Multi-Channel Pooling Graph Neural Networks

Jinlong Du[1] , **Senzhang Wang**[2*] , **Hao Miao**[1] and **Jiaqiang Zhang**[1]
[1] Nanjing University of Aeronautics and Astronautics
[2]Central South University
{kingloon, haomiao, zhangjq}@nuaa.edu.cn , szwang@csu.edu.cn

Abstract

Graph pooling is a critical operation to downsample a graph in graph neural networks. Existing coarsening pooling methods (*e.g.* DiffPool) mostly focus on capturing the global topology structure by assigning the nodes into several coarse clusters, while dropping pooling methods (*e.g.*, SAGPool) try to preserve the local topology structure by selecting the top-k representative nodes. However, there lacks an effective method to integrate the two types of methods so that both the local and the global topology structure of a graph can be well captured. To address this issue, we propose a Multi-channel Graph Pooling method named MuchPool, which captures the local structure, the global structure and node features simultaneously in graph pooling. Specifically, we use two channels to conduct dropping pooling based on the local topology and node features respectively, and one channel to conduct coarsening pooling. Then a cross-channel convolution operation is designed to refine the graph representations of different channels. Finally, the pooling results are aggregated as the final pooled graph. Extensive experiments on six benchmark datasets present the superior performance of MuchPool. The code of this work is publicly available at Github[1].

1 Introduction

Convolution Neural Networks (CNNs) [Montavon *et al.*, 2012] have shown great capability in various machine learning tasks, such as image recognization [He *et al.*, 2016] and video classification [Karpathy *et al.*, 2014]. As images and texts are Euclidean data, which have regular grid-like structures, convolution and pooling operations can be conveniently conducted in. Given the powerful capability of CNNs on grid-like data, it is appealing to generalize convolution and pooling operations on graph-structured data [Gori *et al.*, 2005]. However, significantly different from grid-like data, the locality of nodes cannot be clearly defined in a graph, making the generalization of convolution and pooling operations on graphs extremely challenging.

In recent years, there has been a myriad of attempts to extend convolution operations to arbitrarily shaped graphs, referred as Graph Neural Networks (GNNs). The application of various GNNs for node representation learning [Veličković *et al.*, 2018] have obtained outstanding performance in graph-related learning tasks, such as node classification [Veličković *et al.*, 2018] and link prediction [Schlichtkrull *et al.*, 2018]. However, in some scenarios, the graph-level representation is also needed to perform graph-level machine learning tasks such as predicting the physical property of a given molecule [Wang *et al.*, 2019]. Although considerable efforts have been made in studying the convolution operation in GNNs, one essential problem in graph representation learning, how to effectively perform the pooling operation to downsample the graphs, is not fully studied and remains to be a challenging research problem.

Early graph representation learning models generally utilize simple readout function (such as mean pooling and max pooling) [Henaff *et al.*, 2015] to summarize all the nodes' representations to represent the graph. Such simple readout function methods are inherently flat [Zhang *et al.*, 2019] as the graph topology information is largely ignored. To capture the global topology structure of a graph, coarsening pooling methods, such as DiffPool [Ying *et al.*, 2018], utilize differentiable pooling operators to learn soft assignment matrices to map each node into sets of clusters. Then the nodes of a cluster are pooled as one node to form a coarsened graph. Another line of researchers propose the dropping pooling methods, including Graph U-Nets [Gao and Ji, 2019], AttPool [Huang *et al.*, 2019] and SAGPool [Lee *et al.*, 2019], to better capture the local topology structure. The general idea behind these approaches is to first select top-k important nodes of a graph and then drop all the other nodes to downsample the graph. However, there lacks an effective method to integrate the two types of methods so that both the local and the global topology structure of a graph can be well captured.

To address the limitations of existing graph pooling methods, we propose a novel multi-channel graph pooling method named MuchPool, which can effectively integrate the global topology structure, the local topology structure and the node features in graph pooling via using three channels. To be spe-

*Corresponding author
[1]https://github.com/kingloon/MultiChannelPooling

cific, we use two channels to conduct dropping pooling based on the local structure and node features respectively, and one channel to conduct coarsening pooling based on the global topology structure. Then a cross-channel graph convolution operation is proposed to integrate and refine the node representations of the three channels. Finally, the pooling results of different channels are aggregated to form the final pooled graph. To utilize MuchPool in the task of graph classification, we implement an end-to-end architecture by stacking graph convolution layers and the MuchPool layers to learn a hierarchical representation for a graph. We conduct extensive experiments over six widely used graph datasets in GNNs. The results show that our proposal achieves significant performance improvement on graph classification compared with current state-of-the-art models. We summarize our main contributions as follows.

- We for the first time propose a novel multi-channel graph pooling method that can capture and integrate local patch, coarse-grained structure and node features simultaneously.
- We design a cross-channel graph convolution operation to refine the node representations, and propose a pooling aggregation operation to integrate the pooling results of different channels.
- We conduct extensive experiments on multiple public datasets to demonstrate MuchPool's capability of learning effective graph representation by comparing it with several state-of-the-art methods.

2 Related work

2.1 Graph neural networks

GNNs can be generally divided into two categories: spectral and spatial approaches. Based on spectral graph theory, Bruna et al. [Bruna *et al.*, 2014] first defined convolution operation in Fourier transform domain. When utilizing spectral filters, graph Laplacian has to be computed, causing heavy computation cost, so this method is hard to generalize to graphs with large size. To address this challenge, Kipf and Welling [Kipf and Welling, 2017] proposed a simplified model by utilizing 1-*st* approximation of the Chebyshev expansion. The spatial approaches define convolution by aggregating the information of central node and its neighbors, aiming at making it work on graphs directly. Among them, GraphSAGE [Hamilton *et al.*, 2017] generates node embeddings via aggregating nodes information from sampling strategy. GAT [Veličković *et al.*, 2018] integrates attention mechanism into the process of information aggregation. Both spectral and spatial methods can fit within the framework of "neural message-passing" [Gilmer *et al.*, 2017]. Under this framework, GNN is regarded as a message-passing algorithm where node representations are computed iteratively by aggregating messages from its neighboring nodes through edges using a differentiable function.

2.2 Graph Pooling

Pooling operation can downsize inputs, thus reduce the number of parameters and enlarge receptive fields, leading to better generalization performance. Recent graph pooling methods can be grouped into two big branches: global pooling and hierarchical pooling.

Global graph pooling, also known as a graph readout operation [Xu *et al.*, 2019; Lee *et al.*, 2019], adopts summation operation or neural networks to integrate all the node embeddings into a vector as graph representation. Set2set [Vinyals *et al.*, 2015] uses the LSTM model to find the important nodes in a graph and generate graph representation. DGCNN [Zhang *et al.*, 2018] sorts node embeddings and then summarizes the graph by concatenating some of the node embeddings. Though flexible to apply on graphs with different sizes, global graph pooling methods perform pooling operations only based on node features, and thus some structural information could be lost.

Hierarchical graph pooling methods aim at learning a hierarchical representation via building hierarchical GNNs. DiffPool [Ying *et al.*, 2018] uses graph neural networks to learn a soft assignment matrix mapping nodes to a set of clusters, so it is computation expensive. U-Nets [Gao and Ji, 2019], SAGPool [Lee *et al.*, 2019] and AttPool [Huang *et al.*, 2019] design top-k node selection strategy to pick out the most important nodes to form a pooled graph with them. Such methods are more effective as they only need to calculate an important score for each node, but they neglect to consider the graph topology during this procedure. EdgePool [Diehl, 2019] gets a pooled graph by contracting the edges in a graph. It is unflexible since it can only cut the number of nodes in half each time. EigenPool [Ma *et al.*, 2019] performs pooling operation based on graph Fourier transform. Due to operating spectral clustering, it is very time-consuming. To sum up, there still lacks an effective method that can combine the top-k based and graph coarsening based methods to yield a better hierarchical graph representation.

3 Problem Statement

In this paper, we focus on the graph classification task, which maps each graph to a set of labels. Considering an arbitrary graph $G = (A, X)$, we have $A \in \{0,1\}^{n\times n}$ denotes the adjacency matrix, and $X \in R^{n\times d}$ denotes the feature matrix in which each row represents a d-dimentional feature vector of a node. Given a dataset $D = \{(G_1, y_1), (G_2, y_2), ...\}$, the task of graph classification is to learn a mapping function $f : \mathcal{G} \rightarrow \mathcal{Y}$, where $\mathcal{G} = \{G_1, G_2, ..., G_n\}$ is the set of input graphs and $\mathcal{Y} = \{y_1, y_2, ..., y_n\}$ is the set of labels associated with the graphs.

3.1 Graph Convolution Networks

Graph Convolution Network (GCN) [Kipf and Welling, 2017] is an effective tool to extract discriminative features in graphs and achieves state-of-the-art performance in many machine learning tasks. We use GCN as a module of our graph classification model, which is defined as:

$$X^{l+1} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{\frac{1}{2}}X^{(l)}W^{(l)}) \quad (1)$$

where $\hat{A} = A+I$ is the adjacency matrix with self-loop added to each node, $\hat{D} = \sum_j \hat{A}_{i,j}$ is the degree matrix of $\hat{A}$ and

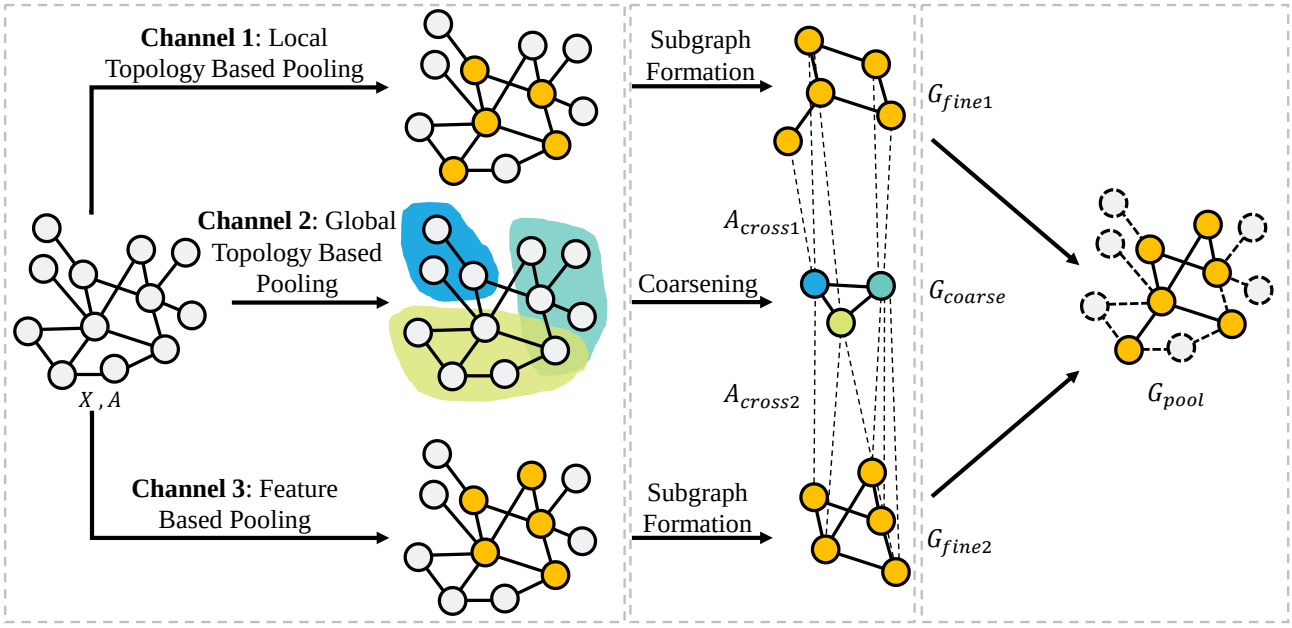

Figure 1: The framework of the proposed MuchPool model.

$W^{(l)}$ is a learnable matrix at layer l. The initial node features are used at the first graph convolution, i.e., $X^{(0)} = X$.

4 MuchPool: Multi-Channel Graph Pooling

The main idea of MuchPool is to use three channels to perform graph pooling to capture different characteristics of a graph, and then aggregate the three channels' pooling results. The original graph is first fed into three GCN layers to learn node embeddings. Based on the intermediate result, we generate two fine-grained pooled graphs G_{fine1} and G_{fine2}, and a coarse-grained graph G_{coarse} as shown in step 1 of Figure 1. At step 2, we conduct cross-channel convolution between channel 2 and channels 1 and 3, respectively. At step 3, we aggregate the outputs of channel 1 and channel 3 to form the final pooled graph G_{pool}.

4.1 Multi-Channel Pooling

Channel 1: Local topology based pooling In channel 1, we propose to use the local topology-based mask method to generate a fine-grained pooled graph G_{fine1}. Specifically, we use the degree measurement to rank the importance of the nodes. In other words, the larger the degree of a node, the more important it is to the entire graph. For example, the nodes with high degrees in road networks may represent some central or important areas and these nodes can better represent the outline of the whole road network [Song *et al.*, 2020; Wang *et al.*, 2020]. The following equations are applied to get the importance scores of each node:

$$s_i = sum(A[i,:]) \tag{2}$$

$$Idx_1 = rank(\boldsymbol{s_1}, k) \tag{3}$$

where s_i denotes the degree of i-th node, $\boldsymbol{s_1} = [s_1, s_2, ..., s_n]$, A denotes the adjacency matrix of the graph, k is the number of nodes selected to reserve in the fine-grained pooled graph. $rank(\boldsymbol{s}, k)$ function is used to sort nodes based on their degree, and it returns the index set of the k-largest values in $\boldsymbol{s}$.

Channel 2: Global topology based pooling In channel 2, we aim at forming a coarse-grained graph G_{coarse}. Many graphs have rich hierarchical structures [Ma *et al.*, 2018], and capturing this kind of structural information is significant for downstream graph classification tasks. Motivated by DiffPool [Ying *et al.*, 2018], we adopt a graph clustering method that uses a GNN to learn a cluster assignment matrix for generating G_{coarse}. The assignment matrix is generated by the following equation:

$$S = softmax(GNN_{pool}(A, X)) \in R^{n \times c} \tag{4}$$

where n is the number of nodes in the input graph and c is the predefined cluster number, GNN_{pool} is the GNN used to cluster nodes, X and A are the feature and adjacency matrices of the input graph, respectively. Based on the assignment matrix, we can obtain G_{coarse} by the following formulas:

$$X_{coarse} = S^T Z \in R^{c \times d} \tag{5}$$

$$A_{coarse} = S^T A S \in R^{c \times c} \tag{6}$$

$$Z = GNN_{embed}(A, X) \in R^{n \times d} \tag{7}$$

where $Z \in R^{n \times d}$ denotes the node embeddings learned from the input graph, X_{coarse} and A_{coarse} denote the feature and adjacency matrices of G_{coarse}. S and Z are both obtained by GNN modules, and the softmax function is used in row-wise manner on S^T to determine the assignment probability of each node of the input graph to the clusters.

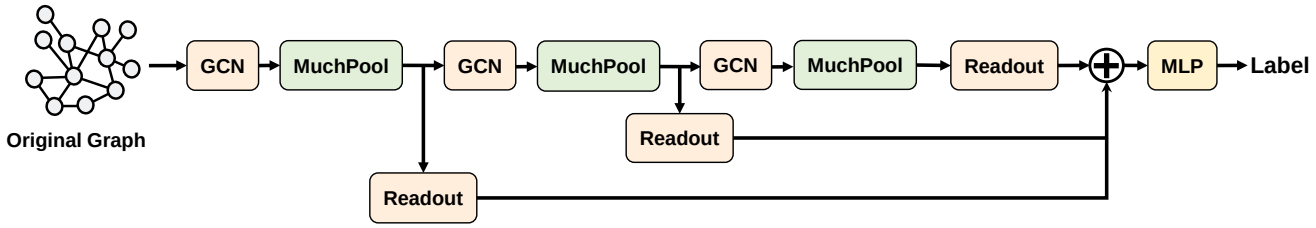

Figure 2: Hierarchical graph representation learning framework with MuchPool model for graph classification.

Channel 3: Node feature based pooling. In channel 3, we select the most important nodes based on their feature embeddings. To be specific, we first use a readout module to get a representation of the current graph, and then transform it with a learnable matrix. The process can be formulated as:

$$Z = readout(X^{(l)}), \qquad Z \in R^{1\times d} \tag{8}$$

$$\widetilde{Z} = \sigma(ZW), \qquad \widetilde{Z} \in R^{1\times d} \tag{9}$$

where Z is the shallow graph representation obtained by using the mean readout operation and $\widetilde{Z}$ is its variant through a nonlinear transformation, $W \in R^{d\times d}$ is a learnable parameter matrix. To select the top-k important nodes in the current graph based on features, we calculate an importance score for each node by applying the following formula.

$$s_i = X[i] \cdot \widetilde{Z} \tag{10}$$

where i is the index of node v_i, s_i denotes the importance score of it, $\boldsymbol{s_2} = [s_1, s_2, ..., s_n]$, and $\cdot$ denotes the operation of inner-product. With this score, a fraction of most important nodes from the aspect of features can be selected, whose indexes are obtained with the same $rank(\boldsymbol{s}, k)$ function:

$$Idx_2 = rank(\boldsymbol{s}, k) \tag{11}$$

4.2 Cross-Channel Convolution

As shown in step 2 of Figure 1, with the outputs of channels 1 to 3, we get three pooled graphs G_{fine1}, G_{fine2} and G_{coarse}. Since the three pooled graphs reflect the different aspects of the initial input graph, we next need to fuse them. To this end, we propose a cross-channel convolution operation between G_{fine1} and G_{coarse}, G_{fine2} and G_{coarse}. The cross-channel convolution operation is defined as follows.

$$H_{fine} = \sigma([H_{fine} + A_{cross} \cdot H_{coarse}] \cdot W) \tag{12}$$

$H_{fine} \in R^{k\times d}$ denotes the node embedding matrix of the fine-grained graph G_{fine1} or G_{fine2}, H_{coarse} denotes the node embedding matrix of G_{coarse}, and $A_{cross} \in R^{k\times c}$ denotes the connection relationship between coarse- and fine-grained pooled graphs, where k and c are the node numbers in the two pooled graphs, respectively. In particular, we obtain A_{cross} by utilizing the assignment matrix S and the important node indices via applying the following equation:

$$A_{cross}[i] = S[i], \quad i \in Idx_1 \quad or \quad Idx_2 \tag{13}$$

The GNNs within the message-passing scheme can be expressed as:

$$H_k = \sigma([H_{k-1} + AH_{k-1}] \cdot W_k) \in R^{n\times d} \tag{14}$$

where H_k denotes the hidden node embeddings after k steps of graph convolution, $A \in \{0,1\}^{n\times n}$ denotes the adjacent matrix, and $W_k \in R^{n\times d}$ denotes a learnable weight matrix, which needs to operate on the same graph. Compared with the formula (12), we replace the adjacency matrix and embedding matrix of the fine-grained graph by A_{cross} and H_{coarse} separately.

4.3 Pooling Aggregation

With the above operations, we have two refined pooled graphs from channels 1 and 3 as shown in the right part of Figure 1. To reduce the loss of discriminative information for graph classification, we need to aggregate them. We denote the indices of the selected nodes in channel 1 as Idx_1 and the embedding matrix after the cross-channel convolution step as X_1. Likewise, the indices of selected nodes and the embedding matrix in channel 3 are Idx_2 and X_2, respectively. The pooled graphs aggregation process can be performed by the following formulations:

$$Idx = Idx_1 \cup Idx_2 \tag{15}$$

With the index, a subgraph that consists of the most representative nodes of the original graph can be extracted, whose adjacency matrix is denoted as:

$$\hat{A} = A[Idx, :] \in \{0,1\}^{K\times N} \tag{16}$$

where $K = |Idx|$ is the order of the indices set of nodes in the original graph that will be reserved, N is the total number of nodes in the original graph. Then, we apply the following two formulas to generate the final pooled graph:

$$A_p = \hat{A}A\hat{A}^T \tag{17}$$

$$X_p[i,:] = \begin{cases} X_1[i,:] & if\ i\ in\ Idx - Idx_2 \\ \frac{X_1[i,:]+X_2[i,:]}{2} & if\ i\ in\ Idx_1 \cap Idx_2 \\ X_2[i,:] & if\ i\ in\ Idx - Idx_1 \end{cases} \tag{18}$$

where $X_p \in R^{K\times d}$ is the node feature matrix for the aggregated pooled graph, and $A_p \in \{0,1\}^{K\times K}$ is its adjacency matrix accordingly.

4.4 Hierarchical Pooling Framework with MuchPool

To use MuchPool in the graph classification task, we implement an end-to-end trainable model by stacking several GCN layers with MuchPool module inserted. The model framework is illustrated in Figure 2. Concretely, we view a GCN layer followed by a MuchPool layer as a complete function unit and name it MuchPool GCN layer for convenience. For a MuchPool GCN layer, it takes a graph as input and outputs a pooled graph that is represented with a new feature matrix and adjacency matrix. Then, the pooled graph is fed into the

Datasets	G_{avg}	C_{avg}	V_{avg}	E_{avg}
PROTEINS	344	2	14.29	715.66
ENZYMES	600	6	32.63	62.14
D&D	1178	2	284.32	14.69
NCI1	4110	2	29.87	32.30
NCI109	4127	2	29.68	32.13
COLLAB	5000	3	74.49	2457.78

Table 1: Statistics of the datasets. G_{avg}, C_{avg}, V_{avg} and E_{avg} denote the average number of graphs, classes, nodes and edges, respectively.

next MuchPool GCN layer, and it is fed into a readout module at the same time, in which the embedding is added up as the graph embedding in this layer. Finally, the graph embeddings in all layers are added up to generate the final graph representation, and it is taken as the input of a MLP classifier to predict the label of the original graph.

5 Experiments

5.1 Datasets and Baselines

We use the following 6 widely used datasets in the classification tasks to evaluate the performance of our proposed model. **D&D** and **PROTEINS** [Dobson and Doig, 2003] are two protein graph datasets, where nodes represent the aminoacids. The label indicates whether or not a protein is a non-enzyme. **NCI1** and **NCI109** [Shervashidze *et al.*, 2011] are two biological datasets for anticancer activity classification, where each graph is a compound graph with nodes and edges representing atoms and chemical bonds separately. **ENZYMES** is a dataset of protein tertiary structures, and each enzyme belongs to one of the 6 EC top-level classes. **COLLAB** [Leskovec *et al.*, 2005] is a scientific collaboration dataset, where nodes represent scientists and edges represent collaboration relation between two scientists; each graph is labeled to a physics field that the researcher belongs to. Statistics of the datasets are shown in Table 1.

We take three kinds of methods as baselines: (1) the kernel-based methods including **WL** [Morris *et al.*, 2019] subtree; (2) GNN-based methods including **GCN** [Kipf and Welling, 2017], **GIN** [Xu *et al.*, 2019], **GAT** [Veličković *et al.*, 2018], **GRAPHSAGE** [Hamilton *et al.*, 2017], **Set2Set** [Vinyals *et al.*, 2015] and **DGCNN** [Zhang *et al.*, 2018]; (3) hierarchical graph pooling methods including **DiffPool** [Ying *et al.*, 2018], **Graph U-Nets** [Gao and Ji, 2019] and **AttPool** [Huang *et al.*, 2019].

5.2 Implementation and Experiment Settings

We implement our MuchPool model with the PyTorch framework. The part of cluster formation in channel 2 follows the implementation of DiffPool [Ying *et al.*, 2018] and we make some adjustments according to our problem setting. The dimensions of node representation and graph representation are set as 64. The node retention ratio r is set as 0.5 for three channels in all layers. We adopt the mean pooling function to read out the graph representation. GCN is used as our backbone network. The first MuchPool GCN layer uses 2 GCN layers while the subsequent MuchPool GCN layers use only 1 GCN layer for aggregating information. A MLP consisting of two fully connected layers with 128 neurons is set to follow the final MuchPool GCN layer, followed by a softmax classifier. It takes the graph representation as input and outputs the categories probability to finish the prediction task.

We follow the experiment setting of the state-of-the-art model Graph U-Nets [Gao and Ji, 2019] and evaluate our model over 20 random seeds using 10-fold cross validation. A total of 200 testing results are used to report the average accuracy and standard deviation for each model on each dataset. We use Xavier normal distribution [Glorot and Bengio, 2010] for weight initialization, Adam optimizer [Kingma and Ba, 2015] to initialize our model and negative log-likelihood loss function is utilized to train our model. For all datasets, we train our model for 300 epochs and the batch size is set to 16 or 32 (depending on the graph size). The optimal hyperparameters are obtained by applying the strategy of grid search.

5.3 Performance Comparison

We compare the performance of our proposed MuchPool with baseline methods on the 6 benchmark datasets for the graph classification task, and the accuracy and standard deviation are reported in Table 2. First, one can observe that the performance of MuchPool is superior to its counterparts on five out of six benchmarks. To be specific, MuchPool improves the performance by 3.72% and 5.66% over the best baselines DiffPool and AttPool on ENZYMES and PROTEINS datasets respectively, which demonstrates the effectiveness of MuchPool. Next, MuchPool consistently outperforms GCN on all the datasets significantly, indicating the necessity of adding pooling modules in the learning process. This is because MuchPool can extract more useful graph topology information than the global pooling based GCN. By comparing with other GNN based models including GCN, GAT, GIN and GraphSAGE, both DiffPool and MuchPool achieve better performance. It reals that capturing coarse-grained structure is helpful for graph representation learning, especially for those graphs with obvious community structure. One can also see that DiffPool does not always show better results than local topology and feature based pooling methods AttPool and Graph U-Nets. This further verifies the effectiveness of combining the node features, local topology and global topology. Moreover, compared with AttPool, MuchPool scores relatively moderately on COLLAB datasets. This is because many collaboration graphs in COLLAB show only single-layer community structures. MuchPool uses three layers model to learn the graph representation, which is too complex and may lead to overfitting.

5.4 Ablation Study

In this subsection, we conduct an ablation study to verify that all three channels are helpful to graph pooling. We compare the complete MuchPool model with its variants by removing one of the three channels in it and keep other parts the same. For convenience, we name the MuchPool model without local topology, global topology and feature based pool channel as MuchPool_{NL}, MuchPool_{NG} and MuchPool_{NF}, respectively. As shown in Figure 3, the performances of the three variants are all inferior to MuchPool over the four datasets EN-

Methods	Datasets					
	ENZYMES	DD	PROTEINS	NCI1	NCI109	COLLAB
WL	40.97 ± 4.05	73.64 ± 3.84	75.75 ± 4.57	75.21 ± 1.64	72.40 ± 1.86	78.90 ± 1.90
GCN	50.00 ± 5.87	75.13 ± 4.14	74.75 ± 4.63	79.68 ± 2.05	78.05 ± 1.59	71.92 ± 3.24
GAT	51.00 ± 5.23	72.65 ± 3.18	77.37 ± 2.95	79.88 ± 0.88	79.93 ± 1.52	75.80 ± 1.60
GIN	31.11 ± 1.92	65.94 ± 1.87	68.17 ± 2.39	57.49 ± 0.73	56.62 ± 0.61	$\mathbf{80.20 \pm 1.90}$
GraphSAGE	53.33 ± 3.42	77.48 ± 3.20	76.73 ± 3.00	78.98 ± 1.84	77.27 ± 1.66	79.70 ± 1.70
Set2Set	38.00 ± 7.81	64.00 ± 6.82	70.26 ± 4.06	68.95 ± 2.51	66.37 ± 6.18	65.34 ± 6.44
DGCNN	49.67 ± 4.27	77.50 ± 2.87	78.08 ± 3.38	75.72 ± 1.77	74.02 ± 2.08	73.82 ± 1.74
Graph U-Nets	40.50 ± 3.88	82.51 ± 3.17	78.26 ± 4.52	69.73 ± 1.91	70.25 ± 2.09	69.94 ± 1.54
AttPool	50.17 ± 4.04	81.07 ± 3.21	79.52 ± 3.52	81.17 ± 2.24	80.23 ± 0.72	77.08 ± 2.26
DiffPool	62.83 ± 7.07	80.99 ± 2.98	77.62 ± 4.97	80.36 ± 1.56	78.51 ± 1.20	73.94 ± 3.28
MuchPool	$\mathbf{65.17 \pm 3.98}$	$\mathbf{85.06 \pm 3.34}$	$\mathbf{84.01 \pm 1.73}$	$\mathbf{81.29 \pm 1.31}$	$\mathbf{80.50 \pm 1.48}$	74.58 ± 2.63

Table 2: Mean accuracy (10 folds) and standard deviation on the 6 graph classification datasets. We use bold to highlight the best result.

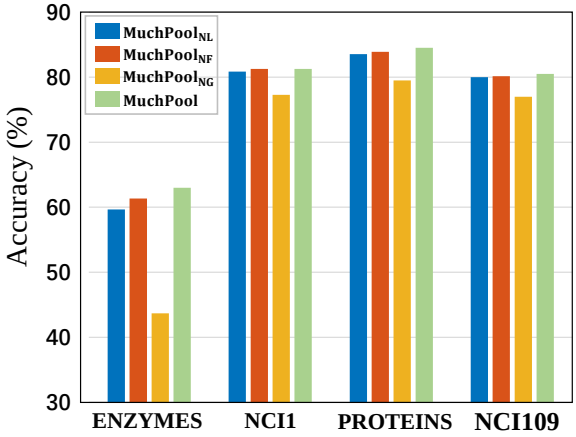

Figure 3: Comparison between MuchPool and its variations.

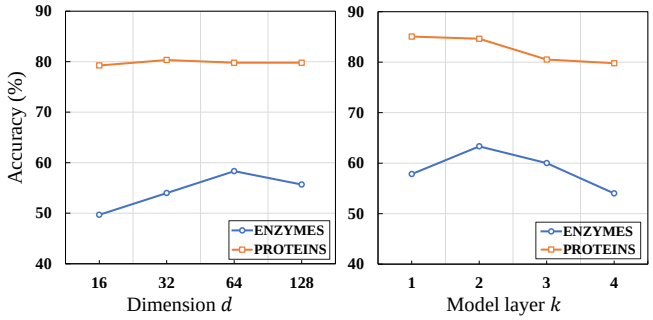

Figure 4: Classification accuracy curve on ENZYMNS and PROTEINS datasets with different values of d and k.

ZYMES, PROTEINS, NCI1 and NCI109. This demonstrates the effectiveness of the three channels. Especially, MuchPool outperforms MuchPool$_{NG}$ by 36.64%, 4.63%, 5.08% and 3.90% on four datasets, respectively. The result suggests that global topology is significantly important in some graphs (for instance, protein molecule graphs), and capturing the global structure is especially useful for the classifier to distinguish the graphs with fixed functional units. One can also see that the results of MuchPool$_{NL}$ and MuchPool$_{NF}$ are close to each other. It is probably because some nodes are important in terms of both local topology and node features, which leads to the large overlap between the selected nodes from channels 1 and 3.

5.5 Parameter Sensitivity Analysis

We further investigate the effects of some important hyperparameters on MuchPool. In detail, we study how the number of neural network layers k, graph representation dimension d will affect the graph classification performance. As shown in Figure 4, MuchPool achieves better performance when setting $d = 64$ and $k = 2$, respectively. One can also observe that with the dimension d increasing, the classification accuracy presents a slight increase trend accordingly. This is because different types of graphs have larger margins in higher-dimensional representation space, making it easier to distinguish their categories. Considering the neural network layer k, a larger k provides a more accurate classification result, but too large a k will hurt the performance.

6 Conclusion

In this paper, we propose a novel multi-channel graph pooling method MuchPool. MuchPool uses three channels to capture the local topology, the global topology and node features separately, and then uses a cross-channel convolution operation to integrate them. Based on MuchPool, we construct an end-to-end hierarchical graph representation learning model for graph classification. We evaluate our proposal on several graph classification benchmark datasets. The result shows that MuchPool achieves superior or comparable results compared with state-of-the-art methods.

Acknowledgement

This work is supported by the Fundamental Research Funds for the Central Universities (No.: NZ2020014) and Guanddong Basic and Applied Basic Research Foundatoin (2021A1515012239).

References

[Bruna *et al.*, 2014] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *In Proceedings of ICLR*, 2014.

[Diehl, 2019] Frederik Diehl. Edge contraction pooling for graph neural networks. *CoRR*, 2019.

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

[Gao and Ji, 2019] Hongyang Gao and Shuiwang Ji. Graph u-nets. *In Proceedings of ICML*, 2019.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[Glorot and Bengio, 2010] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS*, 2010.

[Gori *et al.*, 2005] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings of IJCNN*. IEEE, 2005.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceddings of NueIPS*, 2017.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of CVPR*, 2016.

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

[Huang *et al.*, 2019] Jingjia Huang, Zhangheng Li, Nannan Li, Shan Liu, and Ge Li. Attpool: Towards hierarchical feature representation in graph convolutional networks via attention mechanism. In *Proceedings of ICCV*, 2019.

[Karpathy *et al.*, 2014] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of CVPR*, 2014.

[Kingma and Ba, 2015] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2015.

[Kipf and Welling, 2017] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *In Proceedings of ICLR*, 2017.

[Lee *et al.*, 2019] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. *In Proceedings of ICML*, 2019.

[Leskovec *et al.*, 2005] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of SIGKDD*, 2005.

[Ma *et al.*, 2018] Jianxin Ma, Peng Cui, Xiao Wang, and Wenwu Zhu. Hierarchical taxonomy aware network embedding. In *Proceedings of SIGKDD*, 2018.

[Ma *et al.*, 2019] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In *Proceedings of SIGKDD*, 2019.

[Montavon *et al.*, 2012] Grégoire Montavon, Geneviève Orr, and Klaus-Robert Müller. *Neural networks: tricks of the trade*, volume 7700. springer, 2012.

[Morris *et al.*, 2019] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of AAAI*, 2019.

[Schlichtkrull *et al.*, 2018] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*. Springer, 2018.

[Shervashidze *et al.*, 2011] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[Song *et al.*, 2020] Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *Proceedings of the AAAI*, 2020.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *In Proceedings of ICLR*, 2018.

[Vinyals *et al.*, 2015] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *In Proceedings of ICLR*, 2015.

[Wang *et al.*, 2019] Xiaofeng Wang, Zhen Li, Mingjian Jiang, Shuang Wang, Shugang Zhang, and Zhiqiang Wei. Molecule property prediction based on spatial graph embedding. *Journal of chemical information and modeling*, 59(9):3817–3828, 2019.

[Wang *et al.*, 2020] Senzhang Wang, Jiannong Cao, and Philip Yu. Deep learning for spatio-temporal data mining: A survey. *IEEE TKDE*, 2020.

[Xu *et al.*, 2019] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *In Proceedings of ICLR*, 2019.

[Ying *et al.*, 2018] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceddings of NeurIPS*, 2018.

[Zhang *et al.*, 2018] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceddings of AAAI*, 2018.

[Zhang *et al.*, 2019] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. Hierarchical graph pooling with structure learning. *In Proceedings of NeurIPS*, 2019.