Graph Convolutional Networks with EigenPooling

Yao Ma
Michigan State University
mayao4@msu.edu

Suhang Wang
Pennsylvania State University
szw494@psu.edu

Charu C. Aggarwal
IBM T. J. Watson Research Center
charu@us.ibm.com

Jiliang Tang
Michigan State University
tangjili@msu.edu

arXiv:1904.13107v2 [cs.LG] 18 May 2019

ABSTRACT

Graph neural networks, which generalize deep neural network models to graph structured data, have attracted increasing attention in recent years. They usually learn node representations by transforming, propagating and aggregating node features and have been proven to improve the performance of many graph related tasks such as node classification and link prediction. To apply graph neural networks for the graph classification task, approaches to generate the *graph representation* from node representations are demanded. A common way is to globally combine the node representations. However, rich structural information is overlooked. Thus a hierarchical pooling procedure is desired to preserve the graph structure during the graph representation learning. There are some recent works on hierarchically learning graph representation analogous to the pooling step in conventional convolutional neural (CNN) networks. However, the local structural information is still largely neglected during the pooling process. In this paper, we introduce a pooling operator EigenPooling based on graph Fourier transform, which can utilize the node features and local structures during the pooling process. We then design pooling layers based on the pooling operator, which are further combined with traditional GCN convolutional layers to form a graph neural network framework EigenGCN for graph classification. Theoretical analysis is provided to understand EigenPooling from both local and global perspectives. Experimental results of the graph classification task on 6 commonly used benchmarks demonstrate the effectiveness of the proposed framework.

ACM Reference Format:
Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. 2019. Graph Convolutional Networks with EigenPooling. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Recent years have witnessed increasing interests in generalizing neural networks for graph structured data. The stream of research on this topic is usually under the name of "Graph Neural Networks" [34], which typically involves transforming, propagating and aggregating node features across the graph. Among them, some focus on node-level representation learning [18, 22, 35] while others investigate learning graph-level representation [4, 8, 11, 14, 15, 19, 25, 48]. While standing from different perspectives, these methods have been proven to advance various graph related tasks. The methods focusing on node representation learning have brought improvement to tasks such as node classification [14–16, 18, 22, 35] and link prediction [35] and those methods working on graph-level representation learning have mainly facilitated graph classification. In this paper, *we work on graph level representation learning with a focus on the task of graph classification.*

The task of graph classification is to predict the label of a given graph utilizing its associated features and graph structure. Graph Neural Networks can extract graph representation while using all associated information. Majority of existing graph neural networks [7, 11, 17, 25] have been designed to generate good node representations, and then globally summarize the node representations as the graph representation. These methods are inherently "flat" since they treat all the nodes equivalently when generating graph representation using the node representations. In other words, the entire graph structure information is totally neglected during this process. However, nodes are naturally of different statuses and roles in a graph, and they should contribute differently to the graph level representation. Furthermore, graphs often have different local structures (or subgraphs), which contain vital graph characteristics. For instance, in a graph of a protein, atoms (nodes) are connected via bonds (edges); some local structures, which consist of groups of atoms and their direct bonds, can represent some specific functional units, which, in turn, are important to tell the functionality of the entire protein [3, 11, 37]. These local structures are also not captured during the global summarizing process. To generate the graph representation which preserves the local and global graph structures, a hierarchical pooling process, analogous to the pooling process in conventional convolutional neural (CNN) networks [23], is needed.

There are very recent works investigating the pooling procedure for graph neural networks [8, 13, 39, 48]. These methods group nodes into subgraphs (supernodes), coarsen the graph based on these subgraphs and then the entire graph information is reduced to the coarsened graph by generating features of supernodes from their corresponding nodes in subgraphs. However, when pooling the features for supernodes, average pooling or max pooling have been usually adopted where the structures of these group nodes (the local structures) are still neglected. With the local structures,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, July 2017, Washington, DC, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the nodes in the subgraphs are of different statuses and roles when they contribute to the supernode representations. It is challenging to design a general pooling operator while incorporating the local structure information as 1) the subgraphs may contain different numbers of nodes, thus a fixed size pooling operator cannot work for all subgraphs; and 2) the subgraphs could have very different structures, which may require different approaches to summarize the information for the supernode representation. To address the aforementioned challenges, we design a novel pooling operator EigenPooling based on the eigenvectors of the subgraphs, which naturally have the same size of each subgraph and can effectively capture the local structures when summarizing node features for supernodes. EigenPooling can be used as pooling layers to stack with any graph neural network layers to form a novel framework EigenGCN for graph classification. Our major contributions can be summarized as follows:

- We introduce a novel pooling operator EigenPooling, which can naturally summarize the subgraph information while utilizing the subgraph structure;
- We provide theoretical understandings on EigenPooling from both local and global perspectives;
- We incorporate pooling layers based on EigenPooling into existing graph neural networks as a novel framework EigenGCN for representation learning for graph classification; and
- We conduct comprehensive experiments on numerous real-world graph classification benchmarks to demonstrate the effectiveness of the proposed pooling operator.

2 THE PROPOSED FRAMEWORK – EigenGCN

In this paper, we aim to develop a Graph Neural Networks (GNN) model, which consists of convolutional layers and pooling layers, to learn graph representations such that graph level classification can be applied. Before going to the details, we first introduced some notations and the problem setting.

Problem Setting: A graph can be represented as $\mathcal{G} = \{\mathcal{E}, \mathcal{V}\}$, where $\mathcal{V} = \{v_1, \ldots, v_N\}$ is the set of N nodes and $\mathcal{E}$ is the set of edges. The graph structure information can also be represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$. Furthermore, each node in the graph is associated with node features and we use $\mathbf{X} \in \mathbb{R}^{N\times d}$ to denote the node feature matrix, where d is the dimension of features. Note that this node feature matrix can also be viewed as a d-dimensional graph signal [38] defined on the graph $\mathcal{G}$. In the graph classification setting, we have a set of graphs $\{\mathcal{G}_i\}$, each graph $\mathcal{G}_i$ is associated with a label y_i. The task of the graph classification is to take the graph (structure information and node features) as input and predict its corresponding label. To make the prediction, it is important to extract useful information from both graph structure and node features. We aim to design graph convolution layers and EigenPooling to hierarchically extract graph features, which finally learns a vector representation of the input graph for graph classification.

2.1 An Overview of EigenGCN

In this work, we build our model based on Graph Convolutional Networks (GCN) [22], which has been demonstrated to be effective in node-level representation learning. While the GCN model is originally designed for semi-supervised node classification, we only discuss the part for node representation learning but ignoring the classification part. The GCN is stacked by several convolutional layers and a single convolutional layer can be written as:

$$\mathbf{F}^{i+1} = ReLU(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{F}^i\mathbf{W}^i) \tag{1}$$

where $\mathbf{F}^{i+1} \in \mathbb{R}^{N\times d_{i+1}}$ is the output of the i-th convolutional layer for $i > 0$ and $\mathbf{F}^0 = \mathbf{X}$ denotes the input node features. A total number of I convolutional layers are stacked to learn node representations and the output matrix $\mathbf{F}^I$ can be viewed as the final node representations learned by the GCN model.

As we described above, the GCN model has been designed for learning node representations. In the end, the output of the GCN model is a matrix instead of a vector. The procedure of the GCN is rather "flat", as it can only "pass message" between nodes through edges but cannot summarize the node information into the higher level graph representation. A simple way to summarize the node information to generate graph level representation is global pooling. For example, we could use the average of the node representations as the graph representation. However, in this way, a lot of key information is ignored and the graph structure is also totally overlooked during the pooling process.

To address this challenge, we propose eigenvector based pooling layers EigenPooling to hierarchically summarize node information and generate graph representation. An illustrative example is demonstrated in Figure 1. In particular, several pooling layers are added between convolutional layers. Each of the pooling layers pools the graph signal defined on a graph into a graph signal defined on a coarsened version of the input graph, which consists of fewer nodes. Thus, the design of the pooling layers consists of two components: 1) graph coarsening, which divides the graph into a set of subgraphs and form a coarsened graph by treating subgraphs as supernodes; and 2) transform the original graph signal information into the graph signal defined on the coarsened graph with EigenPooling. We coarsen the graph based on a subgraph partition. Given a subgraph partition with no overlaps between subgraphs, we treat each of the subgraphs as a supernode. To form a coarsened graph of the supernodes, we determine the connectivity between the supernodes by the edges across the subgraphs. During the pooling process, for each of the subgraphs, we summarize the information of the graph signal on the subgraph to the supernode. With graph coarsening, we utilize the graph structure information to form coarsened graphs, which makes it possible to learn representations level by level in a hierarchical way. With EigenPooling, we can learn node features of the coarsened graph that exploits the subgraph structure as well as the node features of the input graph.

Figure 1 shows an illustrative example, where a binary graph classification is performed. In this illustrative example, the graph is coarsened three times and finally becomes a single supernode. The input is a graph signal (the node features), which can be multi-dimensional. For the ease of illustration, we do not show the node features on the graph. Two convolutional layers are applied to the graph signal. Then, the graph signal is pooled to a signal defined on the coarsened graph. This procedure (two convolution layers and one pooling layer) is repeated two more times and the graph signal is finally pooled to a signal on a single node. This pooled signal

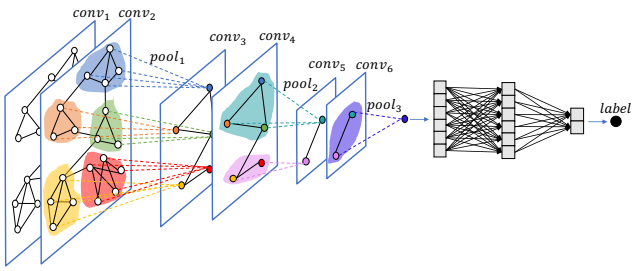


Figure 1: An illustrative example of the general framework

on the single node, which is a vector, can be viewed as the graph representation. The graph representation then goes through several fully connected layers and the prediction is made upon the output of the last layer. Next, we introduce details of graph coarsening and EigenPooling of EigenGCN.

2.2 Graph Coarsening

In this subsection, we introduce how we perform the graph coarsening. As we mentioned in the previous subsection, the coarsening process is based on subgraph partition. There are different ways to separate a given graph to a set of subgraphs with no overlapping nodes. In this paper, we adopt spectral clustering to obtain the subgraphs, so that we can control the number of the subgraphs, which, in turn, determines the pooling ratio. We leave other options as future work. Given a set of subgraphs, we treat them as supernodes and build the connections between them as similar in [40]. An example of the graph coarsening and supernodes is shown in Figure 1, where a subgraph and its supernodes are denoted using the same color. Next, we introduce how to mathematically describe the subgraphs, supernodes, and their relations.

Let $\mathbf{c}$ be a partition of a graph $\mathcal{G}$, which consists of K connected subgraphs $\{\mathcal{G}^{(k)}\}_{k=1}^{K}$. For the graph $\mathcal{G}$, we have the adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ and the feature matrix $\mathbf{X} \in \mathbb{R}^{N\times d}$. Let N_k denote the number of nodes in the subgraph $\mathcal{G}^{(k)}$ and $\Gamma^{(k)}$ is the list of nodes in subgraph $\mathcal{G}^{(k)}$. Note that each of the subgraph can be also viewed as a supernode. For each subgraph $\mathcal{G}^{(k)}$, we can define a sampling operator $\mathbf{C}^{(k)} \in \mathbb{R}^{N\times N_k}$ as follows:

$$\mathbf{C}^{(k)}[i,j] = 1 \quad \text{if and only if} \quad \Gamma^{(k)}(j) = v_i, \tag{2}$$

where $\mathbf{C}^{(k)}[i,j]$ denotes the element in the (i,j)-th position of $\mathbf{C}^{(k)}[i,j]$ and $\Gamma^{(k)}(j)$ is the j-th element in the node list $\Gamma^{(k)}$. This operator provides a relation between nodes in the subgraph $\mathcal{G}^{(k)}$ and the nodes in the original graph. Given a single dimensional graph signal $\mathbf{x} \in \mathbb{R}^{N\times 1}$ defined on the original entire graph, the induced signal that is only defined on the subgraph $\mathcal{G}^{(k)}$ can be written as

$$\mathbf{x}^{(k)} = (\mathbf{C}^{(k)})^T\mathbf{x}. \tag{3}$$

On the other hand, we can also use $\mathbf{C}^{(k)}$ to up-sample a graph signal $\mathbf{x}^{(k)}$ defined only on the subgraph $\mathcal{G}^{(k)}$ to the entire graph $\mathcal{G}$ by

$$\bar{\mathbf{x}} = \mathbf{C}^{(k)}\mathbf{x}^{(k)}. \tag{4}$$

It keeps the values of the nodes in the subgraph untouched while setting the values of all the other nodes that do not belong to the subgraph to 0. The operator can be applied to multi-dimensional signal $\mathbf{X} \in \mathcal{R}^{N\times d}$ in a similar way. The induced adjacency matrix $\mathbf{A}^{(k)} \in \mathbb{R}^{N_k\times N_k}$ of the subgraph $\mathcal{G}^{(k)}$, which only describes the connection within the subgraph $\mathcal{G}^{(k)}$, can be obtained as

$$\mathbf{A}^{(k)} = (\mathbf{C}^{(k)})^T\mathbf{A}\mathbf{C}^{(k)}. \tag{5}$$

The intra-subgraph adjacency matrix of the graph $\mathcal{G}$, which only consists of the edges inside each subgraph, can be represented as

$$\mathbf{A}_{int} = \sum_{k=1}^{K} \mathbf{C}^{(k)}\mathbf{A}^{(k)}(\mathbf{C}^{(k)})^T. \tag{6}$$

Then the inter-subgraph adjacency matrix of graph $\mathcal{G}$, which only consists of the edges between subgraphs, can be represented as $\mathbf{A}_{ext} = \mathbf{A} - \mathbf{A}_{int}$.

Let $\mathcal{G}_{coar}$ denote the coarsened graph, which consists of the supernodes and their connections. We define the assignment matrix $\mathbf{S} \in \mathbb{R}^{N\times K}$, which indicates whether a node belongs to a specific subgraph as:

$$\mathbf{S}[i,j] = 1 \quad \text{if and only if } v_i \in \Gamma^{(j)}.$$

Then, the adjacency matrix of the coarsened graph is given as

$$\mathbf{A}_{coar} = \mathbf{S}^T\mathbf{A}_{ext}\mathbf{S}. \tag{7}$$

With Graph Coarsening, we can obtain the connectivity of $\mathcal{G}_{coar}$, i.e., $\mathbf{A}_{coar}$. Obviously, $\mathbf{A}_{coar}$ encodes the network structure information of $\mathcal{G}$. Next, we describe how to obtain the node features $\mathbf{X}_{coar}$ of $\mathcal{G}_{coar}$ using EigenPooling. With $\mathbf{A}_{coar}$ and $\mathbf{X}_{coar}$, we can stack more layers of GCN-GraphCoarsening-EigenPooling to learn higher level representations of the graph for classification.

2.3 Eigenvector-Based Pooling – EigenPooling

In this subsection, we introduce EigenPooling, aiming to obtain $\mathbf{X}_{coar}$ that encodes network structure information and node features of $\mathcal{G}$. Globally, the pooling operation is to transform a graph signal defined on a given graph to a corresponding graph signal defined on the coarsened version of this graph. It is expected that the important information of the original graph signal can be largely preserved in the transformed graph signal. Locally, for each of the subgraph, we summarize the features of the nodes in this subgraph to a single representation of the supernode. It is necessary to consider the structure of the subgraph when we perform the summarizing, as the subgraph structure also encodes important information. However, common adopted pooling methods such as max pooling [8, 48] or average pooling [11] ignored the graph structure. In some works [30], the subgraph structure is used to find a canonical ordering of the nodes, which is, however, very difficult and expensive. In this work, to use the structure of the subgraphs, we design the pooling operator based on the graph spectral theory by facilitating the eigenvectors of the Laplacian matrix of the subgraph. Next, we first briefly review the graph Fourier transform and then introduce the design of EigenPooling based on graph Fourier transform.

2.3.1 Graph Fourier Transform. Given a graph $\mathcal{G} = \{\mathcal{E}, \mathcal{V}\}$ with $\mathbf{A} \in \mathbb{R}^{N\times N}$ being the adjacency matrix and $\mathbf{X} \in \mathbb{R}^{N\times d}$ being the node feature matrix. Without loss of generality, for the following description, we consider $d = 1$, i.e., $\mathbf{x} \in \mathbb{R}^{N\times 1}$, which can be viewed as a single dimensional graph signal defined on the graph $\mathcal{G}$ [33].

This is the spatial view of a graph signal, which maps each node in the graph to a scalar value (or a vector if the graph signal is multi-dimensional). Analogous to the classical signal processing, we can define graph Fourier transform [38] and spectral representation of the graph signal in the spectral domain. To define the graph signal in the spectral domain, we need to use the Laplacian matrix [6] $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the diagonal degree matrix with $\mathbf{D}[i,i] = \sum_{j=1}^{N} A[i,j]$. The Laplacian matrix $\mathbf{L}$ can be used to define the "smoothness" of a graph signal [38] as follows:

$$s(\mathbf{x}) = \mathbf{x}^T \mathbf{L} \mathbf{x} = \frac{1}{2} \sum_{i,j}^{N} \mathbf{A}[i,j](\mathbf{x}[i] - \mathbf{x}[j])^2. \tag{8}$$

$s(\mathbf{x})$ measures the smoothness of the graph signal $\mathbf{x}$. The smoothness of a graph signal depends on how dramatically the value of connected nodes can change. The smaller $s(\mathbf{x})$, the more smooth it is. For example, for a connected graph, a graph signal with the same value on all the nodes has a smoothness of 0, which means "extremely smooth" with no change.

As $\mathbf{L}$ is a real symmetric semi-positive definite matrix, it has a completed set of orthonormal eigenvectors $\{u_l\}_{l=1}^{N}$. These eigenvectors are also known as the graph Fourier modes [38], which are associated with the ordered real non-negative eigenvalues $\{\lambda_l\}_l^N$. Given a graph signal $\mathbf{x}$, the graph Fourier transform can be obtained as follows

$$\hat{\mathbf{x}} = \mathbf{U}^{\mathbf{T}} \mathbf{x}, \tag{9}$$

where $\mathbf{U} = [u_1, \ldots, u_N] \in \mathbb{R}^{N \times N}$ is the matrix consists of the eigenvectors of $\mathbf{L}$. The vector $\hat{\mathbf{x}}$ obtained after the transform is the representation of the graph signal in the spectral domain. Correspondingly, the inverse graph Fourier transform, which transfers the spectral representation back to the spatial representation, can be denoted as:

$$\mathbf{x} = \mathbf{U}\hat{\mathbf{x}}. \tag{10}$$

Note that we can also view each the eigenvector u_l of the Laplacian matrix $\mathbf{L}$ as a graph signal, and its corresponding eigenvalue λ_l can measure its "smoothness". For any of the eigenvector u_l, we have:

$$s(\mathbf{u}_l) = \mathbf{u}_l^T \mathbf{L} \mathbf{u}_l = \mathbf{u}_l^T \lambda_l \mathbf{u}_l = \lambda_l. \tag{11}$$

The eigenvectors (or Fourier modes) are a set of base signals with different "smoothness" defined on the graph $\mathcal{G}$. Thus, the graph Fourier transform of a graph signal $\mathbf{x}$ can be also viewed as linearly decomposing the signal $\mathbf{x}$ into the set of base signals. $\hat{\mathbf{x}}$ can be viewed as the coefficients of the linear combination of the base signals to obtain the original signal $\mathbf{x}$.

2.3.2 The Design of Pooling Operators. Since graph Fourier transform can transform graph signal to spectral domain which takes into consideration both graph structure and graph signal information, we adopt graph Fourier transform to design pooling operators, which pool the graph signal defined on a given graph $\mathcal{G}$ to a signal defined on its coarsened version $\mathcal{G}_{coar}$. The design of the pooling operator is based on graph Fourier transform of the subgraphs $\{\mathcal{G}^k\}_{k=1}^{K}$. Let $\mathbf{L}^{(k)}$ denote the Laplacian matrix of the subgraph $\mathcal{G}^{(k)}$. We denote the eigenvectors of the Laplacian matrix $\mathbf{L}^{(k)}$ as $u_1^{(k)}, \ldots, u_{N_k}^{(k)}$. We then use the up-sampling operator $C^{(k)}$ to up-sample these eigenvectors (base signals on this subgraph) into the entire graph and get the up-sampled version as:

$$\bar{\mathbf{u}}_l^{(k)} = \mathbf{C}^{(k)} \mathbf{u}_l^{(k)}, l = 1 \ldots N_k. \tag{12}$$

With the up-sampled eigenvectors, we organize them into matrices to form pooling operators. Let $\Theta_l \in \mathbb{R}^{N \times K}$ denote the pooling operator consisting of all the l-th eigenvectors from all the subgraphs

$$\Theta_l = [\bar{u}_l^{(1)}, \ldots, \bar{u}_l^{(K)}] \tag{13}$$

Note that the subgraphs are not necessary all with the same number of nodes, which means that the number of eigenvectors can be different. Let $N_{max} = \max_{k=1,\ldots,K} N_k$ be the largest number of nodes among all the subgraphs. Then, for a subgraph $\mathcal{G}^{(k)}$ with N_k nodes, we set $\mathbf{u}_l^{(k)}$ for $N_k < l \leq N_{max}$ as $\mathbf{0} \in \mathbb{R}^{N_k \times 1}$. The pooling process with l-th pooling operator Θ_l can be described as

$$\mathbf{X}_l = \Theta_l^T \mathbf{X} \tag{14}$$

where $\mathbf{X}_l \in \mathbb{R}^{K \times d}$ is the pooled result using the l-th pooling operator. The k-th row of $\mathbf{X}_l$ contains the information pooled from the k-th subgraph, which is the representation of the k-th supernode.

Following this construction, we build a set of N_{max} pooling operators. To combine the information pooled by different pool operators, we can concatenate them together as follows:

$$\mathbf{X}_{pooled} = [\mathbf{X_0}, \ldots, \mathbf{X}_{N_{max}}]. \tag{15}$$

where $\mathbf{X}_{pooled} \in \mathbb{R}^{K \times d \cdot N_{max}}$ is the final pooled results. For efficient computation, instead of using the results pooled by all the pooling operators, we can choose to only use the first H of them as follows:

$$\mathbf{X}_{coar} = \mathbf{X}_{pooled} = [\mathbf{X_0}, \ldots, \mathbf{X}_H]. \tag{16}$$

In Section 3.1 and Section 3.2, we will show that with $H \ll N_{max}$, we can still preserve most of the information. We will further empirically investigate the effect of choice of H in Section 4

3 THEORETICAL ANALYSIS OF EigenPooling

In this section, we provide a theoretical analysis of EigenPooling by understanding it from local and global perspectives. We prove that the pooling operation can preserve useful information to be processed by the following GCN layers. We also verify that EigenGCN is permutation invariant, which lays the foundation for graph classification with EigenGCN.

3.1 A Local View of EigenPooling

In this subsection, we analyze the pooling operator from a local perspective focusing on a specific subgraph $\mathcal{G}^{(k)}$. For the subgraph $\mathcal{G}^{(k)}$, the pooling operator tries to summarize the nodes' features and form a representation for the corresponding supernode of the subgraph. For a pooling operator Θ_l, the part that is effective on the subgraph $\mathcal{G}^{(k)}$, is only the up-sampled eigenvector $\bar{\mathbf{u}}_l^{(k)}$ as the other eigenvectors have 0 values on the subgraph $\mathcal{G}^{(k)}$. Without the loss of generality, let's consider a single dimensional graph signal

$\mathbf{x} \in \mathbb{R}^{N\times 1}$ defined on the $\mathcal{G}$, the pooling operation on $\mathcal{G}^{(k)}$ can be represented as:

$$(\bar{\mathbf{u}}_l^{(k)})^T\mathbf{x} = (\mathbf{u}_l^{(k)})^T\mathbf{x}^{(k)}, \tag{17}$$

which is the Fourier coefficient of the Fourier mode $\mathbf{u}_l^{(k)}$ of the subgraph $\mathcal{G}^{(k)}$. Thus, from a local perspective, the pooling process is a graph Fourier transform of the graph signal defined on the subgraph. As we introduced in the Section 2.3.1, each of the Fourier modes (eigenvectors) is associated with an eigenvalue, which measures its smoothness. The Fourier coefficient of the corresponding Fourier mode provides the information to indicate the importance of this Fourier mode to the signal. The coefficient summarizes the graph signal information utilizing both the node features and the subgraph structure as the smoothness is related to both of them. Each of the coefficients characterizes a different property (smoothness) of the graph signal. Using the first H coefficients while discarding the others means that we focus more on the "smoother" part of the graph signal, which is common in a lot of applications such as signal denoising and compression [5, 29, 40]. Therefore, we can use the squared summation of the coefficients to measure how much information can be preserved as shown in the following theorem.

Theorem 3.1. *Let $\mathbf{x}$ be a graph signal defined on the graph $\mathcal{G}$ and $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_N]$ be the Fourier modes of this graph. Let $\hat{\mathbf{x}}$ be the corresponding Fourier coefficients, i.e., $\hat{\mathbf{x}} = \mathbf{U}^T\mathbf{x}$. Let $\mathbf{x}' = \sum_{l=1}^{H} \hat{\mathbf{x}}[l]\cdot\mathbf{u}_l$ be the signal re-constructed using only the first H Fourier modes. Then $\frac{||\hat{\mathbf{x}}[1:H]||_2^2}{||\hat{\mathbf{x}}||_2^2}$ can measure the information being preserved by this re-construction. Here $\hat{\mathbf{x}}[1:H]$ denotes the vector consisting of the first H elements of $\hat{\mathbf{x}}$.*

Proof. According to Eq.(10), $\mathbf{x}$ can be written as $\mathbf{x} = \sum_{l=1}^{N} \hat{\mathbf{x}}[l]\cdot\mathbf{u}_l$. Since $\mathbf{U}$ is orthogonal, we have

$$\frac{\|\mathbf{x}'\|_2^2}{\|\mathbf{x}\|_2^2} = \frac{\|\sum_{l=1}^{H}\hat{\mathbf{x}}[l]\cdot\mathbf{u}_l\|_2^2}{\|\sum_{l=1}^{N}\hat{\mathbf{x}}[l]\cdot\mathbf{u}_l\|_2^2} = \frac{\|\hat{\mathbf{x}}[1:H]\|_2^2}{\|\hat{\mathbf{x}}\|_2^2} \tag{18}$$

which completes the proof. □

It is common that for natural graph signal that the magnitude of the spectral form of the graph signal is concentrated on the first few coefficients [33, 38], which means that $\frac{\|\hat{\mathbf{x}}[1:H]\|_2^2}{\|\hat{\mathbf{x}}\|_2^2} \approx 1$ for $H \ll N_k$. In other words, by using the first H coefficients, we can *preserve the majority of the information* while *reducing the computational cost*. We will empirically verify it in the experiment section.

3.2 A Global View of EigenPooling

In this subsection, we analyze the pooling operators from a global perspective focusing on the entire graph $\mathcal{G}$. The pooling operators we constructed can be viewed as a filterbank [40]. Each of the filters in the filterbank filters the given graph signal and obtains a new graph signal. In our case, the filtered signal is defined on the coarsened graph $\mathcal{G}_{coar}$. Without the loss of generality, we consider a single dimensional signal $\mathbf{x} \in \mathbb{R}^{N\times 1}$ of $\mathcal{G}$, then the filtered signals are $\{\mathbf{x}_l\}_{l=1}^{N_{max}}$. Next, we describe some key properties of these pooling operators.

Property 1: Perfect Reconstruction: The first property is that when N_{max} number of filters are used, the input graph signal can be perfectly reconstructed from the filtered signals.

Lemma 3.2. *The graph signal $\mathbf{x}$ can be perfectly reconstructed from its filtered signals $\{\mathbf{x}_l\}_{l=1}^{N_{max}}$ together with the pooling operators $\{\Theta_l\}_{l=1}^{N_{max}}$ as $\mathbf{x} = \sum_{l=1}^{N_{max}} \Theta_l\mathbf{x}_l$.*

Proof. With the definition of Θ_l given in Eq.(13), we have

$$\sum_{l=1}^{N_{max}} \Theta_l\mathbf{x}_l = \sum_{l=1}^{N_{max}}\sum_{k=0}^{K} \bar{\mathbf{u}}_l^{(k)}\cdot\mathbf{x}_l[k] = \sum_{k=0}^{K}\sum_{l=1}^{N_{max}} \bar{\mathbf{u}}_l^{(k)}\cdot\mathbf{x}_l[k] \tag{19}$$

From Eq.(14), we know that $\mathbf{x}_l[k] = (\bar{\mathbf{u}}_l^{(k)})^T\mathbf{x}$. Together with the fact that $\bar{\mathbf{u}}_l^{(k)} = \mathbf{C}^{(k)}\mathbf{u}_l^{(k)}$ in Eq.(12), we can rewrite $\sum_{l=1}^{N_{max}} \bar{\mathbf{u}}_l^{(k)}\cdot\mathbf{x}_l[k]$ as

$$\sum_{l=1}^{N_{max}} \bar{\mathbf{u}}_l^{(k)}\cdot\mathbf{x}_l[k] = \mathbf{C}^{(k)}(\sum_{l=1}^{N_{max}} \mathbf{u}_l^{(k)}\mathbf{u}_l^{(k)^T})\mathbf{C}^{(k)^T}\mathbf{x} \tag{20}$$

Obviously, $\sum_{l=1}^{N_{max}} \mathbf{u}_l^{(k)}\mathbf{u}_l^{(k)^T} = \sum_{l=1}^{N_K} \mathbf{u}_l^{(k)}\mathbf{u}_l^{(k)^T} = \mathbf{I}$, since that $\{\mathbf{u}_l^{(k)}\}_{l=1}^{N_k}$ are orthonormal and $\{\mathbf{u}_l^{(k)}\}_{l=N_k+1}^{N_{max}}$ are all $\mathbf{0}$ vectors. Thus, $\sum_{l=1}^{N_{max}} \bar{\mathbf{u}}_l^{(k)}\cdot\mathbf{x}_l[k] = \mathbf{C}^{(k)}\mathbf{x}^{(k)}$. Substitute this to Eq.(19), we arrive at

$$\sum_{l=1}^{N_{max}} \Theta_l\mathbf{x}_l = \sum_{k=1}^{K} \mathbf{C}^{(k)}\mathbf{x}^{(k)} = \mathbf{x} \tag{21}$$

which completes the proof. □

From Lemma 3.2, we know if N_{max} number of filters are chosen, the filtered signals $\{\mathbf{x}_l\}_{l=1}^{N_{max}}$ can preserve all the information from $\mathbf{x}$. Thus, together with graph coarsening, eigenvector pooling can preserve the signal information of the input graph and can enlarge the receptive filed, which allows us to finally learn a vector representation for graph classification.

Property 2: Energy/Information Preserving The second property is that the filtered signals preserve all the energy when N_{max} filters are chosen. To show this, we first give the following lemma, which serves as a tool for demonstrating property 2.

Lemma 3.3. *All the columns in the operators $\{\Theta_l\}_{l=1}^{N_{max}}$ are orthogonal to each other.*

Proof. By definition, we know that, for the same k, i.e, the same subgraph, $\mathbf{u}_l^{(k)}, l = 1, \ldots N_{max}$ are orthogonal to each other, which means $\bar{\mathbf{u}}_l^{(k)}, l = 1, \ldots N_{max}$ are also orthogonal to each other. In addition, all the $\bar{\mathbf{u}}_l^{(k)}$ with different k are also orthogonal to each other as they only have non-zero values on different subgraphs. □

With the above lemma, we can further conclude that the ℓ_2 norm of graph signal $\mathbf{x}$ is equal to the summation of the ℓ_2 norm of the pooled signals $\{\mathbf{x}_l\}_{l=1}^{N_{max}}$. The proof is given as follows:

Lemma 3.4. *The ℓ_2 norm of the graph signal $\mathbf{x}$ is equal to the summation of the ℓ_2 norm of the pooled signals $\{\mathbf{x}_l\}_{l=1}^{N_{max}}$:*

$$||\mathbf{x}||_2^2 = \sum_{l=1}^{N_{max}} ||\mathbf{x}_l||_2^2 \tag{22}$$

Proof. From Lemma 3.2 and 3.3, we have

$$\|\mathbf{x}\|_2^2 = \| \sum_{l=1}^{N_{max}} \Theta_l \mathbf{x}_l\|_2^2 = \left\| \sum_{k=0}^{K} \sum_{l=1}^{N_{max}} \bar{\mathbf{u}}_l^{(k)} \cdot \mathbf{x}_l[k] \right\|_2^2$$
$$= \sum_{k=0}^{K} \sum_{l=1}^{N_{max}} \mathbf{x}_l^2[k] = \sum_{l=1}^{N_{max}} ||\mathbf{x}_l||_2^2$$

which completes the proof. □

Property 3: Approximate Energy Preserving Lemma 3.4 describes the energy preserving when N_{max} number of filters are chosen. In practice, we only need $H \ll N_{max}$ of filters for efficient computation. Next we show that even with H number of filters, the filtered signals preserve most of the energy/information.

Theorem 3.5. *Let $\mathbf{x}' = \sum_{l=1}^{H} \Theta_l \mathbf{x}_l$ be the graph signal reconstructed only using the first H pooling operators and pooled signals $\{\mathbf{x}_l\}_{l=1}^{H}$. Then the ratio $\frac{\sum_{l=1}^{N_H} ||\mathbf{x}_l||_2^2}{\sum_{l=1}^{N_{max}} ||\mathbf{x}_l||_2^2}$ can measure the portion of information being preserved by this $\mathbf{x}'$.*

Proof. As shown in Lemma 3.2, $\|\mathbf{x}\|_2^2 = \sum_{l=1}^{N_{max}} \|\mathbf{x}_l\|_2^2$. Similarly, we can show that $\|\mathbf{x}'\|_2^2 = \sum_{l=1}^{H} \|\mathbf{x}_l\|_2^2$. The portion of the information being preserved can be represented as

$$\frac{\|\mathbf{x}'\|_2^2}{\|\mathbf{x}\|_2^2} = \frac{\sum_{l=1}^{N_H} ||\mathbf{x}_l||_2^2}{\sum_{l=1}^{N_{max}} ||\mathbf{x}_l||_2^2}. \tag{23}$$

which completes the proof. □

Since for natural graph signals, the magnitude of the spectral form of the graph signal is concentrated in the first few coefficients [33], which means that even with H filters, EigenPooling can preserve the majority of the information/energy.

3.3 Permutation Invariance of EigenGCN

EigenGCN takes the adjacency matrix $\mathbf{A}$ and the node feature matrix $\mathbf{X}$ as input and aims to learn a vector representation of the graph. The nodes in a graph do not have a specific ordering, i.e., $\mathbf{A}$ and $\mathbf{X}$ can be permuted. Obviously, for the same graph, we want EigenGCN to extract the same representation no matter which permutation of $\mathbf{A}$ and $\mathbf{X}$ are used as input. Thus, in this subsection, we prove that EigenGCN is permutation invariant, which lays the foundation of using EigenGCN for graph classification.

Theorem 3.6. *Let $\mathbf{P} = \{0,1\}^{n\times n}$ be any permutation matrix, then* $\mathsf{EigenGCN}(\mathbf{A}, \mathbf{X}) = \mathsf{EigenGCN}(\mathbf{PAP}^T, \mathbf{PX})$*, i.e.,* EigenGCN *is permutation invariant.*

Proof. In order to prove that EigenGCN is permutation invariant, we only need to show that it's key components GCN, graph coarsening and EigenPooling are permutation invariant. For GCN, before permutation, the output is $\mathbf{F} = ReLU(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}^i)$. With permutation, the output becomes

$$ReLU\Big((\mathbf{P}\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{P}^T)(\mathbf{PX})\mathbf{W}\Big) = ReLU(\mathbf{P}\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{XW}) = \mathbf{PF}$$

where we have used $\mathbf{P}^T\mathbf{P} = \mathbf{I}$. This shows that the effect of permutation on GCN only permutes the order of the node representations but doesn't change the value of the representations. Second, the graph coarsening is done by spectral clustering with $\mathbf{A}$. No matter which permutation we have, the detected subgraphs will not change. Finally, EigenPooling summarizes the information within each subgraph. Since the subgraph structures are not affected by the permutation and the representation of each node in the subgraphs is also not affected by the permutation, we can see that the supernodes' representations after EigenPooling are not affected by the permutation. In addition, the inter-connectivity of supernodes is not affected since it's determined by spectral clustering. Thus, we can say that one step of GCN-Graph Coarsening-EigenPooling is permutation invariant. Since finally EigenGCN learns one vector representation of the input graph, we can conclude that the vector representation is the same under any permutation $\mathbf{P}$. □

4 EXPERIMENT

In this section, we conduct experiments to evaluate the effectiveness of the proposed framework EigenGCN. Specifically, we aim to answer two questions:

- Can EigenGCN improve the graph classification performance by the design of EigenPooling?
- How reliable it is to use H number of filters for pooling?

We begin by introducing datasets and experimental settings. We then compare EigenGCN with representative and state-of-the-art baselines for graph classification to answer the first question. We further conduct analysis on graph signals to verify the reasonableness of using H filters, which answers the second question.

4.1 Data sets

To verify whether the proposed framework can hierarchically learn good graph representations for classification, we evaluate EigenGCN on 6 widely used benchmark data sets for graph classification [21], which includes three protein graph data sets, i.e., **ENZYMES** [3, 36], **PROTEINS** [3, 10], and **D&D** [10, 37]; one mutagen data set **Mutagenicity** [20, 31] (We denoted as MUTAG in Table 1 and Table 2); and two data sets that consist of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, **NCI1** and **NCI109** [44]. Some statistics of the data sets can be found in Table 1. From the table, we can see that the used data sets contain varied numbers of graphs and have different graph sizes. We include data sets of different domains, sample and graph sizes to give a comprehensive understanding of how EigenGCN performs with data sets under various conditions.

Table 1: Statistics of datasets

	ENZYMES	PROTEINS	D& D	NCI1	NCI109	MUTAG
# graphs	600	1,113	1,178	4,110	4,127	4,337
mean |V|	32.63	39.06	284.32	29.87	29.68	30.32
# classes	6	2	2	2	2	2

4.2 Baselines and Experimental Settings

To compare the performance of graph classification, we consider some representative and state-of-the-art graph neural network models with various pooling layers. Next, we briefly introduce these baseline approaches as well as the experimental settings for them.

- **GCN** [22] is a graph neural network framework proposed for semi-supervised node classification. It learns node representations by aggregating information from neighbors. As the **GCN** model does not consist of a pooling layer, we directly pool the learned node representations as the graph representation. We use it as a baseline to compare whether a hierarchical pooling layer is necessary.
- **GraphSage** [18] is similar as the GCN and provides various aggregation method. As similar in **GCN**, we directly pool the learned node representations as the graph representation.
- **SET2SET**. This baseline is also built upon **GCN**, it is also "flat" but uses set2set architecture introduced in [43] instead of averaging over all the nodes. We select this method to further show whether a hierarchical pooling layer is necessary no matter average or other pooling methods are used.
- **DGCNN** [49] is built upon the GCN layer. The features of nodes are sorted before feeding them into traditional 1-D convolutional and dense layers [49]. This method is also "flat" without a hierarchical pooling procedure.
- **Diff-pool** [48] is a graph neural network model designed for graph level representation learning with differential pooling layers. It uses node representations learned by an additional convolutional layer to learn the subgraphs (supernodes) and coarsen the graph based on it. We select this model as it achieves state-of-art performance on the graph classification task.
- EigenGCN-H represents various variants of the proposed framework EigenGCN, where H denotes the number of pooling operators we use for EigenPooling. In this evaluation, we choose $H = 1, 2, 3$.

For each of the data sets, we randomly split it to 3 parts, i.e., 80% as training set, 10% as validation set and 10% as testing set. We repeat the randomly splitting process 10 times, and the average performance of the 10 different splits are reported. The parameters of baselines are chosen based on their performance on the validate set. For the proposed framework, we use the 9 splits of the training set and validation set to tune the structure of the graph neural network as well as the learning rate. The same structure and learning rate are then used for all 9 splits.

Following previous work [48], we adopt the widely used evaluation metric, i.e., Accuracy, for graph classification to evaluate the performance.

4.3 Performance on Graph Classification

Each experiment is run 10 times and the average graph classification performance in terms of accuracy is reported in Table 2. From the table, We make the following observations:

Table 2: Performance comparison.

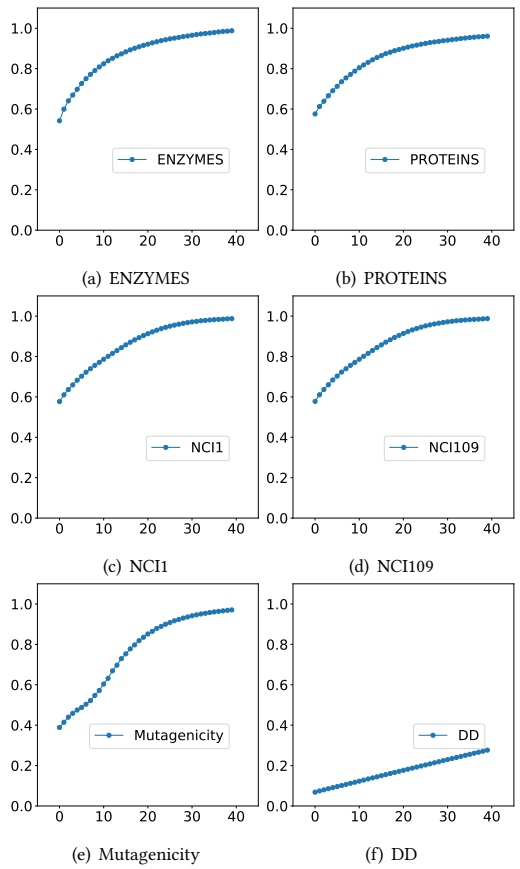
Baselines	Data sets					
	ENZYMES	PROTEINS	D&D	NCI1	NCI109	MUTAG
GCN	0.440	0.740	0.759	0.725	0.707	0.780
GraphSage	0.554	0.746	0.766	0.732	0.703	0.785
SET2SET	0.380	0.727	0.745	0.715	0.686	0.764
DGCNN	0.410	0.732	0.778	0.729	0.723	0.788
Diff-pool	0.636	0.759	0.780	0.760	0.741	**0.806**
EigenGCN-1	**0.650**	0.751	0.775	0.760	0.746	0.801
EigenGCN-2	0.645	0.754	0.770	0.767	0.748	0.789
EigenGCN-3	0.645	**0.766**	**0.786**	**0.770**	**0.749**	0.795

- Diff-pool and the EigenGCN framework perform better than those methods without a hierarchical pooling procedure in most of the cases. Aggregating the node information hierarchically can help learn better graph representations.
- The proposed framework EigenGCN shares the same convolutional layer with GCN, GraphSage, and SET2SET. However, the proposed framework (with different H) outperforms them in most of the data sets. This further indicates the necessity of the hierarchical pooling procedure. In other words, the proposed EigenPooling can indeed help the graph classification performance.
- In most of the data sets, we can observe that the variants of the EigenGCN with more eigenvectors achieve better performance than those with fewer eigenvectors. Including more eigenvectors, which suggests that we can preserve more information during pooling, can help learn better graph representations in most of the cases. In some of data sets, including more eigenvector does not bring any improvement in performance or even make the performance worse. Theoretically, we are able to preserve more information by using more eigenvectors. However, noise signals may be also preserved, which can be filtered when using fewer eigenvectors.
- The proposed EigenGCN achieves the state-of-the-art or at least comparable performance on all the data sets, which shows the effectiveness of the proposed framework EigenGCN.

To sum up, EigenPooling can help learn better graph representation and the proposed framework EigenGCN with EigenPooling can achieve state-of-the-art performance in graph classification task.

4.4 Understanding Graph Signals

In this subsection, we investigate the distribution of the Fourier coefficients on signals in real data. We aim to show that for natural graph signals, most of the information/energy concentrates on the first few Fourier models (or eigenvectors). This paves us a way to only use H filters in EigenPooling. Specifically, given one data set, for each graph $\mathcal{G}_i$ with N_i nodes and its associated signal $\mathbf{X}_i \in \mathbb{R}^{N_i \times d}$, we first calculate the graph Fourier transform and obtain the coefficients $\hat{\mathbf{X}}_i \in \mathbb{R}^{N_i \times d}$. We then calculate the following ratio: $r_i^H = \frac{\|\hat{\mathbf{X}}_i[1{:}H,:]\|_2^2}{\|\hat{\mathbf{X}}_i\|_2^2}$, where $\hat{\mathbf{X}}_i[1:H,:]$ denotes the first i rows of the matrix $\hat{\mathbf{X}}_i$ for various values of H. According to Theorem 3.5, this ratio measures how much information can be preserved by the first H coefficients. We then average the ratio over the entire data



(a) ENZYMES (b) PROTEINS (c) NCI1 (d) NCI109 (e) Mutagenicity (f) DD

Figure 2: Understanding graph signals

set and obtain

$$r^H = \sum_i r_i^H. \tag{24}$$

Note that if $H > N_i$, we set $r_i^H = 1$. We visualize the ratio for each of the data set up to $H = 40$ in Figure 2. As shown in Figure 2, for most of the data set, the magnitude of the coefficients concentrated in the first few coefficients, which demonstrates the reasonableness of using only $H \ll N_{max}$ filters in EigenPooling. In addition, using H filters can save computational cost.

5 RELATED WORK

In recent years, graph neural network models, which try to extend deep neural network models to graph structured data, have attracted increasing interests. These graph neural network models have been applied to various applications in many different areas. In [22], a graph neural network model that tries to learn node representation by aggregating the node features from its neighbors, is applied to perform semi-supervised node classification. Similar methods were later proposed to further enhance the performance by including attention mechanism [42]. GraphSage [48], which allows more flexible aggregation procedure, was designed for the same task. There are some graph neural networks models designed to reason the dynamics of physical systems where the model is applied to predict future states of nodes given their previous states [1, 32]. Most of the aforementioned methods can fit in the framework of "message passing" neural networks [17], which mainly involves transforming, propagating and aggregating node features across the graph through edges. Another stream of graph neural networks was developed based on the graph Fourier transform [4, 8, 19, 24]. The features are first transferred to the spectral domain, next filtered with learnable filters and then transferred back to the spatial domain. The connection between these two streams of works is shown in [8, 22]. Graph neural networks have also been extended to different types of graphs [9, 26, 27] and applied to various applications [12, 28, 35, 41, 45, 47]. Comprehensive surveys on graph neural networks can be found in [2, 46, 50, 51].

However, the design of the graph neural network layers is inherently "flat", which means the output of pure graph neural network layers is node representations for all the nodes in the graph. To apply graph neural networks to the graph classification task, an approach to summarize the learned node representations and generate the graph representation is needed. A simple way to generate the graph representations is to globally combine the node representations. Different combination approaches have been investigated, which include averaging over all node representation as the graph representation [11], adding a "virtual node" connected to all the nodes in the graph and using its node representation as the graph representation [25], and using conventional fully connected layers or convolutional layers after arranging the graph to the same size [17, 49]. However, these global pooling methods cannot hierarchically learn graph representations, thus ignoring important information in the graph structure. There are a few recent works [8, 13, 39, 48] investigating learning graph representations with a hierarchical pooling procedure. These methods usually involve two steps 1) coarsen a graph by grouping nodes into supernode to form a hierarchical structure and 2) learn supernode representations level by level and finally obtain the graph representation. These methods use mean-pooling or max-pooling when they generate supernodes representation, which neglects the important structure information in the subgraphs. In this paper, we propose a pooling operator based on local graph Fourier transform, which utilizes the subgraph structure as well as the node features for generating the supernode representations.

6 CONCLUSION

In this paper, we design EigenPooling, a pooling operator based on local graph Fourier transform, which can extract subgraph information utilizing both node features and structure of the subgraph. We provide a theoretical analysis of the pooling operator from both local and global perspectives. The pooling operator together with a subgraph-based graph coarsening method forms the pooling layer, which can be incorporated into any graph neural networks to hierarchically learn graph level representations. We further proposed a graph neural network framework EigenGCN by combining the proposed pooling layers with the GCN convolutional layers. Comprehensive graph classification experiments were conducted on 6 commonly used graph classification benchmarks. Our proposed framework achieves state-of-the-art performance on most of the data sets, which demonstrates its effectiveness.

7 ACKNOWLEDGEMENTS

Yao Ma and Jiliang Tang are supported by the National Science Foundation (NSF) under grant numbers IIS-1714741, IIS-1715940, IIS-1845081 and CNS-1815636, and a grant from Criteo Faculty Research Award.

REFERENCES

[1] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. 2016. Interaction networks for learning about objects, relations and physics. In *NIPS*. 4502–4510.

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[3] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.

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

[5] Siheng Chen, Aliaksei Sandryhaila, Jose MF Moura, and Jelena Kovacevic. 2014. Signal denoising on graphs via graph filtering. In *GlobalSIP*.

[6] Fan RK Chung and Fan Chung Graham. 1997. *Spectral graph theory*. Number 92. American Mathematical Soc.

[7] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *ICML*. 2702–2711.

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

[9] Tyler Derr, Yao Ma, and Jiliang Tang. 2018. Signed graph convolutional networks. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 929–934.

[10] Paul D Dobson and Andrew J Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *JMB* 330, 4 (2003), 771–783.

[11] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*. 2224–2232.

[12] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph Neural Networks for Social Recommendation. *arXiv preprint arXiv:1902.07243* (2019).

[13] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. 2018. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *CVPR*. 869–877.

[14] Hongyang Gao and Shuiwang Ji. 2019. Graph representation learning via hard and channel-wise attention networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.

[15] Hongyang Gao and Shuiwang Ji. 2019. Graph U-nets. In *Proceedings of The 36th International Conference on Machine Learning*.

[16] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1416–1424.

[17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*. 1263–1272.

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.

[19] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).

[20] Jeroen Kazius, Ross McGuire, and Roberta Bursi. 2005. Derivation and validation of toxicophores for mutagenicity prediction. *JMC* 48, 1 (2005), 312–320.

[21] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. 2016. Benchmark Data Sets for Graph Kernels. http://graphkernels.cs.tu-dortmund.de

[22] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.

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

[25] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[26] Yao Ma, Ziyi Guo, Zhaochun Ren, Eric Zhao, Jiliang Tang, and Dawei Yin. 2018. Dynamic graph neural networks. *arXiv preprint arXiv:1810.10627* (2018).

[27] Yao Ma, Suhang Wang, Chara C Aggarwal, Dawei Yin, and Jiliang Tang. 2019. Multi-dimensional Graph Convolutional Networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 657–665.

[28] Federico Monti, Michael Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*. 3697–3707.

[29] Sunil K Narang and Antonio Ortega. 2012. Perfect reconstruction two-channel wavelet filter banks for graph structured data. *IEEE Transactions on Signal Processing* 60, 6 (2012), 2786–2799.

[30] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*. 2014–2023.

[31] Kaspar Riesen and Horst Bunke. 2008. IAM graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on SPR and (SSPR)*. Springer, 287–297.

[32] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. 2018. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242* (2018).

[33] Aliaksei Sandryhaila and José MF Moura. [n. d.]. Discrete signal processing on graphs. *IEEE transactions on signal processing* 61, 7 ([n. d.]), 1644–1656.

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

[35] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*. Springer, 593–607.

[36] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. 2004. BRENDA, the enzyme database: updates and major new developments. *Nucleic acids research* 32, suppl_1 (2004), D431–D433.

[37] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *JMLR* 12, Sep (2011), 2539–2561.

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

[39] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *CVPR*. 3693–3702.

[40] Nicolas Tremblay and Pierre Borgnat. [n. d.]. Subgraph-based filterbanks for graph signals. *IEEE Transactions on Signal Processing* 64, 15 ([n. d.]), 3827–3840.

[41] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. 2017. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 3462–3471.

[42] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2017. Graph Attention Networks. *arXiv preprint arXiv:1710.10903* (2017).

[43] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2015. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391* (2015).

[44] Nikil Wale, Ian A Watson, and George Karypis. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems* 14, 3 (2008), 347–375.

[45] Xiaolong Wang, Yufei Ye, and Abhinav Gupta. 2018. Zero-shot recognition via semantic embeddings and knowledge graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6857–6866.

[46] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

[47] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 974–983.

[48] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *NIPS*. 4805–4815.

[49] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. (2018).

[50] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2018. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202* (2018).

[51] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *arXiv preprint arXiv:1812.08434* (2018).