3D Scene Understanding with Deep Learning

by

Junming Zhang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in The University of Michigan
2022

Junming Zhang
junming@umich.edu
ORCID iD: 0000-0003-1464-7676

To Xinping

ACKNOWLEDGEMENTS

This thesis would not have been possible without many people who supported me. First of all, I would like to thank my advisor Professor Matthew Johnson-Roberson. The first time I met Matt was in 442 Computer Vision class, and you were always cool looking with big headphones. Matt is a great mentor and you are always there when I feel stuck at research and support me by discussions and encouragement. I appreciate Matt's openness since I always have the freedom to go for the direction that interests me. I am also thankful to Professor Ram Vasudevan, and you are an academic role model to me. I appreciate your help for my research papers and support at the regular meetings with Ford. Thank you also to Professor Andrew Owens and Professor Jeff Fessler for being on my dissertation committee and for your support, guidance and responsiveness in all communications.

I would like to thank for labmates and they are from Ford Center for Autonomous Vehicles (FCAV), Deep Robot Optical Perception (DROP) lab, and the Robotics and Optimization for the Analysis of Human Motion (ROAHM) lab: Ming-Yuan Yu, Manikandasriram Srinivasan Ramanagopal, Alexandra Carlson, Matthew Porter, Jinsun Liu, Pengcheng Zhao, Fan Bu, Cyrus Aderson, Wonhui Kim, Xiaoxiao Du, Tianyi Zhang, Katherine Skinner. I learn a lot from you and feel grateful to be in a team of friendly and supportive lab mates. I also want to thank Waymo Inc. for providing me a memorable internship experiences during summer of 2021, and I am grateful to my mentors and my amazing co-workers at Waymo: Casey Goodlett, Bill Orr, William Baxter, Matthew Rasmussen, and Dave Gallup. I would like to express appreciation and gratitude to Ford for funding my thesis. Thank you Enrique Corona, Gaurav Pandey, Armin Parchami and Eric Tseng for valuable feedback at the monthly meeting.

In the end, I want to express my great appreciation for my family. Especially thank you to Xinping for always being my companion, my strength and my wife.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

<u>**Table**</u>

LIST OF APPENDICES

ABSTRACT

3D scene understanding is crucial for robotics, augmented reality and autonomous vehicles. In those applications, the 3D structure can be computed by using stereo cameras or depth sensors. One can process these 3D measurements using deep learning techniques to achieve remarkable performance in various perception related tasks. However, different from images that have a dominant representation as 2D pixel arrays, 3D data has many representations, including voxels, meshes, depth images, point clouds, and etc. Among all of them, depth images and point clouds are closer to the direct output from 3D measurements, as depth images are computed by stereo cameras and point clouds are generated from LiDAR. The recent improved accessibility of those 3D measurements requires the need of algorithms to interpret them. Therefore, this dissertation develops algorithms for 3D scene understanding with deep learning techniques for depth images and point clouds.

The first portion of this dissertation describes an algorithm to estimate accurate depth maps from stereo images. In particular, by solving the stereo matching problems, one can generate a disparity map and convert it into a dense depth image. During the processing, the semantic embedding learned from semantic segmentation further helps to guide the disparity estimation, especially for smooth, reflective and occluded regions. With the computed depth images and semantic segments, we can efficiently produce semantic 3D models.

The second portion of the dissertation addresses the challenges of processing point cloud data which may be arbitrarily rotated. To solve perception tasks with random rotation in real-world point cloud data, traditional techniques employ data augmentation. However, this can increase training time and may require more complex deep learning models. To address rotations that may not exist in the training data, this dissertation proposes a 3D representation of point clouds that is designed to be rotationally invariant and introduces a novel neural network architecture to utilize this representation.

The third portion of the dissertation devises methods to address the challenge of processing real-world point clouds due to partial observations. This dissertation applies a multivariate Gaussian distribution to model the output from each local point set and illustrates how to use each such local point set to infer the latent feature encoding information contained by a complete point cloud. This strategy ensures accurate prediction with a partial observed point set for different tasks, such as shape classification, part segmentation, and point cloud completion.

The final portion of the dissertation focuses on point cloud completion. At processing the point clouds, existing approaches adopt encoder-decoder structures and output sparse distributed embeddings, which may lead to worse generalizability at testing. In addition, analysis of point cloud completion trained jointly with other tasks are lacking. To address those limitations, this dissertation proposes a novel module that includes a normalization layer to normalize embeddings into unit one, and the module can be integrated into existing approaches. Both the theory and empirical results are shown to demonstrate the effectiveness of the proposed method on improving point cloud completion performance.

CHAPTER I

Introduction

1.1 Motivation

In recent years, we have observed many emerging applications using algorithms to interpret images from cameras and generate high-level semantic outputs. For example, a security system equipped with high-resolution cameras requires identifying people and vehicles while minimizing unwanted alerts by others, such as animals or shadows; medical image analysis may need to partition images into semantic segments to diagnose pathologies or guide medical interventions such as surgical planning, or for research purposes. Those applications are based on 2D measurements from sensors, while more recent applications in robotics, augmented reality (AR), and autonomous vehicles are equipped with advanced sensors and some of them are able to measure the 3D environment. For instance, AR glasses are equipped with depth cameras to perceive the 3D geometry of environment and correctly display virtual objects; the perception system in autonomous vehicles contains depth sensors, and they are used to identify locations of obstacles in the world to constrain the feasible region for driving, as well as understand their high-level categories, such as pedestrian, to decide whether to yield. In those applications, 3D sensors are able to generate a fair amount of 3D measurements and this increased accessibility of 3D data requires algorithms to process and analyze them. Inspired by the success of deep learning in solving 2D image analysis tasks [1, 2, 3, 4, 5], we applied deep learning to address the challenges in 3D scene understanding.

Different from images that have a dominant representation as 2D pixel arrays, 3D data has many representations, including voxels, meshes, depth images, point clouds, and etc. The diversity of representations are generated by either different depth sensors or tailored to different computational algorithms. Volumetric representation contains a voxel grid converted from point clouds or meshes, and a voxel is one if there is a point in it, or zero if it is empty. By organizing the data in a regular way, volumetric representation enables the usage of existing kernel methods, such as 3D convolutions [6, 7, 8]. However, the efficiency of processing volumetric representation is limited by the grid resolution, and a worse case is that much computation is wasted on processing empty voxels within the volume of objects since valid voxels are converted only on the surfaces of objects. Moreover, quantization loss is inevitable when converting into voxels from other representations, which may lead to degradation of analysis compared to directly processing raw representations.

Meshes define the 3D shapes of objects in a more compact way by using a collection of vertices, edges and faces, which requires less storage space compared to voxels. However, the number of elements in a mesh may vary dramatically and have different combinations, which raise challenges for mesh processing. Some works have been proposed to perform mesh classification and generation using Deep Neural Networks (DNNs) methods [9, 10, 9], but the progress in mesh processing is lagging behind other 3D data representations. One possible reason is that meshes are not a direct data format from any depth sensors, so representation conversion is needed before processing meshes, which negatively affects the efficiency in real-world applications. Therefore, this dissertation is particularly interested in analyzing depth images and point clouds since they are a data format closer to the output from depth sensors.

Depth images are 2D pixel arrays, which contain the pixel-wise distance between the image plane and the surfaces of scene objects. Usually, depth images are obtained by depth cameras or solving the stereo matching problem. Because of the lower cost of stereo setup, this dissertation is motivated to focus on how to accurately derive depth images from stereo images. A general pipeline of generating depth images from stereo images is to first compute disparity maps by solving stereo matching problems and then convert disparity into depth with the calibrated camera parameters, in which the key step is to obtain an accurate disparity map. To solve the stereo matching problem, traditional methods either use local descriptors to find the matching points within a predefined window [11], or they minimize an energy function globally to get an optimal solution [12]. Unfortunately, the results of them are not reliable due to the ineffective hand-crafted features, and the long processing time makes it impractical for real-time applications. To address those limitations, recent works applied convolutional neural networks (CNNs) methods to solve stereo matching problem and achieved impressive performance by effectively learning features from large datasets in an end-to-end training manner[13, 14, 15, 16]. However, this end-to-end disparity regression requires dense annotation of disparity maps that are difficult and expensive to acquire[17, 18], which partially explains that the size of available disparity datasets is relatively smaller than those for other tasks, such as classification and detection. For example, KITTI 2012 and KITTI 2015, the most popular benchmarks for the stereo matching task collected in the real-world scenarios, contain no more than 400 stereo images for training. Despite the advances of disparity estimation using CNNs methods, obtaining correct disparity in regions of reflection, occlusion, and slow-texture is still challenging. Empirically, those difficult regions are located within large semantic segments, so we are motivated to incorporate more contextual semantic information during estimating disparity. Therefore, this dissertation proposes a self-supervised model in which the disparity estimation is improved by guidance of semantic information learned from semantic segmentation task.

Point clouds are probably the closest representation to raw sensor output and encode full information. Similar to mesh representation, directly processing point clouds is challenging due to the irregularity of the points they contain. Previous methods converted point clouds into other representations, such as depth images and voxels, before proccessing point clouds, but the data conversion degrades the resolution of measured objects which can adversely affect point cloud analysis. More recently, PointNet is the pioneer work in achieving end-to-end learning for the irregular point data by using symmetric function to address the permutation issues in the point set [19]. Since then many works have extended PointNet by proposing different structures to effectively extract information from points clouds [20, 21, 22]. However, the representations learned by most existing approaches are from point clouds with canonical poses and lack invariance to rotations, while they are common in real-world scenarios. A typical way to increase the robustness of models to rotation is augmenting the training point clouds with additional data and random rotation. However, data augmentation will require more computation resources at training and designing models with larger capacity. To address above limitations, this dissertation proposes a new representation of point clouds that is designed to be rotationally invariant and introduces a novel neural network architecture to utilize this representation.

Incomplete measurement is another challenge when processing point clouds. Recent works have achieved impressive performance on point cloud analysis [19, 20, 23, 24, 25, 26, 27] by learning meaningful representations from large 3D datasets [17, 28, 29]. However, point clouds provided in those dataset are often incomplete and sparse due to occlusions, low resolution and the limited view of 3D sensors, which lacks complete information and raises challenges for analyzing point clouds. Therefore, the ability to predict complete shapes of objects from partial observation is desirable. Moreover, the complete shapes may be useful in certain applications requiring finer shapes. For example, the predicted complete shapes will provide finer collision-check boundaries than 3D bounding boxes to help autonomous vehicles navigate in narrow lanes or crowded urban regions; compared to incomplete measurement, the robot

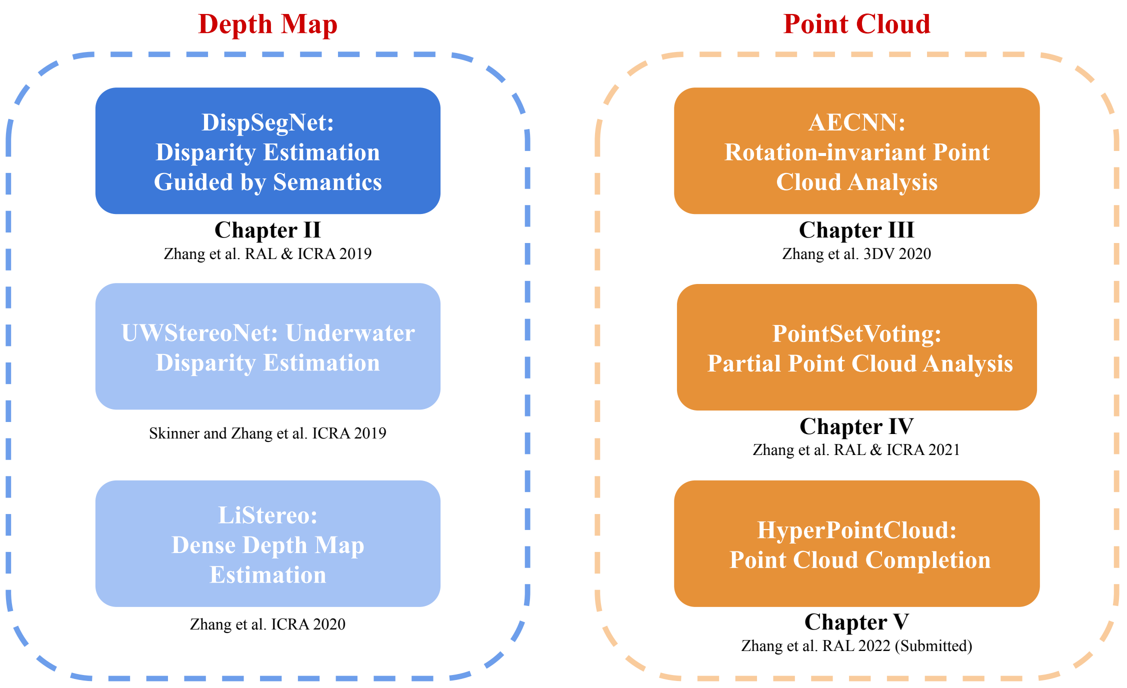

Depth Map
Point Cloud
DispSegNet: Disparity Estimation Guided by Semantics
Chapter II
Zhang et al. RAL & ICRA 2019
UWStereoNet: Underwater Disparity Estimation
Skinner and Zhang et al. ICRA 2019
LiStereo: Dense Depth Map Estimation
Zhang et al. ICRA 2020
AECNN: Rotation-invariant Point Cloud Analysis
Chapter III
Zhang et al. 3DV 2020
PointSetVoting: Partial Point Cloud Analysis
Chapter IV
Zhang et al. RAL & ICRA 2021
HyperPointCloud: Point Cloud Completion
Chapter V
Zhang et al. RAL 2022 (Submitted)

Figure 1.1: Dissertation outlines. Algorithms for effective disparity estimation and point cloud analysis are proposed for 3D scene understanding.

arm can grasp objects in more reasonable poses predicted from the complete shapes, in particular, for those unknown objects. Thus, this dissertation targets challenges due to incomplete measurement and devises algorithms for shape classification, part segmentation, and point cloud completion in partial point cloud analysis.

1.2 Outline

The outlines of this dissertation is illustrated in Figure 1.1, and contents for each chapter are as follows:

Chapter II targets at improving disparity estimation in certain difficult regions with reflection, occlusions and low-textures. We set out to exploit the connection between these two pixel labeling tasks – disparity estimation and semantic segmentation – to improve the performance for disparity estimation. We argue that segment embedding learned from semantic segmentation can provide more cues for estimating disparity by constraining smoothness of disparity within difficult regions. From this perspective, models for disparity estimation need to have a high-level understanding of objects or at least segments, so stereo matching is no longer a low-level vision problem. To achieve this, this chapter introduces a multi-task model which outputs a disparity map and semantic segments simultaneously while the two tasks are coupled, and we can obtain 3D semantics by projecting both outputs into 3D space.

The content of this chapter is based primarily on Zhang et al. [30].

In Chapter III, we focus on learning a novel 3D representation of point clouds that is invariant under rotations and introduces a new neural network architecture to utilize this representation. By leveraging the notion of local reference frame (LRF), we ensure different orientations of a point cloud are mapped into the same representation by projecting into a local LRFs. The introduced architecture processes these local internal features and aligns them with local internal features drawn from other LRFs before fusing them together in a hierarchical fashion to define global features. We demonstrate the invariance to rotation of the proposed method on point cloud classification and part segmentation. The content of this chapter is based primarily on Zhang et al. [31].

In Chapter IV and Chapter V, we target the challenges due to incomplete measurement in analyzing point clouds. In contrast to prior works on learning a representation of all points within the point clouds, Chapter IV proposes to utilize a shared-weight encoder to embed each local point set by a multivariate Gaussian distribution. The embedded local point sets vote to infer the information contained in a complete shape, which resides in the latent space characterized by a distribution. This probabilistic modeling enables output more accurate prediction when observing more parts of shapes and generates multiple possible outputs due to the uncertainty of the partial observation. The proposed methods are general to different point cloud tasks, and we show state-of-the-art results on shape classification, part segmentation, and point cloud completion with partial clouds. The content of this chapter is based primarily on Zhang et al. [32].

We narrow down our focus to point cloud completion tasks in Chapter V. Most point cloud completion networks consist of an encoder-decoder structure, in which the encoder extracts embeddings that are used to generate output for different tasks. However, the learned embeddings are shown to be sparsely distributed in the feature space, which may lead to worse generalization results at testing when unseen embeddings are not captured by embeddings at training. Moreover, the existing approaches lack analysis of point cloud completion trained in multi-task learning. To address these problems, we propose a hyperspherical module, which takes the embeddings from encoders as inputs and transforms and normalizes them to be on a unit hypersphere before passing them to following decoders. The consistent improvement of point cloud completion observed in both single-task and multi-task learning verify the effectiveness of the proposed method.

Finally, Chapter VI outlines several main ideas for future research, based on the findings in this dissertation, and my vision of future directions for 3D perception.

CHAPTER II

Leveraging Semantics for End-to-End Learning of Disparity Estimation from Stereo Imagery

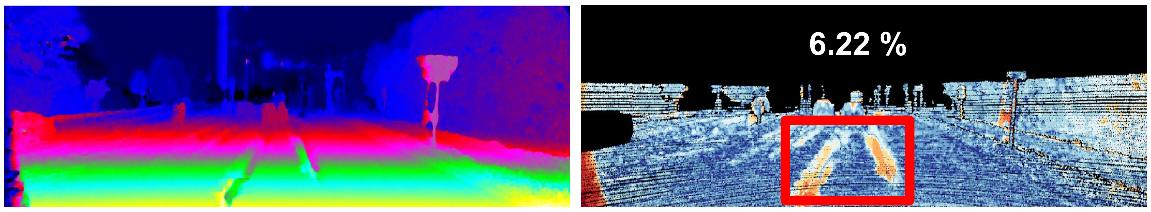
Recent work has shown that CNNs can be applied successfully in disparity estimation, but these methods still suffer from errors in regions of low-texture, occlusions and reflections. Concurrently, deep learning for semantic segmentation has shown great progress in recent years. In this chapter, we design a CNN architecture that combines these two tasks to improve the quality and accuracy of disparity estimation with the help of semantic segmentation. Specifically, we propose a network structure in which these two tasks are highly coupled. One key novelty of this approach is the two-stage refinement process. Initial disparity estimates are refined with an embedding learned from the semantic segmentation branch of the network. The disparity estimation task in the proposed model is trained using an unsupervised approach, in which images from one half of the stereo pair are warped and compared against images from the other camera. Another key advantage of the proposed approach is that a single network is capable of outputting disparity estimates and semantic segments. These outputs are of great use in autonomous vehicle operation; with real-time constraints being key, such performance improvements increase the viability of driving applications. Experiments on KITTI and Cityscapes datasets show that our model can achieve state-of-the-art results and that leveraging embedding learned from semantic segmentation improves the performance of disparity estimation. [1]

2.1 Introduction

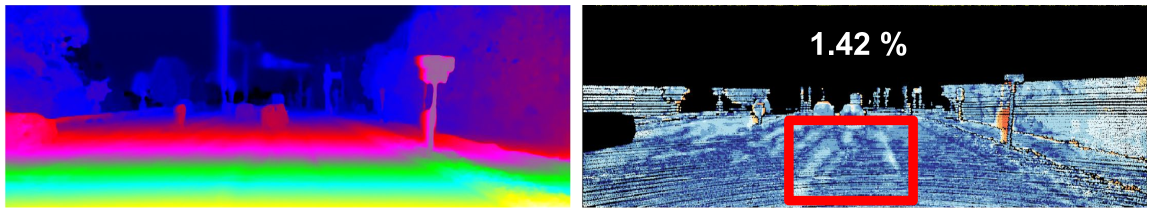
Disparity estimation is an important problem in low-level vision. Given two stereo rectified images, disparity refers to the relative horizontal displacement of two corresponding pixels in the left and right images. From dense disparity maps, we can estimate three dimensional geometry, which is critical for many computer vision applications, including autonomous vehicle navigation and 3D model reconstruction.

[1]This chapter is based on [30]

(a) Input stereo images



(b) Disparity prediction and error map without segment embedding



(c) Disparity prediction and error map with segment embedding

Figure 2.1: Examples of advantage of fusing segment embedding into disparity estimation. With fused segment embedding, our model performs better in ill-posed regions, such as the area within the red bounding box in each figure. The white numbers in the error maps indicate percentage of incorrect pixels in all regions.

Traditionally, dense disparity has been estimated using window-based correlation, with smoothing, occlusion and globally-optimal matching constraints applied [33, 34, 35, 12]. However, it is difficult to hand-craft these constraints. Additionally, global optimization is not practical for real-time applications. Recently, stereo matching has greatly advanced with the help of CNNs, which proves that features learned by CNNs are more effective than hand-crafted ones. More sophisticated architectures are able to estimate dense disparity through end-to-end training. This end-to-end disparity regression from stereo pairs requires a large amount of image pairs with ground truth disparities during training. However, there is currently no such real dataset available. Instead, models are pretrained on large synthetic datasets [15, 36] and then fine-tuned on the real-world target dataset. With this training pipeline, recent papers [16, 37, 38, 39] achieve an impressive error rate below 2% in the KITTI benchmark stereo matching task [17, 40]. Still, there are challenges for training on synthetic data and testing on real data. In this chapter, we focus on developing an unsupervised method to do stereo matching for dense disparity estimation to help overcome these challenges.

Despite advances in disparity estimation since the application of CNNs, finding correspondences in regions of high specularity, occlusions or low-texture regions is still a challenging problem. These areas manifest themselves as noise or missing regions in the resulting disparity map. For example, in Figure 2.1, the disparity for the center of the road is incorrect because it is an area of low-texture and it is hard to find correspondence in this region. We argue that more contextual semantic information is needed to determine accurate disparity in these challenging regions.

With the rise and success of object classification [41, 1], a new task known as semantic segmentation has also gained popularity and benefited from access to large amounts of labeled data [42, 43]. This problem moves beyond simple bounding boxes and attempts to assign every pixel in an image a semantic label. The dense nature of this problem is complimentary to the disparity estimation task. Moreover, segment embedding learned from semantic segmentation can provide further cues for estimating disparity within ill-posed regions, because disparity tends to be smooth within an object or segment. From this perspective, models for disparity estimation need to have a high-level understanding of objects or at least segments, so stereo matching is no longer a low-level vision problem. Here, we set out to exploit the connection between these two pixel labeling tasks – disparity estimation and semantic segmentation – to improve the performance for disparity estimation. In this chapter, we focus on unsupervised stereo matching guided with the semantic segmentation task and the main contributions are as follows:

- We propose a model which outputs a disparity map and semantic segments simultaneously, and then both can be used to acquire 3D semantic information.
- We propose a structure and a smoothness loss which better fuses segment embeddings learned from the semantic segmentation task into the process of disparity estimation. Experiments show that these are helpful for disparity estimation.
- Our unsupervised model is able to achieve state-of-the-art results in the KITTI stereo vision benchmark dataset, and can also beat some supervised methods in certain regions.

2.2 Related Work

Typical stereo matching pipelines consist of four steps: matching cost computation, cost aggregation, disparity estimation and refinement. Traditional methods either use local descriptors to find the matching points within a predefined window [11], or they minimize an energy function globally to get an optimal solution [12].

Supervised Disparity Estimation. Stereo matching has greatly advanced since CNNs were applied to this task by [13]. That method was supervised, requiring large datasets with stereo images and disparity ground truth. With this supervised approach, after meaningful features are extracted from a deep Siamese architecture, a cost volume can be computed by simply concatenating features from both sides [13], dot products [14, 13], a correlation function [15], or by concatenating all potential corresponding feature vectors from both sides [16]. Several other papers have also focused on using information from cost volumes. They proposed different methods and structures, including simple convolutional layers [15], learning context from 3D convolution [16], using a spatial pyramid pooling module to incorporate more global context [38], a two-stage refinement structure [44] and two separate branches for small and large disparities [45, 39]. In line with these suggestions, we form a five-dimensional cost volume by concatenating features from both sides and extracting information from it using 3D convolution. We then refine the initial disparity using extra information from segment embedding.

Although some large datasets are now available for training in stereo matching, the size of available datasets is still relatively small compared to popular datasets for classification and detection. For example, KITTI 2012 and KITTI 2015, the most popular datasets for the stereo matching task, contain no more than 400 stereo images for training. In cases like this, unsupervised stereo matching has gained attention because it does not require ground truth disparity for training. Because of this, we focus on unsupervised learning in our approach for the stereo branch of our network. This maximizes the flexibility of the training sources, which is important because stereo ground truth is difficult to obtain.

Unsupervised Disparity Estimation. Deep unsupervised stereo matching relies heavily on warping error [46, 47, 48, 49]. This error is measured as the visual difference between a warped image from one half of a stereo pair and the real image from the other camera in the stereo setup. End-to-end training has become popular recently thanks to differentiable bilinear sampling, which can be used to warp images [47]. Additionally, a smoothness loss and left-right consistency loss also help improve the quality of results [47, 50]. Although results of these unsupervised methods are reasonable, a large performance gap still exists between these approaches and supervised methods. In this chapter, we mainly focus on unsupervised stereo matching, and seek to use supervised semantic segmentation to help narrow this gap.

Guided Disparity Estimation. Both supervised and unsupervised stereo matching methods still have difficulty estimating correct disparity in flat, reflective and occluded regions. Thus, recent papers have sought to leverage extra information such as object-level knowledge [51] and segment embedding [37]. Their results show that exploiting available high-level information is useful for improving performance on the task of dense disparity estimation.

In this chapter, we propose a fused model for semantic segmentation and dispar-

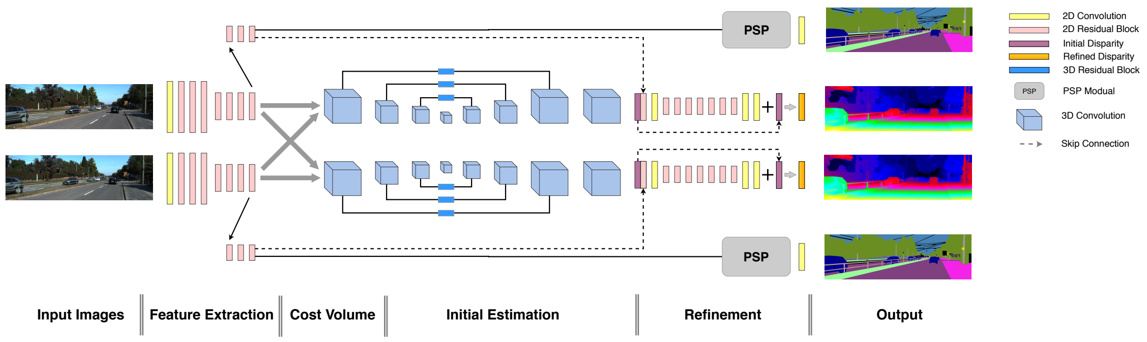


Figure 2.2: Architecture of our model. The pipeline of our model consists of the following parts. (a) Input images: rectified input stereo images. (b) Feature extraction: useful features are extracted from input stereo images. (c) Cost volume: cost volume is formed by concatenating corresponding features from both sides. (d) Initial estimation: initial disparity is estimated from cost volume using 3D convolution. (e) Refinement: initial disparity is further improved by fusing segment embedding. The PSP module is used to incorporate more context information for the semantic segmentation task. (f) Output: estimated disparity and semantic segmentation from both left and right views are generated from the model. In this figure, 2D and 3D residual blocks are similar to identity blocks that are defined in the [1].

ity estimation that does not require ground truth disparity maps. Our proposed method is most similar to SegStereo [37], which was developed simultaneously with our approach. However, our methods differ in several important ways. We focus on unsupervised stereo matching, where segment embedding is not only fused into disparity estimation, similar to SegStereo, but also is used to regularize disparity in the loss. Additionally, SegStereo computes a correlation layer, which may lose information, but we form a cost volume retaining all features, which enables the network to learn more complete feature representations. With additional refinement on the initial disparity, the results of our model outperform SegStereo by over a 2.5% error rate on KITTI benchmark.

2.3 Methods

We present a joint model for disparity estimation and semantic segmentation. These two tasks are highly coupled in the network, with the semantic segment embedding being directly fused into the refinement process for disparity estimation. The whole architecture of our model is illustrated in Figure 2.2.

2.3.1 Architecture for Disparity Estimation

ResNet 50 structure [1] is used in the Siamese structure, which processes both the left and right images and generate high-level features for stereo matching and semantic segmentation. Features for segmentation task come from deeper layers of the network than those used for the stereo matching task, as the former requires more contextual information than the latter. Each task corresponds to a branch in the network. In the disparity branch, the size of input features is 1/4 of original stereo images. We concatenate features for stereo matching from the left and right viewpoints, and this produces a five-dimensional cost volume. An eight-layer encoder-decoder with 3D convolution is then used to predict an initial disparity map from this volume. The structure of encoder-decoder is shown in Figure 2.2 and the relative size of cube indicates the relative size of each layer. 3D transpose convolution is used in the decoder, and skip layers are processed by 3D residual blocks. The segment embedding is first resized to the same shape of original image and concatenated with the initial disparity map to do refinement. In details, the convolution layers in this chapter refer to a convolution layer followed by a batch normalization layer [52] and a leaky ReLU layer [53], except for the final output layer which only contain a regular convolution. The size of all kernels is 3 except for the first convolution layer in Siamese structure, which is 7. The 2D residual block is three layers deep and the 3D residual block is two layers deep.

2.3.2 Cost Volume and Learning Context

After calculating left and right features for stereo matching, we form a cost volume by concatenating them. Every feature vector from one side is concatenated with all potential corresponding feature vectors from the other side. This results in a cost volume with a dimensionality of Batchsize $\times$ (Max_disparity+1) $\times$ Height $\times$ Width $\times$ Feature_size. We form both left and right cost volumes to calculate a disparity for both views. Unlike other methods that use dot product or other metrics to measure correlation between feature vectors, the five-dimensional cost volume here enables the network itself to learn better correlation metric in an end-to-end training manner.

To extract information from the cost volume, a 3D convolution filter loops over all three dimensions of height, width and potential disparity values, which captures broader contextual information. Since 3D convolution is memory intensive, an encoder-decoder structure is used to reduce the memory footprint. In the end, soft argmin is used to produce the initial disparity map from this intermediate result.

2.3.3 Disparity Refinement

The initial disparity estimation contains too much noise and its accuracy is limited by error from poor matching in ill-posed regions, such as occluded, reflective and texture-less areas. However, the semantic segment embedding can be used to improve correspondence in those regions. Disparity in the ill-posed regions should have similar values as regions from the same semantic segment. Essentially, the same smoothness constraint that is often applied globally can more accurately be applied within object boundaries. To this end, after producing the initial disparity map, we use semantic segment information to refine the disparity. The residual structure of the refinement process is shown in Figure 5.1.

After convergence, we assume the initial disparity is reasonable in most regions, so in the refinement stage we then focus on refining the disparity in ill-posed regions. The residual structure is used here and forces the model to learn this highly non-linear relationship in such regions. The initial disparity and the semantic segment embedding are concatenated as the input to later process. The output is then summed with the initial disparity to get the final estimation.

2.3.4 Architecture for Semantic Segmentation

In both the KITTI and Cityscapes [43] datasets, only the left image from the stereo pair is labeled with ground truth semantic segments. However, we perform semantic segmentation on both images in the pair. Left disparity is used to warp the right predicted semantic segments to the left view, which is in turn regularized by the left labels during training. The PSP module [5] is used to incorporate more contextual information from different scales. The size of input features to the PSP module is 1/8 of original stereo image. In the PSP module, input features are downsampled into three different sizes using averaging pooling, at scales of 1/2, 1/4 and 1/8 of the original input size. Then they are followed by a convolution with a 1x1 filter individually to reduce feature dimension to 1/4 of the original input feature dimension. Different scales of features are then concatenated after they are upsampled to the shape of the input feature space through bilinear interpolation. Finally it is followed by a 1x1 convolution to mix features at different scales.

2.3.5 Loss Function

For our approach, we pose stereo matching as an unsupervised problem. The object loss consists of three items that are defined as the following:

$$Loss = \alpha_1 L_{init} + \alpha_2 L_{ref} + \alpha_3 L_{seg} \tag{2.1}$$

$$L_{init} = \beta_1 L_p + \beta_2 L_c + \beta_3 L_r \tag{2.2}$$

$$L_{ref} = \gamma_1 L_p + \gamma_2 L_c + \gamma_3 L_s \tag{2.3}$$

where L_{init} supervises initial estimated disparity, L_{ref} supervises refined estimated disparity and L_{seg} supervises predicted semantic segments. We set $\alpha_1 = 0.3$, $\alpha_2 =$

0.7, $\alpha_3 = 0.1$, $\beta_1 = 0.8$, $\beta_2 = 0.01$, $\beta_3 = 0.001$, $\gamma_1 = 0.8$, $\gamma_2 = 0.05$ and $\gamma_3 = 0.005$ during the training. Other terms in the equation are defined as follows:

Photometric loss (L_p): Let I_L and I_R be the input left and right images, and D_L and D_R be the predicted left and right disparity maps. The warping function $F(I, D)$ is able to warp image I to the other view based on the disparity map D using bilinear sampling. The reconstructed left image is $I_L^{'} = F(I_R, D_L)$, and the reconstructed right image is $I_R^{'} = F(I_L, D_R)$. The reconstructed image should be very similar to the original input image. We use both Euclidean distance and a structure similarity term SSIM $S(\cdot)$ to improve the robustness in ill-posed regions [47]. For the left image, photometric loss is defined as follows:

$$L_p = \lambda_1 S(I_L, I_L^{'}) + \lambda_2 |I_L - I_L^{'}| + \lambda_3 |\nabla I_L - \nabla I_L^{'}| \tag{2.4}$$

where we set $\lambda_1 = 0.85$, $\lambda_2 = 0.15$, $\lambda_3 = 0.15$. These values were selected through experimentation.

Regularization loss (L_r): Regularization loss is used to smooth local disparity with information directly from input images, and we only use it in estimating initial disparity. We assume disparity in the local region tends to be smooth, so we add a regularization loss to suppress high frequency noise introduced by the photometric loss term. This regularization loss is the sum of the weighted second derivative of the disparity map, and the weight is the exponential of the second derivative of the input image. The higher the second derivative of the input image, the higher the probability of a change in disparity. For the left side, regularization loss is defined as follows:

$$L_r = \frac{1}{N} \sum |\nabla_x^2 D_L| e^{-|\nabla_x^2 I_L|} + |\nabla_y^2 D_L| e^{-|\nabla_y^2 I_L|} \tag{2.5}$$

where N is number of pixels, ∇_x^2 and ∇_y^2 are second derivatives along the X and Y axes.

Consistency loss (L_c): We can also synthesize a left image from the reconstructed right image $I_L^{''} = F(I_R^{'}, D_L)$ and a right image from the reconstructed left image $I_R^{''} = F(I_L^{'}, D_R)$. Consistency loss is defined as follows:

$$L_c = |I_L - I_L^{''}| + |I_R - I_R^{''}| \tag{2.6}$$

This consistency forces the left and right branches to be consistent with one another [50].

Smoothness loss (L_s): For difficult regions, we argue that the network should be able to infer the disparity from its neighbors within a segment, and we propose to use a left-right consistency check to find these regions. Smoothness loss is computed in the refinement stage. We warp the right disparity D_R using the left disparity D_L, and we form a reconstructed image $D_L^{'} = F(D_R, D_L)$. Then we threshold the absolute difference between D_L and $D_L^{'}$:

$$Diff = \begin{cases} ||D_L - D_L^{'}||, & ||D_L - D_L^{'}|| <= t \\ t, & ||D_L - D_L^{'}|| > t \end{cases} \tag{2.7}$$

where t is the threshold and is set to 3 during the experiment. Too large of a threshold will result in a trivial solution. In addition, the disparity should be smooth inside a segment. These segments are learned from the semantic segmentation task. Shallower layers are used here instead of the final semantic segmentation layer, biasing to smaller segments being learned. We apply a cost to enforce smoothness within a segment. For the left side,

$$\begin{aligned} L_s = \frac{1}{N} \sum |\nabla_x^2 D_L| (e^{-|\nabla_x^2 f_L|} + e^{(Diff - t)}) \\ + |\nabla_y^2 D_L| (e^{-|\nabla_y^2 f_L|} + e^{(Diff - t)}) \end{aligned} \tag{2.8}$$

where f_L is feature vectors from the left view. This loss is only applied during refinement because it is conditioned on relatively good initial disparity.

Segmentation loss (L_{seg}): Conventional softmax cross entropy loss is used to measure the difference between the logits map and the ground truth segment labels. For the stereo image dataset, only images from one side will be labeled. For example, KITTI and Cityscapes only have segment labels for the left images. However, the left disparity map will relate the left and right images. So we can use the left disparity map to warp the right output segments to the left, and then we can use the left ground truth label for supervision.

2.3.6 Post Processing

Simple post processing can be used to improve the final results. Although the loss of smoothness can reduce the effects of occlusion, our model is still prone to error in those regions. Our post processing consists of two steps: left-right consistency check and interpolation.

After calculating both left and right disparity, we perform a left-right consistency check. For left view images, a pixel will fail the check if the difference between disparity values from the left view and the corresponding pixel from the right view is greater than a certain threshold. We set this threshold to 1, and we end up with a boolean mask. We also apply a median filter to this mask because it contains a fair degree of noise. Then, in these failure regions, we assign them disparity values from the background. As proposed by [13], we interpolate by moving left until finding a position with a valid disparity and use this as its value. No further global optimization is applied.

2.4 Experiments

In this section, we explain our implementation details and present qualitative and quantitative results.

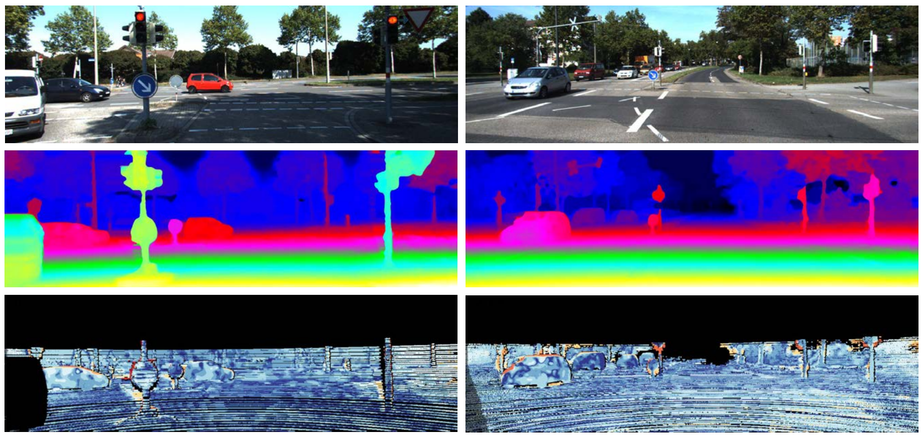
2.4.1 Datasets

KITTI: KITTI 2012 and 2015 are two benchmark real-world driving datasets. They provide ground truth disparity computed from a calibrated high-resolution 3D LIDAR. There are approximately 200 rectified stereo images with ground truth disparity for evaluation in both KITTI 2012 and 2015. We primarily focus on the KITTI 2015 benchmark. Compared to KITTI 2012, challenging regions (e.g. car windshields) from KITTI 2015 are more correctly represented in the ground truth because it uses CAD models to produce disparity values for evaluation. Additionally, only KITTI 2015 contains ground truth for semantic segmentation. For evaluation, pixels are divided into two overlapping categories: strictly non-occluded regions (NOC) and all pixel regions (ALL). The KITTI 2015 benchmark considers a pixel to be ”correct” if the disparity error is less than 3 pixels and within 5% disparity error.

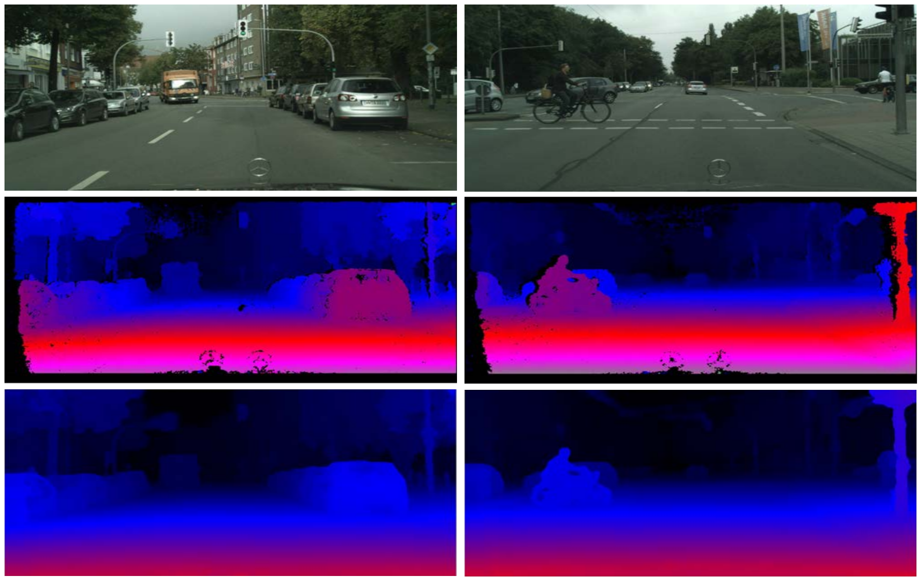
Cityscapes: Cityscapes is a dataset for semantic urban scene understanding. It contains 5,000 stereo color images collected from 50 cities, with high quality pixel-level ground truth semantic labels for the left view of each pair. These images are split into sets, with 2,975 for training, 500 for validation and 1,525 for testing. There are no ground truth disparity maps in the Cityscapes dataset, but disparity maps are provided using the SGM [12] algorithm.

2.4.2 Implementation Details

In the experiments, we implement our architecture in TensorFlow. All experiments are run on a single NVIDIA Titan-X GPU. Original stereo images are normalized to values ranging from -1 to 1. Due to GPU memory limitation, we have a maximum batch size of 1, the maximum disparity is set to 192 and images are randomly cropped down to 256x512 patches before feeding into network. During optimization, we use the Adam optimizer [54] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e{-}8$. The learning rate is set to $2e{-}4$ for pre-training on Cityscapes and $1e{-}4$ for fine-tuning on KITTI, and it is halved every $20,000$ iterations. Pre-training on Cityscapes is done for $100,000$ iterations. We then fine-tune the model on KITTI for an additional $50,000$ iterations. The finetuning process takes approximately 1 day. No data



(a) Sample results on test set in KITTI 2015. From top: left stereo input image, disparity prediction, error map.



(b) Sample results on Cityscapes. From top: left stereo input image, disparity prediction from SGM, disparity prediction from ours.

Figure 2.3: Qualitative results on KITTI 2015 and Cityscapes datasets. (a)Sample results on KITTI 2015 test set. The last column shows the error maps which are captured from KITTI benchmark. Error regions are displayed in orange color. (b)Sample results on Cityscapes. Compared with predication from SGM, our model generates more smooth and complete disparity map.

augmentation is performed in the experiments.

2.4.3 Evaluation

Here, we report the results of our model on the KITTI and Cityscapes datasets and compare our approach to other state-of-the-art methods.

KITTI Benchmark

We report results on 40 validation images split from 200 training stereo images from KITTI 2015 to evaluate our model. We compare our model with other unsupervised learning methods in Table 2.1. Note that our model outperforms other unsupervised methods by a notable margin. In the table, 'CS' refers to training model on Cityscapes dataset, 'K' refers to training model on KITTI and 'PP' refers to refining disparity with post processing. With pre-training on the Cityscapes dataset and simple post processing, results of our model are further improved. In addition, Table 2.2 compares our method to other supervised approaches on the KITTI 2015 leaderboard. Although there is a gap between performance of current state-of-the-art supervised methods, our model achieves comparable results and even beats DispNet, a supervised method, on background regions. Sample qualitative results are shown in Figure 2.3 (a).

We note that our proposed method has relatively large error in the foreground region. We argue that it is because of significantly larger and more common occlusion and reflection in foreground regions, such as surfaces of vehicles. There exists no correspondence on these regions in the input stereo images. However, unlike other supervised methods that have access to ground truth disparity, our proposed method highly relies on these correspondence to form photometric loss and uses it as supervision. So it is reasonable that our method performs poorly on foreground regions. Although semantic segments and post processing have been used to greatly reduce such errors, our method cannot reach the accuracy of those supervised methods.

Cityscapes

We only show qualitative results from the Cityscapes dataset because it does not provide ground truth disparity maps. The results are shown in Figure 2.3. Note that compared with the SGM approach, our model is able to generate much more complete and visually accurate disparity maps.

2.4.4 Ablation Study on Loss Components

We perform ablation experiments to evaluate the different components of our developed loss function. Results of the ablation study are shown in Table 2.4. Models are trained and evaluated on the KITTI 2015 without pretraining or any post processing. The results of our model are improved due to the two-stage refinement, designed smoothness loss and incorporation of semantic segmentation supervision. Specifically, the error rate is reduced from 7.04 to 6.53 with designed smoothness loss and is further reduced from 6.53 to 5.93 with segment supervision. Figure 2.1

Table 2.1: Comparison with other unsupervised models on disparity estimation. Results are reported on the KITTI 2015 stereo validation set manually splitted from 200 training images. 'CS' refers to training model on Cityscapes dataset; 'K' refers to training model on KITTI; 'PP' refers to refining disparity with post processing. With pretraining on Cityscapes, fine-tuning on KITTI and post processing, our model outperforms other unsupervised methods by a large margin.

Model	NOC pixels	All pixels
USCNN [55]	11.17	16.55
Zhou et al. [48]	8.61	9.91
Godard et al. [47]	-	9.19
SegStereo [37]	7.70	8.79
Luo et al. [49]	6.31	6.63
Ours(CS)	6.55	7.24
Ours(K)	5.93	6.32
Ours(CS & K)	5.84	6.29
Ours(K & pp)	5.29	5.69
Ours(CS & K & pp)	**5.20**	**5.67**

Table 2.2: Comparison with other supervised methods on disparity estimation. Results are reported on KITTI 2015 test set. Numbers indicate the percentage of pixels which have greater than three pixels or 5% disparity error. 'D1-bg', 'D1-fg' and 'D1-All' refer to background pixels which contain static elements, dynamic object pixels and all pixels respectively. Although there is a gap between performance of supervised methods and ours, our model shows decent results and even beats DispNet on the D1-bg region.

	NOC			All			
Model	D1-bg	D1-fg	D1-All	D1-bg	D1-fg	D1-All	Runtime
DispNet [15]	4.11	**3.72**	4.05	4.32	**4.41**	4.34	**0.06**
Content-CNN [14]	3.32	7.44	4.00	3.73	8.58	4.54	1
MC-CNN [13]	2.48	7.64	3.33	2.89	8.88	3.89	67
GC-Net [16]	2.02	5.58	2.61	2.21	6.16	2.87	0.9
PSMNet [38]	**1.71**	4.31	**2.14**	**1.86**	4.62	**2.32**	0.41
Ours	3.86	15.89	5.84	4.20	16.97	6.33	0.9

shows a qualitative result. With semantic segmentation supervision, it corrects the wrongly estimated disparity on the center of the road, which is a region with high reflection.

2.4.5 Performance Analysis

In Table 2.3, we present details error rates on regions of each semantic segmentation class before and after adding smoothness loss and fusing segment embedding. We wish to delve into how segment embedding learned from semantic segmentation benefits disparity estimation. In the table, 'smo' refers to smoothness loss; 'seg' refers to segmentation loss; the first row shows the name of semantic classes; the second row shows error rates of the model with all components of losses except smooth loss and segmentation loss; the third row shows error rates of the model with all losses

Table 2.3: Error rates of disparity estimation on regions of each semantic class. In the table, 'smo' refers to smoothness loss; 'seg' refers to segmentation loss; the first row shows the name of semantic classes; the second row shows error rates of the model with all components of losses except smoothness loss and segmentation loss; the third row shows error rates of the model with all losses except segmentation loss; the fourth row shows error rates of the model with all losses; the final row shows percentage of error rate reduction after adding smoothness loss and segmentation loss. Smoothness loss improves performance on relatively large semantic classes but not on small semantic classes. With supervision of the semantic segmentation task, error rates on regions of small semantic classes decrease substantially.

Method	road	pole	car	tsign	bus	swalk	train	wall	build.	tlight	veg.	fence	truck	person	bike	terrain	rider	mbike
Model	2.65	11.26	12.94	6.33	2.31	5.53	1.25	2.45	14.82	2.34	10.60	11.34	6.23	1.52	2.51	6.06	1.06	0.34
Model(smo)	1.51	13.13	10.53	6.35	1.85	4.36	1.15	2.28	13.53	2.63	9.57	11.15	6.26	1.50	2.40	5.81	1.24	**0.33**
Model(smo&seg)	**1.35**	**7.62**	**8.83**	**4.67**	**1.77**	**4.25**	**0.99**	**2.09**	**13.05**	**2.06**	**9.36**	**10.13**	**5.76**	**1.43**	**2.36**	**5.27**	**1.04**	0.36
Improvement %	48.88	32.37	31.72	26.20	23.29	23.09	20.37	14.40	11.94	11.75	11.64	10.73	7.50	6.07	5.95	4.21	2.50	-6.38

except segmentation loss; the fourth row shows error rates of the model with all losses; the final row shows percentage of error rate reduction after adding smooth loss and segmentation loss. As shown in the table, the smoothness loss helps improve disparity estimation for large semantic classes but not for small semantic classes. For example, error rates on regions of large semantic classes like roads, cars and buses decrease substantially, but error rates on regions of small semantic classes, such as poles, traffic lights and traffic signs, actually increase after imposing smoothness loss. This is because, without guidance of semantic segmentation, smoothness loss tends to blindly force local disparity smooth and disparities for small objects are smoothed to their neighbors which results in more error.

However, with supervision of the semantic segmentation task, the model is able to learn semantic features. In this case, disparity smoothness loss will force the disparity to be smooth within segments with the same semantic meanings rather than blindly with neighboring segments. Thus, disparities for small objects will remain coherent. It is shown in the table that error rates on regions of poles, traffic lights, traffic signs and other small semantic classes decrease to the lowest level after supervision of the semantic segmentation task.

The focus of this work is on improving state-of-the-art for unsupervised disparity estimation guided by semantic segmentation. We also evaluate our method on semantic segmentation performance. Our baseline IoU is 47.6%. After disparity refinement, segmentation performance decreases slightly to 46.9% when evaluating on 40 validation images from KITTI 2015. This suggests that the disparity loss forces features to be different even within a semantic class.

2.4.6 Qualitative Results: 3D Models

We triangulate the disparity maps with the camera extrinsics into 3D point clouds with semantic labels as shown in Figure 2.4. We only consider pixels where disparities are above 5. Note that simultaneously calculating both disparity and semantic class

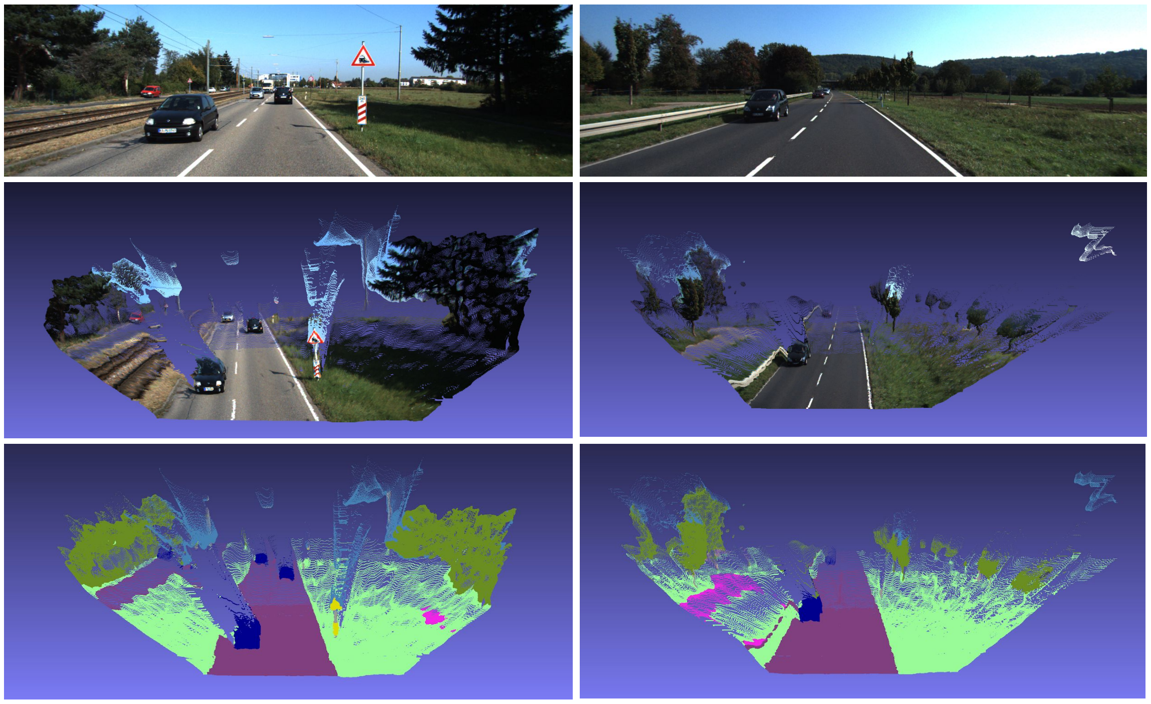
Table 2.4: Ablation study on loss components. Results of models with different losses are reported on KITTI 2015 training set without pretraining or post processing. There are two stages in our model. The superscript 'init' refers to losses in the initial stage and the superscript 'ref' refers to losses in the refinement stage. The subscript 'p' refers to photometric loss, 'r' refers to regularization loss, 'c' refers to consistency loss, 's' refers to smoothness loss and 'seg' refers to supervised semantic segmentation loss. Results shown here justify our two-stage architecture and designed components of total loss.

L_p^{init}	L_c^{init}	L_r^{init}	L_p^{ref}	L_c^{ref}	L_s^{ref}	L_{seg}	NOC pixels	All pixels
√	√	√					7.18	8.75
√	√	√	√	√			7.04	8.60
√	√	√				√	6.70	8.14
√	√	√	√	√	√		6.53	6.94
			√	√	√	√	5.99	6.42
√	√	√	√	√	√	√	**5.93**	**6.32**

enables us to efficiently produce semantic 3D models, which can be used more directly for driving tasks than other independent outputs.

2.5 Conclusion

We propose a model in which segment embedding learned from semantic segmentation is fused into the process for disparity estimation. This segment embedding is helpful for estimating disparity in ill-posed regions. We demonstrate the efficacy of our method on both KITTI and Cityscapes datasets. Our unsupervised method achieves comparable results to supervised methods on KITTI and even outperforms some of them in background regions. Outputting disparities and semantic segments simultaneously enables us to efficiently produce semantic segments in 3D space.


(a) Sample 3D semantic results on KITTI 2015. From top: left stereo input images, 3D cloud points, semantic segmentation on 3D point clouds.

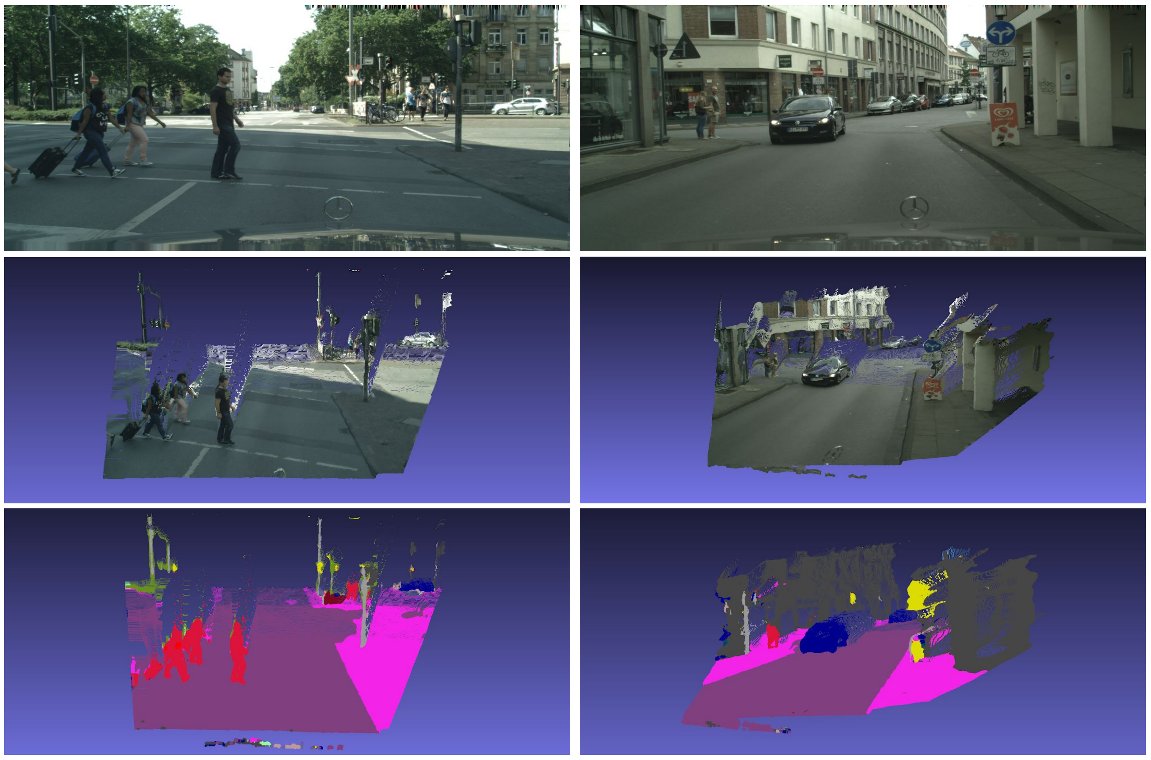

(b) Sample 3D semantic results on Cityscapes. From top: left stereo input images, 3D cloud points, semantic segmentation on 3D point clouds.

Figure 2.4: Sample qualitative 3D semantic results on KITTI 2015 and Cityscapes datasets. (a) 3D Reconstruction Results on KITTI 2015. (b)3D Reconstruction Results on Cityscapes. The last rows in both (a) and (b) show the 3D semantic results. Different color refers to different semantic class.

CHAPTER III

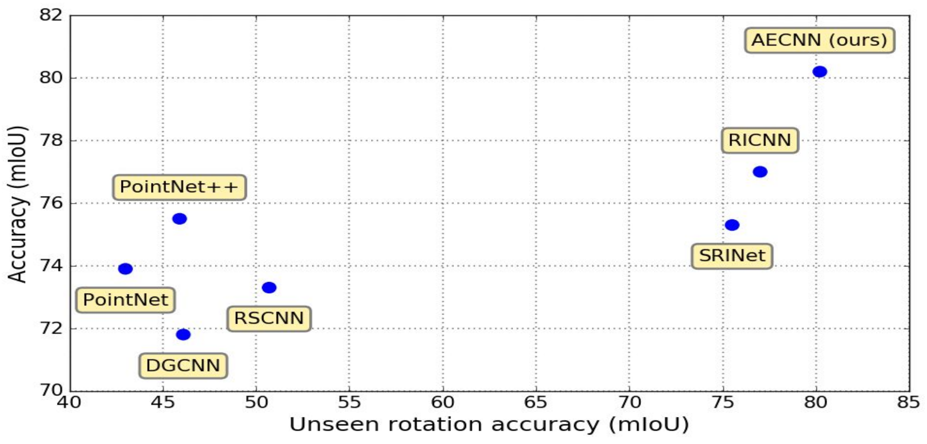
Learning Rotation-Invariant Representations of Point Clouds

Point cloud analysis is an area of increasing interest due to the development of 3D sensors that are able to rapidly measure the depth of scenes accurately. Unfortunately, applying deep learning techniques to perform point cloud analysis is non-trivial due to the inability of these methods to generalize to unseen rotations. To address this limitation, one usually has to augment the training data, which can lead to extra computation and require larger model complexity. In this chapter, we propose a new neural network called the Aligned Edge Convolutional Neural Network (AECNN) that learns a feature representation of point clouds relative to Local Reference Frames (LRFs) to ensure invariance to rotation. In particular, features are learned locally and aligned with respect to the LRF of an automatically computed reference point. The proposed approach is evaluated on point cloud classification and part segmentation tasks. This chapter illustrates that the proposed technique outperforms a variety of state of the art approaches (even those trained on augmented datasets) in terms of robustness to rotation without requiring any additional data augmentation. [1]

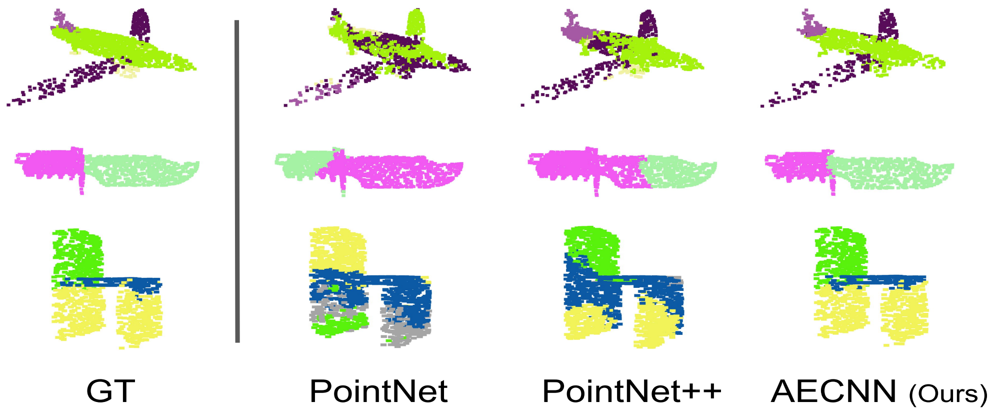
3.1 Introduction

The development of low-cost 3D sensors has the potential to revolutionize the way robots perceive the world. For this revolution to be realized, algorithms to interpret and classify the large volumes of point clouds generated by these sensors must be developed. To construct such algorithms, one could be inspired by the successes of deep learning approaches that robustly interpret 2D images in the presence of noise or lighting, rotation, and scaling variability. These deep learning approaches achieve impressive performance by relying on representations that enforce lighting, rotation, and scaling invariance. Unfortunately, the lack of a representation that is able to

[1] This chapter is based on [31]



(a)



(b)

Figure 3.1: An comparison of the performance of state-of-the-art techniques to the method developed in this paper while performing part segmentation on rotated point clouds from the ShapeNet dataset. We report results on models of arbitrary rotation during testing while they trained with only rotation along vertical direction (x axis) and on models of arbitrary rotation during both training and testing (y axis) (subfigure a). This degradation in performance can be seen on the quality of part segmentation in unseen views (subfigure b, where different colors correspond to different part categories).

enforce rotation invariance has hindered the application of deep learning techniques to analyze point clouds.

To address this challenge, researchers have typically converted point clouds into regular 2D [6, 56, 57, 58] or 3D [7, 8, 59] grids before applying CNNs to learn a meaningful representation. Unfortunately, this conversion process degrades the resolution of measured objects which can adversely affect point cloud analysis. More recently, PointNet [19] was proposed to preserve some of this geometric information by using a symmetric kernel that could enforce permutation invariance. This approach allowed a user to directly treat a point cloud as an input into a DNN and get a global vector representing the input point cloud as an output. This work was subsequently extended in a variety of ways to preserve local structure within a point cloud that proved to be important while performing classification [20, 22, 21]. However, the representations developed by these methods rely on an individual point's absolute position, which hinders their ability to develop algorithms that are invariant to rigid body transformations. Figure 3.1, for instance, illustrates the deficiency of these methods when they are applied to perform part segmentation on views that are unseen during training.

Typically, one can address this limitation and improve the robustness of DNNs to rigid body transformations by augmenting the training set with additional examples. However, this requires additional computation and increased model capacity. For instance, during classification, a model must learn a function that maps the same object under different rigid body transformations into a similar feature in a feature space. Rather than augment the training set, other approaches have focused on developing representations that can preserve rotational symmetry by directly converting point clouds into a spherical voxel grid and then extracting rotation-equivariant features [60, 61, 62]. Unfortunately this conversion still sacrifices resolution that can adversely affect point cloud analysis. To remedy this loss of information, others have proposed to represent point clouds relative to a local reference frame (LRF), which is determined by a local subset of a point cloud [63, 64]. Each point in a neighborhood of a constructed LRF is represented with respect to that LRF and a local feature is learned for each point set. Subsequently these local features are fused together to define global features. These LRF-based learned representations are invariant to rotation; however, as we illustrate in this chapter, the accuracy of techniques utilizing LRF-based representation are only marginally better than those utilizing an absolute coordinate based representation for point cloud classification tasks. This is in part because the learned local features for a pair of points are not aligned before they are fused together.

To address the limitations of existing approaches, this chapter proposes a novel 3D representation of point clouds that is invariant under rotation and introduces a new neural network architecture, the Aligned Edge Convolutional Neural Network (AECNN), to utilize this representation. As in prior work, we leverage the notion of LRFs to ensure that different orientations of a point cloud are mapped into the same representation. Each point in a neighborhood of a constructed LRF is represented with respect to that LRF before subsequent processing. This ensures that the model is able to learn internal geometric relationships between points rather than learning geometric relationships that are a function of the absolute coordinates of the points that may change after rotation. Our proposed AECNN architecture processes these local internal features and aligns them with local internal features drawn from other LRFs before fusing them together in a hierarchical fashion to define global features. Importantly, in contrast to prior work that utilizes a spherical coordinate system [63] or non-orthogonal basis [65], we construct a basis for the LRFs that is orthonormal. This ensures that feature alignment can be computed in a straightforward manner which makes the hierarchical fusion of local features tenable. The contributions of this work are three-fold:

- We propose a novel representation of points clouds that is invariant to arbitrary rotations.
- We propose a novel alignment strategy to align neighboring features within distinct LRFs. This makes it feasible to perform feature fusion within a hierarchical network, which makes a reasonable feature fusion between local and global features.
- We illustrate that our propose representation is robust to rotation and achieves state-of-the-art results in both point cloud classification and segmentation tasks.

3.2 Related Work

3.2.1 View-Based and Volumetric Methods

A variety of methods have represented 3D shape as a sequence of 2D images since they can leverage existing algorithms from 2D vision [1, 4, 66]. These view-based methods typically project a 3D shape onto 2D planes from different views. These different images are then processed by CNNs. Though these methods achieve good performance at classification tasks using off-the-shelf architectures and pre-trained model [6, 56, 57, 58], the projection of 3D shape onto 2D planes sacrifices 3D geometric structures that are critical during point cloud analysis.

Converting point clouds into 3D voxels can preserve some of this geometric information. The higher the resolution of quantization, the more geometric information is preserved. These converted point clouds can then leverage existing CNNs with 3D kernels [7, 8, 59]. Since points are only sampled from the surface of objects, regular quantization can waste valuable resolution on the empty space within or outside objects. Better partitioning methods, such as KD-tree [67] and Oct-tree [68, 69] have been proposed to address this limitation. In contrast to these methods, our approach works directly with point clouds without requiring any conversion.

3.2.2 Point Set Learning

Point set learning methods take raw point clouds as input. The pioneering work in this area is PointNet [19], which independently transforms each point and outputs a global feature vector describing the input point cloud by aggregation using a max pooling layer. Unfortunately, PointNet is unable to learn local structure over increasing scales, which is important for high-level learning tasks. Extensions that utilize hierarchical structure [20, 70], graph network [22, 71], or relation-aware features [21] have been proposed to preserve local geometric structure during processing. Other extensions that rethink the convolution operation to better accommodate point cloud processing have also been proposed [72, 73]. However, the representation that is learned by these approaches changes when the point clouds in the training set are rotated. As a result, these representations perform poorly when utilized during classification or segmentation tasks on rotated versions of point clouds that were not included during training.

3.2.3 Rotation Learning

Various methods have been proposed to either learn rotation-invariant or rotation-equivariant representations. For instance, spatial transformer networkss (STNs) have been proposed to learn rotation-invariant representations [19]. The STN learns a transformation matrix to align input point clouds without requiring that the alignment place the point clouds in some fixed ground-truth orientation. As a result, the transformation matrix is not guaranteed to align objects to a consistent orientation which restricts its utility. Other approaches achieve rotation invariance by relying on LRFs [63, 64, 65]. However, the difficulty of aligning the features learned with respect to LRFs, as described earlier, has limited the potential expressive capabilities of these techniques. Since designing a model that is invariant to rotation is difficult, a variety of methods have attempted to achieve rotation-equivariance using spherical convolutions [60, 61, 62]. Spherical convolutions require the spherical representation of a point cloud. Unfortunately, projecting 3D point clouds into 2D sphere results in a loss of information.

3.3 Learning Point Cloud with Rotation Invariance

This section introduces our proposed rotation-invariant representation (RIR) of point clouds using LRFs, our proposed aligned edge convolution designed for RIR, and the proposed hierarchical network architecture for classification and segmentation tasks.

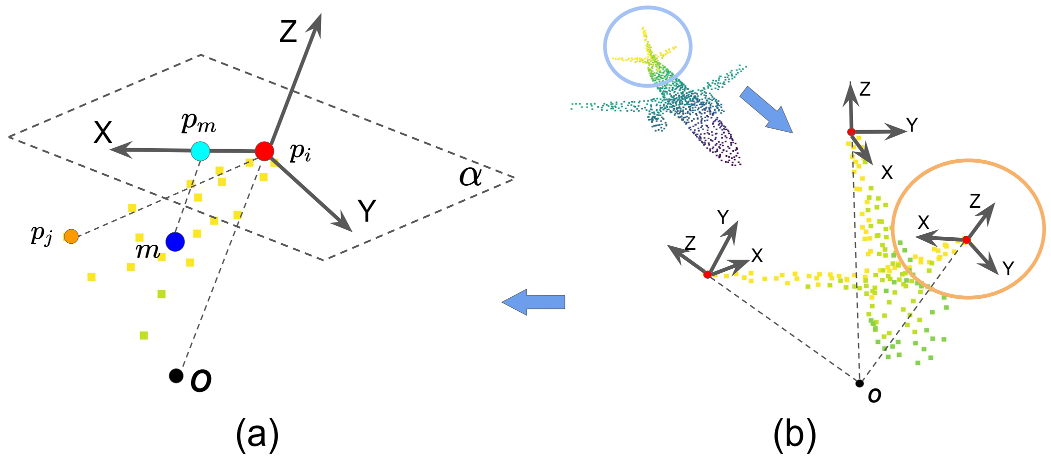


Figure 3.2: An illustration of aligned edge convolution and the LRFs that help define it. To construct the RIR, one takes the reference point p_i and the k-nearest points to it (subfigure a). The LRF is determined by p_i and the anchor point m which is defined as the barycenter of the k-nearest points to p_i. The coordinates of k-nearest points are described with respect to the LRF. Note that LRFs may not be aligned due to independence of the local neighborhood of points that define each LRF (subfigure b).

3.3.1 Rotation-Invariant Representation

To construct a representation that is invariant to rotation, we represent a point's coordinates relative to a LRF. Each LRF is defined using three orthonormal basis vectors and is designed to be dependent on local geometry, as is depicted in Figure 3.2 (a). To define this LRF, suppose we are given a reference point p_i in a point cloud and a set of neighboring points $\{p_1, \ldots, p_k\}$ to p_i in the point cloud whose coordinates are all described with respect to a coordinate system with global origin o. Note, we describe how to select these reference points in Section 3.3.3, and neighboring points are those within a certain radius to the reference point. Next, define an anchor point m as the barycenter of the neighboring points:

$$m = \frac{1}{k}\sum_{j=1}^{k} p_j, \tag{3.1}$$

and the plane α that is orthogonal to $\overrightarrow{op_i}$ and intersects with p_i. Using these definitions, we can define the projection of m onto the plane α:

$$\overrightarrow{p_ip_m} = \overrightarrow{om} - \frac{\overrightarrow{op_i}}{|\overrightarrow{op_i}|} \cdot \langle \overrightarrow{om}, \frac{\overrightarrow{op_i}}{|\overrightarrow{op_i}|} \rangle. \tag{3.2}$$

With this definition, we can construct the following coordinate axes for the LRF:

$$\vec{x} := \frac{\overrightarrow{p_ip_m}}{|\overrightarrow{p_ip_m}|}, \quad \vec{z} := \frac{\overrightarrow{op_i}}{|\overrightarrow{op_i}|}, \quad \vec{y} := \vec{z} \times \vec{x}. \tag{3.3}$$

Note that z axis is defined as the direction from global origin o pointing at p_i; the x axis is defined as the direction from p_i pointing at p_m; and the y axis is defined as the direction of cross product of z and x axis. The origin of LRF is at the reference point p_i. We assume that the global origin o is known, and in our case we use the center of point clouds.

We introduce rotation invariance by representing the set of neighboring points relative to their LRF:

$$t^i_j = (\langle p_{ij}, \vec{x}\rangle, \langle p_{ij}, \vec{y}\rangle, \langle p_{ij}, \vec{z}\rangle) \tag{3.4}$$

where $p_{ij} = p_j - p_i$ for each $p_j \in \{p_1, \ldots, p_k\}$. Note t^i_j is the RIR for the point p_j relative to the LRF at point p_i. We then use a PointNet structure to capture the geometry within the neighboring points using the RIR:

$$f(\{t^i_1, ..., t^i_k\}) = MAX(\{h(t^i_1), ..., h(t^i_k)\}) \tag{3.5}$$

where f is a learning function, which takes a point set as input, and outputs a feature vector representing input point clouds, h is a feature transformation function and is approximated by a MLP. Note, the max pooling layer aggregates information.

3.3.2 Aligned Edge Convolution

To capture the geometric relationship between points in a point cloud, the notion of edge convolution via DGCNN has been developed [22]. To understand how edge convolution works, suppose we are given a point cloud with n points, denoted by $P = \{p_1, ..., p_n\} \subseteq \mathbb{R}^3$, along with F-dimensional features corresponding to each point, denoted by $X = \{x_1, ..., x_n\} \subseteq \mathbb{R}^F$. Suppose we construct the k-NN graph (V, E) in the feature space, where $V = 1, ..., n$ and $E \subseteq V \times V$ are vertices and edges, then the edge convolution output at i-th vertex is given by:

$$x'_i = \underset{j:(i,j)\in E}{MAX}\ g(x_i, x_j - x_i) \tag{3.6}$$

where g is a MLP layer.

Note, edge convolution is essentially performing feature fusion. It fuses the global information captured by x_i with local neighborhood information captured by $x_i - x_j$. For right feature fusion, x_i and x_j must be learned in the same coordinate system. Unfortunately, it is nonviable to directly apply edge convolutions to features in our case. This is because in our case, x_i and x_j are learned relative to two different LRFs, and the LRFs of x_i and x_j may not be aligned, as is shown in the Figure 3.2 (b). As a result, applying edge convolution directly on our learned features may create inconsistent features.

To resolve this problem, we propose aligning x_j into the LRF of x_i before performing feature fusion. We call our approach, which is depicted in Figure 3.3, Aligned Edge Convolution (Aligned EdgeConv). To construct our approach, we begin by understanding the relationship between different LRFs, which can be described using a rotation R and translation T. To construct this rotation and translation, suppose the basis of the LRF for each feature is denoted by $E = \{e_1, ..., e_n\} \subseteq \mathbb{R}^{3\times3}$. Then the rotation matrix and translation vector can be computed by:

$$R_j^i = e_i \cdot e_j^{-1} = e_i \cdot e_j^\top \tag{3.7}$$

$$T_j^i = t_j^i \tag{3.8}$$

where e_j is an orthogonal matrix defined in (3.3) and t_j is defined in (3.4).

R and T describe the relationship between LRFs, so we use them to transform x_j into the LRF of x_i. Though it is easy to invert a rotation and translation in 3D, extending it to the high dimensional feature space that x_j lives in would be challenging. One option to resolve this problem is to apply an approach similar to the STN proposed in the PointNet wherein one predicts a transformation matrix from R_j^i and T_j^i and applies it to x_j:

$$\hat{x}_j = \phi(R_j^i, T_j^i) \cdot x_j \tag{3.9}$$

where ϕ is a MLP layer and outputs an $F \times F$ matrix compatible with x_j. Typically a regularization term is added to the loss during training to constrain the feature transformation matrix to be close to an orthogonal matrix. Another option is to take R_j^i, T_j^i and x_j as inputs and directly output a transformed feature:

$$\hat{x}_j = \phi(R_j^i, T_j^i, x_j) \tag{3.10}$$

In this paper, we utilize the second option. As we show in Section 3.4.5, the first option requires more graphics processing units (GPUs) memory and has more parameters. Therefore we update the (3.6) by:

$$x_i' = \underset{j:(i,j)\in E}{MAX}\, q(x_i, \hat{x}_j - x_i). \tag{3.11}$$

Similar to PointNet++ [20], we also include the RIR t_j^i in the edge convolution to maintain more information. So the aligned edge convolution is given by

$$x_i' = \underset{j:(i,j)\in E}{MAX}\, q(x_i, \hat{x}_j - x_i, t_j^i). \tag{3.12}$$

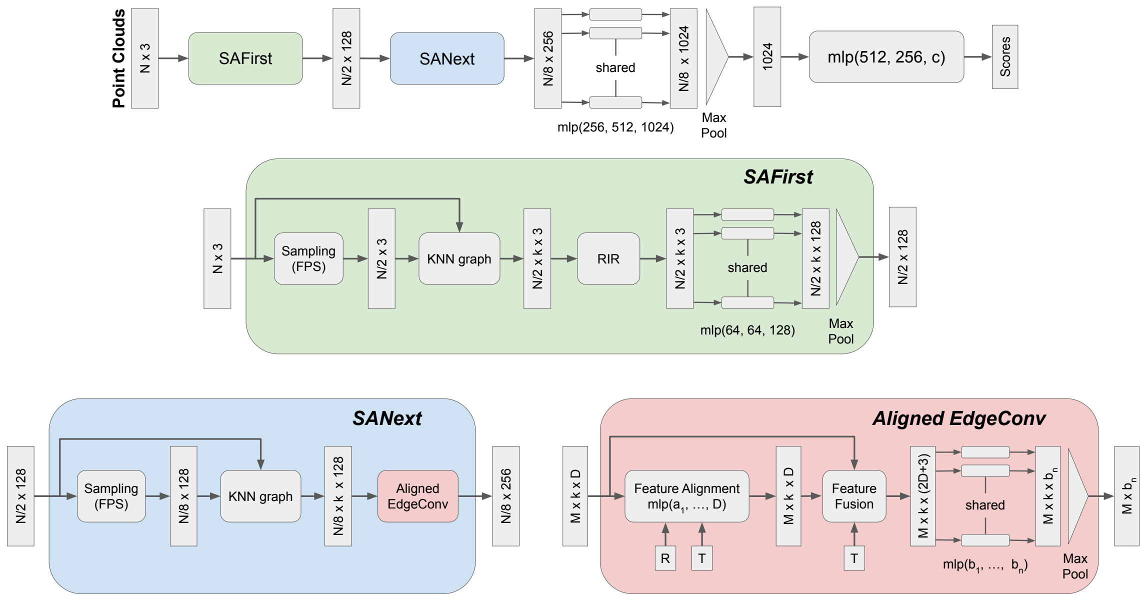


Figure 3.3: An illustration of the deep hierarchical architecture proposed in this paper to learn a rotationally invariant representation of point clouds for classification. The network takes N points as inputs and uses two SA blocks to hierarchically learn a representation of larger and larger regions. The final part aggregates features from the last SA block and outputs a feature vector encoding the input point set. The SAFirst block samples reference points and builds a k-NN graph in Euclidean space. It converts points into the RIR and transforms them using a shared-weights PointNet structure that outputs feature vectors encoding information for each reference point' neighbors. The SANext block applies a similar process to the SAFirst block except for building the k-NN graph in feature space and extracting features using Aligned EdgeConv. The feature alignment module within Aligned EdgeConv aligns local features to the LRF of the reference point using the rotation matrix R and the translation vector T that are defined in the Section 3.3.1. Then the aligned local features are fused with the feature at the reference point along with translation vector T. Note mlp in the figure is abbreviated for MLP. The output is classification scores for c classes.

3.3.3 Network Architecture

Our proposed network architecture that takes raw point clouds as input and learns a representation is depicted in Figure 3.3. Our architecture is inspired by techniques that perform local-to-global learning which has been successfully applied to 2D images [5] and has been shown to effectively extract contextual information. We exploit a hierarchical structure to learn both local and global representation. Our approach captures larger and larger local regions using two SA blocks, proposed by PointNet++ [20], and the global features are constructed by aggregating outputs from the last SA block using max pooling.

The two SA blocks each have distinct structures; however, they share the same processing pipeline: sampling, grouping, and processing. The structure of the first SA block (SAFirst) is illustrated in the green block in Figure 3.3. Given the input point clouds, we use farthest point sampling (FPS) to subsample the point cloud while preserving its geometric structure. The constructed points serve as the reference points during the construction of a k-NN graph. The k-NN graph is computed in Euclidean space, and it is used to generate the RIR described in Section 3.3.1. A shared-weights PointNet structure then processes the RIRs for each local set of points and outputs a feature vector describing the set of points near each reference point.

The structure of SANext is shown within the blue block in Figure 3.3. The sampling and grouping strategy in SANext is identical to the one in SAFirst except that one quarter the number of points are selected as reference points and the k-NN graph is dynamically updated and computed in the feature space, which has been shown to be more beneficial and have larger receptive fields than a fixed graph version [22]. Then the proposed aligned edge convolution extracts features encoding larger local regions than the previous SA block. The rotation matrix R and translation vector T, which can be derived from basis and positions of LRFs, are fed into a feature alignment module to align local features to the frame of reference point. Essentially, the SANext is the building block that can be used to iteratively capture larger and larger local regions.

Note for the segmentation task, we require a feature for each point. We adopt a similar strategy to PointNet++ [20], which propagates features from subsampled points to original points. Specifically, the interpolated features from the previous layer are concatenated with skip linked features output from SA blocks. The interpolation is done via the inverse distance weighted average based on k-NN. Importantly, the proposed feature alignment idea is also integrated in this feature propagation pipeline.

3.4 Experiments

This section describes how we implement the network described in Section 3.3.3 and how we validate its utility. First, we evaluate our method on a shape classification task (Sec 3.4.2) and part segmentation task (Sec 3.4.3). Second, we evaluate the design of our LRFs (Sec 3.4.4) and illustrate the effectiveness of the proposed aligned edge convolution (Sec 3.4.5).

3.4.1 Implementation details

We implement our network in PyTorch. All experiments are run on a single NVIDIA Titan-X GPU. During optimization, we use the Adam optimizer with batch size 32. Models are trained for 250 epochs. The learning rate starts with $1e{-}3$ and scale by 0.2 every 100 epochs.

In some experiments, we augment the dataset with arbitrary rotations. However,

Method	Inputs	Input size	Y/Y	Y/AR	AR/AR
MVCNN 80x [56]	view	80×224^2	90.2	81.5	86.0
Spherical CNN [60]	voxel	2×64^2	88.9	76.9	86.9
PointNet [19]	point	1024×3	88.5	21.8	83.6
PointNet++ [20]	point	1024×3	89.3	31.7	84.9
RS-CNN [21]	point	1024×3	89.6	24.7	85.2
DG-CNN [22]	point	1024×3	91.7	31.5	88.0
RI-CNN [64]	point	1024×3	86.5	86.4	86.4
SRINet [65]	point	1024×3	87.0	87.0	87.0
ClusterNet [63]	point	1024×3	87.1	87.1	87.1
SF-CNN [61]	point	1024×3	**92.3**	84.8	90.1
Ours	point	1024×3	91.0	**91.0**	**91.0**

Table 3.1: Classification results on ModelNet40 dataset. We report the accuracy (%) in three different settings: training and testing with rotation along the vertical direction (Y/Y), training with rotation along vertical direction and testing with arbitrary rotation (Y/AR), and performing arbitrary rotation during training and testing (AR/AR). Though our model is only the third best performer in the Y/Y setting it is the top model in each of the other categories. In particular our proposed model has superior performance in the Y/AR and AR/AR, which means that it can generalize well to unseen rotations.

it is impossible to cover all rotations in 3D space. Similar to ClusterNet [63], we uniformly sample possible rotations. Each rotation is characterized by a rotation axis v and a rotation angle θ that is given by:

$$R = I + (\sin\theta)K + (1 - \cos\theta)K^2 \tag{3.13}$$

where K denotes the cross-product matrix for the rotation axis v which has a unit length and I is the identity matrix. In the experiments, we sample 3-dimensional vectors from a normal distribution and normalize v to be a unit vector.

We follow the approach presented in [60] to perform experiments in three different settings: 1) training and testing with rotation along the vertical direction (Y/Y), 2) training with rotation along vertical direction and testing with arbitrary rotation (Y/AR) and 3) performing arbitrary rotation during training and testing (AR/AR). The last two settings in particular are used to evaluate the generalization ability of the model under unseen rotations.

3.4.2 Shape Classification

One of the primary point clouds analysis tasks is to recognize the category of point clouds. This task requires a model to learn a global representation.

Dataset. We evaluate our model on ModelNet40, which is a shape classification benchmark [59]. It provides 12,311 CAD models from 40 object categories, and there are 9,843 models for training and 2,468 models for testing. We use their corresponding point clouds provided by PointNet [19], which contain 1024 points in

Method	Input size	Y/Y	Y/AR	AR/AR
PointNet [19]	2048×3	79.3	43.0	73.9
PointNet++ [20]	2048×3	**80.6**	45.9	75.5
DG-CNN [22]	2048×3	79.2	46.1	71.8
RS-CNN [21]	2048×3	80.0	50.7	73.3
RI-CNN [64]	2048×3	-	75.3	75.5
SRINet [65]	2048×3	77.0	77.0	77.0
Ours	2048×3	80.2	**80.2**	**80.2**

Table 3.2: Part segmentation results on ShapeNet dataset. Point cooridnates are taken as inputs, and mIoU across all classes is reported in three different settings including Y/Y, Y/AR and AR/AR. Our model outperforms all approaches except PointNet++ in Y/Y setting. Our model has superior performance in the Y/AR and AR/AR settings, which means that it can generalize well to unseen rotations.

each point clouds. During training we augment the point clouds with random scaling in the range $[-0.66, 1.5]$ and random translation in the range $[-0.2, 0.2]$ as in [67]. During testing, we perform ten voting tests while randomly sampling 1024 points and average the predictions.

Point clouds classification. We report the results of our model and compare it with other approaches in the Table 3.1. Three different training and testing setting are performed, which are introduced in Section 3.4.1. All approaches, except for the last five which are specially designed for rotation learning, perform well in the Y/Y setting, but experience a significant drop in accuracy when evaluated on unseen rotation as shown in the Y/AR setting. We conclude that these approaches only generalize well to rotations that they are trained on. However, our proposed method performs equally well across all three settings. Every evaluated approach has lower accuracy in the AR/AR setting than the Y/Y setting, except for RI-CNN, SRINet, ClusterNet, and our proposed method. This is most likely due to the difficulty of mapping identical objects in different poses to a similar feature space. This requires larger model complexity and is difficult to address with just dataset augmentation. RI-CNN (86.4%), SRINet (87.0%) and ClusterNet (87.1%) are specially designed for rotation invariance. Our model has superior performance in the setting of Y/AR and AR/AR (91.0%), which means our proposed method generalizes well to unseen rotations.

3.4.3 Part Segmentation

The part segmentation task requires assigning each point in a point cloud a category label. Since this is a point-wise classification task, part segmentation is typically more challenging than classification.

Dataset. We evaluate our model on ShapeNet part dataset [74], which con-

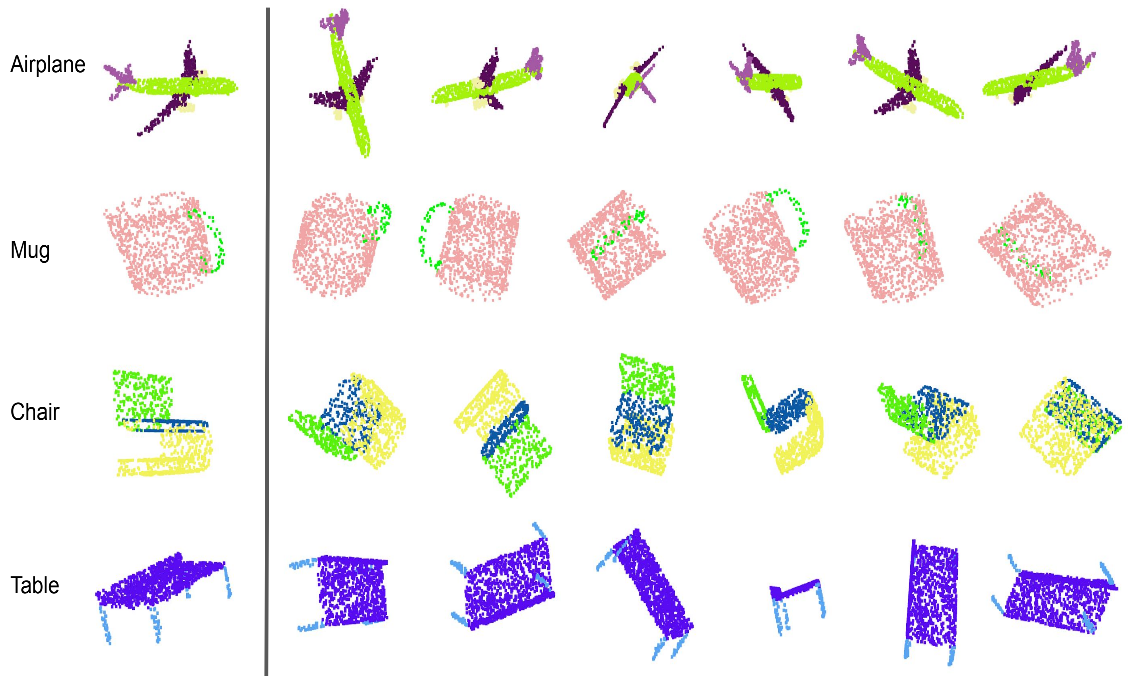


Figure 3.4: Qualitative results of our proposed method on part segmentation task on the ShapeNet dataset. From top to bottom, segmentation results of different categories are shown. From left to right, we show ground truth label and results when the input point clouds are arbitrarily rotated during testing. Different colors correspond to different part categories. Our model is robust to arbitrary rotations of the input point clouds.

tains 16,881 shapes from 16 categories and annotated with 50 parts in total. We split the dataset into training, validation and test sets following the convention in PointNet++ [20]. 2048 points are randomly picked on the shape of objects. We concatenate the one-hot encoding of the object label to the last feature layer in the model as in [20]. During evaluation, mean inter-over-union (mIoU) that are averaged across all classes is reported.

3D part segmentation. We report the result of our model and compare it with other approaches in Table 3.2 and (subfigure a) in the Figure 3.1. Results align well with performance in the classification task. In addition, our method outperforms other approaches in all three settings, except for PointNet++ which is slightly better than our proposed method in the Y/Y setting. The consistent performance of our method in all three settings demonstrates good generalization to unseen rotations. Qualitative results of part segments are illustrated in Figure 3.4. The comparison results with other approaches in the Y/AR setting are also visualized in Figure 3.5.

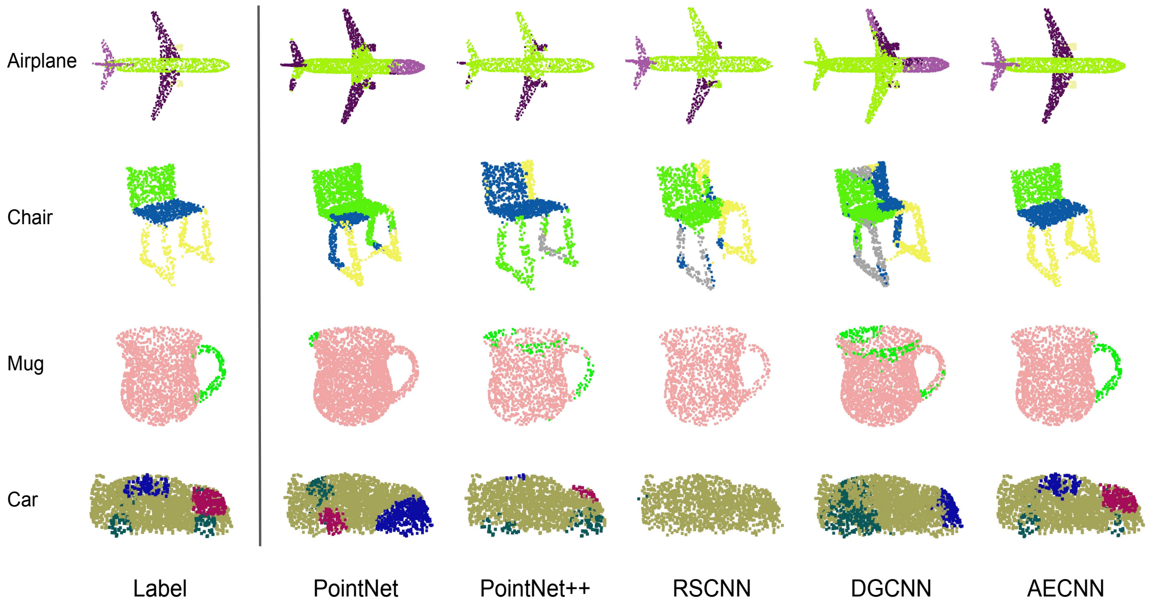


Figure 3.5: Qualitative results of part segmentation on ShapeNet. The Y/AR setting is adopted for all models. From top to bottom, segmentation results of different categories are shown. From left to right, we show ground truth label and the results from different approaches. Different colors correspond to different part categories. Our method achieves state-of-the-art performance while other approaches fail to generalize to unseen rotation.

Method	EdgeConv	AEConv1	AEConv2	AEConv3
Acc.	89.6	90.2	48.5	91.0
Para.	1.94M	2.14M	-	1.99M
FLOPs	4170M	6393M	-	4841M

Table 3.3: Ablation study on aligned edge convolution and comparison with original edge convolution. Accuracy, number of parameters and FLOPs per sample are reported. No further experiments were done on AEConv2 due to its poor accuracy. The number of neighbors is 48.

3.4.4 Analysis on Local Reference Frame

We achieve rotation invariance by expressing points coordinates respect to LRFs. Note that the LRF is handcrafted rather than learned from raw data. As a result, one may be concerned about the effectiveness of the designed LRF [75]. Currently, common ways of designing LRFs use the eigenvectors of the covariance matrix of the local point set [76, 77] or rely on the surface normal as the reference axis [78]. However, computing eignvectors for all points in some local neighborhood of points is time-consuming and estimating an accurate surface normal from point clouds is still challenging. In this study, we consider several alternative ways to compute LRFs and illustrate their effects on performance of models.

Searching		Grouping		# neighbors				Acc.
knn	ball	Mean	Max. D	10	16	32	48	
	✓	✓					✓	90.4
	✓		✓				✓	90.3
✓			✓				✓	90.3
✓		✓					✓	**91.0**
✓		✓		✓				89.6
✓		✓			✓			90.3
✓		✓				✓		90.8

Table 3.4: Ablation study on LRFs. Models are evaluated on ModelNet40 dataset in the Y/AR setting. Three aspects which effect LRFs are studied: searching methods, grouping ways and the number of neighbors.

Radii	(0.1, 0.2)	(0.1, 0.4)	(0.2, 0.2)	(0.2, 0.4)
Acc.	89.1	89.3	90.3	**90.4**

Table 3.5: Ablation study on radius in ball query. Models are evaluated on ModelNet40 dataset in the Y/AR setting. Model's performance is marginally sensitive to radius if ball query is used. (r1, r2) indicates the radius in the first SABlock and the second SABlock respectively.

The definition of LRF is introduced in the Section 3.3.1. Note that given a reference point, the only variation in the definition of the LRF arises from the x axis. Because the z axis is defined as the direction from the global origin to the reference point, and the y axis is defined by the cross product of the z and x axis. Recall that the x axis is associated with the anchor point m, which is shown in the Figure 3.2. Here we investigate three different aspects that influence the determination of the anchor point: searching methods, ways of grouping the data, and the number of neighbors. We perform experiments on shape classification, and Table 3.4 shows the results.

We compare two searching methods: ball query and k-NN search. Ball query finds all points within a certain radius to the reference point, but only up to k points are considered in the experiments. k-NN finds a fixed k nearest points to the reference point. As is shown in the first four rows of Table 3.4, given the same grouping method models with k-NN search have equal or higher accuracy than ball query. However, the performance of the ball query method is marginally sensitive to the size of the radius chosen, which needs to be assigned manually, as is shown in Table 3.5.

We study two ways of determining the anchor point: we define the anchor point as the mean of neighboring points or as the point with largest projected distance to reference point on plane α shown in the Figure 3.2 (b). From rows three and four in Table 3.4, we conclude that anchor points computed from mean of neighbors is preferred (91.0%) over anchor points with largest projection distance (90.3%). The last four rows of Table 3.4 illustrate the performance of our model with different numbers of neighboring points. We find performance drops with decreasing of k and the model with 48 nearest points achieves the best performance (91.0%). Further increasing the number of nearest points leads to extra computation burden.

3.4.5 Aligned EdgeConv Analysis

The proposed AECNN is specially designed to learn a rotation-invariant representation. Recall that we align features before doing feature fusion within the edge convolution and call it aligned edge convolution (AEConv). This study compares the original edge convolution which does not perform feature alignment [22] with our proposed aligned edge convolution. We also experiment with different strategies for doing alignment. The results are shown in Table 3.3.

We report three different strategies for alignment: transforming the source feature by a transformation matrix in the feature space (AEConv1), which is defined in (3.9); taking source feature x_j, LRF e_i of source point, LRF e_j of reference point and translation T as inputs to predict the aligned feature (AEConv2); taking source x_j along with rotation matrix R and translation T as inputs to predict the aligned feature (AEConv3), which is defined in (3.10). Our proposed feature alignment idea is verified by comparison between the first column and the last columns, where aligned edge convolution has higher accuracy (91.0 %) than the original edge convolution (89.6 %) which has no alignment process. AEConv1 (90.2 %) also outperforms edge convolution, but it loses slightly to AEConv3. Due to limited GPU memory, we need to reduce the number of learning kernels within the SAFirst block of AEConv1, so that it can be fed into a single GPU during training. Even in this case, AEConv1 still has more parameters (2.14M) to learn than AEConv3 (1.99M) and more FLOPs per sample during the test (6393M) than AEConv3 (4841M). Additionally, AEConv2 is not able to converge.

3.5 Conclusions

This work proposes AECNN, or the Aligned Edge Convolutional Neural Network, which addresses the challenges of learning rotationally-invariant representations for point clouds. Rotation invariance is achieved by representing points' coordinates relative to local reference frame. The proposed AECNN architecture is designed to better extract and fuse information from local and global features. In this way, the AECNN architecture is able to generalize well to unseen rotation. Extensive experiments are performed in classification and segmentation and demonstrate effectiveness of AECNN.

CHAPTER IV

Point Set Voting for Partial Point Cloud Analysis

The continual improvement of 3D sensors has driven the development of algorithms to perform point cloud analysis. In fact, techniques for point cloud classification and segmentation have in recent years achieved incredible performance driven in part by leveraging large synthetic datasets. Unfortunately these same state-of-the-art approaches perform poorly when applied to incomplete point clouds. This limitation of existing algorithms is particularly concerning since point clouds generated by 3D sensors in the real world are usually incomplete due to perspective view or occlusion by other objects. In this chapter, we propose a general model for partial point clouds analysis wherein the latent feature encoding a complete point cloud is inferred by applying a point set voting strategy. In particular, each local point set constructs a vote that corresponds to a distribution in the latent space, and the optimal latent feature is the one with the highest probability. This approach ensures that any subsequent point cloud analysis is robust to partial observation while simultaneously guaranteeing that the proposed model is able to output multiple possible results. This chapter illustrates that this proposed method achieves the state-of-the-art performance on shape classification, part segmentation and point cloud completion. [1]

4.1 Introduction

The quality of 3D sensors has rapidly improved in recent years as their ability to accurately and quickly measure the depth of scenes has surpassed vision-based methods [79, 16, 30]. This improved accessibility to point clouds demands the development of algorithms to interpret and analyze them. Inspired by the success of DNNs in solving 2D image analysis tasks, approaches with DNNs have been successfully applied to perform similar point cloud analysis tasks such as shape classification and part segmentation [20, 73, 19, 21, 72, 22, 80]. These DNNs methods achieve the

[1]This chapter is based on [32]

state-of-the-art performance on these point cloud analysis tasks by learning representations from large synthetic datasets constructed from sampling the surfaces of CAD objects [59, 81]. Unfortunately, since point clouds generated from 3D sensors in the real-world scenarios are often incomplete, these approaches struggle when tasked to perform analysis on partial point clouds. Since real-world point clouds are often incomplete due to perspective of view or occlusions, this shortcoming of existing point cloud analysis methods is particularly cumbersome.

To address the limitations of existing approaches, this chapter proposes a novel model for partial point clouds analysis. We model the point clouds as a partition of point sets. Each local point set independently contributes to infer the latent feature encoding the complete point cloud. In contrast to prior work on learning a representation of all points within the point clouds, we utilize an encoder to embed each local point set. All embeddings vote to infer a latent space characterized by a distribution. As we show in this chapter, giving each local point set a vote ensures that the model has the ability to address the incomplete nature of point clouds. Inspired by recent progress in variational inference [82, 83], we output a distribution for the latent variable and then use a decoder to generate a prediction from the latent value with the highest probability. In particular, each local point set generates a Gaussian distribution in the latent space and independently votes to form the distribution of the latent variable. This voting strategy ensures that the model outputs more accurate predictions when more partial observations are given, and the probabilistic modeling enables the model to generate multiple possible outputs. The contributions of this chapter are:

- We propose that each local point set independently votes to infer the latent feature. This voting strategy is shown to be robust to partial observation;
- We propose to construct each vote as a distribution in the latent space and this distribution modeling allows for diverse predictions;
- The proposed model trained with complete point clouds using the proposed training strategy performs robustly on partial observation at test, which reduces the cost of collecting large partial point clouds dataset;
- The proposed model achieves state-of-the-art results on shape classification, part segmentation, and point clouds completion. In particular, it outperforms approaches trained with pairs of partial and complete point clouds on point clouds completion.

4.2 Related Work

To perform point cloud analysis, researchers have traditionally converted point clouds into 2D grids or 3D voxels since they can leverage existing CNNs. With the help of CNNs, those approaches achieve impressive results on 3D shape analysis [6, 7, 8]. Unfortunately, these 2D grid or 3D voxel representations degrade the resolution of objects. Researchers have attempted to address this issue by utilizing sparse representations [69, 68]. However, these representations are still less efficient than point clouds and are unable to avoid quantization effects. More recently, PointNet [19] has pioneered the approaches of directly taking point clouds as inputs and processed them using DNNs. To accomplish this objective, it uses a symmetric function to aggregate information from each point which is transformed to a high-dimensional space. A variety of extensions have been applied to PointNet [20, 21, 22, 84]; however, none of them is robust to the partial observation that is common in real-world scenarios.

To address this challenge posed by partial observations, researchers have relied on training DNNs on partial point clouds collected in real-world scenarios [26, 25, 85, 28]. Each of these approaches rely on networks that are proposed to perform feature extraction on complete point clouds. Unfortunately, collecting and annotating those partial point cloud datasets are expensive. Another approach seeks to first infer the complete data of the partially observed point clouds before later analysis. A common pipeline to perform this completion first encodes partial observations into a feature vector and then decodes it to complete point clouds. A variety of methods have been proposed for designing the decoder [86, 87, 88, 89, 90, 91]. However, each of these methods outputs a single prediction given partial shapes and lacks the ability to generate multiple plausible results. One notable exception is able to generate diverse results by modeling the spatial distribution of all points [87]. Unfortunately, this approach is only able to address partial point clouds from specific locations. In contrast, the method developed in this chapter has no such requirement on an observed partial point cloud, and leverages the distribution over the latent space to generate diverse predictions.

Hough transform and its variations have been applied to solve many problems, such as pattern detection, object detection and pose estimation [92, 93, 94, 95, 96, 97]. Some recent work extend it to 3D object detection and demonstrates that hough voting is well suited for 3D point clouds analysis [26]. Specifically, each local point set votes for the center of the object, and only those votes which locate within the cluster are considered for later process. Similarly, we take advantage of this voting strategy to accumulate small bits of partial information to form a confident latent

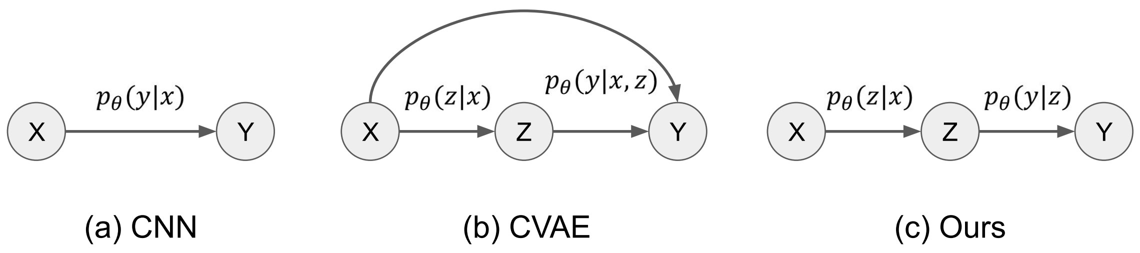


Figure 4.1: An illustrative comparison between our proposed method and other conditional graphical models (CGMs).

feature. However, we model each vote as a distribution in the latent space instead of a deterministic feature vector. This probabilistic modeling enables the model to learn the weights of votes when predicting latent features and is also useful to generate multiple possible outputs if needed.

The variational autoencoder (VAE) is one of the popular methods to model a generative distribution [82]. It assumes a prior distribution of latent variables, which is often a Gaussian distribution. More recently, the conditional variational autoencoder (CVAE) was proposed to extend the VAE by modeling the conditional distribution [98]. Unfortunately, directly applying a CVAE to partial point clouds requires a collection of annotated partial point cloud datasets for training. This work addresses this limitation by proposing each local point set serve as the unit voter to contribute to the latent feature. Encoding features learned for local point sets of complete point clouds can be leveraged for embedding local point sets of partial point clouds at test, which allows us to train on complete point clouds and perform on partial point clouds at test.

4.3 Method

This section describes the method we use to accomplish the aforementioned objective.

4.3.1 Preliminary: Conditional Variational Auto-encoder

A CVAE [98] is a directed graphical model. The conditional generative process of CVAE is as follows: for a given observation $\mathbf{x}$, a noise vector $\mathbf{z}$ is drawn from the prior distribution $p_\theta(\mathbf{z}|\mathbf{x})$, and an output $\mathbf{y}$ is generated from the distribution $p_\theta(\mathbf{y}|\mathbf{x};\mathbf{z})$. The training objective, variational lower bound, of CVAE is written as follows:

$$\log p_\theta(\mathbf{y}|\mathbf{x}) \geq \mathcal{L}_{\phi,\theta}(\mathbf{y}|\mathbf{x}) = -\mathbf{KL}(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})||p_\theta(\mathbf{z}|\mathbf{x})) + E_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}[\log p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z})] \tag{4.1}$$

The CVAE is composed of recognition network $q_\phi(\mathbf{z}|\mathbf{x};\mathbf{y})$, prior network $p_\theta(\mathbf{z}|\mathbf{x})$, and generation network $p_\theta(\mathbf{y}|\mathbf{x};\mathbf{z})$. In this framework, the recognition network $q_\phi(\mathbf{z}|\mathbf{x};\mathbf{y})$ is used to approximate the prior network $p_\theta(\mathbf{z}|\mathbf{x})$. All distributions are modeled using neural networks. During training, the reparameterization trick [82] is applied to propagate the gradients of ϕ and θ through the latent variables $\mathbf{z}$.

4.3.2 Proposed Point Cloud Model

We model the point clouds $\mathbf{x}$ as an overlapping partition of point sets, denoted by $\{x_1, x_2, ..., x_n\}$ if there are n point sets in the partition. In the simplest setting, each point is described by just its 3D coordinates, i.e. $x_i \subseteq \mathbb{R}^3$. Each point set is defined by a centroid and scale, as is shown in the Figure 4.2. To evenly cover the whole point clouds, we use Farthest Point Sampling (FPS) [20] algorithm to sample centroids. The number of centroids and the scale are manually set.

4.3.3 Proposed Method

In contrast to CVAEs, in which the generation network takes $(\mathbf{x}, \mathbf{z})$ as inputs, we model the generation process as a Markov chain, as is shown in the Figure 4.1. Specifically, given the latent variable $\mathbf{z}$ sampled from $p(\mathbf{z}|\mathbf{x})$, $\mathbf{y}$ is independent on $\mathbf{x}$. As a result, the generation of the output $\mathbf{y}$ satisfies the following equation: $p(\mathbf{y}|\mathbf{x},\mathbf{z}) = p(\mathbf{y}|\mathbf{z})$. The variational lower bound of this model is written as follows:

$$\mathcal{L}_{\phi,\theta}(\mathbf{y}|\mathbf{x}) = -\mathbf{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) + E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{y}|\mathbf{z})] \tag{4.2}$$

One problem with this learning framework is that the generation network $p_\theta(\mathbf{y}|\mathbf{z})$ takes values sampled from the recognition network $q_\phi(\mathbf{z}|\mathbf{x})$ at training while takes values sampled from the prior network $p_\theta(\mathbf{z}|\mathbf{x})$ at testing. This makes training inconsistent with testing. Similar to [98], we force consistency between these settings by making the recognition network $q_\phi(\mathbf{z}|\mathbf{x})$ the same as the prior network $p_\theta(\mathbf{z}|\mathbf{x})$. By doing this, $\mathbf{z}$ can be drawn from the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ at both training and testing, and the KL divergence term becomes zero. We approximate the resulting version of Equation (4.2) with the Monte Carlo estimator formed by sampling $\mathbf{z}^{(i)}$ from the recognition network $q_\phi(\mathbf{z}|\mathbf{x})$:

$$\mathcal{L}_{\phi,\theta}(\mathbf{y}|\mathbf{x}) \approx \frac{1}{L}\sum_{i=1}^{L} \log p_\theta(\mathbf{y}|\mathbf{z}^{(i)}),\ \mathbf{z}^{(i)} \sim q_\phi(\mathbf{z}|\mathbf{x}). \tag{4.3}$$

Recall that in our case $\mathbf{x}$ is modeled as a set of local point sets $\{x_i\}_{i=1}^n$. Thus, we propose to generate a vote for each of them, all of which are used to compute the latent variable $\mathbf{z}$. This voting strategy is inspired by the Hough transform [92] and VoteNet [26], and it is shown to accumulate small bits of partial information and output confident predictions. We model each vote as a distribution in the latent space. By assuming the independence of each vote, the recognition network $q_\phi(\mathbf{z}|\mathbf{x})$ can be expanded as follows:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \prod_{i=1}^{n} q_\phi(\mathbf{z}|x_i). \tag{4.4}$$

In the experiments, we assume $q_\phi(\mathbf{z}|x_i)$ is a Gaussian distribution characterized by mean vector and covariance matrix, which enables the use of a closed-form solution to optimizing $q_\phi(\mathbf{z}|\mathbf{x})$ with respect to $\mathbf{z}$. We denote the maximizing argument of $q_\phi(\mathbf{z}|\mathbf{x})$ by $\mathbf{z}_{opt}$. Equation (4.3) is equivalent to estimation with a single point when setting $L = 1$. Combining this with the highest probability sample of $\mathbf{z}$, given by $\mathbf{z}_{opt}$, the objective function can be further written as follows:

$$\mathcal{L}_{\phi,\theta}(\mathbf{y}|\mathbf{x}) \approx \log p_\theta(\mathbf{y}|\mathbf{z}_{opt}), \quad \mathbf{z}_{opt} = \underset{z}{\operatorname{argmax}} \prod_{i=1}^{n} q_\phi(\mathbf{z}|x_i). \tag{4.5}$$

Previous variational models choose latent features by sampling. However, $\mathbf{z}_{opt}$ in our case can be computed directly (shown in A.1), so all operations are differentiated with respect to the parameters θ and ϕ, which means that the reparameterization trick is no longer needed.

Note that the loss function differs as the task changes. A common softmax cross-entropy loss is used for classification and segmentation whereas the Chamfer distance [86] is used for training models on point cloud completion task. After training, the generative process is as follows: for the given observation $\mathbf{x}$, $\mathbf{z}_{opt}$ is the result of voting from $\{q_\phi(\mathbf{z}|x_i)\}_{i=1}^n$, and then the output $\mathbf{y}$ is generated from $p_\theta(\mathbf{y}|\mathbf{z}_{opt})$. The use of $\mathbf{z}_{opt}$ produces a single deterministic prediction. Diverse predictions are generated by instead sampling $\mathbf{z}^{(i)} \sim q_\phi(\mathbf{z}|\mathbf{x})$ followed by applying the generation network as in the deterministic case.

4.4 Experiment

4.4.1 Implementation

Network architecture We implement our network in PyTorch and use PyTorch Geometric Library [99]. The architecture of the proposed model is illustrated in the Figure 4.2. We use DNNs to model $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{y}|\mathbf{z})$. A shared-weights network is

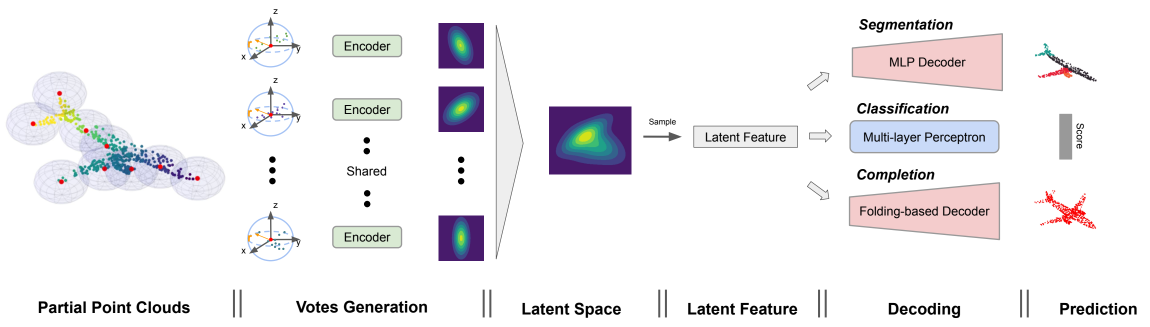


Figure 4.2: An illustration of the model developed in this work. The input point clouds are modeled as a partition of point sets and each local point set is defined by its centroid and scale r. Local point sets are embedded by a shared-weight PointNet encoder. The voting via this encoder is used to infer the space for latent features. The latent feature with the highest probability is sampled from the inferred latent space and is then passed to a decoding module for prediction. We perform experiments on three tasks: classification, part segmentation and point clouds completion.

used to represent $q_\phi(\mathbf{z}|x_i)$. Given the local point set, we represent each point within the local region relative to the centroid and then use a shared-weight PointNet as the basic feature extractor to encode the local region. Both the encoding feature and coordinates of centroid are processed by a MLP layer and it outputs a distribution of the latent space. We assume a simple case of multivariate Gaussian distribution with diagonal covariance matrix, so the output from the MLP consists of two vectors representing the mean and diagonal elements in the covariance matrix respectively. Different downstream tasks correspond to different networks for modeling $p_\theta(\mathbf{y}|\mathbf{z})$ as is shown in Figure 4.2. A folding-based decoder [88] is used for point clouds completion.

We use similar notations as PointNet++ [20] to describe the network architecture of the proposed model. $SA(k, r, [l_0, l_1 ..., l_d])$ is a set abstraction (SA) level with k local regions of ball radius r using a shared-weights PointNet structure, which contains d fully connected layers $l_i (i = 1, ..., d)$ and l_0 is the width of inputs. $FC([l_0, l_1], dp)$ represents a fully connected layer with input width l_0, output width l_1, and dropout ratio dp. All layers are followed by a batch normalization [52] layer and a Leaky ReLU [100] layer except for the last prediction layer, last layer in the vote generation, and layers within the folding-based decoder.

Point coordinates are first transformed to a high-dimensional space by a fully connected layer. For all experiments, the architecture for generating votes is the same. The outputs from it are the concatenation of two vectors, representing the mean and variance respectively, as we model each vote as a multivariate Gaussian distribution:

$$FC([3, 64]) \longrightarrow SA(64, 0.2, [64 + 3, 64, 128, 512]) \longrightarrow FC([512 + 3, 512, 1024 * 2])$$

For shape classification experiments, we decode the latent vector into K category scores by fully-connected layers:

$$FC([1024, 512], 0.5) \longrightarrow FC([512, 256], 0.5) \longrightarrow FC([256, K])$$

For part segmentation experiments, the point-wise features for predicting part category consists of: lifted point coordinates, a one-hot vector for representing the object category, and the latent vector. The architecture for point-wise prediction of K part category scores is as follows:

$$\begin{aligned} FC([1024 + 64 + 16, 512], 0.5) \longrightarrow FC([512, 256], 0.5) \longrightarrow \\ FC([256, 128], 0.5) \longrightarrow FC([128, K]) \end{aligned} \tag{4.6}$$

For point cloud completion experiments, the model with 0.1 ball-search radius achieves the best performance. The architecture of decoder is inspired by the folding idea proposed in [88], which folds 2D grids into 3D shapes:

$$FC([1024 + 2, 512, 512, 3]) \longrightarrow FC([1024 + 3, 512, 512, 3])$$

Training Strategy During optimization, we use the Adam optimizer [101] with default parameters except for the learning rate. We train models for three different tasks. (1) For shape classification, the learning rate starts with 0.001 and is scaled by 0.2 every 200 epochs and the maximum epoch is 500. Batch size is 64 and we split them into 2 NVIDIA Tesla V100 GPUs during training. (2) For part segmentation, the learning rate starts with 0.001 and is scaled by 0.2 every 200 epochs and the maximum epoch is 500. Batch size is 128 and we split them into 4 NVIDIA Tesla V100 GPUs during training. (3) For point cloud completion, the learning rate starts with 0.0002 and is scaled by 0.2 every 200 epoch and the maximum epoch is 500. Batch size is 64 and we split them into 4 NVIDIA Tesla V100 GPUs during training.

During training, we partition the point cloud into 64 local point sets, and each of them generates a vote in the latent space. To make the voting strategy tenable, we propose to drop some votes and only a small portion of votes contribute to infer the latent feature at training. We do random selection and in extreme case only one vote is selected. This ensures that a single vote still has the potential to be decoded to a reasonable prediction. Thus, the learned embedding can be leveraged by the basic unit voter at test and improves the robustness to any type partial shapes. Note that all votes contribute to compute latent feature at testing.

Partial Point clouds Partial point clouds are only used at testing. In this work, we experiment on two kinds of partial point clouds. Sample partial point clouds are visualized in the Figure 4.3. For point cloud datasets that do not provide partial point clouds, we adapt a simple strategy to simulate partial point clouds, which

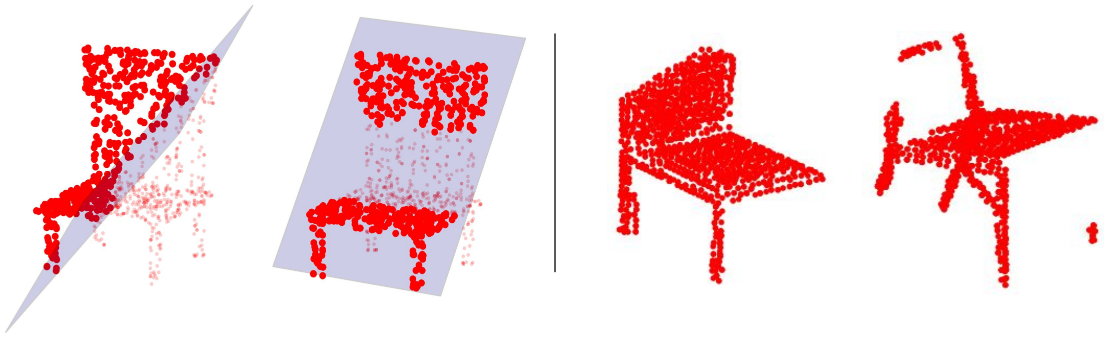


Figure 4.3: Sample partial point clouds. Left: partial point clouds synthesized by choosing points (bolded red) falling into one side of a random 2D plane. Right: partial point clouds generated by back-projecting depth images into 3D space.

are synthesized by selecting points falling into one side of a plane. The plane goes through the origin and is defined by the normal, which is a 3D vector and generated by randomly sampling from a normal distribution. The other kind of partial point clouds used in this work are generated by back-projecting depth images into 3D space, and they are provided by Completion3D dataset [90]. Compared to simulating partial point clouds using a subset of complete point clouds, these generated partial point clouds are closer to measurement from real-world sensors.

4.4.2 Point Clouds Classification

We first consider the task of point cloud classification. This requires that the model extract a global feature describing distinct geometric information and decode it into a predicated category.

Dataset We use the ModelNet40 [59] dataset to evaluate our proposed method on shape classification of point clouds. It contains 12,311 shapes from 40 object categories. In the experiments, point clouds are generated by evenly sampling 1024 points from the surface of objects. We follow the same strategy of set splitting as in [19]. Before being passed into the model, point clouds are centered and normalized within a unit sphere. No data augmentation techniques are applied during training.

Results Quantitative results on Modelnet40 are shown in the Table 4.1. Overall classification accuracy is reported on both complete point clouds in the column (Complete) and simulated partial point clouds in the column (Partial). All listed methods are trained on provided training set and achieve the state-of-the-art results on complete point clouds. Our proposed method performs slightly better than PointNet and PointNet++. When evaluated on simulated partial point clouds, however, other approaches experience a significant drop in accuracy. This is unsurprising since existing approaches are designed and trained for complete point clouds, and generalize poorly to novel partial point clouds. In contrast, our method trained on complete

Method	Input	Complete	Partial	Partial*
PointNet [19]	xyz	88.8	20.9	76.5
PointNet++ [20]	xyz	91.0	61.5	81.9
RS-CNN [21]	xyz	92.3	43.3	71.1
DG-CNN [22]	xyz	**92.9**	51.5	64.9
Ours	xyz	91.4	**86.4**	**86.4**

Table 4.1: Classification accuracy on ModelNet40. Overall classification accuracy is reported on complete point clouds (Complete) and simulated partial point clouds (Partial). The last column (Partial*) shows the results when models trained using the proposed training strategy

point clouds using the proposed training strategy is robust to partial observation and achieves 86.4% classification accuracy on partial point clouds. Analysis of the proposed training strategy will be discussed in the later section.

4.4.3 Part Segmentation

Given the point clouds and object category, the part segmentation tasks requires predicting a part label for each point. As a result, this task requires that the model extract both global and local information.

Dataset We use the ShapeNet part dataset [59] to evaluate our proposed method on part segmentation of point clouds. It contains 16,881 shapes from 16 object categories with 50 parts. Each point cloud contains 2048 points which are generated by evenly sampling from the surface of objects. We follow the same set splitting conventions in [20]. During evaluation, mean inter-over-union (mIoU) that are averaged across all classes is reported. No data augmentation techniques are applied during training.

Results We report the results of our proposed method and compare its performance to existing approaches in the Table 4.2. All methods are trained on provided training set. Our proposed method has superior performance on partial point clouds and achieves 78.1 mIoU, while others achieve around 30 mIoU. This robustness to partial observation comes at the expense of slightly lower accuracy on segmentation of complete point clouds when compared with other approaches. We also test the robustness of our approach by applying it to the completion3D dataset [90] which contains partial point clouds but no part labels. Qualitative results on part segmentation by applying the proposed method proposed are shown in the Figure 4.4.

4.4.4 Point Clouds Completion

Given partial point clouds, point cloud completion aims to generate points to recover the complete shapes. This requires a model that has the ability to infer a global feature which encodes complete point clouds from the partial inputs.

Method	Input	Complete	Partial
PointNet [19]	xyz	80.5	29.9
PointNet++ [20]	xyz	82.0	30.9
DG-CNN [22]	xyz	82.3	29.8
RS-CNN [21]	xyz	**82.4**	30.6
Ours	xyz	79.0	**78.1**

Table 4.2: Part segmentation results on ShapeNet part dataset. Mean intersection of unions (mIoUs) is reported on complete point clouds (Complete) and simulated partial point clouds (Partial).

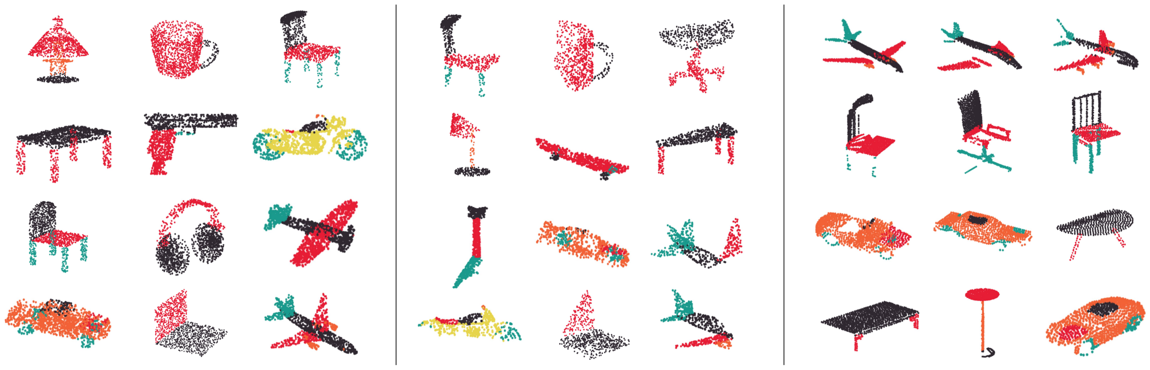


Figure 4.4: Qualitative results on part segmentation of point clouds using the method presented in this work. Different colors correspond to distinct segments. Left: segmentation on the ShapeNet part dataset. Middle: segmentation on the simulated partial point clouds in the ShapeNet. Right: segmentation on the Completion3D dataset.

Dataset We evaluate our model on Completion3D [90], which is a 3D object point cloud completion benchmark. It contains pairs of partial and complete point clouds from 8 categories which are derived from the Shapenet dataset with 2048 points per object point clouds. We apply the set splitting given by the dataset. Partial point clouds are generated by back-projecting 2.5D depth images into 3D space and complete point clouds are used as ground truth.

Results Quantitative results on Completion3D's withheld test dataset are shown in Table 4.4 where the Chamfer distance multiplied by 10^4 is reported. Our proposed model achieves 18.18 average Chamfer distance and outperforms FoldingNet (19.07) and PCN (18.22), which are trained with both partial and complete point clouds, while only complete point clouds are used during training for the method developed in this work.

4.4.5 Ablation study

Results of ablation study are shown in the Table 4.3 and the overall classification accuracy is reported on the simulated partial test set of ModelNet40. Unsurprisingly, the accuracy of models is improved after using the batchnorm [52] and dropout [103]

Model	BN	DP	# v. train	# v. test	radius	BK	Acc.
A_1			10	16	0.25	1024	78.4
A_2	✓		10	16	0.25	1024	80.9
A_3	✓	✓	10	16	0.25	1024	**81.0**
B_1	✓	✓	4	16	0.25	1024	79.1
B_2	✓	✓	10	16	0.25	1024	**81.0**
B_3	✓	✓	16	16	0.25	1024	79.4
B_4	✓	✓	32	16	0.25	1024	78.1
B_5	✓	✓	64	16	0.25	1024	76.2
C_1	✓	✓	10	8	0.25	1024	76.8
C_2	✓	✓	10	16	0.25	1024	81.0
C_3	✓	✓	10	32	0.25	1024	82.7
C_4	✓	✓	10	64	0.25	1024	83.8
C_5	✓	✓	10	128	0.25	1024	84.4
C_6	✓	✓	10	256	0.25	1024	**84.6**
D_1	✓	✓	10	256	0.15	1024	83.1
D_2	✓	✓	10	256	0.20	1024	**86.4**
D_3	✓	✓	10	256	0.25	1024	84.6
D_4	✓	✓	10	256	0.35	1024	82.5
E_1	✓	✓	10	256	0.20	512	85.3
E_2	✓	✓	10	256	0.20	1024	86.4
E_3	✓	✓	10	256	0.20	2048	**86.8**

Table 4.3: Ablation study. The metric is overall classification accuracy on the simulated partial ModelNet40 test set. "BN" indicates using batch normalization; "DP" indicates using the dropout technique in fully connected layers except the final one; "# v. train" indicates the maximum number of votes selected to contribute to the latent feature at training; "# v. test" indicates the number of votes at test; "radius" indicates the ball radius of local regions; "BK" indicates the dimension (bottleneck) of the latent space.

Model	Plane	Cabinet	Car	Chair	Lamp	Sofa	Table	W.craft	Average
FoldingNet [88]	12.83	23.01	14.88	25.69	21.79	21.31	20.71	11.51	19.07
PCN [89]	9.79	22.70	12.43	25.14	22.72	20.26	20.27	11.73	18.22
AtlasNet [102]	10.36	23.40	13.40	24.16	20.24	20.82	17.52	11.62	17.77
TopNet [90]	7.32	**18.77**	**12.88**	**19.82**	**14.60**	**16.29**	**14.89**	**8.82**	**14.25**
Ours	**6.88**	21.18	15.78	22.54	18.78	28.39	19.96	11.16	18.18

Table 4.4: The performance of various state-of-the-art algorithms on partial point cloud completion on the Completion3D benchmark dataset. Results are reported on the Completion3D's withheld test set. The Chamfer distance (CD) is reported, multiplied by 10^4.

Model	Plane	Cabinet	Car	Chair	Lamp	Sofa	Table	W.craft	Average
FoldingNet [88]	25.79	40.52	16.12	39.90	43.01	43.76	40.88	26.54	34.56
PCN [89]	21.58	41.87	16.56	38.86	50.19	43.37	39.44	27.57	34.93
AtlasNet [102]	23.13	49.60	16.80	43.34	60.83	48.21	41.94	33.96	39.73
TopNet [90]	21.61	**15.24**	38.14	35.23	44.42	38.36	36.18	25.97	31.87
Ours	**7.54**	17.96	**9.22**	**19.49**	**29.97**	**15.82**	**24.58**	**13.15**	**17.22**

Table 4.5: The performance of various state-of-the-art algorithms trained via the Completion3D dataset on partial point cloud completion on a simulated partial point cloud dataset. Results are reported on simulated partial point clouds of validation set. The Chamfer distance (CD) is reported, multiplied by 10^4.

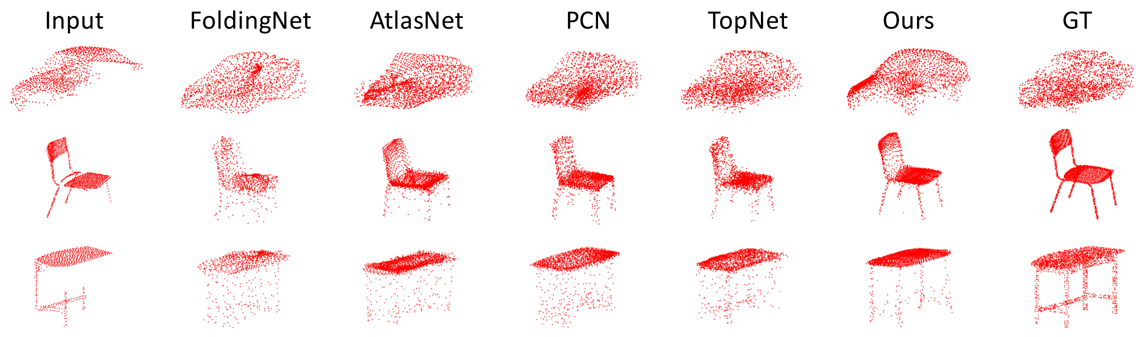


Figure 4.5: Qualitative point clouds completion results on partial point clouds provided in Completion3D dataset.

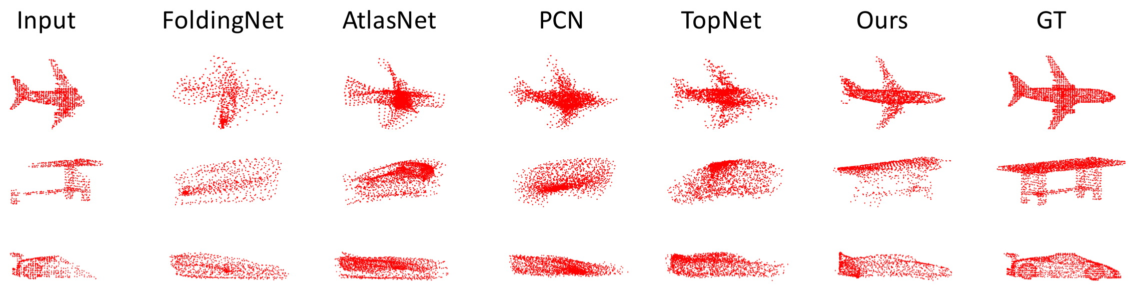


Figure 4.6: Qualitative point clouds completion results on partial point clouds simulated on Completion3D dataset using the introduced strategy in the work.

techniques, from 78.4% to 81.0%, shown by Model A_i. We model the point clouds as an overlapping partition of point sets, each of which is defined by its centroid and scale. As it is shown by Model D_i, the performance of this modeling is sensitive to the scale and accuracy peaks at 0.2 (86.4%). This can be in part explained by that local regions with small scale contain insufficient distinct geometry features to infer the complete point clouds, while the learned features for local regions with large scale tend to be different from those in partial point clouds with unexpected edges due to missing parts. Model E_i illustrates that the performance of model grows as we increase the dimension of the latent space (bottleneck), and it saturates at 2048 (86.8%).

We propose to infer latent features voted from local point sets. To make this voting strategy tenable, we design a training strategy that random number of votes are selected to compute the latent feature. In the experiments, the maximum number of selected votes is manually set during training. As it is shown by Model B_i, the performance peaks when the maximum number of votes is set to 10 (81.0%) and degrades as more votes are considered. We suspect that this is because more votes at training mean that more points are observed, and it leads to degradation at testing when only a small portion of partial objects are observed. Model C_i illustrates that

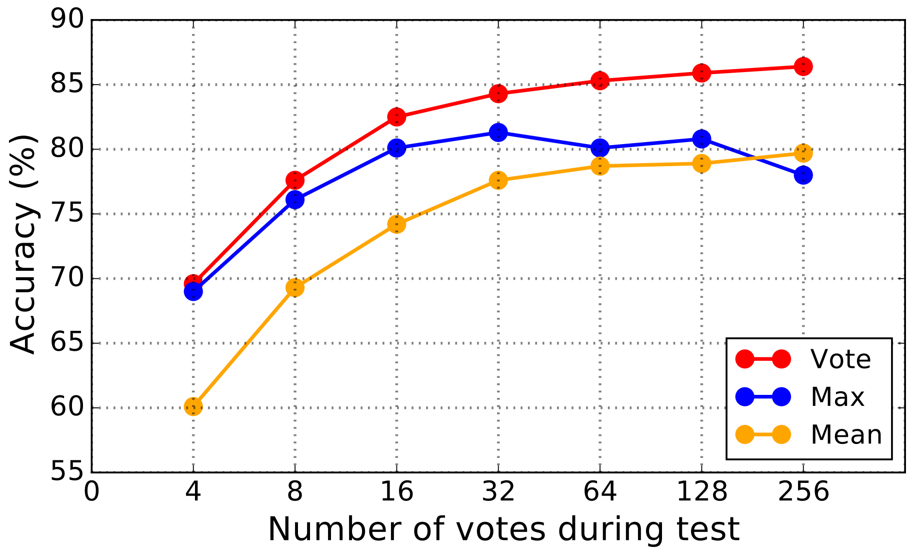


Figure 4.7: Results of different aggregation strategies for computing latent features on simulated partial point clouds in ModelNet40.

the accuracy of the trained model grows as the number of votes increases at test from 76.8% to 84.6%, which indicates that more votes accumulate information for a more confidant prediction.

4.4.6 Training Strategy Analysis

We propose that a random number of votes are selected to compute the latent feature at training, which makes the voting strategy tenable and robust to the variant number of votes. Since the latent feature is inferred from a subset of votes, the model at training makes predictions from partial shapes though complete shapes are taken as inputs. Thus, the proposed training strategy can be seen as a way of data augmentation. To make a fair comparison, we also report results of baseline models trained with the proposed training strategy, and they are shown in the column (Partial*) of the Table 4.1. For other methods, the partial point clouds fed into the models at training consist of points covered by random sampled votes. Compared to results on partial shape while trained on complete shape in the Table 4.1, the proposed training strategy boosts the performance of PointNet (from 20.9% to 76.5%), PointNet++ (from 61.5% to 81.9%), RSCNN (from 43.3% to 71.1%), and DGCNN (from 61.5% to 81.9%). However, our proposed method (86.4%) still outperforms all of them with noticeable improvement, which verifies the effectiveness of our method.

4.4.7 Voting Strategy Analysis

As proposed in this work, we infer the latent feature from votes of local point sets, and each vote is a distribution in the latent space. The optimal latent fea-

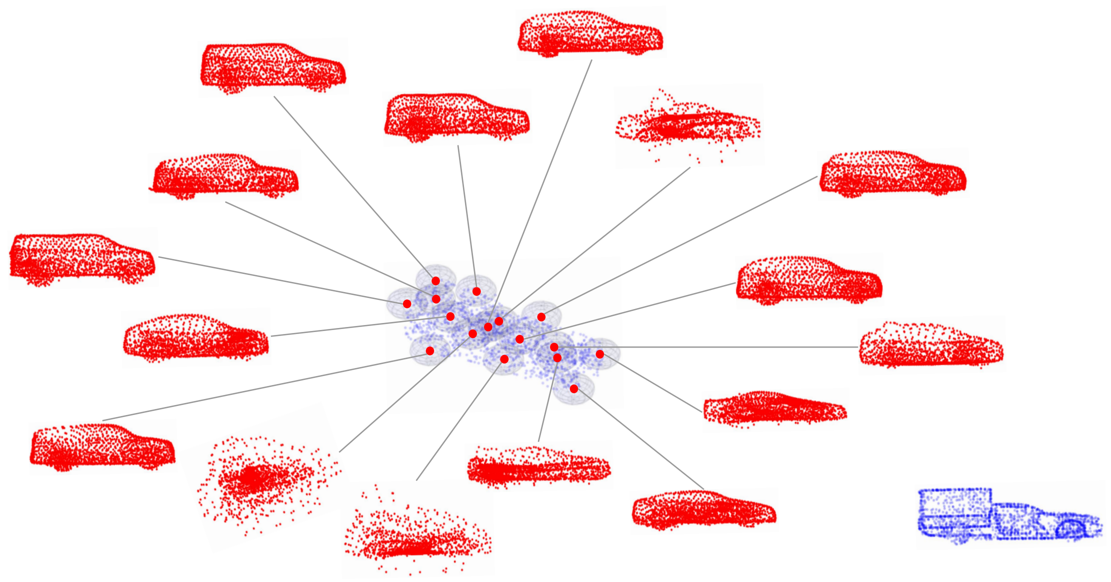


Figure 4.8: Visualization of voting strategy. Each vote is decoded into a complete shape by using the model trained on Completion3D dataset. The points in blue show the complete shapes.

ture is the sampled one with the highest probability. In this section, we analyze this voting strategy and compare it with the aggregation strategy that the extracted features from local point sets are aggregated using a symmetric function. We study two symmetric functions: max pooling and mean pooling. To utilize the aggregation strategy, we change the vote of each local point set from a distribution to a deterministic feature vector.

The comparison results are shown in the Figure 4.7. The evaluation metric is the average classification accuracy on the simulated partial ModelNet40. All models are trained using the proposed training strategy where the maximum number of votes considered is set to 10. Our proposed voting strategy is verified by the improved accuracy when compared to the aggregation strategy using either max pooling or mean pooling. Classification accuracy grows as the number of votes increases during testing in both the voting strategy and mean pooling. This can be explained by more accumulated information as increasing the number of votes. However, this is not the case for max pooling, whose accuracy peaks at 32 votes, which indicates that max pooling is sensitive to the number of selected votes.

One of the novelties of the proposed method is to represent the latent space by a set of independent multivariate Gaussian distributions, which is generated from local point sets. With the help of the designed training strategy, the model learns to infer the information contained in the complete shape from an individual local point set. To verify the proposed voting strategy, we visualize each vote by decoding it into a complete shape with the model trained on Completion3D dataset. As shown in Figure 4.8, local point sets located at different parts of the object generate votes encoding information of different shapes. In particular, votes at the front of the vehicle tend to infer vehicles with sloping rears, while votes at the rear tend to infer truck-like vehicles. Moreover, compared to votes located at the front and the rear of the vehicle, votes in the middle contain less distinct geometry information since their decoded point clouds are blurrier. Therefore, we argue that the proposed voting strategy are able to model the variations when observing different parts of shapes.

4.4.8 Generalizability to different partial point clouds

We experiment on different partial point clouds to verify the generalizability of the proposed model. In this section, we target at the point cloud completion task. In addition to evaluating test set withheld in Completion3D dataset, which are shown in the Table 4.4, we more exhaustively evaluate all approaches by experimenting on simulated partial point clouds as in the left subfigure of Figure 4.3. Note that all models are trained on the provided training set in Completion3D dataset.

As shown in Table 4.5, all approaches except the one developed in this work experience a significant performance drop when compared to the results shown Table 4.4. We suspect that this is because the partial point clouds in the Completion3D are different from the simulated partial point clouds, as it is shown in the Figure 4.3. Unlike other approaches embedding all input points into an encoding feature, we propose to rely on each local point set as the basic voter, which is more generalized and less affected by partial shapes. This in part explains that our proposed method performs equally well on both experiments. Qualitative completion results are shown in Figure 4.6, and it shows that our proposed model outputs sharper shapes. More challenging tests are performed on real-world point clouds and are shown in the Figure 4.10. These partial point clouds are extracted within the labeled object bounding boxes provided in ScanNet [28]. Note that the input point clouds need to be transformed to the box's coordinates and the same scale as the training dataset before being fed into model. This is because that the representations learned by the proposed model are not designed to be invariant to rigid body transformation.

4.4.9 Visualization of multiple predictions

The method developed in this work is designed to be able to generate multiple possible outputs. This is achieved by the latent space model. The latent space is represented by a set of multivariate Gaussian distributions generated by local point sets. The use of latent value with the highest probability produces a single deterministic prediction. Diverse predictions can be generated by sampling from the latent space and followed by the decoding module. Since it is not easy to sample

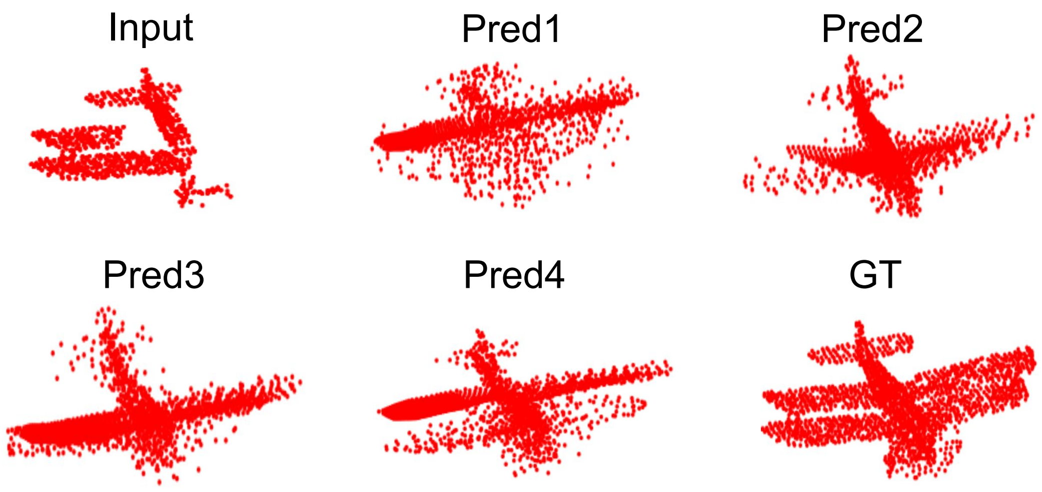


Figure 4.9: Visualization of diverse predictions on point clouds completion on the Completion3D dataset.

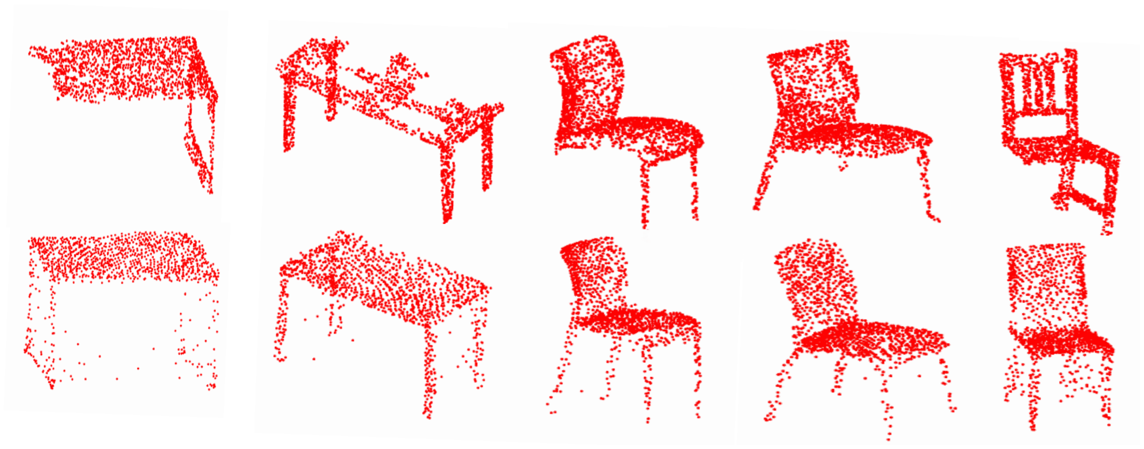


Figure 4.10: Completion on real-world point clouds from ScanNet. Top row: input partial point clouds. Bottom row: complete point clouds generated by ours.

in the latent space as it is represented by a set of distributions, we instead sample latent value by interpolating between the optimal latent feature inferred by all votes and a optimal latent feature inferred by a single vote. We perform experiments on point clouds completion and visualize the results in Figure 4.9.

4.5 Conclusion

This chapter proposes a general model for partial point clouds analysis. In particular, point clouds are modeled as a partition of point sets which generate votes to model a latent space distribution. This voting strategy is shown to accumulate partial information and be robust to partial observation. The sampled latent feature in the latent space is then decoded for prediction. Extensive experiments are performed in classification, part segmentation, and completion and state-of-the-art results on all of them demonstrate the effectiveness of the proposed method.

CHAPTER V

Hyperspherical Embedding for Point Cloud Completion

Thanks to deep learning techniques and availability of large 3D datasets, recent works have achieved impressive performance on point cloud analysis by effectively learning representations needed for the corresponding learning tasks. In those datasets, most 3D measurements lack complete information of objects and lead to variations between training and test data, which raises challenges for analyzing point clouds. Thus, it is desirable to predict the complete shapes of objects from partial observations. The common point cloud completion network consists of an encoder-decoder structure, in which the encoder extracts embeddings that will be used to generate outputs for different tasks. Unfortunately, the learned embeddings shown in this chapter are sparsely distributed in the feature space, which leads to worse generalization results at testing. To address these problems, this chapter proposes a hyperspherical module, which takes embeddings from the encoder as inputs and transforms and normalizes them to be on a unit hypersphere. With the proposed hyperspherical embeddings, the magnitude and direction information are decoupled and only the direction information is optimized. Both the theory and experiments show that the proposed method enables more stable training with a wider range of learning rates and more compact embedding distributions. The consistent improvement of point cloud completion in both single-task and multi-task learning scenarios demonstrates the effectiveness of the proposed method. [1]

5.1 Introduction

The continual improvement of 3D sensors provides more accessibility of point clouds and demands the need of algorithms to analyze them. Thanks to deep learning techniques, recent works have achieved impressive performance on point cloud analysis [19, 20, 25, 26, 27] by effectively learning representations from large 3D datasets [17, 28, 29]. Point clouds provided in those dataset are often incomplete and

[1]This chapter is based on a manuscript under review

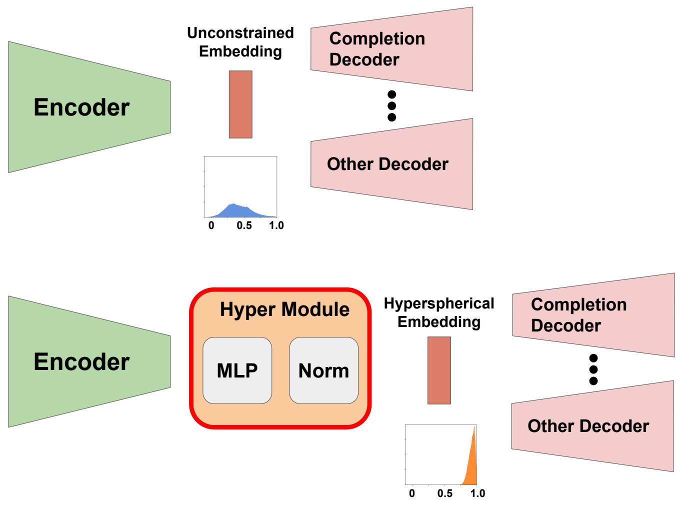


Figure 5.1: Architecture of point cloud learning. The upper subfigure shows the general point cloud analysis structure, where the embedding is directly output from the encoder without constraints. The lower subfigure shows the structure of the model with the proposed hyperspherical module. The figures under the embeddings illustrate the cosine similarity distribution between embeddings, which indicates a more compact distribution achieved by the proposed method.

sparse due to occlusions, low resolution and the limited view of 3D sensors. These measurements lack complete information and contain variations between training and test data, which raises challenges for analyzing point clouds. Therefore, the ability to predict complete shapes of objects from partial observations is desirable. For example, the predicted complete shapes will provide finer collision-check boundaries than bounding boxes to help autonomous vehicles navigate in narrow lanes or crowded urban regions; compared to incomplete measurements, the robot arm can grasp objects in more reasonable poses with the information of complete shapes, in particular, for those unknown objects.

More recently, many works have been proposed to perform point cloud completion [89, 90, 104, 105, 106, 107]. All of them adapt encoder-decoder structures, in which the encoder takes a partial point cloud as input and outputs an embedding vector, and then it is processed by the decoder that predicts a complete point cloud. The embedding space is designed to be high-dimensional as it is considered to have enough capacity containing the information needed for downstream tasks. However, the learned high-dimensional embeddings, as shown in Figure 5.4, tend to be sparsely distributed in the embedding space which increases the possibility that unseen features at testing are not captured by the representation learned at training and leads to worse generalizability of models.

Usually, building a real-world application, such as autonomous vehicle and robot arm, requires not only the information of complete shapes but also predictions from other tasks. Compared to training all tasks individually from scratch, a more compact way is to train all tasks jointly, which can reduce the number of parameters and achieve better efficiency by sharing parts of networks for different tasks [108, 109, 110]. However, training all tasks together is a difficult optimization problem. Conflicts between different tasks are observed and training them jointly sometimes leads to worse performance compared to learning them individually [111, 112]. Existing point cloud completion works conduct experiments in single-task learning, while in this chapter we extend them to multi-task learning. We show that the structure with embeddings shared by different tasks raises conflicts at training between point cloud completion and other semantic tasks, which leads to worse completion performance and hinders accomplishing point cloud completion jointly with other tasks.

To address above limitations of the existing point cloud completion approaches, this chapter proposes a hyperspherical module and the overall architecture is shown in Figure 5.1. Specifically, the hyperspherical module consists of a MLP layer and a normalization layer, and it can be easily integrated into the existing approaches. The normalization layer constrains the embedding onto the surface of a hypersphere by normalizing the embedding's magnitude to unit, which we called hyperspherical embedding. By doing that, the magnitude and direction information in embeddings are decoupled and only the direction information is kept for later use. Extensive evaluations are conducted to investigate the effects of hyperspherical embeddings on point cloud completion in both single-task and multi-task learnings. We show that with the hyperspherical embedding the models are improved by enabling larger learning rates, more stable training, and a more compact embedding distribution. The reported improvements of the existing state-of-the-art approaches on several public datasets illustrates the effectiveness of the proposed method. The main contributions of this chapter are summarized as follows:

- We propose a hyperspherical module which outputs embeddings constrained onto the surface of a hypersphere, which leads to improved performance of existing approaches for point cloud completion in both single-task and multi-task learning.

- We prove that the hyperspherical embedding benefits the training of models with a wider learning rate range, stable training, and a compact embedding distribution.

- We demonstrate the proposed method on public datasets with extensive experiments and show the state-of-the-art results.

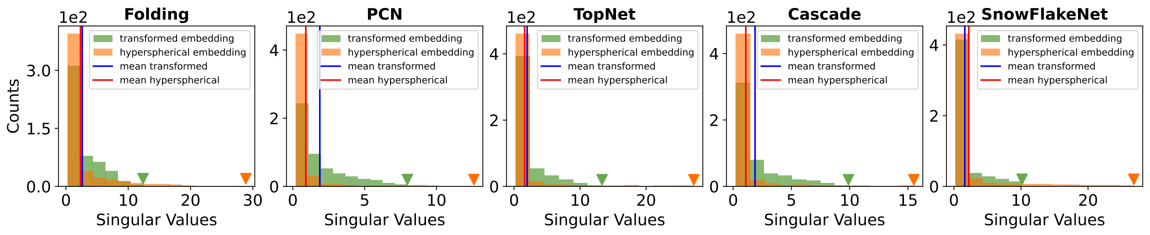


Figure 5.2: An illustration of distributions of singular values. We compute singular values from weights in the layer right before the embedding, obtained from the point cloud completion models on MVP dataset with different architectures described in the plot title. The mean of singular values are denoted by vertical lines, and the inverted triangle denotes the largest singular value. Normalization leads to learning more singular weights.

5.2 Related Work

Point Cloud Completion. The point clouds in most 3D dataset [17, 29, 28, 113, 114] are incomplete and sparse due to occlusions, low resolution and the limited view of 3D sensors. Unfortunately, those point clouds may lack important information contained in complete shapes and raise challenges for point cloud analysis. To address this problem, many works have been proposed to perform point cloud completion, in which people seek to recover the full shape of objects based on the partial observations [89, 90, 32, 104, 107, 106]. Most of them adapt a pipeline of encoding the partial observations into an embedding feature and then decoding it to complete point clouds. Since the structures of encoders have been successfully explored by other 3D tasks [20, 19, 22, 21], most completion approaches focus on designing different completion decoders [89, 88, 90]. However, those one-stage decoders are shown to output unevenly distributed points over the surface of objects, and they also struggle to preserve the detail structures in the inputs. Later approaches address this issue by using a coarse-to-fine decoding strategy, in which the decoding process generates several complete shapes at different resolutions, and the partial point clouds are used along with other intermediate decoding results to maximally preserve structures in the inputs [104, 107, 106, 115]. Different from designing a completion decoder, the method developed in this chapter focuses on representations learned from encoders, which is a more universal problem since the developed method can be integrated into existing structures. We show the improvement of point cloud completion when using the proposed method on both one-stage and coarse-to-fine decoders.

Hyperspherical Embedding. Effectively obtaining a representation that is needed for the corresponding learning task from raw input data is very important. More recently, deep learning techniques with the help of large datasets are able to learn powerful representations and have demonstrated the superiority over the handcraft features. Compared to representations learned without constraints, hyperspherical representations have shown advantages in many fields, such as representation learning, metric learning, and face verification [116, 117, 118, 119, 120, 121, 122]. In those works, the hyperspherical embedding is obtained by normalizing the vectors to be on a unit hypersphere. The observed improvement by hyperspherical embeddings are investigated and can be partly explained by the uniformity of embeddings in the hypersphere space [117, 123], and more stable training [124]. All those works indicate the success of hyperspherical embedding and suggested unit hypersphere is a nice feature space. Thus, this chapter is inspired to apply hyperspherical embeddings in learning point cloud completion. We propose a new module that outputs hyperspherical embeddings and shows consistent improvement of point cloud completion in both single-task and multi-task learnings.

5.3 Review of Learning Point Cloud

Since our work targets the representations learned from the encoders, we start by reviewing the general structures of point cloud models.

Inspired by the success in the 2D field, most approaches for point cloud analysis adopt encoder-decoder or its variant structures. Various encoders have been designed to process point clouds, but most of them use a symmetric function, such as max pooling, in the last layer to address the permutation issue contained in the unordered point set. The outputs from the encoder are learned embeddings, and they will be then processed by following decoders to generate predictions. In multi-task learning, suppose the input point cloud is denoted by $\mathbf{x} \subseteq \mathbb{R}^3$, and the pipeline can be summarized as follows:

$$\mathbf{y}_i = D_i(E(\mathbf{x})) \tag{5.1}$$

where $E(.)$ denotes the encoder with a max pooling in the last layer, $D_i(.)$ denotes the decoder for ith task, $\mathbf{y}_i$ demotes the prediction for ith task, and the output from encoder $E(\mathbf{x})$ is the embedding. The overall architecture is shown in Figure 5.1.

Existing problems Most existing encoders learn representations, or embeddings, from large datasets using the end-to-end training. This end-to-end learning strategy does not add constraints on the embeddings, and the embedding distribution tends to be sparse as shown by previous works [121, 120]. However, a sparse embedding distribution increases the possibility of testing inputs accidentally falling into the regions unseen during training due to gaps or holes in the embedding space. On the contrary, a compact embedding distribution encourages to preserve more information and helps improve the generalization robustness to unseen features [123, 117]. When it comes to multi-task learning, embeddings from the encoder are shared by different decoders to improve efficiency of models. Unfortunately, those decoders learned by different task losses may require different embedding distributions, which may lead to optimization conflicts during training, and an unconstrained embedding space makes such an issue happen more easily. In this chapter, we focus on point cloud completion in single-task learning and multi-task learning with other semantics tasks, such as classification and segmentation. We use fully connected layers to predict semantic scores and are supervised by cross entropy loss, while decoders with more sophisticated structures are used to generate complete point clouds trained using Chamfer Distance [86].

5.4 Hyperspherical Embedding for Learning Point Cloud

In this section, we describe the proposed hyperspherical module and investigate the effects of hyperspherical embedding.

5.4.1 Proposed Hyperspherical Module

To address the issues caused by sparse embedding distribution, we propose a new hyperspherical module and its structure is shown in Figure 5.1. The proposed module contains two layers, a MLP layer and a normalization layer. Outputs from a encoder are first transformed by the MLP layer and then the normalization layer constrains the features onto the surface of hypersphere by l_2 normalization,

$$\hat{f} = \frac{f}{\|f\|_2} \tag{5.2}$$

where the $\|f\|_2 = \sqrt{\sum_i f_i^2}$, and $\hat{f}$ denotes the l_2 normalized embedding of f. With the normalization, we remove the magnitude information contained in the embeddings and only the direction information is optimized.

5.4.2 Effects of Hyperspherical Embedding

In this part, we investigate the effects of optimizing the l_2 normalized embedding and we draw several conclusions: 1) the gradient of the embedding before normalization is orthogonal to itself; 2) the magnitude of the embedding before normalization increases at each update; 3) the increased magnitude enables wider range of learning rates and more stable training compared to unconstrained embeddings; 4) the embedding distribution is compact in angular space, resulting better completion performance in both single-task and multi-task learnings.

Proposition V.1. *During optimizing l_2 normalized embedding $\hat{f}$, the computed gradient to the embedding before normalization denoted by f is orthogonal to itself,* $\langle f, \frac{\partial L}{\partial f} \rangle = 0$.

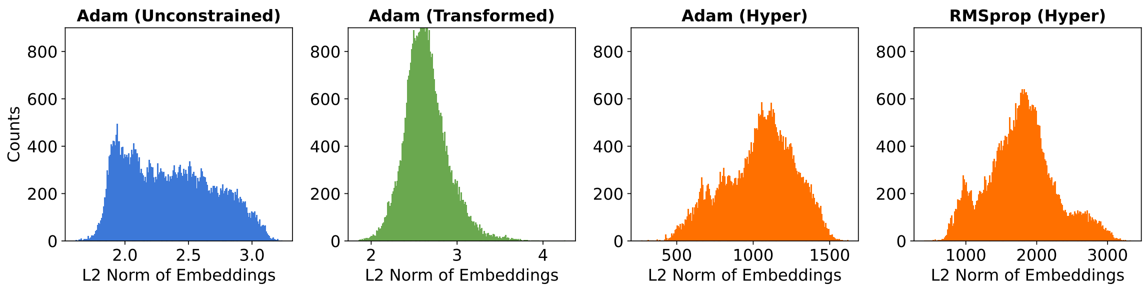


Figure 5.3: Distribution of embedding's norm. The embeddings are derived from the MVP test set on point cloud completion, obtained by using different embeddings or different optimizers, as described in the plot titles. Unconstrained embeddings are from models without the proposed module. Transformed embeddings are from models with the proposed module but removing the normalization layer. Hyperspherical embeddings are from models using the proposed module. Hyperspherical embeddings have large norms.

Proof. Suppose the normalization process follows Equation 5.2 and the loss at optimization is denoted by L. The gradient to embedding f can be formulated as:

$$\frac{\partial L}{\partial f} = \frac{\frac{\partial L}{\partial \hat{f}} - \hat{f}\langle \frac{\partial L}{\partial \hat{f}}, \hat{f}\rangle}{\|f\|_2} \tag{5.3}$$

Based on it, we can show the orthogonality by computing the inner product between the embedding and its gradient. More details can be found in Appendix B. Similar conclusions are also reported in [118, 121, 119].

Proposition V.2. *For standard stochastic gradient descent (SGD), each update of embedding f will monotonically increase its norm, $\|f\|_2$.*

Proof. The orthogonality between the embedding and its gradient from Proposition V.1 indicates that applying gradient at each update increases the norm of an embedding, which is validated in Figure 5.3 by showing the distribution of embedding's norm. Similar observations are also reported in [118, 119]. This property is based on the vanilla gradient descent algorithm and does not strictly hold for optimizers that use momentum or separate learning rates for individual parameters. However, we still find the same effect empirically hold for other SGD-based optimizers, such as Adam [101] and RMSprop [125], as they are illustrated in Figure 5.3.

Proposition V.3. *The magnitude of the gradient is inversely proportional to the norm of the embedding, $\frac{\partial L}{\partial f} \propto \frac{1}{\|f\|_2}$.*

Proof. Considering the norm $\|f\|_2$ in the denominator in Equation 5.3, the magnitude of the gradient is inversely proportional to the norm of an embedding. This conclusion is similar to the one in [119]. However, we further note that this effect enables optimizing neural networks with a wider range of learning rates. In particular, the norm of embedding trained with a large learning rate quickly increases shown by Proposition V.2 until an appropriate effective learning rate is reached, while the same setting puts the model at risk of overshooting the minima, so it may not be able to converge when using unconstrained embeddings. Similar conclusion is observed when normalizing the weights of neural networks proposed by [124], while this chapter focuses on normalizing the layer of embedding. Figure 5.9 shows the comparison results with hyperspherical embedding and unconstrained embeddings trained using different learning rates, and more results can be found in Figure B.2 and Figure B.1 in Appendix B. All of them show that models with hyperspherical embeddings enable stable training with a wider range of learning rates and better performance of point cloud completion.

Proposition V.4. *Considering a vector is transformed by a matrix, i.e., $f = Wx$. During optimization, the increased norm of f requires a poorly conditioned matrix W.*

Proof. The matrix W can be decomposed by singular value decomposition (SVD):

$$W = U\Sigma V^T \tag{5.4}$$

where U and V are orthonormal matrices and Σ is a diagonal matrix. Based on Equation 5.4, the norm of the transformed output f is only related to singular values contained by Σ, since orthonormal matrices U and V do not modify the magnitude of inputs. Recall that the norm of f increases at optimization from Proposition V.2. Thus, the weight W will be updated with increasing singular values during optimization. Increasing all singular values makes a greater weight. However, the large weights lead to overfitting the training data and adversely affect the performance of models [126]. Regularization can be used to alleviate this issue, but we empirically find that the scale of weights trained with normalized embeddings behaves similarly to the one without using it when regularization is not used. Therefore, the increase of certain singular values in Σ will inevitably lead to decrease of other singular values, which causes the weight W to be poorly conditioned as it is illustrated in Figure 5.2.

One benefit of learning a singular weight is that it will reduce the complexity in the input high-dimensional data by keeping information on principle axes determined by large singular values while ignoring information on other axes. By doing this, outputs transformed by a singular weight will point in a similar direction. Moreover, the discrepancy of embeddings in magnitude is removed by l_2 normalization. We compute pairwise cosine distance of embeddings trained with or without l_2 normalization in the test set and visualize the distribution in Figure 5.4. It shows that normalization leads to a more compact angular distribution, while the unconstrained

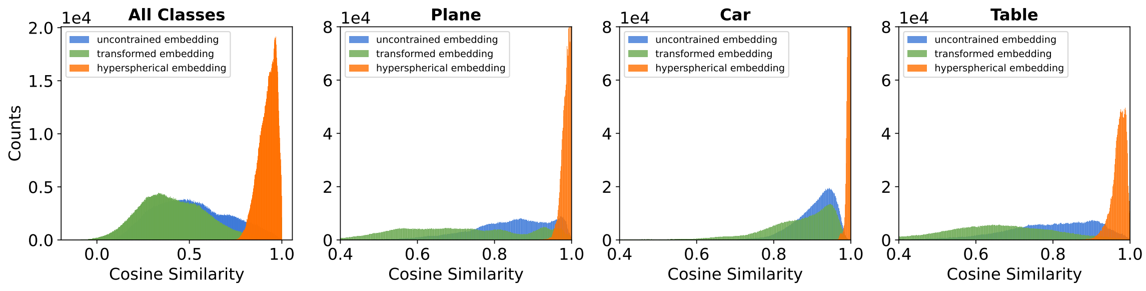


Figure 5.4: Cosine similarity distribution of embeddings. We compute pairwise cosine distance between embeddings obtained from the test set in MVP dataset. We visualize distribution in either one class or overall classes as described in the plot titles. Hyperspherical embeddings have more compact angular distribution.

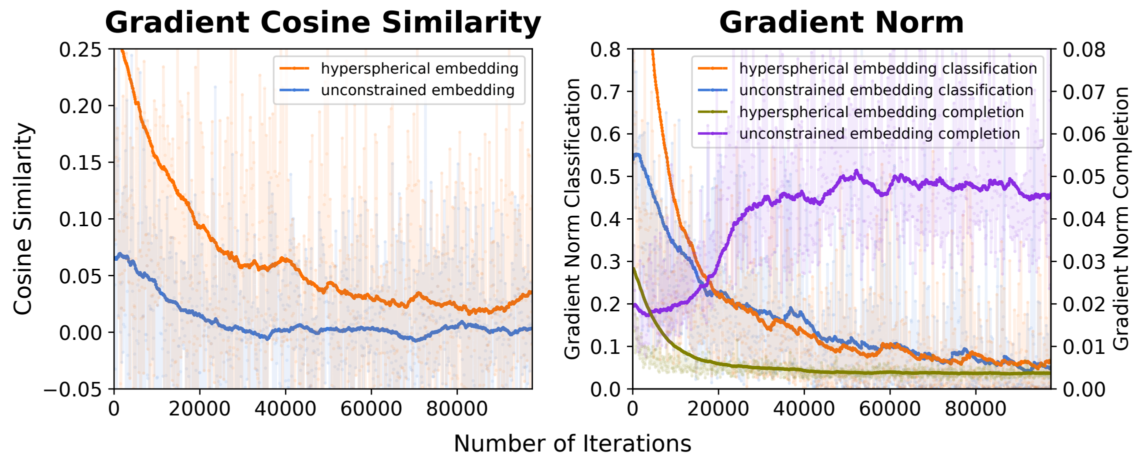


Figure 5.5: An illustration of gradient conflicts between tasks in multi-task learning during training. We visualize the gradient cosine similarity and gradient magnitude as indicated by the titles of subfigures, obtained by training point cloud completion and shape classification on MVP dataset. Hyperspherical embeddings lead to smaller gradient conflicts between tasks in multi-task learning.

embeddings are sparsely distributed. This compact embedding distribution helps the model generalize well on unseen data at test and increase generalizability of models.

The learned compact embedding distribution also helps reconcile the learning conflicts in multi-task learning. The resulting compact embedding distribution forces different tasks to learn within the shared space, while the unconstrained embedding space provides tasks the freedom to land on optimal embedding distributions with discrepancy. We use the gradient cosine similarity to measure the conflicts between different tasks [112] and visualize the training process in Figure 5.5. The figure shows that the hyperspherical embeddings lead to smaller gradient conflicts at training, as larger cosine similarity indicates smaller gradient conflicts. More discussion about the effects on multi-task learning can be found in Sec 5.5.3.

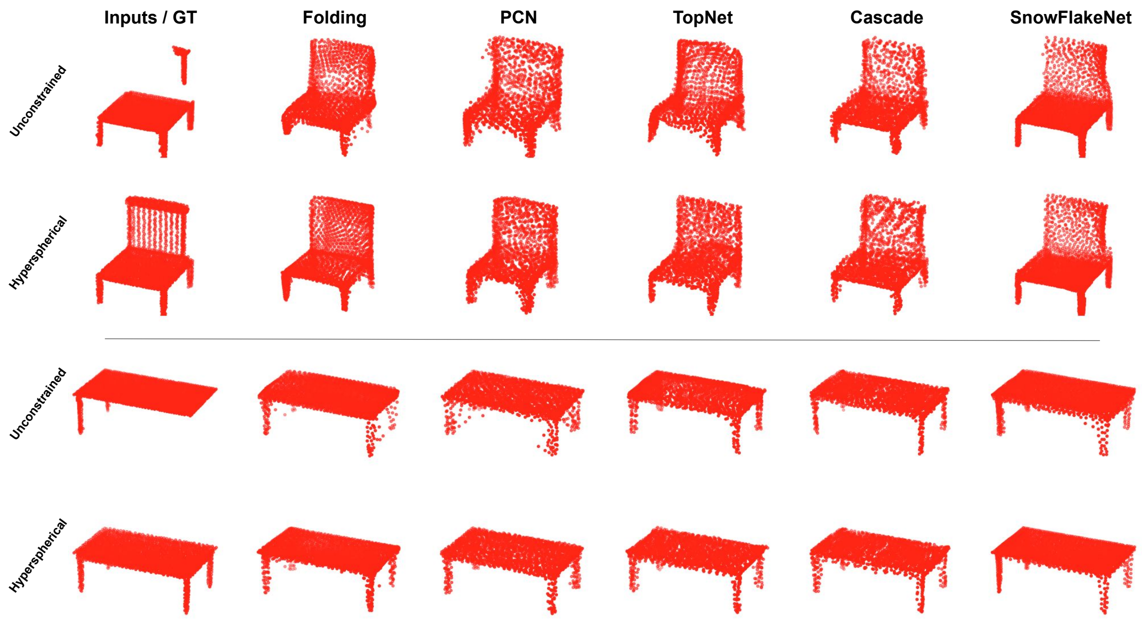


Figure 5.6: Qualitative results of various state-of-the-arts point cloud completion approaches on MVP test set.

Model	plane	cabinet	car	chair	lamp	sofa	table	wcraft	bed	bench	bshelf	bus	guitar	mbike	pistol	sboard	average
Folding [88]	4.71	9.08	6.81	15.22	23.12	10.28	14.32	9.90	22.02	10.28	14.48	5.24	**2.02**	6.91	7.21	4.59	10.39
Folding (H)	**4.48**	**8.82**	**6.68**	**13.79**	**21.44**	**9.66**	**12.98**	**8.57**	**18.93**	**8.96**	**13.44**	**4.95**	2.03	**6.43**	**6.22**	**4.20**	**9.47**
PCN [89]	**4.23**	9.35	6.73	13.56	**20.94**	10.51	14.20	9.81	21.32	9.98	15.08	5.45	1.90	6.23	6.23	5.03	10.03
PCN (H)	4.24	**9.14**	**6.49**	**13.04**	22.47	**10.04**	**12.99**	**8.75**	**18.95**	**9.33**	**13.93**	**5.06**	**1.84**	**6.00**	**5.92**	**4.15**	**9.52**
TopNet [90]	4.63	9.23	6.79	14.31	19.50	10.48	14.30	9.65	20.54	10.12	15.53	5.36	2.09	6.77	7.74	4.94	10.12
TopNet (H)	**4.07**	**9.13**	**6.75**	**13.08**	**19.45**	**10.03**	**12.85**	**8.89**	**19.50**	**9.63**	**14.33**	**5.23**	**2.03**	**6.66**	**6.42**	**3.92**	**9.50**
Cascade [107]	2.66	8.69	6.02	10.22	13.07	8.76	9.90	6.67	16.44	7.56	**11.00**	4.97	1.98	4.58	4.54	2.78	7.49
Cascade (H)	**2.61**	**8.52**	**5.97**	**9.52**	**12.03**	**8.71**	**9.83**	**6.46**	**15.78**	**7.17**	11.15	**4.90**	**1.88**	**4.50**	**4.24**	**2.76**	**7.25**
SnowFlakeNet [104]	1.94	7.61	5.61	6.77	**6.82**	7.09	7.21	**4.65**	10.98	4.76	7.54	4.16	1.14	3.78	3.15	**2.67**	5.37
SnowFlakeNet (H)	**1.89**	**7.26**	**5.36**	**6.50**	7.59	**6.72**	**6.63**	4.67	**10.39**	**4.39**	**7.37**	**4.03**	**0.95**	**3.60**	**3.15**	2.84	**5.21**

Table 5.1: The performance of completion approaches trained via the MVP dataset on point cloud completion. The Chamfer Distance is reported, multiplied by 10^4, on the provided test set. “H” indicates using the proposed hyperspherical module.

5.5 Experiments

Experiments are divided into three parts. We first report results on different datasets to evaluate the effectiveness of the proposed methods in Sec 5.5.1. Second, we conduct a detailed ablation study to validate the design of our hyperspherical module in Sec 5.5.2. Finally, we analyze and visualize the effects of hyperspherical embeddings introduced in Sec 5.5.3.

5.5.1 Experiments on Different Datasets

More experiments on **ModelNet40** and **ShapeNet** can be found in Appendix B.

MVP MVP dataset [105] contains pairs of partial and complete point clouds from 16 categories. Partial point clouds are generated by back-projecting 2.5D depth images into 3D space and complete point clouds are used as ground truth. In the ex-

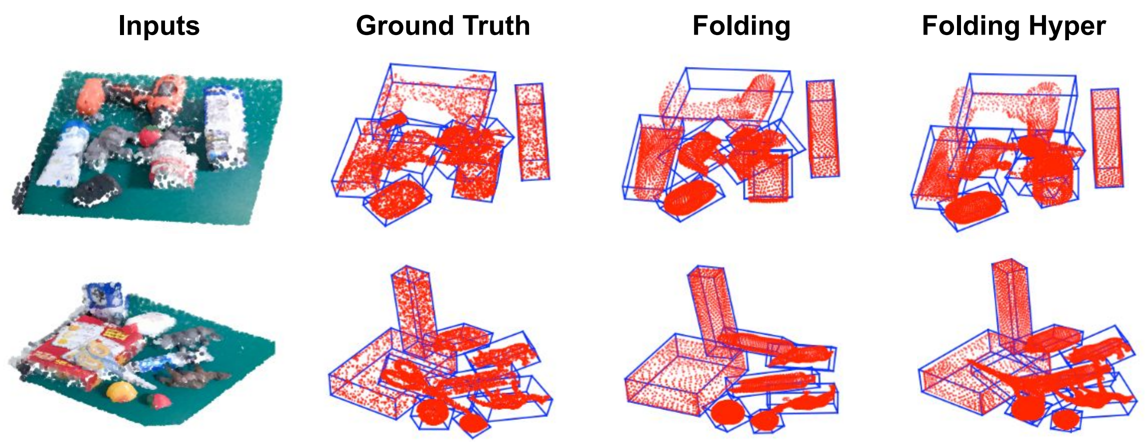


Figure 5.7: Qualitative 3D detection, pose estimation, and point cloud completion results on GraspNet test set.

periments, we apply the set splitting given by the dataset and no data augmentation is used.

We evaluate point cloud completion on MVP [105] and compare results of generating complete shape with 2048 points, and qualitative results can be found in Figure 5.6. Compared to predictions without our hyperspherical module, the predicted point clouds tend to be a bit more blurry and contain more noise points. We assume it is due to the test embeddings falling into regions not close to features captured at training. More quantitative results are shown in Table 5.1. We compare the baseline models by adding the proposed module after encoders and denote them by (H). One exception is SnowFlakeNet, in which the decoding process consists of encoding modules, so we also add hyperspherical modules in its decoder. To achieve fair comparison, we train all methods with best training setting and report their results. Table 5.1 shows that using our proposed module brings consistent improvements of all existing completion approaches with $3 \sim 9\%$ decrease of average class Chamfer Distance, which demonstrates the effectiveness of the method developed in this chapter. Multi-task learning on MVP dataset will be discussed in Sec 5.5.3.

GraspNet Most point cloud completion approaches reported results on synthetic datasets, since collecting a real-world dataset with annotation of complete shapes is expensive. Unfortunately, incomplete measurements from real-world 3D sensors differ from those point clouds synthesized in a simulated environment, and approaches trained on synthetic dataset struggle when tasked to perform completion in real-world scenarios. More recently, the GraspNet [127] dataset was released and it contains the groundtruth complete shapes of objects, which helps evaluate point cloud completion. GraspNet contains 190 cluttered and complex scenes captured by RGBD cameras, bringing 97,280 images in total. For each image, the accurate 6D

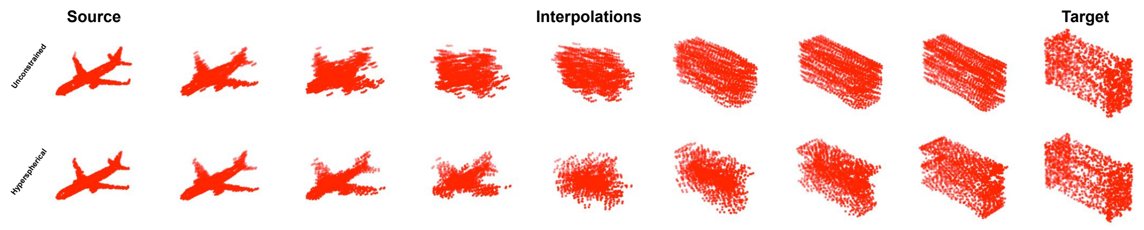


Figure 5.8: Illustration of point cloud interpolation on embedding space. The generated point clouds with hyperspherical embeddings have more clear clues from source or target shapes than those with unconstrained embeddings.

pose and the dense grasp poses are annotated for each object. There are in total 88 objects with provided CAD 3D models, and we use them to generate complete shape groundtruth with 1024 points.

To demonstrate on real-world scenarios, we aim to detect objects in 3D space along with predicting their complete shapes on GraspNet [127]. We modified the structures of VoteNet [26], which was designed to detect 3D objects from point clouds. The input point clouds are converted from the depth image captured by RGBD sensors. After extracting the embedding from each proposal, three branches of decoders are followed to generate predictions of 6DoF 3D bounding boxes, semantics and objectness scores, and complete shape of point clouds, respectively. In the evaluation, object detection is measured by mean average precision (mAP), poses of objects are measured by the symmetry metric proposed in [128], and the point cloud completion is measured by Chamfer Distance. We evaluate the effectiveness of the proposed hyperspherical module in this multi-task learning scenario, and the results shown in Table 5.2. Since the structures between models except decoders are the same, we denote the model by its decoders in the first column. Two-stage completion decoders, such as Cascade and SnowFlakeNet, are demonstrated to have better performance on synthetic dataset. Unfortunately, those decoders refine on perfect partial point clouds located on object's surfaces, which are inaccessible in this case. As it is shown in Table 5.2, hyperspherical embeddings help all three metrics compared to unconstrained embeddings with noticeable improvement. Qualitative results can be found in Figure 5.7 and Figure B.4 in Appendix B.

5.5.2 Ablation Study

The results of ablation study are shown in Table 5.3 and numbers in the table are average class Chamfer Distance of completion models with folding decoder reported on the MVP test set. As shown by the first and third column, adding the proposed hyperspherical module reduces the completion Chamfer Distance by 8.9%, from 10.39 to 9.47, which demonstrates the effectiveness of the method developed in

Model	mAP (0.25)	CD	Pose Acc.
Folding	70.50	0.18	52.42
Folding (H)	**71.21**	**0.14**	**54.01**
PCN	69.11	0.21	50.33
PCN (H)	**70.93**	**0.15**	**52.39**

Table 5.2: Performance on GraspNet test set. Average precision with 3D IoU threshold 0.25 (mAP 0.25) is reported for object detection, and Chamfer Distance (CD) is reported for point cloud completion, multiplied by 10^4, and pose accuracy (Pose Acc.) is reported for 6D pose estimation. The first column indicates the structure of decoder used in the model, and "H" indicates using the proposed hyperspherical module.

Unconstrained	Transformed	Hyperspherical	0.001 reg	0.01 reg	0.1 reg
10.39	10.12	**9.47**	10.40	10.73	10.89

Table 5.3: Results of ablation study. The reported metric is Chamfer Distance, multiplied by 10^4, of point cloud completion on MVP test set. The first three column indicates the results of models with different types of embeddings described similarly in Figure 5.3. "reg" indicates training with regularization term to discourage large norms of embeddings, and the number next to it indicates the weights when adding the term to training loss.

this chapter. There are two layers designed in the hyperspherical module, which are a MLP layer and a normalization layer. To validate this design, we report the results of removing the normalization layer. As shown in the second column, completion results are improved with only the MLP layer, from 10.39 to 10.12, and we argue the improvement is caused by less variation of embeddings' norm after transformation shown in Figure 5.3. By comparing the first three columns, we conclude that major improvement brought by the hyperspherical module resides in the normalization layer, from 10.12 to 9.47. From Proposition V.2, the norm of embeddings increases continuously at training, which is the key to learn a compact embedding distribution as shown in Proposition V.4. To verify the importance of the increased norm of embeddings, we experiment on adding a weighted constraint on embeddings to discourage large norms of embeddings along with other losses. The constraint in this case is the average norm of embeddings within a batch, which is similar to a regularization term. The results are shown in the last three columns denoted by (reg) in Table 5.3, and the number next to (reg) indicates the weights added to the training loss. Since the constraint offset the effect of hyperspherical embeddings to increase norm, we observe that performance of point cloud completion keeps getting worse when the weight of constraints increases.

5.5.3 Experiments on the Effects of Hyperspherical Embedding

In this section, we study the effects of hyperspherical embedding and provide empirical results and visualizations that align with the conclusions claimed in Sec 5.4.

Increased Magnitude of Embedding From Proposition V.2, the magnitude of an embedding get increased during optimization, and we visualize the distribution of embedding magnitude from test set on MVP dataset in Figure 5.3. When optimizing our proposed hyperspherical embedding denoted by (Hyper), the scale of embedding's magnitude before normalization is significantly larger than those unnormalized denoted by (Unconstrained and Transformed). Furthermore, we observe in two leftmost subfigures that the range of embeddings' magnitude changes little after only transformed by a MLP layer, which validates that normalization is the key to increase embedding magnitude. Even though the effect is based on vanilla gradient descent, we still find it empirically holds for modern optimizers, such as Adam and RMSProp shown in the two rightmost subfigures of Figure 5.3.

Enabling Wider Range of Learning Rates When optimizing a normalized embedding, the gradient at each update can be computed by following the Equation 5.3, and it indicates that the magnitude of gradient is inversely proportional to the norm of the embedding shown in Proposition V.3. One benefit of this finding is that the increased norms of embeddings help neural networks gain robustness to varying values of learning rate. For instance, if the learning rate is large, the norm of embeddings before normalization quickly increases and makes the effective learning rate small enough to stabilize training. Figure 5.9 (Figure B.2 and Figure B.1 are in Appendix B) show the results of models on different tasks using different learning rates. Compared to the point cloud completion results with unconstrained embedding, the models using hyperspherical embeddings perform more stably under a wider range of learning rates with better performance in particular for large learning rates.

Compact Embedding Distribution Based on Proposition V.2 and Proposition V.4, optimizing hyperspherical embedding drives the model to learn a more singular weight. We empirically verify this by visualizing the singular value distribution of weights trained with point cloud completion task on MVP dataset, and the results are shown in Figure 5.2. We observe that singular values learned with hyperspherical embedding have a larger value span than those without normalized embedding, while the majority of them are located near zero.

The resulting singular weights make the learned embedding distributed more compactly in angular space. Figure 5.4 illustrates the angular distribution of embeddings by computing cosine similarity of pairwise embeddings from the MVP test set. Adding a MLP layer to transform the unconstrained embedding does not change the span of angular distribution significantly, while the hyperspherical embeddings have much narrower angular span and are distributed more compactly on both single class and overall classes. Compared to a compact embedding distribution, one disadvan-

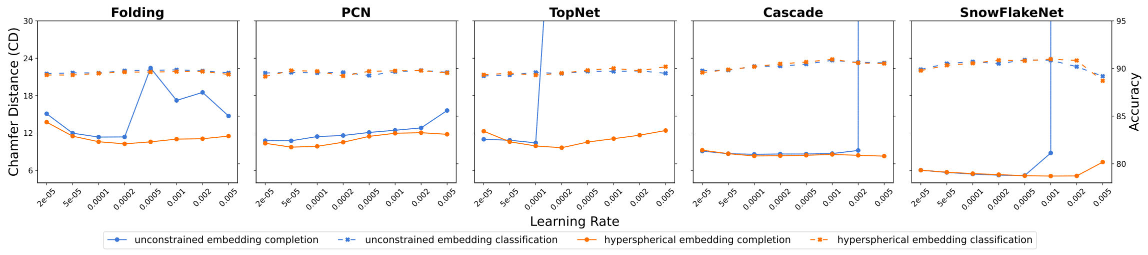


Figure 5.9: Performance of multi-task learning of point cloud completion and classification on MVP with different learning rates.

	Single Task		Equal Weights		PCGrad [112]		Uncert. [109]		Weight Search		
Model	Acc	CD	Acc	CD	Acc	CD	Acc	CD	Acc	CD	S. vs. M.
Folding	89.68	10.39	89.77	11.37	89.67	11.21	89.81	11.22	89.12	10.45	-0.58
Folding (H)	89.91	**9.47**	89.63	**10.26**	89.77	**10.13**	89.51	**10.07**	89.43	**9.40**	**0.74**
PCN	89.62	10.03	89.58	10.75	89.41	10.75	89.33	10.77	89.26	10.37	-3.39
PCN (H)	89.55	**9.52**	89.79	**9.73**	89.56	**9.58**	89.69	**9.58**	89.78	**9.45**	**0.74**
TopNet	89.49	10.12	89.59	10.42	89.84	10.33	89.58	10.52	89.43	10.24	-1.19
TopNet (H)	89.55	**9.50**	89.51	**9.64**	89.80	**9.59**	89.90	**9.48**	89.74	**8.79**	**7.47**
Cascade	90.91	7.49	90.23	8.58	90.33	8.53	90.27	8.51	90.18	7.50	-0.13
Cascade (H)	90.51	**7.25**	90.19	**8.32**	90.02	**8.18**	90.32	**8.32**	90.48	**7.22**	**0.41**
SnowFlakeNet	90.93	5.37	90.90	5.19	90.99	5.29	90.18	5.27	90.75	5.04	**6.15**
SnowFlakeNet (H)	90.91	**5.21**	90.98	**5.09**	90.95	**5.21**	90.13	**5.11**	90.82	**5.02**	3.65

Table 5.4: Comparison results of models using different multi-task training strategies on MVP dataset. Results of shape classification (Acc) and point cloud completion (CD) are reported, multipied by 10^4. "S. vs. M." shows the percentage of performance change comparing best completion results in multi-task learning to those in single-task learning.

tage of sparse embedding distribution is to increase the possibility of unseen features falling far away from seen features at training, which inevitably worsens the generalization of models at testing. To visually demonstrate the degree of sparsity in the embedding space, we use trained models that perform point cloud reconstruction on ModelNet40 to interpolate embeddings on the embedding space and generate point clouds, and the results are shown in Figure 5.8. Compared to interpolated results from unconstrained embedding space, the generated point clouds with hyperspherical embeddings have more clear clues from source or target shapes, because the interpolated hyperspherical embeddings are closer to features captured at training. Thus, it helps with generating more reasonable shapes of objects at testing.

Improvement on Multi-task Learning In addition to single-task learning, the proposed hyperspherical module also improves the performance of point cloud completion in multi-task learning with other semantic tasks. To verify this, we report results of multi-task learning on three different datasets: ModelNet40, ShapeNet, and MVP. We report the results of joint training point cloud completion and shape classification on MVP in Figure 5.9. Since the ModelNet40 and ShapeNet do not provide pairs of partial and complete point clouds, results of point cloud reconstruction are reported along with shape classification on ModelNet in Figure B.2 (in Appendix B) and part segmentation on ShapeNet in Figure B.1 (in Appendix B) . By comparing the results of models with unconstrained embeddings, the proposed hyperspherical module has little effect on the performance of semantic tasks. However, models with the proposed module have more stable performance when using large learning rates than those with unconstrained embeddings, since the same setting tend to cause training unconverged. In terms of converged results, models with our method still outperform their baselines with noticeable improvement.

To make a fair comparison, we also report results of other approaches developed for multi-task learning using different types of embeddings trained on MVP dataset as shown in Table 5.4. The overall classification accuracy and Chamfer Distance multiplied by 10^4 are reported. The second to fifth columns show results of models with different training strategies indicated by the column title. Unsurprisingly, models with the proposed hyperspherical modules outperform the baselines on point cloud completion in both single-task and multi-task learnings under all settings. By comparing the completion results in multi-task learning, manually searching the optimal weights between completion task and classification task takes much time, but it achieves the best completion results with little affection on classification performance. Furthermore, the rightmost column (S. vs. M.) presents the percent of changes comparing the best completion results in multi-task learning to those in single-task learning. It shows improvement of completion performance trained in multi-task learning when using our method, while the models with unconstrained embeddings struggle in degradation of completion performance when they are trained with shape classification. One exception is SnowFlakeNet with unconstrained embeddings, which gets improved on completion when trained with classification. We argue that it is due to multi-task learning helps with reducing the overfitting issue observed when training SnowFlakeNet in single-task learning, but our proposed method (5.02) still outperforms its baseline (5.04).

To delve into the effects of hyperspherical embedding on multi-task learning, we visualize the conflicts between tasks when training them jointly in Figure 5.5. Specifically, the measure we visualize is the cosine similarity of gradients on the shared encoders with respect to different task losses, where negative values indicate conflicting gradients, as proposed by [112]. The visualization shows that the gradient cosine similarity with hyperspherical embeddings are almost positive, while those with unconstrained embeddings tend to be more negative, which explains the improvement of point cloud completion in multi-task learning with classification observed in Table 5.4. Moreover, we find that classification dominates the training since its magnitude of gradient is significantly larger than gradients with respect to completion loss, as shown in the right subfigure in Figure 5.5. This aligns well with the results of weight search experiments, where smaller classification weight or larger completion weight lead to better completion performance.

5.6 Conclusions

This chapter proposes a general module for point cloud completion. In particular, our hyperspherical module transforms and normalizes the output from the encoder onto the surface of a hypersphere before it is processed by the following decoders. We study the effects of the proposed hyperspherical embeddings in both theoretical and experimental ways. Extensive experiments are performed on synthetic and real-world dataset, and the achieved state-of-the-art results in both single-task and multi-task learnings demonstrate the effectiveness of the proposed method.

CHAPTER VI

Future Directions

This thesis targets the challenges in 3D scene understanding by developing algorithms for generating depth maps and point cloud analysis, but there are ample room for development of other directions. This chapter outlines several main ideas for future research as follows.

3D scene understanding with limited data

Thanks to the deep learning techniques and the accessibility of large datasets, existing models have achieved impressive performance on tasks in 3D scene understanding. The methods developed in this thesis also rely on large datasets and focus on how to effectively learn the representations from point clouds. However, accessibility of large datasets is not always guaranteed since annotating data, especially in 3D space, is known to be difficult and expensive. Furthermore, the lack of ample training data will significantly degrade the performance of deep learning models. Thus, obtaining models with the ability to perform similarly well with limited amounts of training data is desirable. Specifically, learning 3D data with one-shot learning or unsupervised learning has the potential to broaden the applications that require 3D scene understanding while have limited access to training data.

Transferring knowledge from synthetics to real-world applications

Another way to address the inaccessibility of large 3D datasets is to obtain relatively cheap synthetic datasets. For example, the methods developed in this thesis as well as other approaches propose to experiment on synthetic point clouds generated by sampling on the surfaces of CAD models of objects; sophisticated simulators have been designed to simulate the interaction between vehicles for generating self-driving car datasets. Since there is a discrepancy between the synthetic data and real-world measurement, directly applying the models trained on synthetic data perform worse on real-world data. In particular, we are interested in transferring the knowledge learned from completion on synthetic point clouds to real-world completion applications, since annotating the groundtruth complete shapes of objects in real-world measurement is almost infeasible. And the ability to predict complete shapes from 3D measurement would benefit many downstream tasks, such as robot arms for grasping unknown objects.

Compact modeling of shapes

Compared to voxels, point clouds are a more compact representation of objects without quantization loss, so methods developed in this thesis focus on representing objects using point clouds. The architectures we used are able to generate point clouds with fine details at a preset resolution. However, this fixed resolution of point clouds may raise challenges for different downstream tasks. For example, the high-resolution point clouds are needed for analyzing tasks, while saving those point clouds will consume a fair amount of storage space. Therefore, it is desirable to represent objects in a compact representation and have the ability to be converted into arbitrary resolution of point clouds at the same time. More recent works proposed to model the shape of objects using a sign distance function by learning from point clouds, in which the function outputs a value for any point to indicate whether it is inside or outside the objects, and arbitrary resolution of shapes can be generated by sampling in the space. And I believe representing 3D objects in this way has the potential to broaden the analysis of 3D measurement and will also be useful for applications requiring collision check.

APPENDIX

APPENDIX A

Appendix to Point Set Voting

A.1 Appendix to Point Set Voting

Here we provide the derivation of $\mathbf{z}_{opt}$, the optimal latent feature with highest probability in the latent space. The distribution of the latent space $q_\phi(\mathbf{z}|\mathbf{x})$ is represented by a set of multivariate Gaussian distributions shown in Equation (4.5). By assuming that votes are independent and $q_\phi(\mathbf{z}|x_i)$ is Gaussian distributed, the derivation of $\mathbf{z}_{opt}$ is as follows:

$$\begin{aligned}
\mathbf{z}_{opt} &= \underset{z}{\text{argmax}}\ q_\phi(\mathbf{z}|\mathbf{x}) \\
&= \underset{z}{\text{argmax}}\ \log(q_\phi(\mathbf{z}|\mathbf{x})) \\
&= \underset{z}{\text{argmax}}\ \sum_{i=1}^{n} \log(q_\phi(\mathbf{z}|x_i)) \\
&= \underset{z}{\text{argmax}}\ \sum_{i=1}^{n} \log\left(\frac{1}{(2\pi)^{m/2}|\Sigma|_i^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{z}-\mu_i)^T\Sigma_i^{-1}(\mathbf{z}-\mu_i)\right)\right) \\
&= \underset{z}{\text{argmax}}\ -\sum_{i=1}^{n} \log\left((2\pi)^{m/2}|\Sigma|_i^{1/2}\right) + \sum_{i=1}^{n}\left(-\frac{1}{2}(\mathbf{z}-\mu_i)^T\Sigma_i^{-1}(\mathbf{z}-\mu_i)\right) \\
&= \underset{z}{\text{argmax}}\ \sum_{i=1}^{n}\left(-\frac{1}{2}(\mathbf{z}-\mu_i)^T\Sigma_i^{-1}(\mathbf{z}-\mu_i)\right)
\end{aligned} \tag{A.1}$$

where a multivariate Gaussian distribution is characterized by mean vector μ_i and covariance matrix Σ_i; n is the number of votes; m is the dimension of the latent space. The solution to optimizing $q_\phi(\mathbf{z}|\mathbf{x})$ can be computed by setting derivative to zero,

$$\begin{aligned}
0 &= \frac{\partial \sum_{i=1}^{n}\left(-\frac{1}{2}(\mathbf{z}-\mu_i)^T\Sigma_i^{-1}(\mathbf{z}-\mu_i)\right)}{\partial \mathbf{z}} \\
&= \sum_{i=1}^{n} \Sigma_i^{-1}(\mathbf{z}-\mu_i)
\end{aligned} \tag{A.2}$$

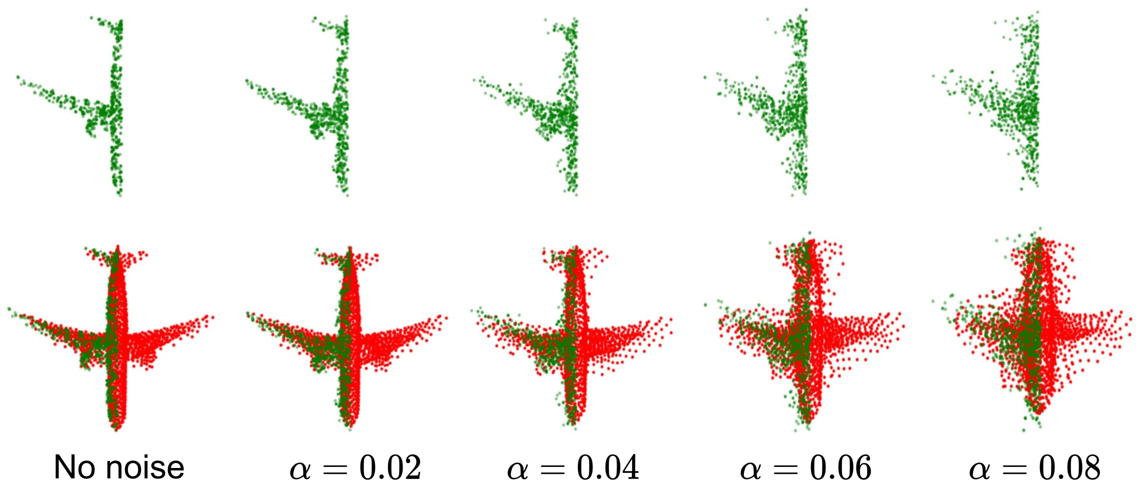


Figure A.1: Results of point clouds completion obtained from the noisy partial observation. Gaussian noise with zero mean is assumed and the standard deviation is indicated at the bottom. Top: input partial observation. Bottom: prediction (red) overlapped with inputs (green).

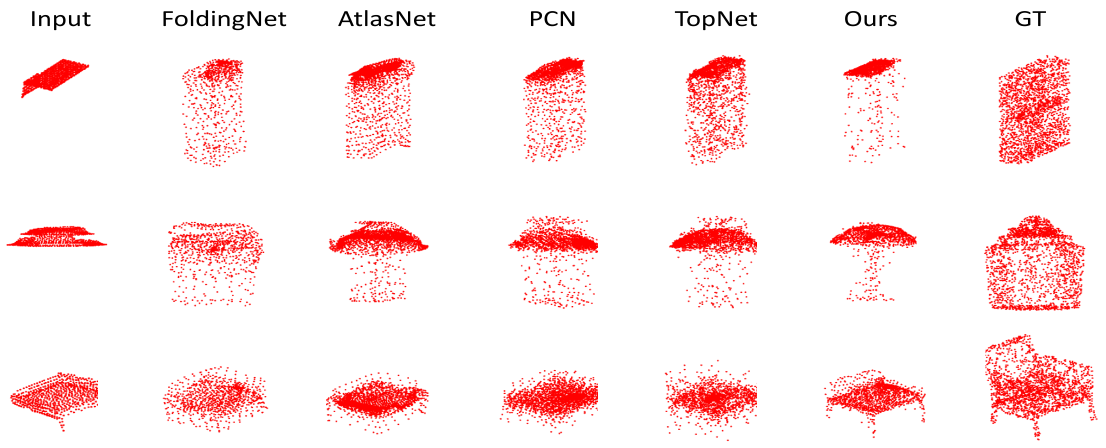


Figure A.2: Failure cases of point clouds completion on the Completion3D dataset.

and the maximizing argument $\mathbf{z}_{opt}$ of $q_\phi(\mathbf{z}|\mathbf{x})$ is given by

$$\mathbf{z}_{opt} = \frac{\sum_{i=1}^{n} \Sigma_i^{-1} \mu_i}{\sum_{i=1}^{n} \Sigma_i^{-1}} \tag{A.3}$$

For simplicity, we assume diagonal covariance matrix during experiments. Both μ_i and Σ_i are generated from each local point set, and modeled by neural networks.

A.2 Visualization of Point Clouds Completion with Noisy Inputs

We visualize the results of point clouds completion with added noise in the Figure A.1. Input partial point clouds are perturbed using Gaussian noise with zero mean, and the standard deviation differs in experiments as they are indicated at bottom of the figure. It shows that the proposed model tends to maintain input partial shapes and lacks the ability to distinguish noise points.

A.3 Failure Cases on Point Clouds Completion

We show failure cases of point clouds completion on Completion3D in the Figure A.2. Given partial observation with no distinct geometric information, all models fail to generate correct complete point clouds. However, the method developed in this paper is able to generate sharp and reasonable completion, while outputs of other approaches are blurry. We argue the reason is that other approaches tend to generate a mean shape of training data when difficult partial point clouds are observed. However, the proposed model predicts reasonable complete shapes.

APPENDIX B

Appendix to Hyperspherical Point Cloud Completion

B.1 Proof of Proposition

Suppose the normalization process follows Equation 5.2, and the loss at optimization is denoted by L, and the gradient to embedding f follows Equation 5.3. Based on them, we show the orthogonality between an embedding and its gradient by computing the their inner product:

$$\begin{aligned}\langle f, \frac{\partial L}{\partial f}\rangle &= \frac{\langle f, \frac{\partial L}{\partial \hat{f}}\rangle - \langle f, \hat{f}\rangle\langle \frac{\partial L}{\partial \hat{f}}, \hat{f}\rangle}{\|f\|_2} \\ &= \frac{\langle f, \frac{\partial L}{\partial \hat{f}}\rangle - \langle \hat{f}, \hat{f}\rangle\langle \frac{\partial L}{\partial \hat{f}}, f\rangle}{\|f\|_2} \\ &= \frac{\langle f, \frac{\partial L}{\partial \hat{f}}\rangle - \langle \frac{\partial L}{\partial \hat{f}}, f\rangle}{\|f\|_2} \\ &= 0\end{aligned} \tag{B.1}$$

B.2 More Experiments

ModelNet40 ModelNet40 dataset contains 12,311 shapes from 40 object categories, and they are split into 9,843 for training and 2,468 for testing. Since the dataset does not provide partial point clouds, we evaluate our proposed method on performing point cloud reconstruction and shape classification. We generate input point clouds by evenly sampling 1024 points from the surface of objects and normalize them within a unit sphere, and no data augmentation is used at training.

Quantitative results on Modelnet40 [59] are shown in Table B.2. To make a fair comparison, we report results of combination of two popular encoders, PointNet [19] without T-Net and DGCNN [22], and three different point cloud decoders that are Folding [88], PCN [89] and TopNet [90]. The baseline models are compared

Model	Seg Acc.	CD
PointNet-Folding	92.06	50.08
PointNet-Folding (H)	92.01	**34.75**
PointNet-PCN	92.06	43.61
PointNet-PCN (H)	92.01	**38.18**
PointNet-TopNet	92.06	37.4
PointNet-TopNet (H)	92.01	**35.50**
DGCNN-Folding	92.50	49.21
DGCNN-Folding (H)	92.39	**33.88**
DGCNN-PCN	92.50	42.42
DGCNN-PCN (H)	92.39	**37.11**
DGCNN-TopNet	92.50	36.80
DGCNN-TopNet (H)	92.38	**35.10**

Table B.1: Single-task learning on ShapeNet. Overall point segmentation accuracy (Seg Acc.) is reported for part segmentation, and Chamfer Distance (CD) is reported for point cloud reconstruction, multiplied by 10^4. The first column describes the encoders and decoders used in the model, and "H" indicates using the proposed hyperspherical module.

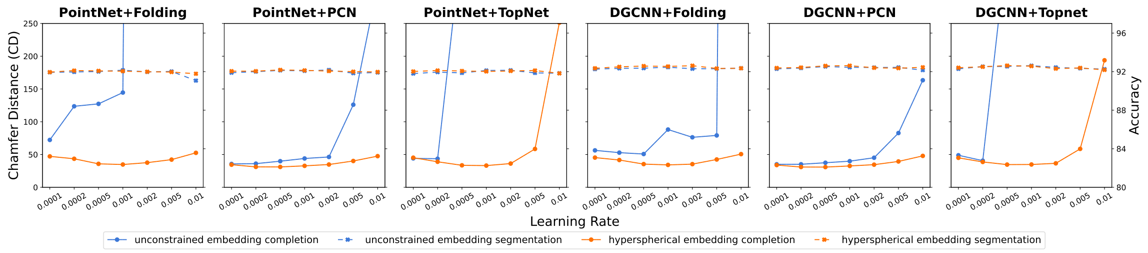


Figure B.1: Performance of multi-task learning of point cloud reconstruction and part segmentation on ShapeNet with different learning rates.

to its variants added with our proposed hyperspherical modules denoted with (H). As shown by the last column in Table B.2, our proposed hyperspherical module helps baseline approaches gain noticeable decrease of Chamfer Distance in all cases. We also test the proposed module in shape classification by removing point cloud decoders. As shown in the second column, the proposed method module leads to slightly better performance of shape classification. Multi-task learning results on shape reconstruction and classification are shown in Figure B.2. By comparing the results of models with unconstrained embeddings, the proposed hyperspherical module have little effect on the performance of semantic tasks. However, models with the proposed module have more stable performance when using large learning rates than those with unconstrained embeddings, since the same setting tend to cause training unconverged. In terms of converged results, models with our method still outperform their baselines with noticeable improvement.

ShapeNet ShapeNet part dataset [59] dataset contains 16,881 shapes from 16 object categories with 50 parts. Each point cloud contains 2048 points which are generated by evenly sampling from the surface of objects, and we follow the same

Model	Cls Acc.	CD
PointNet-Folding	87.33	75.86
PointNet-Folding (H)	87.36	**48.88**
PointNet-PCN	87.33	48.17
PointNet-PCN (H)	87.36	**43.55**
PointNet-TopNet	87.33	55.04
PointNet-TopNet (H)	87.36	**49.65**
DGCNN-Folding	89.22	70.32
DGCNN-Folding (H)	89.47	**45.37**
DGCNN-PCN	89.22	46.54
DGCNN-PCN (H)	89.47	**42.70**
DGCNN-TopNet	89.22	55.87
DGCNN-TopNet (H)	89.47	**48.75**

Table B.2: Single-task learning on ModelNet40. Overall classification accuracy (Cls Acc.) is reported for shape classification, and Chamfer distance (CD) is reported for point cloud reconstruction, multiplied by 10^4. The first column describes the encoders and decoders used in the model, and "H" indicates using the proposed hyperspherical module.

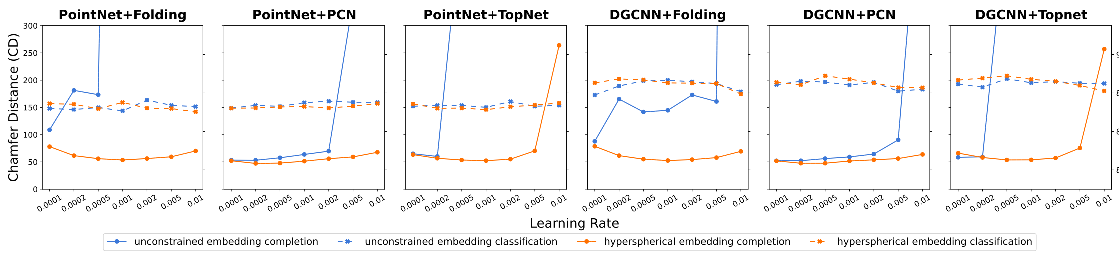


Figure B.2: Performance of multi-task learning of point cloud reconstruction and classification on ModelNet40 with different learning rates.

set splitting as in [20]. Since the dataset does not provide partial point clouds, we evaluate our proposed method on performing point cloud reconstruction and part segmentation.

We report the results on ShapeNet [59] in Table B.1. Similar to models constructed for experiments on ModelNet40, we experiment on a combination of different encoders and decoders. As shown by the last column in Table B.1, the proposed hyperspherical module improves point cloud reconstruction consistently in all cases. When tasked on part segmentation, the point cloud decoders are removed from the models, and the embeddings concatenated with lifted point-wise features are processed by fully connected layers to predict part category. From the second column in Table B.1, part segmentation performance is not affected by the proposed module. Multi-task learning results on shape reconstruction and part segmentation are shown in Figure B.1. By comparing the results of models with unconstrained embeddings, the proposed hyperspherical module have little effect on the performance of semantic tasks. However, models with the proposed module have more stable performance when using large learning rates than those with unconstrained embeddings, since the

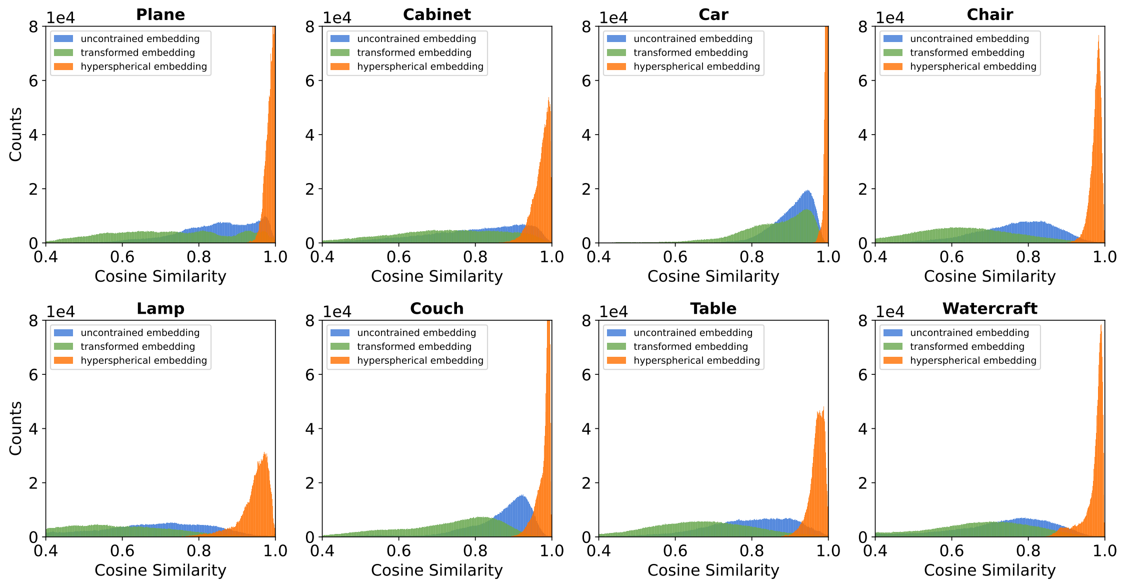


Figure B.3: Cosine similarity distribution of embeddings. We compute pairwise cosine distance between embeddings obtained from the test set in MVP dataset. We visualize distribution of different classes as described in the plot titles. Hyperspherical embeddings have more compact angular distribution.

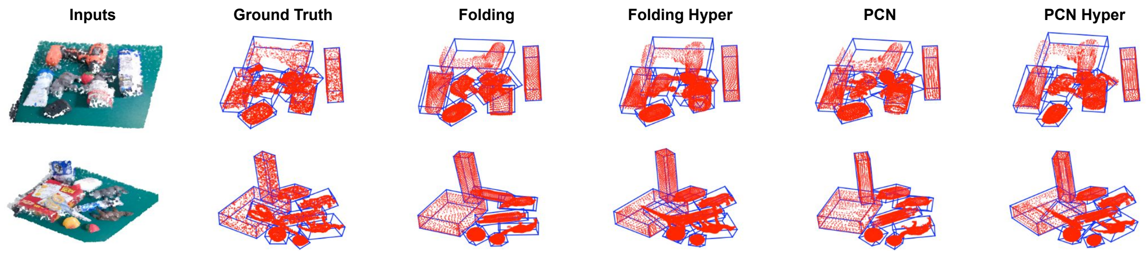


Figure B.4: More qualitative 3D detection, pose estimation, and point cloud completion results on GraspNet test set

same setting tend to cause training unconverged. In terms of converged results, models with our method still outperform their baselines with noticeable improvement.

B.3 More visualization

We show the angular distribution of embeddings by computing the pairwise cosine similarity obtained from the test set in MVP dataset. More visualizations of different classes as described in the plot titles are shown in Figure B.3, and the distribution of overall classes is shown in Figure 5.5

More qualitative results of 3D object detection, pose estimation, and point cloud completion can be found in Figure B.4.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[4] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[5] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2881–2890.

[6] Y. Feng, Z. Zhang, X. Zhao, R. Ji, and Y. Gao, "GVCNN: Group-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 264–272.

[7] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 922–928.

[8] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view CNNs for object classification on 3d data," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5648–5656.

[9] Y. Feng, Y. Feng, H. You, X. Zhao, and Y. Gao, "Meshnet: Mesh neural network for 3d shape representation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 8279–8286.

[10] G. Gkioxari, J. Malik, and J. Johnson, "Mesh R-CNN," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9785–9795.

[11] J. Joglekar, S. S. Gedam, and B. K. Mohan, "Image matching using sift features and relaxation labeling technique—a constraint initializing method for dense stereo matching," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 9, pp. 5643–5652, 2014.

[12] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[13] J. Zbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *Journal of Machine Learning Research*, vol. 17, no. 1-32, p. 2, 2016.

[14] W. Luo, A. G. Schwing, and R. Urtasun, "Efficient deep learning for stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5695–5703.

[15] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4040–4048.

[16] A. Kendall, H. Martirosyan, S. Dasgupta, and P. Henry, "End-to-end learning of geometry and context for deep stereo regression," in *IEEE International Conference on Computer Vision*. IEEE, 2017, pp. 66–75.

[17] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[18] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 652–660.

[20] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in neural information processing systems*, 2017, pp. 5099–5108.

[21] Y. Liu, B. Fan, S. Xiang, and C. Pan, "Relation-shape convolutional neural network for point cloud analysis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8895–8904.

[22] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 5, p. 146, 2019.

[23] X. Li, J. K. Pontes, and S. Lucey, "Pointnetlk revisited," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12 763–12 772.

[24] J. Lee, S. Kim, M. Cho, and J. Park, "Deep hough voting for robust global registration," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15 994–16 003.

[25] S. Shi, X. Wang, and H. Li, "PointRCNN: 3d object proposal generation and detection from point cloud," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 770–779.

[26] C. R. Qi, O. Litany, K. He, and L. J. Guibas, "Deep hough voting for 3d object detection in point clouds," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9277–9286.

[27] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "Learning semantic segmentation of large-scale point clouds with random sampling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[28] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[29] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2446–2454.

[30] J. Zhang, K. A. Skinner, R. Vasudevan, and M. Johnson-Roberson, "Dispsegnet: Leveraging semantics for end-to-end learning of disparity estimation from stereo imagery," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1162–1169, 2019.

[31] J. Zhang, M.-Y. Yu, R. Vasudevan, and M. Johnson-Roberson, "Learning rotation-invariant representations of point clouds using aligned edge convolutional neural networks," in *2020 International Conference on 3D Vision (3DV)*. IEEE, 2020, pp. 200–209.

[32] J. Zhang, W. Chen, Y. Wang, R. Vasudevan, and M. Johnson-Roberson, "Point set voting for partial point cloud analysis," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 596–603, 2021.

[33] W.-S. Jang and Y.-S. Ho, "Efficient disparity map estimation using occlusion handling for various 3d multimedia applications," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, 2011.

[34] S. Adhyapak, N. Kehtarnavaz, and M. Nadin, "Stereo matching via selective multiple windows," *Journal of Electronic Imaging*, vol. 16, no. 1, p. 013012, 2007.

[35] T. Kanade and M. Okutomi, "A stereo matching algorithm with an adaptive window: Theory and experiment," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. IEEE, 1991, pp. 1088–1095.

[36] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2758–2766.

[37] G. Yang, H. Zhao, J. Shi, and J. Jiaya, "Segstereo: Exploiting semantic information for disparity estimation," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 636–651.

[38] J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5410–5418.

[39] Z. Liang, Y. Feng, Y. G. H. L. W. Chen, and L. Q. L. Z. J. Zhang, "Learning for disparity estimation through feature constancy," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2811–2820.

[40] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3061–3070.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Adv. Neural Info. Process. Syst.*, 2012, pp. 1097–1105.

[42] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[43] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[44] J. Pang, W. Sun, J. S. Ren, C. Yang, and Q. Yan, "Cascade residual learning: A two-stage convolutional neural network for stereo matching," in *2017 IEEE International Conference on Computer Vision Workshop*. IEEE, 2017, pp. 878–886.

[45] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2462–2470.

[46] R. Garg, V. K. BG, G. Carneiro, and I. Reid, "Unsupervised CNN for single view depth estimation: Geometry to the rescue," in *European Conference on Computer Vision.* Springer, 2016, pp. 740–756.

[47] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 270–279.

[48] C. Zhou, H. Zhang, X. Shen, and J. Jia, "Unsupervised learning of stereo matching," in *2017 IEEE International Conference on Computer Vision.* IEEE, 2017, pp. 1576–1584.

[49] N. Luo, C. Yang, W. Sun, and B. Song, "Unsupervised stereo matching with occlusion-aware loss," in *Pacific Rim International Conference on Artificial Intelligence.* Springer, 2018, pp. 746–758.

[50] Y. Zhong, Y. Dai, and H. Li, "Self-supervised learning for stereo matching with self-improving ability," *arXiv preprint arXiv:1709.00930*, 2017.

[51] F. Guney and A. Geiger, "Displets: Resolving stereo ambiguities using object knowledge," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4165–4175.

[52] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[53] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[54] D. Kinga and J. B. Adam, "A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, vol. 5, 2015.

[55] A. Ahmadi and I. Patras, "Unsupervised convolutional neural networks for motion estimation," in *Image Processing (ICIP), 2016 IEEE International Conference on.* IEEE, 2016, pp. 1629–1633.

[56] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[57] A. Kanezaki, Y. Matsushita, and Y. Nishida, "Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5010–5019.

[58] J.-C. Su, M. Gadelha, R. Wang, and S. Maji, "A deeper look at 3d shape classifiers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 0–0.

[59] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[60] C. Esteves, C. Allen-Blanchette, A. Makadia, and K. Daniilidis, "Learning so (3) equivariant representations with spherical CNNs," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 52–68.

[61] Y. Rao, J. Lu, and J. Zhou, "Spherical fractal convolutional neural networks for point cloud recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 452–460.

[62] Y. You, Y. Lou, Q. Liu, L. Ma, W. Wang, Y. Tai, and C. Lu, "Prin: Pointwise rotation-invariant network," *arXiv preprint arXiv:1811.09361*, 2018.

[63] C. Chen, G. Li, R. Xu, T. Chen, M. Wang, and L. Lin, "Clusternet: Deep hierarchical cluster network with rigorously rotation-invariant representation for point cloud analysis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4994–5002.

[64] Z. Zhang, B.-S. Hua, D. W. Rosen, and S.-K. Yeung, "Rotation invariant convolutions for 3d point clouds deep learning," in *2019 International Conference on 3D Vision (3DV)*. IEEE, 2019, pp. 204–213.

[65] X. Sun, Z. Lian, and J. Xiao, "Srinet: Learning strictly rotation-invariant representations for point cloud classification and segmentation," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 980–988.

[66] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[67] R. Klokov and V. Lempitsky, "Escape from cells: Deep kd-networks for the recognition of 3d point cloud models," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 863–872.

[68] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN: Octree-based convolutional neural networks for 3d shape analysis," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 72, 2017.

[69] M. Tatarchenko, A. Dosovitskiy, and T. Brox, "Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2088–2096.

[70] J. Li, B. M. Chen, and G. Hee Lee, "So-net: Self-organizing network for point cloud analysis," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9397–9406.

[71] Y. Zhang and M. Rabbat, "A graph-CNN for 3d point cloud classification," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 6279–6283.

[72] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9621–9630.

[73] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution on x-transformed points," in *Advances in Neural Information Processing Systems*, 2018, pp. 820–830.

[74] L. Yi, V. G. Kim, D. Ceylan, I. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, L. Guibas, *et al.*, "A scalable active framework for region annotation in 3d shape collections," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, p. 210, 2016.

[75] R. Spezialetti, S. Salti, and L. D. Stefano, "Learning an effective equivariant 3d descriptor without supervision," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 6401–6410.

[76] A. Mian, M. Bennamoun, and R. Owens, "On the repeatability and quality of keypoints for local feature-based 3d object retrieval from cluttered scenes," *International Journal of Computer Vision*, vol. 89, no. 2-3, pp. 348–361, 2010.

[77] F. Tombari, S. Salti, and L. Di Stefano, "Unique signatures of histograms for local surface description," in *European conference on computer vision*. Springer, 2010, pp. 356–369.

[78] A. Petrelli and L. Di Stefano, "On the repeatability of the local reference frame for partial shape matching," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 2244–2251.

[79] J. Žbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *The journal of machine learning research*, vol. 17, no. 1, pp. 2287–2318, 2016.

[80] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kp-conv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 6411–6420.

[81] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[82] P. K. Diederik and M. Welling, "Auto-encoding variational bayes," in *Proceedings of the International Conference on Learning Representations (ICLR)*, vol. 1, 2014.

[83] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *arXiv preprint arXiv:1906.02691*, 2019.

[84] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining point cloud local structures by kernel correlation and graph pooling," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4548–4557.

[85] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 918–927.

[86] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3d object reconstruction from a single image," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 605–613.

[87] Y. Sun, Y. Wang, Z. Liu, J. Siegel, and S. Sarma, "Pointgrow: Autoregressively learned point cloud generation with self-attention," in *The IEEE Winter Conference on Applications of Computer Vision*, 2020, pp. 61–70.

[88] Y. Yang, C. Feng, Y. Shen, and D. Tian, "Foldingnet: Point cloud auto-encoder via deep grid deformation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 206–215.

[89] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert, "Pcn: Point completion network," in *2018 International Conference on 3D Vision (3DV)*. IEEE, 2018, pp. 728–737.

[90] L. P. Tchapmi, V. Kosaraju, H. Rezatofighi, I. Reid, and S. Savarese, "Topnet: Structural point cloud decoder," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 383–392.

[91] D. Stutz and A. Geiger, "Learning 3d shape completion from laser scan data with weak supervision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1955–1964.

[92] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[93] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[94] D. Borrmann, J. Elseberg, K. Lingemann, and A. Nüchter, "The 3d hough transform for plane detection in point clouds: A review and a new accumulator design," *3D Research*, vol. 2, no. 2, p. 3, 2011.

[95] M. Sun, G. Bradski, B.-X. Xu, and S. Savarese, "Depth-encoded hough voting for joint object detection and shape recovery," in *European Conference on Computer Vision.* Springer, 2010, pp. 658–671.

[96] W. Kehl, F. Milletari, F. Tombari, S. Ilic, and N. Navab, "Deep learning of local rgb-d patches for 3d object detection and 6d pose estimation," in *European conference on computer vision.* Springer, 2016, pp. 205–220.

[97] S.-Y. Guo, Y.-G. Kong, Q. Tang, and F. Zhang, "Probabilistic hough transform for line detection utilizing surround suppression," in *2008 International Conference on Machine Learning and Cybernetics*, vol. 5. IEEE, 2008, pp. 2993–2998.

[98] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in neural information processing systems*, 2015, pp. 3483–3491.

[99] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[100] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[101] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[102] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, "A papier-mâché approach to learning 3d surface generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 216–224.

[103] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[104] P. Xiang, X. Wen, Y.-S. Liu, Y.-P. Cao, P. Wan, W. Zheng, and Z. Han, "Snowflakenet: Point cloud completion by snowflake point deconvolution with skip-transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5499–5509.

[105] L. Pan, X. Chen, Z. Cai, J. Zhang, H. Zhao, S. Yi, and Z. Liu, "Variational relational point completion network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8524–8533.

[106] X. Wen, T. Li, Z. Han, and Y.-S. Liu, "Point cloud completion by skip-attention network with hierarchical folding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1939–1948.

[107] X. Wang, M. H. Ang Jr, and G. H. Lee, "Cascaded refinement network for point cloud completion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 790–799.

[108] J. Uhrig, M. Cordts, U. Franke, and T. Brox, “Pixel-level encoding and depth layering for instance-level semantic labeling,” in *German conference on pattern recognition.* Springer, 2016, pp. 14–25.

[109] A. Kendall, Y. Gal, and R. Cipolla, “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7482–7491.

[110] S. Liu, E. Johns, and A. J. Davison, “End-to-end multi-task learning with attention,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 1871–1880.

[111] O. Sener and V. Koltun, “Multi-task learning as multi-objective optimization,” *Advances in neural information processing systems*, vol. 31, 2018.

[112] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, “Gradient surgery for multi-task learning,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 5824–5836, 2020.

[113] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, “Argoverse: 3d tracking and forecasting with rich maps,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[114] X. Huang, P. Wang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, “The apolloscape open dataset for autonomous driving and its application,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 10, pp. 2702–2719, 2019.

[115] X. Wen, P. Xiang, Z. Han, Y.-P. Cao, P. Wan, W. Zheng, and Y.-S. Liu, “Pmp-net: Point cloud completion by learning multi-step point moving paths,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7443–7452.

[116] S. Pu, K. Zhao, and M. Zheng, “Alignment-uniformity aware representation learning for zero-shot video classification,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 19 968–19 977.

[117] T. Wang and P. Isola, “Understanding contrastive representation learning through alignment and uniformity on the hypersphere,” in *International Conference on Machine Learning.* PMLR, 2020, pp. 9929–9939.

[118] D. Zhang, Y. Li, and Z. Zhang, “Deep metric learning with spherical embedding,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 772–18 783, 2020.

[119] X. Zhang, F. X. Yu, S. Karaman, W. Zhang, and S.-F. Chang, “Heated-up softmax embedding,” *arXiv preprint arXiv:1809.04157*, 2018.

[120] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, “Sphereface: Deep hypersphere embedding for face recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 212–220.

[121] F. Wang, X. Xiang, J. Cheng, and A. L. Yuille, “Normface: L2 hypersphere embedding for face verification,” in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1041–1049.

[122] W. Liu, Y.-M. Zhang, X. Li, Z. Yu, B. Dai, T. Zhao, and L. Song, “Deep hyperspherical learning,” *Advances in neural information processing systems*, vol. 30, 2017.

[123] W. Liu, R. Lin, Z. Liu, L. Xiong, B. Schölkopf, and A. Weller, “Learning with hyperspherical uniformity,” in *International Conference On Artificial Intelligence and Statistics.* PMLR, 2021, pp. 1180–1188.

[124] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[125] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning," *Coursera, video lectures*, vol. 264, no. 1, pp. 2146–2153, 2012.

[126] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[127] H.-S. Fang, C. Wang, M. Gou, and C. Lu, "Graspnet-1billion: A large-scale benchmark for general object grasping," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 444–11 453.

[128] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6d object pose estimation in cluttered scenes," *arXiv preprint arXiv:1711.00199*, 2017.