Bottom-Up and Top-Down Graph Pooling

Jia-Qi Yang, De-Chuan Zhan(✉), and Xin-Chun Li

National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
{yangjq,lixc}@lamda.nju.edu.cn, zhandc@nju.edu.cn

Abstract. Pooling layers are crucial components for efficient deep representation learning. As to graph data, however, it's not trivial to decide which nodes to retain in order to represent the high-level structure of a graph. Recently many different graph pooling methods have been proposed. However, they all rely on local features to conduct global pooling over all nodes, which contradicts poolings in CNNs that only use local features to conduct local pooling. We analyze why this may hinder the performance of graph pooling, then propose a novel graph pooling method called *Bottom-Up and Top-Down graph POOLing* (BUTDPool). BUTDPool aims to learn a more fine-grained pooling criterion based on coarse global structure information produced by a bottom-up pooling layer, and can enhance local features with global features. Specifically, we propose to use one or multiple pooling layers with a relatively high retain ratio to produce a coarse high-level graph. Injecting the high-level information back into low-level representation, BUTDPool enhances learning a better pooling criterion. Experiments demonstrate the superior performance of the proposed method over compared methods.

Keywords: Graph convolution · Graph pooling · Bottom-up and top-down · Graph classification

1 Introduction

The revolution of deep learning [8] has profoundly affected the development of many application fields, such as computer vision [10], natural language processing [12], and audio signal processing [11]. The architecture of neural networks has evolved a lot in recent years, convolutional neural network (CNN) [10] remains the most successful model in applications that can exploit grid-like data structure.

Some structured data, e.g., images, can be represented as grid-like graph structure, yet CNNs are not directly applicable to general graph data. The traditional approach for dealing with graph data is utilizing graph kernel [5]. These methods suffered from drawbacks of kernel methods such as high computational complexity and very shallow model, thus didn't perform well on relatively large datasets. Graph neural networks (GNN) [21] aims at building deep learning methods for graph data such as social networks, citation networks, and the world wide web, and its effectiveness has been shown in many real-world

H. W. Lauw et al. (Eds.): PAKDD 2020, LNAI 12085, pp. 568–579, 2020.
https://doi.org/10.1007/978-3-030-47436-2_43

applications. The basic idea of most GNN models is to migrate successful deep learning building-blocks to graph models. Graph convolutional neural networks (GCN) [13] is a prominent GNN variant, which is an analogy of CNN.

Pooling layer is a critical component of most deep neural networks [8]. State-of-the-art CNNs usually use successive convolutional layers with small kernel sizes followed by one or more pooling layers, which can increase the effective receptive field while keeping the efficiency and representation learning power of convolutional layers. Graph poolings play a similar role in GCNs, and pooling layers are also necessary for tasks such as graph classification, graph encoding, or sub-graph sampling. However, the most popular pooling methods such as spacial max pooling and average pooling can't be incorporated into GCNs easily. Some recent researches focus on providing pooling methods that applicable in GCNs, such as DiffPool [23] and SAGPool [14], and they have shown significant improvement on many graph classification tasks.

Existing graph pooling methods [7,24] rely on features extracted by graph convolution layers to calculate a score for each node. Graph convolution works by aggregating information from neighbor nodes defined by the adjacency matrix, so the feature produced by graph convolution is *local feature* like in common CNNs, where a convolution kernel considers only a small area. The local feature can only reflect local structure around a node itself, but can hardly reflect the importance of a node within a larger area since macroscopic information is never available in local features.

In CNNs the local features are exploited in an intuitive way: 1) Only spacial *nearby* areas are compared during pooling, which is well-defined when considering *local features* 2) Pooling layers guarantee a fixed fraction of *local features* are retained (e.g. $\frac{1}{4}$). So even if the most important part is dropped, any one of its neighbor features will hopefully be fine enough because of the local similarity. On the other hand, existing graph pooling methods work in a very different way: 1) *All* nodes are compared based on their *local features*, including those nodes that are far apart from each other. 2) A fixed fraction of *all* nodes are retrained, so the global structure of the graph may change a lot if some critical parts are dropped completely.

To solve this problem, we propose a new graph pooling method called *Bottom-Up and Top-Down Graph Pooling* (BUTDPool). The core idea is to gather high-level information with one or more coarse pooling layers, then feed this information back to the low-level representation to help to learn a more fine-grained pooling function. Specifically, we propose to apply a bottom-up pooling layer to enlarge the receptive field of each node, then use a top-down unpooling layer to map the pooled graph back and add to the original graph. Finally, a fine-grained pooling layer is applied to the graph with global information. Experiments showed the advantage of our method compared to state-of-the-art graph pooling methods, especially on large graphs.

Generally, we make several noteworthy contributions as follows:

- We analyzed a drawback of existing graph pooling methods that have not been noticed before: they are all pooling over graph globally but only based on local features.
- We proposed a new graph pooling structure called *Bottom-Up and Top-Down Graph Pooling* to tackle that drawback, which is generally applicable and can be complementary with existing methods.
- Experiments on real-world network datasets are conducted. The experimental results demonstrate that the *Bottom-Up and Top-Down Graph Pooling* achieves better results than many existing methods.

2 Related Work

We give a brief overview of some related works in this section.

Graph Convolution. (GC) is a graph representation learning method with a sound theoretical foundation and has been the backbone of many successful graph learning methods, which is also the workhorse of most graph pooling methods. The most widely used graph convolution [13] is an approximation of localized spectral convolution. There are variants of graph convolution, e.g. Graph-SAGE [9] use LSTM instead of mean as aggregation function in graph convolution, however, they preserves the localized property so the drawback discussed in Sect. 3.2 remains.

Hierarchical and Global Graph Pooling. Global pooling aims at obtaining a global summary of a whole graph, which gives GCNs the ability to process graphs with different node numbers and graph structure. Typical global pooling methods including SortPool [24] and Set2Set [18]. However, global pooling methods lack the ability to learn hierarchical or high-level representations. Hierarchical pooling methods aim to provide a texture-level to object-level representations at different layers and wipe off useless information progressively by multiple pooling layers, which is very important in deep models. SAGPool [14] is a recently proposed hierarchical pooling method that has state-of-the-art performance on many benchmark datasets. Since it's unlikely to learn the global structure of a large graph using a single global pooling layer, hierarchical pooling is more important in deep learning. So we only consider hierarchical pooling methods in the rest of our paper.

Score-Based Graph Max-Pooling. A nature idea considering graph pooling is to learn the importance of each node, then only keep the most important nodes and simply drop other unimportant nodes. This is the basic idea of a bunch of graph pooling methods we call as score-based methods, since they calculate a score as a metric of relative importance of each node. SAGPool [14] and gPool [7] are typical score-based pooling methods.

Differentiable Graph Pooling. Score-based methods have a drawback that the top-k operation used in score-based methods is not differentiable. Fully differentiable models are usually easier to optimize than models with non-differentiable components, differentiable graph pooling methods are proposed to tackle this issue. DiffPool [23] is a representative differentiable pooling method where a large graph is downsampled to a small graph with pre-fixed size in a fully differentiable approach. Since the time and space complexity of DiffPool is $\mathcal{O}(|\mathcal{V}|^2)$ rather than $\mathcal{O}(|\mathcal{E}|)$ of sparse methods such as SAGPool, DiffPool can be very slow or even not applicable on large graphs.

Bottom-Up and Top-Down Visual Attention. Visual attention mechanisms have been widely used in image captioning and visual question answering (VQA) [1,16], and similar attention mechanism has been proved to exist in human visual system [3].

The combination of bottom-up and top-down attention was suggested by [1] in VQA task, where the bottom-up attention uses an object detection model to focus on concerned regions, then the top-down attention utilizes language feature to take attention on the image regions most related to the question. Although our method shares a similar name with the bottom-up and top-down attention in VQA, they are completely different in motivation, application, and implementation.

3 Proposed Method

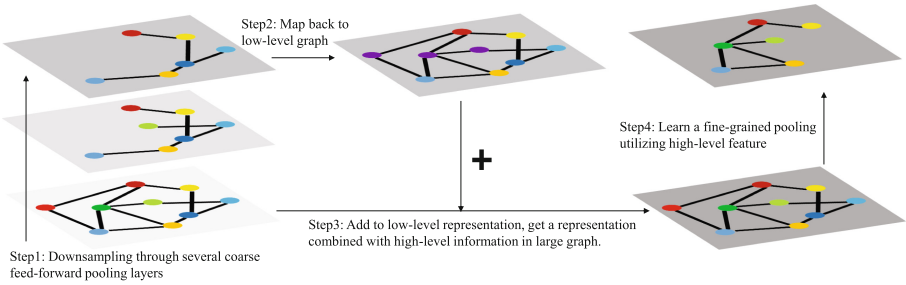

Fig. 1. A bottom-up and top-down graph pooling layer. We use different background color to denote effective receptive field, the darker the background the larger the effective receptive field. Nodes with same color denote the same nodes at different level. This figure depicted a possible pooling procedure on a simple graph. (Color figure online)

In order to learn a more fine-grained graph pooling criterion with larger effective receptive field, we adopt a bottom-up and top-down fashion architecture, which will be defined in detail in the following sections (Fig. 1).

3.1 Notations

We define a graph as $\mathcal{G} = (\mathcal{V}, A)$, where $\mathcal{V}$ is the vertex set that consisting of nodes $\{v_1, \ldots, v_n\}$, and $A \in \mathbb{R}^{n \times n}$ is the adjacency matrix (typically symmetric and sparse) where a_{ij} denotes the edge weight between nodes v_i and v_j. $a_{ij} = 0$ denotes the edge does not exist. The degree matrix is defined as a diagonal matrix $D = \text{diag}\,(d_1, \ldots, d_n)$ where $d_i = \sum_j a_{ij}$. In graph convolution, the adjacency matrix with self-connections $\tilde{A} \in \mathbb{R}^{n \times n}$ is usually used instead of A, where $\tilde{A} = A + I_n$, and $\tilde{D} \in \mathbb{R}^{n \times n}$ is the degree matrix of $\tilde{A}$. In a deep neural network, X^ℓ denotes the input of the ℓth layer, X^0 denote the original feature matrix of the input graph.

Each node v_i in a graph has a feature vector denoted by $\mathbf{x}_i \in \mathbb{R}^d$. For a graph with n nodes, the feature matrix is denoted by $X \in \mathbb{R}^{n \times d}$, where row i correspond to $\mathbf{x}_i$. In graph classification task, each graph belongs to one out of C classes, given a graph we want to predict it's class.

3.2 Local Features and Pooling Criterion

We analyze a drawback of existing pooling methods in this section, which we attribute to the contradiction of local features and global pooling criterion.

We define *local features* as the features that only contain information gathered from the very nearby area. For example, in CNNs, the features are typically calculated by a small kernel (e.g., 3×3). The kernel size determines the receptive field of a convolutional layer, which is the largest area a feature can see, and that feature is innocent to the outside of this receptive field.

The features produced by graph convolutions are also local: typical graph convolutions only consider neighbor nodes, i.e., one-hop connection [13]. This is the expected behavior of GCNs by design to inherit merits of CNNs: parameter sharing at different local area, which can be regarded as a strong prior that local patterns are applicable everywhere [8].

Typical pooling layers in CNNs, say max-pooling without loss of generality, select one feature out of a small feature set (e.g., 1 out of 4 in a max-pooling with size 2×2), then all selected features form a smaller feature map. These local features are comparable since they have information about each other, and even if the feature of most importance is dropped, the selected feature may still be representative of this small area because of local similarity in most nature data like image. The overall structure won't get disturbed after pooling even if the pooling is totally random, which makes poolings in CNNs robust and relatively easier to train.

However, existing graph poolings utilize these local features in a very different and counterintuitive way compared to common poolings in CNNs. They use local features to calculate a score [7,14] to define the importance of each node, then they simply select the nodes with the largest scores. The scores are also local features as we defined before, since they only rely on local features produced by GCNs. The scores of two far apart nodes can't reflect any information of their relationship or their relative importance in the whole graph, which harms

the ability of poolings to retain global structure thus will harm performance in downstream tasks.

The analysis of the drawback of existing methods also sheds light on the idea to resolve it: we need to make local features non-local, i.e., have a larger receptive field, in order to apply score-based pooling in a more fine-grained way.

3.3 Bottom-Up Pooling

Stacking GC layers can increase receptive field, however not efficient since a GC layer can only increase receptive field by 2 hops. And it's indeed hard to train a deep GCN [15,20].

To gather a macroscopic view of the whole graph, we use a stack of base pooling layers to do a coarse pooling. A base pooling layer can be any pooling layer that can reduce node number by a fixed ratio r such as most score-based pooling methods. We define a base pooling layer (BPL) as

$$\tilde{X}, \tilde{A} = \mathrm{BPL}_{\Theta}(X, A, r) \tag{1}$$

where $\tilde{X}$ and $\tilde{A}$ is the feature matrix and adjacency matrix of the graph after pooling, Θ is the parameter of this base pooling layer.

Without loss of generality, we use SAGPool [14] as our base pooling layer. Given input X, A and r, the SAGPool calculates a score vector $\mathbf{y} = GC(X)$. Based on this score, the k nodes with largest scores are selected and their indexes are denoted as idx. The output features is calculated by $\tilde{X} = tanh(y[idx]) \odot X[idx]$, the output adjacent matrix $\tilde{A} = A[idx, idx]$, where the [] operator selects elements based on row and column indexes. The GC is a graph convolution layer with output feature size 1 so that the output of this layer can be used as the score. When a retain-ratio r is given instead of k, we define $k = |\mathcal{V}| * r$.

A bottom-up pooling layer (BUPL) can be defined as a stack of base pooling layers, for example, a bottom-up pooling layer with 2 base pooling layers can be defined by

$$\tilde{X}_1^{\ell}, \tilde{A}_1^{\ell} = \mathrm{BPL}_{\Theta_{bu1}}(X^{\ell}, A^{\ell}, r_{bu}) \tag{2}$$

$$\tilde{X}^{\ell}, \tilde{A}^{\ell} = \mathrm{BPL}_{\Theta_{bu2}}(\tilde{X}_1^{\ell}, \tilde{A}_1^{\ell}, r_{bu}) \tag{3}$$

The corresponding bottom-up pooling layer is denoted by

$$\tilde{X}^{\ell}, \tilde{A}^{\ell}, \mathrm{idx}_{\mathrm{bu}} = \mathrm{BUPL}_{\Theta_{bu1};\Theta_{bu2}}(X^{\ell}, A^{\ell}, r_{bu}) \tag{4}$$

Where $\tilde{X}^{\ell}, \tilde{A}^{\ell}$ is the output of a bottom-up pooling layer. Different from SAGPool layer, we return an index $\mathrm{idx}_{\mathrm{bu}}$ in BUPL to memorize the map between input nodes and output nodes.

The bottom-up pooling layer can produce a much smaller graph, since the retain ratio is relatively small. This graph can be viewed as a rough summary

of the original graph and the receptive field of each node is larger. Roughly, the receptive field of remaining nodes can still cover most nodes, thus this is a high-level graph.

Notice that DiffPool is not applicable as a bottom-up pooling layer since we prefer a sparse pooling method.

3.4 Top-Down Unpooling and Fine-Grained Pooling Layer

Now we have a high-level pooling result produced by the bottom-up pooling layer denoted by $\tilde{X}^\ell, \tilde{A}^\ell$.

In order to feed high-level information back to the low-level graph, we should define a mapping from the downsampled small graph to the original large graph with more nodes, which is unpooling. A nature idea is to apply attention to the small graph and large graph. However, this attention operation will take $\mathcal{O}(|\mathcal{V}_{small}| \times |\mathcal{V}_{large}|)$ time complexity, which loses the merit of the sparse property of GCN.

To keep efficiency and simplicity, we save the index of selected nodes at every bottom-up pooling step so that we can recover the mapping of nodes easily. This is similar to the gUnpool layer proposed by [7]. However, they use zero value in the dropped nodes, which does not feed any information back to those nodes. To fix this, we take the mean of all retained nodes as their corresponding high-level features.

The top-down unpooling can be denoted as follows:

$$\hat{X}^\ell, \hat{A}^\ell = \text{TDUPL}(\tilde{X}^\ell, \tilde{A}^\ell, \text{idx}_{\text{bu}}) \tag{5}$$

Where TDUPL is the top-down unpooling layer.

The retained nodes in unpooling result have information of their own receptive field, and other averaged nodes have information of the whole graph. When this graph is injected to low-level graph, each nodes will have both local and global information (an averaged node will have a retained neighbour with large probability, viceversa. Then one hop relation is considered in following graph convolution).

Then the high-level features are summed with low-level features as input of the fine-grained pooling layer. Which can be defined as

$$Z^\ell = X^\ell + \hat{X}^\ell \tag{6}$$

$$X^{\ell+1}, A^{\ell+1} = \text{FGPL}_{\Theta_{fg}}(Z^\ell, A^\ell, r) \tag{7}$$

Where $\text{FGPL}_{\Theta_{fg}}$ is the fine-grained pooling layer, which can be any score-based pooling layer. The r is the retain ratio, which is larger than the retain ratio r_{bu} used in bottom-up pooling layers. Z^ℓ is the feature combined with local information X^ℓ and higher level information $\hat{X}^\ell$, which gives FGPL the power to learn a more fine-grained pooling.

The proposed bottom-up and top-down pooling layer combines bottom-up layer and top-down layer defined before and can be denoted by

$$X^{\ell+1}, A^{\ell+1} = BUTD_{\Theta_{fg};\Theta_{bu1};\Theta_{bu2}}(X^{\ell}, A^{\ell}, r_{bu}, r) \tag{8}$$

The procedure of a bottom-up and top-down pooling layer is summarized in Algorithm 1.

Algorithm 1: A Bottom-Up and Top-Down Pooling Layer

Input : Adjacent matrix A^{ℓ}; Input feature matrix X^{ℓ}; Bottom-up pooling layer parameters Θ_{bu}^{ℓ} and fine-grained pooling layer parameters Θ_{fg}^{ℓ}; Bottom-up pooling ratio r_{bu} and pooling ratio r

Output: A smaller graph with adjacent matrix $A^{\ell+1}$; Output feature matrix $X^{\ell+1}$

$\tilde{X}^{\ell}, \tilde{A}^{\ell}, \mathrm{idx}_{\mathrm{bu}} = \mathrm{BUPL}_{\Theta_{bu}^{\ell}}(X^{\ell}, A^{\ell}, r_{bu})$

$\hat{X}^{\ell}, \hat{A}^{\ell} = \mathrm{TDUPL}(\tilde{X}^{\ell}, \tilde{A}^{\ell}, \mathrm{idx}_{\mathrm{bu}})$

$X^{\ell+1}, A^{\ell+1} = \mathrm{FGPL}_{\Theta_{fg}^{\ell}}(\hat{X}^{\ell} + X^{\ell}, \hat{A}^{\ell}, r)$

4 Experiments

We give a brief introduction of compared methods and experiment protocol in the following sections.

4.1 Compared Methods

We give a brief introduction of compared methods in the following sections.

gPool is a score-based method used in the Graph U-Nets [7]. gPool suppose that there is a direction defined by vector $\mathbf{p}^{\ell}$ at the ℓth layer that the nodes v_i with feature $\mathbf{x}_i$ align with $\mathbf{p}^{\ell}$ best is the most relative nodes. So the score of node v_i is defined as $\mathbf{x}_i^T \mathbf{p}^{\ell} / \left\|\mathbf{p}^{\ell}\right\|$. The score (after a sigmoid function) is multiplied to the input of next layer to make $\mathbf{p}^{\ell}$ optimizable.

SAGPool is a score-based method proposed by [14]. The authors of SAGPool argue that gPool does not consider topology relationship in graphs since all nodes are projected to the same $\mathbf{p}^{\ell}$. SAGPool uses graph convolution blocks to calculate score instead.

DiffPool is a fully differentiable graph pooling method introduced in [23]. DiffPool uses GNN layers to learn a soft assignment of each node to a cluster, then pool a cluster into a node.

4.2 Experiment Protocol

As mentioned in [17], the data split is a crucial factor that affects evaluation a lot. So we generate 20 different random splits (80%train, 10%validate, 10%test) of every dataset at first, then evaluate each model on these 20 splits and take the average accuracy as our measurement. All methods are implemented using pytorch-geometric [6].

Model Architecture. We follow the model architecture proposed in [14] to make a fair comparison, with the graph pooling layers replaced by compared methods, Figure 2 depicts the model architecture.

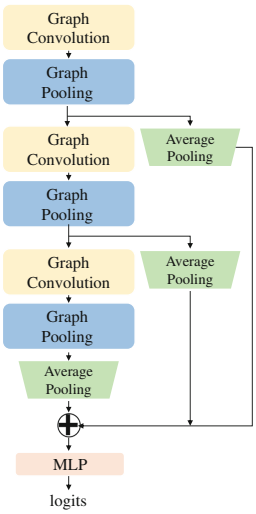

Fig. 2. Model architecture

Datasets. We selected several graph classification benchmark datasets from real biological and chemical applications. The D&D dataset [4] and PROTEINS dataset [2,4] are protein classification datasets. The NCI1 dataset and NCI109 dataset [19] represent two balanced subsets of datasets of chemical compounds classification. REDDIT-MULTI-5K [22] is a social network dataset with large graphs.

The datasets are summarized in Table 1.

Table 1. Summary of datasets

Dataset	Num. Graphs	Num. Classes	Avg. Num. of Nodes	Avg. Num. of Edges
DD	1178	2	284.32	715.66
PROTEINS	1113	2	39.06	72.82
NCI1	4110	2	29.87	32.30
NCI109	4127	2	29.68	32.13
REDDIT	4999	5	508.52	594.87

4.3 Summary of Results

Table 2. Average accuracy and standard deviation of 20 random runs. *: About 2% largest of graphs in D&D dataset are dropped because of being too large for efficiency when training and evaluating DiffPool. NA: We found DiffPool on REDDIT dataset is very slow or cause out-of-memory, since we focus on efficient pooling method we excluded this experiment.

Models	D&D	PROTEINS	NCI1	NCI109	REDDIT
DiffPool	77.24*	74.55	70.87	**72.73**	NA
gPool	75.12	73.61	71.71	69.14	48.02
SAGPool	75.50	75.22	**73.09**	72.01	49.70
BUTDPool(ours)	**77.43**	**75.44**	73.00	**72.28**	**52.14**

From Table 2, we can see that, BUTDPool achieves clear performance gain over the compared methods, especially on D&D and REDDIT dataset with large graphs.

4.4 Complexity Analysis

BUTDPool is a stack of existing graph pooling methods, the overhead of time complexity is determined by the number of base pooling layers. For a BUTDPool layer with 2 bottom-up layers, the running time will be roughly 3 times of a single base pooling layer (2 bottom up layer, 1 fine-grained layer). The space complexity is similar. In a typical deep neural network, the number of pooling layers is small compare to other convolutional layers, so this overhead is affordable.

On the other hand, the overhead of DiffPool is $|\mathcal{V}|$ times complexity of time and space compared to sparse methods, which limits its usability on even medium size graphs.

5 Conclusion

We analyzed the contradiction of local feature and global pooling in existing graph pooling methods, then introduced a novel and easy-to-implement improvement BUTDPool over existing graph pooling methods to mitigate this problem.

The large graph is pooled by a bottom-up pooling layer to produce a high-level overview, and then the high-level information is feedback to the low-level graph by a top-down unpooling layer. Finally, a fine-grained pooling criterion is learned. The proposed bottom-up and top-down architecture is generally applicable when we need to select a sub-graph from a large graph and the quality of the sub-graph matters. Experiments demonstrated the effectiveness of our approach on several graph classification tasks. The proposed BUTDPool can be an alternative building block in GNNs with the potential to make improvements in many existing models.

Acknowledgments. This research was supported by National Key R&D Program of China (2018YFB1004300), NSFC (61773198, 61632004, 61751306), NSFC-NRF Joint Research Project under Grant 61861146001, and Collaborative Innovation Center of Novel Software Technology and Industrialization. De-Chuan Zhan is the corresponding author.

References

1. Anderson, P., et al.: Bottom-up and top-down attention for image captioning and visual question answering. In: Proceedings of 2018 IEEE Conference on Computer Vision and Pattern Recognition, pp. 6077–6086 (2018)
2. Borgwardt, K.M., Ong, C.S., Schoenauer, S., Vishwanathan, S.V.N., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**(Suppl 1), i47–i56 (2005)
3. Corbetta, M., Shulman, G.L.: Control of goal-directed and stimulus-driven attention in the brain. Nat. Rev. Neurosci. **3**(3), 201 (2002)
4. Dobson, P.D., Doig, A.J.: Distinguishing enzyme structures from non-enzymes without alignments. Mol. Biol. **330**(4), 771–783 (2003)
5. Feragen, A., Kasenburg, N., Petersen, J., de Bruijne, M., Borgwardt, K.M.: Scalable kernels for graphs with continuous attributes. In: Advances in Neural Information Processing Systems, vol. 26, pp. 216–224 (2013)
6. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
7. Gao, H., Ji, S.: Graph U-Nets. In: Proceedings of the 36th International Conference on Machine Learning, pp. 2083–2092 (2019)
8. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive Computation and Machine Learning. MIT Press, Cambridge (2016)
9. Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems, vol. 30, pp. 1024–1034 (2017)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
11. Hershey, S., et al.: CNN architectures for large-scale audio classification. In: Proceedings of 2017 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 131–135 (2017)
12. Kim, Y.: Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, pp. 1746–1751 (2014)

13. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th International Conference on Learning Representations (2017)
14. Lee, J., Lee, I., Kang, J.: Self-attention graph pooling. In: Proceedings of the 36th International Conference on Machine Learning, pp. 3734–3743 (2019)
15. Li, Q., Han, Z., Wu, X.: Deeper insights into graph convolutional networks for semi-supervised learning. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, pp. 3538–3545 (2018)
16. Lu, J., Xiong, C., Parikh, D., Socher, R.: Knowing when to look: adaptive attention via a visual sentinel for image captioning. In: Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3242–3250 (2017)
17. Shchur, O., Mumme, M., Bojchevski, A., Günnemann, S.: Pitfalls of graph neural network evaluation. CoRR (2018)
18. Vinyals, O., Bengio, S., Kudlur, M.: Order matters: sequence to sequence for sets. In: Proceedings of the 4th International Conference on Learning Representations (2016)
19. Wale, N., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. In: Proceedings of the 6th IEEE International Conference on Data Mining, pp. 678–689 (2006)
20. Wu Jr., F., A.H.S., Zhang, T., Fifty, C., Yu, T., Weinberger, K.Q.: Simplifying graph convolutional networks. In: Proceedings of the 36th International Conference on Machine Learning, pp. 6861–6871 (2019)
21. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. CoRR (2019)
22. Yanardag, P., Vishwanathan, S.V.N.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1365–1374 (2015)
23. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W.L., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: Advances in Neural Information Processing Systems, vol. 31, pp. 4805–4815 (2018)
24. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, pp. 4438–4445 (2018)