Multi-scale Graph Pooling Approach with Adaptive Key Subgraph for Graph Representations

Yiqin Lv
National University of Defense Technology
Changsha, China
lvyiqin98@nudt.edu.cn

Zhiliang Tian
National University of Defense Technology
Changsha, China
tianzhilianghit@gmail.com

Zheng Xie
National University of Defense Technology
Changsha, China
xiezheng81@nudt.edu.cn

Yiping Song*
National University of Defense Technology
Changsha, China
songyiping@nudt.edu.cn

ABSTRACT

The recent progress in graph representation learning boosts the development of many graph classification tasks, such as protein classification and social network classification. One of the mainstream approaches for graph representation learning is the hierarchical pooling method. It learns the graph representation by gradually reducing the scale of the graph, so it can be easily adapted to large-scale graphs. However, existing graph pooling methods discard the original graph structure during downsizing the graph, resulting in a lack of graph topological structure. In this paper, we propose a multi-scale graph neural network (MSGNN) model that not only retains the topological information of the graph but also maintains the key-subgraph for better interpretability. MSGNN gradually discards the unimportant nodes and retains the important subgraph structure during the iteration. The key subgraphs are first chosen by experience and then adaptively evolved to tailor specific graph structures for downstream tasks. The extensive experiments on seven datasets show that MSGNN improves the SOTA performance on graph classification and better retains key subgraphs.

CCS CONCEPTS

• **Information systems** → *Data mining*; • **Computing methodologies** → *Neural networks.*

KEYWORDS

Graph representation learning; Graph classification; Graph pooling; Graph neural network; Key-subgraph

ACM Reference Format:

Yiqin Lv, Zhiliang Tian, Zheng Xie, and Yiping Song. 2023. Multi-scale Graph Pooling Approach with Adaptive Key Subgraph for Graph Representations. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3614981

*Corresponding Author

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614981

1 INTRODUCTION

Graph neural networks have gained considerable popularity for modeling graph data, exhibiting remarkable performance in tasks such as node classification [26, 28, 45] and link prediction [33, 53]. However, to obtain an informative graph representation, the utilization of a pooling function becomes imperative. This function maps a collection of node representations into a condensed form. Current graph pooling methods can be grossly divided into two categories: *flat pooling* methods and *hierarchical pooling* methods. The flat pooling averages or sums all node representations in the given graph [1, 46]. These methods have limited ability to capture the graph structure, so there are some works such as DGCNN [54] proposing to read the graph in a meaningful and consistent order to keep the structure information. However, their flat architecture designs restrict their capability toward hierarchical pooling.

The hierarchical pooling retains the graph structure to some extent by recursively reducing the graph size, so it is adaptive to large graphs. One line is the Top-K pooling method, which learns a score vector to annotate the node importance and selects the top-k nodes. A representative work Graph U-net [13] builds an encoder-decoder framework with graph-pooling and unpooling operations, which ensures the downsized graph after pooling can better restore the original graph. SAGPool [22] exploits an attention mechanism to distinguish whether to retain or drop nodes. GIB [51] employs the information bottleneck (IB) [38] to detect informative subgraphs. Top-K pooling discards unimportant nodes during downsizing, so this method may lose partial information of the original graph.

Another line of hierarchical pooling does not discard any nodes but downsizes the graph by clustering nodes. Diffpool [50] employs GNNs to obtain node embeddings and learns an assignment matrix for differential clustering. ASAP [31] further improves the model with a sparse pooling operator so that it can preserve node and edge information effectively. However, graphs often contain diverse substructures that provide distinctive contributions to the overall graph representation. For instance, in protein-protein interaction graphs, specific substructures may signify important functions that greatly impact the prediction of the whole graph characteristics [55]. The hierarchical clustering method does not contain any key

Table 1: Differences between various hierarchical graph pooling methods

Methods	Each Iteration			Multi-scale Interaction
	Sample	Cluster	Evolution	
Graph U-net [13]	✓			
DiffPool [50]		✓		
GXN [23]	✓			✓(Features)
MSGNN(Ours)	✓	✓(Last)	✓	✓(Structures)

subgraphs of the original graph during downsizing, so it can hardly preserve the structural information of the graph.

There is another challenge besides the above issues: we cannot remedy the intermediate graph if we discard some important nodes by mistake. The reason is that the existing downsizing strategies for hierarchical pooling only aggregate nodes or directly discard nodes. Note that the discarding operation is not reversible. To make up for possible errors, GXN [23] leverages cross-feature interaction layers to enable the communication and fusion of multi-scale features. However, it cannot change the structure of the intermediate graph at each scale or interaction step.

In this paper, to address the above issues, we propose a Multi-scale Graph Neural Network (MSGNN) to downsize the graph with an adaptive graph pooling approach. MSGNN retains key subgraphs with their original structures, while not discarding remaining nodes, but clustering these unimportant nodes at the last layer to maintain the graph topology. To compensate for the irreversibility of discarding important nodes during downsizing, we propose an adaptive approach to amend the structure of the key-subgraph. By utilizing neural architecture search, we apply several operations on nodes and edges to evolve the key-subgraph, enhancing its adaptability for downstream tasks such as graph classification and node classification. Our MSGNN stands out for the following reasons: (1) It effectively preserves both key subgraphs through sampling and the original graph structure through clustering. (2) It allows adaptive modification of subgraph structures by performing evolutionary operations on nodes and edges at multiple scales. The comparison between MSGNN and other representative hierarchical pooling methods is shown in Table 1, with detailed descriptions provided in Appendix A. Our contributions are threefold,

- We propose a multi-scale graph pooling approach MSGNN, which preserves the graph structure during graph pooling to a large extent. It retains important subgraphs instead of important nodes and retains the information of unimportant nodes through clustering.
- The subgraphs can be evolved adaptively to cater to downstream tasks through operations on nodes or edges, which is a novel hierarchical pooling strategy for graph representation learning.
- Extensive experiments of graph classification demonstrate that our proposed MSGNN can achieve state-of-the-art performance on most biological and social network datasets.

2 PROBLEM DEFINITION

Given a graph, our task is to predict the class label of the graph. We denote a graph G as (A, X), where $A \in \{0,1\}^{n\times n}$ is the adjacent matrix, n is the number of nodes, $X \in \mathbb{R}^{n\times d}$ is the node feature matrix, and d is the dimension of feature. Given a set of graph data $\mathcal{G} = \{(G_1, y_1), (G_2, y_2), \ldots, (G_N, y_N)\}$, where y_i is the label of graph G_i, the goal of graph classification is to learn a mapping from graphs to labels, where the key is to learn graph representations.

3 METHOD

3.1 Overview

In this section, we present an overview of the proposed Multi-Scale Graph Neural Network (MSGNN) for learning graph representations. The MSGNN architecture is elaborated in Section 3.2, which involves a series of layers for hierarchical graph representation learning, including the graph convolution layer, graph pooling layer, clustering layer, and graph classification. In particular, to adaptively preserve key substructures during downsizing, we upgrade the graph pooling layer to an adaptive graph pooling layer, discussed in Section 3.3. Furthermore, to enable adaptive learning tailored specifically for downstream tasks, we study the training of the adaptive graph pooling layer in Section 3.4.

3.2 Multi-scale graph neural network

To learn graph representations, we propose a multi-scale graph neural network architecture, as shown in Figure 1. The architecture employs multiple iterations to capture graph information at different scales. In each iteration, graph convolution encodes node features through GCNs for information propagation. Then, graph pooling downsizes the graph to capture critical subgraph structures. These operations progressively reduce the graph size while preserving both graph features and substructures. After L iterations, we construct a clustering layer to preserve the graph topology. In the last, the representations at each scale are merged as the final representation for downstream tasks such as graph classification. The algorithm of the multi-scale graph neural network is reported in Appendix B.

Graph convolution. In the graph convolution layer, we use the graph convolutional networks (GCNs) [20] to learn node representations. The GCNs aggregate messages propagating between nodes and their neighbors and then update node representations. Specifically, for the l-th iteration, it normalizes the graph adjacent matrix A^{l-1} and transforms the node feature matrix X^{l-1} through the weight matrix W^{l-1} to get the hidden node representations H^l, which can be formulated as:

$$H^l = \sigma(\widetilde{A}^{l-1} X^{l-1} W^{l-1}), \tag{1}$$

where $\widetilde{A}^{l-1} = (\hat{D}^{l-1})^{-\frac{1}{2}} \hat{A}^{l-1} (\hat{D}^{l-1})^{-\frac{1}{2}}$ is the process of normalization, $\hat{A}^{l-1} = A^{l-1} + I$ is the adjacency matrix with self-connections, $\hat{D}^{l-1}$ is the diagonal degree matrix of $\hat{A}^{l-1}$, and $\sigma(\cdot)$ is an activation function. The graph convolution operator preserves the graph structure, resulting in the formation of a new graph (A^{l-1}, H^l). Note that Eq. (1) can be substituted into various formulas of GNNs.

Graph pooling. The pooling layer, the primary focus of this study, plays a key role in downsizing the graph while preserving the key subgraphs. For the l-th iteration, the input is the graph (A^{l-1}, H^l) obtained by the graph convolution. After downsizing, the output is a smaller graph (A^l, X^l) that contains key subgraphs.

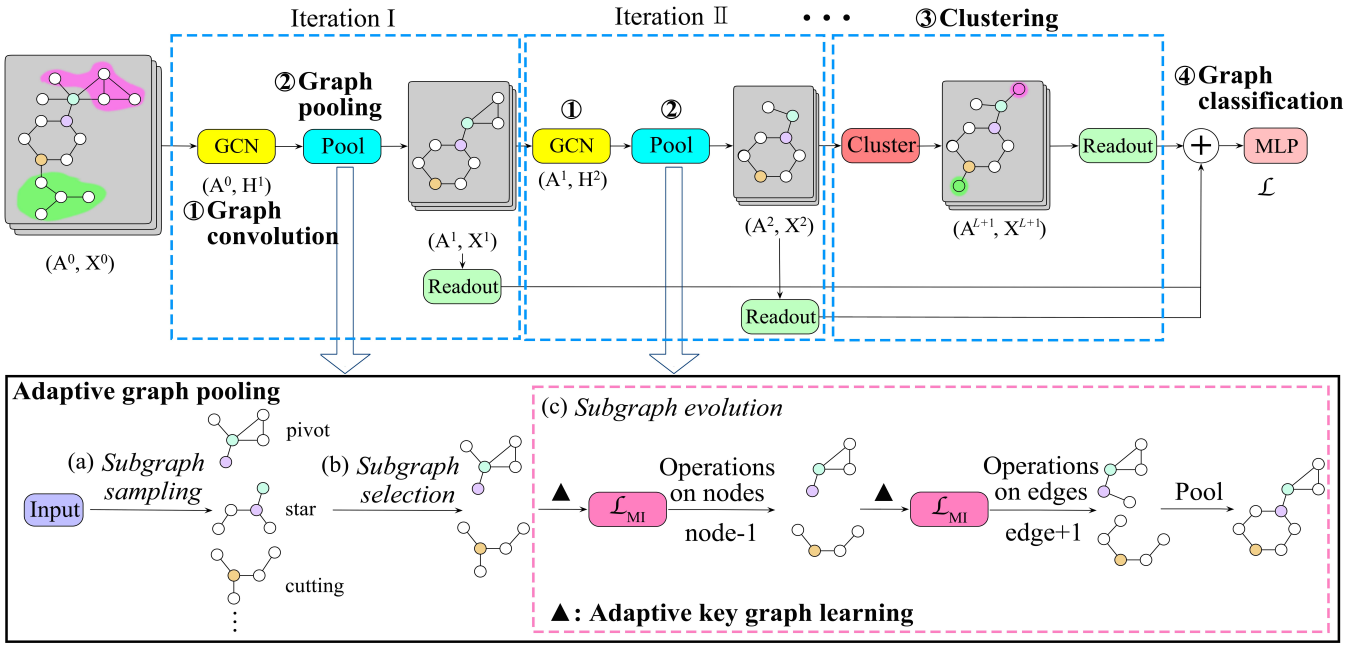

Figure 1: (Top) An illustration of the proposed MSGNN architecture. Given a graph input, MSGNN predicts the graph category. Each iteration consists of a graph convolution layer and a graph pooling layer. The clustering layer maintains the graph topology and the MLP is used for graph classification. (Bottom) The adaptive graph pooling layer contains subgraph sampling which utilizes prior knowledge to sample subgraphs, subgraph selection which selects striking subgraphs by the top-k method, and subgraph evolution which evolves subgraphs through operations on nodes and edges by adaptive key graph learning.

To adaptively preserve these key subgraphs, we upgrade the graph pooling to an adaptive graph pooling in Section 3.3.

Clustering. We construct a clustering layer at the end to preserve the graph topology, since some nodes may be lost during downsizing, resulting in the incomplete graph topology. After conducting the GCNs and pooling for L iterations, the pooled graph (A^L, X^L) contains important structural information. We regard all nodes in the graph as important nodes, denoted as NODE_{imp}. The remaining nodes in the original graph (A^0, H^1) are considered unimportant nodes, denoted as $\text{NODE}_{\text{unimp}}$. To preserve the information of unimportant nodes while preserving the graph topology, we utilize a structural clustering algorithm for networks (SCAN) [48] to cluster $\text{NODE}_{\text{unimp}}$. A cluster is regarded as a new node, and a new edge links the new node and an important node. The new node representation is the mean of node representations in the cluster. Thus, we get a graph (A^{L+1}, X^{L+1}) in the clustering layer.

Graph classification. To obtain the graph-level representation R^l, we leverage a graph readout function to summarize node representations into a vector: $R^l = \text{READOUT}(\{x_i^l\}_{i=1}^{n^l})$, where x_i^l is the i-th row of X^l and n^l is the number of nodes in the l-th iteration. The READOUT function can be any permutation-invariant function, and we apply a simple sum strategy here. To merge the graph representation of each scale, we concatenate them as the final graph representation $R = \|_{l=1}^{L+1} R^l$, where $\|$ is the concatenation. For the graph classification task, we feed the representation into an MLP classifier to predict the graph class label $\hat{y}$. The cross-entropy loss between the predictions and ground-truth graph labels is:

$$\mathcal{L} = -\sum_{i=1}^{N} y_i \log \hat{y}_i. \tag{2}$$

3.3 Adaptive graph pooling

To adaptively preserve key subgraphs, we construct an adaptive graph pooling. This is accomplished through the design of three essential components: subgraph sampling, subgraph selection, and subgraph evolution.

Subgraph sampling. To downsize the graph while preserving crucial substructures, we sample several candidate representative subgraphs based on prior knowledge. Specifically, we choose three types of nodes: pivot nodes, star nodes, and cutting nodes (see the bottom left of Figure 1). The three types of nodes have been proven to be important elements in graph analysis [56].

- A *pivot node* is a high-degree node whose one-step neighbors have at least one interconnection. It plays a crucial role in maintaining graph connectivity.
- A *star node* is a node whose one-step neighbors are not directly connected to each other. This structural pattern is common in various graph types.
- A *cutting node* is a node that connects two communities. If it is removed, the graph will not be connected.

We uniformly sample pivot nodes in the first 5% of degrees, star nodes in the first 50% of degrees, and all cutting nodes, where the sampling rate are the same as [56]. Subsequently, we employ the breadth-first search (BFS) algorithm to extract subgraphs over

these selected nodes. The number of nodes in each subgraph is limited to s, which is set to ensure the sampled subgraph contains fundamental functional building blocks. We illustrate the specific setting of s in Section 4.1. As a result, we obtain a set of subgraphs denoted as $\mathcal{G}^l_{\text{sub(sam)}} = \{g^l_1, g^l_2, \ldots, g^l_{n_1}\}$, where n_1 is the number of sampled nodes and l represents the l-th iteration.

Subgraph selection. If all sampled subgraphs are preserved, they may be similar in size to the original graph without reducing the scale of the graph. To tackle this issue, we utilize graph features to select a portion of subgraphs with a pooling ratio $r \in (0, 1]$. This step adopts the top-k method [13] to score and select important subgraphs. We first calculate subgraph representations $Z^l = \{z^l_1, z^l_2, \ldots, z^l_{n_1}\}$, where the representation z^l_i of subgraph g^l_i is obtained by aggregating the representations of its constituent nodes. Next, we project the subgraph representations Z^l via a trainable vector p^l to a one-dimensional vector v^l, where each element serves as the importance score of each subgraph. Then, we rank these scores in a descending order and select the top $n_2 = \lceil r \cdot n_1 \rceil$ subgraphs, where r is the pooling ratio for subgraphs. The procedure is denoted as:

$$v^l = Z^l p^l / \|p^l\|_2, \quad idx = rank(v^l, n_2), \tag{3}$$

where $rank(v^l, n_2)$ denotes the subgraph ranking and returns indices idx of the n_2-largest values in v^l. We use idx to extract the corresponding subgraphs and create a new set of subgraphs $\mathcal{G}^l_{\text{sub(sel)}}$ and their representations Z^l_{sel}.

Subgraph evolution. To handle redundant or missing structures in selected subgraphs, we propose subgraph evolution to adaptively modify the subgraph structures. Specifically, we perform several learnable operations on nodes and edges within these subgraphs. This evolution of the subgraph structures is closely related to downstream tasks, thus enhancing the adaptability of the pooling strategy.

We adopt subgraph evolution to overcome two underlying problems. Firstly, the subgraph structures obtained in above steps are irreversible. Once an inappropriate subgraph is acquired at an intermediate scale, there is no chance to modify it later. Secondly, the pooling strategy used in subgraph selection is universal, while the downstream tasks are various. To achieve the structural evolution, we perform operations on nodes and edges in selected subgraphs.

Operations on nodes. We consider three types of operations on nodes:

- Adding nodes, which rectifies the incompleteness of missing subgraph structures;
- Removing nodes, which facilitates the pruning of redundant subgraph structures;
- No operation, which keeps current subgraph structures.

It is worth noting that the criterion for adding and removing nodes is to ensure the evolved subgraph is as connected as possible so that subgraph structures can be preserved. We carefully select neighboring nodes within the subgraph during node addition, aiming to preserve the connectivity of the graph structure. Concretely, we form an operation set O_{node} with five operations: adding one node, adding two nodes, removing one node, removing two nodes, and no operation. To adaptively evolve subgraphs, we need to choose an optimal operation to satisfy the downstream task and then perform the operation o^l_{node} to obtain the set of evolved subgraphs $\mathcal{G}^l_{\text{sub(node)}}$. The adaptive learning is detailed in Section 3.4.

Operations on edges. Similarly, we form an operation set O_{edge} with five operations: adding one edge, adding two edges, removing one edge, removing two edges, and no operation. Note that the act of adding or removing edges does not necessarily lead to the addition or removal of nodes. Consequently, these operations differ from the operations on nodes in terms of their effects and characteristics. Upon identifying the optimal edge operation o^l_{edge}, it is executed on subgraphs $\mathcal{G}^l_{\text{sub(node)}}$, resulting in a set of evolved subgraphs $\mathcal{G}^l_{\text{sub(edge)}}$. We denote it as $\mathcal{G}^l_{\text{sub(evo)}}$, representing the final evolved subgraphs. Note that the order in which evolution is conducted on nodes and edges can be changed.

Following the evolution process, a new pooled graph (A^l, X^l) is generated. The indices of all nodes within the evolved subgraphs are collected and denoted as ind, representing the indices of the selected nodes for the new graph. Subsequently, row and/or column extraction operations are performed to construct the adjacency matrix and feature matrix of the newly formed graph:

$$A^l = A^{l-1}(ind, ind), \quad X^l = H^l(ind, :). \tag{4}$$

3.4 Adaptive key graph learning

To search for the optimal operation, we borrow the idea from neural architecture search [25, 57]. As to the search algorithm, most of the existing methods use the RL (Reinforcement Learning) and EA (Evolutionary Algorithm) based methods to select architectures from the search space [21, 30, 32, 57]. However, the RL and EA based algorithms need thousands of evaluations which are computationally expensive [25]. To solve the efficiency problem, we adopt a differentiable search algorithm [25] to relax the discrete search space into a continuous one by mixing all candidate operations. Technically, the discrete selection of operations is relaxed by a weighted summation of all possible operations as $\sum_{i=1}^{|O_{\text{node}}|} \alpha_i o_i$, where $\alpha_i \in (0, 1)$ is the weight of the i-th operation o_i in set O_{node}. The α_i is generated by a softmax function.

The operation o_i cannot be directly calculated, so we substitute it with the representation of the subgraph after the corresponding operation. For each subgraph after the operation o_i, which is called the evolved graph, its representation Z^l_{evo} is updated by adding representations of new nodes or subtracting representations of removed nodes. Then, all the evolved subgraphs' representations are further aggregated through a CNN. It can be seen that o_i is an entire graph representation, so we take the weighted summation $\sum_{i=1}^{|O_{\text{node}}|} \alpha_i o_i$ as the representation of the graph after evolving over nodes, denoted as R^l_{node}.

We define a criterion to optimize the above search problem. To minimize the differences between the original graph and evolved graph, we design the loss as the mutual information (MI) between R^l_{node} and the original graph representation R^0. The MI measures the mutual dependence between two vectors, reflecting whether the evolved graph captures the structure of the original graph. It is modeled through a discriminator $\mathcal{D}$ to determine whether two vectors are from the same graph. Moreover, the self-supervised MI

Algorithm 1: SubGraphEvo(·) function

Input: Selected subgraphs $\mathcal{G}^l_{\text{sub(sel)}}$;
Output: Evolved subgraphs $\mathcal{G}^l_{\text{sub(evo)}}$
// Search optimal operation on nodes
Define operations on nodes: add one node, add two nodes, remove one node, remove two nodes, and no operations;
Form the operation set (known as the search space) O_{node};
$R^l_{\text{node}} = \sum_{i=1}^{|O_{\text{node}}|} \alpha_i o_i$;
Calculate the MI loss $\mathcal{L}_{MI} \leftarrow$ Eq.(5)
Maximize $\mathcal{L}_{MI}$ through a bi-level optimization $\leftarrow$ Eq.(6)
Perform the optimal operation on nodes $\leftarrow$ Eq.(7)
// Search optimal operation on edges
Define operations on edges: add one edge, add two edges, remove one edge, remove two edges, and no operations;
Form the operation set O_{edge};
Implement the same process as nodes to obtain the optimal operation o^l_{edge};
Perform the optimal operation on edges
$\mathcal{G}^l_{\text{sub(evo)}} = \mathcal{G}^l_{\text{sub(edge)}}$.

mechanism is contrastive and the negative samples $\mathcal{H}$ are obtained by a corruption function [40]. Therefore, the MI objective can be defined as a binary cross-entropy (BCE) loss:

$$\mathcal{L}_{MI} = \frac{1}{1+|\mathcal{H}|}(\log\sigma(\mathcal{D}(R^0, R^l_{\text{node}})) + \sum_{\widetilde{H}\in\mathcal{H}} \mathbb{E}_{R^0,\widetilde{H}}[\log(1-\sigma(\mathcal{D}(R^0,\widetilde{H})))]). \tag{5}$$

The larger the MI is, the closer the evolved graph is to the original graph. To maximize the loss, we utilize a bi-level optimization approach to learn the search parameters α and CNN model parameters w (e.g. weights of convolution filters). The search parameters α are considered as the upper-level variable and w as the lower-level variable. The bi-level optimization maximizes the training loss $\mathcal{L}_{MI\text{train}}(w,\alpha)$ to get the optimal model parameters $w^*(\alpha)$ and then maximizes the validation loss $\mathcal{L}_{MI\text{val}}(w^*(\alpha),\alpha)$ to get the optimal search parameters α^*:

$$w^*(\alpha) = \text{argmax}\mathcal{L}_{MI\text{train}}(w,\alpha); \quad \alpha^* = \text{argmax}\mathcal{L}_{MI\text{val}}(w^*(\alpha),\alpha). \tag{6}$$

We choose the corresponding operation with the largest weight in α^* as the optimal operation on nodes and then perform the operation o^l_{node} to obtain the set of evolved subgraphs:

$$\alpha^l_{\text{node}} = \max\alpha^*; \quad \mathcal{G}^l_{\text{sub(node)}} = o^l_{\text{node}}(\mathcal{G}^l_{\text{sub(sel)}}). \tag{7}$$

The search for edge operations follows a similar approach to that for node operations. The key distinction lies in the encoding of evolved subgraphs, which is not achieved through a simple summation of node representations. Rather, a graph neural network is employed to propagate messages along edges, thereby emphasizing the distinctive features associated with edges. The detailed implementation of the subgraph evolution function, denoted as SubGraphEvo(·), is outlined in Algorithm 1.

The training within the subgraph evolution is step-by-step. Specifically, we initiate the training by optimizing the MI objective with respect to the nodes, which enables the identification of the optimal operation and the consequent subgraphs $\mathcal{G}^l_{\text{sub(node)}}$. Subsequently, we proceed to optimize the MI objective pertaining to the edges, leading to the acquisition of the evolved subgraphs $\mathcal{G}^l_{\text{sub(edge)}}$. Moreover, the training is truncated to the training of MSGNN.

4 EXPERIMENTS

4.1 Experimental setups

Datasets. To test our MSGNN, we conduct extensive experiments for graph classification on seven datasets. We use protein datasets D&D [10], PROTEINS [6], biological datasets NCI1, NCI109 [42], MUTAG [18], and social datasets IMDB-B and IMDB-M [49].

Baselines. To verify the effectiveness of MSGNN, we compare it with state-of-the-art methods, including flat pooling (Set2Set [41], DGCNN [54]), hierarchical pooling (DiffPool [50], Graph U-net [13], SAGPool [22], ASAP [31], GIB [51], GXN [23], PAS [44], GMT [2]) and graph kernel-based methods (SHORTPATH [5], WL [34]).

Model configuration. We perform 10-fold cross validation to evaluate the model performance based on hyperparameters and report the averaged test accuracy and the standard deviations over 10 folds. The multi-scale graph neural network used in the experiments consists of three graph convolution and pooling layers, i.e. $L = 3$. In graph encoding, the initial node features are one-hot vectors of node categories. We adopt the GCN [20] with 1 layer, 128 hidden units, and Exponential Linear Unit (ELU) activation function as the graph encoder. In subgraph sampling, the subgraph size s is set according to the average size of graphs in a dataset. For datasets D&D, PROTEINS, NCI1, NCI109, MUTAG, IMDB-B and IMDB-M, s is set to 15, 10, 10, 10, 5, 5 and 5. In subgraph selection, among the three iterations, the subgraph pooling ratio r is set to 0.7, 0.6, and 0.5 respectively. In subgraph evolution, we adopt the CNN with 1 layer, 2 kernel size, and Rectified Linear Unit (RELU) activation function. The learning rate for the search optimization is 0.08, the dropout is 0.3. In the training of graph classification, we adopt Adam [19] as the optimizer, the learning rate is initialized as 0.001, and the weight decay is 0.3. A detailed description of datasets, baselines and configuration is provided in Appendix C, D, and E.

4.2 Overall performance

Table 2 shows the accuracies of MSGNN and baselines for graph classification on seven datasets, and we use bold to highlight wins. We observe that MSGNN achieves state-of-the-art performance on all datasets except NCI1, where the accuracy of MSGNN is only 0.6% lower than that of the best baseline. Our approach also demonstrates the standard deviation is in a moderate range. Moreover, the datasets cover diverse domains, which reveals that MSGNN yields great performance in different domains.

The **flat pooling** approaches Set2Set and DGCNN are surpassed by most of the hierarchical pooling methods on the D&D dataset. This is because the graph size in D&D is large and flat pooling methods ignore the multi-scale structure information in the graph,

Table 2: Graph classification accuracies (%) of various methods on seven datasets. Average accuracy and standard deviation are reported. Bold results indicate the best performance.

Methods		Datasets						
		D&D	PROTEINS	NCI1	NCI109	MUTAG	IMDB-B	IMDB-M
Flat	Set2Set	71.60 ± 0.87	72.16 ± 0.43	66.97 ± 0.74	61.04 ± 2.69	77.69 ± 0.55	72.90 ± 0.75	50.19 ± 0.39
	DGCNN	71.87 ± 0.96	73.91 ± 0.72	68.74 ± 1.07	68.59 ± 0.67	77.95 ± 0.67	72.12 ± 1.12	48.18 ± 0.83
Hierarchical	DiffPool	66.95 ± 2.41	68.20 ± 2.02	62.32 ± 1.90	61.98 ± 1.98	80.44 ± 0.82	73.14 ± 0.70	51.31 ± 0.72
	Graph U-net	75.01 ± 0.86	71.10 ± 0.90	67.02 ± 2.25	66.12 ± 1.60	79.14 ± 0.76	71.58 ± 0.95	48.59 ± 0.72
	SAGPool	76.45 ± 0.97	71.86 ± 0.97	67.45 ± 1.11	67.86 ± 1.41	79.18 ± 0.82	72.55 ± 1.28	50.23 ± 0.44
	ASAP	76.87 ± 0.70	74.19 ± 0.79	71.48 ± 0.42	70.07 ± 0.55	80.12 ± 0.88	72.81 ± 0.50	50.78 ± 0.75
	GIB	74.70 ± 0.04	74.90 ± 0.05	–	–	83.90 ± 0.06	73.70 ± 0.07	–
	GXN	82.68 ± 4.10	79.91 ± 4.10	**77.49 ± 1.81**	76.73 ± 1.62	80.19 ± 1.02	78.60 ± 2.30	55.20 ± 2.50
	PAS	78.96 ± 0.04	76.64 ± 0.03	–	76.84 ± 0.03	–	75.10 ± 0.05	52.20 ± 0.04
	GMT	78.72 ± 0.59	75.09 ± 0.59	–	–	83.44 ± 1.33	73.48 ± 0.76	50.66 ± 0.82
Graph kernel	SHORTPATH	78.72 ± 3.89	75.71 ± 2.73	67.44 ± 2.76	67.72 ± 2.28	71.63 ± 2.19	59.20 ± 0.41	40.50 ± 0.37
	WL	76.44 ± 2.35	76.16 ± 3.99	76.65 ± 1.99	76.19 ± 2.45	80.32 ± 1.71	72.50 ± 0.52	51.50 ± 0.49
	MSGNN (Ours)	**87.29 ± 2.17**	**88.39 ± 2.35**	76.89 ± 3.23	**77.72 ± 2.16**	**94.74 ± 2.05**	**79.21 ± 1.07**	**57.38 ± 1.09**

Table 3: Ablation study of several variants (%). Average accuracy and standard deviation are reported. Bold results indicate the best performance. The results show the contribution of different components in the proposed method.

Methods	Datasets						
	D&D	PROTEINS	NCI1	NCI109	MUTAG	IMDB-B	IMDB-M
w/o evolution	85.15 ± 0.03	79.07 ± 0.04	71.65 ± 0.02	70.90 ± 0.01	90.99 ± 0.07	72.63 ± 0.23	50.62 ± 0.51
w/o clustering	82.20 ± 2.52	86.61 ± 2.26	72.67 ± 2.79	71.95 ± 1.93	90.33 ± 1.65	71.59 ± 1.07	52.48 ± 1.16
w/o both	76.48 ± 0.62	72.07 ± 0.81	69.10 ± 1.05	68.26 ± 0.89	84.62 ± 0.54	71.41 ± 0.89	49.61 ± 0.75
MSGNN-NoMI	85.79 ± 2.09	87.03 ± 2.14	74.25 ± 2.72	73.61 ± 2.05	91.72 ± 1.39	76.88 ± 0.72	55.48 ± 0.61
MSGNN (Ours)	**87.29 ± 2.17**	**88.39 ± 2.35**	**76.89 ± 3.23**	**77.72 ± 2.16**	**94.74 ± 2.05**	**79.21 ± 1.07**	**57.38 ± 1.09**

which plays a vital role in predicting large graph labels. For small datasets NCI1 and NCI109, hierarchical pooling methods are not always superior to the flat ones, because the information of small graphs may be lost during the hierarchical pooling process.

Among **hierarchical pooling** methods, compared with selection-based baselines (e.g., Graph U-net and SAGPool), MSGNN achieves better results. It shows that it is more effective to preserve important substructures while retaining the information of unimportant nodes than to keep only several nodes. Compared with clustering-based methods (e.g., DiffPool and ASAP), MSGNN performs better and the accuracy is even 5% higher than DiffPool on all datasets, which demonstrates the positive effect of key subgraph structures. Although the GXN on NCI1 achieves the best performance, the result of MSGNN is slightly lower. Moreover, the average accuracy of MSGNN improves by 4.61%, 8.48%, and 14.55% on D&D, PROTEINS, and MUTAG datasets, which indicates that multi-scale structures obtained by our method introduce more information than multi-scale features obtained by GXN. Compared with the GIB, PAS, and GMT, MSGNN consistently outperforms them on given datasets.

Compared with **graph kernel-based** methods, MSGNN significantly improves the accuracy by more than 8% on the D&D, PROTEINS and MUTAG datasets. Graph kernel-based methods mainly follow a two-stage learning framework in which graph representation learning and classification are processed respectively. It is difficult to optimize them jointly and might result in sub-optimal performance, whereas our MSGNN simplifies the training process in an end-to-end manner. Moreover, SHORTPATH and WL employ paths and subtrees as graph kernels, relying on domain-specific knowledge and incurring high computational complexity for large-scale graphs. In contrast, our MSGNN utilizes hierarchical pooling learning to effectively address these challenges.

4.3 Model analysis

The high classification accuracy and slight standard deviation of MSGNN on most datasets demonstrate the superiority and stability of our method. We further conduct a detailed analysis.

Ablation study. To demonstrate the effectiveness of the subgraph evolution and clustering, we conduct an ablation study. Table 3 shows the experimental results, in which "w/o evolution" and "w/o clustering" refer to removing subgraph evolution and clustering steps from MSGNN separately while keeping other parts unchanged, "w/o both" indicates the removal of both steps.

As shown, the performance has been affected with the removal of the subgraph evolution. It highlights the necessity of achieving subgraph reversibility during pooling, validating our underlying motivation. The performance gain of MSGNN with respect to "w/o clustering" verifies the necessity of clustering unimportant nodes. The performance improvement is especially obvious on D&D, NCI109, and MUTAG datasets, which are 5.09%, 5.77%, and 4.41%, respectively. The results of "w/o both" verify that our method

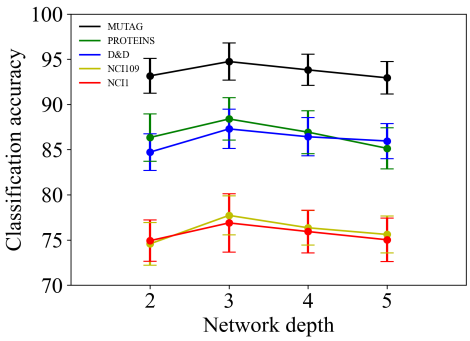

Figure 2: Performance with different network depths. The figure displays a vertical error bar at each data point, the error bars represent the standard deviation, and the data points represent the average accuracy.

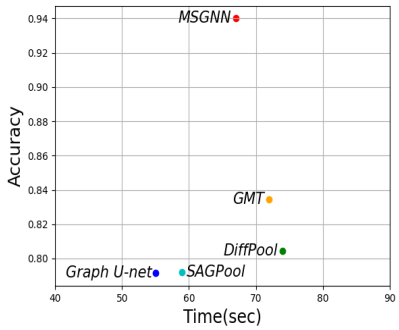

Figure 3: Graph classification results of baselines, reported with accuracy over training time on MUTAG.

incorporating with the carefully designed subgraph evolution and clustering strategies can effectively exploit valuable information to enhance the accuracy of graph classification. To demonstrate the importance of each part, we conduct a case study of the ablation experiment, see Appendix F.

Self-supervised MI study. To evaluate the power of the self-supervised MI between representations of the original graph and evolved graph in the step of subgraph evolution, we compare MSGNN to its variant MSGNN-NoMI. The variant removes the self-supervised MI mechanism, that is, it does not determine how to evolve subgraphs by optimizing the MI loss, but directly by the final classification loss. The results are reported in Table 3. We can observe that the experimental results of MSGNN consistently outperform the variant MSGNN-NoMI on all datasets, providing evidence for the informative self-supervision facilitated by MI enhancement in subgraph evolution. Furthermore, the performance improvement suggests that the self-supervised MI-guided evolution enables key substructures to exhibit enhanced adaptability for downstream graph classification tasks.

Network depth study. To examine the impact of network depth on the MSGNN performance, we use different network depths on the graph classification tasks. Figure 2 indicates the influence of classification accuracy on five datasets with different network depths. We can observe that increasing the network depth from 2 to 3 improves performance by capturing more structural information. When the depth is 3, we obtain the best classification performance. However, further increasing the depth gradually decreases classification accuracy due to the over-smoothing problem in deep networks. Additionally, the lengths of error bars on different datasets are almost similar, illustrating the stability of MSGNN.

Adaptive study of subgraph evolution. To verify that subgraphs can evolve adaptively for different downstream tasks, we conduct node classification experiments to compare the subgraph operations on node classification tasks and graph classification tasks, as shown in Table 4. To achieve node classification, we add unpooling layers to keep all nodes. Other detail implementations of node classification can be found in Appendix G.

Table 4 shows the differences in operations when evolving subgraphs at each iteration on the two tasks. For node classification, MSGNN prefers to remove nodes and edges to simplify the subgraph structures. The small and compact subgraph structures retain useful information for easy propagation and discard redundant information. As a result, we could obtain expressive node representations that contribute to node classification. For graph classification, MSGNN prefers to add nodes and edges to enrich the subgraph structures. The big and plentiful subgraph structures largely preserve the information of the entire graph, resulting in rich graph representations. Thus, achieving superior performance in graph classification becomes more attainable. The differences reflect that the evolved subgraph structures vary by downstream tasks, demonstrating the adaptability of subgraph evolution.

Efficiency of MSGNN. To demonstrate the efficiency of MSGNN, we compare it with Graph U-net [13], SAGPool [22], DiffPool [50], and GMT [2], for graph classification accuracy on MUTAG, shown in Figure 3. MSGNN achieves significantly higher classification accuracy than baselines, while its training time per epoch falls between the training time of the baselines. MSGNN incorporates subgraph sampling, selection, and evolution, with the evolution requiring training to determine operations on nodes and edges. However, MSGNN only incurs a slightly higher training time compared to Graph U-net, which focuses solely on selecting important nodes. This indicates that our improved results do not come at the expense of training time, and further MSGNN is efficient.

Interpretability. In this subsection, we study the power of MSGNN to find subgraphs with prominent patterns and provide insightful interpretations for the formation of pooled graphs. Figure 4 shows the pooling results on the MUTAG dataset. We utilize networkx and matplotlib in python packages to visualize the pooling results of the MSGNN. We randomly sample a graph from MUTAG dataset, which contains 25 nodes. We build a 3-layer graph neural network with subgraph pooling ratios set to 0.7, 0.6, and 0.5, and then obtain three pooled graphs with nodes as 14, 12, and 11 respectively. It can be seen that MSGNN is capable of preserving several sub-structures at multiple scales during pooling.

Furthermore, we evaluate the interpretability of MSGNN based on chemical domain knowledge. The graphs in MUTAG are labeled based on the mutagenic effects on bacteria, including mutagenic and non-mutagenic. The graph shown in Figure 4 (a) is labeled as mutagenic. According to expertise, carbon rings and NO2 groups tend to be critical for identifying mutagenic molecules [8]. We can observe that graphs in Figure 4 (b) and (c) totally preserve key subgraphs, and the graph in (d) retains partial structures. In the design

Table 4: Differences in operations when evolving subgraphs at each iteration on node classification and graph classification.

Tasks	Iterations	Operations on nodes					Operations on edges				
		node+1	node+2	node-1	node-2	no	edge+1	edge+2	edge-1	edge-2	no
Node classification	1			✓					✓		
	2					✓			✓		
	3					✓				✓	
Graph classification	1	✓							✓		
	2	✓					✓				
	3		✓								✓

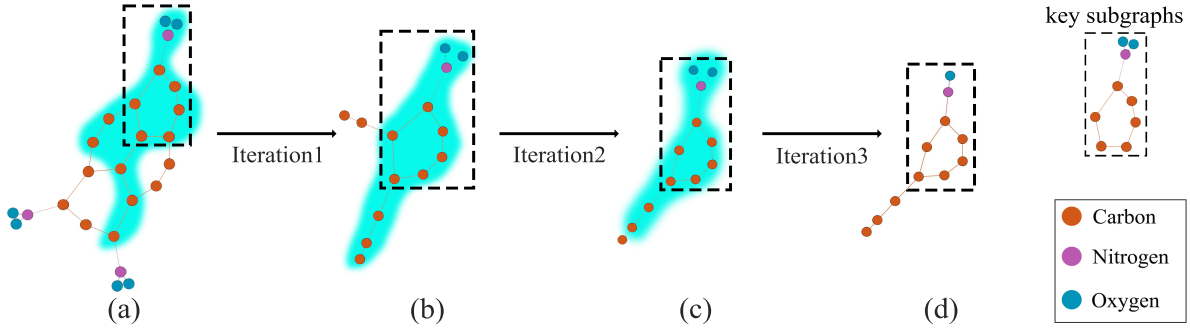

Figure 4: Result visualization on MUTAG. (a) is a mutagenic molecular graph where nodes represent atoms and edges represent chemical bonds. (b), (c), and (d) are pooled graphs obtained by MSGNN. The shaded part of blue in (a) corresponds to (b), which represents the result of pooling. The following case is similar to this. Professionally, carbon rings and NO2 groups tend to be key structures for identifying mutagenic molecules. Inside each dashed box is a complete or partial key subgraph.

of MSGNN, all three of them play a role in graph classification, leading to a correct result. It shows that MSGNN is able to preserve the relatively reasonable topology of the original graph during pooling and has great promise to provide sufficient interpretability.

5 RELATED WORK

5.1 Graph neural networks

Recently, graph neural networks (GNNs) have become a hot research topic and can be applied to node classification [20, 26, 28, 45], link prediction[33, 53], and graph classification [4, 54]. Most of GNNs follow a message passing framework to embed nodes or graphs into a low-dimensional and continuous space [14]. In the framework, nodes recursively aggregate messages from neighbor nodes through edges to update their representations. Kipf and Welling [20] propose GCNs to aggregate and transform node features, showing promising results on classification tasks. Hamilton et al. [15] extend the model to large graphs by using fixed-size node sampling. Veličković et al. [39] introduce the attention mechanism to assign different weights to neighbor nodes. Moreover, other works also make incremental improvements [36, 47]. However, these models aim to generate node-level representations, which may not be suitable for graph classification task.

5.2 Graph pooling

Graph pooling is an essential component of graph neural networks. It aims to reduce graph size by downsampling nodes, thereby learning graph-level representations. It broadly falls into two categories: flat pooling and hierarchical pooling.

Flat pooling. This approach learns flat graph representations by simply summing up or averaging all node representations in the last layer [7, 12, 29]. Set2Set [41] employs Long Short-Term Memory (LSTM) [16] to aggregate node features for the pooling operation. However, this way has limitations in capturing graph structure. Zhang et al. [54] propose the DGCNN model, which employs the sortPooling layer to maintain the graph structure by sorting node feature descriptors. These methods are straightforward but fail to capture the potential hierarchical information in real-world graphs.

Hierarchical pooling. This method progressively reduces the entire graph into a coarsened graph [13, 17, 50, 55]. They can be roughly divided into two categories: selection-based methods and clustering-based methods. Selection-based methods sample a set of nodes according to various node score functions. Graph U-net [13] scores nodes with a learnable projection vector and samples high-scoring nodes with a pooling ratio. SAGPool [22] considers the graph structure and employs GNNs to score nodes. GIB [51] utilizes IB [38] to recognize the maximally informative yet compressive subgraphs. Clustering-based methods group nodes and coarsen the graph based on a cluster assignment matrix. [9, 36] use deterministic graph clustering algorithms to achieve pooling. [31, 50, 52] use GNNs to compute assignment functions that assign nodes to different clusters in the next layer. GMT [2] treats the graph pooling problem as a multiset encoding problem and can be extended to the clustering-based methods. Furthermore, there are other studies that fall outside the aforementioned categories. GXN [23] and MSGCNN [43] concentrate on feature aggregation across multiple scales. SUGAR [37] adopts a reinforcement learning

algorithm to optimize the pooling ratio. MuchPool [11] combines selection-based and clustering-based methods. PAS [44] searches for adaptive pooling architectures through neural architecture search.

In addition to learning graph representations through graph pooling for graph classification, there is also a category of graph classification methods based on similarity calculation [3, 24, 34]. The implicit assumption of this approach is that if two graphs are similar, they belong to the same class. Therefore, the key is to calculate the similarity between graphs. Graph kernel-based methods decompose graphs into substructures, such as paths [5, 27], subgraphs [35], and subtrees [34], and then compare these substructures from different graphs to determine how similar they are. Graph match-based methods [3, 24] learn the mapping relationship between nodes of two graphs or use graph edit distance to measure similarity. They are inflexible and computationally expensive.

6 CONCLUSION

In this paper, we propose a multi-scale graph neural network (MSGNN) for graph representation learning. MSGNN reduces the graph size by completely reserving important substructure and roughly reserving unimportant information. MSGNN retains key subgraphs to preserve the details of the graph structure, and clusters unimportant nodes to maintain the graph topology. Meantime, key subgraphs are adaptively generated by subgraph evolution algorithm to cater to downstream tasks. In the experiments, we not only verified the effectiveness and superiority of our MSGNN on seven graph classification datasets but also analyzed its efficiency, the interpretability of subgraphs and the adaptability of subgraph evolution.

ACKNOWLEDGMENTS

This paper is supported by National Natural Science Foundation of China (NSFC Grant No. 62106275) and Natural Science Foundation of Hunan Province (Grant No. 2022JJ40558).

APPENDIX

Due to the space limit, the appendix is provided in the link: https://github.com/lvyiqin/CIKM-MSGNN/blob/master/appendix.pdf

REFERENCES

[1] James Atwood and Don Towsley. 2016. Diffusion-Convolutional Neural Networks. In *NeurIPS*. 2001–2009.
[2] Jinheon Baek, Minki Kang, and Sung Ju Hwang. 2021. Accurate Learning of Graph Representations with Graph Multiset Pooling. In *ICLR*. https://openreview.net/forum?id=JHcqXGaqiGn
[3] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. 2019. SimGNN - A Neural Network Approach to Fast Graph Similarity Computation. *WSDM* (2019), 384–392.
[4] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. 2022. Hierarchical Representation Learning in Graph Neural Networks With Node Decimation Pooling. *IEEE Transactions on Neural Networks and Learning Systems* 33, 5 (2022), 2195–2207. https://doi.org/10.1109/TNNLS.2020.3044146
[5] M. Karsten Borgwardt and Hans-Peter Kriegel. 2005. Shortest-Path Kernels on Graphs. *ICDM* (2005), 74–81.
[6] M. Karsten Borgwardt, Soon Cheng Ong, Stefan Schönauer, V. N. S. Vishwanathan, J. Alexander Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *ISMB (Supplement of Bioinformatics)* (2005), 47–56.
[7] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. *ICML* (2016).
[8] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry* 34, 2 (1991), 786–797.
[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *NeurIPS* (2016), 3844–3852.
[10] Paul D. Dobson and Andrew J. Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology* 330, 4 (2003), 771–783.
[11] Jinlong Du, Senzhang Wang, Hao Miao, and Jiaqiang Zhang. 2021. Multi-Channel Pooling Graph Neural Networks. *IJCAI* (2021), 1442–1448.
[12] 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. *NeurIPS* (2015).
[13] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. In *ICML*. 2083–2092.
[14] 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.
[15] L. William Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. *NeurIPS* (2017), 1024–1034.
[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (11 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735 arXiv:https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf
[17] Jiatao Jiang, Chunyan Xu, Zhen Cui, Tong Zhang, Wenming Zheng, and Jian Yang. 2020. Walk-Steered Convolution for Graph Classification. *IEEE Transactions on Neural Networks and Learning Systems* 31, 11 (2020), 4553–4566. https://doi.org/10.1109/TNNLS.2019.2956095
[18] Jeroen Kazius, Ross McGuire, and Roberta Bursi. 2005. Derivation and Validation of Toxicophores for Mutagenicity Prediction. *Journal of medicinal chemistry* 48, 1 (02 2005), 312–20. https://doi.org/10.1021/jm040835a
[19] Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *ICLR* (12 2015).
[20] N. Thomas Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. *ICLR* (2017).
[21] Kwei-Herng Lai, Daochen Zha, Kaixiong Zhou, and Xia Hu. 2020. Policy-GNN: Aggregation Optimization for Graph Neural Networks. In *KDD*. 461–471.
[22] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. In *ICML*. 3734–3743.
[23] Maosen Li, Siheng Chen, Ya Zhang, and Ivor Tsang. 2020. Graph cross networks with vertex infomax pooling. In *NeurIPS*, Vol. 33. 14093–14105.
[24] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. 2019. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. *ICML* (2019), 3835–3845.
[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. *ICLR* (2019).
[26] Yixin Liu, Yu Zheng, Daokun Zhang, Hongxu Chen, Hao Peng, and Shirui Pan. 2022. Towards Unsupervised Deep Graph Structure Learning. In *WWW*. 1392–1403.
[27] Pierre Mahé, Nobuhisa Ueda, Tatsuya Akutsu, Jean-Luc Perret, and Jean-Philippe Vert. 2004. Extensions of marginalized graph kernels. *ICML* (2004), 70–70.
[28] Yujie Mo, Liang Peng, Jie Xu, Xiaoshuang Shi, and Xiaofeng Zhu. 2022. Simple Unsupervised Graph Representation Learning. In *AAAI*. 7797–7805.
[29] Nicolò Navarin, Van Dinh Tran, and Alessandro Sperduti. 2019. Universal Readout for Graph Convolutional Neural Networks. *IJCNN* (2019), 1–7.
[30] Wei Peng, Xiaopeng Hong, Haoyu Chen, and Guoying Zhao. 2020. Learning Graph Convolutional Network for Skeleton-based Human Action Recognition by Neural Searching. In *AAAI*. 2669–2676.
[31] Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. 2020. ASAP: Adaptive structure aware pooling for learning hierarchical graph representations. In *AAAI*, Vol. 34. 5470–5477.
[32] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*. 4780–4789.
[33] 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 *The Semantic Web*. 593–607.
[34] Nino Shervashidze, Pascal Schweitzer, Jan van Erik Leeuwen, Kurt Mehlhorn, and M. Karsten Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *JMLR* 12 (2011), 2539–2561.
[35] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*. 488–495.
[36] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic Edge-Conditioned Filters In Convolutional Neural Networks On Graphs. *CVPR* (2017), 29–38.
[37] Qingyun Sun, Jianxin Li, Hao Peng, Jia Wu, Yuanxing Ning, Phillip S Yu, and Lifang He. 2021. SUGAR: Subgraph Neural Network with Reinforcement Pooling and Self-Supervised Mutual Information Mechanism. In *WWW*. 2081–2091.
[38] Naftali Tishby, Fernando Pereira, and William Bialek. 1999. The Information Bottleneck Method. *Proceedings of the 37th Allerton Conference on Communication, Control and Computation* (1999), 368–377.
[39] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *ICLR* (2018).

[40] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2019. Deep Graph Infomax. *ICLR* (2019).
[41] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2016. Order Matters: Sequence to sequence for sets. *ICLR* (2016).
[42] Nikil Wale and George Karypis. 2008. Comparison of Descriptor Spaces for Chemical Compound Retrieval and Classification. *Knowledge and Information Systems* (2008), 678–689.
[43] Chengcheng Wei, Wengang Zhou, Junfu Pu, and Houqiang Li. 2019. Msgcnn: Multi-Scale Graph Convolutional Neural Network For Point Cloud Segmentation. *BIGMM* (2019), 118–127.
[44] Lanning Wei, Huan Zhao, Quanming Yao, and Zhiqiang He. 2021. Pooling Architecture Search for Graph Classification. In *CIKM*.
[45] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386
[46] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks? *ICLR* (2019).
[47] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*. 5453–5462.
[48] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger. 2007. SCAN: A Structural Clustering Algorithm for Networks. In *KDD*. 824–833.
[49] Pinar Yanardag and SVN Vishwanathan. 2015. A structural smoothing framework for robust graph comparison. *NeurIPS* 28 (2015).
[50] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, Vol. 31. 4800–4810.
[51] Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. 2021. Graph Information Bottleneck for Subgraph Recognition. In *ICLR*.
[52] Hao Yuan and Shuiwang Ji. 2020. Structpool: Structured graph pooling via conditional random fields. In *ICLR*.
[53] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. In *NeurIPS (NIPS'18)*. 5171–5181.
[54] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *AAAI*.
[55] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Zhao Li, Chengwei Yao, Dai Huifen, Zhi Yu, and Can Wang. 2021. Hierarchical multi-view graph pooling with structure learning. *IEEE Trans. Knowl. Data Eng.* (2021).
[56] Ying Zhao, Haojin Jiang, Qi'an Chen, Yaqi Qin, Huixuan Xie, Yitao Wu, Shixia Liu, Zhiguang Zhou, Jiazhi Xia, and Fangfang Zhou. 2021. Preserving Minority Structures in Graph Sampling. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2021), 1698–1708. https://doi.org/10.1109/TVCG.2020.3030428
[57] Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *ICLR*.