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
October 20, 2017

Contents

Abstract

Graph Convolutional Neural Networks (Graph-CNNs) are an extension of the traditional Convolutional Neural Networks (CNNs), which are designed to handle non-Euclidean domain such as data from social networks, transportation networks, molecules, etc. Recently, researchers have proposed various of Graph-CNNs algorithms that can achieve state of the art performance on structured data related tasks, especially the node classification problems, from both graph spatial and spectral perspectives. Besides, as shown by previous researchers, spectral Graph-CNNs can also achieve competitive performance in image classification problem, which can be regarded as a sanity check for Graph-CNNs algorithms, since images can be seen as a grid graph. However, the image data doesn't benefit from the underlying graph structure of data from different classes because of its regular grid structure. In our project, we explore the use case of the Graph-CNNs on 3D point cloud data sampled from a manifold where the data naturally lies on various graph structures. We designed effective network structure which can achieve competitive performance in point cloud classification problem using the 3D object classification benchmark dataset ModelNet, which demonstrates the effectiveness of Graph-CNNs on not only the node classification, homogeneous graph related classification task but also the heterogeneous graph related classification problem.

Chapter 1

Introduction

Convolutional Neural Networks (CNNs) [1] have been extremely successful in the task where the data has a regular structure. For example, CNNs and its variants break all records in 2D image recognition, segmentation, and detection task. And recent studies [2, 3] have also demonstrated the effectiveness of CNNs on sentence classification task in Natural Language Processing domain. In this use case, the language data can be regarded as 1D sequential data. However, there are cases where the data has meaningful information encoded in their structure which does not lie in the Euclidean domain. For example, in the case of the social network, we would love to make some inference based on their social relationship. 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 model has limited learning capacity and flexibility, since this is a relatively shallow model and 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 been investigating on how to generalize the traditional CNNs to the general graph structured data, which can maintain the powerful latent feature learning nature of CNNs and effectively incorporate information from node features and its graph structure. 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 convolution definition in graph spectral domain to construct their convolution block and using the graph multi-resolution clustering approach to achieve pooling steps in their network. Defferrard et al.[9] further proposed using fast localized convolution operation by introducing recursive format of the Chebyshev polynomials which not only avoids explicit calculating the Fourier graph basis but also enables the learnable parameter number for each graph filter to be independent of the input node number. The approaches we mentioned above can be categorized as the spectral approach since the convolution operation is defined in the graph spectral domain. Another branch of approach is called spatial Graph-CNNs such as DCNNs [10], which use 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 image classification task with decent performance and demonstrated the effectiveness of using Graph-CNNs to explores the similarity between features by setting 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. The simplified model is showed to be an efficient way to solve node prediction problem in the social network, citation network. Berg et al. [12] has introduced 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 focus on when the graph structure is homogeneous. It means the graph feature maps extracted from the convolutional process is based on one single graph structure, whether it's one huge knowledge graph, citation graph or the same 2D grid graph on every image or the same feature graph on every instance. In our project, we have explored the extension use case of the spectral defined Graph-CNNs approach from vertex level prediction or uniform grid graph prediction to heterogeneous graph related prediction task. We aim to design graph filters that can be adaptive to multiple graph structures. Our study is mainly focusing on applying the Graph-CNNs method on 3D point cloud data.

3D sensing has been a hot topic with the fast development of self-driving car technology. LiDAR (short for Light Detection And Ranging) sensor has shown promising prospects because of its 360 degrees of visibility and large sensing range. The data gathered from LiDAR sensor is called 3D point clouds data, which has become an important way to represent the 3D data. Though 3D point cloud data usually is an unordered point set, the spatial distribution of the point cloud data encodes the underlying structure of the points set. Thus, 3D point cloud processing can be formulated as the graph signal processing problem. The effectiveness of the graph signal processing in 3D point cloud data processing has shown in the applications of point cloud data visualization, denoising, inpainting and compression [13, 14, 15]. In our project, we have explored the possibility of using Graph-CNNs to achieve 3D point cloud classification and designed a Graph-CNNs based deep learning framework which has shown promising results in 3D object classification benchmark dataset ModelNet [16].

The main contributions of this project include:

- Detailed review of state of the art Graph Convolutional Neural Networks algorithms and investigate the similarity and difference between them.
- Implemented ChebyNet [9], GCN [11] and DCNN [10] , three most representative Graph Convolutional Neural Networks approaches.
- Explored the use case of Graph Convolutional Neural Networks in point cloud data and propose a Graph-CNNs framework that could reach the state of art performance in point-based 3D object classification problem.
- Visualized the Graph-CNNs learning process in 3D point cloud data, which demonstrated Graph-CNNs learn the latent global signature of each point cloud object by summarizing a unique subset points for each class from the original point cloud at different receptive field scales.

This report is organized as follow: Chapter 2 elaborates on the background knowledge and related work of graph signal processing and Graph-CNNs. Chapter 3 describes the proposed model in Graph-CNNs based 3D point cloud classification problem and discusses its advantages. Chapter 4 concentrates on the experiments details of the proposed model and its comparison with the other state of art methods for 3D object classification in deep learning. Besides, 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

Before the emerging of graph signal processing (GSP) field, there has been lacking theoretically study of the interaction between graph and graph signal. Instead, most of the prior researchers have been concentrated on the spectral graph theory which only focuses on the graph structure itself. 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. Graph Laplacian matrix is defined as $L = D - W \in \mathbf{R}^{n\times n}$; and the normalization of the Laplacian is defined as $L_n = I_n - D^{-1/2}WD^{-1/2}$ where $W \in \mathbf{R}^{n\times n}$ is the (weighted) Adjacency matrix and $D \in \mathbf{R}^{n\times n}$ is the diagonal degree matrix. For simplicity, the graph we considered is undirected graph in this report. Because for undirected graph, L is symmetric it has a real-valued eigendecomposition $L = U\Lambda U^T$. We call the eigenvectors U the graph Fourier basis and the eigenvalues $\Lambda = ([\lambda_0, \dots, \lambda_{n-1}])$ the graph frequency components, where $0 = \lambda_0 < \lambda_1 < \cdots < \lambda_{n-1} < 2$.

The graph Fourier transform and its inverse of input signal $x \in \mathbf{R}^n$ are defined as following:

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

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

where u_k and u'_k are the kth column of U and U^T.

Graph spectral filtering. The generalization of the fundamental operations such as filtering, convolution are also worth mentioning here, which has been constantly used in our project. In classical signal processing, the filtering process is defined as amplifying or attenuating certain frequency components. In graph signal processing, similar definition is proposed where graph filtering is defined as the manipulation of the graph frequency component Λ. There are two ways to define graph filtering on both spectral and vertex domain, which are also called non-parametric graph filtering and parametric filtering respectively [6, 9]. The final form of spectral convolution process between graph filter and the input graph signal $x \in \mathbf{R}^n$ is shown as follows:

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

where y is the filtered signal and $\hat{h}_f(L)$ is the graph filter. In graph spectral domain, the filter $\hat{h}_f(L)$ is defined as:

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

Polynomial parameterization for localized filters. In vertex domain, the filter $\hat{h}_f(L)$ is defined as:

$$\hat{h}_f(L) = U(\sum_{k=0}^{K-1} \theta_k \Lambda^k) U^T \tag{2.5}$$

where θ is a vector of polynomial coefficients and K is the polynomial order. One of the advantages of the vertex filtering 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.

Chebyshev polynomial filtering approximation. Parametric filtering scheme guarantees the graph convolution process to be localized in space. However, when we encounter large scale graph, the requirement of getting the Fourier basis and graph frequency components will introduce large computational burden. Eigendecomposition on the Laplacian matrix to get the Fourier basis is $O(N^3)$ computational complexity when using the basic QR decomposition method [17]. To overcome these problems, [18] proposed to use the recursive format of the Chebyshev polynomial to approximate the convolution process. For $y \in [-1, 1]$, the Chebyshev polynomials $T_k(y)_{k=0,1,2,...}$ are generate by:

$$T_k(y) = \begin{cases} 1 & \text{if } k = 0 > 0 \\ y & \text{if } k = 1 > 0 \\ 2yT_{k-1}(y) - T_{k-2}(y) & \text{if } k = 2 \end{cases} \tag{2.6}$$

The Chebyshev polynomials form an orthogonal basis for $L^2([-1, 1], \frac{dy}{\sqrt{1-y^2}})$. Every $h \in L^2([-1, 1], \frac{dy}{\sqrt{1-y^2}})$ has a uniformly convergent Chebyshev series[18]:

$$h(y) = \frac{1}{2}c_o + \sum_{k=1}^{\infty} c_k T_k(y) \tag{2.7}$$

In the case of calculating h(x) for $x \in [0, \lambda_{n-1}]$, it can be achieved by shifting the domain using the transformation $x = a(y + 1)$ with $a = \frac{\lambda_{n-1}}{2}$. And we can denote the shifted Chebyshev polynomials $\bar{T}_k(x) = T_k(\frac{x-a}{a})$. The Chebyshev polynomials approximation is done by truncating the expansion term in equation 2.7 by its first K terms. Then Chebyshev polynomial approximation with domain shift can be written as:

$$h(x) = \frac{1}{2}c_0 + \sum_{k=1}^{K-1} c_k \bar{T}_k(x) \tag{2.8}$$

where

$$c_k := \frac{2}{\pi} \int_0^{\pi} cos(k\theta) h(\frac{\lambda_{n-1}}{2}(cos(\theta) + 1))d\theta \tag{2.9}$$

Thus, for input graph signal $x \in \mathbf{R}^n$, we have:

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

With the introduction of the recursive format of Chebyshev polynomial filter approximation process, we don't need to do the eigendecomposition explicitly and we avoid dense matrix multiplication with the Fourier basis. Previous researchers have investigated the usage of the Chebyshev graph filtering in distributed signal processing [19], Graph-CNNs [9] and etc. 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 colors, textures, etc. The examples of 3D point cloud data are shown in 2.1. Figure Previous researchers have formulated the point cloud data processing as graph signal processing problem and proved the effectiveness and efficiency in [13, 14, 15]. The advantages of using GSP tools to solve point cloud related problems lies in two folds. First, the graph structure of the point set encodes valuable geometric information. Both local and global features of the point set are well captured. Second, GSP provides theoretical ground for various generalization operations of classical signal processing operations which guarantee the flexible interaction between graph structure and graph signal.

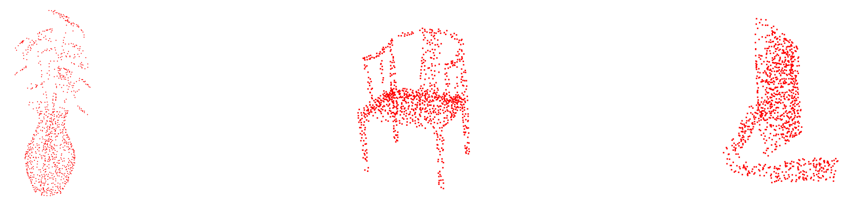

(a) Flower pot (b) Chair (c) Monitor

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 the k-nearest neighbor graph directly, and the other is set up this connection based on a radius threshold ρ. And edge weights are determined by the Gaussian kernel function on the $L2$ norm of the Euclidean distance between points:

$$W_{i,j} = \begin{cases} e^{\frac{-\|x_i^{(c)} - x_j^{(c)}\|^2}{\sigma^2}} & \text{if } j \in \mathcal{A}^K(i) \quad \text{or} \quad \|x_i^{(c)} - x_j^{(c)}\|^2 < \rho \\ 0 & \text{others} \end{cases} \tag{2.11}$$

where $\mathcal{A}^K(i)$ is the K nearest neighbors of $\mathcal{V}_i$ and ρ is the radius threshold of the affecting neighbor distance. In our experiments, we discovered using the fixed K nearest neighbor to set up the weight matrix works better than fix radius graph empirically in the point cloud classification task. Part of our project is inspired by the work proposed by Chen et al. [13]. Besides, one of our point cloud data preprocessing step is based on their proposed contour extraction method. We will elaborate the details in the following.

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, under 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. The 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. [13] 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, and we should also guarantee the resample distribution to be shift-invariant, rotation-invariant and scale-invariant. In most of the object classification or data visualization tasks, we don't have point cloud attributes other than the coordinate information of each point. Then, in this case, the input graph signal is just the coordinate signal from the x,y,z direction.

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 [13], they designed a high pass Haar 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 that can break the trend in its neighbor. As Chen et al. have proved in the paper, the optimal sampling strategies π is proportional to the magnitude of the output features after the graph transformation.

$$\pi_i^\star \propto ||f_i(X)||_2 \tag{2.12}$$

where $f_i(X) \in \mathcal{R}^3$ is the i^{th} row of $f(X)$. And under the case of contour enhance task, the output feature $f(X)$ is the graph convolution results with the input coordinate data and the high pass graph filter. To guarantee the K-localized and shift-invariant, we can design graph filter as the polynomial of the graph shift which is similar to the vertex defined graph filter in equation 2.5:

$$\sum_{l=0}^{K-1} h_l A^l \tag{2.13}$$

where A is the graph shift operator, and K is the order of your filter. And for the high pass graph filtering case, we use the transition matrix $A = D^{-1}W$ as our graph shift operator. To extract better contour after the filter process, we want to sample points that have high local variation, which means those points have high filter responses energy after the high pass filter process [13].

$$(h_{HH}(A)X)_i = x_i - \sum A_{i,j} x_j \tag{2.14}$$

The intuition behind this is that when the local variation of a point is high, its 3D coordinates cannot be well approximated from the 3D coordinates of its neighboring points. The final form of the high pass filter process involves designing a high pass Haar filter (shown in equation 2.15)where it is extremely efficient and doesn't involve any eigendecomposition.

$$h_{HH}(A) = I - A = U \begin{bmatrix} 1-\lambda_0 & \cdots & \cdots & 0 \\ 0 & 1-\lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1-\lambda_{n-1} \end{bmatrix} U^T \tag{2.15}$$

And the final form of the resampling scheme is in the following:

$$\pi_i^\star \propto ||(h_{HH}(A)X)_i||_2^2 = ||x_i - \sum A_{i,j}x_j||_2^2 \tag{2.16}$$

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's an effective way to translate the original point cloud data to a feature map that can capture the characteristics of the object. 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.

2.2.2 Deep Learning on Point Sets

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 [16, 20] or treat it as a collections of images [21]. 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 [22]. Qi et al. proposed a unified architecture showed 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. PoinNet uses the coordinate of the point cloud directly as input data without introducing the discretization error.

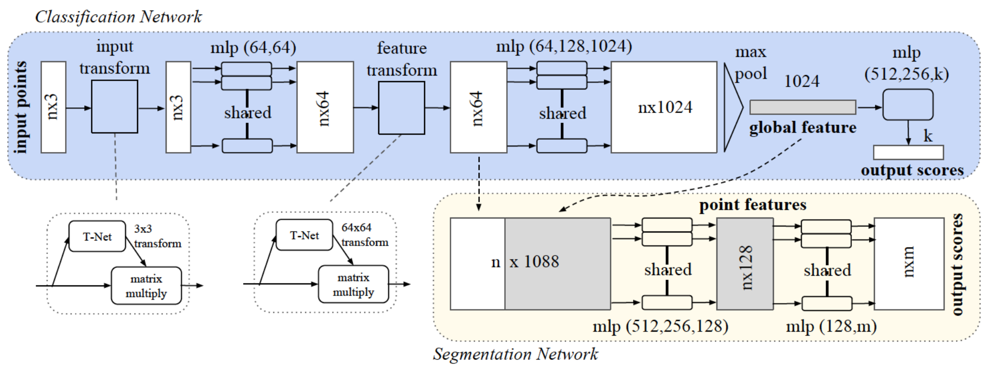


Figure 2.2: PointNet architecture (Reproduced from from Figure 2 in [22])

The three main problems PointNet tries to tackle with 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 3D 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 doesn't contain any local structure information about the point cloud data. However, the neighbors of each point have meaningful information.

- How to reach invariance under transformation (rotation, scaling, etc.)
  .

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

There are two key operations to solve the three problems presented above. First, to achieve permutation invariant, PointNet introduces a series of symmetric transformation functions. The authors have proposed simply using addition and multiplication operations which could easily realize by multilayer perceptron (MLP) network to transform the input point data into a latent representation that is invariance to order. Then, using the max-pooling operation to aggregate global information from each feature maps. This approach is found to work effectively by experiments. Besides, 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 [22]. Second, 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's an idea originates from Spatial transformation network [23]. 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 [22]. It could align features from different point cloud data which guarantee the order invariance and transformation invariance property of the network.

2.3 Graph Convolutional Neural Networks

An image can be represented as a 2D grid graph which each pixel is one node in the graph. Then, we can regard traditional CNNs as a special case of the Graph-CNNs which the filter size and the receptive field number of each node are consistent. There are two properties in image or grid graph that don't exist in general graph structure data. The first thing is in arbitrary graph each node has a different number of neighbors. Besides, the receptive field for 2d grid data naturally has this sense of order where the order sequence can be regarded as from top to bottom, left to right. But this spatial order doesn't exist in general graph structure. Thus, we need to come up new convolutional operation on the graph. To analog to the traditional CNNs, we need to redefine two key operations, convolution step and pooling step. Under the graph setting, previous researchers have proposed various of 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 can be defined in both spectral domain or spatial domain. We will elaborate some representative algorithms in both spectral and spatial approach.

2.3.1.1 Spectral Defined Convolutional Approach

In the spectral approach, the convolutional operation is realized in the graph spectral domain as defined in Equation 2.3. The difference between various of spectral Graph-CNNs mainly lies in how to define the graph filter and how much freedom or restriction you put it on the learnable parameters. Besides, the graph convolution process involves the explicit computation of the eigendecomposition which could be avoided by approximations as mentioned in Section 2.1. We will elaborate three spectral Graph-CNNs approaches including Spectral CNN [7], which is the groundbreaking work in Graph-CNNs and two follow up works ChebyNet [9] and GCN [11] which both achieve competitive performance and scalable learning capacity.

Spectral CNNs [7] Bruna et al.[7] was first to propose defining the graph convolutional layer in the graph spectral domain, which directly operates on the spectrum of the graph weight. The graph convolution layer is defined as follows:

$$x_{t+1,j} = h(U \sum_{i=0}^{d_t-1} F_{i,j} U^T x_{t,i}) \quad (j = 0, 1, \ldots, d_{t+1} - 1) \tag{2.17}$$

where h is a non-linear function applied on the vertex-wise values. $F_{k,i,j}$ is a $n \times n$ diagonal matrix. $\mathbf{x_t} = (x_{t,0}, x_{t,1}, \ldots, x_{t,d_t-1}) \in \mathbf{R}^{n \times d_t}$ and $\mathbf{x_{t+1}} = (x_{t+1,0}, x_{t+1,1}, \ldots, x_{t+1,d_{t+1}-1}) \in \mathbf{R}^{n \times d_{t+1}}$ 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 [7] 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's not a scalable model since explicit getting the Fourier basis will involve $O(N^3)$ computation complexity. Third, the proposed method directly operating on the Fourier basis of graphs which poses difficulties to transform the learned graph filter from one graph structure to the other since using the same filter in different Fourier basis will get very different responses, which narrow the use case to only homogeneous graph classification problem.

GCN 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 polynomial parametric filtering scheme as follows:

$$\hat{h}_s(L) = U(\sum_{k=0}^{K-1} \theta_k \Lambda^k) U^T \tag{2.18}$$

where Λ is the diagonal frequency component matrix and K decides how many hops neighbors will affect the convolution process of each node. This polynomial filtering scheme guarantees the K-hop space localization and reduces the learning complexity to $O(K)$. In the non-parametric graph filter scheme 2.4, the learning complexity for every filter is $O(N)$, where N is the node number of the graph. Thus, the learnable parameter number is related to the input graph dimension which is not a scalable model. However, this K-localized filtering not only guarantees the feature we extract is localized in spatial domain but also enable the learnable filter parameters to be independent of the input graph dimension, which dramatically reduces the learning complexity.

Second, as for the computational complexity, the Spectral-CNNs approach involves doing eigendecomposition on the Laplacian matrix to get the Fourier basis which is $O(N^3)$ computational complexity using QR decomposition. Besides, in each training step, we need to multiply with the Fourier basis twice which is $O(N^2)$ computational complexity. To overcome these problems, [9] proposed to use the recursive format of the Chebyshev polynomial to approximate the convolution process as mentioned in Section 2.1. And with the introduction of the recursive format of Chebyshev polynomial filter approximation process, we don't need to do the eigendecomposition explicitly, and we avoid dense matrix multiplication with the Fourier basis. Derive from [18], the final form of the graph convolution operation can be written as:

$$y = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x \tag{2.19}$$

with the Chebyshev polynomial term at order K defined as:

$$T_k(\tilde{L}) = 2\tilde{L}T_{k-1}(\tilde{L}) - T_{k-2}(\tilde{L}) \tag{2.20}$$

the first two terms $T_0 = I$, $T_1 = \tilde{L}$, where $\tilde{L}$ is the rescaled version of the Laplacian matrix $\tilde{L} = 2L/\lambda_{n-1} - I_n$. The reason we do rescale on the Laplacian matrix 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. 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 to write in a more compact way, Defferrard et al [9] defined $\overline{x} = T_k(\tilde{L})x \in \mathbf{R}^n$. And the Chebyshev filtering approximation can be written as $y = [\overline{x}_0, ..., \overline{x}_{K-1}]\theta$. The vector θ is the Chebyshev coefficients you 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 parameter from one graph structure 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 [0,2], 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 2.19 is the filtering steps used in [9], which is the proper way to do Chebyshev approximation with recursive format. And equation 2.21 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:

$$g_\theta * x \approx \theta_0 x + \theta_1 (L - I_N) x \tag{2.21}$$

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

$$g_\theta * x \approx \theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x = \theta \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} x \tag{2.22}$$

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:

$$H = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \tag{2.23}$$

where filter matrix $\Theta \in R^{m \times h}$ (m is the input features number and h is the feature map number in the convolution layer), X and 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 is proved to be the most effective method in semi-supervised learning problems so far such as node prediction problem in the social network, citation network, 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 spare graph or high 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 and hard to converge.

2.3.1.2 Spatial Defined Convolutional Approach

Different from the above spectral approach, spatial graph convolution is purely defined in the spatial domain, which uses similar spatial filter definition as the traditional CNNs. The difficult part is, unlike the traditional CNNs where the receptive field is just the neighbor pixels, because each vertex may have a different number of neighbors and structures, spatial convolution mainly focuses on how to measure the influence of each neighbor nodes on the central node and how to find the proper mechanism to define the receptive field of each node.

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 'square' 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. And 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 severs as aggregating 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 pooling layer, we can achieve invariant to translation, rotation, 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 [24], which grouping two nodes together by first finding an unmarked nodes i and then finding its unmarked neighbor j that maximizes the local normalized cut $W_{ij}(1/d_i + 1/d_j)$. And 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 [24, 25, 26].

Chapter 3

Proposed method

3.1 Graph Convolutional Based Object Recognition Method

3.1.1 Intuition Explain

In section 2.2.1, we explain that previous works have shown the effective of using graph signal processing approach to deal with 3D point cloud data. In our project, we explore how we can use graph aspect to solve point cloud object classification problem. First, we used similar method as Chen et al. [13] to hand craft various of graph filter to convolve with the input point cloud data. The point cloud data filtering process is shown in Equation 3.1.

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

where $x \in \mathbf{R}^{n \times 3}$ is the input point cloud data and $h(L)$ is the defined graph filter and $y \in \mathbf{R}^{n \times 3}$ is the filtered graph signal. We did experiment on how different graph convolution operations would affect the output point cloud translation features on the 3D object classification benchmark dataset ModelNet [16].

We discovered that doing graph convolution with the same type graph filter will generate similar features of the objects coming from the same object category. Besides, when we visualized the points that have the largest l2 norm after the convolution translation, we found out that points with largest l2 norm will summarize similar points for each object category. It's 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 l2 norm value after the different convolution process 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 high pass graph filter and low pass graph filter respectively from three objects in airplane class. More examples can be found in Appendix Figure A.3,A.4,A.5. We can observe that after the similar graph filtering process, all the objects coming from the same class can be summarized by the similar key points. In other word, objects that come from the same class have similar energy distribution after the same graph filtering operation.

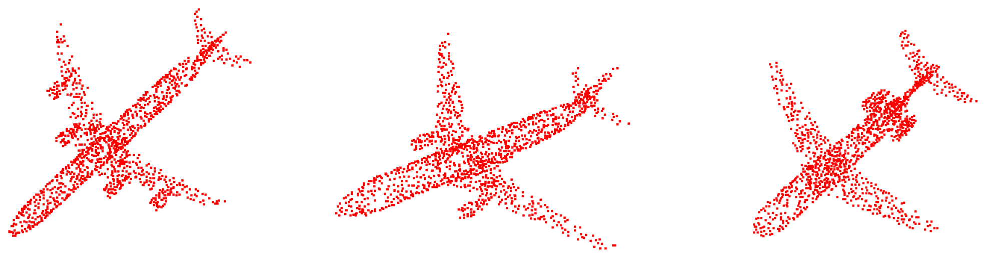

(a) Airplane 1 (b) Airplane 2 (c) Airplane 3

Figure 3.1: Three point cloud objects from Airplane class

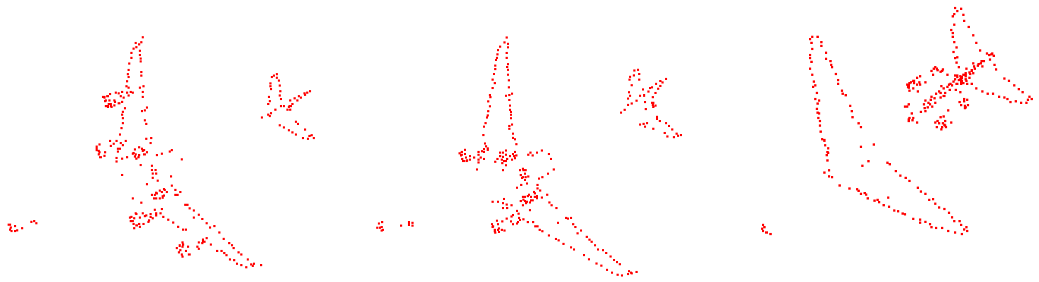


Figure 3.2: 300 points with highest l2 norm after high pass graph filtering

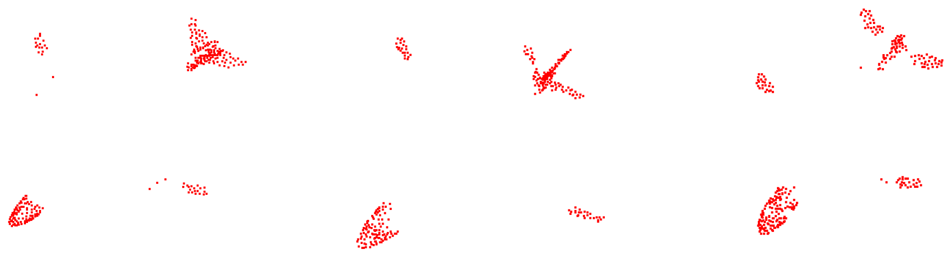


Figure 3.3: 300 points with highest l2 norm after low pass graph filtering

In the typical signal classification problem, various of the 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 $y \in \mathbf{R}^{n \times 3}$. One feature stands out from others is the variance of the filtered graph signal. From the boxplot of some representative class in ModelNet 40 in Figure 3.4, we can see that variance of the filtered signal encodes some extra information that could help distinguish between classes.

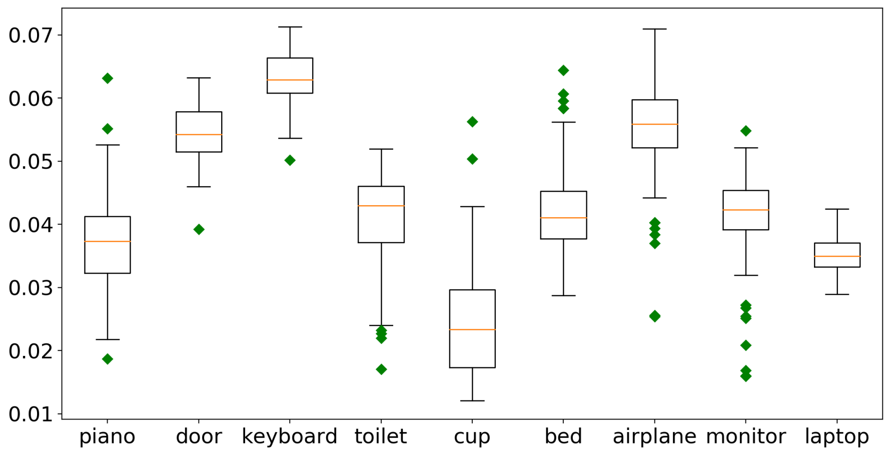


Figure 3.4: Boxplot of the variance after low pass filter from some representative class in ModelNet 40

Thus, using predefined graph filter can extract meaningful features from the original point cloud which can be used to separate different classes. We came up this idea that instead of using hand-craft graph filter, 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 the model as follows.

3.1.2 Proposed Model

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 subsampling scheme. Two subsampling methods, graph-based contour enhanced method (according to section 2.2) and uniform sampling method, are implemented for comparison. Then, we use the preprocessed data to construct a k nearest neighbor graph based on equation 2.11. Similar to the classical CNNs architecture, we use three main types of layers including convolutional layer, pooling layer, and fully-connected layer.

3.1.2.1 Convolutional Layer

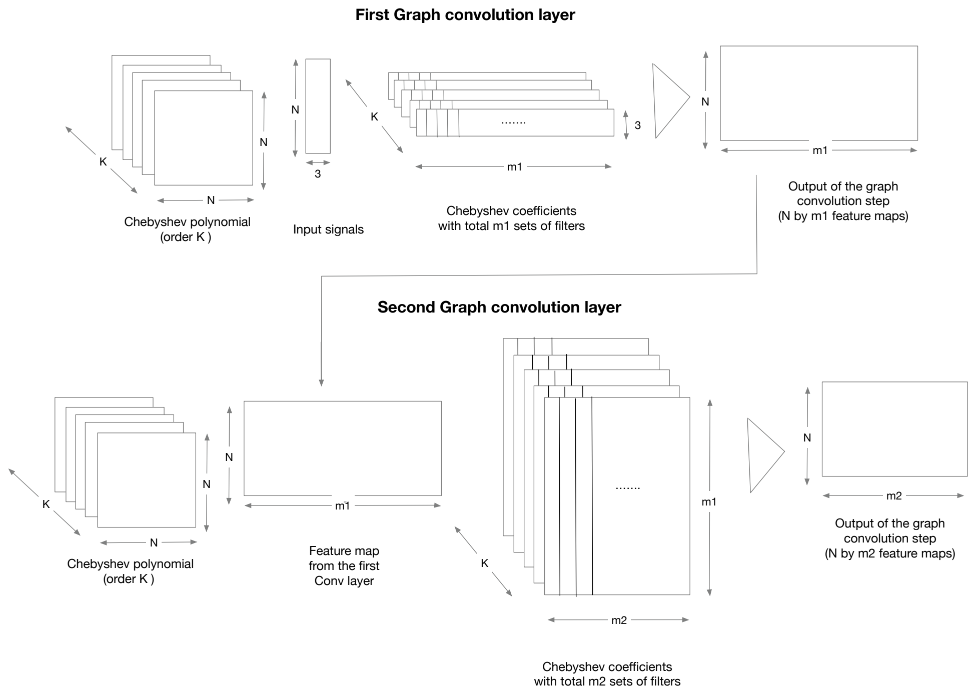
During the training process, our goal is to train a set of graph filter coefficients that can translate the input signal to latent feature maps that can capture the structure information about this object. We treat the x,y,z coordinates of the point cloud as input graph signals. And the convolution is done in the graph spectral domain using localized filtering scheme. As we presented in section 2.3.1, spectral convolution process can be well-approximated by a truncated expansion in terms of the recursive format of order k Chebyshev polynomial [18], which avoids the need to explicit calculate the graph Fourier basis. Defferrard et al. [9] has proved the effectiveness of this convolution block in homogeneous graph related prediction task. Thus, we are using a similar scheme to deal with heterogeneous graphs in our proposed method. The final form of the convolutional layer is shown as follows:

$$y = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x \tag{3.2}$$

where $T_k(\tilde{L})$ is the Chebyshev polynomial at order K, θ is learnable graph filter parameters, x is input graph signals, y is the output of the convolution layer. The Chebyshev polynomial term at order K is defined as:

$$T_k(\tilde{L}) = 2\tilde{L}T_{k-1}(\tilde{L}) - T_{k-2}(\tilde{L}) \tag{3.3}$$

with the first two terms $T_0 = I$, $T_1 = \tilde{L}$, $\tilde{L}$ is the rescaled Laplacian matrix. To incorporate features from different abstraction level and increase the learning capacity of the model. We are using multi-convolutional layer structure. The architecture details for convolution layer is shown in figure 3.5.



First Graph convolution layer
K
N
N
Chebyshev polynomial (order K)
N
3
Input signals
K
....
3
m1
Chebyshev coefficients with total m1 sets of filters
N
m1
Output of the graph convolution step (N by m1 feature maps)
Second Graph convolution layer
K
N
N
Chebyshev polynomial (order K)
N
m1
Feature map from the first Conv layer
K
....
m1
m2
Chebyshev coefficients with total m2 sets of filters
N
m2
Output of the graph convolution step (N by m2 feature maps)

Figure 3.5: Convolution layer

3.1.2.2 Pooling Layer

The feature maps we get from the convolutional layer is the pointwise latent representation of each point. 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 point cloud classification problem. One is purely using global pooling operation on each of the feature maps coming out of the graph convolutional layers pooling without doing multi-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 shown in section 3.1.1, the maximum value and the variance of the graph convolution output encode meaningful information. Thus, we choose to implement 1-max-pooling and variance pooling on the output of each filter to obtain the global signature of each object. And 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. And variance pooling gives us the energy fluctuation level after the convolution, which is an effective way to separate the object with similar overall structure but different interior details, for example in our experiment, we found out that adding this feature can have better performance when separating door, curtain, and key-broad. A similar global feature aggregation layer is also proved to be effective in PointNet [22]. 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 features abstraction. And at last, we add fully connected layers on top of the global feature vector at the end of our neural network to perform final classification step. The architecture details for pooling and fully connected layer is shown in figure 3.6.

We believe this filter learning and multi-pooling layer could potentially be effective when dealing with signal processing problem which involves frequency domain analysis. The typical way to deal with this kind of problem is first doing feature engineering work on the input data and pre-defined specific filter to filter the input signal. And most of the time we need extensive grid search to find the pre-defined filter that can have better performance. After we have the handcrafted filter, then we can analyze the signal in different frequency bands and extract features including mean energy, max energy, and variance, etc. This process can be realized by our multi-pooling layer, which we give freedom to the neural network to learn the filter parameters that can best optimize the downstream prediction task instead using pre-defined filters.

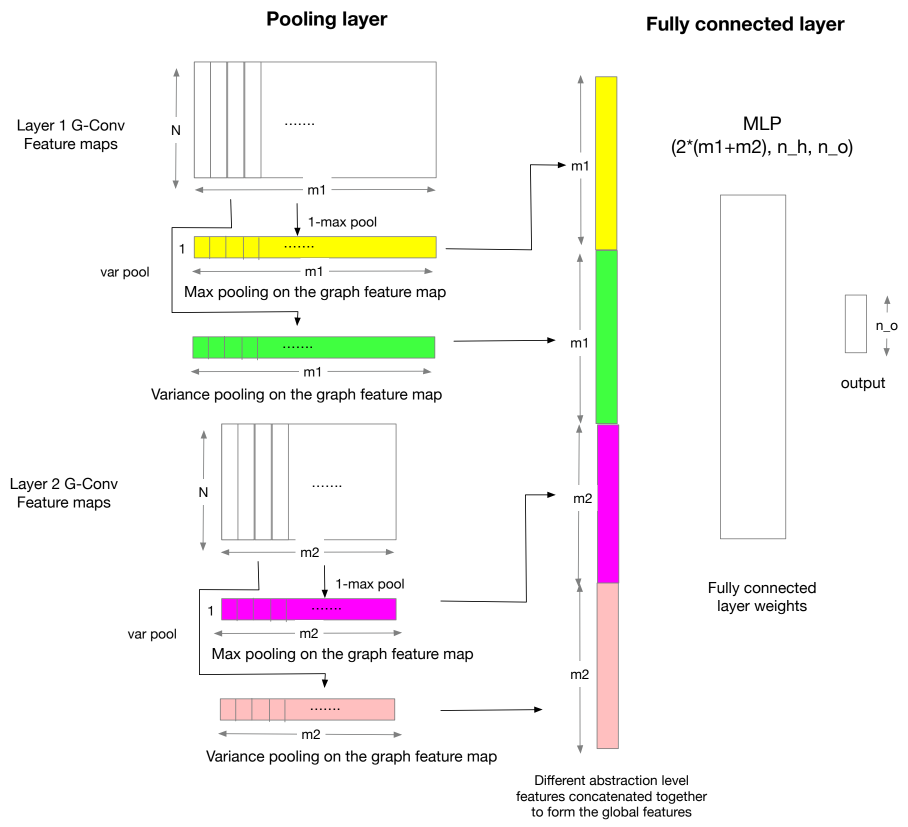

Pooling layer
Fully connected layer
Layer 1 G-Conv Feature maps
N
....
m1
1-max pool
1
var pool
Max pooling on the graph feature map
Variance pooling on the graph feature map
Layer 2 G-Conv Feature maps
m2
MLP (2*(m1+m2), n_h, n_o)
n_o
output
Fully connected layer weights
Different abstraction level features concatenated together to form the global features

Figure 3.6: Global pooling/MLP layer

Multi-resolution pooling Graph clustering process usually consumes huge computation power. However, in the special case of point cloud data, we can benefit from the fact that similar nodes locate close to each other spatially to avoid explicit calculating the graph coarsening process. So instead of using graph coarsening process as proposed in [7, 9], we realized the multi-resolution pooling by sub-sampling a set of points that are most scattered points from each other. This process is called farthest sampling. And each point in this subset will serve as a centroid point. And based on these centroid points, we find the k-nearest neighbor of each centroid point and identify them as a cluster. Then the pooling step is realized by doing max-pooling on the feature maps gathered after the gcn layer at each of the cluster. We can apply this hierarchical pooling scheme after each of the graph convolutional layers. And by doing so, after the first pooling layer, we can shrink dimension of the graph to the cluster number M, which reduces the computation complexity during the training as well as the reduce the learnable parameter number. However, the tradeoff here is the calculation of the farthest sampling and the clustering process will add more computation complexity. The example of applying the multi-resolution pooling is shown in Figure 3.7.

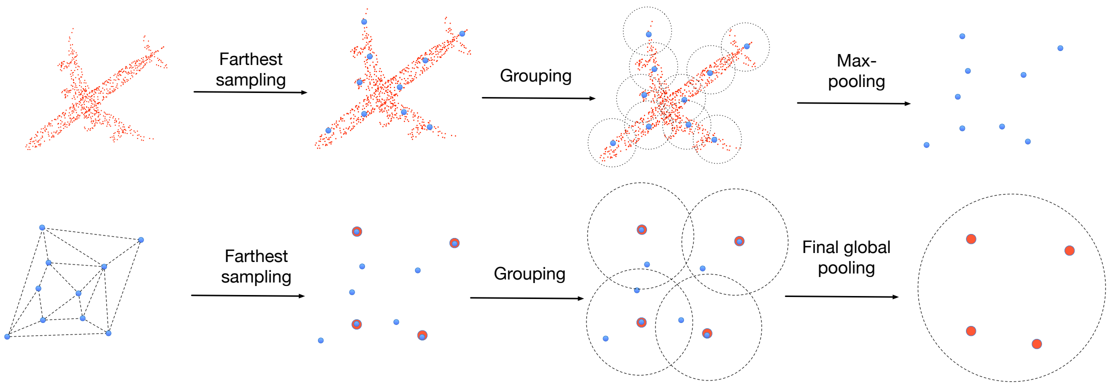


Figure 3.7: Multi-resolution pooling in point set

3.1.3 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, from the learning complexity perspective, we introduce this prior knowledge of the geometric interconnection between points, which, with this prior knowledge, can lower the model's learning complexity and add more convergence stability. However, in PointNet [22], it seems that the local geometric information hasn't been included effectively and needs to be learned implicitly. Secondly, one of the biggest problems the PointNet [22] tried to tackle with is how to achieve the order invariance. PointNet overcomes this problem by symmetric function and spatial transform network. In our proposed algorithm, since the feature map we learn is the spatially localized feature by designing K localized graph filters, which the change in the point order doesn't change the local structure context of each point as long as you set up the same graph structure. And in our pooling layer, both pooling schemes guarantee the order invariance. For global pooling approach, the generation of every final global feature incorporates information from every point's latent representation in the point set. As for the multi-resolution pooling, the hierarchically pooling scheme also guarantee the feature invariance. Then both pooling methods avoid the point order or permutation problem. Third, using the proposed model can also guarantee the rotation and scaling invariance requirement. Because by nature, the graph structure is invariance to rotation, the local connection won't change because of the rotation. As for the scaling problem, our input signals has been normalized into a unit sphere, which as long as the proportional relationship between x,y and z axis doesn't change during the transformation, the model will classify them as the same category. At last, the final feature we extract from our model is coming from different level representations, whether it is the global pooling feature concatenation from different graph convolution layer or the feature from different graph resolution using multi-resolution pooling. Besides, the proposed algorithm also leverages the different scale of the geometric neighbor information because of the K-localized filtering nature.

Chapter 4

Experiment

4.1 3D Point Cloud Classification

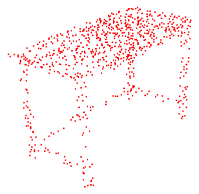
4.1.1 Dataset Description

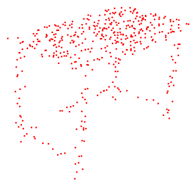
In our experiment, we evaluate our algorithm on the ModelNet40 and ModelNet10, which have been broadly used as shape classification benchmark for 3D object recognition task [16]. ModelNet10 contains 4899 CAD models, split into 3991 for training and 908 for testing. As for ModelNet40, there are 12311 CAD models from 40 categories, split into 9843 for training and 2468 for testing. And the PointNet authors provide the Polygon format of the ModelNet data where they uniformly sample 2048 points on the CAD object mesh face. The object class distribution for both ModelNet 10 and ModelNet40 are listed in the Appendix A.1, A.2. From the figure, we can see that both datasets have unbalanced class distribution, which poses a challenge in model training.

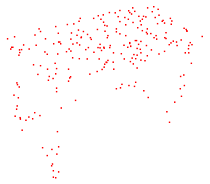
4.1.2 Data Preparation

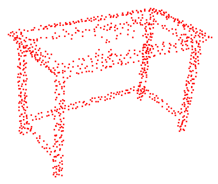
The point number of different objects may vary. Besides, to reduce the computation complexity, we want to compress the point cloud and use a smaller number of point cloud to represent the original one. We preprocess the dataset by using downsampling scheme. Uniform subsampling would be an easy choice. Besides, contour extraction resampling method proposed by Chen et al. [13] has been demonstrated to be an effective way to preserve better structure information from the original point cloud with less total point number. Thus, we implement contour enhance method using high-pass graph filter as 2.15, which the convolution results measure each point's signal recovery performance by its neighbors. And instead of using a re-sampling scheme which is proportional to its l2 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 is very beneficial when we want to do real-time classification where we need more efficient computation.

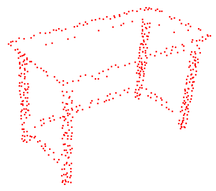
The example of using high pass graph filtering to extract the top 1024, 512, 256 key points from the original object and the comparison results with the uniform subsampling are presented in Figure 4.1 and Figure 4.2. As the figures show, with the same sampling number, doing high-pass graph filter based sub-sampling scheme can achieve more clear contour comparing with the uniform sub-sampling scheme. And even with limited amount of points we can still maintain the contour and structure information of the original object fairly well and capture the points that can discriminate among classes. In our experiment, we will compare the classification results from uniform resampling and this contour enhanced resampling method on the 3D point cloud object recognition task.


(a) N = 1024 (uni)


(b) N = 512 (uni)


(c) N =256 (uni)


(d) N = 1024 (hpf)


(e) N= 512 (hpf)

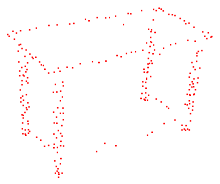
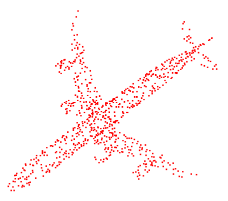
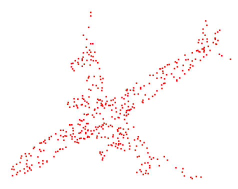
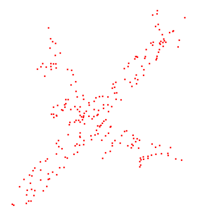
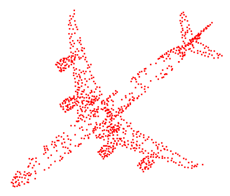

(f) N = 256 (hpf)

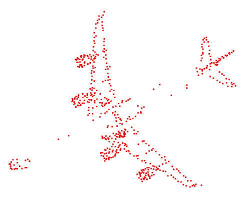
Figure 4.1: Uniform subsampling and contour-enhanced subsampling comparison with point number N = 1024, 512 and 256 (Desk)


(a) N =1024 (uni)


(b) N =512 (uni)


(c) N =256 (uni)


(d) N =1024 (hpf)


(e) N =512 (hpf)

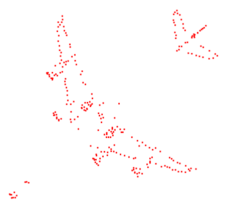

(f) N =256 (hpf)

Figure 4.2: Uniform subsampling and contour-enhanced subsampling comparison with point number N = 1024, 512 and 256 (Airplane)

4.1.3 Network Architecture and Training Details

The basic structure of the model has been shown in section 3.1.2. In this section, we will 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 structure of each object is constructed by 50 nearest neighborhood graph. We will 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 global pooling networks, the model uses two graph convolutional layers with 1200 and 1000 different sets of filters each. One pooling layer follows each convolutional layer containing 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 multi-resolution pooling networks, 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 35 centroid points and each cluster containing 50 nearest neighbors for every centroid point. The second pooling layer contains 6 centroid points and each cluster containing 10 nearest neighbors. The final fully connected layer has 350 nodes. 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 training fashion with batch size 28 to achieve fast convergence and memory-efficient. No data augmentation method is incorporated in our model since no performance improvement has shown by doing so. And 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 achieve higher mean class accuracy. 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. Also, after the graph convolution layer, we use batch normalization [27] with decay=0.90 to reduce the covariance shift in the network. Using batch norm doesn't seem to give performance boost but do seem to help in the sense of convergence speed.

For the global pooling networks, the model uses two graph convolutional layers with 1200 and 1000 different sets of filters each. One pooling layer follows each convolutional layer containing 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 multi-resolution pooling networks, 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 35 centroid points and each cluster containing 50 nearest neighbors for every centroid point. The second pooling layer contains 6 centroid points and each cluster containing 10 nearest neighbors. The final fully connected layer has 350 nodes. For both networks, the convolution steps are done with order 3 Chebyshev filter approximation

The training is done with mini-batch training fashion with batch size 28 to achieve fast convergence and memory-efficient. And to add robustness to our model, the input coordinates are perturbed with Gaussian noise $\sim N(0, 0.08)$. To alleviate the unbalance problem of the dataset, weighted gradient descent has been applied to achieve higher mean class accuracy. 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 been tackled 3D object classification problem from different perspectives. A 3D object can be represented as voxelized shapes, 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 classical or state-of-art methods using volume input [16, 20, 28], images input [21] and point sets [22] input respectively. Since the dataset is a highly unbalanced dataset, we use two metrics including 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-arts 3D object classification approach MVCNN [21]. However, we slightly improve the performance comparing to the state-of-arts point-based 3D object classification method PointNet [22] in the sense of mean instance accuracy. And as shown in Table 4.2 and Figure 4.3, we also improve the stability of the model performance. The fluctuation of the performance during the training is one of the weaknesses in PointNet because its performance is sensitive to the initial state. And in our model, since we introduce the structure of the data by providing the interconnection between points, our proposed algorithm shows more stability performance during the training.

We believe by introducing more deeper network properly, we could potentially improve the performance further. However, at the current stage, we haven't found an effective way to scale our network structure to more than two graphs convolutional layer. Using contour-enhanced subsampling scheme doesn't give us the performance boost as expected. The reason behind it will be presented in section 4.1.5. However, the contour enhanced subsampling scheme will give us more robustness of the model, which will be presented in section 4.1.6. The value of this pre-feature extraction step will be more obvious when we have more limited point number to represent each object.

Besides, our proposed method is scalable for large point cloud data. Because the complexity of the point-based 3D object classification approaches grow linearly with the increase of point number, which is more suitable for real-time object classification, compare to MVCNN [21] where complexity grows squarely on image resolution and volumetric convolution based method like 3D ShapeNets and VoxNet [16, 20] where complexity grows cubically with the volume size [22]. The gap between overall accuracy and average class accuracy is mainly coming from the unbalance data distribution problem. From Table 4.1 we can see this problem is more severe in ModelNet40. And as we mentioned in the previous section, the features map learned from the graph convolutional layer are local geometric structure related features. Thus the data augmentation technique such as permutation and rotation don't contribute too much to the increase of the variety of the data in the under represent classes.

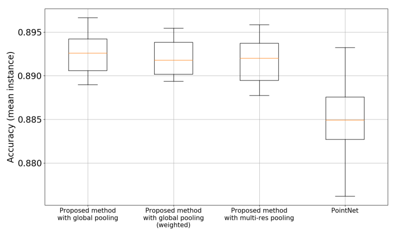
Algorithm	Input format	ModelNet 10 Prediction Accuracy (avg. class)	ModelNet 10 Prediction Accuracy (overall)	ModelNet 40 Prediction Accuracy (avg. class)	ModelNet 40 Prediction Accuracy (overall)
3D ShapeNets[16]	volume (1 view)	83.5%	-	77%	84.7%
VoxNet [20]	volume (12 views)	92%	-	83%	85.9%
SSCN [28]	volume (20 views)	-	-	88.2%	-
MVCNN [21]	image (80 views)	-	-	90.1%	-
PointNet*[22]	point (1024 points per object)	91.33%	91.59%	85.48%	88.49%
Proposed algorithm gloabl pooling	point (1024 points per object)	91.17%	91.87%	85.44%	89.26 %
Proposed algorithm gloabl pooling (weighted)	point (1024 points per object)	91.64%	91.81%	85.70%	89.19 %
Proposed algorithm multi-resolution pooling	point (1024 points per object)	91.24%	91.69%	84.70%	89.17 %

Table 4.1: Results comparison with state-of-art methods on ModelNet [16].

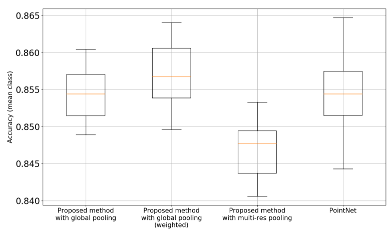
*Means the result is reproduced by the code provided by the paper author. Otherwise, the baseline results presented here is provided by the original paper. And all the result reported above is the average results of 50 trails for each scenario. And only the best model setting is shown. In MVCNN [21], we report the best performance here which used ImageNet1K for pre-train and pooling on input data from multiple rotations or views.

	Mean instance accuracy std.	Mean class accuracy std.
PointNet	0.3193%	0.4143%
Proposed method (global pooling)	**0.1783%**	**0.3367%**
Proposed method (multi-resolution pooling)	0.2245%	0.3368%

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



(a) Overall accuracy



(b) Mean class accuracy

Figure 4.3: Detailed comparison with PointNet and our proposed method (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 compared with volume and image based methods. However, by introducing the pairwise relation between points into the training process, we do need more parameters comparing PointNet when using only global pooling operations. We trade more learning complexity with convergence stability and better performance. With multi-resolution pooling, we reduce the learnable parameter number by half and achieve decent 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)	6.3 M
Proposed algorithm (multi-resolution pooling)	3.3 M
PointNet [22]	3.5 M
Subvolume [29]	16.6 M
MVCNN [21]	60.0 M

Table 4.3: **Learning complexity**.

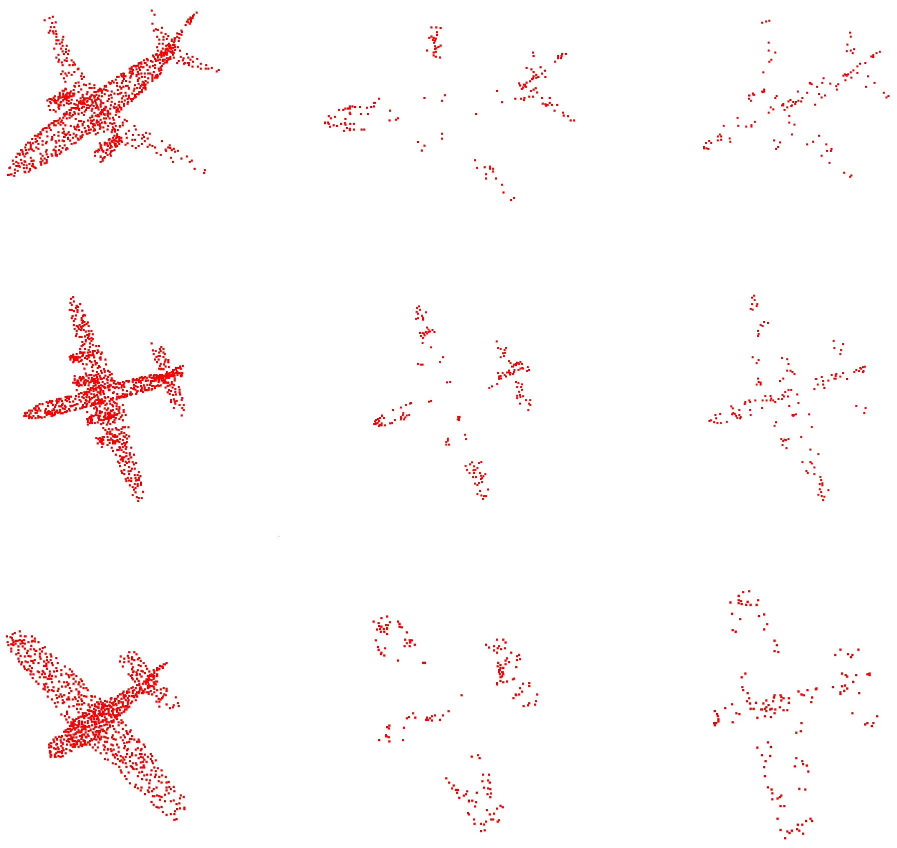
*The "M" stands for million.

4.1.5 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 called the points that have the maximum values among each feature map the active point, which are the points that 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 [4.4, 4.5 4.6]. We can see that the point clouds from 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, which is consistent with our assumption. Besides, another observation we can make is that those two graph convolutional layers learn point features from different abstraction level since the second graph convolutional layer have a larger receptive field which can incorporate the pairwise point interaction from a larger scale compared with the first convolutional layer. Thus, the first graph convolutional active points more inclineto explore local cluster, and the second graph convolutional active points contains more global structure information. Thus, with the global features from different region scale, we can gather more robust and abundant features. In PointNet [22], the similar conclusion is drawn that the global

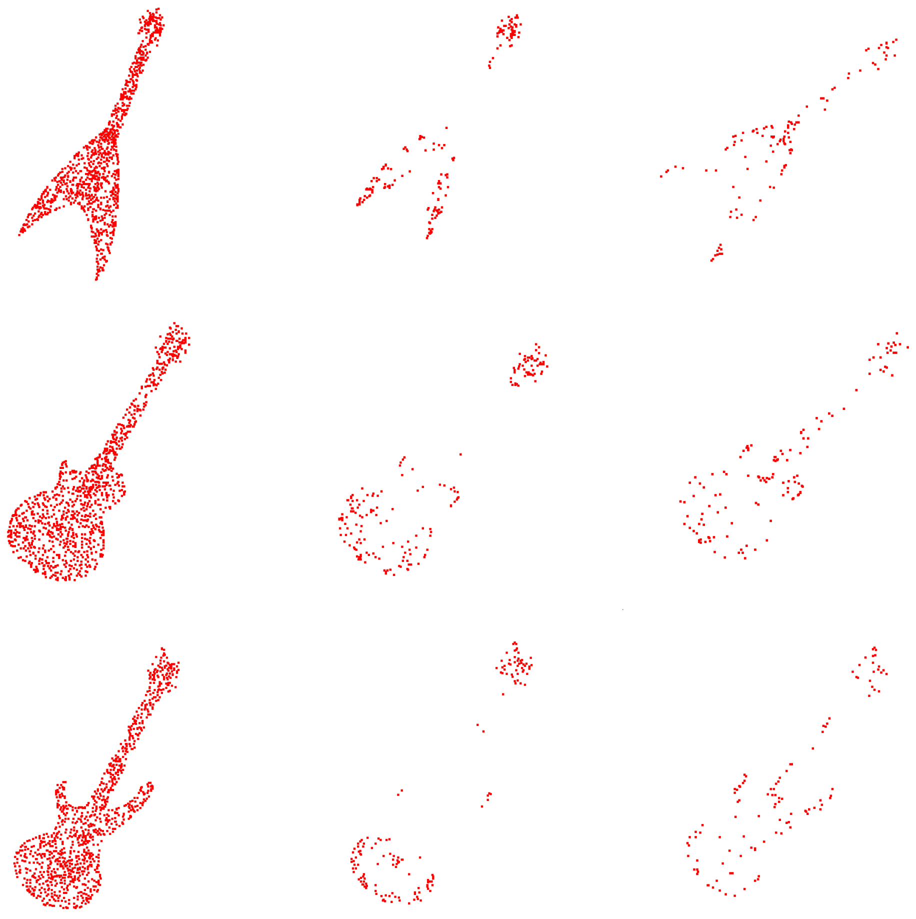
1-max-pooling operation has the ability to summarize the distinctive point set from each category.

One of the interesting things we found out in the experiment is that when using high pass filtering subsampling scheme instead of the uniform subsampling one, 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, and the selected key point is optimized for the classification task without any presumption about the importance of each point, which explains why using the uniform preprocessing scheme outperform slightly than the contour enhanced subsampling scheme.



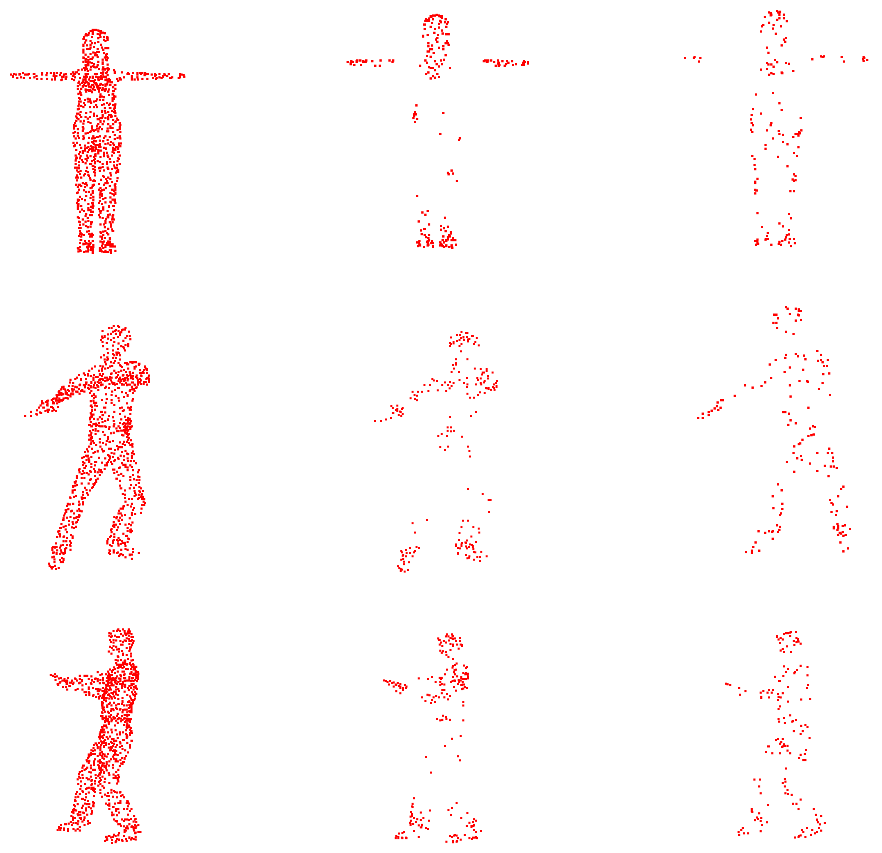
(a) Original point cloud (b) Layer one G-Conv active points (c) Layer two G-Conv active points

Figure 4.4: 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 4.5: Max pooling contribution points from different layers (Guitar)

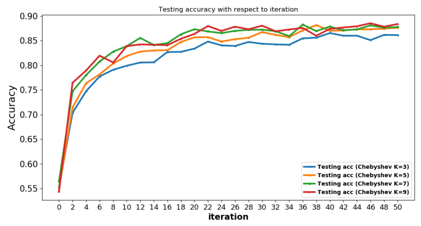


(a) Original point cloud (b) Layer one G-Conv active points (c) Layer two G-Conv active points

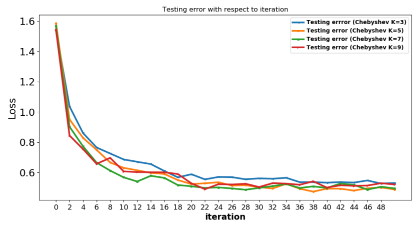
Figure 4.6: Max pooling contribution points from different layers (Person)

4.1.6 The Impact of Hyper-parameters

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 4.7a and 4.7b. 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. And in our Graph-CNNs 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 $K = 7$ is this setting.



(a) Testing accuracy with respect to training epoch



(b) Testing error with respect to training epoch

Figure 4.7: 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 is using 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 0.36% and 0.63% respectively. However, when we further incorporate the average pooling feature, 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	83.98%	85.44%	84.07%
Overall acc	88.82%	89.26%	88.85%

Filter number The filter number control the abundance of your feature maps. Thus, we investigate the impact of the filter number on our model's performance. The experiment is done with one graph convolution layer using different filter size from 200 to 3000. The result is shown in Figure 4.8. 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's 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.

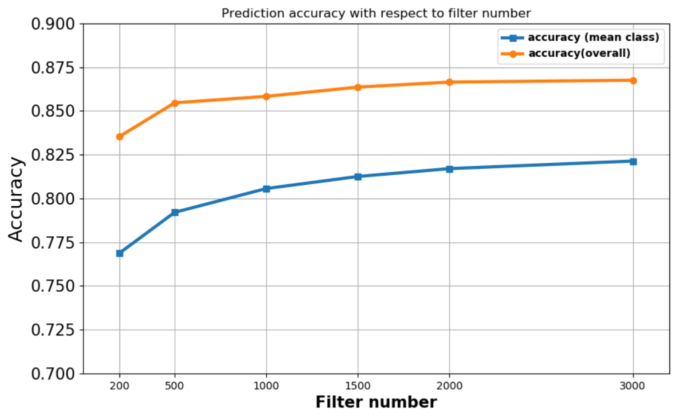


Figure 4.8: Prediction accuracy with respect to filter number

Layer number The advantage with deeper structure is that we can capture highly nonlinear relationships between nodes. And stacking more convolutional layer by nature is also one way to enlarge the influence neighbor region to the center nodes 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. 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. However, adding the third convolution layer doesn't give us the further performance enhancement as we expected. There are several possible reasons. First, the fact that the increase of the receptive field doesn't necessarily guarantee the improvement of performance which could blur the features and jeopardize the performance. Second, all the experiment is conducted with 250 learning epoch, and according to our observation, using three convolution layers have relatively slow convergence rate. 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.

Table 4.5: The impact of layer number

Layer number	one layer	two layers	three layers
Overall acc	86.65%	89.26%	87.50%
Mean class acc	82.44%	85.44%	83.15%

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 ModelNet40 dataset. We attempted to implement weighted gradient 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 minority class. The loss function for the weighted gradient descent is in the following:

To alleviate the unbalance problem of the dataset, weighted gradient descent has been applied to achieve higher mean class accuracy. We attempted to implement weighted gradient 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 minority class.

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

where $\hat{y}_i$ is the prediction class, and y_i is the true label. c_i is the weight associated with each of the samples, which will be related to the probability distribution of its class p_i, specifically, $c_i = \alpha q_i + 1$ and q_i is the normalized value of the inverse of probability distribution of each sample's accordingly category. 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 0.3% which alleviate the problem of overfitting to categories with more instances. However, it drops the overall accuracy slightly.

	mean instance accuracy	mean class accuracy
without weighting scheme	**89.26%**	85.44%
with weighting scheme	89.19%	**85.70%**

Table 4.6: The effect of the weighting scheme.

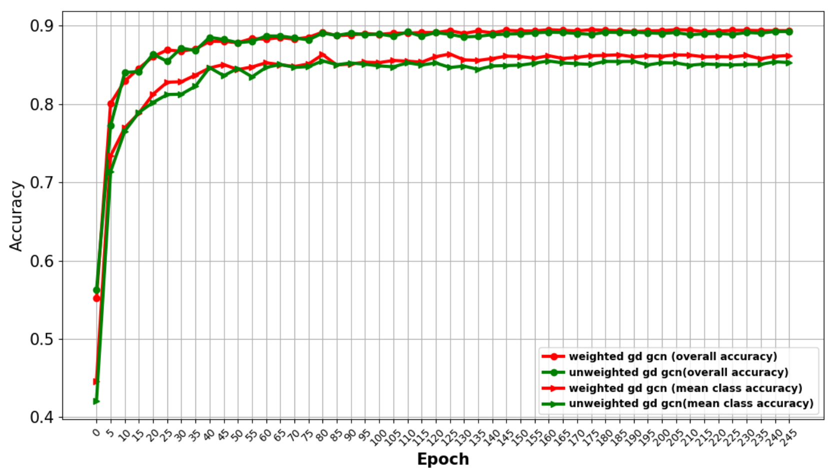


Figure 4.9: Convergence comparison of different weighting scheme

4.1.7 Learning Convergence Analysis

The test set prediction accuracy from the model learning process for both PointNet and our proposed model is shown in Figure 4.10. Generally speaking, our proposed model has faster convergence rate than the PointNet, which is consistent with our assumption. Since we add the prior knowledge about the structure of the data, it can lower the learning complexity with these geometry structure assumptions. As for PointNet [22], the local structure information doesn't incorporate into their input data. Thus, PointNet has to implicit retrieve this geometric information by a series of transformation process including spatial transformation (t-Net block) and series of symmetric functions (addition, multiplication operations and max-pooling). Because the dataset is an unbalanced dataset, we measure both mean instance accuracy and mean class accuracy. As shown in the plot, our proposed model has better performance in overall accuracy compares to PointNet, however, it has slightly worse mean class accuracy. When adding the weighting scheme during training to add extra penalties in the loss function for the minority class, we increase the mean class accuracy farther, as shown in Figure 4.11.

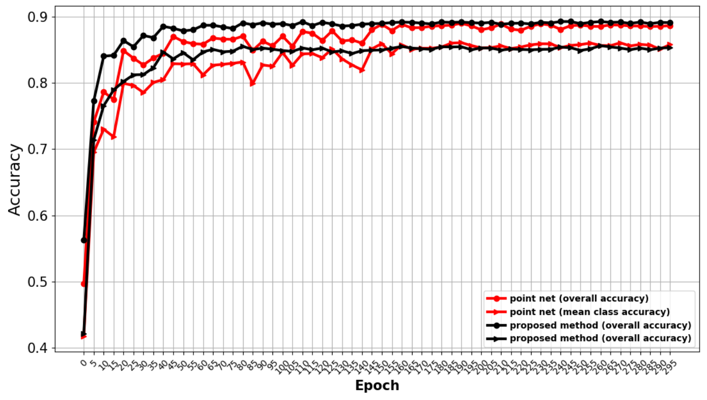


Figure 4.10: Test set accuracy comparison using PointNet and our proposed method on the ModelNet40 dataset

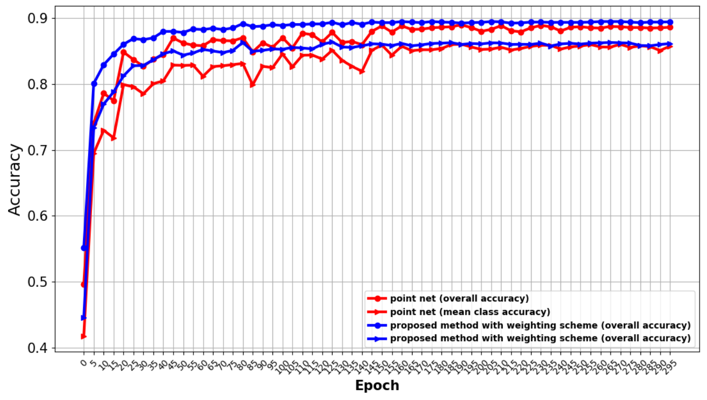
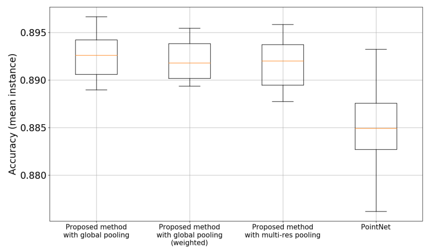
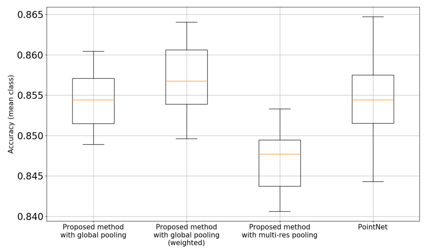


Figure 4.11: Test set accuracy comparison using PointNet and our proposed with weighting scheme on the ModelNet40 dataset

We also compare the stability of the model performance for both our proposed model and PointNet. We record the results from 50 trails of experiments after the model converges. The result is shown in Figure 4.12. It shows that comparing to PointNet, our model produces more consistent results with different initial state and less performance fluctuation after the model converges.



(a) Overall accuracy



(b) Mean class accuracy

Figure 4.12: Detailed comparison with PointNet and our proposed method (with and without weighting scheme)

4.1.8 Robustness Test

In real world's point cloud data, they might exist various kinds of artifacts. To evaluate our model's robustness to noise, we introduce Gaussian noise with different standard deviation 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 models trained with and without contour enhancing prepossessing as well as PointNet [22]. The result is shown in Figure 4.13. Generally speaking, our model is fairly robust to additive Gaussian noise. As we mentioned in the results analysis above, using the contour enhanced preprocessing methods on the input point cloud doesn't 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 subsampling, proposed method with hpf subsampling 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 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 and increase the robustness of the data.

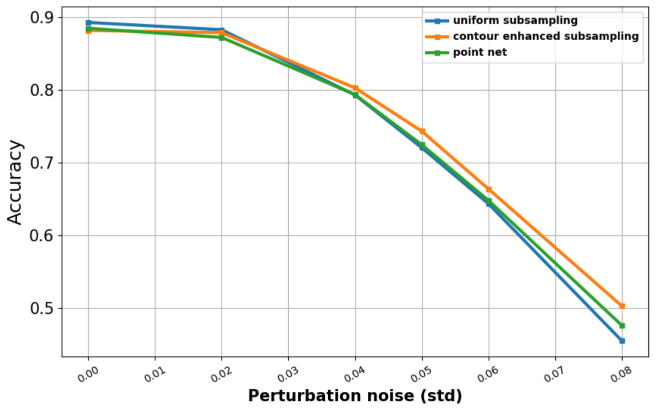


Figure 4.13: Testing set accuracy with respect to noise perturbation

4.1.9 Characteristic of the Wrong Class

The first thing we could notice from the confusion matrix for both ModelNet10 and ModelNet40 (shown in Appendix A.6, A.7) dataset is that the categories with low accuracy are the categories which have relatively less training data. The class with less training samples can't cover enough variety and cause the inferior performance. For example, in the 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. And 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 is some misclassification 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. And in some case, it's even hard to distinguish those point cloud objects by humans if we visualize those objects. Third, there are classes where the overall structure is similar but there are pattern details that can separate those categories, which cannot capture by only using max-pooling. When adding variance pooling, which provides the graph signal variation information after the convolutional layer, it clears the confusion among some class such as the confusion between door and curtain and keyboard.

Chapter 5

Conclusion

5.1 Summary of Contributions

In our project, we explore by far the most effective framework that measures the graph structure and graph signal interaction, namely the Graph Convolutional Neural Networks. We detailed review the existing methods in Graph-CNNs including Spectral CNNs [7], ChebyNet [9], GCN [11] and Diffuse-CNN [10]. Besides, we explore the graph based point cloud processing method. And we further this idea by proposing a Graph-CNNs based 3D point cloud classification model. With multiple fast localized graph convolutional layers and pooling layers with global pooling with 1-max pooling and variance pooling or multi-resolution pooling, our proposed approach is proved by the 3D benchmark dataset ModelNet [16] to be a competitive point-based 3D classification approach.

5.2 Future Work

Point cloud part segmentation In our project, we explored the point cloud object recognition problem. There is another type of classification problem called part segmentation, which is a point-wise classification problem. Part segmentation is extremely meaningful for a lot of application such as motion tracking, etc. 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. And as an efficient way to learn the point-wise latent representation, we believe the proposed method has potential in part segmentation problem as well.

Relax the fix input point number requirement In the 3D point cloud classification task, currently, our proposed model needs to have uniform input point number for each of the objects. Exploring how can we extend the model to achieve graph classification with different numbers of input nodes could be an interesting direction to go in. Because for other graph classification problems, unlike the point cloud data where we can easily do subsampling to unify the point number, there are cases where there may be hard to unify each input graph's node number.

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. 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. And 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 incorporate more graph information or the interaction between the graph signal and graph structure from the theoretical point of view.

Appendix A

Additional Figures and Results

A.1 Data distribution for ModelNet

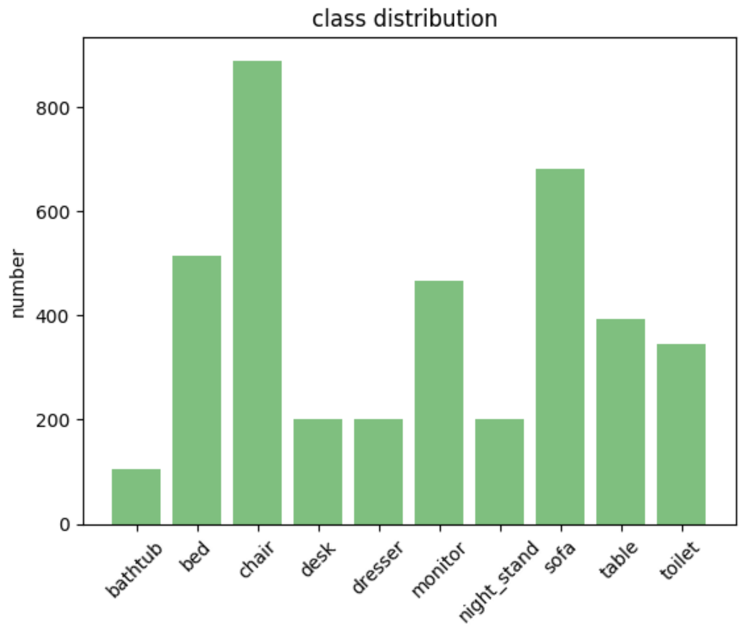


Figure A.1: data distribution for ModelNet10

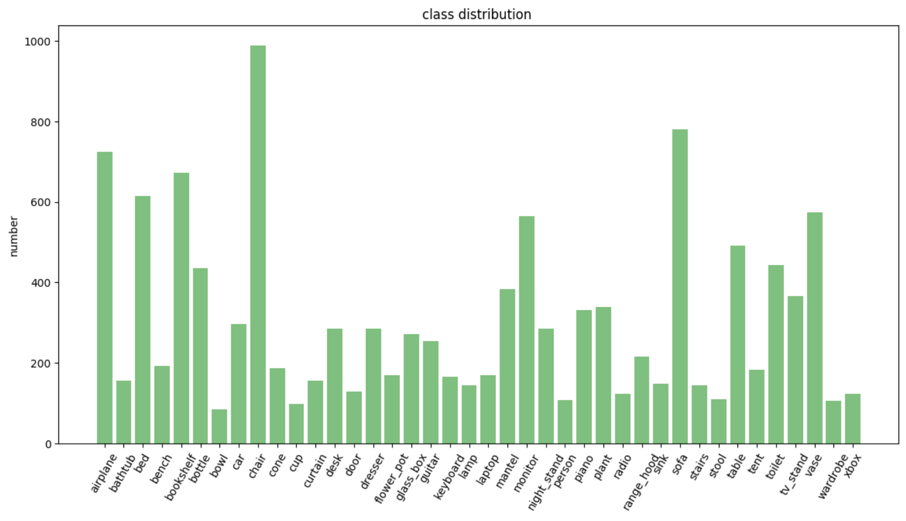
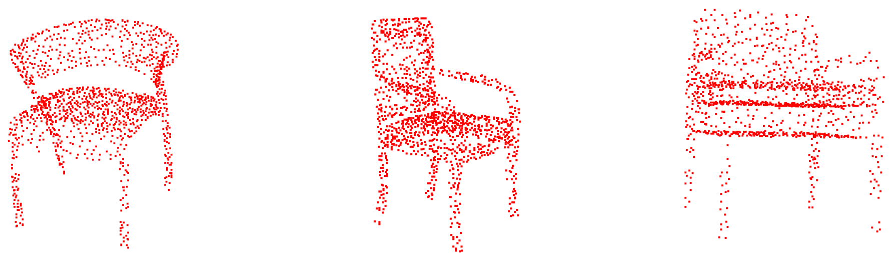


Figure A.2: data distribution for ModelNet40

A.2 The Effect of Different Graph Filters on the Objects from Same Category



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

Figure A.3: Three point cloud objects from chair class

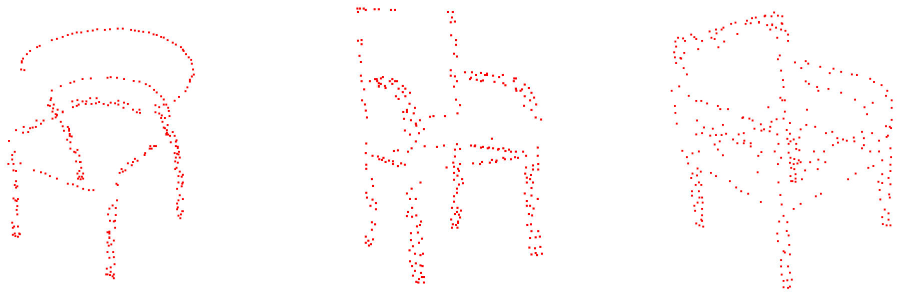


Figure A.4: 300 points with highest l2 norm after high pass graph filtering

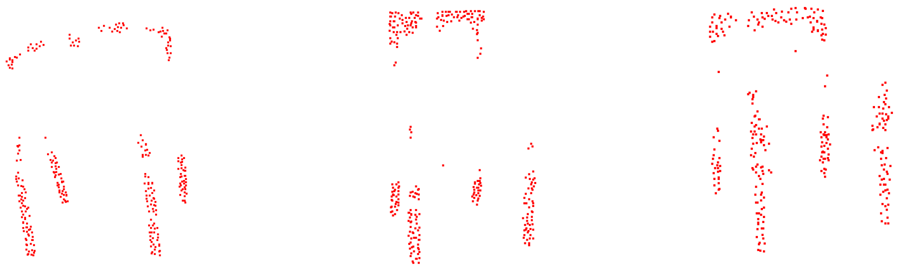


Figure A.5: 300 points with highest l2 norm after low pass graph filtering

A.3 Confusion matrix for ModelNet10 and ModelNet40

The detailed confusion matrix for ModelNet10 and ModelNet40 are shown in the following, the results presented are trained using the proposed model with two convolutional layers:

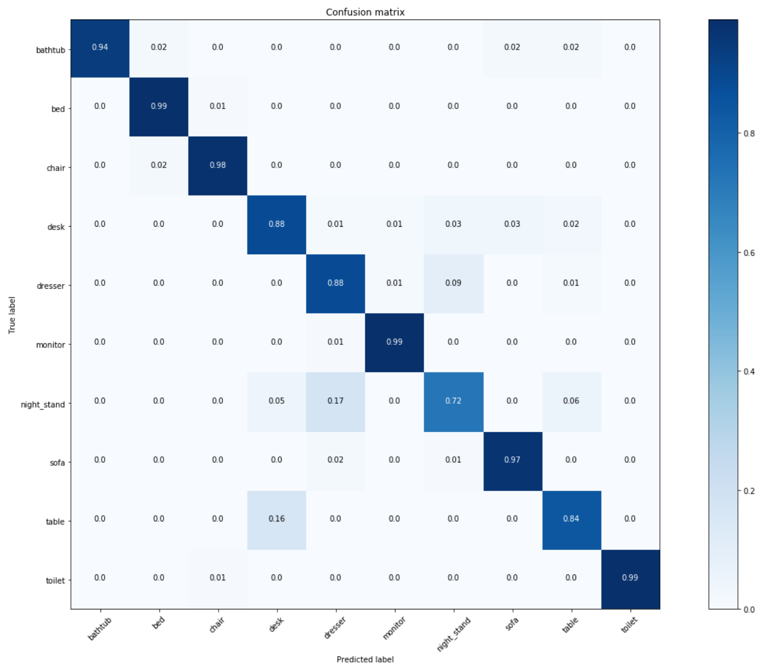


Figure A.6: confusion matrix for ModelNet10 (overall accuracy: 91.85%, avg. class accuracy: 91.35%)

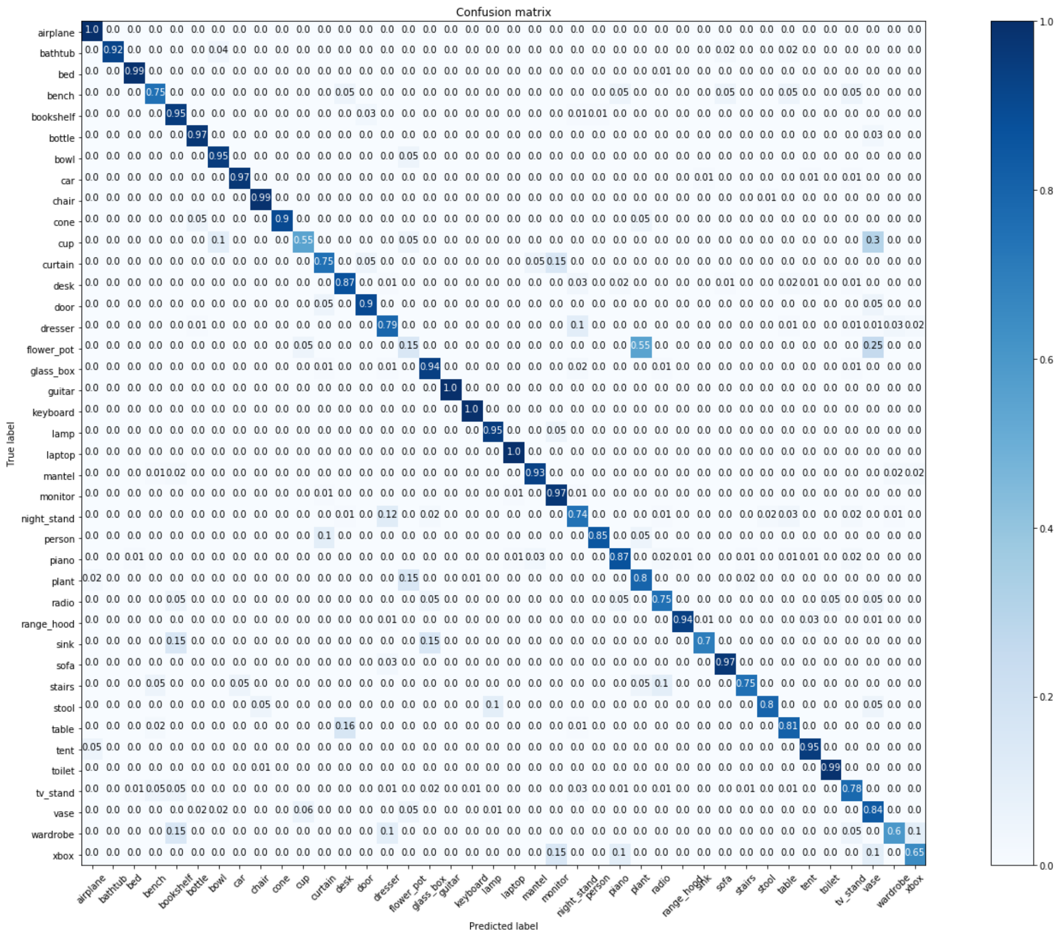


Figure A.7: confusion matrix for ModelNet40 (overall accuracy: 89.28%, avg. class accuracy: 85.54%)

Bibliography

[1] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[2] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, pages 1746–1751. ACL, 2014.

[3] Ye Zhang and Byron C. Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *CoRR*, abs/1510.03820, 2015.

[4] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, pages 912–919. AAAI Press, 2003.

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

[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.*, 30(3):83–98, 2013.

[7] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2013.

[8] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *CoRR*, abs/1506.05163, 2015.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3837–3845, 2016.

[10] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NIPS*, pages 1993–2001, 2016.

[11] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[12] Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph convolutional matrix completion. *CoRR*, abs/1706.02263, 2017.

[13] Siheng Chen, Dong Tian, Chen Feng, Anthony Vetro, and Jelena Kovacevic. Fast resampling of 3d point clouds via graphs. *CoRR*, abs/1702.06397, 2017.

[14] 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.*, 32(4):103–111, 2015.

[15] Dorina Thanou, Philip A. Chou, and Pascal Frossard. Graph-based compression of dynamic 3d point cloud sequences. *IEEE Trans. Image Processing*, 25(4):1765–1778, 2016.

[16] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920. IEEE Computer Society, 2015.

[17] David S. Watkins. *The matrix eigenvalue problem - GR and Krylov subspace methods.* SIAM, 2007.

[18] D.K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[19] D. I Shuman, P. Vandergheynst, and P. Frossard. Chebyshev polynomial approximation for distributed signal processing. In *Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems*, Barcelona, Spain, June 2011.

[20] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS*, pages 922–928. IEEE, 2015.

[21] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*, pages 945–953. IEEE Computer Society, 2015.

[22] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[23] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *NIPS*, pages 2017–2025, 2015.

[24] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors A multilevel approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(11):1944–1957, 2007.

[25] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[26] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[28] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *CoRR*, abs/1706.01307, 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 *CVPR*, pages 5648–5656. IEEE Computer Society, 2016.