.5^2)$	21.129	-2.830	0.064	0.917

(f) Data noise.

Table 7.5: Ablation studies.

present on both during and testing. SO(3)/SO(3) means the object is moved with a random motion in SO(3) during both training and testing. Although other methods are supervised, ours achieves comparable performance.

7.5 Ablation Study

To understand the effectiveness of each part, we conduct additional experiments in Table 7.5; to save space, we only show MAE and R^2. (a) First, we consider alternatives to keypoint selection: in the first alternative, the two sets of keypoints are chosen independently and randomly on the two surfaces ($\mathcal{X}$ and $\mathcal{Y}$); in the second alternative,

# points	ICP	Go-ICP	FGR	PointNetLK	DCP	PRNet
512	0.134	14.763	0.230	0.049	0.014	0.042
1024	0.170	14.853	0.250	0.061	0.024	0.073
2048	0.242	14.929	0.248	0.069	0.058	0.152

Table 7.6: Inference time (in seconds).

we use *centrality* to choose keypoints, keeping the k points whose average distance (in feature space) to the rest in the point cloud is minimal. Empirically, the L^2 norm used in our pipeline to select keypoints outperforms others. (b) Second, we compare our method to others on full point clouds. In this experiment, 768 points are sampled from each point cloud to cover the full shape using farthest-point sampling. In the full point cloud setting, PRNet still outperforms others. (c) Third, we verify our choice of discount factor λ; small large discount factors encourage alignment within the first few passes through PRNet while large discount factors promote longer-term return. (d) Fourth, we test the choice of number of keypoints: the model achieves surprisingly good performance even with 64 keypoints, but performance drops significantly when $k < 32$. (e) Fifth, we test its robustness to missing data. The missing data ratio in original partial-to-partial experiment is 25%; we further test with 50% and 75%. This test shows that with 75% points missing, the method still achieves reasonable performance, even compared to other methods tested with only 25% points missing. (f) Finally, we test the model robustness to noise level. Noise is sampled from $\mathbb{N}(0, \sigma^2)$. The model is trained with $\sigma = 0.01$ and tested with $\sigma \in [0.01, 0.1, 0.5]$. Even with $\sigma = 0.1$, the model still performs reasonably well.

Alternatives to Gumbel-Softmax. Methods to tackle non-differentiability usually fall into two categories: REINFORCE and Gumbel-Softmax. REINFORCE produces unbiased high-variance gradient estimation while Gumbel-Softmax produces biased gradients with low variance. Empirically, we tried vanilla REINFORCE to estimate the gradients of the matching function; due to its instability, the training did not converge. Studying unbiased low-variance gradient estimation is extremely valuable to reinforcement learning and/or discrete optimization, but introducing complicated

gradient estimator is beyond the scope of this paper.

Efficiency. We benchmark the inference time of different methods on a desktop computer with an Intel 16-core CPU, an Nvidia GTX 1080 Ti GPU, and 128G memory. Table 7.6 shows learning based methods (on GPUs) are faster than non-learning based counterparts (on CPUs). PRNet is on a par with PointNetLK while being slower than DCP.

7.6 Conclusion

PRNet tackles a general partial-to-partial registration problem, leveraging self-supervised learning to learn geometric priors directly from data. The success of PRNet verifies the sensibility of applying learning to partial matching as well as the specific choice of Gumbel–Softmax, which we hope can inspire additional work linking discrete optimization to deep learning. PRNet is also a reinforcement learning-like framework; this connection between registration and reinforcement learning may provide inspiration for additional interdisciplinary research related to rigid/non-rigid registration. These two chapters collectively show that in addition to high-level reasoning tasks, deep learning also benefits low-level geometric registration problems.

Our experiments suggest several avenues for future work. For example, as shown in Figure 7-6, the matchings computed by PRNet are not bijective, evident e.g. in the point clouds of cars and chairs. One possible extension of our work to address this issue is to use Gumbel–Sinkhorn [120] to encourage bijectivity. Improving the efficiency of PRNet when applied to real scans also will be extremely valuable. As described in §7.4.2, PRNet currently requires inference-time fine-tuning on real scans to learn useful data-dependent representations; this makes PRNet slow during inference. Seeking universal representations that generalize over broader sets of registration tasks will improve the speed and generalizability of learning-based registration. Another possibility for future work is to improve the scalability of PRNet to deal with large-scale real scans captured by LiDAR.

Finally, we hope to find more applications of PRNet beyond the use cases we have

shown in the paper. A key direction bridging PRNet to applications will involve incorporating our method into SLAM or structure-from-motion can demonstrate its value for robotics applications and robustness to realistic species of noise. Additionally, we can test the effectiveness of PRNet for registration problems in medical imaging and/or high-energy particle physics.

Chapter 8

Conclusion

Learning from 3D data is a long-standing problem in visual computing. In this thesis, we proposed several methods to tackle perception problems involving 3D data. The critical insight behind all these projects is that we should identify unique properties of each 3D dataset and of each task; these properties suggest inductive biases that should be incorporated into architectural design. To summarize, this thesis makes the following contributions:

- DGCNN [215] laid a foundation for graph-based point cloud processing. This method achieves state-of-the-art performance on multiple benchmarks and provides a new perspective on deep learning from point clouds. Beyond computer vision, it has been even widely adopted in drug discovery, disease prediction, and high-energy physics.

- In the high-level regime, we studied 3D object detection for autonomous driving. Pillar-based Object Detection [209] addresses the feature learning problem and introduces an efficient feature extractor. Object DGCNN [214] leverages DGCNN to model object relations in 3D object detection and enables a post-processing free architecture. DETR3D [211] generalizes Object DGCNN to tackle object detection from multi-view images.

- In the low-level regime, I proposed a novel paradigm to address rigid motion registration. In contrast to prior works, DCP [212] and PRNet [213] combine deep

learning with Iterative Closest Point. This algorithm has become the standard approach to point cloud registration and inspires many follow-up works. Potentially, this architecture is applicable to Simultaneous Localization and Mapping (SLAM) to enable robot localization.

- We also made progress in self-supervised learning. PRNet leverages registration as supervision to learn representations. PRNet is capable of transferring the low-level features to high-level semantic reasoning tasks. PRNet is the first architecture of its kind to learn features with self-supervision.

Learning from 3D data remains an unsolved problem. This thesis also suggests several avenues for future work:

- **Multi-modal perception.** Humans can perceive the world through vision, touch, and sound. Multi-modal signals play important roles in human recognition. Therefore, learning from multi-modal signals provides complementary information. There are several immediate ways to use multi-modal signals. For example, we can plug different modalities into the DETR3D architecture to improve object detection performance. We can further leverage multi-modal correspondences to enable self-supervised learning.

- **Combining computer graphics with machine learning.** Simulating data with computer graphics techniques is another way to provide annotations for 3D deep learning. Neural rendering has achieved significant progress in the past two years. However, generating point clouds and triangle meshes remains an open and challenging task. One solution to the question is to rethink the roles of computer graphics and machine learning. I hope to build better generative models that can produce photorealistic data to enable sim-to-real transfer learning.

- **End-to-end embodied agents.** Current robotic systems still adopt independently modular designs. Errors from the control module are not propagated back to the perception module. Combining 3D deep learning with planning

and control will further simplify the current robotic pipelines and enable full-stack embodied agents. These agents can explore surrounding environments instinctually and learn with self-supervision.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Gabriel Agamennoni, Simone Fontana, Roland Siegwart, and Domenico Sorrenti. Point clouds registration with probabilistic data association. In *Proceedings of The International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[3] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Transaction on Graphics (TOG)*, 2018.

[5] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *The International Conference on Computer Vision (ICCV) Workshops*, 2011.

[6] Yusuf Aytar, Lluis Castrejon, Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Cross-modal scene networks. *CoRR*, abs/1610.09003, 2016.

[7] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, 2016.

[8] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. In *International Conference on Learning Representations, ICLR*, 2017.

[9] Ivan Barabanau, Alexey Artemov, Evgeny Burnaev, and Vyacheslav Murashkin. Monocular 3d object detection via geometric reasoning on keypoints. *arXiv:1905.05618*, 2019.

[10] Deniz Beker, Hiroharu Kato, Mihai Adrian Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Monocular differentiable rendering for self-supervised 3d object detection. *arXiv:2009.14524*, 2020.

[11] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2001.

[12] Yoshua Bengio. Estimating or propagating gradients through stochastic neurons. *CoRR*, abs/1305.2982, 2013.

[13] Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, February 1992.

[14] Silvia Biasotti, Andrea Cerri, A Bronstein, and M Bronstein. Recent trends, applications, and perspectives in 3d shape similarity assessment. *Computer Graphics Forum*, 35(6):87–119, 2016.

[15] Davide Boscaini, Jonathan Masci, Simone Melzi, Michael M Bronstein, Umberto Castellani, and Pierre Vandergheynst. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. *Computer Graphics Forum*, 34(5):13–23, 2015.

[16] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[17] Sofien Bouaziz, Andrea Tagliasacchi, and Mark Pauly. Sparse iterative closest point. In *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing*, SGP '13, pages 113–123, Aire-la-Ville, Switzerland, Switzerland, 2013. Eurographics Association.

[18] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[19] Michael M Bronstein and Iasonas Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[20] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv:1312.6203*, 2013.

[21] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '05, pages 60–65, Washington, DC, USA, 2005. IEEE Computer Society.

[22] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *SIGKDD*, 2006.

[23] 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:1903.11027*, 2019.

[24] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into high quality object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[25] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.

[26] Lluis Castrejon, Yusuf Aytar, Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Learning aligned cross-modal representations from weakly aligned data. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[27] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv:1512.03012*, 2015.

[28] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *Neural Information Processing Systems (NeurIPS)*, 2017.

[29] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[30] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2156, 2016.

[31] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[32] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal ConvNets: Minkowski convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[33] Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc V. Le. Bam! Born-again multi-task networks for natural language understanding. In *ACL*, 2019.

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

[35] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *The International Conference on Computer Vision (ICCV)*, 2017.

[36] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[37] Haowen Deng, Tolga Birdal, and Slobodan Ilic. PPF-FoldNet: Unsupervised learning of rotation invariant 3D local descriptors. *ArXiv*, abs/1808.10322, 2018.

[38] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[39] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc., 2015.

[40] Francis Engelmann, Theodora Kontogianni, Alexander Hermans, and Bastian Leibe. Exploring spatial context for 3d semantic segmentation of point clouds. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[41] Ariel Ephrat, Inbar Mosseri, Oran Lang, Tali Dekel, Kevin Wilson, Avinatan Hassidim, William T. Freeman, and Michael Rubinstein. Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation. *ACM Transactions on Graphics*, 37(4):112:1–112:11, July 2018.

[42] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) equivariant representations with spherical CNNs. In *European Conference on Computer Vision (ECCV)*, 2017.

[43] Danielle Ezuz, Justin Solomon, and Mirela Ben-Chen. Reversible harmonic maps between discrete surfaces. *ACM Transactions on Graphics (TOG)*, 38(2):15:1–15:12, March 2019.

[44] Danielle Ezuz, Justin Solomon, Vladimir G Kim, and Mirela Ben-Chen. Gwcnn: A metric alignment layer for deep shape analysis. *Computer Graphics Forum*, 36(5):49–57, 2017.

[45] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 869–877, 2018.

[46] Andrew W. Fitzgibbon. Robust registration of 2D and 3D point sets. In *British Machine Vision Conference*, pages 662–670, 2001.

[47] Tommaso Furlanello, Zachary Chase Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born-again neural networks. In *ICML*, 2018.

[48] Ross Girshick. Fast R-CNN. In *The International Conference on Computer Vision (ICCV)*, 2015.

[49] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[50] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into self-supervised monocular depth prediction. In *ICCV*, 2019.

[51] Hunter Goforth, Yasuhiro Aoki, R. Arun Srivatsan, and Simon Lucey. PointNetLK: Robust & efficient point cloud registration using PointNet. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[52] Zan Gojcic, Caifa Zhou, Jan Dirk Wegner, and Wieser Andreas. The perfect match: 3D point cloud matching with smoothed densities. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[53] Aleksey Golovinskiy, Vladimir G. Kim, and Thomas Funkhouser. Shape-based recognition of 3d point clouds in urban environments. In *The International Conference on Computer Vision (ICCV)*, 2009.

[54] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[55] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[56] Vitor Guizilini, Rares Ambrus, Sudeep Pillai, Allan Raventos, and Adrien Gaidon. 3d packing for self-supervised monocular depth estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[57] Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, and Jianwei Wan. 3d object recognition in cluttered scenes with local surface features: a survey. *Trans. PAMI*, 36(11):2270–2287, 2014.

[58] Oshri Halimi, Or Litany, Emanuele Rodolà, Alex Bronstein, and Ron Kimmel. Self-supervised learning of dense shape correspondence. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[59] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

[60] Kaiming He, Ross B. Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *The International Conference on Computer Vision (ICCV)*, 2019.

[61] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *The International Conference on Computer Vision (ICCV)*, 2017.

[62] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition Recognition (CVPR)*, 2016.

[63] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NeurIPS Deep Learning and Representation Learning Workshop*, 2015.

[64] Timo Hinzmann, Thomas Stastny, Gianpaolo Conte, Patrick Doherty, Piotr Rudol, Marius Wzorek, Enric Galceran, Roland Siegwart, and Igor Gilitschenski. Collaborative 3D reconstruction using heterogeneous UAVs: System and experiments. In *International Symposium on Experimental Robotics*, 2016.

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

[66] Paul V.C. Hough. Machine analysis of bubble chamber pictures. *The International Conference in High Energy Accelerators and Instrumentation*, 1959.

[67] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. *arXiv:1711.11575*, 2017.

[68] Qi-Xing Huang, Bart Adams, Martin Wicke, and Leonidas J. Guibas. Non-rigid registration under isometric deformations. In *Proceedings of the Symposium on Geometry Processing*, 2008.

[69] Dirk Hähnel and Wolfram Burgard. Probabilistic matching for 3D scan registration. In *Proceedings of the VDI Conference*, 2002.

[70] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *The International Conference on Machine Learning (ICML)*, 2015.

[71] Ivelin Ivanov. Review of combinatorial optimization - theory and algorithms. *SIGACT News*, 33(2):14–16, June 2002.

[72] Gregory Izatt, Hongkai Dai, and Russ Tedrake. Globally optimal object pose estimation in point clouds with mixed-integer programming. In *International Symposium on Robotics Research*, 12 2017.

[73] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[74] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

[75] Andrew E. Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *Trans. PAMI*, 21(5):433–449, 1999.

[76] Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In Joachim Diederich, editor, *Artificial Neural Networks*, pages 112–127. IEEE Press, Piscataway, NJ, USA, 1990.

[77] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. RotationNet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[78] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *Proceedings of the IEEE international conference on computer vision*, pages 1521–1529, 2017.

[79] Vladimir G. Kim, Yaron Lipman, and Thomas Funkhouser. Blended intrinsic maps. In *ACM SIGGRAPH*, 2011.

[80] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *The International Conference on Learning Representations (ICLR)*, 2014.

[81] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[82] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *The International Conference on Computer Vision (ICCV)*, 2017.

[83] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[84] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. Joint 3d proposal generation and object detection from view aggregation. In *The International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[85] 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*, pages 11867–11876, 2019.

[86] H. W. Kuhn and Bryn Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

[87] John Lambert, Ozan Sener, and Silvio Savarese. Deep learning under privileged information using heteroscedastic dropout. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[88] 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.

[89] Hei Law and Jia Deng. CornerNet: Detecting objects as paired keypoints. In *European Conference on Computer Vision (ECCV)*, September 2018.

[90] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[91] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. From big to small: Multi-scale local planar guidance for monocular depth estimation. *arXiv:1907.10326*, 2019.

[92] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *arXiv:1705.07664*, 2017.

[93] Quanquan Li, Shengying Jin, and Junjie Yan. Mimicking very efficient network for object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[94] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution on X-transformed points. In *Advances in Neural Information Processing Systems 31*, pages 828–838, 2018.

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

[96] 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)*, 2018.

[97] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[98] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. In *The International Conference on Computer Vision (ICCV)*, 2017.

[99] Haibin Ling and David W Jacobs. Shape classification using the inner-distance. *Trans. PAMI*, 29(2):286–299, 2007.

[100] Yaron Lipman and Thomas Funkhouser. Möbius voting for surface correspondence. In *ACM SIGGRAPH*, 2009.

[101] Or Litany, Tal Remez, Emanuele Rodolà, Alex Bronstein, and Michael Bronstein. Deep functional maps: Structured prediction for dense shape correspondence. In *International Conference on Computer Vision (ICCV)*, 2017.

[102] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *The European Conference on Computer Vision (ECCV)*, 2016.

[103] Xialei Liu, Hao Yang, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Continual universal object detection. *ArXiv*, abs/2002.05347, 2020.

[104] Xingyu Liu, Charles R. Qi, and Leonidas J. Guibas. FlowNet3D: Learning scene flow in 3d point clouds. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[105] Xingyu Liu, Mengyuan Yan, and Jeannette Bohg. MeteorNet: Deep learning on dynamic 3d point cloud sequences. In *The International Conference on Computer Vision (ICCV)*, 2019.

[106] Zechen Liu, Zizhang Wu, and Roland Tóth. Smoke: single-stage monocular 3d object detection via keypoint estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 996–997, 2020.

[107] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[108] David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. In *International Conference on Learning Representations (ICLR)*, 2016.

[109] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *The International Conference on Learning Representations (ICLR)*, 2017.

[110] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[111] Min Lu, Yulan Guo, Jun Zhang, Yanxin Ma, and Yinjie Lei. Recognizing objects in 3d point clouds with multi-scale local features. *Sensors*, 14(12):24156–24173, 2014.

[112] Weixin Lu, Guowei Wan, Yao Zhou, Xiangyu Fu, P. Yuan, and Shiyu Song. DeepICP: An end-to-end deep neural network for 3D point cloud registration. *ArXiv*, abs/1905.04153, 2019.

[113] Alexander M Bronstein, Michael Bronstein, and Ron Kimmel. Generalized multidimensional scaling: A framework for isometry-invariant partial surface matching. In *Proceedings of the National Academy of Sciences of the United States of America*, 2006.

[114] Siddharth Manay, Daniel Cremers, Byung-Woo Hong, Anthony J Yezzi, and Stefano Soatto. Integral invariants for shape matching. *Trans. PAMI*, 28(10):1602–1618, 2006.

[115] Haggai Maron, Nadav Dym, Itay Kezurer, Shahar Kovalsky, and Yaron Lipman. Point registration via efficient convex relaxation. *ACM Transactions on Graphics*, 35(4):73:1–73:12, July 2016.

[116] Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G Kim, and Yaron Lipman. Convolutional neural networks on surfaces via seamless toric covers. In *ACM SIGGRAPH*, 2017.

[117] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proc. 3dRR*, 2015.

[118] Daniel Maturana and Sebastian Scherer. VoxNet: A 3d convolutional neural network for real-time object recognition. In *International Conference on Intelligent Robots and Systems*, 2015.

[119] Nicolas Mellado, Dror Aiger, and Niloy J. Mitra. Super 4pcs fast global pointcloud registration via smart indexing. *Computer Graphics Forum*, 33(5):205–215, 2014.

[120] Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with Gumbel–Sinkhorn networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[121] Gregory P. Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K. Wellington. LaserNet: An efficient probabilistic 3d object detector for autonomous driving. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[122] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.

[123] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[124] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7074–7082, 2017.

[125] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *The International Conference on Machine Learning (ICML)*, 2010.

[126] Mahyar Najibi, Guangda Lai, Abhijit Kundu, Zhichao Lu, Vivek Rathod, Tom Funkhouser, Caroline Pantofaru, David Ross, Larry S. Davis, and Alireza Fathi. DOPS: Learning to detect 3d objects and predict their 3d shapes. *ArXiv*, abs/2004.01170, 2020.

[127] Jiquan Ngiam, Benjamin Caine, Wei Han, Brandon Yang, Yuning Chai, Pei Sun, Yin Zhou, Xi Yi, Ouais Alsharif, Patrick Nguyen, Zhifeng Chen, Jonathon Shlens, and Vijay Vasudevan. StarNet: Targeted computation for object detection in point clouds. *arXiv*, 2019.

[128] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *ArXiv*, abs/1807.03748, 2018.

[129] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional maps: a flexible representation of maps between shapes. *ACM Transactions on Graphics (TOG)*, 31(4):30, 2012.

[130] Andrew Oweppns and Alexei A Efros. Audio-visual scene analysis with self-supervised multisensory features. In *European Conference on Computer Vision (ECCV)*, 2018.

[131] Théodore Papadopoulo and Manolis IA Lourakis. Estimating the Jacobian of the singular value decomposition: Theory and applications. In *European Conference on Computer Vision*, pages 554–570. Springer, 2000.

[132] Dennis Park, Rares Ambrus, Vitor Guizilini, Jie Li, and Adrien Gaidon. Is pseudo-Lidar needed for monocular 3d object detection? In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[133] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. Relational knowledge distillation. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3962–3971, 2019.

[134] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[135] Gabriel Peyré and Marco Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

[136] David Pfau and Oriol Vinyals. Connecting generative adversarial networks and actor-critic methods. In *NeurIPS Workshop on Adversarial Training*, 2016.

[137] François Pomerleau, Francis Colas, and Roland Siegwart. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends in Robotics*, 4(1):1–104, May 2015.

[138] Charles R Qi, Xinlei Chen, Or Litany, and Leonidas J Guibas. Imvotenet: Boosting 3d object detection in point clouds with image votes. *arXiv preprint arXiv:2001.10692*, 2020.

[139] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep Hough voting for 3d object detection in point clouds. In *The International Conference on Computer Vision*, 2019.

[140] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. *arXiv:1711.08488*, 2017.

[141] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3d classification and segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[142] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[143] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[144] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[145] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[146] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[147] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[148] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NeurIPS)*, 2015.

[149] Thomas Roddick, Alex Kendall, and Roberto Cipolla. Orthographic feature transform for monocular 3d object detection. *arXiv:1811.08188*, 2018.

[150] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *The International Conference on Learning Representations (ICLR)*, 2015.

[151] David M. Rosen, Luca Carlone, Afonso S. Bandeira, and John J. Leonard. SE-Sync: A certifiably correct algorithm for synchronization over the Special Euclidean group. In *The Workshop on the Algorithmic Foundations of Robotics*, San Francisco, CA, December 2016.

[152] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, June 2001.

[153] Raif M Rustamov. Laplace-beltrami eigenfunctions for deformation invariant shape representation. In *Proc. SGP*, 2007.

[154] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Proc. ICRA*, 2009.

[155] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In *Proc. IROS*, 2008.

[156] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3D Point Cloud Based Object Maps for Household Environments. *Robotics and Autonomous Systems Journal*, 56(11):927–941, 30 November 2008.

[157] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. 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 4967–4976. Curran Associates, Inc., 2017.

[158] Vinit Sarode, Xueqian Li, Hunter Goforth, Yasuhiro Aoki, Animesh Dhagat, Rangaprasad Arun Srivatsan, Simon Lucey, and Howie Choset. One framework to register them all: PointNet encoding for point cloud alignment. *arXiv*, 2019.

[159] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *Transactions on Neural Networks*, 20(1):61–80, January 2009.

[160] Aleksandr Segal, Dirk Hähnel, and Sebastian Thrun. Generalized ICP. In Jeff Trinkle, Yoky Matsuoka, and José A. Castellanos, editors, *Robotics: Science and Systems*. The MIT Press, 2009.

[161] Syed Afaq Ali Shah, Mohammed Bennamoun, Farid Boussaid, and Amar A El-Sallam. 3d-div: A novel local surface descriptor for feature matching and pairwise range image registration. In *Proc. ICIP*, 2013.

[162] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Neighbors do help: Deeply exploiting local structures of point clouds. *arXiv:1712.06760*, 2017.

[163] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. PV-RCNN: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10529–10538, 2020.

[164] 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.

[165] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[166] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Groß. Complex-YOLO: Real-time 3d object detection on point clouds. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[167] Andrea Simonelli, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, and Elisa Ricci. Demystifying pseudo-LiDAR for monocular 3d object detection. 12 2020.

[168] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[169] Ayan Sinha, Jing Bai, and Karthik Ramani. Deep learning 3d shape surfaces using geometry images. In *The European Conference on Computer Vision (ECCV)*, 2016.

[170] Leslie N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

[171] Justin Solomon. Optimal transport on discrete domains. *CoRR*, abs/1801.07745, 2018.

[172] Justin Solomon, Leonidas Guibas, and Adrian Butscher. Dirichlet energy for analysis and synthesis of soft maps. In *Proceedings of the Symposium on Geometry Processing*, 2013.

[173] Justin Solomon, Andy Nguyen, Adrian Butscher, Mirela Ben-Chen, and Leonidas Guibas. Soft maps between surfaces. *Computer Graphics Forum*, 31(5):1617–1626, August 2012.

[174] Justin Solomon, Gabriel Peyré, Vladimir G. Kim, and Suvrit Sra. Entropic metric alignment for correspondence problems. *ACM Transactions on Graphics (TOG)*, 35(4):72:1–72:13, July 2016.

[175] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[176] Russell Stewart, Mykhaylo Andriluka, and Andrew Y. Ng. End-to-end people detection in crowded scenes. In *2016 IEEE Conference on Computer Vision and Pattern Recognition CVPR*, pages 2325–2333. IEEE Computer Society, 2016.

[177] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2530–2539, 2018.

[178] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *International Conference on Computer Vision (ICCV)*, 2015.

[179] Jong-Chyi Su and Subhransu Maji. Adapting models to signal degradation using distillation. In *British Machine Vision Conference (BMVC)*, 2017.

[180] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. *Computer Graphics Forum*, 28(5):1383–1392, 2009.

[181] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tak-wing Tsui, James C. Y. 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, Jon Shlens, Zhi-Feng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. *arXiv*, 2019.

[182] Peize Sun, Yi Jiang, Enze Xie, Zehuan Yuan, Changhu Wang, and Ping Luo. OneNet: Towards end-to-end one-stage object detection. *arXiv: 2012.05780*, 2020.

[183] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, and Ping Luo. SparseR-CNN: End-to-end object detection with learnable proposals. *arXiv:2011.12450*, 2020.

[184] Zhiqing Sun, Shengcao Cao, Yiming Yang, and K. Kitani. Rethinking transformer-based set prediction for object detection. *ArXiv*, abs/2011.10881, 2020.

[185] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[186] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[187] Supasorn Suwajanakorn, Noah Snavely, Jonathan J Tompson, and Mohammad Norouzi. Discovery of latent 3d keypoints via end-to-end geometric reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[188] Chengzhou Tang and Ping Tan. BA-net: Dense bundle adjustment networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[189] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *The International Conference on Computer Vision (ICCV)*, 2017.

[190] Hugues Thomas, Charles Ruizhongtai Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, Franccois Goulette, and Leonidas J. Guibas. KPConv: Flexible and deformable convolution for point clouds. In *International Conference on Computer Vision (ICCV)*, 2019.

[191] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *European Conference on Computer Vision (ECCV)*, 2019.

[192] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive representation distillation. In *The International Conference on Learning Representations (ICLR)*, 2020.

[193] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *The International Conference on Computer Vision (ICCV)*, 2019.

[194] Federico Tombari, Samuele Salti, and Luigi Di Stefano. A combined texture-shape descriptor for enhanced 3d feature matching. In *Proc. ICIP*, 2011.

[195] Frederick Tung and Greg Mori. Similarity-preserving knowledge distillation. In *The International Conference on Computer Vision (ICCV)*, pages 1365–1374, 2019.

[196] Greg Turk and Marc Levoy. The Stanford 3D Scanning Repository.

[197] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 311–318, New York, NY, USA, 1994. ACM.

[198] Oliver Van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. A survey on shape correspondence. *Computer Graphics Forum*, 30(6):1681–1707, 2011.

[199] Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 2009.

[200] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

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

[202] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems 28*, NIPS'15, pages 2692–2700, Cambridge, MA, USA, 2015. MIT Press.

[203] Tai Wang, Xinge Zhu, Jiangmiao Pang, and Dahua Lin. FCOS3D: Fully convolutional one-stage monocular 3d object detection. *arXiv:2104.10956*, 2021.

[204] Tai Wang, Xinge ZHU, Jiangmiao Pang, and Dahua Lin. Probabilistic and geometric depth: Detecting objects in perspective. In *5th Annual Conference on Robot Learning*, 2021.

[205] Tao Wang, Li Yuan, Xiaopeng Zhang, and Jiashi Feng. Distilling object detectors with fine-grained feature imitation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[206] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[207] 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.

[208] Yu Wang, Vladimir Kim, Michael Bronstein, and Justin Solomon. Learning geometric operators on meshes. *Representation Learning on Graphs and Manifolds (ICLR workshop)*, 2019.

[209] Yue Wang, Alireza Fathi, Abhijit Kundu, David Ross, Caroline Pantofaru, Thomas Funkhouser, and Justin Solomon. Pillar-based object detection for autonomous driving. In *The European Conference on Computer Vision*, 2020.

[210] Yue Wang, Alireza Fathi, Jiajun Wu, Thomas Funkhouser, and Justin Solomon. Multi-frame to single-frame: Knowledge distillation for 3d object detection. In *The Workshop on Perception for Autonomous Driving at the European Conference on Computer Vision*, 2020.

[211] Yue Wang, Vitor Guizilini, Tianyuan Zhang, Yilun Wang, Hang Zhao, and Justin M. Solomon. DETR3D: 3d object detection from multi-view images via 3d-to-2d queries. In *The Conference on Robot Learning (CoRL)*, 2021.

[212] Yue Wang and Justin Solomon. Deep closest point: Learning representations for point cloud registration. In *The International Conference on Computer Vision (ICCV)*, 2019.

[213] Yue Wang and Justin Solomon. PRNet: Self-supervised learning for partial-to-partial registration. In *Neural Information Processing Systems (NeurIPS)*, 2019.

[214] Yue Wang and Justin M. Solomon. Object DGCNN: 3d object detection using dynamic graphs. In *2021 Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[215] Yue Wang, Yongbin Sun, Sanjay E. Sarma Ziwei Liu, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38:146, 2019.

[216] Lingyu Wei, Qixing Huang, Duygu Ceylan, Etienne Vouga, and Hao Li. Dense human body correspondences using convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[217] Kelvin Wong, Shenlong Wang, Mengye Ren, Ming Liang, and Raquel Urtasun. Identifying unknown instances for autonomous driving. In *The Conference on Robot Learning (CORL)*, 2019.

[218] Zhirong Wu, Shuran Song, Aditya Khosla, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, USA, June 2015.

[219] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[220] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L. Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. *CoRR*, abs/1812.03411, 2018.

[221] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. PointFusion: Deep sensor fusion for 3d bounding box estimation. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[222] Junchi Yan, Xu-Cheng Yin, Weiyao Lin, Cheng Deng, Hongyuan Zha, and Xiaokang Yang. A short survey of recent advances in graph matching. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, ICMR '16, pages 167–174, New York, NY, USA, 2016. ACM.

[223] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. In *Sensors*, 2018.

[224] Bin Yang, Ming Liang, and Raquel Urtasun. HDNET: Exploiting hd maps for 3d object detection. In *The Conference on Robot Learning (CORL)*, 2018.

[225] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[226] Heng Yang and Luca Carlone. A polynomial-time solution for robust registration with extreme outlier rates. In *Robotics: Science and Systems*, 2019.

[227] Jiaolong Yang, Hongdong Li, Dylan Campbell, and Yunde Jia. Go-ICP: A globally optimal solution to 3D ICP point-set registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2241–2254, November 2016.

[228] Zi Jian Yew and Gim Hee Lee. 3DFeat-Net: Weakly supervised local 3D features for point cloud registration. In *European Conference on Computer Vision*, 2018.

[229] Li Yi, Haibin Huang, Difan Liu, Evangelos Kalogerakis, Hao Su, and Leonidas Guibas. Deep part induction from articulated object pairs. *ACM Transactions on Graphics (TOG)*, 37(6):209:1–209:15, December 2018.

[230] Li Yi, Vladimir G Kim, Duygu Ceylan, I Shen, Mengyan Yan, Hao Su, ARCewu Lu, Qixing Huang, Alla Sheffer, Leonidas Guibas, et al. A scalable active framework for region annotation in 3d shape collections. *TOG*, 35(6):210, 2016.

[231] Li Yi, Hao Su, Xingwen Guo, and Leonidas Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[232] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[233] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3d object detection and tracking. *CVPR*, 2021.

[234] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[235] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *The International Conference on Learning Representations (ICLR)*, 2017.

[236] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. In *Neural Information Processing Systems (NeurIPS)*, 2017.

[237] Vinícius Flores Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David P. Reichert, Timothy P. Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Relational deep reinforcement learning. *CoRR*, abs/1806.01830, 2018.

[238] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3DMatch: Learning local geometric descriptors from

RGB-D reconstructions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[239] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision (ECCV)*, 2016.

[240] Richard Zhang, Phillip Isola, and Alexei A Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[241] Xiaosong Zhang, Fang Wan, Chang Liu, Rongrong Ji, and Qixiang Ye. FreeAnchor: Learning to match anchors for visual object detection. In *Neural Information Processing Systems*, 2019.

[242] Hang Zhao, Chuang Gan, Andrew Rouditchenko, Carl Vondrick, Josh McDermott, and Antonio Torralba. The sound of pixels. In *European Conference on Computer Vision (ECCV)*, 2018.

[243] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

[244] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In *European Conference on Computer Vision*, pages 766–782, 2016.

[245] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

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

[247] Yin Zhou, Pei Sun, Yu Zhang, Dragomir Anguelov, Jiyang Gao, Tom Ouyang, James Guo, Jiquan Ngiam, and Vijay Vasudevan. End-to-end multi-view fusion for 3d object detection in lidar point clouds. In *The Conference on Robot Learning (CoRL)*, 2019.

[248] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[249] 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.

[250] Chenchen Zhu, Yihui He, and Marios Savvides. Feature selective anchor-free module for single-shot object detection. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[251] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

[252] Xinge Zhu, Yuexin Ma, Tai Wang, Yan Xu, Jianping Shi, and Dahua Lin. SSN: Shape signature networks for multi-class object detection from point clouds. In *Proceedings of the European Conference on Computer Vision*, 2020.

[253] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations*, 2021.

[254] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Proc. ICRA*, 2017.