McGill University

Graph Convolutional Neural Networks in Point Cloud Object Classification

A Master Project Report
by

Yingxue Zhang

Supervisor: Prof. Michael Rabbat

A report presented for the degree of
Master of Engineering

Department of Electrical and Computer Engineering
December 12, 2017

Contents

Abstract

Graph Convolutional Neural Networks (Graph-CNNs) are an extension of the traditional Convolutional Neural Networks (CNNs). Graph-CNNs are designed to handle data that does not have a regular structure such as data from social networks, transportation networks, molecule networks. Recently, researchers have proposed various of Graph-CNNs from both graph spectral and spectral-free perspectives that can achieve state of the art performance on structured data related tasks including the node classification (i.e., semi-supervised learning) as well as the whole graph classification (i.e., image classification, chemical compounds classification). In our project, we explore a new application of Graph-CNNs on classifying 3D point cloud data which has been obtained from sampling a 3D manifold, where the data naturally lies on various graph structures. We propose an architecture combines localized graph convolutions with two types of graph pooling operations which can achieve competitive performance on the point cloud classification problem using the 3D object classification benchmark dataset ModelNet. Besides, by leveraging the geometric information between points and learning features from different influence regions, our proposed method also achieves fast convergence and more stable performance compared with the competing schemes.

Acknowledgment

First, I want to thank my families. My debt to them is unbounded. Especially, a sincere thank is owed to my mom for showing me what is a strong woman should be like and providing me constant love and encouragement.

During my master study in McGill, I have been blessed with the friendship of many amazing people. First, I want to express my gratitude to Jay for all his genuine help and comments about my research work as well as all the funny one-liners he gave during the time we spent together. I owe a big thank-you to Srikanth for being the first person that I got to know and hung out with in the lab, and soon after became one of my best friends in Montreal. Next, I want to express my acknowledgment to Pal who provided me useful help and advice about my research and life. Besides, a special thank to Cody, who brings so much enthusiasm to the lab, and being your geek partner in the lab is my honor. I want to thank Min for being the person I can fully open my heart with, and for providing me comfort and encouragement in the time I needed the most. Next, I want to thank Xiaoqing Ma, for treating me as my big sister and helping me through tough times. Last but not least, I want to thank Yan Li, Yue Wen, Chen Xi, Chen Ma, Xiliang Zhu, Mido Assran, Yunhuang Zheng, Alex Saucan, Mohsen Rezaei, Hao Li, Kaiyuan Xie and all the other friends who given me unconditional encouragement and support throughout my master study.

I want to thank Prof. Mark Coates, for giving me suggestions about the course choice when I first joined McGill, for giving extremely useful advice and tips on how to do research in a professional way during the lab meeting and for providing me great help for job hunting.

At last, I would like to express my sincere gratitude to my supervisor, Prof. Michael Rabbat, who believed in me from the very start and guided me with extremely patience and encouragement during the past year, who as well provided me so many opportunities to experience new things, my first internship, my first paper, my first poster presentation. Thanks for those valuable opportunities, I have grown so much during the time we work together. Being his student is the most precious part of my experience at McGill, and I could not successfully finish my master study without all his great help and support.

Chapter 1

Introduction

Convolutional Neural Networks (CNNs) [1] have been extremely successful in the task where the data has a regular structure. For example, variants of CNNs break all records in 2D image recognition, segmentation, and detection tasks. Recent studies [2, 3] have also demonstrated the effectiveness of CNNs on sentence classification task in natural language processing. In this task, the language data can be regarded as 1D sequential data. However, there are cases where the data has meaningful information encoded in its support structure which does not lie in the Euclidean domain. For example, in the case of social networks, we would like to make some inferences based on social relationships. Thus, how to measure the interaction between the structure of data and its corresponding features is an area of interest.

One of the classical ways to tackle the problem when having both structure of the data and local node features is to use explicit graph based regularization [4, 5] such as graph Laplacian regularization. We penalize the difference between two nodes if they connect to each other to guarantee the smoothness of the graph data. However, this approach is not a powerful feature extraction tool as CNNs. It has limited learning capacity and flexibility, since it has this strong assumption that two nodes connect to each other should produce similar prediction label, which is not necessarily valid in some cases.

Previous researchers have investigated how to generalize the traditional CNNs to general graph structured data, which can effectively incorporate information from node features and its underlying graph structure. And at the same time, we want it to maintain the powerful latent feature learning nature as classical CNNs. Especially, with the emerging field of *Graph Signal*

Processing (GSP) [6] which provides insights and tools on how to handle the signals that lying on graphs, we have more theoretical background to better interpret the interaction between graph and its corresponding graph signal. Besides, in GSP, the generalized definition of filtering and convolution operations on graph signal have become the key ingredients in graph convolutional neural networks.

Bruna et al. [7, 8] first proposed the idea of using the definition of convolution in the graph spectral domain to construct their convolution block and using the graph multi-resolution clustering approach to achieve pooling steps in their network. Since the convolution operation is defined in the graph spectral domain, we call it spectral defined Graph-CNN.

Another line of work is called spectral free Graph-CNN, which does not directly operate on the graph spectral domain. Defferrard et al. [9] proposed using fast localized convolution operation by introducing recursive format of the Chebyshev polynomials which not only avoids explicitly calculating the graph Fourier basis but also enables the learnable parameters for each graph filter to be independent of the number of nodes. Besides, another special property is that the learned features are localized in spatial domain. *Diffusion-Convolutional Neural Networks* (DCNNs) [10] uses similar localized filtering idea as [9] but defined the convolution process directly on the spatial domain by searching the receptive filed at different scale using random walk.

The formulation of Graph-CNNs opens up a range of applications. Defferrard et al. [9] validated their model on an image classification task with decent performance and demonstrated the effectiveness of using Graph-CNNs to explore the similarity between features by forming up feature graph. Kipf and Welling [11] studied the application of the Graph-CNNs in semi-supervised learning and further simplified Defferrard's work by only using 1-hop neighborhoods of the graph during graph convolution. The simplified model is shown to be an efficient way to solve the node prediction problem in social networks and citation networks. Berg et al. [12] introduce a graph convolutional matrix completion method for end-to-end learning on bipartite user-item interaction graphs where users and items can be modeled in a joint representation space. Most of the application of Graph-CNNs so far has focused on when the graph structure is homogeneous. This means the graph feature maps extracted from the convolutional process are based on one single graph support, whether it is one social relationship graph, citation graph, the same 2D grid graph for every image or the same feature graph on every instance.

In our project, we mainly focused on applying the Graph-CNN method into a new application which is the 3D point cloud classification task. We aim to design graph filters that can be adaptive to multiple point cloud graph structures.

3D sensing has been a hot topic with the fast development of self-driving car technology. *Light detection and ranging* (LiDAR) sensors have shown promising prospects because of their 360 degrees of visibility and large sensing range. The data gathered from a LiDAR sensor is 3D point cloud data. Rather than following the typical way to process the 3D data and binning point cloud data into voxels, we are more interesting in directly using the raw point cloud data to proceed the downstream task without introducing the discretization error. Although 3D point cloud data is an unordered point set, the spatial distribution of the point cloud data encodes the underlying structure of the point set. Thus, 3D point cloud processing can be formulated as a graph signal processing problem. The effectiveness of graph signal processing for 3D point cloud data processing has been shown in the applications of point cloud data visualization, denoising, inpainting and compression [13, 14, 15, 16].

In this project, we explore the possibility of using Graph-CNN based method to achieve 3D point cloud classification and designed a Graph-CNN based deep learning framework which has shown promising results in 3D object classification benchmark dataset ModelNet [17].

The main contributions of this project include:

- Detailed review of state of the art Graph Convolutional Neural Networks algorithms and implemented ChebyNet [9], GCN [11] and DCNN [10], three most representative Graph Convolutional Neural Networks approaches.

- Explored the use case of Graph-CNNs in point cloud data and propose an architecture combines localized graph convolutions with two types of point cloud data specific designed pooling layer global pooling and multi-resolution pooling.

- By leveraging geometric information encoded in the point cloud data to explore local structure information, the proposed method not only achieve competitive performance but also improves the model stability and achieves fast model convergence comparing to previous point-based work on the 3D classification benchmark dataset.

This report is organized as follows. Chapter 2 elaborates on the background and related work of graph signal processing and Graph-CNNs. Chapter 3 describes the proposed Graph-CNN based 3D point cloud classification model. Chapter 4 presents the experiments and compare the proposed approach with the other state of the art methods for 3D object classification in deep learning. The investigation of key hyper-parameters is also presented. Finally, Chapter 5 summarizes the entire project and proposes future work.

Chapter 2

Background and Related Work

2.1 Graph Signal Processing

Graph Signal Processing (GSP) provides theoretical study of the interaction between graph and its corresponding graph signal. In GSP [6], the definition of the graph spectral domain is an analog to the classical frequency domain. The Fourier basis and frequency components of the graph are defined as the eigenvectors and eigenvalues of the graph Laplacian matrix or its variant normalized Laplacian matrix respectively. In graph setting, we have the definition of (weighted) Adjacency matrix $W \in \mathbb{R}^{n \times n}$ and diagonal degree matrix $D \in \mathbb{R}^{n \times n}$. The graph Laplacian matrix is defined as $L = D - W \in \mathbb{R}^{n \times n}$; and the normalized Laplacian is defined as $L_n = I_n - D^{-1/2} W D^{-1/2}$ where $I_n \in \mathbb{R}^{n \times n}$ is the identity matrix. For simplicity, the graphs we consider are the undirected and connected graph in this report. For an undirected graph, L_n is symmetric and it has a real-valued eigendecomposition $L_n = U \Lambda U^T$. We call the eigenvectors U of the normalized Laplacian matrix L_n the graph Fourier basis and the eigenvalues $\Lambda = diag([\lambda_0, \ldots, \lambda_{n-1}])$ of L_n the graph frequency components, where $0 = \lambda_0 < \lambda_1 \leq \cdots \leq \lambda_{n-1} \leq 2$ (We only consider connected graph setting).

The graph Fourier transform $x \in \mathbb{R}^n$ is defined as:

$$\hat{x} = Ux = \sum_{k=0}^{n-1} x_k u_k, \tag{2.1}$$

and the inverse Fourier transform is defined as:

$$x = U^T \hat{x} = \sum_{k=0}^{n-1} \hat{x}_k u'_k, \tag{2.2}$$

where u_k and u'_k are the k^{th} column of U and U^T.

Graph spectral filtering. The generalization of the fundamental operations such as filtering, convolution can also be defined. In classical signal processing, the filtering process is defined as amplifying or attenuating certain frequency components. In spectral graph signal processing, a similar definition is proposed where graph filtering is defined as the manipulation of the graph frequency components Λ. There are two ways to define graph filtering; in both spectral and vertex domain [6, 9]. The final form of spectral graph convolution process between graph filter and the input graph signal $\boldsymbol{x} \in \mathbb{R}^n$ is shown as follows:

$$\boldsymbol{y} = \hat{h}_f(L)\boldsymbol{x} = \hat{h}_f(U\Lambda U^T)\boldsymbol{x} = U\hat{h}_f(\Lambda)U^T\boldsymbol{x}, \tag{2.3}$$

where $\boldsymbol{y} \in \mathbb{R}^n$ is the filtered signal and $\hat{h}_f(L)$ is the graph filter.

Thus, for spectral graph filtering, the crucial step is to define the $\hat{h}_f(\Lambda)$ function:

$$\hat{h}_f(\Lambda) = \begin{bmatrix} \hat{h}(\lambda_0) & \cdots & \cdots & 0 \\ 0 & \hat{h}(\lambda_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{h}(\lambda_{n-1}) \end{bmatrix} \tag{2.4}$$

Polynomial parameterization for localized filters. The spectral graph filter $\hat{h}_f(\Lambda)$ can be represented by the polynomial expansion in terms of the eigenvalues:

$$\hat{h}_f(\Lambda) = \sum_{k=0}^{K} \theta_k \Lambda^k, \tag{2.5}$$

where $\theta \in \mathbb{R}^{K+1}$ is a vector of polynomial coefficients and K is the polynomial order.

This spectral graph filter has a graph vertex domain interpretation. When we plug in the Fourier basis term U, the graph filter $\hat{h}_f(L)$ can be written as

the polynomial of the Laplacian matrix:

$$\hat{h}_f(L) = U(\sum_{k=0}^{K} \theta_k \Lambda^k) U^T = \sum_{k=0}^{K} \theta_k (U \Lambda^k U^T) = \sum_{k=0}^{K} \theta_k L^k. \tag{2.6}$$

One of the advantages of this filtering method is that the filtering process is K-localized, meaning only the neighbor nodes within K hop region will affect the center nodes which guarantees the smoothness of the graph signal in spatial domain.

Chebyshev polynomial filtering approximation. Spectral graph filtering scheme guarantees flexibility of the graph filter. However, when we encounter large scale graph, the requirement of getting the Fourier basis and graph frequency components will introduce large computational burden as well as the memory burden since we need to store the dense Fourier basis U. Eigendecomposition of the Laplacian matrix to get the Fourier basis is around $O(N^2 \sim N^3)$ computational complexity [18].

To overcome these problems, [19] proposed to use the recursive format of the Chebyshev polynomial to approximate the convolution process. For $y \in [-1, 1]$, the Chebyshev polynomials $T_k(y)_{k=0,1,2,\ldots}$ are generate by:

$$T_k(y) = \begin{cases} 1 & \text{if } k = 0 \\ y & \text{if } k = 1 \\ 2yT_{k-1}(y) - T_{k-2}(y) & \text{if } k \geq 2. \end{cases} \tag{2.7}$$

The Chebyshev polynomials form an orthogonal basis for $L^2([-1, 1], \frac{dy}{\sqrt{1-y^2}})$, the Hilbert space of square integrable functions with respect to the measure [19]. If $h \in L^2([-1, 1], \frac{dy}{\sqrt{1-y^2}})$, it satisfies that:

$$\int_{-1}^{1} |h(y)|^2 \frac{dy}{\sqrt{1-y^2}} < \infty. \tag{2.8}$$

For every $h \in L^2([-1, 1], \frac{dy}{\sqrt{1-y^2}})$, it has a uniformly convergent Chebyshev series[19]:

$$h(y) = \frac{1}{2} c_o + \sum_{k=1}^{\infty} c_k T_k(y). \tag{2.9}$$

In the case of calculating $h(\lambda)$ for $\lambda \in [0, \lambda_{n-1}]$, it can be achieved by shifting the domain using the transformation $\lambda = a(y+1)$ with $a = \frac{\lambda_{n-1}}{2}$. We can denote the shifted Chebyshev polynomials $\bar{T}_k(\lambda) = T_k(\frac{\lambda - a}{a})$. The Chebyshev polynomials approximation is done by truncating the expansion term in equation 2.9 by its first K terms. Then Chebyshev polynomial approximation with domain shift can be written as:

$$h(\lambda) = \frac{1}{2}c_0 + \sum_{k=1}^{K} c_k \bar{T}_k(\lambda) \tag{2.10}$$

where

$$c_k := \frac{2}{\pi} \int_0^{\pi} \cos(k\theta) h(\frac{\lambda_{n-1}}{2}(\cos(\theta) + 1)) d\theta. \tag{2.11}$$

Thus, spectral graph filtering process can be well-approximated by a truncated expansion in terms of Chebyshev polynomials of the frequency component matrix Λ:

$$\boldsymbol{y} = h_\theta(L)\boldsymbol{x} = U \sum_{k=0}^{K} c_k T_k(\tilde{\Lambda}) U^T \boldsymbol{x}, \tag{2.12}$$

with the Chebyshev polynomials terms calculated recursively by:

$$T_k(\tilde{\Lambda}) = 2\tilde{\Lambda} T_{k-1}(\tilde{\Lambda}) - T_{k-2}(\tilde{\Lambda}). \tag{2.13}$$

The first two terms $T_0 = I$, $T_1 = \tilde{\Lambda}$, where $\tilde{\Lambda}$ is the rescaled version of the frequency component matrix $\tilde{\Lambda} = 2\Lambda/\lambda_{n-1} - I$.

Derive from Equation 2.12, we can farther simplify the filtering approximate process as the Chebyshev polynomials of the rescaled Laplacian matrix:

$$\boldsymbol{y} = \sum_{k=0}^{K} c_k T_k(\tilde{L}) \boldsymbol{x}, \tag{2.14}$$

with the Chebyshev polynomial term at order $K \geq 2$ calculated recursively as:

$$T_k(\tilde{L}) = 2\tilde{L} T_{k-1}(\tilde{L}) - T_{k-2}(\tilde{L}), \tag{2.15}$$

where the first two terms $T_0 = I$, $T_1 = \tilde{L}$ and $\tilde{L}$ is the rescaled version of the Laplacian matrix $\tilde{L} = 2L/\lambda_{n-1} - I$.

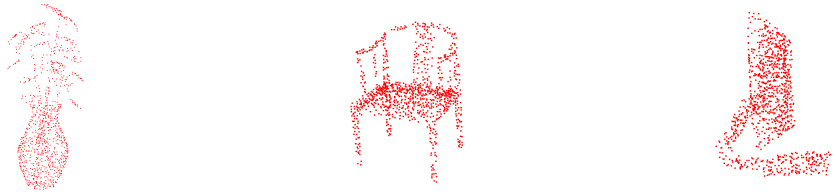
With the introduction of the recursive format of Chebyshev polynomial filter approximation process, we do not need to do the eigendecomposition

explicitly and we avoid dense matrix multiplication with the Fourier basis. Previous researchers have investigated the usage of Chebyshev graph filtering in distributed signal processing [20] and Graph-CNNs [9, 11]. To sum up, the Chebyshev polynomial filtering approximation guarantees the scalability of the model.

2.2 Point Cloud Data

2.2.1 Graph-Based Point Cloud Data Processing

3D point cloud data is the unordered point set with attributes associated with each point. In most cases, the attributes are the location coordinates. There could be other information such as RGB, textures. Examples of 3D point cloud data are shown in Figure 2.1. Since we want to directly use the irregular structured point cloud format of the data, instead of voxelizing it onto a 3D grid to avoid densifying the spare data and discretizing error, we need to introduce the graph to encode the irregular interconnection between points. Previous researchers have formulated the point cloud data processing as graph signal processing problem and demonstrated the effectiveness and efficiency in [14, 15, 16]. The advantages of using GSP tools to solve point cloud related problems lie in two folds. First, the graph support of the point set encodes valuable geometric information. Both local and global features of the point set are well captured. Second, GSP provides theoretical grounds for various generalization of classical signal processing operations which guarantee the flexible interaction between graph structure and graph signal.



(a) Flower pot (b) Chair (c) Monitor

Figure 2.1: 3D point cloud data examples

The graph structure of the 3D point set is not explicit. We need to

define the weighted adjacency matrix as follows, where the similarity and dependency between two points are measured by their Euclidean distance. There are two ways to measure the connectivity of the graph and decide each point's neighbors. One way is to set up a k-nearest neighbor graph directly, and the other one is set up this connection based on a proximity threshold ρ. In our project, we use the k-nearest neighbor graph for data preparation. Edge weights are typically determined using the Gaussian kernel function: for $i, j = 1, \ldots, n$,

$$W_{i,j} = \begin{cases} \exp(-\|x_i - x_j\|^2/\sigma^2) & \text{if } j \in \mathcal{A}_k(i) \\ 0 & \text{otherwise,} \end{cases} \tag{2.16}$$

where $\mathcal{A}_k(i)$ is the set of k nearest neighbors of vertex i.

In some cases, the point cloud data we gather has millions of points, which jeopardizes the efficient storing and fast processing the point cloud data. Besides, in some situations, we need to unify the point number among different objects such as 3D object classification problem. Thus, we need to perform some resampling scheme to compress the point set number. At the same time, we would love to select the subset of the original point cloud data that can better preserve the original data's information. We can easily do uniform subsampling, but the points that can distinguish one object from the other mainly rely on the contour of the object. A wiser choice would be preserving as much contour points as possible. We need contour-enhancing resampling technique to better captures the required contour information. Chen et al. [14] proposed a way to do fast resampling of point cloud via graph filtering, which we can specify the graph filter type to adapt to various of tasks such as contour-enhancing and graph signal denoising. The optimal goal is to find a series of resampling probabilities $\{\pi\}_{i=1}^{N}$ of each point to satisfy our design purpose.

Contour-enhancing using high pass graph filter In contour enhancing task, our goal is to design a graph filter to lower the probability of picking the low varying part of the data points and increase the probability of capturing the fast varying part which usually is on the contour of the objects. In [14], they designed a high pass Haar like graph filter to achieve this contour enhance resampling by relating the possibility of a point being sampled to the local variation of this point. We want to sample the points which are very different from its neighbors.

Given a point cloud data, we can represent it into a matrix format $X \in$

$\mathbb{R}^{n\times 3}$ with n points and each point is associated with the coordinate from x, y, z direction. As Chen et al. have proved in the paper, the sampling strategy π is proportional to the magnitude of the output features after the graph transformation,

$$\pi_i^\star \propto ||f_i(X)||_2, \tag{2.17}$$

where $f_i(X) \in \mathbb{R}^3$ is the i^{th} row of the graph transformation $f(X)$. In the case of contour enhancing task, the output feature $f(X)$ is the graph convolution result by applying high pass graph filter on the input coordinate data.

To guarantee the K-localized and shift-invariant, Chen et al. design graph filter as the polynomial of the graph shift which is similar to the vertex defined graph filter in equation 2.6:

$$f(X) = \sum_{k=0}^{K-1} \theta_k A^k X, \tag{2.18}$$

where A is the graph shift operator, $\theta \in \mathbb{R}^K$ is the coefficient vector of the polynomial term, and K is the order of the filter. For the high pass graph filtering case, we use the transition matrix $A = D^{-1}W \in \mathbb{R}^{n\times n}$ as our graph shift operator. The high pass filtering process involves designing a high pass Haar filter $h_{HH}(A)$:

$$f(X) = h_{HH}(A)X = (I - A)X. \tag{2.19}$$

The intuition behind this is that when the local variation of a point is high, its coordinates cannot be well approximated from the coordinates of its neighboring points. To extract better contour after the filter process, we want to sample points that have high local variation [14]. The final form of the resampling scheme is defined as:

$$\pi_i^\star \propto ||(h_{HH}(A)X)_i||_2^2 = ||x_i - \sum A_{i,j}x_j||_2^2. \tag{2.20}$$

Other applications Besides using high-pass graph filter to extract contour points of the original point cloud, we can handcraft graph filters to achieve other tasks such as signal denoising, point cloud registration, etc. Designing hand-craft graph filter to manipulate the point cloud data is insightful. It is an effective way to translate the original point cloud data to a feature map that can capture certain characteristics of the object. By designing a graph filter bank, it could potentially be a feature extraction step in learning based algorithm to generate salient features in object classification task. We will discuss the details of this perspective in Chapter 3.

2.3 Graph Convolutional Neural Networks

An image can be represented as a 2D grid graph where each pixel is one node in the graph. Then, we can regard traditional CNNs as a special case of the Graph-CNNs which the edge weights between each nodes are the same and the neighbor number of each node is consistent. There are two properties in image or grid graph that do not exist in general graph structure data. The first thing is that in an arbitrary graph each node has a different number of neighbors. Besides, the influence region of each node for 2D grid data naturally has a sense of order where the order sequence can be regarded as from top to bottom, left to right. But this spatial ordering does not exist in general graph structure. Thus, we need to come up new convolutional operation on the graph. In analogy to the traditional CNNs, we need to redefine two key operations, the convolution step and the pooling step. In the graph setting, previous researchers have proposed various algorithms to replace the convolution and pooling layer in tradition CNNs. We will discuss them respectively in the following section.

2.3.1 Graph Convolutional Step

Graph convolution step can be defined in purely graph spectral domain or in a spectral free fashion. We will elaborate some representative algorithms in both aspects. The difference between various of Graph-CNNs mainly lies in how to define the graph filter and how much freedom or restriction you put it on the learnable parameters.

2.3.1.1 Spectral Defined Convolutional Approach

Spectral CNNs [7] Bruna et al.[7] was the first one to propose defining the graph convolutional layer in the graph spectral domain. In the spectral approach, the convolutional operation is realized in the graph spectral domain as defined in Equation 2.3. It directly operates on the spectrum of the graph weight. The graph convolution layer is defined as follows:

$$\boldsymbol{x_{t+1,j}} = h(U \sum_{i=0}^{d_t-1} F_{t,i,j} U^T \boldsymbol{x_{t,i}}) \quad (j = 0, 1, \ldots, d_{t+1} - 1), \tag{2.21}$$

where h is a non-linear function applied on the vertex-wise values. $F_{t,i,j}$ is a $n \times n$ diagonal matrix. $\boldsymbol{x_t} = (x_{t,0}, x_{t,1}, \ldots, x_{t,d_t-1}) \in \mathbb{R}^{n \times d_t}$ and $\boldsymbol{x_{t+1}} =$

$(x_{t+1,0}, x_{t+1,1}, \ldots, x_{t+1,d_{t+1}-1}) \in \mathbb{R}^{n \times d_{t+1}}$ are the input and output of the graph convolution layer respectively. d_t and d_{t+1} are the input and output feature dimension respectively. U is the Eigenvectors of the graph Laplacian matrix and serves as the Fourier basis in graph convolutional process. In practice, U can be replaced by the first k Eigenvectors of the Laplacian matrix and denote as U_k since only the first Laplacian eigenvectors describing the smooth structure of the graph are useful in practice [7].

Using the learned parameters in graph spectral domain to measure the interaction between the graph signals is ground breaking. However, there are several aspects that can be improved. First, the learned filter is not localized in spatial domain. Second, it is not a scalable model since explicit getting the Fourier basis will involve around $O(N^2 \sim N^3)$ computation complexity. Third, the proposed method directly operates on the Fourier basis of graphs. It poses difficulties to transform the learned graph filter from one graph support to the other since using the same filter in different Fourier basis will get very different responses, which narrow the use case to only the data with the same graph support. At last, the learning complexity for this approach is high since the generation of each feature map needs N learnable parameters.

2.3.1.2 Spectral Free Convolutional Approach

We will elaborate three spectral free Graph-CNNs approaches including ChebyNet [9] and GCN [11] two follow up works of Spectral-CNN [7] using Chebyshev graph filtering approximations to avoid explicit going into the graph spectral domain, and Diffusion CNN (DCNN) which defines the graph convolution in a purely spatial fashion. Both algorithms achieve competitive performance and scalable learning capacity.

Graph-CNN with fast localized filtering (ChebyNet) [9] Defferrard et al. proposed the ChebyNet model. Comparing to the first Graph-CNNs model Spectral CNNs [7], ChebyNet tackles the limitation of Spectral CNNs mentioned above. The improvements lie in three folds: First, in Spectral CNNs [7], the feature learning process is not localized in space. The ChebyNet proposed to use the Chebyshev polynomial of the Laplacian matrix, which is defined in 2.14 as:

$$\boldsymbol{y} = \sum_{k=0}^{K} \theta_k T_k(\tilde{L})\boldsymbol{x}. \tag{2.22}$$

It is worth to mention that the reason we do rescale on the Laplacian matrix in 2.22 is to map the graph frequency components from [0,λ_{n-1}] to [-1,1] which not only guarantees the Chebyshev polynomial forms an orthogonal basis in [-1,1] but also guarantees the stability of the network. Since the Chebyshev polynomial term of the rescaled Laplacian matrix $T_k(\tilde{L})$ in nature is the linear combination of the polynomial of the Laplacian matrix. Laplacian matrix only works on 1-hop neighborhoods. So the linear combination of order K Laplacian matrix guarantees the K-hop space localization and reduces the learning complexity from $O(N)$ to $O(K)$. Thus, this K-localized filtering not only guarantees the feature we extract is localized in spatial domain but also enables the learnable filter parameters to be independent of the input graph dimension, which dramatically reduces the learning complexity.

Second, since Chebyshev polynomial approximation use this recursive format to calculate each polynomial term instead of going in to the graph spectral domain. It guarantees the computational efficiency as well as the memory efficient. In this way, we reduce the computational complexity to $O(K \cdot \varepsilon)$ if we use sparse matrix computation, where ε is the edge number of the graph and we can avoid storing the dense matrix U. To write in a more compact way, Defferrard et al [9] defined $\overline{\boldsymbol{x}} = T_k(\tilde{L})\boldsymbol{x} \in \mathbb{R}^n$. The Chebyshev filtering approximation can be written as $y = [\overline{x}_0, ..., \overline{x}_K]\theta$. The vector $\theta \in \mathbb{R}^{K+1}$ is the Chebyshev coefficients we need to learn during the model training process.

Third, the Fourier basis of the graph structure is never directly used in the learning process, which poses the potential to transform the learned graph filter parameters from one graph support to the other, which could be a powerful tool to deal with heterogeneous graph structure related problem.
Graph Convolutional Network (GCN) [11] This is a continuous work based on ChebyNet. The main difference between those two approaches is that in [11], it simplifies the convolution process with an order 1 Chebyshev approximation. Besides, since the graph frequency components for normalized Laplacian matrix lie in the range [0,λ_{n-1}], it makes another approximation that the biggest eigenvalue λ_{n-1} of the normalized Laplacian matrix from each graph is a fixed number 2. With those two assumptions, it simplifies the model structure dramatically, and because of its simplicity of the output representation after the graph convolutional layer, it is easy to scale to deeper layers structure without the involvement of pooling layer. Equation 2.22 is the filtering steps used in [9], which is the proper way to do Chebyshev approximation with recursive format. And equation 2.23 is the

results of order one approximation and with the assumption that $\lambda_{n-1} \approx 2$. The approximation convolution process in GCN is defined as:

$$\boldsymbol{y} \approx \theta_0 \boldsymbol{x} + \theta_1 (L - I_N)\boldsymbol{x}. \tag{2.23}$$

It even goes further to force $\theta = \theta_0 = -\theta_1$ so that the kernel only has one single trainable parameter.

$$\boldsymbol{y} \approx \theta (I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})\boldsymbol{x} = \theta \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \boldsymbol{x}, \tag{2.24}$$

where $\tilde{A} = A + I_N$ and$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. So the output of the graph convolution layer is going to be simple as:

$$\boldsymbol{H} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \boldsymbol{X} \Theta, \tag{2.25}$$

where filter matrix $\Theta \in \mathbb{R}^{m \times h}$ (m is the input features number and h is the feature map number in the convolution layer), $\boldsymbol{X}$ and $\boldsymbol{H}$ is the input and output graph features in the convolution layer. In this way, the generation of every feature map in the convolution layer only needs one trainable parameter, which is extremely beneficial for large scale of networks.

GCN has demonstrated to be the most effective method in semi-supervised learning problems so far such as node prediction problem in the social networks, citation networks, etc. It leverages the local node features and the structure of the data together effectively. However, because it only has one trainable parameter for each graph filter and only one-hop neighbor nodes will affect each center node. The model is only suitable for dealing with sparse graph or highly irregular graph related problem. When it comes to extremely regular graph with large average node degree, the model will tend to give extremely smooth output value.

Diffusion CNN (DCNN) [10] In DCNN, Atwood and Towsley proposed to use the random walk as the diffusion process on graphs. Rather than scanning a template of parameters across a grid-structured input like the standard convolution operation, the diffusion-convolution operation builds a latent representation by scanning a diffusion process across each node in a graph-structured input. Different abstraction level features are produced by applying diffusion of different hops of random walk. Besides, the authors also provide the theoretical proof on two isomorphic graphs will have the same diffusion-convolutional activation using their proposed model.

2.3.2 Pooling Step

In classical CNNs, the pooling step serves to aggregate the localized feature by retaining the most important information such as the average, sum or max value in each spatial neighborhood. The advantages of using the pooling layer lie in two folds. First, after the pooling step, we reduce the dimension of the input data which can lower the computational complexity so that we can trade off the spatial resolution for higher filter resolution [9]. Second, by adding a pooling layer, we can achieve invariance to translation and shifting. Since we aggregate the information in each spatial neighborhood, even if the feature shifts slightly in each receptive field, the extracted feature like max value, average value in each of the spatial neighborhood will mainly maintain the same so that we are able to capture common features between images. In graph related classification problem, there is no need to introduce pooling operation if we want to achieve vertex classification. However, to realize the whole graph classification problem, we need to define similar pooling operation to achieve similar goals as mentioned above.

Previous researchers proposed to use the concept of graph coarsening to achieve the graph pooling step. Bruna et al.[7] first proposed using multiresolution analysis on graphs to achieve nodes clustering, and then performing max-pooling in each of the clusters. Defferrard et al [9] designed more efficient clustering process by using Graclus method [21], which groups two nodes together by first finding an unmarked nodes i and then finding its unmarked neighbor j that maximizes the local normalized cut $W_{ij}(1/d_i + 1/d_j)$. To further achieve fast pooling, Defferrard et al. proposed to use a balanced binary tree to store the matched vertices. However, one of the limitation is that every time we can only reduce the size of the graph by a factor two at each resolution level. Other multi-resolution clustering methods can be found in [21, 22, 23].

2.4 Deep Learning on Point Sets

2.4.1 PointNet

Due to the irregular format of the point cloud data, when it comes to 3D object recognition, most previous researchers have been reconstructing a regular 3D voxel grid based on the original point cloud [17, 24] or treat it as

a collections of images [25]. Only a few have been looked into how to directly used the point cloud data to achieve the object classification. One of the point-based 3D classification algorithm we will emphasize here is called PointNet [26]. Qi et al. proposed a unified architecture shown in Figure 2.2 that reaches the state of art performance in both 3D object classification task and part segmentation task (per point label prediction) and at the same time relatively lower the learning complexity. PointNet uses the coordinate of the point cloud directly as input data without introducing discretization error.

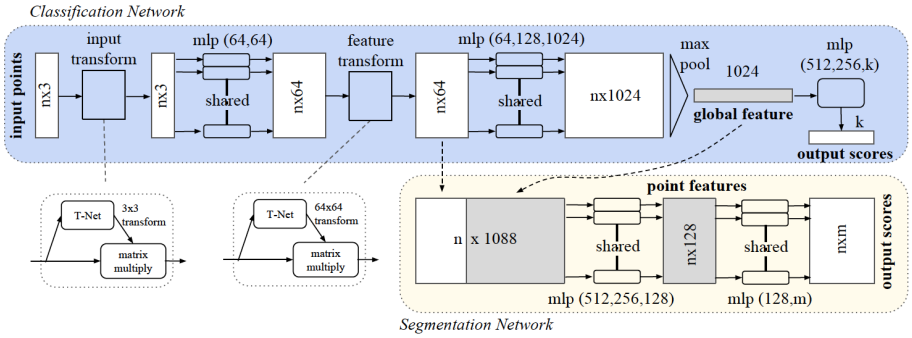


Figure 2.2: PointNet architecture (Reproduced from from Figure 2 in [26])

The three main challenges when using point cloud data directly to perform the classification task are as follows:

- Unordered input features

  Assume the input cloud point object has N points, we want any permutation of the point order can be identified as the same output categories. There are total $N!$ different order for the same cloud point object.

- How to maintains the interconnection among points

  The raw point cloud data does not explicit contain local structure information about the point cloud data. However, the neighbors of each point have meaningful information. We want to preserve this information during training.

- How to achieve invariance under transformation (rotation, flip, etc.)

We want our model to be invariant to certain transformations on the whole point cloud data such as rotation, flip and mirror operations.

PointNet tries to achieve the invariance to input order permutation by projecting the raw point cloud data into a high dimensional embedding space using the same symmetric functions on each point. Those functions are defined by *multi-layer perceptron* (MLP) and the MLP functions is shared across the whole point set. At the end of the network, it applies the final max-pooling layer to aggregate the global information from each feature map. All the operation in the Network architecture is done in the global fashion, which guarantees the invariance to input order. This approach is found to work effectively by experiments. A theoretical interpretation is provided from the perspective of universal approximation. It proves that their proposed network can approximate any set function that is continuous [26].

The other important step in PointNet is that the input data and intermediate latent representation will go through another step called T-Net transformation. It is an idea that originates from spatial transformation network [27]. The general idea is applying rigid or affine transformations to the input data or the middle latent feature map. This data-dependent spatial transformation process will serve as a canonicalization step [26]. Applying T-Net on the input data could align the input to a canonical space by learning the transformation matrix, and applying T-Net on the latent feature could align latent features to a canonical setting in feature space, which guarantees the transformation invariance property of the network. However, the local structure of the point cloud or the interconnection between points is not well captured by PointNet since all the operation is defined globally.

Because the lack of local information exploration, some recent works have been investigated on how to further explore the local structure features. Qi et al. [28] proposed to use Hierarchical feature learning by forming local point cluster and performing PointNet [29] in each local cluster to explore local features. Then, it aggregates the information in each cluster. It performs this process recursively to reduce the point dimension at each step.

In our proposed method, we use graph convolutional neural networks to explore the local structure features of the point cloud data at different influence scales. We will elaborate the details in the next chapter.

Chapter 3

Graph Convolutional Based 3D Object Classification Method

In Section 3, we explore using graph-based approach to solve the 3D point cloud classification problem. In Section 3.1, we will first provide some intuitions on the effect of different graph filters on the point cloud data, which inspired us to use graph-based approach to solve the point cloud classification problem. Then, in Section 3.2, we present the proposed model. In analogy to CNN, the graph-CNNs have two key operations including graph convolution step and graph pooling step. Under the point cloud format of data, we proposed an architecture combines localized graph convolutions by Chebyshev polynomial filtering scheme and two types of graph pooling operations, which both achieve competitive performance. We will elaborate on details of each step in the following Section.

3.1 Intuition Explanation

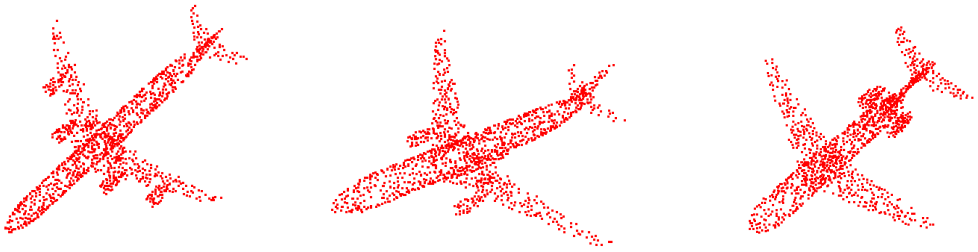
In Section 2.2.1, we explain that previous work has demonstrated the effectiveness of graph-based approach on 3D point cloud data. In this project, we explore how we can use a graph approach to solve the point cloud object classification problem.

Initial, we used similar method as Chen et al. [14] to handcraft various of graph filters and convolve with the input point cloud data. The point cloud data filtering process is shown in Equation 3.1.

$$\boldsymbol{y} = h(L)\boldsymbol{x}, \tag{3.1}$$

where $\boldsymbol{x} \in \mathbb{R}^{n\times 3}$ is the input point cloud data and $h(L)$ is the defined graph filter and $\boldsymbol{y} \in \mathbb{R}^{n\times 3}$ is the filtered graph signal. We experimented on how different graph convolution operations would affect the output point cloud features on the 3D object classification benchmark dataset ModelNet [17].

We discovered that doing graph convolution with the same type graph filter will generate similar point features of the objects coming from the same object category. Besides, when we visualized the points that have the largest ℓ_2 norm after the convolution translation, we found out that points with largest ℓ_2 norm will summarize similar points for each object category. It is an effective way to translate the input graph signal to a feature map that can capture the characteristics of each object class, which could potentially be a feature extraction step in learning based algorithms to generate salient features in object classification task. The visualization of the 300 points with the largest ℓ_2 norm value after different graph convolutions are shown in the following. The original point cloud data from the airplane class is shown in Figure 3.1. Figure 3.2 and 3.3 show the convolution results from three objects in airplane class using high pass graph filter and low pass graph filter respectively. We can observe that after the same graph filtering process, all the objects coming from the same class can be summarized by similar key points. In other word, objects that come from the same class have similar energy distribution after the same graph filtering operation.



(a) Airplane 1 (b) Airplane 2 (c) Airplane 3

Figure 3.1: Three point cloud objects from Airplane class

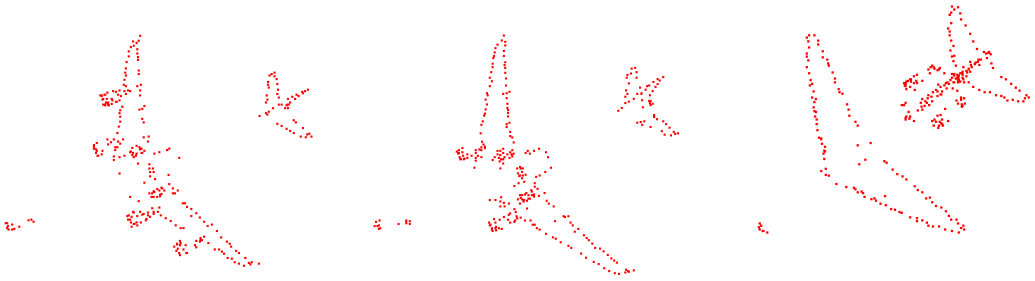


Figure 3.2: 300 points with highest ℓ_2 norm after high pass graph filtering

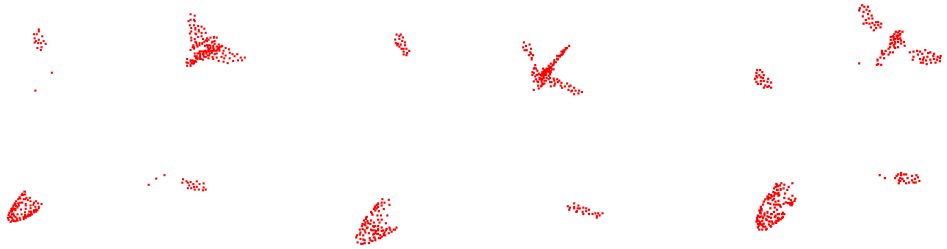


Figure 3.3: 300 points with highest ℓ_2 norm after low pass graph filtering

In the typical signal classification problem, various of features have been used to distinguish different class such as maximum, minimum, variance, zero crossing, etc. Borrowing this idea, we take one step further to analyze the statistic of the filtered graph signal $\boldsymbol{y} \in \mathbb{R}^{n\times 3}$. Besides the maximum value, we experienced on various of global features including mean and variance as well, which will be applied on our proposed model as another way to gather global feature.

Thus, different graph filter can preserve certain part of information about the point cloud, which can be used to separate different classes. Based on this idea, we proposed that instead of using handcraft graph filters, we design learnable graph filter banks which can map the input graph signal to various feature maps. Doing so enables to learn the graph filter parameters that can optimize the downstream classification task. Following this idea, we proposed our model in Section 3.3.

3.2 Data Preprocessing

First, since our proposed method needs to have the same number of input points, we preprocess the dataset by unifying the point number from each point cloud object using a subsampling scheme. Three subsampling methods, the graph-based contour-enhancing sampling method (according to section 2.2), the uniform sampling method and the farthest sampling method, are implemented for comparison. Each of the data subsampling method has its own advantages. Using the contour-enhancing sampling method provides better structure information of the point cloud with limited points. Using uniform subsampling is the most computationally efficient way to prepare the data. The farthest sampling approach guarantees that the sampled points are the most scattered on the point cloud, which provides more stable point distribution.

After we perform data subsampling, we use the preprocessed data to construct a k nearest neighbor graph based on equation 3.2, where edge weights are determined using the Gaussian kernel function: for $i, j = 1, \ldots, n$,

$$W_{i,j} = \begin{cases} \exp(-\|x_i - x_j\|^2/\sigma^2) & \text{if } j \in \mathcal{A}_k(i) \\ 0 & \text{otherwise,} \end{cases} \tag{3.2}$$

where $\mathcal{A}_k(i)$ is the set of k nearest neighbors of vertex i and σ is a parameter.

3.3 Proposed Model

Similar to the classical CNNs architecture, we use three main types of layers including graph convolutional layer, pooling layer, and fully-connected layer in our proposed model. Every convolutional layer and pooling layer is followed by the ReLU activation. We proposed two different architectures, and the difference mainly lies on how to perform the pooling step. The first proposed architecture only targets global features by performing global pooling to aggregate the global signature of each object without performing graph coarsening process. The model architecture of this method is shown in Figure 3.4. The second proposed model involves multi-resolution pooling step which is similar to the graph coarsening step but simplify the process by using the special characteristic of point cloud data. The model architecture is shown

in Figure 3.5. We will provide detailed explanation of the graph convolution step and each pooling scheme we used in the following section.

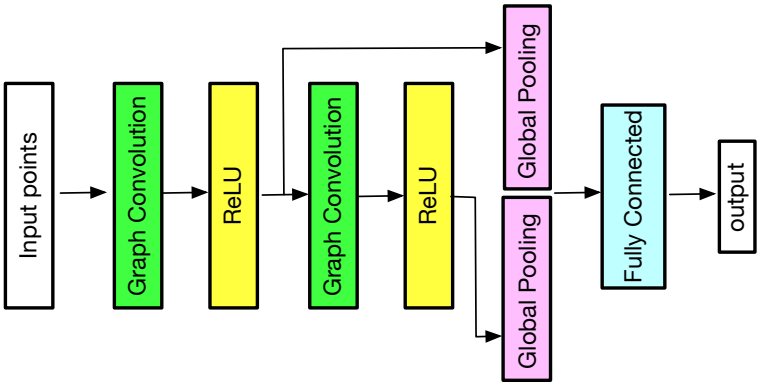


Figure 3.4: Overall architecture with global pooling scheme

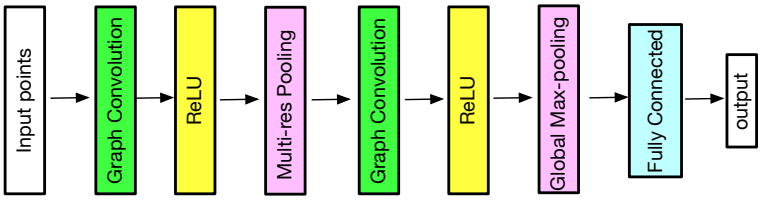


Figure 3.5: Overall architecture with multi-resolution pooling scheme

3.3.1 Convolutional Layer

During the training process, our goal is to train a set of graph filter coefficients that can translate the input signal to latent feature maps that can capture the structure information about this object. We treat the x, y, z coordinates of the point cloud as input graph signals. The convolution is done in a spectral free fashion by using Chebyshev localized filtering scheme. As we present in section 2.1, spectral convolution process can be well-approximated by a truncated expansion in terms of the recursive format of order K Chebyshev polynomial [19], which avoids the need to explicitly calculate the graph Fourier basis. Defferrard et al. [9] has demonstrated the effectiveness of this convolution block in homogeneous graph related prediction task. Thus,

we use a similar scheme to deal with heterogeneous graphs in our proposed method. The final form of the convolutional layer is:

$$y = g_\theta(L)x = \sum_{k=0}^{K} \theta_k T_k(\tilde{L})x, \tag{3.3}$$

where $T_k(\tilde{L})$ is the Chebyshev polynomial at order k, $\theta \in \mathbb{R}^{K+1}$ is the learnable graph filter parameter vector, $x \in \mathbb{R}^N$ is one input graph signal, and $y \in \mathbb{R}^N$ is the output of the convolution step. The Chebyshev polynomial term at order k is defined recursively as:

$$T_k(\tilde{L}) = 2\tilde{L}T_{k-1}(\tilde{L}) - T_{k-2}(\tilde{L}), \tag{3.4}$$

with the first two terms $T_0 = I$, $T_1 = \tilde{L}$, and $\tilde{L}$ is the rescaled Laplacian matrix.

To incorporate features from different abstraction level and increase the learning capacity of the model, we use two convolutional layer structure by feeding the output feature maps from the first convolution layer into the input of the second convolutional layer. Using more than two convolutional layers does not lead to better performance under our current dataset, one major reason could be the limited instance number of the dataset. This convolution process can be easily implemented as multi-layer perceptron. The architecture for convolution layer is shown in Figure 3.6.

3.3.2 Pooling Layers

The feature maps we get from the convolutional layer are a pointwise latent representation of the point set. Thus, we need to implement pooling operation to aggregate the information from the whole graph and its corresponding graph signals. We propose two ways to realize the pooling operation in graph-CNNs based method for point cloud classification. One is purely using global pooling operation on each of the feature maps coming out of the graph convolutional layers pooling without sacrificing the resolution of the graph structure. The other approach is using graph multi-resolution pooling idea introduced in Section 2.3.2. We will discuss the details of those two pooling approaches respectively in the following.

Global pooling As we discussed in section 3.1.1, we summarize global feature of each feature maps by performing global operation such as finding

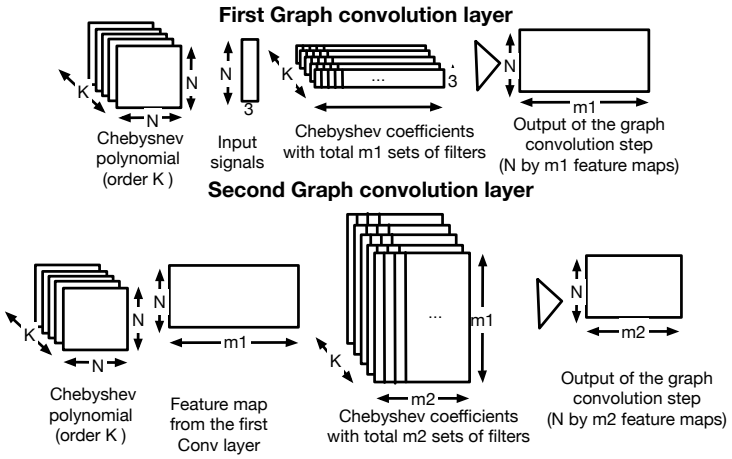


Figure 3.6: Convolution layer

the maximum, mean and variance value of each feature map. Empirically, the combination of the max-pooling and variance pooling provides the best performance. We choose to implement 1-max-pooling and variance pooling on the output of each convolutional layer to obtain the global signature of each object. Since this is a global operation, we guarantee the order invariance requirement. Max-pooling aims to pick the most distinctive points that separate one category from others. Variance pooling gives us the energy fluctuation level after the convolution. A max-pooling global feature aggregation layer is also shown to be effective in PointNet [26]. We do this pooling layer on feature maps from every convolutional layer. The final global feature is the concatenation of pooling results from each convolutional layer to add the variety of the feature abstraction. The architecture details for global pooling layer is shown in figure 3.7.

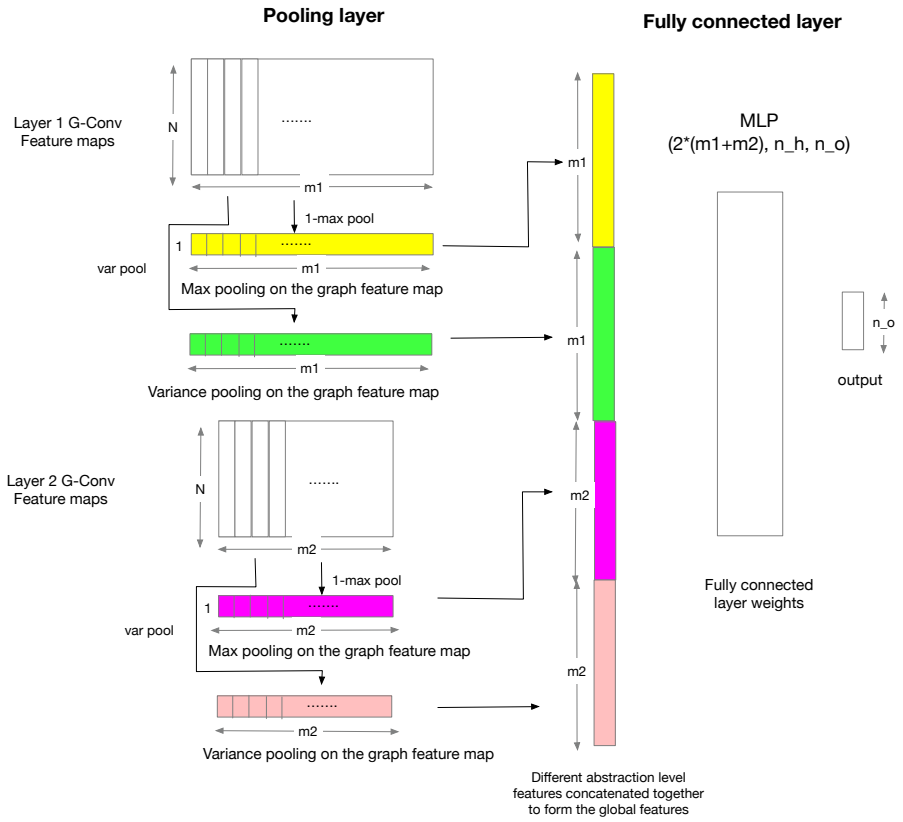


Figure 3.7: Global pooling/MLP layer

Multi-resolution pooling Graph clustering or graph coarsening is a computationally intensive task. Graph coarsening in essence tries to look at the graph structure at different resolution by grouping similar nodes together. It requires that the coarser version of the graph can still preserves local geometric structures. In the special case of point cloud data, we can leverage the geometry inherent to the data to avoid explicitly calculating the graph coarsening or clustering. Instead of using graph coarsening process as proposed in [7, 9], we realize multi-resolution pooling by sub-sampling a set of points that are most scattered from each other. This is done by first sampling a random point and adding it to the sampling set. The next sampling point is taken to be the one furthest from the initial sampling point, and subsequent sampling points are those that are mutually furthest from all other sampling points. After reaching the desired number of sampling points M (resolution at the

next level), every point in this sampling set will server as a centroid point. We find the k-nearest neighbor of each centroid point and identify them as a cluster. Then the local information aggregation step is realized by doing max-pooling on the feature maps gathered after the convolutional layer at each of the cluster. At the end of the last multi-resolution pooling layer, a final global max-pooling layer is attached to the network for the generation of final object latent representation. By introducing the multi-resolution pooling layer, we can shrink the dimension of the graph to the cluster number M, which reduces the computation complexity during the calculation of the following graph convolutional layer as well as it reduces the learnable parameter number. However, the tradeoff here is the calculation of the farthest sampling and the clustering process will add more computation complexity. An example of applying the multi-resolution pooling is shown in Figure 3.8.

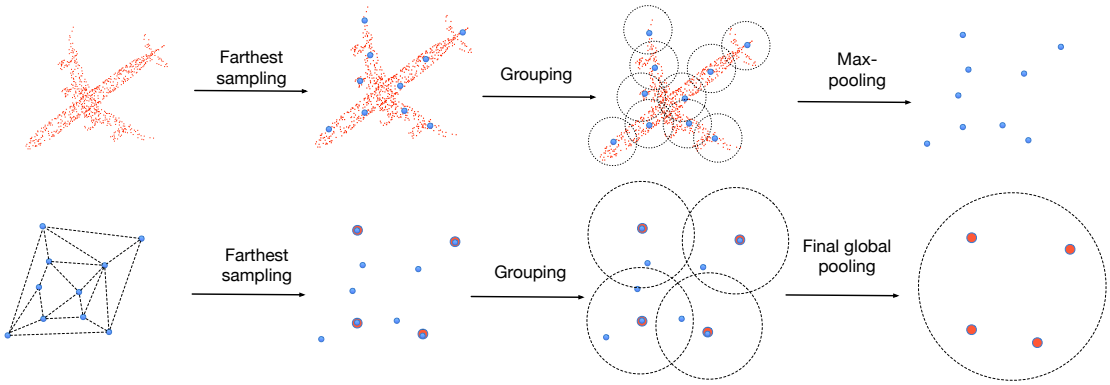


Figure 3.8: Multi-resolution pooling in point set

3.3.3 Fully Connected Layer

The final latent representations of each object produced by the convolutional layers are flattened into a vector and passed through one fully-connected layer and an output layer with softmax activation, where the number of outputs is the number of classes. Thus, the final output corresponds to a vector with entries giving the predicted probability of the input belonging to each of the possible classes. The fully connected layer and output layer for architecture with global pooling scheme is shown in figure 3.7. Similar fully connected layers for architecture with multi-resolution pooling scheme is attached to the last convolutional layer as well.

Chapter 4

Experiment

4.1 3D Point Cloud Classification

4.1.1 Dataset Description

In our experiment, we evaluate our algorithm on the ModelNet40 and ModelNet10 datasets, which have been broadly used as shape classification benchmark for 3D object recognition task [17]. ModelNet10 contains 4899 CAD models, split into 3991 for training and 908 for testing. ModelNet40 contains 12311 CAD models from 40 categories, split into 9843 for training and 2468 for testing. In ModelNet10 all models are oriented, and in ModelNet40 they are not oriented. The PointNet authors provide the point cloud format of the ModelNet data where 2048 points are uniformly sampled on the mesh faces of the CAD object. The object class distribution for both ModelNet 10 and ModelNet40 (training set) are shown in Figure 4.1 and Figure 4.2. From the figure, we can see that both datasets have unbalanced class distribution, which poses a challenge in model training.

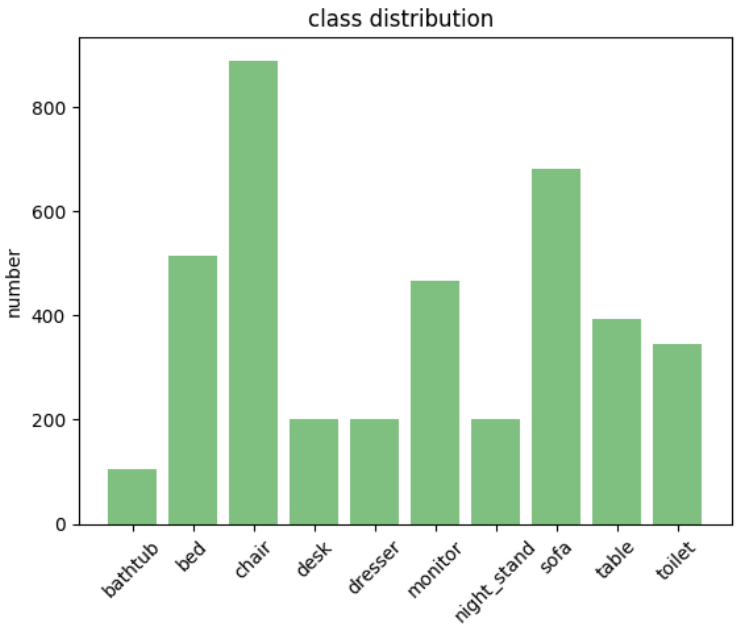


Figure 4.1: Data distribution for ModelNet10 (Training set)

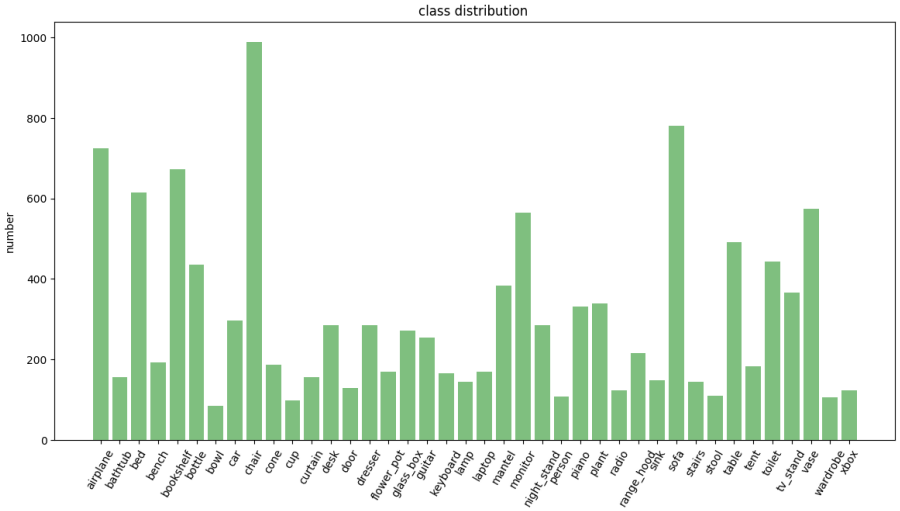
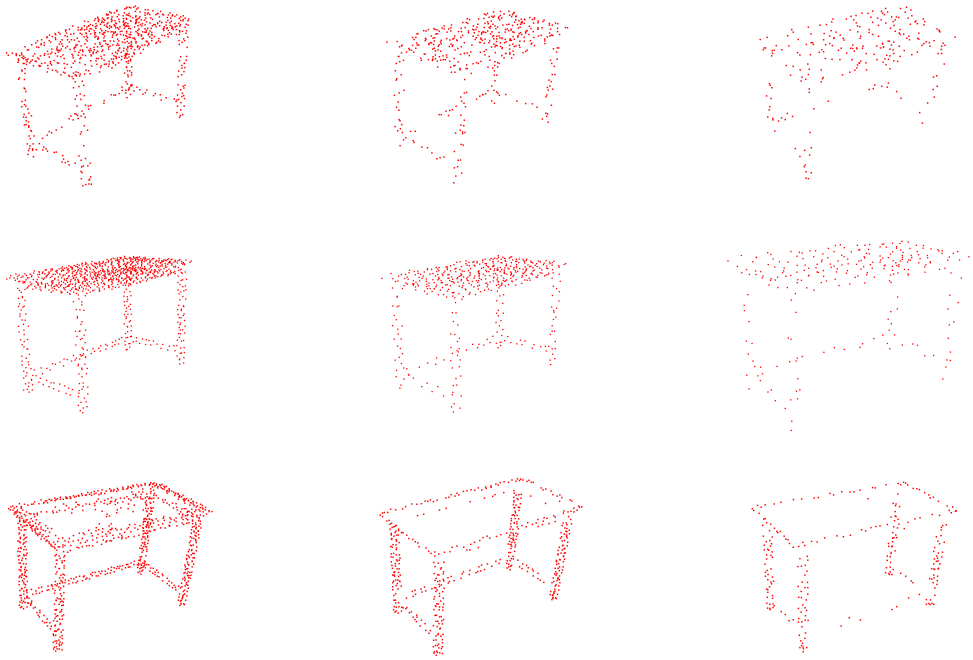


Figure 4.2: Data distribution for ModelNet40 (Training set)

4.1.2 Data Preparation

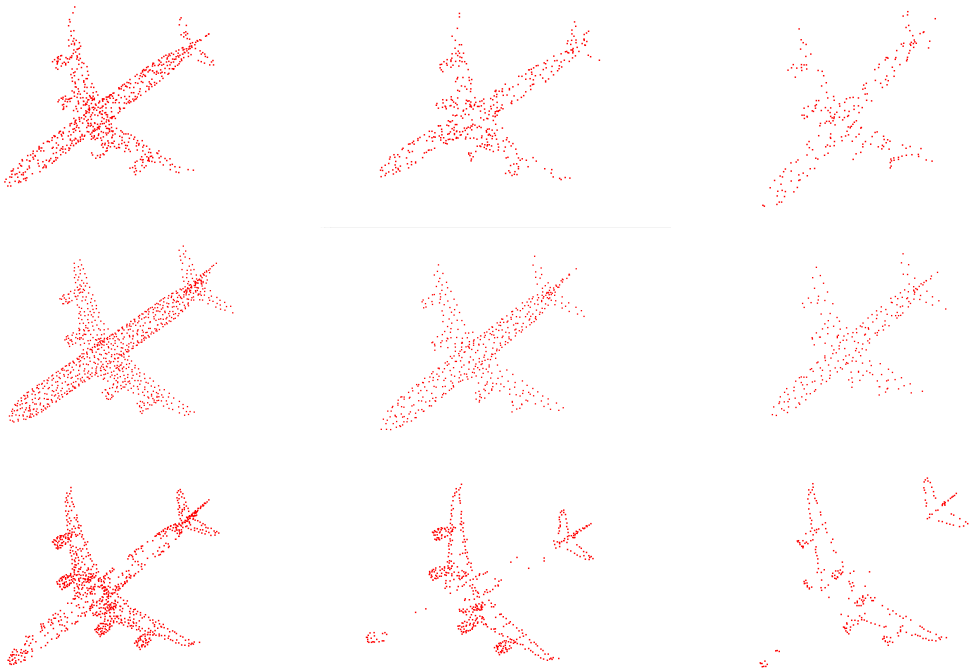
To reduce the computation complexity, we want to compress the point cloud and use a smaller number of points to represent the original one. We preprocess the dataset by using subsampling scheme. As mentioned in Section 3.2, uniform subsampling would be an easy choice. Using the farthest sampling approach guarantees that the sampled points are the most scattered on the point cloud which provides more stable point distribution. Contour-enhancing resampling method proposed by Chen et al. [14] has been demonstrated to be an effective way to preserve better structure information from the original point cloud with fewer points. We implement contour enhance method using high-pass graph filter as 2.19, which the convolution results measure how well each point's signal value can be recovered by its neighbors. Instead of using a re-sampling scheme which is proportional to its ℓ_2 norm of the high pass convolution results, we pick the top m points with the largest local variation directly, and we consider them as key points in the point cloud object. Doing this kind of data preprocessing steps to lower the point number is very beneficial, especially when we want to do real-time classification.

Examples of using the uniform sampling, farthest sampling and high-pass graph filtering to extract the top 1024, 512, 256 key points from the original point cloud are presented in Figure 4.3 and Figure 4.4. As the figures show, with the same number of samples, using farthest sampling preserves more stable point distribution. Doing high-pass graph filter based sub-sampling scheme can achieve more clear contour visually, comparing with other schemes. In our experiment, we will compare the classification results using uniform resampling, farthest sampling and contour-enhanced resampling method to prepare the input point set data.



(a) N =1024 (b) N =512 (c) N =256

Figure 4.3: Uniform sampling and contour-enhanced sampling comparison with point number N = 1024, 512 and 256 (Desk). *Top:* point cloud data prepared by doing uniform sampling. *Middle:* point cloud data prepared by doing farthest sampling. *Bottom:* point cloud data prepared by doing contour-enhanced sampling



(a) N =1024 (b) N =512 (c) N =256

Figure 4.4: Uniform sampling and contour-enhanced sampling comparison with point number N = 1024, 512 and 256 (Airplane). *Top:* point cloud data prepared by doing uniform sampling. *Middle:* point cloud data prepared by doing farthest sampling. *Bottom:* point cloud data prepared by doing contour-enhanced sampling

4.1.3 Network Architecture and Training Details

The basic structure of the model is described in Section 3.3. In this section, we present some details about our model setting which produced the best performance. First, all the point cloud objects are normalized into a unit sphere. The graph support of each object is constructed by 50 nearest neighborhood graph. We present the hyper-parameters setting that produces the best result for our proposed model using global pooling and and multi-resolution separately. (1) For the proposed method with global pooling scheme, the model uses two graph convolutional layers with 1200 and 1000 different sets of filters respectively. One pooling layer follows each convolutional layer con-

taining max-pooling and variance pooling. One fully connected layer with 600 nodes and one output layer are at the end of the network. (2) For the proposed method with multi-resolution pooling scheme, the model uses two graph convolutional layers with 1000 and 1000 different sets of filters each. The first convolutional layer followed by the pooling layer with 52 centroid points and each cluster containing 55 nearest neighbors for every centroid point. The second pooling layer has the graph dimension of 55 nodes. One global-max pooling layer is followed to aggregate the final latent representation of each object. One fully connected layer with 300 nodes and one output layer are at the end of the network to serve as classifier. For both networks, the convolution steps are done with order 3 Chebyshev filter approximation, giving a relatively high order approximation allows the model to design more complex graph filter and incorporate information from a larger receptive field.

The training is done with mini-batch size 28. To add robustness to our model, the input coordinates are perturbed with Gaussian noise $\sim N(0, 0.08)$. During the training process, we use Adam optimizer with initial learning rate 1.2×10^{-3} and the learning rate is divided by 1.7 after every 20 epoch, which allows the network to explore the weight space in a more detailed fashion. To alleviate the unbalance problem of the dataset, weighted gradient descent has been applied to force the model to pay more attention on the under-represent categories. To prevent the model from overfitting, two methods have been used. One is adding dropout after both convolutional layers and fully connected layer with 0.5 dropout rate, and the other one is adding regularization term with $\alpha = 2 \times 10^{-4}$ on the weights to avoid learning complicated model.

4.1.4 Performance Comparison

Previous researchers have approached the 3D object classification problem from different perspectives. A 3D object can be represented as voxelized shapes lying on 3D grid, a combination of 2D rendered images from different views of the object, or a point cloud set. These different representations of 3D objects lead to different approaches to solve the problem. We compare our algorithm with state-of-the-art methods using volume input [17, 24, 30], images input [25] and point sets [26] input respectively.

Since the dataset is a highly unbalanced dataset, we use two metrics: mean instance accuracy and mean category accuracy to evaluate the performance. The mean instance accuracy measures the overall performance

while mean category accuracy measures the average performance of each class. From Table 4.1, we can see that there is still a gap between our method and the state-of-the-arts 3D object classification approach MVCNN [25], which uses ImageNet1K to pre-train and uses an extensive data augmentation scheme. However, we improve the performance in both metrics by a small margin comparing to the state-of-arts point-based 3D object classification method PointNet [26].

A more significant improvement of our model is in the sense of model stability. As shown in Table 4.2 and Figure 4.8, we improve the stability of the model performance by a large margin comparing to PointNet. The fluctuation of the performance during the training is one of the weaknesses in PointNet because its performance is sensitive to the initial weights. In our model, we introduce the structure of the data by providing the interconnection between points and assigning different weights based on their geometric distances. Besides, we explore feature from different scale of influence regions by the K-localized filtering process, which generates more distinctive feature maps between classes. Thus, our proposed algorithm shows better performance stability.

All the performance reported uses farthest sampling to prepare the input point set since it has the best performance among the three different data preparation schemes. The main reason that using farthest sampling perform better than uniform sampling is that it gives more stable point distribution. Using contour-enhanced subsampling scheme does not give us the performance boost as expected. The reason behind it is presented in section 4.1.5. However, the contour enhanced subsampling scheme will give us more robustness of the model, which is presented in section 4.1.7. The value of this pre-feature extraction step will be more obvious when we use fewer point number to represent each object.

We believe by introducing more deeper network properly, we could potentially improve the performance further. However, at the current stage, we have not found an effective way to scale our network structure to more than two graph convolutional layers. One main reason is that although the dataset we use is a widely used 3D benchmark dataset, it is still a relatively small dataset. Applying on larger scale dataset can further test the performance of our proposed model.

It is also worth to mention that our proposed method has impressive empirical performance in the sense of rotation invariance. All the results reported in table 4.1 do not use rotation to augment the input data. When

we randomly rotate the input data during the model training process, the improvement of performance is trivial. Especially, considering that the orientation of the ModelNet40 is not aligned, the methods with volume and image as input typically involves intensive data augmentation such as rotation, mirror and flip to compensate this misalignment in the input data. However, our proposed method performs well even without this data augmentation step. The possible reason is that our graph support is in nature invariance to rotation, and the feature maps learned from the graph convolutional layer are local geometric structure related features. Thus, the learned features are much more robust to data transformation operation such as rotation.

Algorithm	Input format	ModelNet 10 Prediction Accuracy (avg. class)	ModelNet 10 Prediction Accuracy (overall)	ModelNet 40 Prediction Accuracy (avg. class)	ModelNet 40 Prediction Accuracy (overall)
3D ShapeNets[17]	volume (1 view)	83.5%	-	77%	84.7%
VoxNet [24]	volume (12 views)	92%	-	83%	85.9%
SSCN [30]	volume (20 views)	-	-	88.2%	-
MVCNN [25]	image (80 views)	-	-	**90.1**%	-
PointNet*[26]	point (1024 points per object)	91.33%	91.59%	85.48%	88.49%
Proposed algorithm with gloabl pooling	point (1024 points per object)	91.35%	91.87%	85.77%	89.20 %
Proposed algorithm with multi-resolution pooling	point (1024 points per object)	91.57%	91.91%	85.97%	**89.51** %

Table 4.1: Results comparison with state-of-art methods on ModelNet [17].

*Means the result is reproduced by the code provided by the paper author. Otherwise, the baseline results presented here are reported in the original paper. All the results reported above are the average results of 50 trials for each scenario. Only the best model setting is shown. In MVCNN [25], we report the best performance here which used ImageNet1K for pre-train and input data from multiple rotations or views.

	Mean instance accuracy std.	Mean class accuracy std.
PointNet	0.3773%	0.4007%
Proposed method (global pooling)	0.2563%	**0.3056%**
Proposed method (multi-resolution pooling)	**0.1940%**	0.3127%

Table 4.2: Standard deviation of the test set accuracy with 50 trails.

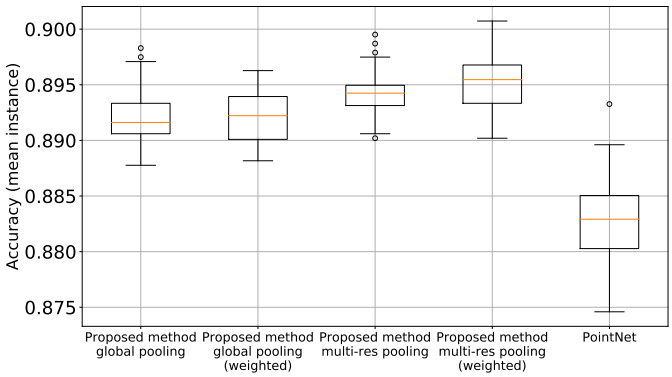


Figure 4.5: Detailed comparison with PointNet and our proposed methods (with and without weighting scheme)

We also compare the learning complexity among different approaches. The results are shown in Table 4.3. As a point set based method, our proposed method does reduce the learnable parameter number by a huge amount comparing with volume and image based methods. However, by introducing the pairwise relation between points into the training process, we do need more learnable parameters comparing PointNet if we do not reduce the graph resolution in the middle step. We trade more learning complexity with high graph resolution. With multi-resolution pooling, we reduce the learnable parameter number by half and achieve better performance at the same time. How to further reduce the learning complexity and at the meantime maintaining high performance will leave it to future work.

	# of learnable parameters
Proposed algorithm (global pooling)	5.44 M
Proposed algorithm (multi-resolution pooling)	**3.3 M**
PointNet [26]	3.5 M
Subvolume [29]	16.6 M
MVCNN [25]	60.0 M

Table 4.3: **Learning complexity**.

4.1.5 Learning Convergence Analysis

The test set loss trained from 5 different trials (with different model initial weights) during the model learning process for PointNet [29] and our proposed models is shown in Figure 4.6. The test set prediction accuracy from the model learning process is shown in Figure 4.7 as well. Generally speaking, our proposed model has better performance in both overall accuracy and mean class accuracy comparing to PointNet. The improvement in overall accuracy is more significant. At the same time, our proposed models take less iterations to converge and has less fluctuation during training than the PointNet. The possible reason could be that we have the K-hops localized filtering scheme to explore more salient features in different abstraction level. Besides, we add the prior assumption that the influence between point is decided by their spatial distances. By incorporating this geometric assumption, the training process is more effective and it gives us the faster convergence rate. As for PointNet [26], only global features are explored by spatial transformation network and symmetric functions, the local structure information does not incorporate into their learned features.

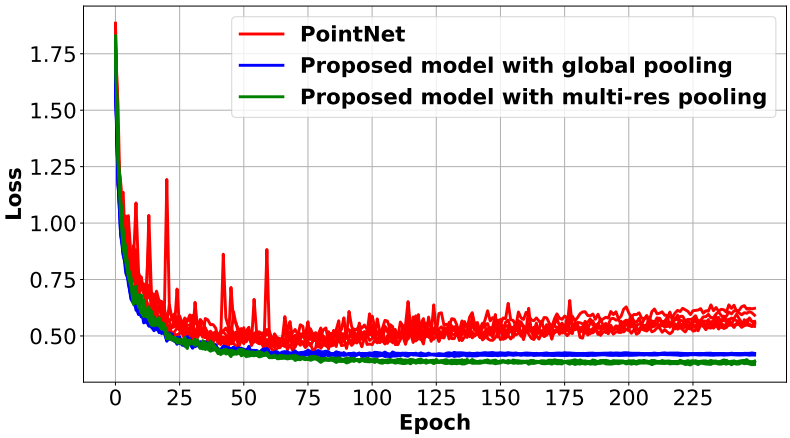


Figure 4.6: Test set loss comparison using PointNet and our proposed methods on the ModelNet40 dataset

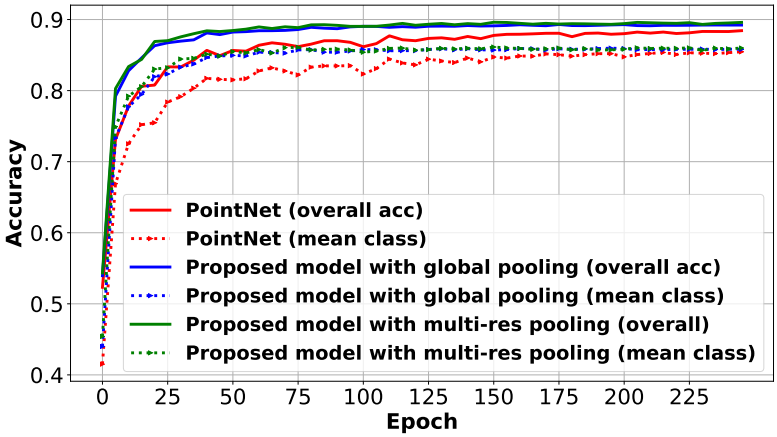


Figure 4.7: Test set accuracy comparison using PointNet and our proposed on the ModelNet40 dataset

We also compare the stability of the model performance for both our proposed model and PointNet. We record the results from 50 trials of experiments after the model converges, trained from 5 model initial weights and 10 trials each. The result is shown in Figure 4.8. It shows that comparing to

PointNet, our model produces more consistent results with different initial states and less performance fluctuation after the model converges.

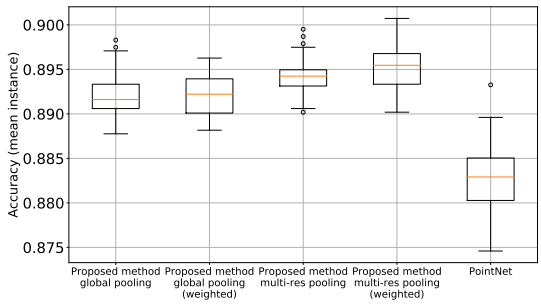


Figure 4.8: Detailed comparison with PointNet and our proposed methods (with and without weighting scheme)

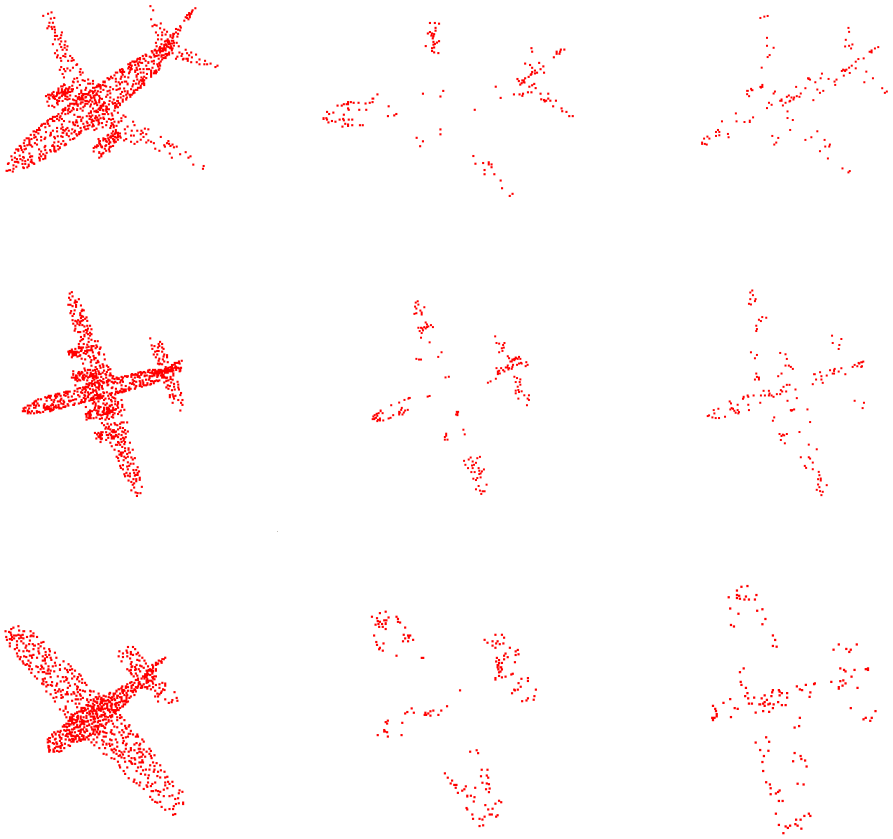
4.1.6 Visualization of the Max-pooling Contribution Point

To uncover what features are learned during the training process, we did some data visualization work to help us understand. One of the key operation in our proposed algorithm is the global operation max pooling, which aims at picking the unique pattern points and summarizing the global signature of each object. We call the points that have the maximum values among each feature map the active point, which are the points that actually contribute to the max-pooling process. We visualize these active points for feature maps coming from both first and second graph convolutional layer. The visualization of the max-pooling contributing points from airplane, guitar and person category are shown in Figure 4.9, 4.10 and 4.11. We can see that the point clouds from the same class have a very similar active point set for both first and second graph convolutional layer, and at the same time the points that are not discriminate among categories are eliminated. Besides, another observation we can make is that those two graph convolutional layers learn point features from different abstraction levels, which is consistent with our assumption. Since we uses this K-localized feature learning scheme, we forces the model to learn features at different influence region. The second graph convolutional layer have a larger receptive field which can incorporate the pairwise point interaction from $2K$-hops region. Thus, the first graph

convolutional active points more incline to explore local cluster, and the second graph convolutional active points start to pick up global skeleton of the object. Thus, with the global features from different region scales, we can gather more robust and abundant features.

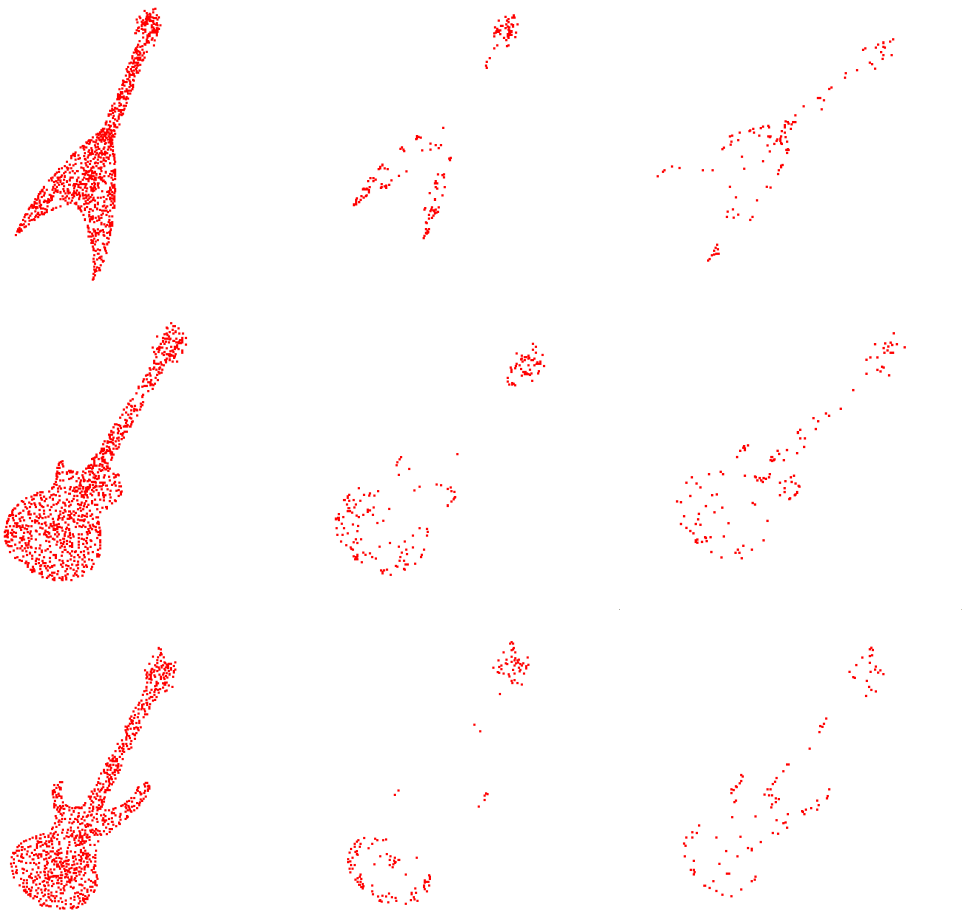
In PointNet [26], the similar conclusion is drawn that the global 1-max-pooling operation has the ability to summarize the distinctive point set from each category, but their feature exploration is limited only to global features.

One of the interesting things we found out in the experiment is that when we use high pass filtering subsampling scheme instead of the uniform subsampling one to prepare the input data, the performance decreases slightly on the classification dataset, which is a bit unexpected. Contour enhance subsampling, in general, preserves better structural information than uniform subsampling. However, if we look at the active points from the second graph convolutional layer, the active points are close to the contour point of the object. Thus, the model learns the contour and key points information itself without explicitly feeding it into the network. The selected key points are optimized for the classification task without any presumption about the importance of each point, which explains why using the uniform and farthest preprocessing scheme to prepare the input data outperform slightly than the contour enhanced subsampling scheme.



(a) Original point cloud (b) Layer one G-Conv active points (c) Layer two G-Conv active points

Figure 4.9: Max pooling contribution points from different layers (Airplane)



(a) Original point cloud (b) Layer one G-Conv active points (c) Layer two G-Conv active points

Figure 4.10: Max pooling contribution points from different layers (Guitar)

(a) Original point cloud (b) Layer one G-Conv active points (c) Layer two G-Conv active points

Figure 4.11: Max pooling contribution points from different layers (Person)

4.1.7 Advantages of the Proposed Method

In Section 2.4, we mentioned three difficulties when dealing with the point set data. In this section, we will elaborate on why our proposed model is an effective approach to deal with point cloud classification problem.

First, by providing the graph support explicitly and assigning different weights between points based on their geometric distances, we directly incorporate the geometric connectivity information between points into the proposed model. Our training process is more effectively and has better

convergence rate shown by the model learning curve in Section 4.1.6.

Second, one of the main problem when dealing with point set data is how to achieve order invariance. By providing the graph support, the local structure context of each point will maintain the same under the permutation of the input order. Thus, the graph convolution step guarantees the point order invariance. As for the pooling step, the pooling process is done among the point-wise latent representation among the whole point set, which also guarantees the order invariance requirement.

Third, because by nature, the graph structure is invariance to rotation, the relative distance between points will maintain the same under rotation transformation. Empirical result shows that our learned features are robust to the dataset which is not aligned, even without data augmentation by rotation.

At last, our proposed method incorporates feature information at different scales of the geometric neighbor information by using the K-localized filtering scheme. Adding every graph convolutional layer will enlarge the influence region by K-hops. By combining features at different influence region scale, we capture point cloud features at different abstraction level which guarantees the exploration of both local features and global features. PointNet [29], which approaches this problem by symmetric function and spatial transform network, only explores global point cloud feature and lacks the exploration of local structure information. Thus, our proposed graph feature extraction scheme enables to learn more distinctive latent representations among different classes, which lead to better model stability.

4.1.8 The Impact of Hyper-parameters

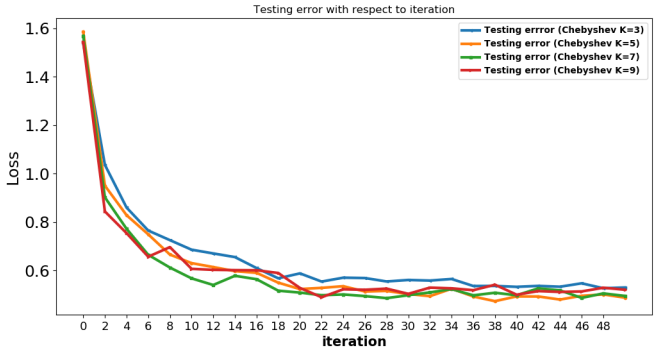
In this section, we explore the impact of hyper-parameters in our proposed architecture including the Chebyshev polynomial order, the number of filters, the number of convolutional layers, different data preparation schemes, different pooling operations, different weighting schemes. All the results listed here without special notice use the proposed model with global pooling scheme, the same trend can be expected for the proposed model with multi-resolution pooling scheme as well.

Chebyshev polynomial order The impact of the Chebyshev polynomial order on the prediction accuracy and prediction error in the convolutional layer is shown in Figure 4.12a and 4.12b. In the graph convolution layer, K controls the flexibility of your learned filter. With larger K, you can

learn more complicated graph filter. Besides, the Chebyshev polynomial order K also decides how many hops away nodes will affect each node's convolution process since the Chebyshev filtering approximation process is K-localized in the spatial domain. In the classical CNNs, the receptive field is decided by the filter size. In our proposed model, the receptive field is decided by the Chebyshev polynomial order K. In this experiment, the initial graph is set up by 15-nearest neighbor graph. As shown in the figure, we can see that the prediction accuracy improves as well as the learning convergence get faster with the increase of the Chebyshev polynomial order. The possible reason is that the learned feature maps after the convolution step can incorporate its neighbor information from a larger region with the increase of K. However, when we further increase our impact region to 9 hops, the improvement saturates. Thus, to consider enough range of neighbor information and enough learning ability of the filter, and at the same time maintain the efficiency of our model. We choose $K = 7$ when the k-nearest neighbor graph is set up by using $k = 15$.

Since both the Chebyshev polynomial order and the set up of your graph support decide how large of the influence region of each point is going to be, there is a trade off between those two factors. As long as the scale of the influence region of each point maintains the same, the model will produce similar performance.

(a) Testing accuracy with respect to training epoch



(b) Testing error with respect to training epoch

Figure 4.12: Prediction accuracy/error with respect to training epoch

The impact of different pooling operation We proposed two different pooling operations in our algorithm. One is using only global pooling without doing multi-resolution of the graph structure. The other one uses cluster max-pooling, which is similar to the graph coarsening process. For the global pooling approach, we only aim to pick the global features. We propose to simply use the max-pooling, and variance-pooling to aggregate feature information on the learned feature map after the convolution step. The result shown in Table 4.4 is the comparison between three different pooling schemes under the two layer graph convolutional network structure. As shown in the table, with only 1-max-pooling layer we can already have competitive performance. With the extra variance information, we further improve the overall and mean class accuracy by 0.17% and 0.07% respectively. However,

when we further incorporate the average pooling features, it seems to incur more confusion.

Table 4.4: Comparison between different pooling operations

Pooling operation	Max-pooling	Max/Var-pooling	Max/Avg/Var-pooling
Mean class acc	85.27%	**85.40%**	84.87%
Overall acc	89.19%	**89.22%**	88.89%

Filter number The filter number controls the abundance of feature maps. Thus, we investigate the impact of the filter number on our model's performance. The experiment is done with one graph convolutional layer using different filter size from 200 to 3000. The same trend can be expected for two convolutional layer model. The result is shown in Figure 4.13. We can see that the model performance enhances with the increase of filter number and tends to be stable at last. The possible reason could be that when we keep increasing the feature number, it will tend to learn repetitive features at some point. Because feature from the same layer have the same level of abstraction, and simply increase the filter number from the same layer have limited power of latent representation. Thus, to further increase our model performance, we need to introduce deeper network structure, which can gather features at various levels of abstraction instead stacking similar features from the same layer.

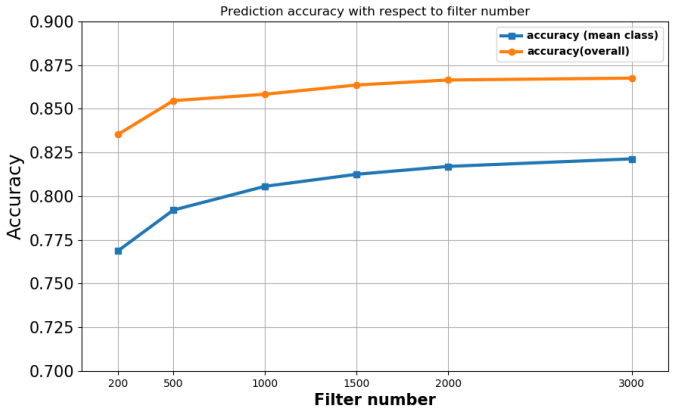


Figure 4.13: Prediction accuracy with respect to filter number

Layer number The advantage with deeper structure is that we can capture highly nonlinear relationships between nodes. Besides, stacking more graph convolutional layer by nature also enlarges the influence neighbor region of each node since each latent representation (feature map) is calculated by a K-localized filter. Thus, feeding those feature map into another convolutional block will once again enlarge the influence region by K hops. Besides, as we discussed above, more graph convolutional layer means we can learn features at various levels of abstraction, which is one of the reasons why deeper classical CNNs usually works well in image classification task. Thus, we argue that in the graph convolutional layer we have the similar trend, which is multiple layers are better at generalizing because they learn all the intermediate features between the raw node features and the high-level classification output.

In our experiment, we can obviously see a performance boost when adding a second convolution layer, shown in Figure 4.5. However, adding the third convolution layer does not give us the further performance enhancement as we expected. There are several possible reasons. First, although the dataset (ModelNet) we use is a widely used 3D data classification benchmark dataset, it is still a relatively small scale dataset. The effect of deeper network could be more obvious when we can apply larger scale dataset. Second, the fact that the increase of the receptive field does not necessarily guarantee the improvement of performance which could blur the features and jeopardize the performance. At last, the gap between training error and testing error becomes bigger with the increase of the layer number, which means the overfitting problem becomes severe. This could be another factor that causes the inferior performance of more deeper layer network structure.

Layer number	one layer	two layers	three layers
Overall acc	86.65%	**89.22%**	88.92%
Mean class acc	82.44%	**85.40%**	84.95%

Table 4.5: The impact of layer number

The impact of different weighting scheme As we mentioned above, the model suffers from overfitting to the categories with more training instances because of the unbalanced nature of the ModelNet dataset. We attempted to implement weighted gradient descent in our model, which we impose a greater cost on the model for making classification mistakes on the

minority class during training. These extra penalties can bias the model to pay more attention to the under-represent categories. The loss function for the weighted gradient descent is defined as:

$$\text{Loss} = \sum_{i=0}^{n-1} c_i \sum_{k=0}^{K-1} -y_i^{(k)} log(\hat{y}_i^{(k)}), \tag{4.1}$$

where $\hat{y}_i$ is the prediction class, y_i is the true label, and c_i is the weight associated with each of the samples, which will be related to the proportion of each class p_i, specifically, $c_i = \alpha q_i + 1$ and q_i is the normalized value of the inverse of q_i. The result of applying weighted gradient descent is shown in the Table 4.6. The result shows that after adding more penalty on the categories with more samples, the mean class accuracy improves by 0.37% and 0.51% for our proposed model using global pooling and multi-resolution pooling respectively, which alleviate the problem of overfitting to categories with more instances.

	mean instance accuracy	mean class accuracy
proposed method (global pooling) without weighting scheme	89.22%	85.40%
proposed method (global pooling) with weighting scheme	89.20%	85.77%
proposed method (multi-res pooling) without weighting scheme	89.41%	85.26%
proposed method (multi-res pooling) with weighting scheme	89.51%	85.97%

Table 4.6: The effect of the weighting scheme.

4.1.9 Robustness Test

In real world point cloud data, one might exist various kinds of artifacts. To evaluate our model's robustness to noise, we introduce Gaussian noise with different standard deviations to simulate different noise level. We randomly

add Gaussian noise with zero mean and standard deviation 0.02, 0.04, 0.05, 0.06 and 0.08 into the testing set data in both data prepared with contour enhancing prepossessing, uniform sampling for our proposed model as well as PointNet [26]. The result is shown in Figure 4.14. Generally speaking, our model is robust to additive Gaussian noise. As we mentioned in the results analysis above, using the contour enhanced preprocessing methods on the input point cloud does not give us an improvement in the performance. However, as shown in our robustness test, when introducing Gaussian noise $N \sim (0, 0.04)$, the accuracy drops by 9.97%, 7.89% and 9.12% using the proposed method with uniform sampling, proposed method with contour-enhancing sampling and PointNet respectively. The high-pass graph filter subsampling scheme does lead to better resistance against data corruption, especially in the high noise perturbation case. The possible reason behind this is that with the same amount of subsampling point number, the contour enhancing subsampling contains more key points than the uniform subsampling scheme so that it increases the robustness of the model.

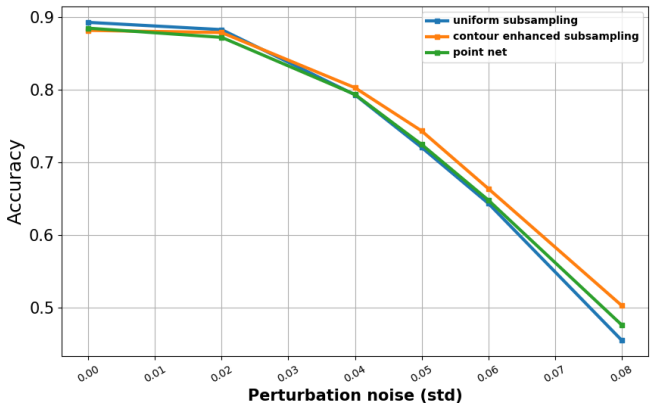


Figure 4.14: Testing set accuracy with respect to noise perturbation (ModelNet 40)

4.1.10 Characteristic of the Wrong Class

The first thing we could notice from the confusion matrix for both ModelNet10 and ModelNet40 dataset shown in Figure 4.15,4.16 is that the categories with low accuracy are the categories which have relatively less training

data. The class with less training samples can not cover enough variety and cause the inferior performance. For example, in ModelNet40, the cup category has the least sample number and its prediction class is dominated by other similar class with much more sample data such as the vase class. A similar case goes to flower pot class as well, where the model misclassifies flower pot as plant and vase which have relatively similar geometric structure. Second, there are some misclassifications happens due to the structural resemblance of the object from different categories. For example, in ModelNet 10, most of the confusion coming from dresser and night stand, desk and table, which makes a lot of sense since the structural differences between those classes are subtle. In some case, it is even hard to distinguish those point cloud objects by humans if we visualize those objects.

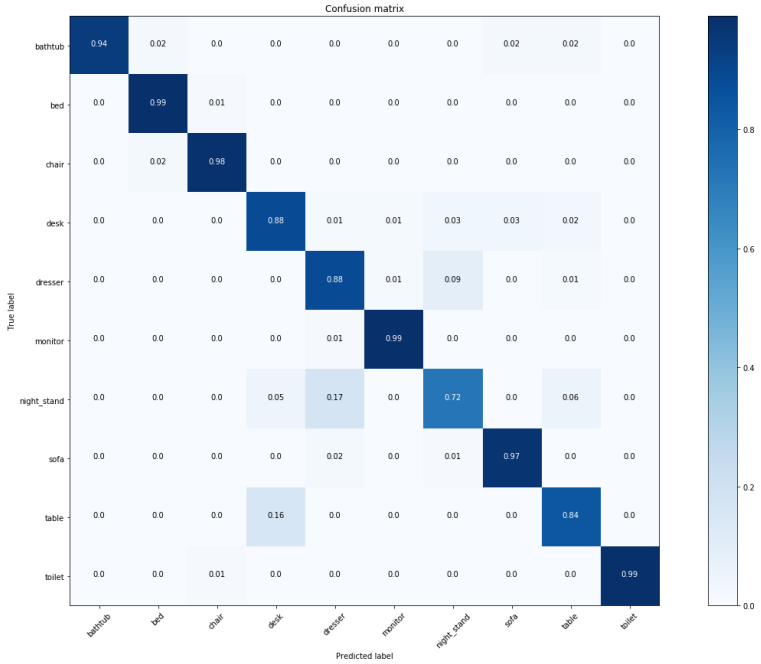


Figure 4.15: confusion matrix for ModelNet10 (overall accuracy: 91.85%, avg. class accuracy: 91.35%)

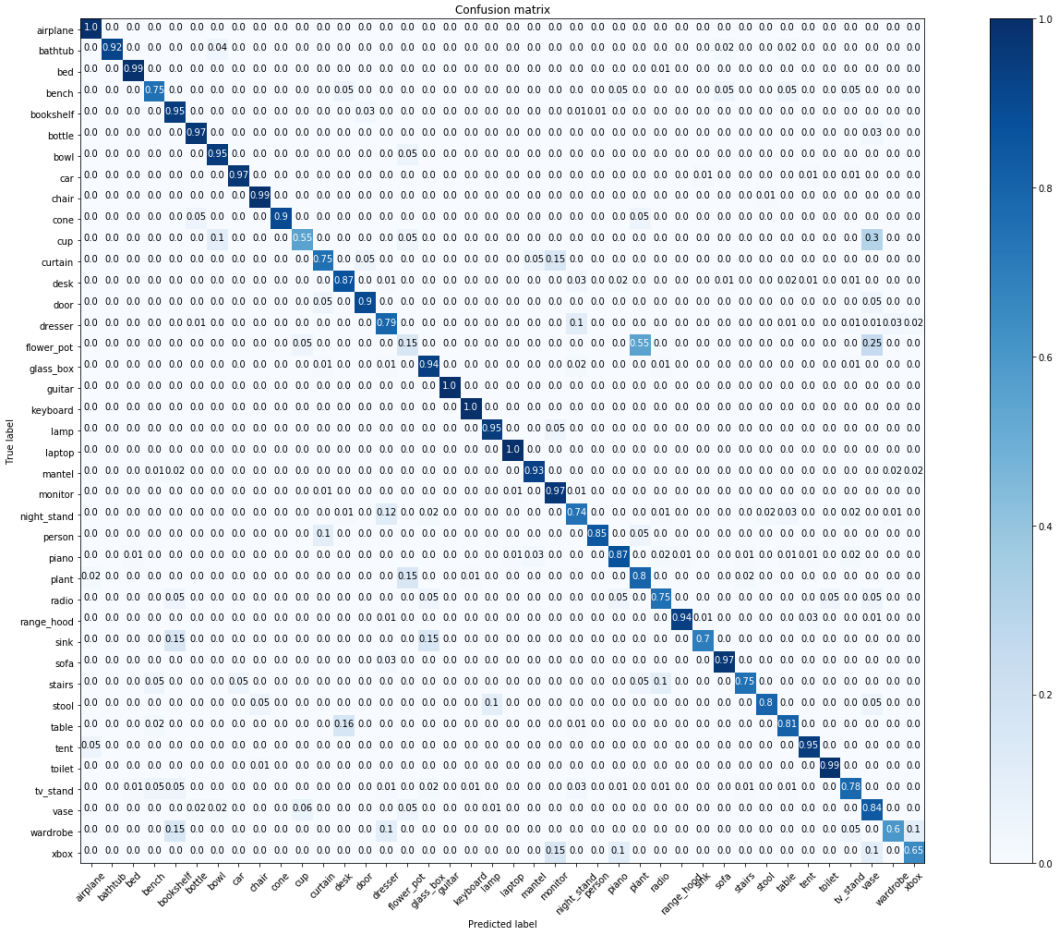


Figure 4.16: confusion matrix for ModelNet40 (overall accuracy: 89.28%, avg. class accuracy: 85.54%)

Chapter 5

Conclusion

5.1 Summary of Contributions

In our project, we explore by far the most effective framework that explores feature maps on graphs, namely the Graph Convolutional Neural Networks. We applied this method to a novel application which allows the 3D object classification to directly operate on the irregular structured data format, the 3D point cloud format. We designed a Graph-CNN based algorithm that specifically targets the 3D point cloud classification problem, which is demonstrated to be competitive on the 3D classification benchmark dataset. By leveraging geometric information encoded in the graph structure, the proposed model can explore local structure from different receptive fields, which enables that the learned latent representation of each object class is more distinctive from the others. The experiment demonstrated that the proposed model converges faster and has relatively more stable performance than the competing schemes.

5.2 Future Work

Point cloud part segmentation In our project, we explored the 3D point cloud object classification problem. There is another type of classification problem on point cloud called part segmentation, which is a point-wise classification problem. Part segmentation is extremely meaningful for a lot of application such as motion tracking. The main difference between point cloud classification and part segmentation is that in part segmentation problem,

we might need both global signature of each object and latent representation of each point to achieve point-wise classification. As an efficient way to learn the point-wise latent representation, we believe the proposed method has potential in part segmentation problem as well.

Extend to input data without graph signal Our proposed method also requires the presence of both graph signal and graph structure. We could explore the possibility of our model to deal with the case when we only have the graph structure information without the local feature information. For example, the chemical component classification problem [31]. One possible solution could be instead of explicit feeding the input graph signal of each point we place a bias signal '1' associated with each node. Instead of doing the convolution with the real graph signal we do the convolution with this proxy bias graph signal.

Explore more tools in Graph signal processing With the fast development of the Graph signal processing field, we could improve our model with more tools from GSP to design more powerful graph filter. One recent work proposed a new class of complex rational Cayley filter [32] which demonstrated the superior performance than Chebyshev graph filter [9]. Thus, to incorporate more graph information or the interaction between the graph signals and graph structure, we need more exploration from the theoretical point of view of GSP.

Bibliography

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] Yoon Kim, "Convolutional neural networks for sentence classification," in *Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, 2014.

[3] Ye Zhang and Byron Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *arXiv preprint arXiv:1510.03820*, 2015.

[4] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Int. Conf. on Machine Learning (ICML)*, Washington D.C., United States, 2003.

[5] Alexander J. Smola and Risi Kondor, "Kernels and regularization on graphs," in *COLT*, 2003, vol. 2777 of *Lecture Notes in Computer Science*, pp. 144–158.

[6] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, 2013.

[7] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun, "Spectral networks and locally connected networks on graphs," in *Int. Conf. Learning Representations (ICLR)*, Banff, Canada, 2014.

[8] M. Henaff, J. Bruna, and Y. LeCun, "Deep Convolutional Networks on Graph-Structured Data," *arXiv preprint arXiv:1506.05163*, 2015.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems (NIPS)*, Barcelona, Spain, 2016.

[10] James Atwood and Don Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, Barcelona, Spain, 2016.

[11] Thomas N. Kipf and Max Welling, "Semi-supervised classification with graph convolutional networks," in *Int. Conf. Learning Representations (ICLR)*, Toulon, France, 2017.

[12] Rianne van den Berg, Thomas N. Kipf, and Max Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[13] Siheng Chen, Dong Tian, Chen Feng, Anthony Vetro, and Jelena Kovačević, "Contour-enhanced resampling of 3d point clouds via graphs," in *Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, United States, 2017.

[14] Siheng Chen, Dong Tian, Chen Feng, Anthony Vetro, and Jelena Kovacevic, "Fast resampling of 3d point clouds via graphs," *arXiv preprint arXiv:1702.06397*, 2017.

[15] Francois Lozes, Abderrahim Elmoataz, and Olivier Lezoray, "Pde-based graph signal processing for 3-d color point clouds : Opportunities for cultural herihe arts and found promising," *IEEE Signal Process. Mag.*, vol. 32, no. 4, pp. 103–111, 2015.

[16] Dorina Thanou, Philip A. Chou, and Pascal Frossard, "Graph-based compression of dynamic 3d point cloud sequences," *IEEE Trans. Image Processing*, vol. 25, no. 4, pp. 1765–1778, 2016.

[17] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[18] David S. Watkins, *The matrix eigenvalue problem - GR and Krylov subspace methods*, Society for Industrial and Applied Mathematics (SIAM), Pullman, Washington, 2007.

[19] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[20] D. I Shuman, P. Vandergheynst, and P. Frossard, "Chebyshev polynomial approximation for distributed signal processing," in *Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, Barcelona, Spain, June 2011.

[21] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis, "Weighted graph cuts without eigenvectors A multilevel approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 11, pp. 1944–1957, 2007.

[22] Ulrike von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[23] George Karypis and Vipin Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[24] Daniel Maturana and Sebastian Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *Int. Conf. on Intelligent Robots (IROS)*, Hamburg, Germany, 2015.

[25] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Int. Conf. on Computer Vision (ICCV)*, Santiago, Chile, 2015.

[26] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*, Hawaii, United States, 2017.

[27] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems (NIPS)*, Montreal, Canada, 2015.

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

[29] Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, United States, 2016.

[30] Benjamin Graham and Laurens van der Maaten, "Submanifold sparse convolutional networks," *arXiv preprint arXiv:1706.01307*, 2017.

[31] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.

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