Point clouds analysis with extreme learning machine and high order point CNN

Xu, Rui

2020

Xu, R. (2020). Point clouds analysis with extreme learning machine and high order point CNN. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/137132>

<https://doi.org/10.32657/10356/137132>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 20:15:09 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**POINT CLOUDS ANALYSIS WITH EXTREME
LEARNING MACHINE AND HIGH ORDER
POINT CNN**

**XU RUI
SCHOOL OF ELECTRICAL AND ELECTRONIC
ENGINEERING
2020**

POINT CLOUDS ANALYSIS WITH EXTREME LEARNING MACHINE AND HIGH ORDER POINT CNN

XU RUI

School of Electrical and Electronic Engineering

A thesis submitted to Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Master of Engineering

2020

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

13-August-2019

.....
Date

XU RUI Xu Rui

.....
Student's Name

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accordance with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

13-August-2019

.....
Date

Huang Guangbin

.....
Supervisor's Name

Authorship Attribution Statement

This thesis contains material from one paper accepted at conference in which I am listed as an author.

Chapter 3 is published as R. Xu, Y. Zeng, J. Chen and G.-B. Huang. Point Clouds-based Extreme Learning Machine Auto-Encoder for 3D Shape Classification. In *Proceedings of Extreme Learning Machines*, 2018.

The contributions of the co-authors are as follows:

- Professor Guang-Bin Huang gave some suggestions and instructions on research problems encountered and helped review and edit the manuscript draft.
- Yijie Zeng and Jichao Chen provided technical support on realization of the idea and revised the manuscript draft.
- I put forward the feasible idea. The idea was mainly realized and coded by myself.
- I conducted all experiments and prepared the manuscript draft.

13-August-2019

.....
Date

XU RUI Xu Rui

.....
Student's Name

Acknowledgement

First of all, I would like to sincerely appreciate my supervisor, Prof. Huang Guang-Bin, for his beneficial suggestions and tremendous support during my research. His valuable advice helped me overcome lots of obstacles, especially for Extreme Learning Machine-based method. His cheerful encouragement motivated me to tackle numerous challenging problems. Moreover, I am grateful for the weekly discussions between my supervisor and me which gave me a plenty of inspirations. Last but not least, I would like to thank my supervisor for reviewing my thesis and giving suggestions on revision.

Secondly, I would like to thank Mr. Jichao Chen for guiding me into the field of point clouds analysis. I also appreciate Mr. Jichao Chen and Mr. Yijie Zeng who offered lots of help with refinement of ideas, paper revision and code optimization. Furthermore, I show my gratitude for the encouragement given by them so that decent achievements were realized. More importantly, I sincerely show my appreciation for their patient explanation of tough questions.

Thirdly, I would like to appreciate Associate Prof. Chan Chok You, John, Associate Prof. Song Qing and Associate Prof. Mao Kezhi for their suggestions on my MEng Confirmation Exercise.

Finally, I would like to thank ST Engineering-NTU Robotics Corporate Laboratory for providing devices and supporting.

Contents

Summary	v
List of Figures	vii
List of Tables	ix
1 Introduction	1
1.1 Analyzing 3D Data	1
1.1.1 View-based Representation	2
1.1.2 Volumetric Representation	3
1.1.3 Point Clouds Representation	3
1.2 Challenges of Point Clouds Analysis	4
1.2.1 Ambiguous Definition of Neighbors	5
1.2.2 Sparsity of Point Clouds	5
1.2.3 Unorderedness of Points	6
1.3 Algorithms	7
1.3.1 Extreme Learning Machine-based Algorithm	8
1.3.2 High Order Point CNN	9
1.4 Organization of the Thesis	11
2 Literature Review	12
2.1 3D Shape Analysis with View-based Representation	12
2.2 3D Shape Analysis with Volumetric Representation	13

2.3	3D Shape Analysis with Point Clouds	18
2.4	Preliminaries of ELM	26
2.4.1	ELM	27
2.4.2	ELM-AE	29
2.4.3	Multi-layer ELM	30
2.5	Bilinear Models	31
2.5.1	Bilinear Pooling	31
2.5.2	Methods to Reduce Complexity of Bilinear Pooling	33
2.5.3	Bilinear Pooling for 3D Object Recognition	36
3	Point Clouds-based Extreme Learning Machine Auto-Encoder	39
3.1	Network Architecture	40
3.1.1	Feature Extractor	40
3.1.2	Local Structure Module	42
3.1.3	ELM Classifier	43
3.2	Network Properties	44
3.2.1	Back-propagation Free	44
3.2.2	Permutation Invariance	45
3.2.3	Hierarchy	46
3.3	Experiments	48
3.3.1	Implementation	48
3.3.2	Performance on Benchmarks	49
3.3.3	Network Design Analysis	51
3.3.4	Ablation Study	53
3.4	Conclusion	54
4	Permutation-Invariant High Order Point CNN	56
4.1	The Problem of Direct Convolution	57
4.2	Bilinear Pooling Bonded to Convolution	58
4.3	Constraining the Rank of Convolutional Kernels	59

4.4	Generalizing to Trilinear Pooling	60
4.5	Merging BLC and TLC	62
4.6	The Architectures	63
4.6.1	ALC Module	64
4.6.2	SLC Module	64
4.6.3	High Order Convolution	64
4.6.4	High Order MLP	65
4.6.5	Object Classification Network	66
4.6.6	Part Segmentation Network	66
4.7	Experiments	67
4.7.1	Datasets	67
4.7.2	Implementation Specifics	69
4.7.3	Classification Results	69
4.7.4	Segmentation Results	71
4.7.5	Ablation Study and Visualization	71
4.8	Conclusion	73
5	Conclusions and Future Work	75
5.1	Point Clouds-based ELM-AE	75
5.2	High Order Point CNN	76
5.3	Comparison	77
5.4	Future Work	78
	Bibliography	80
	Appendices	85
A	Overview	85
B	Illustration of Ablated PC-ELM-AE Networks	85
C	Mathematical Derivation of Formulas in HOPCNN	86
C.1	Formulas of SBLC	86

C.2	Formulas of ABLC	87
C.3	Formulas of STLC	88
C.4	Formulas of ATLC	90
C.5	Formulas of SLC and ALC	92
D	Hyper-parameters of MLPs used in HOPCNN	93
D.1	Object Classification Network	94
D.2	Part Segmentation Network	95
E	Visualization of Part Segmentation of HOPCNN	97

Summary

The thesis is going to focus on point clouds analysis, especially on 3D object classification and part segmentation of point clouds. Point clouds analysis of 3D models and 3D scenes attracts extensive research attention recently, not only due to the availability of several large repositories but also because of its wide applications. Point clouds are commonly used in industry, *e.g.* autonomous vehicles and robotics use LIDAR to collect point clouds of scenes, which works as an auxiliary recognition tool of 3D space. Therefore, proposing reliable and feasible algorithms for point clouds recognition and analysis is significant and beneficial.

The main barrier that prevents advanced machine learning algorithms from generalizing to point clouds stems from the inborn properties of point clouds, *i.e.* sparsity, disconnectedness and unorderedness. Since point clouds are generated by sampling from surfaces of 3D models, they only occupy contours of models, meaning they are naturally sparse in 3D space compared to grid-like data such as volumetric data. Apparently, the distance between two arbitrary points is continuous, and any pair of points is not explicitly connected. Furthermore, any permutation of the same set of points does not distort the physical model they stands for, which means point clouds are unordered. These three properties of point clouds make most of machine learning algorithms incompatible which usually take as input dense, connected and ordered grids of data, *e.g.* images. Consequently, proposing new algorithms or generalizing existing algorithms to point clouds requires breaking the above-mentioned constraints, which is quite challenging.

Two novel algorithms are presented in this thesis. In order to handle the sparsity and disconnectedness of point clouds, both of them utilize k nearest neighbors (kNN) to form a kNN Graph (kNNG) for each model, where every point is a vertex, and every edge connects two neighboring vertices. Thus, kNNGs build connections between neighboring points to some extent. As for the unorderedness, a symmetric pooling function, *i.e.* max pooling, is applied to every local point region for generating order-irrelevant features in the first algorithm which is based on Extreme Learning Machine (ELM). Meanwhile,

bilinear pooling and trilinear pooling are used prior to convolutions to extract order-irrelevant features in the second algorithm. The use of bi/trilinear poolings enables the generalization of convolution operators to sparse and unordered point clouds.

The first ELM-based algorithm is an extremely fast method in training that is suitable for the situation that training time is limited. ELM is a single-hidden layer feedforward neural network (SLFN) that is considered to be a universal approximator getting rid of iterative updates of parameters. Different from classic SLFNs where all parameters should be tuned, ELM randomly generates input weights and biases that need not be tuned afterwards. The introduced ELM-based algorithm stacks multiple Extreme Learning Machine Auto-Encoders (ELM-AEs) to extract discriminative features from raw point clouds and leverages a single-hidden layer ELM classifier for classification. This algorithm is named as Point Clouds-based Extreme Learning Machine Auto-Encoder (PC-ELM-AE). Experiments on ModelNet40 and ModelNet10 datasets show PC-ELM-AE outperforms existing ELM-based methods and surpasses some deep networks with much less training time for 3D object classification.

The second algorithm is a Deep Neural Network (DNN) that stacks multiple convolution layers for hierarchical feature extraction. It directly takes unordered point clouds as input and outputs permutation invariant features. As mentioned before, directly using classic convolutional neural networks (CNNs) on unordered point clouds is unsuited because convolutional output is non-deterministic. The non-determinism may result in conflicting inferences of 3D shapes, thus, poor performance is inevitable for networks processing point clouds by classic CNNs. Inspired by bilinear models proposed for fine-grained recognition and visual question answering (VQA), bilinear pooling and trilinear pooling are implemented before convolutions to generate deterministic features regardless of points order. In detail, bilinear and trilinear poolings sum over outer products of inputs to construct permutation invariant bi/trilinear feature maps that are fed into convolution operators for extracting discriminative features. The integration of bi/trilinear poolings with convolutions forms a novel module named as High Order Convolution (HOC). Given that bilinear pooling can capture the second order statistics and trilinear pooling is able to detect the third order statistics, HOC can simultaneously utilize the expressive capabilities of convolutions and incorporate higher order statistics to enhance feature interactions. To make HOC more efficient, low-rank constraint is explicitly imposed to convolutional kernels to reduce computational cost. A hierarchical network is composed by stacking multiple HOCs. The final network is termed as High Order Point CNN (HOPCNN). Experiments on benchmarks indicate HOPCNN can achieve competitive performance for 3D object classification and 3D object part segmentation.

List of Figures

1.1	Illustration of mesh-based 3D model, view-based representation, volumetric representation and point clouds representation.	4
1.2	Convolution operations on the same local region under different point orders.	7
2.1	The architecture of Multi-view CNN [41].	13
2.2	Illustration of 3D ShapeNets [48].	15
2.3	Hybrid grid-octree structure using a set of shallow octrees encoded by bit representation [36].	17
2.4	The architecture of PointNet [34].	19
2.5	The architecture of PointNet++ [35].	22
2.6	The architecture and pipeline of SO-Net [24].	24
2.7	The illustration of ELM-AE.	29
2.8	The architecture of Multi-layer ELM.	31
2.9	The architecture of Bilinear CNN [28].	32
2.10	MHBN: adaptively assigning weight [55].	37
3.1	The network architecture of PC-ELM-AE.	40
3.2	Illustration of local structure module used in PC-ELM-AE	43
3.3	Illustration of hierarchical feature extraction and incremental receptive field in PC-ELM-AE.	47
3.4	PC-ELM-AE: performance on ModelNet10 and ModelNet40 with varying parameters.	52

4.1	HOPCNN: coarse illustration of HOC module.	57
4.2	HOPCNN: the pipeline of ALC module.	63
4.3	HOPCNN: the pipeline of SLC module.	65
4.4	HOPCNN: detailed demonstration of HOC and HOMLP.	67
4.5	Illustration of HOPCNN object classification network.	68
4.6	Illustration of HOPCNN part segmentation network.	69
4.7	HOPCNN classification network: performance variation w.r.t. ranks. . . .	72
4.8	HOPCNN: visualization of part segmentation.	73
B.1	The local structure Module of PC-ELM-AE with edge operation removed.	85
B.2	PC-ELM-AE with local structure modules ablated.	85
B.3	PC-ELM-AE with the last ELM-AE layer ablated.	86
B.4	PC-ELM-AE with the second and the last ELM-AEs ablated.	86
E.1	Illustration of more part segmentation results of HOPCNN.	97

List of Tables

3.1	Statistics of ModelNet	48
3.2	PC-ELM-AE: classification performance on ModelNet datasets.	50
3.3	PC-ELM-AE: results of ablation study.	54
4.1	HOPCNN: object classification performance on ModelNet40.	70
4.2	HOPCNN: object part segmentation results on ShapeNetPart.	72
4.3	HOPCNN: ablation analysis on ModelNet40.	73

Chapter 1

Introduction

1.1 Analyzing 3D Data

Because numerous automatic machines require algorithms to navigate them through 3D sapce, *e.g.* autonomous vehicles (AVs), robots and unmanned aerial vehicles (UAVs), the decommand of high-performance algorithms processing 3D data springs up. In contrast to 2D data like images, 3D data contains richer spatial information so that machines navigated by 3D-based algorithms are supposed to understand surrounding environment better and act more reasonably in 3D sapce. Hence, 3D data understanding is a rather influential problem which includes 3D object classification [48], 3D object detection [22, 33, 40], 3D semantic segmentation [34, 35, 36], *etc.* However, 3D data is intrinsically sparse and complicated, which is quite different from 2D images that can be taken as input by a large amount of machine learning algorithms. For example, 3D models may only occupy a small portion of space, and their contours are usually depicted by triangular meshes in computer graphics (see Figure 1.1b). Although deep learning networks such as AlexNet [21], ResNet [6] and GoogLeNet [42] are pretty successful in image recognition, they are incompetent to handle mesh-based 3D data. Therefore, mesh-based 3D data is converted to other representations so as to make the generalization of state-of-the-art algorithms more readily.

Three kinds of representations are commonly utilized, *i.e.* view-based representation, volumetric representation and point clouds representation, which are illustrated in Figure 1.1. In view-based representation, 3D models are rendered to a bunch of images from different view points, each of them capturing a portion of spatial information (see Figure 1.1c). Volumetric representation follows the idea of 2D images to evenly split 3D space by small discrete cubical cells - voxels (see Figure 1.1d). The last point clouds represen-

tation represents 3D data by sets of unordered points that are sampled from surfaces of models (see Figure 1.1e). The details of these three representations and their advantages or downsides are discussed below.

1.1.1 View-based Representation

This representation presents 3D models or 3D scenes by a bunch of 2D images taken from a variety of view points, which is somewhat similar to perspective drawing. The process of projecting 3D models to 2D planes is called rendering in computer graphics. In this representation, every image contains a portion of useful information, and the integral information of original 3D data can be recovered by synthetic analysis on them. View-based representation is widely used by many early works, *e.g.* RotationNet [14] and Multi-view CNN [41]. One of appealing advantages of view-based representation is that numerous state-of-the-art CNNs can be readily generalized to it because the input data becomes a batch of 2D images after rendering. The generic pipeline is that one CNN architecture is opted to extract features from images in parallel which are merged by a pooling function to obtain global features for different models. Since state-of-the-art CNNs are applied directly, features of images are quite discriminative and robust, which leads to excellent performance on benchmarks.

However, the drawbacks of view-based representation are obvious as well. Due to the limited number of images rendered from each 3D model, some tiny but critical details would be easily discarded, making distinguishability among similar classes poor. In order to solve the above-mentioned problem, on one hand, the number of rendered images should increase to capture as many details as it can, and on the other hand, the resolution of images should improve to preserve more tiny structures. These two solutions both result in hugely incremental computational cost and storage cost. Meanwhile, selection of view points is a hand-crafted process which requires exhaustive and time-consuming tuning. Hand-crafted process may cause suboptimal networks. Furthermore, almost all algorithms using view-based representation use one consistent view points selection for models from different classes because customizing view points alignment for every class is exceedingly time-consuming. Nonetheless, it cannot be denied that there is not one view points alignment that is optimal for all classes, but view points alignment profoundly influences algorithmic performance. Lastly, images-based representation is not suitable for some tasks requiring detailed space information, *e.g.* 3D semantic segmentation, 3D object detection and 3D object reconstruction.

1.1.2 Volumetric Representation

Volumetric representation is intuitively similar to 2D images. Actually, it is a generalization of images into 3D space. Similar to 2D images that are represented by grids of 2D cells, volumetric representation splits 3D space by an abundance of cubes so that 3D data is represented by grids of voxels. One benefit of volumetric representation is that the generalization of existing CNNs to 3D space is trivial. For example, just replacing 2D convolution operators with 3D convolutions finishes the generalization procedure. So the advantage of volumetric representation is that the generalized state-of-the-art CNNs could extract expressive features which lead to decent and robust performance. The other benefit of using voxels is that detailed spatial information can be preserved if cubes are small enough. Volumetric representation is used by some previous works such as VoxNet [29] and 3D ShapeNets [48].

The disadvantage of volumetric representation is its computational cost and storage cost. Voxelization can be regarded as discretization of original continuous data. As discussed before, the smallest unit, voxel, should be tiny enough so that discretized models are clear. Otherwise, models become vague, which is akin to images in low resolution. Hence, in order to reach more outstanding performance, the resolution of voxelized 3D models is ought to increase. But computational cost and storage cost increase cubically w.r.t. 3D models resolution, which is an undesirable and unaffordable cost. Thus, in real world implementation, the resolution is usually low, leading to unsatisfactory performance. Although volumetric representation can be taken as input to generalized state-of-the-art CNNs and most spatial information is retained, computational cost and storage cost are huge obstacles preventing it from being used on larger-scale and real-world data. There are some networks exploiting tree-like structures to tackle the problem of cubically incremental computational cost, but they induce a new cost for indexing data in tree-like representation [36].

1.1.3 Point Clouds Representation

Point clouds representation is intrinsically different from the previous two representations in which 3D data is embodied by dense grids. It represents 3D data by sets of points which are sampled from surfaces of 3D models. Each point may contain spatial information like 3-dimensional coordinates and normal vector of the surface it lies on, and it can also convey other features like RGB color of the spot it is sampled from. However, point clouds are naturally sparse and unordered. In contrast to dense grids in which 3D space is filled with cells and distance between any two adjacent cells is a discrete

Figure 1.1: Illustration of mesh-based 3D model, view-based representation, volumetric representation and point clouds representation. (a) shows the original 3D model; (b) presents its meshes representation; (c) illustrates its view-based representation; (d) is its volumetric representation; (e) shows its point clouds representation.

integer, point clouds merely occupy a small portion of 3D space and distances between any two adjacent points are continuous. Point clouds representation is computationally affordable because its computational cost increases linearly w.r.t. the number of points sampled. Furthermore, it can capture 3D spatial information and is suitable for tasks like 3D semantic segmentation, 3D object detection and 3D object reconstruction. In addition, point clouds are also quite suitable for real-world applications because it is the direct output of LIDAR. Using point clouds representation is not extensively researched previously, some works like PointNet [34] and PointNet++ [35] are pioneer to explore how to make deep learning compatible to point clouds.

As for its drawback, it is clear that point clouds representation abandons lines and planes, so models may become ambiguous in areas where normal vectors dramatically change. That is one of the reasons why algorithms leveraging point clouds representation perform slightly worse than algorithms manipulating view-based and volumetric representations.

This thesis focuses on algorithmic approaches to effectively process point clouds. Next section is to discuss challenges of processing point clouds followed by explanations of two intuitive and novel ideas to resolve these challenges.

1.2 Challenges of Point Clouds Analysis

This section is to introduce technical challenges of processing point clouds. Some challenges have been handled by previous works, but many of them remain unresolved.

1.2.1 Ambiguous Definition of Neighbors

Point clouds are irregular and not evenly scattered in space. Irregularity adversely affects the analysis of point clouds, especially for definition of neighbors.

It is quite straightforward and reasonable to define neighboring cells in grids by von Neumann stencil or Moore stencil. Cells are defined as neighbors to a center cell because they are the closest ones to it with unit distance. However, things become complicated for point clouds. Points are no longer strictly and evenly aligned but are somewhat irregular and unevenly scattered. So distance between any two points is continuous rather than discrete, which means the definition of neighbors by stencils is not applicable. Querying neighboring points is indispensable because one independent point has little semantic meaning unless it is surrounded by other adjacent points as its context.

Consequently, a plenty of previous works adopt k -nearest neighbors (kNN) to build kNN graphs (kNNGs) as an auxiliary tool to define neighbors and quantize relationships [35, 24, 45, 52, 37]. Specifically, for every point of a model, its k nearest neighbors are defined as those k points with k minimum Euclidean distances to it. Hence, points become vertices, and edges only exist between neighboring points in kNNG whose topology diversifies regarding to the value of k . Afterwards, relationship between two adjacent points can be quantized as their Euclidean distance, *e.g.* small Euclidean distance means they are closely related, otherwise, they are loosely correlated. kNN and kNNG are adopted in this thesis as well.

1.2.2 Sparsity of Point Clouds

Point clouds are inborn sparse that only occupy a tiny portion of space, and the density of point clouds might vary a lot. Therefore, it is exceedingly difficult to implement convolution operations on point clouds in sliding-window fashion. For example, the shape of window must be deformable, and the stride is ought to be continuous and unfixed. However, with the help of the above-mentioned kNNG, the problem is trickily handled. Detailedly, p center points which are evenly sampled from a point cloud and their k nearest neighbors form local point regions that are processed by convolution operators in parallel. This method is used in the thesis if convolution operations are required.

1.2.3 Unorderedness of Points

It is not hard to see that point clouds are intrinsically unordered. Any permutation of points within a point cloud does not distort the model it stands for. This property profoundly influences the performance of convolution operations, as stated in Sec. 1.2.2.

The success of convolution operations on images recognition is critically attributed to the determinism of images, which means the order of pixels cannot be permuted, otherwise, they are converted to another images that may be distinctly interpreted. So convolution operations yield deterministic and consistent outputs w.r.t. the same local patch. Determinism helps improve the performance of networks. Nevertheless, points are usually shuffled thoroughly for data augmentation in point clouds. In consequence, the order of points within a local region is chaotic and varies a lot in different epochs, making convolutional outputs non-deterministic, as shown in Figure 1.2. In Figure 1.2, points are represented by square cells that are aligned into a one-dimensional array to make convolutions possible. Every point carries a feature vector $f_i \in \mathbb{R}^d$ ($i = 1, 2, 3, 4$), and every feature vector corresponds to a weight vector $w_i \in \mathbb{R}$ ($I = 1, 2, 3, 4$). The " $\odot$ " symbolizes dot product in the figure. The convolutional output of condition (i) is computed as

$$o_i = w_1 \cdot f_1 + w_2 \cdot f_2 + w_3 \cdot f_3 + w_4 \cdot f_4, \quad (1.1)$$

where $o_i \in \mathbb{R}$ denotes the convolutional output of condition (i). For condition (ii), the points are permuted, but the order of weight vectors stays the same. Hence, the convolutional output of condition (ii) is

$$o_{ii} = w_1 \cdot f_2 + w_2 \cdot f_4 + w_3 \cdot f_3 + w_4 \cdot f_1, \quad (1.2)$$

where $o_{ii} \in \mathbb{R}$ denotes the convolutional output of condition (ii). It is evident to see that o_i and o_{ii} differ a lot so that the two different outputs may lead to distinct inferences of the same local region. The chaos negatively influences the performance of networks that adopt direct convolutions on point clouds.

One possible solution to address the unorderedness of point clouds could be using symmetric functions to independently process every point. That is what PointNet [34], PointNet++ [35] and SO-Net [24] do. These networks use shared Multi-layer Perceptrons (MLPs) to every locally normalized point (locally normalized point is the point whose coordinate is subtracted by the coordinate of the center it belongs to) to extract point-wise features that are pooled to acquire features of local regions. Probing into this method, it is not difficult to find that local contextual information is obtained by two operations,

Figure 1.2: Convolution operations on the same local region under different point orders by the same kernel.

i.e. normalization and max pooling. Some local information is obtained by moving points to local coordinate systems, while most of local information is attained by pooling over output features of MLPs. However, the capabilities of normalization, MLPs and poolings are so limited that local information or local correlations among neighboring points cannot be captured thoroughly. Therefore, the performance of MLP-based methods is quite constrained.

The other solution is to directly do convolutions on point clouds such that local correlations are extracted by convolution operators. This method abandons poolings but introduces a new problem as well. As stated before, unorderedness would cause chaotic and non-deterministic convolutional outputs. In order to tackle the problem of non-determinism, [25] proposes a network involving $\mathcal{X}$ -transformation network to permute points into latent canonical order. However, permutation of n points is a problem with $\mathcal{O}(n!)$ complexity that can not be perfectly addressed by neural networks. Although $\mathcal{X}$ -transformation network is able to permute points towards latent canonical order to some extent, it still cannot ideally generate a deterministic output given a local point region with any permutation. The objective of the second algorithm introduced in this thesis is mainly to resolve the problem of unorderedness. Inspired by bilinear models for fine-grained recognition and visual question answering (VQA), the second algorithm does bilinear pooling and trilinear pooling before every convolutional layer to extract deterministic local features. The second algorithm can perfectly handle the permutation variance problem of convolutions so that it can utilize the expressive capabilities of convolution operators to extract discriminative features.

1.3 Algorithms

The thesis introduces two novel algorithms aiming at point clouds analysis. The ELM-based algorithm is the first attempt in which ELM theory is applied to point clouds clas-

sification. The algorithm gets rid of iterative updates of parameters and determines parameters via several matrix operations in forward pass instead. Compared to some deep learning methods, the algorithm is rather efficient in training and has superior performance on benchmarks. The first algorithm is introduced in Sec. 1.3.1. The objective of the second algorithm is to generalize classic convolutions to unordered point clouds. This algorithm combines *bilinear pooling* and *trilinear pooling* with convolutions to address the problem of permutation variance. Low-rankness on convolutional kernels is further imposed for efficient computation. The algorithm simultaneously benefits from high order statistics captured by bi/trilinear poolings and expressive capabilities of convolutions. This algorithm can achieve performance comparable or better than state of the art on benchmarks. The introduction of the second algorithm is presented in Sec. 1.3.2.

1.3.1 Extreme Learning Machine-based Algorithm

Among data-driven machine learning algorithms, deep learning-based algorithms are extensively utilized to solve numerous types of tasks due to their strong representative capabilities. While deep learning networks achieve relatively decent performance on many tasks and applications, their training processes involve gradient-based updating schemes like backpropagation that are time-consuming. Considering the fact that the optimization problem of deep networks is high-dimensional and non-convex, it can be trapped into local minima so easily that networks do not have good generalization capability. However, Extreme Learning Machine (ELM) [12] is proved to be a universal approximator theoretically and practically that need not utilize backpropagation methods to update weights. ELM is a Single Layer Feed-forward Neural-network (SLFN) that is highly efficient in training because the input weights and biases are randomly generated and need not be tuned during training. The output weights of ELM are computed via a matrix inverse and several matrix multiplication operations in forward pass. That means ELM is trained without backward procedure. More importantly, the generalization capability of ELM is obtained by minimizing the norm of output weights [12]. Therefore, ELM has excellent generalization performance compared to some deep networks and consumes less time in training phase.

Extreme Learning Machine Auto-Encoder (ELM-AE) proposed by [15, 16] is a variant of ELM whose output is exactly the input of the network. It adopts ELM theory in [12] so as to learn feature representations unsupervisedly that are easy to reconstruct the input. The ability to reconstruct inputs is considered as a vital property of meaningful features. ELM-AEs can be stacked to extract higher-level features which are more informative

and meaningful. Moreover, experimental results on MNIST and NORB datasets [10, 16] demonstrate that ELM-AE outperforms many deep auto-encoders in some tasks. Multi-layer ELM constructed by stacking several ELM-AEs and an ELM classifier is shown to have stronger and stabler ability than a single layer ELM classifier. Therefore, some works such as MVD-ELM [50] and CAE-ELM [46] apply multi-layer ELM to process view-based 3D models and voxelized 3D models respectively. They both reduce time complexity in training and show better performance than 3DShapeNets [48].

Although there have been several networks implementing ELM strategy on 3D shape analysis, no network considers point clouds as input. The introduced *Point Clouds-based Extreme Learning Machine Auto-Encoder* (PC-ELM-AE) is the first network using ELM to process unordered point clouds data. Specifically, it firstly adopts kNNG method to construct local correlations among neighboring points. Then, quantized local correlations are independently processed by a ELM-AE layer to extract high-level features that are pooled within local regions to generate region-wise features. After several cascaded ELM-AE layers and kNNGs, global feature of a model can be obtained by pooling over features of all points. The specific architecture of PC-ELM-AE is demonstrated and explained in Chapter 3.

The performance of PC-ELM-AE is evaluated on two benchmarks, ModelNet10 and ModelNet40 [48]. Results verify that PC-ELM-AE can achieve acceptable performance and surpasses MVD-ELM, CAE-ELM, 3DShapeNets and some previous deep learning networks by a large margin. Although the performance of PC-ELM-AE is definitely inferior to state-of-the-art deep networks, its superiority is that it can achieve acceptable performance with efficient training. The contributions of PC-ELM-AE are three folds which are presented below:

- It is the first network processing point clouds with ELM;
- It outperforms other ELM-based networks processing 3D data on benchmarks;
- It surpasses some previous deep learning methods and achieve acceptable performance with much less training time.

1.3.2 High Order Point CNN

As mentioned in Sec. 1.2.3, unorderedness is the main obstacle hampering the generalization of convolutions to point clouds. So the main concern of designing convolutional networks capable of processing point clouds is how to solve unorderedness effectively. Many works address the unorderedness by using shared multi-layer perceptrons

(MLPs) to process each point individually [34, 35, 24, 52, 37, 45]. This idea was vastly adopted and achieved impressive performance compared to hand-crafted methods. However, shared MLPs are only responsible to extract per-point feature and rarely care about correlations among adjacent points. So these MLP-based networks highly rely on symmetric pooling functions to remedy the deficiency of shared MLPs which barely capture local correlations comprehensively.

Some methods propose to do convolutions on manually designed 3D kernels such that local correlations can be encoded via convolution operators [49, 9]. Nevertheless, binning irregular points into regular cells is unnatural, and manually designing kernels is tedious and not data-driven. Hence, a more intuitive approach is to regard each point as a kernel cell and directly convolve on neighboring points without pre-defined 3D kernels (point convolution). Directly convolving on unordered point clouds may result in poor performance, as explained in Sec.1.2.3, because outputs of classic convolution operators are variable w.r.t. point permutation. That means classic CNNs may extract rather different features for the same point set in different orders, which adversely affects the performance of networks. PointCNN [25] proposes a $\mathcal{X}$ -transformation network to address the permutation problem which can transform local points to canonical order. However, the performance of $\mathcal{X}$ -transformation network is far from ideal in terms of permutation equivalence given huge amount of permutation possibilities.

The second introduced network, High Order Point CNN (HOPCNN), is a quite novel deep network that is designed to do point convolution while tackling the unorderedness of point clouds perfectly. Inspired by bilinear models for visual recognition [28, 5, 20, 32, 26] and visual question answering (VQA) [56, 18], *bilinear pooling* is implemented over local point regions before convolutions. In the network, bilinear pooling sums over outer products of inputs to generate location-ignorant features which are then convolved to inspect correlations among points. Since bilinear-pooled features are irrelevant to points order, the output of convolution is consistent even if points are permuted. Considering the second order statistics incorporated by bilinear pooling is advantageous for detecting fine-grained details [28], *trilinear pooling* is introduced which is also permutation invariant, such that the third order statistics can be leveraged. By jointly implementing bilinear pooling and trilinear pooling, high order statistics, *i.e.* the second order and the third order statistics, is captured in HOPCNN.

One terrifying problem of bi/trilinear poolings is their explosive feature dimensionality, *i.e.* the dimensionality of pooled features becomes square or cubic of that of original features. Excessively high-dimensional features are often redundant and tend to cause unaffordable computational cost. To address the problem, a *low-rank decomposition* is

exploited to convolutional kernels. Similar to [56, 26] on fully connected layers, convolutional kernels are factorized as the product of two low-rank matrices so that explicit calculation of bilinear-pooled and trilinear-pooled features can be avoided. A novel module named as High Order Convolution (HOC) is formed by combining bi/trilinear poolings with low-rank convolutions. HOC can utilize the powerful capabilities of convolutions to inspect local correlations and take advantages of high order statistics at the same time. A sequence of HOCs is finally stacked to build the HOPCNN network to enable hierarchical feature extraction. Experiments on ModelNet40 and ShapeNetPart datasets demonstrate HOPCNN can achieve comparable or superior performance than state of the art, which proves that HOC module is able to capture more meaningful and robust local information via convolutions.

To sum up, the contributions of the introduced HOPCNN are listed as below.

- First of all, convolution operators become suitable for unordered point clouds by doing bi/trilinear poolings over local point regions before convolutions. This approach simultaneously solves the permutation problem and inspects local correlations via convolution operators.
- Secondly, HOPCNN is capable of using pairwise and triplet feature interactions and take advantage of high order statistics. Experiments show that high order statistics is vital for good performance.
- Lastly, an effective decomposition is incorporated to achieve low-rankness on convolutional kernels for efficient computation.

1.4 Organization of the Thesis

The thesis is composed of 5 chapters. The first chapter introduces the background of point clouds analysis and intuitions behind the two algorithms. The following chapter reviews a bunch of related work and literature. PC-ELM-AE is presented in Chapter 3 while HOPCNN is introduced in Chapter 4. The last chapter draws conclusions and discusses future work.

Chapter 2

Literature Review

This chapter is to review literature and related work. First of all, previous works that take as input view-based representation, volumetric representation and point clouds representation are sequentially discussed. Next, preliminaries of ELM are introduced to prepare for the first algorithm PC-ELM-AE in Chapter 3. Finally, bilinear models are presented to prepare for the second algorithm HOPCNN in Chapter 4.

2.1 3D Shape Analysis with View-based Representation

View-based methods project 3D models into 2D planes from different view points so that the generated images are compatible with 2D CNNs [14, 41]. Although some 3D structural information is lost in rendering, this representation is still widely adopted because many state-of-the-art 2D CNNs can be directly applied to rendered images with little modifications. The capabilities of state-of-the-art 2D CNNs are so powerful and robust that many networks taking view-based representation as input are in cutting edge.

Multi-view CNN

Multi-view CNN [41] whose structure is presented in Figure 2.1 is a pioneer exploring how to recognize 3D shapes by 2D CNNs. It puts forward that CNNs should be shared among all images for the consideration of simplicity, efficiency and robustness. For example, it is evident that the same set of images in different orders would stand for the exactly same 3D model, so sharing CNNs among all images followed by symmetrical pooling functions (symmetry means pooling functions yield same results even if features are permuted in order) can generate global features irrelevant to image order. Otherwise,

Figure 2.1: The architecture of Multi-view CNN [41].

the order of images adversely affects the performance of networks *e.g.* networks would extract distinct features for the same set of images in different orders, though, these images actually represent the same 3D model. In other words, sharing CNNs can also be regarded as applying symmetrical functions to all images. The architecture of Multi-view CNN shows that CNN₁ is shared among all images from different views to extract view-wise features that are fed into a view-pooling function to output global features. Global features are then passed to CNN₂ to get compact shape descriptors for classification [41]. In consequence, the class label with the highest confidence is considered as the most possible class for that sample. Multi-view CNN paves the path for latter studies of view-based representation.

2.2 3D Shape Analysis with Volumetric Representation

Volumetric representation is a more intuitive and straightforward way to represent 3D data. Rather than rendering models to a bunch of 2D planes, it follows the fashion analogous to images that space is splitted into tiny cells. Discretization of 3D space is called voxelization, and voxels serve as the smallest units similar to pixels in images. Compared to view-based representation, the loss of spatial information is less, because it retains the original 3D space correlations. However, discretization of continuous data causes information loss to some extent. To preserve more local information, the resolution of voxelized data should increase. One problem is that high resolution voxelized data exploits the storage and computational complexities. This section is to review a traditional method manipulating volumetric representation followed by a novel network designed to handle the problem of cubically incremental complexities.

3D ShapeNets

Compared to a single-view image where a portion of object is blocked, there is no occlusion in voxel data so that more geometric details can be captured which play a crucial role in object recognition. In theory, networks leveraging voxel data are supposed to uncover the occluded parts and benefit from the extra information. However, voxel data (3D geometric shape) is underutilized before 3D ShapeNets [48], presumably because of the lack of a powerful and generic algorithm to process it. 3D ShapeNets is the first deep learning network that takes advantage of 3D geometric shapes in object recognition. It proposes to represent a 3D shape as a probability distribution of binary variables on a voxel grid. More precisely, each shape is voxelized into a regular grid where every voxel is a binary variable whose value is either 1, indicating the voxel is inside the mesh surface, or 0, meaning the voxel is outside the mesh surface. Hence, each shape is embodied by a 3D binary tensor.

3D ShapeNets is designed to recognize objects with 2.5D depth maps as input. Nevertheless, 2.5D depth maps also occlude some parts of objects, possibly making recognition uncertain. Thus, it incorporates mechanisms to predict next best view and do shape completion based on observed 2.5D depth maps. Next-best-view prediction and shape completion would uncover some blocked parts and reduce uncertainty of recognition. The architecture of 3D ShapeNets is shown in Figure 2.2.

Inspired by the good generative power of Deep Belief Network (DBN) [8], 3D ShapeNets proposes Convolutional Deep Belief Network (CDBN) to recover 3D geometric shape and infer category. The energy of a convolution layer in CDBN can be calculated as:

$$E(\mathbf{v}, \mathbf{h}) = - \sum_f \sum_j (h_j^f (W^f * v)_j + c^f h_j^f) - \sum_l b_l v_l, \quad (2.1)$$

where h_j^f denotes a hidden unit with index j in feature channel f , v_j and v_l denote visible unit indexed by j and l respectively, W^f and c^f are convolutional weight and bias of feature channel f , “*” represents convolution operations, and b_l is a bias associated with visible unit v_l . In the pipeline, every 3D shape is represented by a $30 \times 30 \times 30$ binary voxel grid which is manipulated by a chain of 3D convolutional layers. The ultimate output of 3D convolutions is consumed by a fully connected RBM with 1200 hidden nodes. The top layer is an associative memory DBN that consumes the output of RBM and the corresponding one-hot label. The training procedure of 3D ShapeNets follows that of DBN which involves 2 stages, *i.e.* pre-training and fine-tuning. Pre-training encompasses a layer-wise greedy training procedure in which only one layer is trained at a time. The

Figure 2.2: Illustration of 3D ShapeNets [48]. (a) is the architecture of 3D ShapeNets; (b) is the usages of 3D ShapeNets.

network is trained in a bottom-up sequence, meaning once the lower layer converges, its parameters are fixed, and its output becomes the input of the next higher layer. Standard Contrastive Divergence [7] is exploited to pre-train first four layers, and Fast Persistent Contrastive Divergence [43] is utilized to train the top layer. Pre-training is supposed to give good initialization of parameters so that the subsequent fine-tuning converges fast. As for fine-tuning procedure, 3D ShapeNets uses wake sleep algorithm [8] which is a fast learning algorithm introduced to train DBN.

Once pre-training and fine-tuning are done, 3D ShapeNets learns the joint probability distribution *i.e.* $p(\mathbf{x}, y)$, between voxelized data $\mathbf{x}$ and shape label $y \in \{1, 2, \dots, K\}$. Interestingly, the input of inference phase is 2.5D depth maps, though, the entire network is trained on unblocked voxel data. 2.5D depth maps can be converted to voxel data easily, but there are definitely a bunch of occluded voxels whose values are unknown if a single-view depth map is given. Therefore, inference of the category of an object using a single depth map is to estimate the probability of an object being category k conditioning on the observed voxels, *i.e.* $p(y = k | x_o)$ where x_o denotes the observed voxels. 3D ShapeNets utilizes Gibbs sampling to estimate $p(y | x_o)$ and unobserved voxels x_u . The details of Gibbs sampling are beyond the scope of the thesis. Please refer to [48] for more information. Finally, 3D shape is completed based on the observed 2.5D depth map by generating unobserved voxels that the network believes in. The category of an object is inferred as the most potential class to which the completed shape belongs.

[48] states a fact that a single-view 2.5D depth map may block some detailed particles that are essential to recognition. Therefore, it is necessary to focus on the next best view when recognition on a single view is uncertain. 3D ShapeNets can reduce uncertainty of

recognition by view planning. [48] gives more specifics about next-best-view prediction.

3D ShapeNets is an important network in 3D shape recognition because it is the first deep network consuming volumetric data that is able to extract generic 3D shape representation. It is a multipurpose network which can do basic categorization, challenging shape completion and view planning. It exploits the generative power of DBN and generalizes fully connected DBN to Convolutional DBN for the purpose of decreasing parameters.

OctNet

The above discussed 3D ShapeNets is referred as a dense model, *i.e.* a model taking dense grids as input. Dense models tend to be shallow, *e.g.* 3D ShapeNets consists of merely 3 convolutional layers. Deepening dense models is highly hindered by the cubically increased computational cost w.r.t. resolution. As mentioned in Sec. 1.1, 3D data is sparse in nature, which means a lot of space is unoccupied. Storing and operating on unoccupied voxels are meaningless and wasteful. Therefore, OctNet [36] proposes an approach to omit unoccupied space in data representation and focus on operations only associated with occupied voxels. OctNet is regarded as a sparse model that explicitly and efficiently exploits the sparsity of 3D data.

OctNet uses octrees [30] to partition voxel grids. The vanilla octree divides 3D space recursively until the pre-defined resolution reached. In vanilla octree structure, each input is represented by one octree where only nodes with values are further partitioned while nodes associated to free space become leaf nodes of the tree. However, doing convolution operations on a single octree is problematic due to the tree structure. In contrast to dense voxel grids where voxels are aligned according to their spatial locations, octree-based sparse voxel grids store voxels in tree structures where accessing neighboring voxels is nontrivial. Nevertheless, since convolutional operation convolves on neighboring voxels, octree-based sparse grids incorporate additional cost to access neighboring data. And the number of memory accesses for a single voxel equals to the depth of the octree. To alleviate the issue of accessing cost, OctNet proposes to use hybrid grid-octree structure, shown in Figure 2.3a.

In hybrid grid-octree structure, 3D space is partitioned by a set of shallow octrees whose depth is restricted to be small, *e.g.* three. In Figure 2.3a, the model is represented by eight shallow octrees distinguished by different colors whose maximum depth is three. The size of cells in hybrid grid-octree structure diversifies a lot. For example, one cell that contains data in the deepest level corresponds to the size of the finest voxel, but

Figure 2.3: Hybrid grid-octree structure using a set of shallow octrees encoded by bit representation [36].

one cell that does not contain data could correspond up to the size of $8^3 = 512$ voxels based on which level it stays in the octree. In contrast to spatially dense grids which are stored as 3-dimensional tensors in memory, hybrid grid-octree structure re-organizes data into tree structure by bit representation, demonstrated in Figure 2.3b and Figure 2.3c. Every shallow octree is encoded by its own 73-bit bit string where 1 indicates the cell contains data while 0 symbolizes the cell is empty. Given this tree structure, every node is associated with a bit index which embodies the position of the node in bit string, e.g. the root node is indexed by 0. Hence, it is quite convenient to compute the parent node and child nodes for each voxel with bit index k :

$$\text{parent}(k) = \lfloor \frac{k-1}{8} \rfloor, \quad (2.2)$$

$$\text{children}(k) = (8 \cdot k + 1, 8 \cdot k + 8), \quad (2.3)$$

where $(8 \cdot k + 1, 8 \cdot k + 8)$ means all indices starting from $8 \cdot k + 1$ to $8 \cdot k + 8$ inclusive. Since only leaf nodes contain useful data, OctNet put forwards to store data of all leaf nodes of a shallow octree in a contiguous array. Thus, it is necessary to determine array indices of leaf nodes given their bit indices:

$$\text{data_idx}(k) = 8 \sum_{j=0}^{\text{parent}(k)-1} \text{bit}(j) + 1 - \sum_{j=0}^{k-1} \text{bit}(j) + \text{mod}(k-1, 8), \quad (2.4)$$

where $\text{bit}(j)$ returns the bit value at index j in bit representation, and $\text{mod}(\cdot)$ is the modulo operator. Afterwards, one contiguous array is used to store all data arrays of all shallow octrees which are concatenated together in training and testing.

While hybrid grid-octree structure saves memory consumption by storing data of leaf

nodes, it re-arranges data that was previously placed in regular 3D grids into a contiguous 1D array which is intractable by conventional convolutions. Consequently, it is imperative to derive implementation of network operations on hybrid grid-octree structure. Some notations are to be used in the remaining section. $T_{i,j,k}$ denotes the value of the voxel at location (k, j, k) in 3D tensor T . $O[k, j, i]$ denotes the value of the smallest cell in hybrid grid-octree structure that comprises the voxel at location (k, j, i) .

Convolution operations on hybrid grid-octree structure can be derived as follows

$$O^{out}[k, j, i] = \text{pool_voxels}_{(\bar{k}, \bar{j}, \bar{i}) \in \Omega[k, j, i]}(T_{\bar{k}, \bar{j}, \bar{i}})$$

$$T_{k, j, i} = \sum_{a=0}^{A-1} \sum_{b=0}^{B-1} \sum_{c=0}^{C-1} W_{a,b,c} \cdot O^{in}[\hat{k}, \hat{j}, \hat{i}] . \quad (2.5)$$

Max pooling function on hybrid grid-octree structure can be formulated as

$$O^{out}[k, j, i] = \begin{cases} O^{in}[2k, 2j, 2i] & \text{if } vxd(2k, 2j, 2i) < 3 \\ P & \text{else} \end{cases} \quad (2.6)$$

$$P = \max_{a,b,c \in [0,1]} (O^{in}[2k + a, 2j + b, 2i + c]) ,$$

where $vxd(\cdot)$ returns the depth of the voxel in the shallow octree. Please refer to [36] for more detailed explanations of Equation (2.5) and Equation(2.6).

After the derivation of implementation of convolutions and max pooling on hybrid grid-octree structure, OctNet is able to restrict all operations merely on occupied regions, avoiding time cost of operating on empty voxels. Therefore, OctNet is both time-efficient and memory-friendly.

2.3 3D Shape Analysis with Point Clouds

This section focuses on previous deep networks acting on point clouds. As discussed in Sec. 1.1.3, point clouds are handy and flexible, and they truly capture detailed 3D information such that they can be used to do more complicated 3D geometric tasks, *e.g.* semantic segmentation, reconstruction, 3D object detection, *etc.* However, constructing deep neural networks acting on point clouds is nontrivial considering several intrinsic challenges of them (see Sec. 1.2). This section, first of all, reviews the pioneer deep network manipulating point clouds. Then, some subsequent models that try to tackle challenges of point clouds are presented.

Figure 2.4: The architecture of PointNet [34].

PointNet

PointNet [34] is the first deep neural network directly acting on point clouds. Before it, how to construct deep neural networks on point clouds is totally unexplored. PointNet is rather significant in the field of point clouds analysis because it demonstrates that better or comparable performance can be reached with point clouds representation. Since point clouds are the direct output of 3D sensors, *e.g.* LIDAR, it is quite promising to explore algorithms directly operating on point clouds for real-world use.

As the pioneer work, PointNet states some properties of networks that must be realized so as to process point data:

- Since point cloud is a set of unordered points, networks should be invariant to points permutations. This property corresponds to the challenge discussed in Sec. 1.2.3.
- A single point does not have semantic meaning until it is considered with its neighboring points. Hence, networks are supposed to encode local structures from sets of neighboring points and capture interactions among local structures.
- Some transformations do not change the semantic class that a point cloud stands for nor the meaning of each points. Therefore, networks need to be invariant to these transformations, *e.g.* rotation, translation and scaling.

PointNet lists three possible solutions to process unordered point sets. The first solution is to sort points into a canonical order. The second is to treat points as a sequence to train an Recurrent Neural Network (RNN), and augment data by points permutation in training. The last solution is to use a symmetric function to extract and aggregate information from every points. The first solution faces a complexity of $N!$ which is hardly fulfilled

perfectly, and RNN is unlikely to scale to input with thousands of element. Consequently, PointNet opts to the last solution to address the unorderedness of point clouds. The idea of using a symmetric function can be formalized as

$$f(\{x_1, \dots, x_n\}) \approx g(h(x_1), \dots, h(x_n)) , \quad (2.7)$$

where $f : 2^{\mathbb{R}^N} \rightarrow \mathbb{R}$, $h : \mathbb{R}^N \rightarrow \mathbb{R}^K$ and $g : \mathbb{R}^K \times \dots \times \mathbb{R}^K \rightarrow \mathbb{R}$ represents a symmetric function. PointNet proposes to approximate function h by a multi-layer perceptron (MLP) and realize function g by a max pooling function.

The architecture of PointNet (see Figure 2.4) is pretty simple and easy to understand. The input to the network is 3D coordinates of n points, represented by a matrix of size $n \times 3$ in Figure 2.4. The subsequent input transform is fulfilled by a *T-Net* that resembles the big PointNet architecture and consists of MLPs, max pooling and fully connected layers. The purpose of input transform is to enable the network to learn representations invariant to some affine transformations. *T-Net* takes the input matrix as input and learns an affine transformation matrix of size 3×3 associated with each point cloud. Then, the learnt affine transformation matrix is directly multiplied with the input matrix for every point cloud. The subsequent transformed input is fed into two MLPs with 64 hidden neurons. Note that since PointNet approximates the symmetric function h in Equation (2.7) by stacking multiple MLP layers, all MLP layers in PointNet must process each point individually. Hence, feature of each point is projected to 64 dimensions after the first two MLPs. Another feature transform network (another *T-Net*) follows the first two MLPs which learns a sample-specific transformation matrix of size 64×64 that is multiplied by the output of previous MLP. Nevertheless, feature transformation matrix has higher dimension than input transformation matrix, such high dimension that definitely imposes greater challenge on optimization. PointNet addresses this problem by adding regularization, *e.g.* it explicitly restrict the output feature transformation matrix to be orthogonal. The regularization term is formulated as:

$$L_{reg} = ||I - AA^T||_F^2 , \quad (2.8)$$

where I denotes the identity matrix, A denotes the feature transformation matrix, and $|| \cdot ||_F^2$ computes the square of Frobenius norm. Mathematically, orthogonal just rotates or flips the input, meaning information in the input will not be distorted or lost. Three consecutive MLPs with 64, 128 and 1024 hidden neurons follows the feature transform network and projects features from 64 dimensions to 1024 dimensions. After that, a max pooling function pools over all points within a point cloud to generate a global feature

which is fed into a classifier composed of three MLPs.

PointNet proposes to tackle segmentation of point clouds as a per-point classification problem. However, segmentation needs to combine both local and global features. PointNet achieves the combination of local and global knowledge by simple concatenation. For example, as shown in the segmentation network presented in Figure 2.4, the global feature is concatenated with the output of the feature transform network so that every point simultaneously has global view and local view. The concatenated features are subsequently processed by 4 consecutive MLPs to obtain the eventual output scores.

PointNet++

PointNet++ [35] is an extension of PointNet [34] that incorporates local pattern recognition. The global feature obtained by max pooling over features of all input points is quite abstract and tend to neglect detailed local structures in PointNet. Therefore, PointNet++ builds a hierarchical architecture that is able to progressively focus on larger regions. PointNet++ consists of a stack of *set abstraction levels* that have three components - *Sampling layer*, *Grouping layer*, and *PointNet layer*.

Sampling layer. Analogous to DNNs processing images that deeper the layer is, less the size of feature maps and more the number of channels are supposed to be, PointNet++ progressively lessens the number of representative points (equivalent to the size of feature maps) and increases the dimensionality of features. *Sampling layer* is to uniformly sample representative points from input points by iteratively using farthest point sampling (FPS). Suppose the given input contains n points that are denoted by the set $\{x_1, \dots, x_n\}$ and that m centroid points are sampled, denoted by the set $\{x_{i_1}, \dots, x_{i_m}\}$ where x_{i_j} is the most distant point from $\{x_{i_1}, \dots, x_{i_{j-1}}\}$ in Euclidian space. FPS uniformly samples points so that centroid points could entirely cover point clouds, which is vital for local pattern recognition.

Grouping layer. This layer searches neighboring points for every centroid and groups them together. Specifically, the input to this layer is a point set of size $N \times (d + C)$ where N is the number of points, d is the dimensionality of coordinates and C represents the dimensionality of feature associated to each point. Then, given the sampled centroids of size $N' \times d$ where N' denotes the number of centroids sampled, group layer searches K neighboring points for each centroid to form the output tensor of size $N' \times K \times (d + C)$. Each centroid corresponds to a local region that contains K member points. Two kinds of searching methods are compared in PointNet++, *i.e.* ball query and k nearest neighbors (kNN). PointNet++ experimentally proves that ball query outperforms kNN.

Figure 2.5: The architecture of PointNet++ [35].

PointNet layer. The architecture of this layer is similar to that presented in Figure 2.4, except that the input transform and feature transform networks are disabled. The output of grouping layer is fed into PointNet layer. Before processing the input by a stack of MLP layers, the coordinates of points are moved to local coordinate system. For example, if the input is represented by a tensor of size $N' \times K \times (d + C)$, the coordinate of the i -th point belonging to the j -th region is subtracted by the coordinate of the j -th centroid:

$$x_i^{(j)} = x_i^{(j)} - \hat{x}^{(j)}, \quad (2.9)$$

where $i = 1, 2, \dots, K$, $j = 1, 2, \dots, N'$, and $\hat{x}^{(j)}$ is the coordinate of the j -th center point. Although the input is still of size $N' \times K \times (d + C)$, all d -dimensional coordinates are relative coordinates to centroids. Using relative coordinates enables the PointNet layer to learn translation invariant features. PointNet layer utilizes max pooling function to pool over points within each local region, so the output is a feature matrix of size $N' \times C'$ where C' denotes the dimensionality of new features plus a coordinates matrix of size $N' \times d$ that stores coordinates of centroids.

The above-discussed *set abstraction modules* can only extract features at one scale. However, it is common that point clouds are in non-uniform density for different regions. *Set abstraction modules* can focus on fine-grained structures for dense areas, but they become less reliable for sparse areas because detailed structures are likely corrupted. Therefore, for sparse areas, abstraction modules need to inspect larger scale. PointNet++ proposes density adaptive PointNet layers to learn features from multi-scale areas and to group multi-scale features together for addressing non-uniformly sampled point clouds. Two types of density adaptive layers are considered, *i.e.* Multi-scale grouping (MSG) and Multi-resolution grouping (MRG). Please refer to [35] for more details of MSG and

MRG.

As for segmentation task, PointNet++ simulates the pipeline of transpose convolution. In encoding phase, the number of centroid points decreases gradually, and the dimensionality of features grows progressively, while in decoding phase, points are upsampled and their features gradually become low-dimensional. Once high-level features of subsampled points are extracted by *set abstraction modules*, they are backpropagated back to original points via **feature propagation level**. PointNet++ proposes a hierarchical propagation strategy based on distance interpolation across level skip links.

Feature propagation level. As shown in Figure 2.5, feature propagation level propagates features from N_l points to N_{l-1} points where N_l is the number of output points of l -th *set abstraction module* and N_{l-1} is the number of output points of $l-1$ -th *set abstraction module*. Feature propagation level, first of all, utilizes kNN to search k neighbors for N_{l-1} points among N_l points. Then, propagated features are obtained by inverse distance weighted interpolation on k neighbors. The interpolation procedure can be formulated as

$$f(x) = \frac{\sum_{i=1}^k w_i(x) f_i}{\sum_{i=1}^k w_i(x)}, \quad (2.10)$$

where $f(x)$ denotes the feature vector of the recovered point x , f_i represents the feature vector of its i -th neighbor, and $w_i(x)$ is the inverse distance weight for the i -th neighbor that is computed by Equation (2.11).

$$w_i(x) = \frac{1}{d(x, x_i)^p}, \quad (2.11)$$

where $d(x, x_i)$ computes the distance between the point x and its neighbor x_i and p denotes the power of distance which is set to be 2 in practice.

SO-Net

The main concern of SO-Net [24] is how to model the spatial distribution of point clouds. Rather than using FPS to subsample centroids from original points, it utilizes Self-Organizing Map (SOM) to generate a certain number of centroids that can depict the spatial distribution of point clouds. The process of generating SOM centroids is akin to clustering. First of all, initial SOM nodes are randomly generated which are trained with unsupervised competitive learning recursively. Specifically, given a point with coordinate (x, y, z) , the closet node to that point is activated, and the activated node and its pre-defined neighboring nodes are updated so that they move closer to that point (actu-

Figure 2.6: The architecture and pipeline of SO-Net [24].

ally vectors associated with nodes are updated). So, SOM nodes would move gradually towards point clouds until convergence. The converged nodes are supposed to be uniformly distributed on point clouds. However, the above-discussed naive SOM training is variant to permutation of points so that different centroids may be formed w.r.t. different point order. To tackle the problem, SO-Net proposes to use a permutation-invariant SOM instead. Since the result of SOM is also highly influenced by initial nodes, permutation-invariant SOM deploys a potential field to generate fixed but uniformly distributed initial nodes for every SOM update. In detail, it applies a repulsion force (f_j^{node}) to every pair of nodes so that nodes are kept away from each other, and it also applies an attraction force (f_j^{wall}) to each node to attract the node toward the origin. These two forces are weighted by λ , and the learning rate is controlled by η . Besides, instead of updating nodes once per point, permutation-invariant SOM accumulates the effects of all points and updates nodes only when the entire point cloud is seen. This SOM batch update strategy eliminates the impact of permutation of points. During SOM training, the winner node and its neighbors are updated. Details of potential field and SOM batch update rule are available in [24].

The SOM training process is off-line, which means SOM centroids should be constructed in advance before training the main architecture. Once centroids are formed, they are fixed for every point cloud. Off-line SOM training is one of the drawbacks of SO-Net because it can be regarded as a kind of time-consuming pre-processing of data. The pipeline of SO-Net is demonstrated in Figure 2.6. The input to the network is a point cloud matrix of size $N \times 3$ which denotes N points with 3D coordinates and the SOM node matrix of size $M \times 3$ where M refers to the number of nodes. SO-Net computes k nearest nodes (k is 3 in implementation) for each point rather than doing node-to-points kNN search. Then, each point is moved to k local coordinate systems by subtracting

the coordinate of the point from its k nearest nodes. This normalization operation also guarantees that the network would learn translation-invariant features. Afterwards, normalized coordinates are further processed by a stack of shared fully connected layers to extract high-level features. Max pooling function is used to pool over points associated with each node to construct node feature matrix of size $M \times 384$. Node features are further lifted to 1024 dimensions by 4 shared fully connected layers. The last max pooling is a channel-wise max pooling that pools over M nodes to generate a global feature.

As for segmentation, SO-Net adopts the same strategy as PointNet. It concatenates the global feature and intermediate local features together so that every point perceives local structures and global shape simultaneously. The concatenated feature matrix is of size $kN \times 1024$ that is further processed by three shared fully connected layers to reduce feature dimensionality. SO-Net chooses to use average pooling to pool over features associated with each point. The pooled feature matrix is of shape $N \times 256$ which is processed by two shared fully connected layers to obtain segmentation scores.

PointCNN

PointCNN [25] is a network that truly implements convolutions on unordered point clouds. The above-reviewed three networks - PointNet, PointNet++ and SO-Net - deploy the theory proposed in PointNet that in order to generate permutation invariant features, symmetric function should be used. Therefore, shared MLPs, shared fully connected layers, max pooling and average pooling are extensively utilized by them. Although shared MLPs and fully connected layers can be regarded as 1×1 convolution, they merely extract feature for each normalized point and do not inspect correlations among neighboring points because the receptive field is always 1×1 and never increases. Consequently, local correlations are considered just in the process of coordinates normalization (move absolute coordinates to relative coordinates) and pooling. Nevertheless, convolutional kernels with receptive field more than one is able to detect more sophisticated correlations among neighboring points by convolution operators, which can improve performance. Accordingly, it is not staggering that Li *et al.* [25] eager to generalize convolutions with receptive field more than one to point clouds analysis.

The key idea of PointCNN is its $\mathcal{X}$ -Conv Operator that performs $\mathcal{X}$ -transformation on every local region. $\mathcal{X}$ -Conv Operator takes as input concatenated local features and outputs order-aware $\mathcal{X}$ -transformation matrices that can permute unordered local regions back to canonical order. The detailed procedure of $\mathcal{X}$ -Conv Operator is elaborated in [25]. Suppose the centroid (the representative point) of a local region is p and its K

neighbors are represented by the set $\{p_i | i = 1, \dots, K\}$. Thus, the local region can be represented by two matrices, *i.e.* $\mathbf{P}$ of size $K \times D$ and $\mathbf{F}$ of size $K \times C_1$. $\mathbf{P}$, here, denotes D -dimensional coordinates of K neighbors and $\mathbf{F}$ stands for C_1 -dimensional features associated with K neighbors. The trainable parameter of $\mathcal{X}$ -Conv is a tensor $\mathbf{K}$ of size $K \times (C_1 + C_\delta) \times C_2$. The first step of $\mathcal{X}$ -Conv is to move neighbors to local coordinate system by subtracting p from $\mathbf{P}$, resulting in a relative coordinate matrix $\mathbf{P}'$. The next step is to lift every coordinate individually by a shared MLP, *i.e.* MLP_δ . The lifted coordinates are of C_δ dimensions, and $\mathbf{F}_\delta$ is of size $K \times C_\delta$. The third step is concatenating lifted coordinates to features, thus, this step outputs a $K \times (C_\delta + C_1)$ matrix $\mathbf{F}_*$ that is ready to do $\mathcal{X}$ -transformation. The most essential part is presumably the fourth step in which the relative coordinate matrix $\mathbf{P}'$ is reshaped to a 1-dimensional vector of size $K \cdot D$ that is further processed by a MLP to construct a $K \times K$ $\mathcal{X}$ -transformation matrix. Note that $\mathbf{P}'$ should be reshaped to an order-aware vector so that the generated $\mathcal{X}$ -transformation matrix can permute $\mathbf{F}_*$ back to consistent canonical order. The $\mathcal{X}$ -transformation is performed by multiplying $\mathcal{X}$ by $\mathbf{F}_*$, which generates the transformed feature matrix $\mathbf{F}_\mathcal{X}$ of shape $K \times (C_1 + C_\delta)$. The last step is to do normal convolution on $\mathbf{F}_\mathcal{X}$. The kernel size is $K \times 1 \times (C_\delta + C_1) \times C_2$ where C_2 is the dimensionality of the output feature, if it is regarded as a 2D convolution. In addition, PointCNN proposes to use depthwise convolutions as well as dilation of points for the purpose of reducing memory consumption and huge amount of parameters. Please refer to [25] for more information.

As discussed above, PointCNN is distinct from PointNet, PointNet++ and SO-Net in the use of convolution operators. Convolution operators in PointCNN can inspect and encode correlations among neighboring points directly, while the other three networks explore local correlations through pooling. The intention of PointCNN is great, but its $\mathcal{X}$ -Conv operator is far from ideal. $\mathcal{X}$ -Conv is designed to permute unordered points back to canonical order so that convolutions can output robust features regardless of points order. However, as mentioned in Sec. 1.2.3, permutation has $\mathcal{O}(K!)$ complexity where K is the number of neighboring points in local regions, such a huge complexity which cannot be ideally addressed by the neural network proposed in $\mathcal{X}$ -Conv. Consequently, how to generate permutation-invariant features using convolution is still under exploration.

2.4 Preliminaries of ELM

This section, first of all, is to review ELM theory, followed by introduction to ELM Auto-Encoder (ELM-AE). Then, it concentrates on how to construct multi-layer ELM by using ELM-AEs. This section prepares for the first introduced algorithm, PC-ELM-AE.

2.4.1 ELM

Extreme Learning Machine (ELM) [12] is an efficient, effective and simple Single Feed-forward Neural-network (SLFN) with the input weight and bias randomly generated and fixed. No gradient-based update scheme is included during training, which is different from deep learning networks. The training procedure is to calculate the output weight by several matrix operations that can be efficiently computed on powerful CPUs. Therefore, the training of ELM is exceedingly fast. The details of training and evaluation of ELM are stated below.

Training.

First of all, the input of ELM is propagated forward to the hidden layer by random projections. In detail, the output of i -th hidden layer node is computed by

$$h_i(\mathbf{x}) = g(\mathbf{w}_i^T \mathbf{x} + b_i), \quad i = 1, \dots, L, \quad (2.12)$$

where $\mathbf{x} \in \mathbb{R}^d$ is the input data, and $\mathbf{w}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$ are the randomly generated input weight and bias of the i -th node. $g(\cdot)$ is any nonlinear piecewise continuous function like Sigmoid, Relu and Tanh. In ELM theory [12] input weights and bias of SLFN are randomly generated which do not require tuning. Therefore, the input weight $\mathbf{w}_i$ and the bias b_i ($i = 1, \dots, L$) are kept constant even for evaluation.

Suppose the output weight matrix is $\boldsymbol{\beta} = [\beta_1, \dots, \beta_L]^T$ and the hidden layer consists of L neurons. The output function of ELM is

$$f_L(\mathbf{x}) = \sum_{i=1}^L \beta_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x}) \boldsymbol{\beta}, \quad (2.13)$$

where $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \dots, h_L(\mathbf{x})]$ is the output of the hidden layer w.r.t. sample $\mathbf{x}$. ELM randomly projects input data from d -dimensional vectors to L -dimensional vectors. To attain the optimal output weight, the training error is minimized, *i.e.* solving the following least-squares problem:

$$\min_{\boldsymbol{\beta}} \|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\|^2, \quad (2.14)$$

where $\mathbf{T}$ is the ground truth label of training data and $\mathbf{H} \in \mathbb{R}^{N \times L}$ is the output matrix of

hidden layer that can be expressed as

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & \cdots & h_L(\mathbf{x}_1) \\ \vdots & \vdots & \vdots \\ h_1(\mathbf{x}_N) & \cdots & h_L(\mathbf{x}_N) \end{bmatrix}. \quad (2.15)$$

where N denotes the number of training samples.

However, the output weight determined by Equation (2.14) is unstable because the randomly generated $\mathbf{H}$ is possibly singular. Based on ridge regression theory, adding Tikhonov regularization term can improve the stability of the solution. Hence, the training procedure is done by computing β through the following optimization problem:

$$\min_{\beta} \|\mathbf{H}\beta - \mathbf{T}\|^2 + \frac{1}{C} \|\beta\|^2. \quad (2.16)$$

where $\frac{1}{C}$ is the weight of the regularization term.

ELM solves the output weight β by simultaneously minimizing the training error *i.e.* the first term of Equation (2.16), and minimizing the norm of the weight *i.e.* the second term of Equation (2.16). ELM theory [11] states that feedforward neural networks with smaller norm of weights have better generalization performance when it minimizes the training error. Equation (2.16) is a least-squares problem with L_2 regularization (Tikhonov regularization). In theory, Equation (2.16) has the closed-form solution that is

$$\beta^* = \begin{cases} (\mathbf{H}^T \mathbf{H} + \frac{\mathbf{I}}{C})^{-1} \mathbf{H}^T \mathbf{T} & L \leq N \\ \mathbf{H}^T (\mathbf{H} \mathbf{H}^T + \frac{\mathbf{I}}{C})^{-1} \mathbf{T} & L > N \end{cases}, \quad (2.17)$$

where $\mathbf{I}$ is the identity matrix of the same dimension as $\mathbf{H}^T \mathbf{H}$ or $\mathbf{H} \mathbf{H}^T$.

Evaluation.

As for evaluation, the test data is input to the network and is propagated forward. The output of hidden layer for evaluation is also computed by Equation (2.12) with the same input weights and bias as training phase but the $\mathbf{x}$ is test data now. Then, the output of ELM is obtained by

$$\mathbf{f}_L(\mathbf{x}) = \mathbf{h}(\mathbf{x})\beta^*, \quad (2.18)$$

where β^* is the output weights calculated by Equation (2.17) in training phase.

Figure 2.7: The illustration of ELM-AE.

2.4.2 ELM-AE

One of the critically important attributes of auto-encoder is that meaningfully encoded features learned by the encoder network can be used to recover the original input with the help of the decoder network. Extreme Learning Machine Auto-Encoder (ELM-AE) is also an unsupervised auto-encoder that learns meaningful features capable of recovering the input [16]. However, ELM-AE uses ELM training strategy to determine the parameters of the auto-encoder, which is different from deep auto-encoders that update weights iteratively. According to ELM theory [12], ELM-AE whose output is exactly the input of the network is an universal approximator minimizing errors between the original input and the recovered input, which means its encoded feature representations are readily to recover input and have outstanding generalization capability. [16] shows ELM-AE learns representations using singular values and has superior performance to auto-encoder based deep networks and Deep Belief Networks. ELM-AE actually works as a building block of multi-layer ELM (ML-ELM) that stacks several ELM-AEs to improve feature learning ability. The specific training and evaluation procedures of ELM-AE are elaborated below.

Training.

Different from ELM that generates the input weight and bias under a certain probability distribution with no other limitations, ELM-AE forces the random input weights and bias to be orthogonal, that is,

$$\begin{aligned} \mathbf{h} &= g(\mathbf{a}^T \mathbf{x} + \mathbf{b}) \\ \mathbf{a}^T \mathbf{a} &= \mathbf{I}, \mathbf{b}^T \mathbf{b} = 1 \end{aligned} \quad (2.19)$$

where $\mathbf{a} = [\mathbf{a}_1, \dots, \mathbf{a}_L]$ and $\mathbf{b} = [b_1, \dots, b_L]$ are the orthogonal random weights and bias between input nodes and hidden nodes. [15] confirms orthogonality improves the performance of ELM-AE. The output matrix of hidden layer $\mathbf{H} = [\mathbf{h}_1, \dots, \mathbf{h}_N]^T$ is determined

with accordance with Equation (2.15) as well. Given the input data $\mathbf{X} = [x_1, \dots, x_N]^T$, the output weight of ELM-AE is obtained by solving the following optimization problem:

$$\min_{\beta} \|\mathbf{H}\beta - \mathbf{X}\|^2 + \frac{1}{C} \|\beta\|^2. \quad (2.20)$$

This is the revised version of Equation (2.16) with the ground-truth label $\mathbf{T}$ replaced by the input data $\mathbf{X}$. So we can obtain the closed-form solution of Equation (2.20) using Equation (2.17) by replacing ground truth $\mathbf{T}$ with $\mathbf{X}$, that is,

$$\beta^* = \begin{cases} (\mathbf{H}^T \mathbf{H} + \frac{1}{C})^{-1} \mathbf{H}^T \mathbf{X} & L \leq N \\ \mathbf{H}^T (\mathbf{H} \mathbf{H}^T + \frac{1}{C})^{-1} \mathbf{X} & L > N \end{cases}. \quad (2.21)$$

Finally, features extracted by ELM-AE are computed by

$$\mathbf{H}^* = g(\beta^{*T} \mathbf{X}), \quad (2.22)$$

where $\mathbf{H}^*$ denotes the extracted feature matrix, and β^{*T} is the transpose of the weight matrix determined by Equation (2.21).

Evaluation.

During evaluation, features regarding to test data are extracted with only one simple matrix operation and one non-linear operation. Specifically, the extracted feature matrix $\mathbf{F}$ is computed by

$$\mathbf{F} = g(\beta^{*T} \mathbf{X}), \quad (2.23)$$

where $\mathbf{X}$ denotes the test data, $g(\cdot)$ is any non-linear piecewise continuous function, and β^{*T} is the transpose of the weight matrix determined by Equation (2.21) during training.

2.4.3 Multi-layer ELM

Multi-Layer ELM (ML-ELM) proposed by [16] is composed of stacks of ELM-AE layers and an ELM classifier. It can be divided into two parts, *i.e.* stacks of ELM-AEs working as feature extractor and ELM classifier mapping encoded features to possible labels for each sample. The common architecture of ML-ELM is shown in Figure 2.8.

From Figure 2.8, it can be seen that totally k ELM-AEs are stacked to form the feature extractor module and one ELM classifier module is stacked on top of the feature extractor.

Figure 2.8: The architecture of Multi-layer ELM.

The d -dimensional input is propagated forward going through every ELM-AE layer step by step, and the output of the last ELM-AE is L^k -dimensional vectors which are fed into the ELM classifier for classification. ELM classifier firstly randomly projects the L^k -dimensional features to L -dimensional vectors which is multiplied by optimal output weight matrix β^* to compute scores.

The specifics of an ELM-AE layer are elaborated in Figure 2.7. As explained in Sec. 2.4.2, ELM-AE randomly generates orthogonal input weights w^i and bias b^i (for the i -th ELM-AE layer) so that the outputs of hidden nodes are orthogonal as well. Then, the output weights β^{i+1} are obtained based on Equation (2.21). To enable feature extraction, the input of the i -th ELM-AE h^i is multiplied by the transpose of β^{i+1} , so features extracted by that layer is L^{i+1} -dimensional.

2.5 Bilinear Models

This section reviews bilinear pooling proposed for fine-grained recognition followed by some variants of bilinear pooling and bilinear models for a variety of tasks. This section provides basic knowledge for the second introduced algorithm, High Order Point CNN.

2.5.1 Bilinear Pooling

The success of bilinear pooling is unveiled by Lin *et al.* [28] for fine-grained visual recognition. Bilinear models can capture pairwise feature interactions in a translationally invariant manner by outer product. Output of bilinear pooling is the second order statistics that is more powerful and robust than the first order statistics that is usually the output of max pooling, sum pooling or average pooling. [28] utilizes bilinear pooling for fine-grained visual recognition whose architecture is shown in Figure 2.9. Bilinear CNN

Figure 2.9: The architecture of Bilinear CNN [28].

is composed of two parallel CNN streams followed by a bilinear pooling layer and a softmax layer. CNN stream A and CNN stream B are two independent networks that capture two distinct features of the same image. CNN A and CNN B are either two networks in the same architecture or two networks in two different architectures. Interestingly, [28] experimentally shows that two networks can be shared if they are in the same architecture, which means only one CNN stream is used and the output of CNN A is duplicated as the output of CNN B to enable bilinear pooling. Note that if CNN A and CNN B are in two different architectures, they should output same size feature maps for compatibility. The detailed explanation of bilinear pooling is presented below.

Suppose the output of CNN A is a 3D tensor $\mathbf{F}_A$ of size $H \times W \times M$ and the output of CNN B is tensor $\mathbf{F}_B$ of size $H \times W \times N$, where H and W denotes the height and width of feature maps respectively. Bilinear pooling involves two steps *i.e.* local outer product and aggregation. Outer product is implemented on every location of feature maps. If $\mathbf{F}_A(h_i, w_j) \in \mathbb{R}^M$ and $\mathbf{F}_B(h_i, w_j) \in \mathbb{R}^N$ denote two features at location (h_i, w_j) , one detected by CNN A and the other captured by CNN B, the bilinear feature at this location is computed by

$$bilinear(\mathbf{F}_A, \mathbf{F}_B, h_i, w_j) = \mathbf{F}_A(h_i, w_j) \mathbf{F}_B(h_i, w_j)^T, \quad (2.24)$$

where $bilinear(\mathbf{F}_A, \mathbf{F}_B, h_i, w_j) \in \mathbb{R}^{M \times N}$ is actually the outer product of the two features. Afterwards, all bilinear features are aggregated across localtions in the second step. One option is to use sum pooling and an alternative is max pooling, both of them neglecting spatial relationship of bilinear features. Ideally, any pooling fucntion that ignores the location of features can be leveraged, but Bilinear CNN uses sum pooling.

Hence, the aggregated bilinear feature can be formulated as

$$\begin{aligned}
 \text{bilinear_feature} &= \sum_{i=1}^H \sum_{j=1}^W \text{bilinear}(\mathbf{F}_A, \mathbf{F}_B, h_i, w_j) \\
 &= \sum_{i=1}^H \sum_{j=1}^W \mathbf{F}_A(h_i, w_j) \mathbf{F}_B(h_i, w_j)^T,
 \end{aligned} \tag{2.25}$$

where $\text{bilinear_feature} \in \mathbb{R}^{M \times N}$ denotes the aggregated bilinear feature.

Bilinear feature is a generic feature descriptor that is able to capture the second order statistics. It can be reshaped as a $M \cdot N$ vector that can be fed into any classifier like SVM and fully connected classifier. Moreover, inspired by [31], Bilinear CNN incorporates two operations after aggregation. The first is signed square-root step:

$$\mathbf{y} = \text{sign}(\mathbf{x}) \sqrt{|\mathbf{x}|}, \tag{2.26}$$

where $\mathbf{x}$ denotes the reshaped bilinear feature, $\mathbf{y}$ is the feature after signed square-root step, and $\text{sign}(\cdot)$ and $\sqrt{|\cdot|}$ are both element-wise operations. The second operation is l_2 normalization:

$$\mathbf{z} = \frac{\mathbf{y}}{\|\mathbf{y}\|_2}, \tag{2.27}$$

where $\mathbf{z}$ denotes the l_2 -normalized feature which is determined by dividing every element of $\mathbf{y}$ by its l_2 norm.

2.5.2 Methods to Reduce Complexity of Bilinear Pooling

It is evident that bilinear features tend to be high-dimensional. Excessively high-dimensional features could occupy a lot of memory and induce huge amount of parameters, which makes subsequent operations on them time-consuming. Meanwhile, high-dimensional features are always redundant. Consequently, it is imperative to find ways to solve the problem of bilinear pooling.

There exists lots of works trying to address the problem. One of the most important methods is presumably [5]. Gao *et al.* [5] explore the reason why bilinear pooling outperforms other first-order pooling functions and propose a method to overcome the huge complexity of bilinear pooling based on this exploration. Bilinear features are often fed into linear kernel machines such as SVM and logistic regression. Linear kernel machines classify two feature vectors on the basis of their distance in high-dimensional space or

their inner products. Suppose CNN A and CNN B are shared, meaning the outputs of the two CNNs are exactly the same. Let $\mathcal{X}$ denote the CNN-extracted feature of one sample, so the bilinear feature can be computed by

$$\mathbf{B}(\mathcal{X}) = \sum_{s \in \mathcal{S}} \mathbf{x}_s \mathbf{x}_s^T, \quad (2.28)$$

where $\mathbf{x}_s$ is the feature vector at location s and $\mathcal{S}$ is a set representing the whole local-tions. Given the CNN-extracted feature of another sample, $\mathcal{Y}$, and its bilinear feature, $\mathbf{B}(\mathcal{Y})$, the inner product of $\mathbf{B}(\mathcal{X})$ and $\mathbf{B}(\mathcal{Y})$ is

$$\langle \mathbf{B}(\mathcal{X}), \mathbf{B}(\mathcal{Y}) \rangle = \left\langle \sum_{s \in \mathcal{S}} \mathbf{x}_s \mathbf{x}_s^T, \sum_{u \in \mathcal{U}} \mathbf{y}_u \mathbf{y}_u^T \right\rangle = \sum_{s \in \mathcal{S}} \sum_{u \in \mathcal{U}} \langle \mathbf{x}_s \mathbf{x}_s^T, \mathbf{y}_u \mathbf{y}_u^T \rangle = \sum_{s \in \mathcal{S}} \sum_{u \in \mathcal{U}} \langle \mathbf{x}_s, \mathbf{y}_u \rangle^2. \quad (2.29)$$

It is clear that bilinear feature is able to compare every local feature in $\mathcal{X}$ with that in $\mathcal{Y}$. The comparison is the second order polynomial kernel. If $\mathbf{x}_s$ and $\mathbf{y}_u$ are similar, $\langle \mathbf{x}_s, \mathbf{y}_u \rangle^2$ becomes large, so this comparison contributes to the possibility that the two samples are similar. Otherwise, $\langle \mathbf{x}_s, \mathbf{y}_u \rangle^2$ is so small that it does not increase similarity.

Based on the above analysis, [5] declares that if there exists some low-dimensional projections that project CNN features into low-dimensional space and the comparison of those projected features can still approximate the second order polynomial kernel in Equation (2.29), the complexity of bilinear pooling can be tremendously reduced. Let $\phi(x) : \mathbb{R}^N \rightarrow \mathbb{R}^d$ where $d \ll N$ denotes the possible projection function. Then, the approximation can be formulated as

$$\langle \mathbf{B}(\mathcal{X}), \mathbf{B}(\mathcal{Y}) \rangle = \sum_{s \in \mathcal{S}} \sum_{u \in \mathcal{U}} \langle \mathbf{x}_s, \mathbf{y}_u \rangle^2 \approx \sum_{s \in \mathcal{S}} \sum_{u \in \mathcal{U}} \langle \phi(\mathbf{x}_s), \phi(\mathbf{y}_u) \rangle \equiv \langle \mathbf{C}(\mathcal{X}), \mathbf{C}(\mathcal{Y}) \rangle, \quad (2.30)$$

where

$$\mathbf{C}(\mathcal{X}) := \sum_{s \in \mathcal{S}} \phi(\mathbf{x}_s). \quad (2.31)$$

Compact Bilinear Pooling utilizes Random Maclaurin (RM) and Tensor Sketch (TS) as projection functions to realize approximation. Please refer to [5] or more details of RM and TS.

Compact Bilinear Pooling avoids explicit computation of bilinear pooling, but it uses RM projection or TS projection to project CNN-extracted features to low-dimensional space and calculates approximated bilinear pooling by Equation (2.31) instead. Gao *et*

al. [5] mathematically prove that $\mathbf{C}(\mathcal{X})$ can approximate $\mathbf{B}(\mathcal{X})$ in terms of feature comparison, if $\phi_{RM}(\mathbf{x})$ or $\phi_{TS}(\mathbf{x})$ is exploited. RM and TS may lead to same performance in theory, though, TS shows superior performance to RS experimentally. Please refer to [5] for more details.

Besides, there are several methods that consider bilinear pooling layer and the subsequent classification layer (usually a fully connected layer) as a whole unit and that try to tackle the problem of bilinear pooling by imposing constraints on weights of classification layer. These methods are related to low-rank factorized bilinear pooling [18, 26, 56]. For symmetric bilinear pooling (symmetry means CNN A and CNN B in Figure 2.9 are shared), the output of bilinear pooling layer is

$$\mathbf{Z} = \sum_{s \in \mathcal{S}} \mathbf{x}_s \mathbf{x}_s^T, \quad (2.32)$$

where $\mathbf{x}_s \in \mathbb{R}^n$ denotes the feature at location s and $\mathbf{Z} \in \mathbb{R}^{n \times n}$ denotes the symmetric bilinear feature. Then, $\mathbf{Z}$ is reshaped to a vector that is denoted by $\text{vec}(\mathbf{Z}) \in \mathbb{R}^{n^2}$ which is fed into a fully connected layer with a weight matrix $\mathbf{W} \in \mathbb{R}^{c \times n^2}$ and a bias vector $\mathbf{b} \in \mathbb{R}^c$:

$$\mathbf{y} = \mathbf{W} \text{vec}(\mathbf{Z}) + \mathbf{b}, \quad (2.33)$$

where $\mathbf{y} \in \mathbb{R}^c$ denotes the output of the fully connected layer, and the fully connected layer is a linear layer without activation function, batch normalization and dropout. Apparently, the size of $\mathbf{W}$ is tremendously huge, making the fully connected layer inefficient. Low-rank factorized bilinear pooling gives an idea that the problem could be largely mitigated if $\mathbf{W}$ is imposed to be low-rank. The j -th element of $\mathbf{y}$ can be obtained by

$$y_j = \mathbf{w}_j \cdot \text{vec}(\mathbf{Z}) + b_j = \mathbf{w}_j \cdot \text{vec}\left(\sum_{s \in \mathcal{S}} \mathbf{x}_s \mathbf{x}_s^T\right) + b_j = \sum_{s \in \mathcal{S}} \mathbf{x}_s^T \mathbf{W}_j^R \mathbf{x}_s + b_j, \quad (2.34)$$

where $\mathbf{w}_j \in \mathbb{R}^{n^2}$ is the j -th row of $\mathbf{W}$, $\mathbf{W}_j^R \in \mathbb{R}^{n \times n}$ is a matrix reshaped from $\mathbf{w}_j$, and b_j is the j -th element of $\mathbf{b}$. [18], [26] and [56] propose to regularize weight matrix $\mathbf{W}$ as a low-rank matrix to mitigate the risk of overfitting. To explicitly impose the regularization, $\mathbf{W}_j^R$ is factorized as the product of two identical matrices whose rank is not larger than k :

$$\mathbf{W}_j^R = \mathbf{U}_j^T \mathbf{U}_j, \quad (2.35)$$

where $\mathbf{U}_j \in \mathbb{R}^{k \times n}$ with $k \ll n$. By substituting Equation (2.35) into Equation (2.34), y_j

can be computed as

$$y_j = \sum_{s \in \mathcal{S}} \mathbf{x}_s^T \mathbf{U}_j^T \mathbf{U}_j \mathbf{x}_s + b_j. \quad (2.36)$$

It is interesting to see that $\mathbf{U}_j$ projects $\mathbf{x}_s$ to k dimensions, which can be realized by fully connected layer. Thus, y_j can be obtained by the following procedure: 1) project all local feature vectors to k dimensions using shared fully connected layer without bias; 2) calculate the squared norm of every k -dimensional vector; 3) sum up all squared norms and add a bias. Different from Compact Bilinear CNN where projections are non-parametric and projection matrices are randomly generated, low-rank factorized bilinear pooling actually performs parametric projections with projection matrices trainable. Using the low-rank regularization, explicitly computation of the bilinear pooling is circumvented, thus making bilinear pooling followed by fully connected layers efficient.

2.5.3 Bilinear Pooling for 3D Object Recognition

Although bilinear pooling was successfully adopted in many tasks, it is seldomly used for 3D data analysis. The only attempt was done by Multi-view Harmonized Bilinear Network (MHBN) [55] for 3D object recognition. MHBN uses multi-view representation and focuses merely on classification task. It gets inspirations from analyzing similarity between features which is similar to what [5] does. Yu *et al.* started from exploring reasons why sum pooling and max pooling harm performance of networks. They utilized set-to-set similarity (SS) to measure the discriminability of pooled features. Ideally, for classification task, intra-class variance of features should be small, while inter-class variance of features should be as large as possible. These prerequisites determine whether features are discriminative or not. Discriminative features can be easily classified even if some simple classifiers are used. The discriminability of features can be measured by their pairwise similarity which is computed by inner product.

In networks using multi-view representation, suppose one object is represented by v views and each view generates convolutional features of size $h \times w \times c$ where h and w denotes the height and width of a feature map and c is the number of channels. There are totally v sets of features. Note that each c -dimensional feature corresponds to a receptive field in the original image, which is visualized in Figure 2.10 where each cell has a c -dimensional feature. Hence, convolutional features of the i -th view of object A can be represented by a set $\mathcal{F}_i(A) = \{\mathbf{f}_{i1}, \dots, \mathbf{f}_{ij}, \dots, \mathbf{f}_{in}\}$ where $n = h \times w$ and $\mathbf{f}_{ij} \in \mathbb{R}^c$ denotes the feature vector of the i -th view at location j . Features of object A are the union of all feature sets, *i.e.* $\mathcal{F}_A = \cup_{i=1}^v \mathcal{F}_i(A) = \{\mathbf{f}_{11}, \dots, \mathbf{f}_{1n}, \dots, \mathbf{f}_{v1}, \dots, \mathbf{f}_{vn}\}$. Identically,

Figure 2.10: MHBN: adaptively assigning weight. From equally assigning weights to adaptively weighting pairs, matched pairs are emphasized and unmatched pairs are suppressed [55].

features of object B can be denoted by set $\mathcal{F}_B = \cup_{i=1}^v \mathcal{F}_i(B) = \{\mathbf{g}_{11}, \dots, \mathbf{g}_{1n}, \dots, \mathbf{g}_{v1}, \dots, \mathbf{g}_{vn}\}$. Sum pooling sums over all local features of each view to generate view-wise descriptor, e.g. for object A and object B , the view-wise descriptors are calculated by

$$\mathbf{f}_i = \sum_{j=1}^n \mathbf{f}_{ij}, \quad \mathbf{g}_i = \sum_{j=1}^n \mathbf{g}_{ij}, \quad (2.37)$$

where $\mathbf{f}_i \in \mathbb{R}^c$ denotes the sum-pooled descriptor of i -th view of object A , and $\mathbf{g}_i \in \mathbb{R}^c$ denotes the sum-pooled descriptor of i -th view of object B . Afterwards, all view-wise descriptors are summed to construct object-wise features:

$$\mathbf{F} = \sum_{i=1}^v \mathbf{f}_i = \sum_{i=1}^v \sum_{j=1}^n \mathbf{f}_{ij}, \quad \mathbf{G} = \sum_{i=1}^v \mathbf{g}_i = \sum_{i=1}^v \sum_{j=1}^n \mathbf{g}_{ij}, \quad (2.38)$$

where $\mathbf{F} \in \mathbb{R}^c$ denotes the sum-pooled feature of object A , while $\mathbf{G} \in \mathbb{R}^c$ denotes the sum-pooled feature of object B .

The similarity of $\mathbf{F}$ and $\mathbf{G}$ is measured by their inner product which is termed as sum set-to-set similarity (SSS) because they are obtained by sum pooling. SSS between $\mathbf{F}$ and $\mathbf{G}$ can be computed as

$$SSS(\mathbf{F}, \mathbf{G}) = \langle \mathbf{F}, \mathbf{G} \rangle = \left\langle \sum_{i=1}^v \sum_{j=1}^n \mathbf{f}_{ij}, \sum_{i=1}^v \sum_{j=1}^n \mathbf{g}_{ij} \right\rangle = \sum_{\mathbf{f} \in \mathcal{F}_A} \sum_{\mathbf{g} \in \mathcal{F}_B} \langle \mathbf{f}, \mathbf{g} \rangle, \quad (2.39)$$

where $\langle \cdot, \cdot \rangle$ computes inner product between two vectors. It is apparent that SSS compares each local feature of object A with all local features of object B and sums all similarity together. SSS gives equal weights to good matching pairs and bad matching pairs (see the left part of Figure 2.10), so contributions of good matching pairs can be easily corrupted by those of bad matching pairs. Similarly, maximum set-to-set similarity (MSS)

in which $\mathbf{F}$ and $\mathbf{G}$ are obtained by max pooling can be easily derived as

$$MSS(\mathbf{F}, \mathbf{G}) = \max_{\mathbf{f} \in \mathcal{F}_A} \max_{\mathbf{g} \in \mathcal{F}_B} \langle \mathbf{f}, \mathbf{g} \rangle. \quad (2.40)$$

MSS compares every pair as well, but it takes the best matching pair. In other words, the MSS between A and B is entirely determined by best matching pair. However, it is common that objects having quite similar parts could come from different categories. So MSS is not reliable because the similarity between objects cannot merely be determined by one pair and other pairs would give cues for inference as well. To sum up, sum pooling and max pooling both have evident drawbacks in terms of feature similarity.

[55] proposes to use polynomial set-to-set similarity (PSS_p) as similarity measurement which is formulated by

$$PSS_p(\mathbf{F}, \mathbf{G}) = \left(\sum_{\mathbf{f} \in \mathcal{F}_A} \sum_{\mathbf{g} \in \mathcal{F}_B} \langle \mathbf{f}, \mathbf{g} \rangle^p \right)^{\frac{1}{p}}. \quad (2.41)$$

It is not hard to see that SSS and MSS are both special cases of PSS_p . For example, when $p = 1$, PSS_p becomes SSS and when $p = +\infty$, PSS_p becomes MSS . [55] proposes to utilize second order polynomial set-to-set similarity (PSS_2) where $p = 2$:

$$PSS_2(\mathbf{F}, \mathbf{G}) = \sum_{\mathbf{f} \in \mathcal{F}_A} \sum_{\mathbf{g} \in \mathcal{F}_B} \langle \mathbf{f}, \mathbf{g} \rangle^2, \quad (2.42)$$

where the square root is discarded because it does not change the order of similarity. If $\langle \mathbf{f}, \mathbf{g} \rangle$ is denoted by w , then $\langle \mathbf{f}, \mathbf{g} \rangle^2 = \langle \mathbf{f}, \mathbf{g} \rangle \cdot \langle \mathbf{f}, \mathbf{g} \rangle = w \cdot \langle \mathbf{f}, \mathbf{g} \rangle$, which indicates that PSS_2 highlights good matching pairs by giving them higher weights while bad matching pairs are suppressed via multiplying by lower weights, as illustrated in the right part of Figure 2.10. Furthermore, Equation (2.42) can be rewritten as Equation (2.43) given $\langle \mathbf{f}, \mathbf{g} \rangle^2 = \langle \text{vec}(\mathbf{f}\mathbf{f}^T), \text{vec}(\mathbf{g}\mathbf{g}^T) \rangle$:

$$PSS_2(\mathbf{F}, \mathbf{G}) = \langle \text{vec}\left(\sum_{\mathbf{f} \in \mathcal{F}_A} \mathbf{f}\mathbf{f}^T\right), \text{vec}\left(\sum_{\mathbf{g} \in \mathcal{F}_B} \mathbf{g}\mathbf{g}^T\right) \rangle, \quad (2.43)$$

where $\sum_{\mathbf{f} \in \mathcal{F}_A} \mathbf{f}\mathbf{f}^T$ and $\sum_{\mathbf{g} \in \mathcal{F}_B} \mathbf{g}\mathbf{g}^T$ are symmetric bilinear-pooled features of object A and B respectively. Eventually, it is clear that similarity among symmetric bilinear-pooled features is mathematically derived by PSS_2 which takes all possible pair comparisons into account and emphasizes good matching pairs but suppresses bad matching pairs. Equation (2.43) unveils the reason why bilinear pooling outperforms other first-order pooling functions such as sum pooling and max pooling.

Chapter 3

Point Clouds-based Extreme Learning Machine Auto-Encoder

Point Clouds-based Extreme Learning Machine Auto-Encoder (PC-ELM-AE) is the first network that implements ELM and ELM-AEs on point clouds data. Although there are some algorithms that leverage ELM theory to recognize 3D objects, *e.g.* CAE-ELM [46] and MVD-ELM [50], they take as input either volumetric representation or multi-view representation. More importantly, CAE-ELM and MVD-ELM do not explore the capability of stacked ELM-AEs in feature extraction. For instance, CAE-ELM uses one 3D convolutional layer whose kernels are randomly generated and one pooling layer to randomly project voxel data, and merely one ELM-AE is stacked on top of the pooling layer to extract features. Similarly, image data is randomly projected by a stack of random 2D convolutional layers and pooling layers in MVD-ELM, and the output of random projection is directed fed into a single hidden layer ELM for classification. However, PC-ELM-AE is tremendously different from CAE-ELM and MVD-ELM in that high-level features are extracted by stacking multiple ELM-AE layers. PC-ELM-AE forms a hierarchical architecture similar to deep learning networks such that lower layers capture more basic features and higher layers get more abstract knowledge of 3D shapes.

In this chapter, PC-ELM-AE is detailedly elaborated. Above all, the network architecture of PC-ELM-AE is presented and explained. Then, the effectiveness of it is evaluated on benchmarks - ModelNet10 and ModelNet40. Lastly, ablation study is conducted to probe into the mechanism of PC-ELM-AE.

Figure 3.1: The network architecture of PC-ELM-AE.

3.1 Network Architecture

The network architecture of PC-ELM-AE is shown in Figure 3.1. The whole network is composed of two parts - the *feature extractor* module and the *classifier* module. The *feature extractor* network uses three *local structure* modules and three ELM-AE layers to construct an order-invariant global feature for each sample. The *classifier* network consists of a single hidden-layer ELM that takes global features as input and that outputs scores for possible categories. The details of *local structure* module are presented in Figure 3.2. This section is to elaborate the mechanism of the network in detail.

3.1.1 Feature Extractor

The *feature extractor* module takes as input total N samples each of which contains n points. Considering every point carries its 3D coordinate and normal vector, the input to *feature extractor* is a tensor $\mathbf{X}_{in}^1$ of size $N \times n \times 6$, where the superscript and subscript indicate it is the input to the first ELM-AE layer. Note Figure 3.1 illustrates one sample as example. $\mathbf{X}_{in}^1$ is fed into the first *local structure* module to incorporate local information and to extract edge feature so as to obtain a local structure tensor $\mathbf{X}_{loc}^1 \in \mathbb{R}^{N \times n \times 12}$. Notice that the specifics of *local structure* module, depicted in Figure 3.2, is to be explained in Sec. 3.1.2. Then, one shared ELM-AE is stacked on top of the first *local structure* module to learn features from $\mathbf{X}_{loc}^1$ by

$$\mathbf{X}_{out}^1 = g(\mathbf{X}_{loc}^1 \beta_1^{*T}), \quad (3.1)$$

where $\beta_1^* \in \mathbb{R}^{512 \times 12}$ is the output weight of the first ELM-AE that is computed by Equation (2.21) with $\mathbf{X}$ replaced by $\mathbf{X}_{loc}^1$, $\mathbf{X}_{out}^1 \in \mathbb{R}^{N \times n \times 512}$ is the output of the first ELM-AE, and g denotes a non-linear activation function. As illustrated in Figure 3.1, features are lifted from 12 dimensions to 512 dimensions in order to increase discriminability. The

ELM-AE layer is shared among all input points such that features of points are extracted independently. Independent feature extraction is vital because it imposes a symmetric function to all points and renders features invariant to point permutation. Afterwards, the second *local structure* module takes $\mathbf{X}_{out}^1$ as input ($\mathbf{X}_{in}^2 = \mathbf{X}_{out}^1$), and the second local structure tensor $\mathbf{X}_{loc}^2 \in \mathbb{R}^{N \times n \times 1024}$ is obtained in the same fashion as $\mathbf{X}_{loc}^1$. Identical to the first layer, the subsequent ELM-AE consumes $\mathbf{X}_{loc}^2$ and extracts features by the following equation:

$$\mathbf{X}_{out}^2 = g(\mathbf{X}_{loc}^2 \beta_2^{*T}) , \quad (3.2)$$

where $\beta_2^* \in \mathbb{R}^{512 \times 1024}$ is calculated by Equation (2.21) with $\mathbf{X}$ substituted by $\mathbf{X}_{loc}^2$. The second ELM-AE projects 1024-dimensional $\mathbf{X}_{loc}^2$ back to 512 dimensions for memory efficiency. The last layer (the last *local structure* module and the last ELM-AE) just repeats the process of the first two layers except that $\mathbf{X}_{out}^2$ is taken as input. The final output tensor $\mathbf{X}_{out}^3$ of size $N \times n \times 1024$ is derived by multiplying $\mathbf{X}_{loc}^3$ with $\beta_3^* \in \mathbb{R}^{1024 \times 1024}$, as formulated in Equation (3.3).

$$\mathbf{X}_{out}^3 = g(\mathbf{X}_{loc}^3 \beta_3^{*T}) . \quad (3.3)$$

In order to generate a permutation-invariant global feature for each 3D shape, average pooling is applied to the second dimension of $\mathbf{X}_{out}^3$. Average pooling is a symmetric function that pools over all n points channel-wise. So the global feature matrix $\mathbf{G}$ is of size $N \times 1024$. The average pooling operation can be formulated as

$$\mathbf{G} = ave_pool_2(\mathbf{X}_{out}^3) , \quad (3.4)$$

where $ave_pool_2(\cdot)$ represents the average pooling operating along the second dimension of a tensor.

To sum up, *feature extractor* network consists of three building blocks each of which contains one *local structure* module and one ELM-AE. All *local structure* modules and ELM-AEs are shared among points. The extracted global feature for each 3D model is invariant to permutation and possesses high-level knowledge of 3D shape. *Feature extractor* network is hierarchical because receptive field becomes larger and encoded shape information evolves from local regions to global shapes by stacking more building blocks. Hierarchy is an important property of PC-ELM-AE that is further discussed Sec. 3.2.3.

3.1.2 Local Structure Module

The detailed mechanism of *local structure* module is shown in Figure 3.2. This module leverages k-nearest neighbors (kNN) and average pooling to combine local information. Edge features that can be interpreted as correlations between neighboring points are concatenated with original features to get final representations.

In this module, point coordinates are utilized to calculate k-nearest neighbors for all $N \cdot n$ points in all N samples. Hence, the kNN index tensor $\mathbf{K}$ is of size $N \times n \times k$ where every entry indicates the index of a neighbor. For example, as for the p -th sample, the sample-wise index of r -th closest point to the q -th point is stored in the element $\mathbf{K}_{pqr}$. Note that p ranges from 0 to $N - 1$, q ranges from 0 to $n - 1$, and r ranges from 0 to $k - 1$.

Suppose the input to the i -th *local structure* module is $\mathbf{X}_{in}^i \in \mathbb{R}^{N \times n \times m}$ where m is 6 for the first *local structure* module and 512 for the second and last ones. Features of k neighbors of each point are stacked to construct the neighborhood tensor $\mathbf{F} \in \mathbb{R}^{N \times n \times k \times m}$. Specifically, for point p_j in the l -th sample, features of its k neighbors can be fetched from the input tensor $\mathbf{X}_{in}^i$ according to their indices that are accessible from the lj -th slice of kNN index tensor, that is, $\mathbf{K}_{lj}$ (indices are stored in the last dimension of $\mathbf{K}$). Given these fetched k features which are denoted as $\mathbf{x}_{l1}, \mathbf{x}_{l2}, \dots, \mathbf{x}_{lk}$ where each feature is a m -dimensional vector, the lj -th slice of the neighborhood tensor $\mathbf{F}$ for point p_j in the l -th sample is generated by stacking these k features, that is,

$$\mathbf{f}_{lj} = \begin{bmatrix} \mathbf{x}_{l1} & \mathbf{x}_{l2} & \dots & \mathbf{x}_{lk} \end{bmatrix}^T, \quad (3.5)$$

where $\mathbf{f}_{lj} \in \mathbb{R}^{k \times m}$ denotes the lj -th slice of the neighborhood tensor. Afterwards, the neighborhood tensor is compressed by average pooling along the third dimension to obtain the neighborhood mean tensor $\mathbf{M} \in \mathbb{R}^{N \times n \times m}$, as shown in Equation (3.6).

$$\mathbf{M} = \text{ave_pool}_3(\mathbf{F}), \quad (3.6)$$

where $\text{ave_pool}_3(\cdot)$ denotes the average pooling along the third dimension. This pooling operation is equivalent to pooling along the first dimension of every slice, $\mathbf{f}_{lj}$.

Eventually, the edge feature operation is implemented. Detailedly, mean tensor $\mathbf{M}$ is subtracted from input tensor $\mathbf{X}_{in}^i$ to obtain the edge feature tensor $\mathbf{E} \in \mathbb{R}^{N \times n \times m}$, that is,

$$\mathbf{E} = \mathbf{X}_{in}^i - \mathbf{M}. \quad (3.7)$$

Figure 3.2: Illustration of local structure module used in PC-ELM-AE

Then, edge feature tensor is concatenated with the input tensor so that the ultimate local structure tensor $\mathbf{X}_{loc}^i$ is of size $N \times n \times 2m$, which can be formulated as

$$\mathbf{X}_{loc}^i = \text{concat}[\mathbf{X}_{in}^i, \mathbf{E}], \quad (3.8)$$

where $\text{concat}[\cdot]$ denotes the concatenation operation.

3.1.3 ELM Classifier

The *classifier* network consists of a single hidden-layer ELM which randomly projects global features to hidden dimensions (L dimensions) and outputs scores for possible classes according to Equation (2.17) and (2.18). Specifically, the output of the hidden layer is computed based on

$$\mathbf{H} = g(\mathbf{G}\mathbf{W} + \mathbf{b}), \quad (3.9)$$

where $\mathbf{G} \in \mathbb{R}^{N \times 1024}$ is the global feature matrix, $\mathbf{W} \in \mathbb{R}^{1024 \times L}$ and $\mathbf{b} \in \mathbb{R}^L$ are the randomly generated input weight matrix and bias vector, and $g(\cdot)$ is any piecewise non-linear activation function. Therefore, as elaborated in Sec. 2.4.1, the output weight matrix $\beta^* \in \mathbb{R}^{L \times c}$ is calculated based on Equation (2.17) where $\mathbf{T}$ is one-hot label of data. The final scores for all possible classes are determined based on Equation (2.18) where the output of hidden layer $\mathbf{H}$ is multiplied with β^* . Notice that in order to obtain probability distribution over all categories, softmax operation is further applied to the logits, and that category with the highest probability is inferred as the label of an 3D object in testing phase.

ELM classifier is proved to be a universal approximator mapping features to possible labels. In the training phase, no iterative updates are considered, and the output weights are determined based on several efficient matrix operations. The purpose of input weights

and bias is to randomly project features into another feature space, and the norm of output weights is minimized simultaneously so as to improve the generalization capability of the classifier.

3.2 Network Properties

3.2.1 Back-propagation Free

An outstanding attribute of PC-ELM-AE is that the training phase is able to get rid of back-propagation. This attribute discriminates PC-ELM-AE from prevalent deep learning methods where time-costing back-propagation is intensively applied. Deep learning methods usually use gradient-based approaches to back-propagate loss. For instance, Stochastic Gradient Descent (SGD) [3] is the most basic and simplest method, but the convergence speed of it is quite slow because it jitters along steep direction in high-dimensional space. Besides, it has other drawbacks, *e.g.* it gets stuck to local minima easily. The update rule of SGD is

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t) , \quad (3.10)$$

where $\mathbf{x}_t$ is the variable vector at time t , $\nabla f(\mathbf{x}_t)$ denotes the gradient with respect to $\mathbf{x}_t$, and η is the learning rate. In consequence, other more powerful optimization schemes are proposed and used to overcome weaknesses of SGD. Since momentum term can remember historical gradient information, it is added to SGD so as to get out of local minima:

$$\begin{aligned} \mathbf{v}_{t+1} &= \rho \mathbf{v}_t + \nabla f(\mathbf{x}_t) , \\ \mathbf{x}_{t+1} &= \mathbf{x}_t - \eta \mathbf{v}_{t+1} , \end{aligned} \quad (3.11)$$

where $\mathbf{v}_t$ is the velocity at time t that turns into the weighted sum of historical gradients, and ρ weights historical velocity and current gradient. Because $\mathbf{v}_{t+1}$ is not zero even for local minima, SGD plus Momentum can get rid of local minima readily. Other optimizers like AdaGrad, RMSProp and Adam use momentum or bias correction approaches to smooth zig-zagging paths of SGD, which accelerates the convergence progress. The above-mentioned optimizers are all first-order because they only calculate gradient. Nevertheless, there are some second-order optimization schemes like Newton Method, but deep learning-based methods seldomly use them. The first reason is that optimization problem of deep networks is extremely non-convex, which is unsuitable for Newton

Method. The second reason is that second-order optimization schemes need to calculate the inverse of a huge Hessian matrix, which is costly. So only gradient-based optimization methods are taken into account in back-propagation of deep networks.

However, PC-ELM-AE determines the weights and biases without back-propagation. The orthogonal input weights and bias of ELM-AEs are randomly generated under certain continuous probability distributions, and the output weights are calculated by Equation (2.21). The input weights and bias of ELM classifier are also randomly generated, and they need not be updated afterwards. The output weights of ELM classifier are obtained in accordance with Equation (2.17). Different from ELM-AEs, the input weights and bias of ELM classifier do not need to be orthogonal. The entire procedure of training is a single forward pass that only exploits several matrix operations which can be efficiently computed by CPUs. Consequently, PC-ELM-AE reduces training time a lot compared with deep learning methods. More importantly, PC-ELM-AE hardly gets stuck to local minima according to ELM theory that ELM classifier and ELM-AE are universal approximators [12]. Therefore, not only the training speed of PC-ELM-AE is two orders of magnitude faster than Multi-view CNN [41] and 3DShapeNets [48], but its performance also surpasses some previous deep networks by a large margin.

Remark: ELM and ELM-AE are both non-linear networks due to the incorporation of non-linear hidden layers, as reviewed in Sec. 2.4. They become linear once non-linear activations are removed, as same as fully-connected layers. That the training of ELM-related algorithms is fast is attributed to the one-pass ELM training strategy.

3.2.2 Permutation Invariance

Point cloud representation contains sets of unordered points, which differs from view-based and volumetric representations significantly where regular grids of pixels and voxels indicate spatial relationships. Given the same set of points, networks are supposed to extract an identical feature even if points are permuted. Hence, so as to process point clouds effectively, the network should generate a unique global feature for each sample regardless of $n!$ permutations (n is the number of points to represent an object). This can be achieved by applying a symmetry function to permutable inputs [34].

PC-ELM-AE leverages shared ELM-AEs to process every point independently. The weights of ELM-AEs are shared among all points so that if the input is permuted, the order of the extracted features (the output of ELM-AE) will change accordingly. For instance, suppose the input of an ELM-AE layer is $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T$ where n

denotes the number of points and $\mathbf{x}_i \in \mathbb{R}^d$. So, the output of hidden layer is

$$\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n]^T, \quad (3.12)$$

where $\mathbf{h}_i$ is a L -dimensional vector. It is apparent that the order of $\mathbf{h}_i$ is correlated to the order of $\mathbf{x}_i$. Given the output weight matrix $\beta^* \in \mathbb{R}^{L \times d}$, ELM-AE is supposed to recover the input from the output of hidden nodes, which is formulated as

$$\begin{aligned} \mathbf{H}\beta^* &= \mathbf{X} \\ [\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n]^T \beta^* &= [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T. \\ [\mathbf{h}_1\beta^*, \mathbf{h}_2\beta^*, \dots, \mathbf{h}_n\beta^*]^T &= [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T \end{aligned} \quad (3.13)$$

According to the last line of Equation (3.13), point permutation does not change β^* because only one optimal solution of β exists. This can also be verified via Equation (2.21). Considering the output of ELM-AE is computed by

$$\begin{aligned} \mathbf{H}^* &= \mathbf{X}\beta^{*T} \\ [\mathbf{h}_1^*, \mathbf{h}_2^*, \dots, \mathbf{h}_n^*]^T &= [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T \beta^{*T}, \\ [\mathbf{h}_1^*, \mathbf{h}_2^*, \dots, \mathbf{h}_n^*]^T &= [\beta^* \mathbf{x}_1^T, \beta^* \mathbf{x}_2^T, \dots, \beta^* \mathbf{x}_n^T]^T \end{aligned} \quad (3.14)$$

where $\mathbf{h}_i^*$ is a L -dimensional vector. Hence, the order of $\mathbf{h}_i^*$ in output matrix changes based on the order of $\mathbf{x}_i$ in the input matrix.

Besides, a symmetric pooling function, average pooling, is utilized to pool over n point features so that global feature is permutation-invariant. As explained in Sec. 3.1.2, *local structure* module is robust to point permutation as well because all operations like kNN, pooling and edge feature extration are symmetric.

3.2.3 Hierarchy

Hierarchical feature extraction used in CNNs like AlexNet [21] and ResNet [6] is verified to be critically useful for representation learning. In PointNet, no local information is involved, and every point is processed independently by MLP. The receptive field of PointNet is just a single point, such a small region that hinders its performance. Afterwards, PointNet++ [35] tries to overcome this shortage of PointNet with the help of set abstraction module. Set abstraction module uses farthest point sampling (FPS) to sample centroids uniformly from point clouds and groups neighboring points together. Stack-

Figure 3.3: Illustration of hierarchical feature extraction and incremental receptive field in PC-ELM-AE.

ing several abstraction modules on top of each other makes hierarchical feature learning available. Points in deeper layers is an abstraction or an accumulation of more points. The performance gain of PointNet++ is mainly attributed to incremental receptive field and hierarchical feature extraction. SO-Net [24] also applies hierarchical feature learning. It utilizes SOM instead of FPS to determine center points. PointNet++ and SO-Net confirm hierarchical feature extraction contributes to the performance of networks when dealing with unordered point clouds.

Similarly, PC-ELM-AE stacks three building blocks, that is, three ELM-AE layers and three *local structure* modules, to perform hierarchical feature learning. Receptive field of each point increases step by step such that points in deeper layer perceive larger portion of objects and encode more abstract features. The approach of PC-ELM-AE is quite similar to that of PointNet++ except that the number of points is not decreasing when the network goes deeper. The key module in PC-ELM-AE that enables hierarchical feature learning is *local structure* module. It groups neighboring points of every center with the help of kNN. The size of receptive field, controlled by k , grows gradually with a sequence of stacked *local structure* modules, as illustrated in Figure 3.3. Higher-level ELM-AE layer is supposed to extract higher-level features that are more informative, more abstract and more discriminable. PC-ELM-AE stacks only three building blocks for the consideration of efficiency.

Besides, PC-ELM-AE can be considered as a graph-updating network that dynamically updates kNN graph after each *local structure* and ELM-AE module. Vertices of

Dataset	Alignment	The Size of Training Set	The Size of Test Set
ModelNet10	Yes	3991	908
ModelNet40	No	9843	2468

Table 3.1: Statistics of ModelNet

kNN graph is invariable but its edges evolve from relative coordinates and normal vectors to higher-level features gradually. This is pretty similar to what DGCNN [45] does.

3.3 Experiments

This section is to show experimental results of PC-ELM-AE on two benchmark datasets, ModelNet10 and ModelNet40 [41] whose statistics is presented in Table 3.1. ModelNet10, a subset of ModelNet40, consists of 4899 samples categorized in 10 classes. ModelNet40 is a larger repository composed of 12311 models from 40 different categories. Different from ModelNet10, models of ModelNet40 are unaligned, that is, models are randomly rotated along z axis such that directions of models are random. Above all, PC-ELM-AE is compared with many deep learning-based methods and other ELM-based methods in this section. Then, in Sec. 3.3.3, vital hyper-parameters are varied to see their impacts to the performance of the network. In the last section, some crucial parts of the network are ablated to demonstrate their effectiveness.

3.3.1 Implementation

In the implementation, the number of hidden nodes of three stacked ELM-AEs are 512, 512, and 1024 (see Figure 3.1), which means features are projected to 1024 dimensions finally. This choice of number of hidden nodes is a good trade-off between performance and efficiency, because increasing hidden dimensions improves representativeness of features but hugely increases time cost. The trade-off weights C (see Equation (2.20)) of three ELM-AEs are chosen to be 0.001, 0.1 and 0.1 respectively.

As for the ELM classifier, parameters are tuned based on datasets, that is, the number of hidden nodes is 800 and the weight C is 10^4 for ModelNet10 whereas these two parameters are selected to be 1200 and 10^3 for ModelNet40. Moreover, k that controls the size of receptive field is set to be 128 for all *local structure* modules. It will be presented later that the number of neighbors can drastically influence accuracy. Lessening neighbors shrinks local receptive fields, meaning less information is considered for every *local structure*

module. Nevertheless, excessively large receptive fields tend to make low-level *local structure* module focus more on global information rather than local structures.

Datasets provided by PointNet++ [35] in which each object contains 10k points are adopted, and 1024 points are randomly sampled to represent every model. Coordinates and normal vectors are concatenated to form initial features of points. All 3D objects are normalized to unit sphere. Moreover, no normalization operation such as batch normalization is implemented in intermediate layers because it contributes little to performance.

Many activation functions like Sigmoid, Tanh, Relu, Leaky Relu and Elu are taken into account. Leaky Relu activation function with negative slope 0.01 is found to have the best performance among them, and it is applied to all three ELM-AEs and the hidden layer of ELM classifier. The following experiments are conducted on a server with two Intel Xeon E5-2690 v4 2.6GHz CPUs and 504 GB RAM. The code is written by PyTorch.

3.3.2 Performance on Benchmarks

Some prevailing algorithms' performances on ModelNet40 and ModelNet10 are listed in Table 3.2. SPH and LFD use hand-crafted features to do classification whereas others are data-driven with learned features. Learning-based approaches do have better performance than hand-crafted ones. It is clear that PC-ELM-AE outperforms hand-crafted methods and is superior to some deep learning-based approaches, *e.g.* 3DShapeNets, Beam Search, DeepPano, ECC and 3D-GAN. The introduced network outperforms, 3D-GAN, by 2.52% on ModelNet40 and 0.59% on ModelNet10, which is a large margin. These deep learning algorithms usually take rendered images or voxel data as input, while a more flexible and simpler point cloud representation is used in PC-ELM-AE. However, PC-ELM-AE underperforms current state-of-the-art deep networks that take point data as input. As shown in Table 3.2, PointNet surpasses PC-ELM-AE by 3.4% or so even if normal vectors are not utilized, and PointNet++ outperforms PC-ELM-AE by a large margin, reaching 91.9% on ModelNet40.

PC-ELM-AE is also compared with other ELM-based algorithms. It is definitely superior to MVD-ELM by 4.43% on ModelNet40 and 2.6% on ModelNet10. The other ELM-based method, CAE-ELM, is able to take as input two kinds of volumetric data, voxel and SDF data. (Voxel data is binary, indicating whether cells intersect with models or not, and SDF data is a kind of distance data, demonstrating how far non-empty cells are from the closet surfaces.) CAE-ELM combines voxel and SDF data together to reach the best performance which is presented in Table 3.2. PC-ELM-AE outperforms CAE-ELM (with voxel& SDF data) on both two benchmark datasets by 1.47% and 0.18% respec-

Algorithm	ModelNet40		ModelNet10	
	Accuracy(%)	Training time	Accuracy(%)	Training time
SPH [17]	68.23	-	79.97	-
LFD [4]	75.47	-	79.87	-
3DShapeNets [48]	77.32	> 2 days	83.54	2 days
Beam Search [51]	81.26	-	88.00	-
MVD-ELM [50]	81.39	674s	88.99	306.4s
DeepPano [38]	82.54	-	88.66	-
3D-GAN [47]	83.30	-	91.00	-
CAE-ELM [46]	84.35	1224s	91.41	250s
PC-ELM-AE	85.82	1650s	91.59	708s
PointNet [34] *	89.2	-	-	-
PointNet++ [35]	91.9	-	-	-

* The implementation of PointNet on ModelNet40 does not incorporate normal vectors.

Table 3.2: PC-ELM-AE: classification performance on ModelNet datasets.

tively, which confirms that feature learning capability of CAE-ELM is inferior to that of PC-ELM-AE. The accuracy gain of the introduced network is presumably attributed to more meaningful and higher-level feature representations extracted by multiple stacked ELM-AEs because CAE-ELM merely uses a single ELM-AE layer to extract features. Ablation study presented in Sec. 3.3.4 will also indicate that feature learning capability of multi-layer ELM-AEs is much better than that of a single one.

In terms of training time, PC-ELM-AE consumes much less training time compared to deep networks because ELM strategy is dramatically fast and point clouds are simpler than other representations. For example, 3DShapeNets takes more than 2 days to train on ModelNet40 while only 1650 seconds are consumed in the training for PC-ELM-AE. Besides, time complexity of PC-ELM-AE is in the same order as CAE-ELMs (1650s versus 1224s for the training on ModelNet40). That means PC-ELM-AE can achieve better accuracy with almost same time cost.

In conclusion, PC-ELM-AE outperforms some previous deep learning-based methods and all ELM-based methods by a large margin. Nonetheless, its performance is obviously inferior to state-of-the-art deep networks that also consume point clouds. Considering its excellent time efficiency during training, it is reasonable to conclude that PC-ELM-AE can achieve generally acceptable performance with much less training time.

Note: the two PC-ELM-AE accuracies reported in Table 3.2 are instance accuracies averaged over 10 trials.

3.3.3 Network Design Analysis

This section, first of all, evaluates impacts of several hyper-parameters, including the number of hidden nodes of ELM classifier, the value of trade-off weight of ELM classifier, the number of neighbors and the number of sampled points. Then, the effectiveness of some crucial modules, *e.g.* edge feature operation and *local structure* module, are validated through ablation study. Finally, the importance of stacking building blocks to enable hierarchical feature extraction is revealed as well.

Note: accuracy for every set of parameters is a mean accuracy averaged over 10 trails.

Parameter Determination

First of all, the number of hidden nodes of ELM classifier is varied to test its influence on accuracy. Figure 3.4a plots the trends of accuracy on ModelNet10 and ModelNet40. For fair comparison, the trade-off weight of ELM classifier is kept constant, that is, C is 10^4 for ModelNet10 and 10^3 for ModelNet40. Figure 3.4a reveals the best performance is obtained when the number of hidden node is 800 for ModelNet10 and 1200 for ModelNet40. Accuracy collapses dramatically for both datasets if the number of hidden nodes exceeds those two values. Meanwhile, accuracy decreases slightly if the number of hidden nodes is lessened. According to the observation of a series of experiments, it is found that the variance of 10 trails becomes large if the number of hidden nodes is more than 800 or 1200 for ModelNet10 and ModelNet40 respectively, resulting in the crash of mean instance accuracy. However, for the number of hidden nodes less than 800 or 1200, variance of the network is relatively small but mean accuracy still reduces slightly, probably because the representativeness of features decreases. So 800 and 1200 are chosen in the implementation.

Afterwards, the impact of trade-off weight C of ELM classifier on accuracy is tested. C is altered within a wide range, *i.e.* from 1 to 10^5 for ModelNet40 and from 10 to 10^6 for ModelNet10. Results in Figure 3.4b demonstrate that the network reaches the best performance if setting C to be 10^4 and 10^3 for ModelNet10 and ModelNet40 respectively. The number of hidden nodes of ELM classifier is set constant to be 800 and 1200 for these two benchmarks for fair comparison. The trend of accuracy regarding to C matches that with respect to the number of hidden nodes. In detail, accuracy crashes down if C is set to be larger than 10^4 and 10^3 and decreases slightly if C is smaller than these two values. It is also observed that enlarging C imposes greater uncertainty on the network such that variance of 10 trials becomes large. If C becomes smaller, mean accuracy decreases slightly while variance keeps unchanged. Similar to the conclusion

Figure 3.4: PC-ELM-AE: performance on ModelNet10 and ModelNet40 with varying parameters. (a) shows the impact of the number of hidden nodes of ELM classifier; (b) presents the tendency of accuracy w.r.t. varying C ; (c) reveals the impact of the number of neighboring points; (d) demonstrates the network is robust to point dropout.

made in previous experiments, the collapse of accuracy when C is large is attributed to the drastically greater randomness of the network. However, the weight of regularization term in Equation (2.16) is so large when C is small that the optimal solution of Equation (2.16) is far from the optimality of Equation (2.14). Therefore, it is not surprising that mean accuracy becomes slightly worse when C is small.

Then, the number of neighbors of each point is altered to probe into its effects. The best model uses 128 neighbors in all three *local structure* modules. So the number of neighbors is set to be 16, 32, 64 and 194 sequentially to test the performance variation of the network. From Figure 3.4c, it is clear the best performance is achieved when k is 128. The accuracy declines if less neighbors are grouped, *i.e.* 16, 32, and 64 neighbors. Surprisingly, the performance drops slightly when more neighbors are considered, *e.g.* 194 neighbors. It is probably because less neighbors like 16, 32 and 64 neighbors contain

less local information since receptive field is much small, and more neighbors like 194 neighbors indeed enlarge receptive field but the network may tend to focus more on global shape instead. So choosing 128 neighbors seems to be a good trade off between local and global structures.

The last part is to test the robustness of the network to point dropout. In this experiment, a certain percentage of points are randomly dropped from samples. The randomly dropped training samples are used to train the network again, generating a new model to be tested by randomly-dropped test samples. Figure 3.4d shows that PC-ELM-AE is extremely robust to point dropout. The accuracy decreases by only 0.85% on ModelNet40 and 0.51% on ModelNet10 when half of points are discarded. The network can still achieve good performance when the minimum number of points (128 points) are taken as input because the number of neighbors is chosen to be 128. According to the trend indicated in Figure 3.4d, adding more points indeed boosts accuracy but the gain becomes trivial if the number of points is more than 1024. Meanwhile, time cost grows linearly with respect to the number of points, *e.g.* increasing the number of points from 1024 to 2048 doubles time cost but gains little accuracy improvement. Hence, 1024 points are sampled for each object considering the trade off between performance and efficiency.

3.3.4 Ablation Study

To demonstrate the effectiveness of several crucial modules of PC-ELM-AE, ablation experiments are presented in this section. Results are shown in Table 3.3.

Note: ✓ indicates modules preserved and ✗ denotes modules ablated. All results are mean instance accuracies averaged by 10 trials.

Above all, the influence of edge feature operation is evaluated. This is done by removing edge feature module in every *local structure* module such that no subtraction operation is applied and the neighborhood mean matrix is directly concatenated with feature matrix. Please refer to Appendix B for illustration. For fair comparison, other modules are preserved. Results in the second row of Table 3.3 reveal accuracy declines by almost 3% without edge feature operations for both two datasets, which validates the efficacy of edge feature operation.

Then, the efficacy of local structure module is tested by removing three *local structure* modules, meaning edge operation is not implemented as well. The architecture of PC-ELM-AE with three *local structure* modules ablated is presented in Appendix B. After ablating *local structure* modules, no local information is incorporated and every point

Edge Feature	Local Structure	The Number of ELM-AEs	ModelNet10	ModelNet40
✓	✓	3	91.59%	85.82%
✗	✓	3	88.75%	82.70%
✗	✗	3	76.95%	62.41%
✓	✓	2	83.12%	65.27%
✓	✓	1	71.59%	58.15%

Table 3.3: PC-ELM-AE: results of ablation study.

is processed independently by ELM-AEs. Accuracy drops a lot on both two datasets compared to the accuracy in the edge feature ablation experiment (the third row versus the second row of Table 3.3). It is not surprising because local information can help discriminate two classes that look similar globally. The result is consistent with what PointNet++ [35] finds. Therefore, local structures play an important role in 3D shape analysis.

The third experiment is to test the necessity of stacking multiple ELM-AEs. At first, the last ELM-AE layer is removed, which results in a huge decline of accuracy on both datasets, especially for ModelNet40 (the fourth row of Table 3.3). Next, the second ELM-AE layer is further ablated. The last row of Table 3.3 demonstrates that accuracy collapses with the last and the second ELM-AE layers removed. Note that other modules are kept for fair comparison. Results show stacking multiple ELM-AEs definitely improve the performance of the network. In conclusion, the crash of accuracy may be credited to various factors, but the main reason is that the discriminability of low-level features is poor. Ablation of ELM-AE layers renders the deep network shallow, so the performance becomes poor. Architectures with the last and the second ELM-AE layers ablated are shown in Appendix B.

3.4 Conclusion

PC-ELM-AE is the first network processing point clouds with ELM strategy. The network outperforms some deep learning-based approaches by a large margin with much less training time on ModelNet10 and ModelNet40. PC-ELM-AE is also superior to other ELM-based methods, *e.g.* MVD-ELM and CAE-ELM, with relatively same time complexity. Besides, some important attributes such as order invariance and hierarchical feature extraction help the network handle complex point representation effectively. Order invariance enables the network to process permutable inputs, and hierarchy makes feature extraction switch from local regions to global shapes gradually. However, PC-ELM-AE

is far inferior to state-of-the-art networks processing point clouds such as PointNet and PointNet++, which is reasonable because parameters of PC-ELM-AE are not fine-tuned.

Results of ablation study show the efficacy of several crucial modules such as edge feature module, *local structure* module and multiple stacked ELM-AEs. Particularly, *local structure* module enables the network to discover local information, and multiple stacked ELM-AEs improve the discriminability of features gradually.

Additionally, this work confirms that ELM-AE can be used as feature extractor to effectively capture 3D shape information. It also shows that single hidden-layer ELM classifier works properly to project ELM-AE features onto labels. According to performance on benchmarks, PC-ELM-AE outperforms other networks utilizing ELM training scheme and some fine-tuned deep networks, though it is worse than state of the art, and time consumed by PC-ELM-AE is two orders of magnitude less than that used by deep learning-based methods. Hence, PC-ELM-AE can achieve generally acceptable performance with less training time.

Chapter 4

Permutation-Invariant High Order Point CNN

The purpose of High Order Point CNN (HOPCNN) is to generalize CNNs to unordered point clouds while solving the permutation problem that adversely affects the performance of CNNs. HOPCNN does bilinear pooling and trilinear pooling before convolutions and convolves on bilinear-pooled and trilinear-pooled feature maps to achieve permutation invariance. To reduce huge computational cost and large amount of parameters induced by bilinear and trilinear poolings, HOPCNN gets a hint from Li *et al.* [26], Yu *et al.* [56] and Kim *et al.* [18] that low-rank constraint is explicitly imposed to convolutional kernels. Low-rank constrained kernels lead to efficient *bilinear low-rank convolution* (BLC) and *trilinear low-rank convolution* (TLC) which are merged together afterwards. Further, generic building modules, *i.e.* *High Order Convolution* (HOC) module and *High Order MLP* (HOMLP) module, are established based on the fusion of BLC and TLC. HOCs combine bi/trilinear poolings with low-rank convolutions to extract permutation invariant features for unordered point clouds, as illustrated in Figure 4.1. Because bilinear pooling and trilinear pooling can incorporate the second and the third order statistics respectively, HOPCNN built on multiple HOCs and HOMLPs is capable of capturing high order statistics for feature interactions.

This chapter, first of all, presents basic considerations and motivations of HOPCNN, followed by detailed elaboration of the network. Then, the performance of HOPCNN is evaluated on ModelNet40 [48] and ShapeNetPart [53] datasets. Experiments show that the introduced network can achieve competitive or better performance than state of the art on the two datasets. Lastly, ablation study is conducted to figure out key parts that contribute to the great performance.

Figure 4.1: Coarse illustration of HOC module. HOC is able to extract permutation-invariant feature from unordered point clouds by bi/trilinear poolings and low-rank convolutions.

4.1 The Problem of Direct Convolution

In this section, the problem of direct convolution on point clouds is revealed. The uncovered problem stimulates the modification of classical CNNs such that they are adaptive to unordered data.

A center point p and its $k - 1$ neighbors form a local point region that can be identified by a coordinate matrix $C \in \mathbb{R}^{k \times 3}$ and a feature matrix $F \in \mathbb{R}^{k \times d}$. Each point contains its 3D coordinate plus a d -dimensional feature. In order to make subsequent feature extraction translation-invariant, coordinates are normalized by their mean, *i.e.* $\bar{C} = C - \text{mean}(C)$, thus, relative coordinate system is constructed. Then, the local relative coordinates in $\bar{C}$ are individually lifted by MLP_f to generate $F'_m \in \mathbb{R}^{k \times d_c}$ which is concatenated with F to form the feature map $F_m \in \mathbb{R}^{k \times d_f}$ ($d_f = d + d_c$) for direct convolution. Directly convolving on F_m with a kernel $W_f \in \mathbb{R}^{k \times d_f}$ and a bias $b \in \mathbb{R}$ results in

$$\begin{aligned} z_f &= \text{Conv}(W_f, b, F_m) \\ &= \text{vec}(W_f) \cdot \text{vec}(F_m) + b, \end{aligned} \tag{4.1}$$

where $z_f \in \mathbb{R}$ denotes the output of direct convolution, $\text{vec}(\cdot)$ denotes vectorization of a matrix, and “ $\cdot$ ” represents inner product.

Due to unorderedness of points, rows of C and F are permutable, which means if C and F are multiplied by any permutation matrix $P \in \mathbb{R}^{k \times k}$, they still represent the same local point region. Given that $\bar{C}$ and F become $P\bar{C}$ and PF under permutation, F'_m and F_m are supposed to be PF'_m and PF_m respectively. Therefore, the output of direct

convolution on F_m under permutation with the same kernel and bias is

$$z_f = \text{vec}(W_f) \cdot \text{vec}(PF_m) + b. \quad (4.2)$$

Equation (4.2) demonstrates that the output of direct convolution would vary w.r.t. the permutation matrix. That means convolution operator would extract tremendously distinct features for the same local region in different orders. Networks using direct convolution will be awkward if one local region can be encoded by thousands of different descriptors, which is undesirable.

4.2 Bilinear Pooling Bonded to Convolution

As uncovered in Equation (4.2), the core of solving permutation variance problem is to make the input to convolution invariable to points' order. PointCNN [25] proposes to multiply PF_m by a learned transformation matrix $\mathcal{X} \in \mathbb{R}^{k \times k}$, i.e. $\mathcal{X} \times PF_m$. Since $\mathcal{X}$ is designed to be aware of current points' order, it can permute local region back to canonical order theoretically. However, hoping $\mathcal{X}$ to perfectly offset the effect of P or ideally permute local region under any permutation back to canonical order is unrealistic because on one hand, $\mathcal{X}$ cannot just be the inverse of P , and on the other hand, the number of permutations can be so huge that $\mathcal{X}$ -transform network can hardly handle all possibilities.

In view of $P^T P = I$, it is desired to multiply PF_m by P^T , after which the feature map is irrelevant to any permutation. Nevertheless, the specific form of P is unknown, so this method is still infeasible.

One possible approach is to multiply the feature map by its transpose, which is clarified below

$$(PF_m)^T PF_m = F_m^T P^T PF_m = F_m^T F_m = \sum_{i=1}^k f_{mi} f_{mi}^T, \quad (4.3)$$

where $F_m = [f_{m1}, \dots, f_{mk}]^T$ and $f_{mi} \in \mathbb{R}^{d_f}$. This formula shows that the permutation matrix can be ideally eliminated even without knowing its specific form, thus, a new permutation-invariant feature map is generated. It is apparent that the new feature map is symmetric and actually the output of bilinear pooling over F_m . Hence, it is termed as symmetric bilinear-pooled feature map and symbolized as B_m :

$$B_m = \sum_{i=1}^k f_{mi} f_{mi}^T, \quad (4.4)$$

where $B_m \in \mathbb{R}^{d_f \times d_f}$. Afterwards, convolution is done on permutation-invariant B_m rather than PF_m with a kernel $W_b \in \mathbb{R}^{d_f \times d_f}$ and bias $b \in \mathbb{R}$:

$$\begin{aligned} z_b &= \text{Conv}(W_b, b, B_m) \\ &= \text{vec}(W_b) \cdot \text{vec}(B_m) + b, \end{aligned} \quad (4.5)$$

where $z_b \in \mathbb{R}$ denotes the output of convolution that takes symmetric bilinear-pooled feature map as input. Since B_m is invariable to points' order, z_b turns out to be order-invariant as well. This convolution is named as *symmetric bilinear convolution* (SBC).

4.3 Constraining the Rank of Convolutional Kernels

In previous section, symmetric bilinear-pooled feature maps are fed into convolution operators such that the output is permutation-invariant. However, the size of B_m in Equation (4.4) can be huge, *e.g.* its size is the square of original feature dimensionality. That tremendously large size harms the efficiency of SBC. Consequently, the efficiency problem is resolved by constraining the rank of convolutional kernels, *i.e.* the rank of W_b .

Low-rank constraint makes W_b factorizable. Specifically, W_b can be factorized into the product of two low-rank matrices:

$$W_b = U^T U, \quad (4.6)$$

where W_b becomes the symmetric bilinear low-rank kernel and $U \in \mathbb{R}^{r \times d_f}$ with $r \ll d_f$.

Remark: since B_m is a symmetric matrix, it is reasonable to assume the kernel acting on it to be symmetric as well. So the kernel is decomposed as the product of a low-rank matrix and its transpose. With this factorization, the rank of W_b is restricted smaller than or equal to r . If the condition that $r \ll d_f$ is attained, the number of trainable parameters decreases greatly from $\mathcal{O}(d_f^2)$ to $\mathcal{O}(d_f)$ per kernel. Networks with less parameters are more efficient and less possible to overfit. After that, a new formula to compute the output of SBC can be obtained by substituting Equation (4.4) and Equation (4.6) into Equation (4.5):

$$\begin{aligned} z_b &= \text{vec}(W_b) \cdot \text{vec}\left(\sum_{i=1}^k f_{mi} f_{mi}^T\right) + b = \sum_{i=1}^k f_{mi}^T W_b f_{mi} + b \\ &= \sum_{i=1}^k f_{mi}^T U^T U f_{mi} + b. \end{aligned} \quad (4.7)$$

The last line of Equation (4.7) indicates z_b is permutation-invariant and can be computed via several matrix multiplications. In detail, each point feature f_{mi} is first multiplied by U to projected to r dimensions. Then, the squared L_2 norm of all projected features are summed. Finally, the summation plus the bias determines the final output. This new convolution is termed as *symmetric bilinear low-rank convolution* (SBLC).

Bilinear pooling can be asymmetric if two different feature maps are pooled over. Given the second feature map, $H_m \in \mathbb{R}^{k \times d_h}$ or PH_m under permutation, PH_m is multiplied by the transpose of PF_m to eliminate permutation matrix, which is analogous to the symmetric case. Thus, the asymmetric bilinear-pooled feature map can be derived by

$$B'_m = \sum_{i=1}^k f_{mi} h_{mi}^T, \quad (4.8)$$

where $H_m = [h_{m1}, \dots, h_{mk}]^T$, $h_{mi} \in \mathbb{R}^{d_h}$ and $B'_m \in \mathbb{R}^{d_f \times d_h}$ denotes the asymmetric bilinear-pooled feature map. Asymmetric bilinear pooling leads to *asymmetric bilinear low-rank convolution* (ABLC). In ABLC, asymmetric bilinear-pooled feature map is convolved by asymmetric bilinear low-rank kernel that can be factorized using low-rank constraint as

$$W'_b = U^T V, \quad (4.9)$$

where $W'_b \in \mathbb{R}^{d_f \times d_h}$ denotes the asymmetric bilinear low-rank kernel and $V \in \mathbb{R}^{r \times d_h}$. Hence, the output of ABLC is readily obtained in the same fashion as Equation (4.7), which is presented below.

$$\begin{aligned} z'_b &= \text{vec}(W'_b) \cdot \text{vec}(B'_m) + b = \text{vec}(W'_b) \cdot \text{vec}\left(\sum_{i=1}^k f_{mi} h_{mi}^T\right) + b \\ &= \sum_{i=1}^k f_{mi}^T W'_b h_{mi} + b = \sum_{i=1}^k f_{mi}^T U^T V h_{mi} + b. \end{aligned} \quad (4.10)$$

Equation (4.10) indicates a quite similar realization process to Equation (4.7) except that two sets of features are projected to r dimensions by two different low-rank matrices (U and V).

4.4 Generalizing to Trilinear Pooling

Since the second order statistics and pairwise feature interactions incorporated by bilinear pooling is advantageous for detecting fine-grained structures [28], it is intuitive to

generalize bilinear pooling to trilinear pooling such that higher order statistics, *e.g.* the third order statistics, can be captured. The third order statistics involves triplet feature interactions that might be helpful for improving feature discriminability. This section is to discuss whether the above-mentioned low-rank convolutions are applicable to trilinear pooling.

Hinted by Equation (4.4) where symmetric bilinear-pooled feature maps are calculated by summing the out products of features, it is straightforward to realize symmetric trilinear pooling over F_m by two cascaded out products, which is shown below,

$$T_m = \sum_{i=1}^k f_{mi} \circ f_{mi} \circ f_{mi} , \quad (4.11)$$

where “ $\circ$ ” denotes outer product and $T_m \in \mathbb{R}^{d_f \times d_f \times d_f}$ is 3-dimensional tensor representing the permutation-invariant symmetric trilinear-pooled feature map. Then, T_m is convolved by a 3-dimensional kernel $W_t \in \mathbb{R}^{d_f \times d_f \times d_f}$ whose rank is restricted. Given that $W_b = U^T U = \sum_{n=1}^r u_n u_n^T$ where $u_n \in \mathbb{R}^{d_f}$ and $U = [u_1, \dots, u_r]^T$, 3-dimensional W_t can be similarly decomposed by using two sequential outer products to achieve lowrankness:

$$W_t = \sum_{n=1}^r u_n \circ u_n \circ u_n , \quad (4.12)$$

where W_t denotes the symmetric trilinear low-rank kernel. Convolution on T_m with the low-rank kernel W_t leads to *symmetric trilinear low-rank convolution* (STLC) whose output is

$$\begin{aligned} z_t &= \text{vec}(W_t) \cdot \text{vec}(T_m) + b \\ &= \sum_{i=1}^k \mathbf{1}^T (U f_{mi} \odot U f_{mi} \odot U f_{mi}) + b , \end{aligned} \quad (4.13)$$

where $z_t \in \mathbb{R}$ is permutation-invariant, $U = [u_1, \dots, u_r]^T$, $\mathbf{1} \in \mathbb{R}^r$ is an all-ones vector, and “ $\odot$ ” denotes element-wise product. The detailed derivation of Equation (4.13) is shown in Appendix C. Equation (4.13) is the generalization or extension of Equation (4.7), so STLC can be readily realized in the same fashion as SBLC.

STLC is further extended to *asymmetric trilinear low-rank convolution* (ATLC) by using asymmetric trilinear pooling, analogous to how SBLC is generalized to ABLC. The third feature map $E_m \in \mathbb{R}^{k \times d_e}$ is incorporated to do trilinear pooling so that the

asymmetric tilinear-pooled feature map is

$$T'_m = \sum_{i=1}^k f_{mi} \circ h_{mi} \circ e_{mi}, \quad (4.14)$$

where $E_m = [e_{m1}, \dots, e_{mk}]^T$, $e_{mi} \in \mathbb{R}^{d_e}$, and $T'_m \in \mathbb{R}^{d_f \times d_h \times d_e}$. Now that the feature map is asymmetric, the convolutional kernel acting on the feature map should be an asymmetric tensor as well. Thus, the asymmetric trilinear low-rank kernel can be factorized similarly to W_t as

$$W'_t = \sum_{n=1}^r u_n \circ v_n \circ s_n, \quad (4.15)$$

where $v_n \in \mathbb{R}^{d_h}$, $s_n \in \mathbb{R}^{d_e}$ and $W'_t \in \mathbb{R}^{d_f \times d_h \times d_e}$ denotes the asymmetric trilinear low-rank kernel. The output of ATLC is formulated by convolving on T'_m with W'_t , which is

$$\begin{aligned} z'_t &= \text{vec}(W'_t) \cdot \text{vec}(T'_m) + b \\ &= \sum_{i=1}^k \mathbf{1}^T (U f_{mi} \odot V h_{mi} \odot S e_{mi}), \end{aligned} \quad (4.16)$$

where $V = [v_1, \dots, v_r]^T$, $S = [s_1, \dots, s_r]^T$ and $z'_t \in \mathbb{R}$ denotes the permutation-invariant output of ATLC. The derivation of Equation (4.16) is presented in Appendix C as well. The realization of ATLC is akin to that of ABLC except that the third set of features are projected to r dimensions by S .

4.5 Merging BLC and TLC

Bilinear low-rank convolution (BLC) in Sec. 4.3 and *tilinear low-rank convolution* (TLC) in Sec. 4.4 are eventually merged to fuse the second and the third order statistics. The merge is realized via simply adding their outputs, e.g. $z = z_b + z_t$ to determine *symmetric low-rank convolution* (SLC) and $z' = z'_b + z'_t$ to define *asymmetric low-rank convolution* (ALC). The detailed derivations of z and z' are shown in Appendix C. The

Figure 4.2: The figure shows the pipeline of ALC module. “ $\otimes$ ” and “ $\oplus$ ” denote element-wise multiplication and element-wise add respectively. d_o represents the dimensionality of output features. Dropout is introduced to prevent from overfitting, and activation and batch normalization are exploited as common CNNs do.

final formulas are presented below,

$$\begin{aligned}
 z &= z_b + z_t \\
 &= \sum_{i=1}^k \mathbf{1}^T [U f_{mi} \odot U f_{mi} \odot (\mathbf{1} + U f_{mi})] + b, \\
 z' &= z'_b + z'_t \\
 &= \sum_{i=1}^k \mathbf{1}^T [U f_{mi} \odot V h_{mi} \odot (\mathbf{1} + S e_{mi})] + b,
 \end{aligned} \tag{4.17}$$

where $z \in \mathbb{R}$ is the output of SLC, and $z' \in \mathbb{R}$ is the output of ALC. z and z' both can capture the second and the third order statistics simultaneously.

4.6 The Architectures

Above all, ALC and SLC modules are constructed based on Equation (4.17). Then, two building blocks, *i.e.* HOC and HOMLP, are formed by leveraging ALC and SLC respectively. Lastly, HOC and HOMLP are stacked to structure hierarchical networks for 3D object classification and part segmentation.

4.6.1 ALC Module

Given n centroids, each with $k - 1$ neighbors, the three feature maps are F_m , H_m and E_m whose shapes are specified in Figure 4.2. Projections are realized by three different 1×1 convolutions without bias. Considering the final output is d_o -dimensional, d_o sets of projection matrices, *i.e.* $\{U_j, V_h, S_j\} j = 1, \dots, d_o$, are needed. So the weight of $Conv_f$ is of size $rd_o \times 1 \times 1 \times d_f$ that projects each feature in F_m to rd_o dimensions. $Conv_h$ and $Conv_e$ work similarly to $Conv_f$ such that the three projected feature maps are in same size. Then, the element-wise multiplication of F_r and H_r is added to that of F_r , H_r and E_r to form the intermediate feature map A which is summed along the third and the second dimensions sequentially after the dropout layer to generate Z' . Finally, a d_o -dimensional bias is added to each row of Z' , and the biased Z' is fed into activation and batch normalization layers to obtain the output.

4.6.2 SLC Module

The pipeline of SLC is rather similar to that of ALC except that only one feature map is used and each point is assumed to be an individual local region ($k = 1$) so that F_m is in shape $n \times d_f$, as illustrated in Figure 4.3. The intermediate feature map A is attained by adding the square of the projected feature map F_r to the cube of it in this module. A is summed along the second dimension after the dropout layer to obtain the unbiased Z . Activation and batch normalization layers are leveraged to process the biased Z , as common convolutional layers do.

4.6.3 High Order Convolution

HOC is established based on ALC module. It works as the building block of the final network whose pipeline is demonstrated in the left part of Figure 4.4. It comprises three components which are Feature Map module to extract three different feature maps, ALC module to do asymmetric low-rank convolutions and MLP_g to lift all centroids.

Feature Map module whose procedure is formalized in Algorithm 1 takes as input points' coordinates and points' features and outputs three feature maps plus sampled centroids. It uses farthest point sampling (FPS) to uniformly sample n centroids from M points and searches $k - 1$ adjacent points for each centroid by k nearest neighbors (kNN) in Euclidean space. Thus, the coordinate tensor $\mathcal{C}$ is in shape $n \times k \times 3$ and the feature tensor $\mathcal{F}$ is of size $n \times k \times d$. $\mathcal{C}$ is then normalized by its mean to construct relative

Figure 4.3: The figure shows the pipeline of SLC module. “ $\oplus$ ” represents element-wise add. d_o is the dimensionality of output features.

coordinates which are fed into MLP_f , MLP_h and MLP_e sequentially to individually extract per-point feature. At last, the concatenation of F'_m and $\mathcal{F}$ is processed by MLP_m for feature fusion.

Using relative coordinates means features extracted by ALC become translation invariant. However, the position of a local region relative to others is crucial for inference, *e.g.* aircrafts wings should be symmetric regarding to its body. Hence, MLP_g is utilized to lift coordinates of centroids to $\lfloor \frac{d_o}{4} \rfloor$ dimensions, and the lifted coordinates are concatenated to the output of ALC such that feature of each local region is aware of relative position. The output of HOC is two matrices which are symbolized as “new PF” and “new PC” in Figure 4.4.

Remark: although shared MLPs are still utilized to extract per-point feature, local correlations are inspected by convolutions rather than MLPs and poolings in the introduced method, which is tremendously different from MLP-based networks such as PointNet, PointNet++ and SO-Net.

4.6.4 High Order MLP

HOMLP is built on the basis of SLC module. In HOMLP, each input point is regarded as an individual local region ($k = 1$), so FPS and kNN are needless here. Thus, input

Algorithm 1 Feature Map Module**Input:** $PC \in \mathbb{R}^{M \times 3}$, $PF \in \mathbb{R}^{M \times d}$, k **Output:** $F_m, H_m, E_m, \mathcal{P}$

- 1: $\mathcal{P} = \text{FPS}(PC)$ $\triangleright \mathcal{P} \in \mathbb{R}^{n \times 3}$
- 2: $\mathcal{C}, \mathcal{F} = \text{kNN}(PC, PF, \mathcal{P}, k)$
- 3: $\bar{\mathcal{C}} = \mathcal{C} - \text{mean}(\mathcal{C})$ $\triangleright$ Average along 2^{nd} dim of $\mathcal{C}$
- 4: $F'_m = \text{MLP}_f(\bar{\mathcal{C}})$ $\triangleright F'_m \in \mathbb{R}^{n \times k \times d_c}$
- 5: $H_m = \text{MLP}_h(\bar{\mathcal{C}})$ $\triangleright H_m \in \mathbb{R}^{n \times k \times d_h}$
- 6: $E_m = \text{MLP}_e(\bar{\mathcal{C}})$ $\triangleright E_m \in \mathbb{R}^{n \times k \times d_e}$
- 7: $F_m = \text{MLP}_m([F'_m, \mathcal{F}])$ $\triangleright F_m \in \mathbb{R}^{n \times k \times d_f}$

points features are directly passed into SLC module so that F_m is of size $M \times d$. The output of SLC module which is in shape $M \times d_o$ is directly output as new points features, as illustrated in Figure 4.4. Note that MLP_g is not implemented in HOMLP.

4.6.5 Object Classification Network

The network architecture for object classification is presented in Figure 4.5. It stacks four HOCs and one HOMLP to construct a hierarchical network. Parameters of each block are presented in the figure. Please see the caption for explanation. The network optionally takes points' normals as input. Once normal is omitted, PF and $\mathcal{F}$ in the first HOC do not exist, and operations related to them such as MLP_m are disabled. Hence, the output of $\text{MLP}_f(F'_m)$ becomes the feature map instead of F_m . Otherwise, normal is lifted by MLP_n whose output is treated as PF of the first HOC. Moreover, in order to urge the network to learn harder, the network is enforced to classify from a portion of the object during training. The idea is realized by classifying each point after HOMLP via shared Linear layer because the receptive field of each point after that block may not cover the entire object. The shared Linear layer computes score for each point which is passed into Softmax layer to determine Cross Entropy Loss. So in training, the mean of all 128 losses is backpropagated, while in testing, all scores are averaged for inference. Hyperparameters of all MLPs in the classification network including MLP_f , MLP_h , MLP_e , MLP_m and MLP_g are detailed in Appendix D.

4.6.6 Part Segmentation Network

The part segmentation network is designed following downsampling and upsampling pattern, as illustrated in Figure 4.6. The shape information is encoded via four HOCs

Figure 4.4: Detailed demonstration of HOC (left) and HOMLP (right). “PF” and “PC” are abbreviations of “points’ features” and “points’ coordinates” respectively. M denotes the number of input points, and n is the number of sampled centroids.

where the number of points decreases gradually. After that, four High Order Deconvolutions (HODCs) are stacked to raise point clouds back to original scale. HODC is slightly different from HOC in that it does not use FPS to sample centroids but receives points from a preceding HOC block as centroids instead, as visualized by skip links in the figure. So the number of centroids is not less than the number of input points in HODC so as to achieve upsampling, while it is reversed in HOC where the number of centroids is not larger than the number of input points. Furthermore, HOCs also send points’ features to HODCs via skip links. For instance, the first HODC concatenates the received features from the third HOC to its output and uses MLP_s to fuse them. Notice the last HODC interacts with the input layer by receiving original points coordinates and lifted points normal. Hyper-parameters of all MLPs in the segmentation network including MLP_f , MLP_h , MLP_e , MLP_m and MLP_g are presented in Appendix D.

4.7 Experiments

4.7.1 Datasets

The performance of HOPCNN is evaluated for point clouds classification on ModelNet40 [48] which contains 12311 objects from 40 categories. The standard training and testing split is used in all experiments where 9843 objects are used for training and the left 2468 samples for testing. The dataset released by [34] is adopted where 2048 points are uniformly sampled for each object. For fair comparison, the performance is compared

Figure 4.5: Illustration of HOPCNN object classification network. The architecture consists of four HOCs and one HOMLP block followed by one shared Linear layer. Hyperparameters of each block are listed in the figure where n denotes the number of sampled centroids, d_o is the dimensionality of features output by ALC or SLC modules, r is the rank imposed to convolutional kernels, and p is the dropout rate. Notice that HOMLP merely consumes points features.

based on different data augmentations and input representations. According to [25], data is augmented by anisotropic random scale and random horizontal rotation for unaligned scenario while random horizontal rotation is disabled for aligned case.

Remark: ModelNet10 dataset is not evaluated by HOPCNN for several reasons: 1) as the subset of ModelNet40, ModelNet10 contains much less samples so that overfitting is somewhat inevitable when HOPCNN is evaluated on it; yet PC-ELM-AE is a small ELM-based network that has less tunable parameters, it is suited for small dataset like ModelNet10; 2) ModelNet40 is more challenging than ModelNet10, so the evaluation on ModelNet40 is sufficient to demonstrate the better performance of HOPCNN; 3) HOPCNN is compared to recent methods, but recent state of the art networks seldomly evaluate on ModelNet10.

Point clouds part segmentation is conducted on ShapeNetPart dataset [53] that consists of 16881 models from 16 categories which are split into 14007 training samples and 2874 testing samples. Each object is labeled as 2 to 6 parts, and there are 50 different parts in total. The part segmentation network takes as input points' coordinates and points' normal. As for data augmentation, point clouds are firstly normalized to unit sphere and then perturbed by random jittering following [35].

Figure 4.6: Illustration of HOPCNN part segmentation network. The network is composed of four HOCs, four HODCs and one HOMLP followed by one shared Linear layer. Point cloud segmentation is treated as a per-point classification problem. Each HODC is skip linked to a HOC or the input layer to facilitate feature correspondence.

4.7.2 Implementation Specifics

The performance on ModelNet40 is compared according to three different conditions, *i.e.* unaligned dataset without normal, aligned dataset without normal and aligned dataset with normal. 1024 points are randomly sampled for each object in the first two conditions, but 2048 points are input to the network for the last condition. As for part segmentation on ShapeNetPart, the point clouds scale is fixed to 1024 points.

The two networks are realized with PyTorch on a NVIDIA GTX1080Ti GPU for classification and two NVIDIA Tesla P100 GPUs for segmentation. Adam optimizer is used with initial learning rate 0.001. The learning rate is decayed every 20 epochs with decay rate 0.7 and is clamped above 1.0^{-5} . The weight decay is set to 1.0^{-5} . Networks are trained with batch size 16. Batch normalization with momentum 0.9 and ELU activation are applied to every layer. Note that activations precede batch normalizations for ALC and SLC modules while batch normalizations are implemented prior to activations in all MLPs.

4.7.3 Classification Results

The network in Figure 4.5 is utilized to classify on ModelNet40, the performance of which is shown in Table 4.1. As stated before, comparisons are made based on different data augmentations and input representations. Existing state of the art can be roughly

Method	Alignment	Input	ModelNet40	
			Instance Accuracy	Class Accuracy
ECC [39]	unaligned	1000	87.4	83.2
PointNet [34]	unaligned	1024	89.2	86.2
PointNet++ [35]	unaligned	1024	90.7	-
SpecGCN [44]	unaligned	1024	91.5	-
Ψ -CNN [23]	unaligned	10k	92.0	88.7
PointCNN [25]	unaligned	1024	92.2	88.1
HOPCNN	unaligned	1024	92.0	88.5
SO-Net [24]	prealigned	2048	90.9	87.3
KC-Net [37]	prealigned	1024	91.0	-
3DmFV-Net [2]	prealigned	1024	91.4	-
Kd-Net [19]	prealigned	32768	91.8	88.5
PCNN [1]	prealigned	1024	92.3	-
PointCNN [25]	prealigned	1024	92.5	88.8
HOPCNN	prealigned	1024	93.3	90.1
SO-Net [24]	prealigned	5000 + N	93.4	90.8
HOPCNN	prealigned	2048 + N	93.9	91.0

Table 4.1: Object classification performance on ModelNet40. Alignment column declares whether the dataset is aligned or not. “Unaligned” means random horizontal rotation is used for data augmentation in training, while “pre-aligned” indicates random rotation is disabled. Input column specifies input representation with the number representing the number of input points and “+ N” indicating normal is used.

casted into three conditions, *i.e.* unaligned dataset using points’ coordinates only, aligned dataset using points’ coordinates only and aligned dataset using points’ coordinates plus points’ normal. Two metrics are reported for comparison, *i.e.* Instance Accuracy (%) and Class Accuracy (%). Instance accuracy is the overall accuracy that is computed by the number of correctly classified objects over the number of total objects. In order to calculate class accuracy, accuracy of each class is firstly obtained, and then accuracies of all classes are averaged to determine the expected class accuracy.

It is clear that HOPCNN can achieve competitive performance to state of the art in the first condition. In detail, HOPCNN can reach 92.0% for instance accuracy and 88.5% for class accuracy, which is comparable to PointCNN and Ψ -CNN. Considering that Ψ -CNN needs 10k points to achieve that performance, HOPCNN is more efficient and competitive with 1024 points as input. Meanwhile, HOPCNN overwhelmingly outperforms existing methods in aligned cases. It surpasses PointCNN evidently in the second condition by 0.8% for instance accuracy and 1.3% for class accuracy, demonstrating the proposed network is more competent to solve permutation variance problem of point convolution. In

addition, HOPCNN achieves new state of the art in the last condition where the dataset is pre-aligned and 2048 points plus normal are taken as input. The introduced method outperforms SO-Net that consumes 5000 points, verifying that detecting local correlations via convolution operators other than MLPs and poolings is vital.

As illustrated in Figure 4.5, HOPCNN can achieve impressive performance with quite small ranks, *e.g.* the rank of each layer is not larger than 3 for the classification network. Hence, the introduced method is more computationally efficient than directly convolving on bilinear-pooled and trilinear-pooled feature maps. To visualize the performance variation w.r.t. ranks, ranks of the first and the third HOCs are varied. Figure 4.7 shows that the optimal performance is attained when the rank of the first HOC is 1 and the rank of the third HOC is 3, which confirms that small ranks are enough for HOPCNN.

4.7.4 Segmentation Results

The effectiveness of HOPCNN is tested by segmenting object parts on ShapeNetPart dataset. The standard metrics, *i.e.* mean intersection over union (mIOU), is adopted for comparison. In the method, one-hot category labels are concatenated to the output of the last HODC, thus, every object is aware of its category. Category labels are also used to score out invalid scores for computing mIOU following [35].

Table 4.2 summarizes previous outstanding methods and compares them to HOPCNN. It is apparent that HOPCNN outperforms other methods with mIOU 85.5% and wins 6 categories out of 16. Comparing to some renowned MLP-based networks such as PointNet, PointNet++ and SO-Net, the introduced method is definitely superior. That indicates HOCs can capture more detailed and meaningful shape information via convolutions than MLPs and poolings. Besides, HOPCNN tends to achieve better performance on challenging categories like car, motorbike and rocket. It is presumably because HOC and HODC could understand more fine-grained structures with the help of high order statistics.

4.7.5 Ablation Study and Visualization

Ablation study is conducted on ModelNet40 to figure out key factors that contribute to the impressive performance. Results of ablation analysis are presented in Table 4.3 where only the first condition, *i.e.* unaligned dataset with 1024 points, is taken into account for simplicity. Above all, the effectiveness of merging BLC and TLC as proposed in Sec. 4.5 is tested. By comparing model (A) and (B) to model (D), it is clear that fusing BLC and TLC results in huge performance improvement, *e.g.* instance accuracy drops around

Method	mIOU	air-plane	bag	cap	car	chair	ear-phone	guitar	knife	lamp	laptop	motor-bike	mug	pistol	rocket	skate-board	table
Kd-Net [19]	82.3	80.1	74.6	74.3	70.3	88.6	74.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
PointNet [34]	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
A-SCN [49]	84.6	83.8	80.8	83.5	79.3	90.5	69.8	91.7	86.5	82.9	96.0	69.2	93.8	82.5	62.9	74.4	80.8
SSCNN [54]	84.7	81.6	81.7	81.9	75.2	90.2	74.9	93.0	86.1	84.7	95.6	66.7	92.7	81.6	60.6	82.9	82.1
KCNet [37]	84.7	82.8	81.5	86.4	77.6	90.3	76.8	91.0	87.2	84.5	95.5	69.2	94.4	81.6	60.1	75.2	81.3
SO-Net [24]	84.9	82.8	77.8	88.0	77.3	90.6	73.5	90.7	83.9	82.8	94.8	69.1	94.2	80.9	53.1	72.9	83.0
RS-Net [13]	84.9	82.7	86.4	84.1	78.2	90.4	69.3	91.4	87.0	83.5	95.4	66.0	92.6	81.8	56.1	75.8	82.2
PointNet++ [35]	85.1	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
PCNN [1]	85.1	82.4	80.1	85.5	79.5	90.8	73.2	91.3	86.0	85.0	95.7	73.2	94.8	83.3	51.0	75.0	81.8
DGCNN [45]	85.2	84.0	83.4	86.7	77.8	90.6	74.7	91.2	87.5	82.8	95.7	66.3	94.9	81.1	63.5	74.5	82.6
HOPCNN	85.5	83.6	85.0	85.6	80.7	91.0	74.1	91.5	87.2	83.0	96.0	73.6	95.7	82.5	63.5	75.0	81.8

Table 4.2: Object part segmentation results on ShapeNetPart. Instance mean IOU (mIOU %) is reported followed by IOU of each category.

Figure 4.7: Performance variation w.r.t. ranks. The performance is tested on the unaligned ModelNet40 with 1024 points' coordinates only. The first and the third HOC blocks are chosen for illustration.

0.7% to 0.9% if they are used exclusively. Considering that BLC can capture the second order statistics while TLC is able to detect then third order statistics, it can be concluded that the introduced network takes advantage of high order statistics and that the second order and the third order statistics benefit from each other. In addition, it is implicitly found that the model using ALC modules in HOCs is tremendously better than the model adopting SLCs. Instance accuracy drops about 1.8% if ALCs are replaced by SLCs. It is presumably because ALCs consume multiple feature maps, *i.e.* F_m , H_m and E_m , that are able to capture different feature patterns, which is advantageous for inspecting more complex local correlations.

Some object part segmentation results are further visualized in Figure 4.8. As illustrated in the figure, predictions of the introduced network are pretty close to ground truth

HOPCNN	Bilinear	Trilinear	ALC	SLC	Ins. Accuracy
(A)	✓		✓		91.1
(B)		✓	✓		91.3
(C)	✓	✓		✓	90.2
(D)	✓	✓	✓		92.0

Table 4.3: Ablation analysis of HOPCNN on ModelNet40. Instance accuracy is used as metrics for comparison. If “Bilinear” and “Trilinear” are both ticked, the model merges BLC and TLC as proposed. Otherwise, they are utilized exclusively. “ALC” and “SLC” indicate whether HOCs adopt asymmetric low-rank convolutions (ALCs)

Figure 4.8: Visualization of part segmentation. Odd columns are ground truth labels, and even columns are our predictions. More visualizations are available in Appendix E.

labels even for some challenging categories like car and motorbike. However, the visualization also shows that some conjunction parts are often wrongly recognized, *e.g.* the rear leg of the chair is partially segmented as the back and the end of the cable in the earphone is classified as cushion. These deficiencies indicate HOPCNN should improve its capability to detect edges. Although HOPCNN has some drawbacks in performance, it is still quite successful for part segmentation task.

4.8 Conclusion

In this chapter, HOPCNN is introduced to analyze unordered point clouds by convolutions. In order to make convolutional outputs permutation invariant, HOPCNN convolves on bilinear-pooled and trilinear-pooled feature maps for inspecting local correlations. Further, low-rank constraint is explicitly imposed to convolutional kernels so that huge computational cost can be reduced. By fusing BLC with TLC, HOC and HOMLP are constructed which work as the building blocks of HOPCNN. Mathematical derivation in-

icates these low-rank convolutions can be efficiently realized through 1×1 convolutions and element-wise products.

Experimental results show that HOPCNN can achieve competitive or better performance than state of the art on benchmarks. Ablation study demonstrates that the second order statistics captured by bilinear pooling and the third order statistics inspected by trilinear pooling work synergetically. HOPCNN benefits from the fusion of the second order and the third order statistics. Additionally, *asymmetric low-rank convolutions* (ALCs) work much better than *symmetric low-rank convolutions* (SLCs) presumably because ALCs operate on different feature maps of a local region that may encode distinct semantic meanings of the region.

Chapter 5

Conclusions and Future Work

5.1 Point Clouds-based ELM-AE

The thesis, first of all, presents a novel algorithm named as Point Clouds-based Extreme Learning Machine Auto-Encoder (PC-ELM-AE). PC-ELM-AE is the first attempt that successfully employs Extreme Learning Machine (ELM) strategy to process unordered point clouds. PC-ELM-AE can be separated as two parts, *i.e. feature extractor* that is able to extract per-point features and classifier that can project global features to category scores. *Feature extractor* stacks multiple *local structure* modules and Extreme Learning Machine Auto-Encoders (ELM-AEs) for hierarchical feature extraction. *Local structure* modules are responsible to form local correlations among neighboring points by k nearest neighbors (kNN), channel-wise average poolings and edge feature modules. So the output of *local structure* modules is capable of capturing local structural information which is fed into subsequent ELM-AE layers to improve discriminability. Meanwhile, points in deeper layers are supposed to have larger receptive fields so that features of deeper points becomes more abstract. The output of *feature extractor* is average pooled to construct a global descriptor of each object that is consumed by a single hidden-layer ELM classifier for computing scores.

PC-ELM-AE realizes order invariance by applying shared ELM-AEs and symmetric pooling functions across all points. Order invariance is vital for algorithms processing unordered data because it stabilizes the performance of algorithms, otherwise, the same 3D object could be classified as different classes based on its points' order, which makes no sense. In addition, PC-ELM-AE is quite efficient compared to deep learning methods in training because it employs ELM strategy. ELM strategy is backpropagation free so that only the forward pass is implemented in which parameters are determined via several

matrix multiplications.

Comparisons on benchmarks demonstrate that PC-ELM-AE outperforms existing ELM-based methods that manipulate multi-view images and voxel data and achieves better performance than some deep learning networks. These results disclose a promising future of PC-ELM-AE. However, the performance gap between PC-ELM-AE and state-of-the-art networks is quite big, so PC-ELM-AE can only achieve generally acceptable performance with such fast training speed. Ablation study confirms the effectiveness of edge feature module and local structure module. Accuracy drops a lot if they are disabled. Additionally, the efficacy of hierarchical feature extraction is verified by shallowing the network. Considering stacking more layers definitely harms efficiency, three *local structure* modules and three ELM-AEs are utilized trading of performance and efficiency.

In conclusion, PC-ELM-AE is an efficient network that adopts ELM strategy to manipulate unordered point clouds. It can achieve acceptable performance on ModelNet40 outperforming existing ELM-based networks and some deep learning networks.

5.2 High Order Point CNN

A novel deep neural network, High Order Point CNN (HOPCNN), is introduced in the thesis. HOPCNN exploits convolution operators to inspect local correlations among neighboring points. It does bilinear and trilinear poolings before convolutions and convolves on bilinear-pooled and trilinear-pooled feature maps to tackle the permutation variance problem of convolutions. Bilinear pooling incorporates the second order statistics and pairwise feature interactions while trilinear pooling is able to introduce the third order statistics and triplet feature interactions.

However, the size of bi/trilinear-pooled feature maps can be so huge that convolution operations become inefficient. So low-rank constraint is explicitly imposed to convolutional kernels to improve efficiency of convolutions. Four low-rank convolutions are constructed in light of the symmetry of bi/trilinear-pooled feature maps, *i.e.* *symmetric bilinear low-rank convolution* (SBLC), *asymmetric bilinear low-rank convolution* (ABLC), *symmetric trilinear low-rank convolution* (STLC) and *asymmetric trilinear low-rank convolution* (ATLC). HOPCNN merges SBLC with STLC to form *symmetric low-rank convolution* (SLC) and ABLC with ATLC to establish *asymmetric low-rank convolution* (ALC) by simply adding their outputs. Consequently, SLC and ALC both can capture the second and the third order statistics simultaneously. Based on ALC and SLC, High Order Convolution (HOC) module, High Order Deconvolution (HODC) module and High

Order MLP (HOMLP) module are constructed that work as building blocks of the final networks. HOPCNN object classification network stacks four HOCs and one HOMLP to enable hierarchical feature extraction. HOPCNN segmentation network stacks four HOCs, four HODCs and one HOMLP following downsampling and upsampling pattern.

Experimental results on ModelNet40 and ShapeNetPart datasets manifest HOPCNN can achieve competitive or even better performance than state of the art. In detail, HOPCNN outperforms PointCNN [25] by a large margin when ModelNet40 dataset is pre-aligned, which proves that HOC is more competent to handle permutation variance problem of point convolution than $\mathcal{X}$ -transformation network. Moreover, HOPCNN achieves overwhelmingly superior performance than MLP-based networks on ModelNet40 and ShapeNetPart, which confirms that convolution operators can detect more meaningful and detailed local correlations than MLPs and pooling functions. Ablation study demonstrates the effectiveness of merging *bilinear low-rank convolution* (BLC) with *trilinear low-rank convolution* (TLC) and that HOCs should adopt ALC rather than SLC. Finally, the introduced HOPCNN becomes more efficient with rather small ranks, *e.g.* ranks of all HOCs and HODCs are not larger than 4.

In conclusion, the introduced HOPCNN can ideally address the permutation variance problem to enable the use of convolution operators on unordered point clouds while achieving impressive performance on benchmarks.

5.3 Comparison

Although PC-ELM-AE and HOPCNN both achieve good performance on benchmarks, their rationales and ideas are tremendously distinct. PC-ELM-AE is essentially a Multi-Layer ELM (ML-ELM) that does not need backpropagation to update trainable parameters, while HOPCNN is a deep convolutional neural network (CNN) that iteratively updates parameters based on gradient descent methods such as Adam.

The most astonishing property of PC-ELM-AE is its fast training speed, which is induced by ELM strategy. Additionally, PC-ELM-AE has some critical properties for processing point clouds such as permutation invariance and hierarchy as discussed in Chapter 3. However, PC-ELM-AE has no obvious advantage over other methods in terms of inference time, because floating point operations per second (FLOPS) stay same as that of fully-connected layers in inference. The purpose of PC-ELM-AE is to figure out whether ELM strategy is suited for point data, and the main finding is that it can achieve generally acceptable performance with such fleeting training time.

HOPCNN takes around 8 hours to train on NVIDIA GTX 1080Ti for ModelNet40, but it outperforms PC-ELM-AE by 8% or so on ModelNet40. This is not beyond expectation because all parameters of HOPCNN are supervisedly fine-tuned for optimal performance, yet PC-ELM-AE merely determines parameters through one forward pass. HOPCNN achieves comparable or even better performance on classification and segmentation, as presented in Chapter 4, nevertheless, its training speed and inference speed is sort of slow due to the computation of high order statistics. So one possible improvement of HOPCNN is to speed up the computation of high order statistics, which will be discussed in future work.

It is apparent that PC-ELM-AE and HOPCNN have their distinct advantages and weaknesses against each other. PC-ELM-AE is suited for the situation that training time is limited while only acceptable performance is needed. HOPCNN is more accurate than PC-ELM-AE when compared on ModelNet40. Hence, it is adaptive to the circumstance that performance matters most and training time can be sacrificed. In short, HOPCNN is definitely superior to PC-ELM-AE in terms of accuracy, but the strength of PC-ELM-AE is its less training time.

5.4 Future Work

According to the two introduced algorithms, future work can be summarized as following.

Above all, it is possible to further improve the efficiency of PC-ELM-AE by incorporating Online Sequential Extreme Learning Machine (OS-ELM) [27]. The strategy used in PC-ELM-AE is conventional that all training samples are put into memory at one time, which is very memory-costing. Besides, training data may be collected with batches in real world, so the conventional strategy is not suited for online learning. OS-ELM is an online learning ELM algorithm that can be integrated into PC-ELM-AE to reduce memory occupation and make it adaptive to real world use. Meanwhile, it is possible to apply PC-ELM-AE to other tasks after incorporating OS-ELM, *e.g.* object part segmentation, semantic segmentation, object retrieval, *etc.*

Secondly, considering the second order statistics and the third order statistics work synergetically in HOPCNN, it is straightforward to probe into the possibility of incorporating even higher order statistics, *e.g.* the fourth order, the fifth order, *etc.* Higher order statistics may boost performance of HOPCNN but would harm its efficiency as well. It is worthwhile figuring out how to balance between the use of higher order statistics

and network efficiency, or more straightforward, finding methods to speed up the computation of high order statistics. Besides, HOPCNN can be generalized to other more challenging tasks such as semantic segmentation and object reconstruction to evaluate its effectiveness.

Thirdly, a more intuitive way to measure adjacency among points is to use geodesic distance rather than Euclidean distance. Since a point cloud forms a manifold in 3D space, it is not intuitive to use kNN for neighboring points searching. kNN-searched neighboring points (using Euclidean distance) may not be the nearest points to centroids in the aspect of manifold. However, approximating geodesic distances among points is non-trivial and existing algorithms for approximating geodesic distance on point clouds is quite time-consuming, so how to efficiently incorporate geodesic distance into networks for point clouds analysis is under exploration and deserves more research.

Lastly, proposing novel but effective algorithms for 3D object detection and semantic segmentation should be extensively researched. The two introduced algorithms are both designed for 3D object analysis, *i.e.* 3D object classification and 3D object part segmentation, but the application of 3D object analysis is rather limited in industry. On the contrary, 3D object detection and semantic segmentation are intensively demanded in reality, because they are significant for navigating autonomous vehicles and robotics. Therefore, how to design effective and efficient networks or improve the performance of existing networks for 3D object detection and semantic segmentation is imperative.

Bibliography

- [1] M. Atzmon, H. Maron, and Y. Lipman. Point Convolutional Neural Networks by Extension Operators. *arXiv preprint arXiv:1803.10091*, 2018.
- [2] Y. Ben-Shabat, M. Lindenbaum, and A. Fischer. 3D Point Cloud Classification and Segmentation using 3D Modified Fisher Vector Representation for Convolutional Neural Networks. *arXiv preprint arXiv:1711.08241*, 2017.
- [3] L. Bottou. Large-scale Machine Learning with Stochastic Gradient Descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.
- [4] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On Visual Similarity Based 3D Model Retrieval. In *Computer Graphics Forum*, volume 22, pages 223–232. Wiley Online Library, 2003.
- [5] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell. Compact Bilinear Pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 317–326, 2016.
- [6] 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*, pages 770–778, 2016.
- [7] G. E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14(8):1771–1800, 2002.
- [8] G. E. Hinton, S. Osindero, and Y.-W. Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, 2006.
- [9] B.-S. Hua, M.-K. Tran, and S.-K. Yeung. Pointwise Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 984–993, 2018.
- [10] G.-B. Huang, Z. Bai, L. L. C. Kasun, and C. M. Vong. Local Receptive Fields Based Extreme Learning Machine. *IEEE Computational Intelligence Magazine*, 10(2):18–29, 2015.

- [11] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang. Extreme Learning Machine for Regression and Multiclass Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529, 2012.
- [12] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme Learning Machine: Theory and Applications. *Neurocomputing*, 70(1-3):489–501, 2006.
- [13] Q. Huang, W. Wang, and U. Neumann. Recurrent Slice Networks for 3D Segmentation of Point Clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2626–2635, 2018.
- [14] A. Kanezaki, Y. Matsushita, and Y. Nishida. RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints. *arXiv preprint arXiv:1603.06208*, 2016.
- [15] L. L. C. Kasun, Y. Yang, G.-B. Huang, and Z. Zhang. Dimension Reduction with Extreme Learning Machine. *IEEE Transactions on Image Processing*, 25(8):3906–3918, 2016.
- [16] L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong. Representational Learning with Extreme Learning Machine for Big Data. *IEEE Intelligent Systems*, 28(6):31–34, 2013.
- [17] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation Invariant Spherical Harmonic Representation of 3D Shape Descriptors. In *Symposium on Geometry Processing*, volume 6, pages 156–164, 2003.
- [18] J.-H. Kim, K.-W. On, W. Lim, J. Kim, J.-W. Ha, and B.-T. Zhang. Hadamard Product for Low-Rank Bilinear Pooling. *arXiv preprint arXiv:1610.04325*, 2016.
- [19] 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*, pages 863–872, 2017.
- [20] S. Kong and C. Fowlkes. Low-rank Bilinear Pooling for Fine-Grained Classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 365–374, 2017.
- [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.
- [22] J. Lahoud and B. Ghanem. 2D-Driven 3D Object Detection in RGB-D Images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4622–4630, 2017.

- [23] H. Lei, N. Akhtar, and A. Mian. Octree Guided CNN with Spherical Kernels for 3D Point Clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9631–9640, 2019.
- [24] J. Li, B. M. Chen, and G. H. Lee. SO-Net: Self-Organizing Network for Point Cloud Analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9397–9406, 2018.
- [25] 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*, pages 820–830, 2018.
- [26] Y. Li, N. Wang, J. Liu, and X. Hou. Factorized Bilinear Models for Image Recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2079–2087, 2017.
- [27] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks*, 17(6):1411–1423, 2006.
- [28] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear CNN Models for Fine-grained Visual Recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1449–1457, 2015.
- [29] 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)*, pages 922–928. IEEE, 2015.
- [30] D. Meagher. Geometric Modeling Using Octree Encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.
- [31] F. Perronnin, J. Sánchez, and T. Mensink. Improving the Fisher Kernel for Large-scale Image Classification. In *European Conference on Computer Vision*, pages 143–156. Springer, 2010.
- [32] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Bilinear Classifiers for Visual Recognition. In *Advances in Neural Information Processing Systems*, pages 1482–1490, 2009.
- [33] 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*, pages 918–927, 2018.
- [34] 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*, pages 652–660, 2017.

- [35] 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*, pages 5099–5108, 2017.
- [36] G. Riegler, A. Osman Ulusoy, and A. Geiger. Octnet: Learning Deep 3D Representations at High Resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017.
- [37] 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*, pages 4548–4557, 2018.
- [38] B. Shi, S. Bai, Z. Zhou, and X. Bai. Deeppano: Deep Panoramic Representation for 3-D Shape Recognition. *IEEE Signal Processing Letters*, 22(12):2339–2343, 2015.
- [39] M. Simonovsky and N. Komodakis. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3693–3702, 2017.
- [40] S. Song and J. Xiao. Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 808–816, 2016.
- [41] 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*, pages 945–953, 2015.
- [42] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.
- [43] T. Tieleman and G. Hinton. Using Fast Weights to Improve Persistent Contrastive Divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1033–1040. ACM, 2009.
- [44] C. Wang, B. Samari, and K. Siddiqi. Local Spectral Graph Convolution for Point Set Feature Learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–66, 2018.
- [45] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *arXiv preprint arXiv:1801.07829*, 2018.

- [46] Y. Wang, Z. Xie, K. Xu, Y. Dou, and Y. Lei. An Efficient and Effective Convolutional Auto-encoder Extreme Learning Machine Network for 3D Feature Learning. *Neurocomputing*, 174:988–998, 2016.
- [47] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-adversarial Modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.
- [48] 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*, pages 1912–1920, 2015.
- [49] S. Xie, S. Liu, Z. Chen, and Z. Tu. Attentional ShapeContextNet for Point Cloud Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4606–4615, 2018.
- [50] Z. Xie, K. Xu, W. Shan, L. Liu, Y. Xiong, and H. Huang. Projective Feature Learning for 3D Shapes with Multi-View Depth Images. In *Computer Graphics Forum*, volume 34, pages 1–11. Wiley Online Library, 2015.
- [51] X. Xu and S. Todorovic. Beam Search for Learning a Deep Convolutional Neural Network of 3D Shapes. *arXiv preprint arXiv:1612.04774*, 2016.
- [52] 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*, pages 206–215, 2018.
- [53] 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)*, 35(6):210, 2016.
- [54] L. Yi, H. Su, X. Guo, and L. J. Guibas. Syncspeccnn: Synchronized Spectral CNN for 3D Shape Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017.
- [55] T. Yu, J. Meng, and J. Yuan. Multi-view Harmonized Bilinear Network for 3D Object Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 186–194, 2018.
- [56] Z. Yu, J. Yu, J. Fan, and D. Tao. Multi-modal Factorized Bilinear Pooling with Co-Attention Learning for Visual Question Answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1821–1830, 2017.

Figure B.3: PC-ELM-AE with the last ELM-AE layer ablated.

Figure B.4: PC-ELM-AE with the second and the last ELM-AEs ablated.

C Mathematical Derivation of Formulas in HOPCNN

This section provides more detailed derivation of formulas used in HOPCNN, *e.g.* formulas of SBLC, ABLC, STLC, ATLC, SLC and ALC, such that the intuition and idea behind HOPCNN can be understood better.

C.1 Formulas of SBLC

The output of convolution on symmetric bilinear-pooled feature map $B_m \in \mathbb{R}^{d_f \times d_f}$ with a kernel $W_b \in \mathbb{R}^{d_f \times d_f}$ and bias b can be formulated as

$$z_b = \text{vec}(W_b) \cdot \text{vec}(B_m) + b, \quad (1)$$

where $z_b \in \mathbb{R}$ denotes the output of *symmetric bilinear convolution* (SBC), and $\text{vec}(\cdot)$ denotes vectorization of matrix. In SBLC, bilinear-pooled feature map is symmetric and the kernel is assumed to be symmetric as well. To facilitate efficient computation, the rank of the kernel is constrained so that W_b can be factorized as the product of two low-

rank matrices. Thus, B_m and W_b can be formulated as

$$\begin{aligned} B_m &= \sum_{i=1}^k f_{mi} f_{mi}^T, \\ W_b &= U^T U, \end{aligned} \quad (2)$$

where $f_{mi} \in \mathbb{R}^{d_f}$ and $U \in \mathbb{R}^{r \times d_f}$. **Remark:** W_b is decomposed as the product of a low-rank matrix and its transpose because it is assumed to be symmetric. The first line of Equation (2) shows that only one feature map $F_m = [f_{m1}, f_{m2}, \dots, f_{mk}]^T$ where $F_m \in \mathbb{R}^{k \times d_f}$ is used to construct B_m . Then, Equation (2) is substituted into Equation (1), so z_b becomes

$$\begin{aligned} z_b &= \text{vec}(W_b) \cdot \text{vec}\left(\sum_{i=1}^k f_{mi} f_{mi}^T\right) + b = \sum_{i=1}^k f_{mi}^T W_b f_{mi} + b \\ &= \sum_{i=1}^k f_{mi}^T U^T U f_{mi} + b, \end{aligned} \quad (3)$$

where $z_b \in \mathbb{R}$ now becomes the output of SBLC with the low-rank factorization. The last line of Equation (3) demonstrates the realization of SBLC.

C.2 Formulas of ABLC

Different from SBLC where bilinear-pooled feature map and kernel are both symmetric, ABLC operates on asymmetric input by introducing another feature map $H_m = [h_{m1}, h_{m2}, \dots, h_{mk}]^T$ where $H_m \in \mathbb{R}^{k \times d_h}$. The input to ABLC is constructed through bilinear pooling over F_m and H_m :

$$B'_m = \sum_{i=1}^k f_{mi} h_{mi}^T, \quad (4)$$

where $h_{mi} \in \mathbb{R}^{d_h}$ and $B'_m \in \mathbb{R}^{d_f \times d_h}$ denotes the asymmetric bilinear-pooled feature map. Since the input is asymmetric, it is rational to assume the kernel being asymmetric as well. Hence, the asymmetric low-rank kernel can be factorized as the product of two different matrices, which is

$$W'_b = U^T V, \quad (5)$$

where W'_b denotes the asymmetric low-rank kernel, and $V \in \mathbb{R}^{r \times d_h}$. Following SBLC, the output of ABLC can be derived as

$$\begin{aligned} z'_b &= \text{vec}(W'_b) \cdot \text{vec}(B'_m) + b = \text{vec}(W'_b) \cdot \text{vec}\left(\sum_{i=1}^k f_{mi} h_{mi}^T\right) + b \\ &= \sum_{i=1}^k f_{mi}^T W'_b h_{mi} + b = \sum_{i=1}^k f_{mi}^T U^T V h_{mi} + b, \end{aligned} \quad (6)$$

where $z'_b \in \mathbb{R}$ denotes the output of ABLC. The realization of ABLC is clear now.

C.3 Formulas of STLC

Bilinear pooling is generalized to trilinear pooling for capturing the third order statistics. Considering that bilinear pooling is calculated by $B_m = \sum_{i=1}^k f_{mi} f_{mi}^T$ where the out product of features are summed, trilinear pooling over F_m is realized by two cascaded out products and summation, that is,

$$T_m = \sum_{i=1}^k f_{mi} \circ f_{mi} \circ f_{mi}, \quad (7)$$

where “ $\circ$ ” denotes out product, and T_m is a 3-dimensional symmetric tensor of size $d_f \times d_f \times d_f$ representing the symmetric trilinear-pooled feature map. The convolutional kernel acting on T_m must be a symmetric tensor as well, which is denoted by $W_t \in \mathbb{R}^{d_f \times d_f \times d_f}$. Similar to low-rank constraint imposed to W_b in Equation (2), W_t is factorized to restrict its rank. As indicated in Equation (2), the symmetric low-rank kernel for bilinear pooling can be rewritten as

$$W_b = U^T U = \sum_{n=1}^r u_n u_n^T, \quad (8)$$

where $U = [u_1, u_2, \dots, u_r]^T$ and $u_n \in \mathbb{R}^{d_f}$. Hence, the symmetric low-rank kernel for trilinear pooling can be obtained in a similar fashion by using two consecutive out products and summation:

$$W_t = \sum_{n=1}^r u_n \circ u_n \circ u_n. \quad (9)$$

Before doing convolution on T_m with W_t , they are reshaped to 2-dimensional matrices for simpler derivation, that is,

$$\begin{aligned} T_m^{2d} &= \sum_{i=1}^k \text{vec}(f_{mi} \circ f_{mi}) \circ f_{mi} = \sum_{i=1}^k \text{vec}(f_{mi} f_{mi}^T) f_{mi}^T, \\ W_t^{2d} &= \sum_{n=1}^r \text{vec}(u_n \circ u_n) \circ u_n = \sum_{n=1}^r \text{vec}(u_n u_n^T) u_n^T, \end{aligned} \quad (10)$$

where $T_m^{2d} \in \mathbb{R}^{d_f^2 \times d_f}$ denotes the reshaped matrix of T_m , and $W_t^{2d} \in \mathbb{R}^{d_f^2 \times d_f}$ denotes the reshaped matrix of W_t . So the output of STLC becomes

$$\begin{aligned} z_t &= \text{vec}(W_t^{2d}) \cdot \text{vec}(T_m^{2d}) + b = \text{vec}(W_t^{2d}) \cdot \text{vec}\left(\sum_{i=1}^k \text{vec}(f_{mi} f_{mi}^T) f_{mi}^T\right) + b \\ &= \sum_{i=1}^k \text{vec}(f_{mi} f_{mi}^T)^T W_t^{2d} f_{mi}^T + b = \sum_{i=1}^k \text{vec}(f_{mi} f_{mi}^T)^T \left[\sum_{n=1}^r \text{vec}(u_n u_n^T) u_n^T \right] f_{mi}^T + b, \end{aligned} \quad (11)$$

where z_t denotes the output of STLC. Given that $\sum_{n=1}^r \text{vec}(u_n u_n^T) u_n^T$ can be rewritten as

$$\left[\text{vec}(u_1 u_1^T), \quad \dots, \quad \text{vec}(u_r u_r^T) \right] \begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix}, \quad (12)$$

z_t turns out to be

$$\begin{aligned}
z_t &= \sum_{i=1}^k \left[\text{vec}(f_{mi}f_{mi}^T)^T \text{vec}(u_1u_1^T), \dots, \text{vec}(f_{mi}f_{mi}^T)^T \text{vec}(u_ru_r^T) \right] \begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} f_{mi}^T + b \\
&= \sum_{i=1}^k \left[u_1^T f_{mi}f_{mi}^T u_1, \dots, u_r^T f_{mi}f_{mi}^T u_r \right] \begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} f_{mi}^T + b \\
&= \sum_{i=1}^k \left(\begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} f_{mi} \odot \begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} f_{mi} \right)^T \begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} f_{mi}^T + b \\
&= \sum_{i=1}^k \mathbf{1}^T \left(\begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} f_{mi} \odot \begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} f_{mi} \odot \begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} f_{mi}^T \right) + b \\
&= \sum_{i=1}^k \mathbf{1}^T (U f_{mi} \odot U f_{mi} \odot U f_{mi}) + b,
\end{aligned} \tag{13}$$

where $\mathbf{1} \in \mathbb{R}^r$ is a all-ones vector and $U = [u_1, u_2, \dots, u_r]^T$. Up to now, the realization of STLC is derived.

C.4 Formulas of ATLC

The derivation of ATLC is quite similar to that of STLC except that three different feature maps, *i.e.* F_m , H_m and $E_m = [e_{m1}, e_{m2}, \dots, e_{mk}]^T$ where $E_m \in \mathbb{R}^{k \times d_e}$, are used to form asymmetric trilinear-pooled feature map:

$$T'_m = \sum_{i=1}^k f_{mi} \circ h_{mi} \circ e_{mi}, \tag{14}$$

where $e_{mi} \in \mathbb{R}^{d_e}$ and $T'_m \in \mathbb{R}^{d_f \times d_h \times d_e}$ denotes the asymmetric trilinear-pooled feature map. Since T'_m is asymmetric, it is reasonable to assume the kernel being asymmetric too. Inspired by the fact that the asymmetric low-rank kernel for bilinear pooling can be

rewritten as

$$W'_b = U^T V = \sum_{n=1}^r u_n v_n^T, \quad (15)$$

where $v_n \in \mathbb{R}^{d_h}$ and $V = [v_1, v_2, \dots, v_r]^T$, the asymmetric kernel for trilinear pooling $W'_t \in \mathbb{R}^{d_f \times d_h \times d_e}$ is factorized using low-rank constraint as

$$W'_t = \sum_{n=1}^r u_n \circ v_n \circ s_n, \quad (16)$$

where $v_n \in \mathbb{R}^{d_h}$ and $s_n \in \mathbb{R}^{d_e}$. Following STLC, T'_m and W'_t are rewritten as 2-dimensional matrices to facilitate computation, which leads to

$$\begin{aligned} T_m'^{2d} &= \sum_{i=1}^k \text{vec}(f_{mi} \circ h_{mi}) \circ e_{mi} = \sum_{i=1}^k \text{vec}(f_{mi} h_{mi}^T) e_{mi}^T, \\ W_t'^{2d} &= \sum_{n=1}^r \text{vec}(u_n \circ v_n) \circ s_n = \sum_{n=1}^r \text{vec}(u_n v_n^T) s_n^T, \end{aligned} \quad (17)$$

where $T_m'^{2d} \in \mathbb{R}^{d_f d_h \times d_e}$ denotes the reshaped matrix of T'_m and $W_t'^{2d} \in \mathbb{R}^{d_f d_h \times d_e}$ denotes the reshaped matrix of W'_t . So the output of ATLC can be computed by

$$\begin{aligned} z'_t &= \text{vec}(W_t'^{2d}) \cdot \text{vec}(T_m'^{2d}) + b = \text{vec}(W_t'^{2d}) \cdot \text{vec}\left(\sum_{i=1}^k \text{vec}(f_{mi} h_{mi}^T) e_{mi}^T\right) + b \\ &= \sum_{i=1}^k \text{vec}(f_{mi} h_{mi}^T)^T W_t'^{2d} e_{mi} + b = \sum_{i=1}^k \text{vec}(f_{mi} h_{mi}^T)^T \left[\sum_{n=1}^r \text{vec}(u_n v_n^T) s_n^T \right] e_{mi} + b, \end{aligned} \quad (18)$$

where $z'_t \in \mathbb{R}$ denotes the output of ATLC. Because $\sum_{n=1}^r \text{vec}(u_n v_n^T) s_n^T$ can be rewritten as

$$\begin{bmatrix} \text{vec}(u_1 v_1^T), & \dots, & \text{vec}(u_r v_r^T) \end{bmatrix} \begin{bmatrix} s_1^T \\ \vdots \\ s_r^T \end{bmatrix}, \quad (19)$$

z'_t becomes

$$\begin{aligned}
z'_t &= \sum_{i=1}^k \left[\text{vec}(f_{mi} h_{mi}^T)^T \text{vec}(u_1 v_1^T), \dots, \text{vec}(f_{mi} h_{mi}^T)^T \text{vec}(u_r v_r^T) \right] \begin{bmatrix} s_1^T \\ \vdots \\ s_r^T \end{bmatrix} e_{mi}^T + b \\
&= \sum_{i=1}^k \left[u_1^T f_{mi} h_{mi}^T v_1, \dots, u_r^T f_{mi} h_{mi}^T v_r \right] \begin{bmatrix} s_1^T \\ \vdots \\ s_r^T \end{bmatrix} e_{mi}^T + b \\
&= \sum_{i=1}^k \left(\begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} f_{mi} \odot \begin{bmatrix} v_1^T \\ \vdots \\ v_r^T \end{bmatrix} h_{mi} \right)^T \begin{bmatrix} s_1^T \\ \vdots \\ s_r^T \end{bmatrix} e_{mi}^T + b \\
&= \sum_{i=1}^k \mathbb{1}^T \left(\begin{bmatrix} u_1^T \\ \vdots \\ u_r^T \end{bmatrix} f_{mi} \odot \begin{bmatrix} v_1^T \\ \vdots \\ v_r^T \end{bmatrix} h_{mi} \odot \begin{bmatrix} s_1^T \\ \vdots \\ s_r^T \end{bmatrix} e_{mi}^T \right) + b \\
&= \sum_{i=1}^k \mathbb{1}^T (U f_{mi} \odot V h_{mi} \odot S e_{mi}) + b,
\end{aligned} \tag{20}$$

where $U = [u_1, u_2, \dots, u_r]^T$, $V = [v_1, v_2, \dots, v_r]^T$ and $S = [s_1, s_2, \dots, s_r]^T$. The realization of ATLC is uncovered.

C.5 Formulas of SLC and ALC

The outputs of SBLC and STLC are added together to form SLC so that SLC can detect the second and the third order statistics simultaneously. The output of SLC is denoted by z which can be computed by

$$\begin{aligned}
z &= z_b + z_t \\
&= \sum_{i=1}^k f_{mi}^T U^T U f_{mi} + b + \sum_{i=1}^k \mathbb{1}^T (U f_{mi} \odot U f_{mi} \odot U f_{mi}) + b \\
&= \sum_{i=1}^k \mathbb{1}^T (U f_{mi} \odot U f_{mi}) + \sum_{i=1}^k \mathbb{1}^T (U f_{mi} \odot U f_{mi} \odot U f_{mi}) + 2b.
\end{aligned} \tag{21}$$

For simplicity, the scalar $2b$ is replaced by b , thus,

$$\begin{aligned}
z &= \sum_{i=1}^k \mathbb{1}^T(Uf_{mi} \odot Uf_{mi}) + \sum_{i=1}^k \mathbb{1}^T(Uf_{mi} \odot Uf_{mi} \odot Uf_{mi}) + b \\
&= \sum_{i=1}^k \mathbb{1}^T(Uf_{mi} \odot Uf_{mi} + Uf_{mi} \odot Uf_{mi} \odot Uf_{mi}) + b \\
&= \sum_{i=1}^k \mathbb{1}^T[Uf_{mi} \odot Uf_{mi} \odot (\mathbb{1} + Uf_{mi})] + b.
\end{aligned} \tag{22}$$

The output of ALC can be computed in a similar way that z'_b and z'_t are added together:

$$\begin{aligned}
z' &= z'_b + z'_t \\
&= \sum_{i=1}^k f_{mi}^T U^T V h_{mi} + b + \sum_{i=1}^k \mathbb{1}^T(Uf_{mi} \odot Vh_{mi} \odot Se_{mi}) + b \\
&= \sum_{i=1}^k \mathbb{1}^T(Uf_{mi} \odot Vh_{mi}) + \sum_{i=1}^k \mathbb{1}^T(Uf_{mi} \odot Vh_{mi} \odot Se_{mi}) + 2b \\
&= \sum_{i=1}^k \mathbb{1}^T(Uf_{mi} \odot Vf_{mi}) + \sum_{i=1}^k \mathbb{1}^T(Uf_{mi} \odot Vh_{mi} \odot Se_{mi}) + b \\
&= \sum_{i=1}^k \mathbb{1}^T(Uf_{mi} \odot Vf_{mi} + Uf_{mi} \odot Vh_{mi} \odot Se_{mi}) + b \\
&= \sum_{i=1}^k \mathbb{1}^T[Uf_{mi} \odot Vh_{mi} \odot (\mathbb{1} + Se_{mi})] + b.
\end{aligned} \tag{23}$$

D Hyper-parameters of MLPs used in HOPCNN

This section is to clarify hyper-parameters of all MLPs used in object classification network and part segmentation network. Batch normalization with momentum 0.9 and ELU activation are used for all MLPs, and batch normalization is implemented before activation.

D.1 Object Classification Network

Classification network optionally takes points' normal as input. If normal is utilized, hyper-parameters of MLP_n are listed below

$$MLP_n(\mathbf{3} \rightarrow 32, 32), \quad (24)$$

where the bold number is the input dimensionality while numbers after arrow denotes output dimensionality of each layer. So MLP_n lifts 3-dimensional normal to 32 dimensions sequentially. Note that all MLPs discussed below follow this notation.

The 1st HOC. If normal are enabled, the first HOC contains 5 sets of MLPs, *i.e.* MLP_f , MLP_h , MLP_e , MLP_m and MLP_g , whose parameters are presented:

$$\begin{aligned} &MLP_f(\mathbf{3} \rightarrow 32, 32), \quad MLP_h(\mathbf{3} \rightarrow 32, 32, 32) \\ &MLP_e(\mathbf{3} \rightarrow 32, 32, 32), \quad MLP_m(\mathbf{64} \rightarrow 64) \\ &MLP_g(\mathbf{3} \rightarrow 16, 16). \end{aligned} \quad (25)$$

MLP_f , MLP_h and MLP_e all take normalized coordinates as input. MLP_m consumes the concatenation of the output of MLP_n and the output of MLP_f . MLP_g lifts coordinates of centroids to 16 dimensions. However, if normal is not utilized, MLP_n and MLP_m are both disabled.

The 2nd HOC. This layer has 5 MLPs:

$$\begin{aligned} &MLP_f(\mathbf{3} \rightarrow 32, 32), \quad MLP_h(\mathbf{3} \rightarrow 32, 32, 32) \\ &MLP_e(\mathbf{3} \rightarrow 32, 32, 32), \quad MLP_m(\mathbf{112} \rightarrow 96) \\ &MLP_g(\mathbf{3} \rightarrow 32, 32). \end{aligned} \quad (26)$$

Note that the output of MLP_f is concatenated with the output of the first HOC which is 80-dimensional, resulting in 112-dimensional features that are fed into MLP_m . MLP_m s in the third and the fourth layers repeat this procedure as well.

The 3rd HOC. This layer has 5 MLPs:

$$\begin{aligned} &MLP_f(\mathbf{3} \rightarrow 32, 32), \quad MLP_h(\mathbf{3} \rightarrow 32, 32, 32) \\ &MLP_e(\mathbf{3} \rightarrow 32, 32, 32), \quad MLP_m(\mathbf{192} \rightarrow 160) \\ &MLP_g(\mathbf{3} \rightarrow 64, 64). \end{aligned} \quad (27)$$

The 4th HOC. This layer has 5 MLPs:

$$\begin{aligned}
 & \text{MLP}_f(\mathbf{3} \rightarrow 32, 32), \quad \text{MLP}_h(\mathbf{3} \rightarrow 32, 32, 32) \\
 & \text{MLP}_e(\mathbf{3} \rightarrow 32, 32, 32), \quad \text{MLP}_m(\mathbf{352} \rightarrow 288) \\
 & \text{MLP}_g(\mathbf{3} \rightarrow 128, 128) .
 \end{aligned} \tag{28}$$

D.2 Part Segmentation Network

The encoder of part segmentation network works rather similarly to object classification network. Points' normals is lifted to higher dimensions through MLP_n whose hyper-parameters are

$$\text{MLP}_n(\mathbf{3} \rightarrow 32, 32) . \tag{29}$$

The 1st HOC. This layer has 5 MLPs:

$$\begin{aligned}
 & \text{MLP}_f(\mathbf{3} \rightarrow 32, 32), \quad \text{MLP}_h(\mathbf{3} \rightarrow 32, 32, 32) \\
 & \text{MLP}_e(\mathbf{3} \rightarrow 32, 32, 32), \quad \text{MLP}_m(\mathbf{64} \rightarrow 64) \\
 & \text{MLP}_g(\mathbf{3} \rightarrow 16, 16) .
 \end{aligned} \tag{30}$$

The 2nd HOC. This layer has 5 MLPs:

$$\begin{aligned}
 & \text{MLP}_f(\mathbf{3} \rightarrow 32, 32), \quad \text{MLP}_h(\mathbf{3} \rightarrow 32, 32, 32) \\
 & \text{MLP}_e(\mathbf{3} \rightarrow 32, 32, 32), \quad \text{MLP}_m(\mathbf{112} \rightarrow 96) \\
 & \text{MLP}_g(\mathbf{3} \rightarrow 32, 32) .
 \end{aligned} \tag{31}$$

The 3rd HOC. This layer has 5 MLPs:

$$\begin{aligned}
 & \text{MLP}_f(\mathbf{3} \rightarrow 32, 32), \quad \text{MLP}_h(\mathbf{3} \rightarrow 32, 32, 32) \\
 & \text{MLP}_e(\mathbf{3} \rightarrow 32, 32, 32), \quad \text{MLP}_m(\mathbf{192} \rightarrow 160) \\
 & \text{MLP}_g(\mathbf{3} \rightarrow 64, 64) .
 \end{aligned} \tag{32}$$

The 4th HOC. This layer has 5 MLPs:

$$\begin{aligned}
 & \text{MLP}_f(\mathbf{3} \rightarrow 32, 32), \quad \text{MLP}_h(\mathbf{3} \rightarrow 32, 32, 32) \\
 & \text{MLP}_e(\mathbf{3} \rightarrow 32, 32, 32), \quad \text{MLP}_m(\mathbf{352} \rightarrow 288) \\
 & \text{MLP}_g(\mathbf{3} \rightarrow 128, 128) .
 \end{aligned} \tag{33}$$

As mentioned in the main paper, HODC layers ought to receive features from previous HOCs by skip connections. Therefore, we use MLP_s to fuse current output features and received features, thus, each HODC has 6 MLPs.

The 1st HODC. This layer has 6 MLPs:

$$\begin{aligned}
 & \text{MLP}_f(\mathbf{3} \rightarrow 32, 32), \quad \text{MLP}_h(\mathbf{3} \rightarrow 32, 32, 32) \\
 & \text{MLP}_e(\mathbf{3} \rightarrow 32, 32, 32), \quad \text{MLP}_m(\mathbf{672} \rightarrow 672) \\
 & \text{MLP}_g(\mathbf{3} \rightarrow 64, 64), \quad \text{MLP}_s(\mathbf{640} \rightarrow 256) .
 \end{aligned} \tag{34}$$

MLP_m in this layer takes as input the concatenation of the output of MLP_f and the output of the fourth HOC. MLP_s receives the output of current layer (320-dimensional features) and the output of the third HOC (320-dimensional features), so the input dimensionality is 640.

The 2nd HODC. This layer has 6 MLPs:

$$\begin{aligned}
 & \text{MLP}_f(\mathbf{3} \rightarrow 32, 32), \quad \text{MLP}_h(\mathbf{3} \rightarrow 32, 32, 32) \\
 & \text{MLP}_e(\mathbf{3} \rightarrow 32, 32, 32), \quad \text{MLP}_m(\mathbf{288} \rightarrow 288) \\
 & \text{MLP}_g(\mathbf{3} \rightarrow 32, 32), \quad \text{MLP}_s(\mathbf{320} \rightarrow 192) .
 \end{aligned} \tag{35}$$

MLP_m takes as input the concatenation of the output of MLP_f and the output of the first HODC (256-dimensional features). MLP_s consumes the output of current layer (160-dimensional features) and the output of the second HOC (160-dimensional features), so the input dimensionality is 320.

The 3rd HODC. This layer has 6 MLPs:

$$\begin{aligned}
 & \text{MLP}_f(\mathbf{3} \rightarrow 32, 32), \quad \text{MLP}_h(\mathbf{3} \rightarrow 32, 32, 32) \\
 & \text{MLP}_e(\mathbf{3} \rightarrow 32, 32, 32), \quad \text{MLP}_m(\mathbf{224} \rightarrow 224) \\
 & \text{MLP}_g(\mathbf{3} \rightarrow 16, 16), \quad \text{MLP}_s(\mathbf{160} \rightarrow 128) .
 \end{aligned} \tag{36}$$

MLP_m takes as input the concatenation of the output of MLP_f and the output of the second HODC (192-dimensional features). MLP_s consumes the output of current layer (80-dimensional features) and the output of the second HOC (80-dimensional features), so the input dimensionality is 160.

The 4th HODC. This layer has 6 MLPs:

$$\begin{aligned} \text{MLP}_f(\mathbf{3} \rightarrow 32, 32), \quad \text{MLP}_h(\mathbf{3} \rightarrow 32, 32, 32) \\ \text{MLP}_e(\mathbf{3} \rightarrow 32, 32, 32), \quad \text{MLP}_m(\mathbf{160} \rightarrow 160) \\ \text{MLP}_g(\mathbf{3} \rightarrow 16, 16), \quad \text{MLP}_s(\mathbf{112} \rightarrow 96) . \end{aligned} \tag{37}$$

MLP_m takes as input the concatenation of the output of MLP_f and the output of the third HODC (128-dimensional features). MLP_s consumes the output of current layer (80-dimensional features) and lifted normal (32-dimensional features), so the input dimensionality is 112.

E Visualization of Part Segmentation of HOPCNN

More part segmentation results are shown in Figure E.1.

Figure E.1: Illustration of more part segmentation results. Odd columns are ground truth labels, and even columns are predictions.