SGC++: Simplified Attentive Graph Convolutional Network

Anonymous Author(s)*

ABSTRACT

Graph neural networks have attracted wide research attention in processing graph-structured data. On one hand, the advances of deep learning encourage graph neural networks to be deeper for stronger performance. On the other hand, considering the efficiency in real-world applications, shallow and fast graph neural networks are broadly investigated. However, how to build an efficient and effective graph neural network remains an unsolved and intriguing problem. While recent Simple Graph Convolution (SGC) has been introduced as a much simpler and faster alternative for Graph Convolutional Network (GCN) without hurting performance, there is still an evident performance gap between SGC and deeper models. In this work, we propose a simplified attentive graph convolutional network, called SGC++, to incorporate adjacency information of multiple hops in SGC with a set of learnable attentive weights to adjust the significance of different hops. Further, two novel methods, Graph Order Descending (OD) and Graph Order Differentiable Selection (DS) are proposed to facilitate the optimization of attentive weights. Experiments show that SGC++ is able to achieve state-of-the-art performance on node classification tasks without sacrificing the efficiency inherited from SGC. In particular, experiments on fashion clustering task can also demonstrate the generalization ability of our model in vision tasks. The code is available at Anonymous Repo.

CCS CONCEPTS

• **Computing methodologies → Neural networks**.

KEYWORDS

neural networks, graph neural networks, computer vision, node classification

ACM Reference Format:
Anonymous Author(s). 2022. SGC++: Simplified Attentive Graph Convolutional Network. In *Proceedings of the Association for Computing Machinery's (ACM) Special Interest Group (SIG) on Knowledge Discovery and Data Mining (SIGKDD).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

With the proliferation of graph-structured data in real-world applications, how to deal with those data has attracted great attention in these years. Recently, there are many works concentrating on developing deep learning approaches for graph data, bringing expeditious progress in the field of graph neural networks. As a pioneer, Graph Convolutional Networks (GCNs) [13] cascade multiple layers of learned first-order spectral filters and nonlinear activation to obtain graph representations. Great superiority has been shown by GCNs and following variants on many applications such as social networks [31], drug discovery [34], point clouds classification [38], point clouds segmentation [16], 3D human pose estimation [49], person Re-identification [35].

There are mainly two focuses in the community about how to improve graph neural networks - effectiveness and efficiency. For effectiveness, deeper network structures are favored [1, 16, 17, 22, 28]. In some deeper models [16, 17], multiple layers where each layer only includes the first hop neighbors are stacked so that the final output can encompass features from multiple hops [16, 17]. In other models, feature transformation is usually done on several hops with respective fully connected layers, and then extracted features are aggregated to get final representation [1, 22, 28]. For efficiency, Simple Graph Convolution (SGC) [39] comes to be a popular model. SGC reduces surplus complexity by removing nonlinear functions and collapsing into one layer without a negative impact on performance compared with vanilla GCNs. But there is still a significant performance gap between SGC and deeper models.

In this work, we aim at unleashing the capability of multi-hop information in a quite efficient way. We propose SGC++, a simplified attentive graph convolutional network. In SGC++, a set of learnable attentive weights are employed to adjust the importance of adjacency relation from different hops. Then through a weighted summation, information from multiple hops are well fused and an adaptive adjacency matrix is obtained for the following steps. Similar to SGC, we directly multiply the new adjacency matrix with the input feature, and a weight matrix (linear layer) is applied to give the final prediction. However, the learning of the attentive weights for different hops is a non-trivial problem in that the weights tend to be dominated by low-order hops and higher-order hop information is less activated. To tackle the difficulty, we deliver Graph Order Descending (OD) method where we zero out the attentive weight of low-order hops at the beginning of training and gradually enable them with the training goes on. To further automate the above schedule, we propose Graph Order Differentiable Selection (DS) method to engage another set of soft masks, ranging from 0 to 1, on top of attentive weights. In training, DS will iteratively update soft masks and attentive weights so that model could learn to automatically choose which hop to participate in feature aggregation.

To demonstrate the effectiveness and efficiency of SGC++, we apply it to 5 different citation&social network datasets for node classification tasks: Cora, Citeseer, Pubmed, Reddit [8], ogbn-arxiv [11]. The experimental results show that SGC++ can not only outperform the state-of-the-art deeper graph neural networks but also preserve the fastness and simplicity inherited from SGC. Moreover, to validate the superiority of SGC++ on graph-structured scenarios in computer vision, SGC++ is also used in a vision task - fashion clustering [23]. Experimental result shows it improves the state-of-the-art [44] by 0.51% and 1.87% respectively on F_P and F_B metrics.

SIGKDD, August 14-18, 2022, Washington, DC
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/XXXXXXX.XXXXXXX

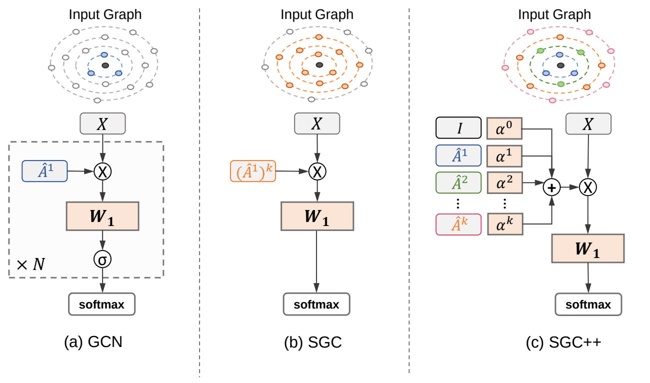


Figure 1: The pipeline of SGC++ on node classification task. Compare with SGC and GCN, SGC++ receives adjacency matrices from multiple hops and fuses them by a set of attentive weights. In this way, SGC++ is able to dynamically accommodate multi-hop adjacency information in a single graph propagation layer.

The main contributions of our paper are summarized as three-fold. (1) We propose the SGC++, a simplified attentive graph convolutional network for graph representation learning. SGC++ is able to attentively fuse the adjacency relation from multiple hops and keep the model simple. (2) We design two novel methods - Graph Order Descending and Graph Order Differentiable Selection to efficaciously update the attentive weights of each hop. (3) We show experimental results on both node classification tasks and Deepfashion clustering task. SGC++ achieves significant performance improvement with much less computation compared with the state-of-the-art models.

2 RELATED WORKS

GCNs generalizes classic Convolutional Neural Networks (CNNs) from euclidean domain to graph domain which has a good deal of real-world applications. We can divide existing GCNs into two overall categories: Spatial GCNs and Spectral GCNs. In the following, we will have a brief review of these two kinds of models and introduce real-world applications of them.

2.1 Spectral GCNs and Spatial GCNs.

Spectral GCNs Spectral GCNs are defined as adding spectral node representations [5, 13, 20, 26, 41] in the Fourier domain on convolution operations. The convolution operation is defined on the eigenvectors of graph Laplacian which can be treated as Fourier basis. Spectral GCNs mainly focus on approximating the decomposition in different ways in order to deal with the high computation cost of the eigendecomposition. For example, Chebyshev polynomials with orders of K to approximate the eigendecomposition is proposed in [5]. In [13], the model is simplified by limiting $K = 1$ and approximating the largest eigenvalue of Laplacian matrix with 2. Also, [20] proposed the Lanczos algorithm by performing a low-rank approximation of graph Laplacian. Meanwhile, without using eigendecomposition, [41] introduces Wavelet transform to spectral GCN. Overall, spectral GCNs might take more computational resources on larger graphs which make it inefficient to be applied on large-scale dataset. This is because spectral GCNs need to perform convolution operations over the whole graph which takes a large amount of memory.

Spatial GCNs Spatial GCNs aggregate the features from spatially close neighbors to perform convolution operations over the graph-structured data. Spatial GCNs mainly focus on exploiting different aggregation methods. [13] uses the weighted sum operator with degree node's reverse as the coefficient. Graph attention mechanism [32] is further proposed to learn the coefficients between nodes. Multi-hop neighbor methods [42] are also proposed to enable better structure-aware representation. There are mainly two directions in improving Spatial GCNs: simplicity and effectiveness. For simplicity and running speed. FastGCN [2] utilizes importance sampling method for fast training but also generalizes well for inference. SGC [39] finds activation function can be redundant in specific tasks and it utilizes multi-hop information in one layer instead of stacking multiple layers as [13]. For more strong representation power, GCNs generally enable more representation ability to have deeper layers. DeepGCN [16] borrows concepts from CNNs including residual/dense connections and dilated convolutions to adapt to GCNs. DeeperGCN [17] unifies different message aggregation operations by utilizing differentiable generalized aggregation functions. Graph Transformer Networks [46] applied the attention to the original adjacency matrix (1st order), but no multi-order adjacency is considered.

Our SGC++ inherits the simplicity of SGC by removing activation function, shrinking to a single layer structure, and pre-computing propagated features that can further speed up training and testing. Then differently, SGC++ employs attentive weights to fuse the adjacency information from multiple hops with minimal added parameters.

Spectral GCNs versus Spatial GCNs Spectral GCNs are solid in graph signal processing theory. Anyhow in an aspect of efficiency and flexibility, spatial models are more influential in mainly 2 ways: (1) Spatial models are more practicable in large graphs because spatial graph convolutions are more computation friendly. (2) Spatial models are more flexible to handle more variants of graphs [7, 19, 30]. For more comprehensive reviews and surveys, please refer to [40].

2.2 Application of GCNs

GCNs are widely applied to many different tasks. In computer vision, those tasks include scene graph generation, point clouds classification, and human-based recognition, etc. Semantic relationships are beneficial to visual scene understanding [18, 43]. [7, 15, 36, 38] utilize the graph information in point clouds to improve the classification or segmentation performance. [35] learns by graph matching to boost occluded person re-identification performance. [49] operates on regression tasks with graph-structured 2D human key points.

Specifically, in our paper, we choose Fashion clustering as the downstream task. Recently supervised clustering with graphs brings considerable performance improvement. [37] performs reasoning by utilizing and inferring the likelihood of linkage between pairs in the sub-graphs. [45] proposes a framework based on GCN, which combines a detection and a segmentation module to pinpoint clusters.

[44] proposes a learnable clustering framework in two sub-problems, GCN-V and GCN-E, to estimate the confidence of vertices and connectivity of edge in the affinity graph. They also propose a GCN utilizing use both input embeddings and embeddings after neighborhood aggregation to learn the transformation weights. In our experiments, we follow the setting of [44] and change the default aggregation manner into our SGC++. Experimental result shows the model with SGC++ can outperform previous state-of-the-art models.

3 METHOD

In this section, the details of the proposed SGC++ are explained. In Sec. 3.1, we introduce the structure of SGC++ by extending the basic SGC with weighted multi-hop adjacency information, where the attentive weights for different hops are learnable. Then, targeting the difficulty of directly updating the weights, we further propose two optimization algorithms - Graph Order Descending (Sec. 3.2) and Graph Order Differentiable Selection (Sec. 3.3) to wisely learn the meaningful attentive weights.

3.1 SGC++

Similar to GCNs [13], we introduce our work in the context of node classification tasks. In these tasks, input of GCNs is a set of labeled nodes and its output is prediction of unlabeled nodes. For a given graph $\mathcal{G} = (\mathcal{V}, \mathbf{A})$, where $\mathcal{V}$ is the vertex set including nodes $\{v_1, \ldots, v_n\}$ and $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the adjacency matrix where a_{ij} stands for the edge weight between nodes v_i and v_j. In $\mathcal{G}$, the node feature matrix is denoted as $\mathbf{X} \in \mathbb{R}^{n\times d}$ where each $\mathrm{x}_i \in \mathbb{R}^d$ corresponds to node v_i as a feature vector. Each node is categorized to one out of C classes with the ground truth one-hot label $\mathbf{y}_i \in \{0,1\}^C$. The node classification task is to predict the labels of test nodes with a subset of labeled training nodes.

Let us first briefly review GCN and SGC. For vanilla GCN [13]:

$$\mathbf{X}^{(l+1)} = \sigma\left(\widehat{A}\mathbf{X}^{(l)}\Theta^{(l)}\right) \tag{1}$$

where $\mathbf{X}^{(l)}$ means the feature of l^{th} layer and $\mathbf{X}^{(0)} = \mathbf{X}$. Θ^l is weight matrix of l^{th} layer and σ is an activation function. The $\widehat{A}$ is a normalized adjacency matrix with self-connection included.

$$\widehat{A} = \mathbf{D}^{-\frac{1}{2}}\widetilde{A}\mathbf{D}^{-\frac{1}{2}} \tag{2}$$

where $\widetilde{A} = \mathbf{A} + I$ and $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ as a diagonal matrix where $\mathbf{D}_{(i,i)} = \sum_j \widetilde{A}_{(i,j)}$.

In SGC, the redundant non-linear function is removed and the number of layers is shrunk to 1. Also, instead of the first hop adjacency matrix as original GCNs, they use k-hops adjacency by raising $\widehat{A}$ to the k-th power, $(\widehat{A})^k$. It's proved that with such a simplified model, SGC can still get comparable performance as GCN. A k-hops SGC for node classification task is formulated as

$$\hat{\mathbf{Y}}_{\mathrm{SGC}} = \mathrm{softmax}\left((\widehat{A})^k\mathbf{X}\Theta\right) \tag{3}$$

However, there is one core problem existing in SGC model. Depending on the property of the applied graph, adjacencies of different hops should have various levels of discriminability. For instance, in visual scene graph [47], the class of a predicate is largely decided by objects it connects to, where first-order adjacency information matters a lot. In [16], a 56-layer GCN is used for point cloud segmentation task, and it significantly boosts performance which means higher-order neighbors matter in point cloud. In SGC++, as shown in Figure 1, we replace the k-hops adjacency $(\widehat{A})^k$ in Eq. 3 with the G^k as a polynomial of various hops of adjacency matrix $\{\widehat{A}^r\}_{r=0}^k$ with learnable coefficients $\{\alpha^r\}_{r=0}^k$. Specifically, $\widehat{A}^0 = I$, where I is an identity matrix.

$$G^k = \sum_{r=0}^{k} \alpha^r \widehat{A}^r \tag{4}$$

In SGC++, we also extract features and classify them in a single graph layer as in SGC. And the generally used cross-entropy loss is applied.

$$\hat{\mathbf{Y}}_{\mathrm{SGC++}} = \mathrm{softmax}\left(G^k\mathbf{X}\Theta\right) \tag{5}$$

$$\mathcal{L} = -\sum_{i\in N_L}\sum_{j=1}^{c} Y_{[i,j]} \ln \hat{\mathbf{Y}}_{[i,j]} \tag{6}$$

where N_L is the set of nodes with labels nodes and $Y \in \mathbb{R}^{n\times c}$ is the label indicator matrix . c is the number of classes. $\hat{\mathbf{Y}}$ is the output of the graph neural network. Specifically, in SGC++, $\hat{\mathbf{Y}} = \hat{\mathbf{Y}}_{\mathrm{SGC++}}$.

In this way, we keep the efficiency of SGC to the most extent, with only $k+1$ trainable parameters added. While the model's capability is greatly improved because SGC++ can learn to enhance the feature aggregation of certain significant hops' information by adjusting the weights $\{\alpha^r\}_{r=0}^k$.

But an interesting phenomenon comes when we try to directly optimize $\{\alpha^r\}_{r=0}^k$ together with other parameters in SGC++. We find that during optimization with gradient descending as previous works [13, 39], the values of $\{\alpha^r\}_{r=0}^k$ tend to be monotonically decreasing with the increasing of hops r, and the α^r of lower-order hops tend to be much higher than higher-order hops. But the adjacency information from higher-order hops should be helpful and ought to be well learned. Based on this observation, we further introduced two novel methods to enforce model focus more on high-order information in optimization.

3.2 Graph Order Descending

Inspired by the generally used technique of learning rate decay, we propose a simple yet effective method named Graph Order Descending (OD). More specifically, at the beginning of the training process, we only allow the adjacency from high-order hops to be aggregated. As the training goes further, we gradually enable the learning of attentive weights for the lower-order hops.

To achieve that, we first decouple α^r to the product of two independent parameters m_r^{OD} and β_r:

$$\alpha^r = m_r^{OD}\beta_r \tag{7}$$

where m_r^{OD} is a binary's mask to regulate the update of α. β is a parameter trained jointly with Θ end to end.

With Eq. 7, we can manually manipulate the m_r^{OD} with certain schedule. In our case, we zeros out the weights of lower hops and keep updating high-order hops' weights at an earlier stage. With

Algorithm 1 Graph Order Descending

```
Input: Feature matrix X and multiple hops adjacency matrix $\widehat{A}^r$ ($r$ from 0 to $k$). The train/validation/test index. Hyperparameter $k$, $\gamma$.
Output: Optimized model parameters.
1: Initialize $m^{OD}$ with $m^{OD}_{k+1} = 1; m^{OD}_{:k+1} = 0$. Initialize $\beta$ with $\beta[:] = 1$. Initialize $\Theta$ with uniform distribution.
2: $t \leftarrow 0$
3: while $t < T$ do
4:     $t \leftarrow t + 1$.
       $id = min(k+1, int(\frac{\gamma t}{T}k))$
5:     if $id \neq 0$ then
           $m^{OD}_{k+1-id} = 1$.
6:     end if
7:     Compute $\alpha$ with Eq. 4.
8:     Compute $\hat{Y}_{SGC++}$ with Eq. 5.
9:     Compute $loss$ with Eq. 6 between $\hat{Y}_{SGC++}$ and training label.
10:    Update model parameters with $loss$.
11: end while
```

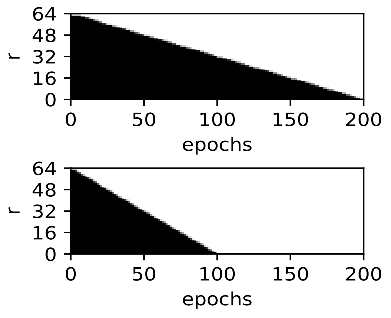


Figure 2: x axis represents t and y axis represents r. T=200, k=64. Black and white represent 0 and 1 respectively. From top to down, each figure's γ is 1.0, 2.0 respectively. γ is a hyperparameter to control the speed to include lower hop information. The smaller γ is, the later lower hop information will be engaged in training, and vice versa.

more training epochs, the m^{OD}_r of lower hops will be set to 1 gradually. Furthermore, we have a hyperparameter γ to control the speed to include lower hop information. The detailed algorithm is illustrated in Algorithm 1. And to intuitively describe the OD schedule, we plot the m^{OD} value along with epochs and different γ values in Figure 2.

With OD schedule, SGC++ is able to prioritize the learning of higher-order feature propagation and obtain more meaningful weights $\{\alpha^r\}^k_{r=0}$, where the bias of lower-order weight is well mitigated. However, OD requires manually adjusting the masks. Even though the algorithm we introduced works generally well in experiments, we can not guarantee that is the best schedule for every dataset. To further make the adjustment of masks automatic and optimized, we propose another method named Graph Order Differentiable Selection (DS), which will be explained in the next section.

Algorithm 2 Graph Order Differentiable Selection

```
Input: Feature matrix X and multiple hops adjacency matrix $\widehat{A}^r$ ($r$ from 0 to $k$). The train/validation/test index. Hyperparameter $k$, $s$.
Output: Optimized model parameters.
1: Initialize $m^{DS}$ with $m^{DS}_{:} = 1$. Initialize $\beta$ with $\beta[:] = 1$. Initialize $\Theta$ with uniform distribution.
2: $t \leftarrow 0$
3: while $t < T$ do
4:     $t \leftarrow t + 1$.
5:     Compute $\hat{Y}_{SGC++}$ with Eq. 5 and Eq. 8 on validation batch.
6:     Compute $loss_{val}$ with Eq. 6 between $\hat{Y}_{SGC++}$ and validation label.
7:     Update $\{m^{DS}_i\}^k_{i=0}$ with $loss_{val}$.
8:     Compute $\hat{Y}_{SGC++}$ with Eq. 5 and Eq. 8 on training batch.
9:     Compute $loss_{train}$ with Eq. 6 between $\hat{Y}_{SGC++}$ and training label.
10:    Update $\beta$ and $\Theta$ with $loss_{train}$.
11: end while
```

3.3 Graph Order Differentiable Selection

Since in the OD method the binary mask is set to be either 0 or 1, a natural way to make it differentiable would be relaxing it to an updatable parameter m^{DS}_i ranging from 0 to 1. The m^{DS}_i can be treated as a soft version of m^{OD}_i with a continuous value. Then how to effectively update the relaxed masks turns out to be a new question.

In the field of network architecture search, DARTS [21] proposed a differentiable method based on bilevel optimization, where they update two sets of weights iteratively. Inspired by DARTS, we introduce the novel Graph Order Differentiable Selection (DS) method to automatically learn m^{DS}_i and β_i by alternatively updating them. When β is fixed, we only update m^{DS} with validation loss, and the model learns to choose which order to use. It's noted that in our experiments, for fair comparison, we randomly take part of training data as validation data for updating $\{m^{DS}_i\}^k_{i=0}$. In the next step, the model updates β and the fully connected layer Θ with training loss to adapt weights for chosen orders. The detailed algorithm is shown in Algo. 2. Further, a softmax function is applied on the relaxed masks to normalize them into $(0, 1)$.

$$\alpha_i = \frac{e^{sm^{DS}_i}}{\sum^k_{r=0} e^{sm^{DS}_r}} \beta_i \tag{8}$$

where s here is a temperature of softmax to control the sparsity as a hyperparameter.

4 EXPERIMENTS

In this section, the experimental results of SGC++ on several datasets about node classification are presented. On these datasets, there are two popular experimental settings: **semi-supervised node classification** and **full-supervised node classification**. The difference between the above two settings is the split of training, validation, and test data. The analysis of results and comparisons with the state-of-the-art models of both settings are provided. Moreover, we also conduct the experiments of applying SGC++ to Deepfashion dataset

Dataset	Cora	Citeseer	Pubmed	Reddit	ogbn-arxiv
Node	2, 708	3, 327	19, 717	233K	169k
Edge	5, 429	4, 732	44, 338	11.6M	1.17M
Feature	1, 433	3, 703	500	602	128
Class	7	6	3	41	40

Table 1: Statistics of used citation&social network datasets [Haoxuan: add reddit, arxiv class]

for **fashion clustering task**, a vision task where the potential graph structure in data is well exploited in previous works. Then some visualization and training time comparisons are done to further analyze SGC++.

4.1 Semi-Supervised Node Classification

4.1.1 On small-scale datasets - Cora, Citeseer, Pubmed.

We first conduct experiments on three relatively small but widely-acknowledged citation datasets.

Dataset and Experimental Setting In the task of semi-supervised node classification, we adopt three widely used citation network datasets: Cora, Citeseer, and Pubmed. In the above three datasets, the feature for each data is the bag-of-words representation of documents. The data connection indicates the citations among those data. The detailed description for the three datasets is listed in Table 1. In semi-supervised node classification, 140/500/1, 000 nodes are used for training/validation/testing in Cora. 120/500/1, 000 nodes are used for training/validation/testing in Citeseer. 60/500/1, 000 nodes are used for training/validation/testing in Pubmed. As for the structure of the proposed SGC++, a single layer consisting of attentive adjacency weights and a linear weight matrix is applied to extract and aggregate features. Then through a softmax function, we get the prediction result. We use Adam optimizer [12] with an initial learning rate of 0.1 to train SGC++ for 100 epochs. The number of hops k is a hyperparameter where it varies for different datasets and optimization schemes. For both SGC and SGC++, we report the best result with the searched optimal k on different datasets.

Performance Comparison To demonstrate the performance of proposed SGC++ on semi-supervised node classification, comprehensive comparison are done on all three datasets. The baseline here is SGC model where the only difference in structure-level is that our SGC++ include attentive multi-hop adjacency information. Various models are put into comparison, including GCN [13] GAT [32], FastGCN [2], AdaLNet and LNet [20] and DGI [33], DAGNN [22], GCNII [3]. We run our models for 10 times and present the mean accuracy. Table 2 reports the throughout comparison results.

Compared with the baseline (SGC model), SGC++ can consistently improve averagely by 1.03% when directly updated (vanilla in Table 2). If Graph Order Descending (OD) is adopted, SGC can get another 0.66% improvement compared with vanilla updating. Then with Graph Order Differentiable Selection (DS) algorithm, SGC++ gains a further 0.33% boost, which is already 2.03% better than baseline. Even though the entire framework of SGC++ is quite simple, it can achieve better performance than current state-of-the-art models. On Citeseer and Pubmed, SGC++ even obtains new state-of-the-art results, 73.9% and 80.7% respectively. On Cora, SGC++ can give a comparable performance as the state-of-the-art models and still outperforms baseline by a large margin. The reason why DAGNN and GCNII perform better than SGC++ on Cora might be their structures' non-linearity operation which is more suited for datasets like Cora. However, it's noted that, in GCNII, multiple layers (64 layers) are cascaded one by one so that the training and inference time costs are much larger than ours. DAGNN is also a deeper model, where feature transformation and propagation are separately done on multiple hops. We will give a more detailed efficiency analysis in Sec. 4.4. The number of k is selected according to the property of the dataset, and further ablation study will be shown in Sec. 4.5.

	Cora	Citeseer	Pubmed
GAT [32]	83.0	72.5	79.0
LNet [20]	79.5	66.2	78.3
AdaLNet [20]	80.4	68.7	78.1
DeepWalk [27]	70.7	51.4	76.8
DGI [33]	82.3	71.8	76.8
DAGNN [22]	84.4	73.3	80.5
GCNII [3] (64 layer)	**85.5**	73.4	80.2
GCN [13]	81.5	70.3	79.0
SGC [39]	81.0(2)	71.9(2)	78.9(2)
SGC++ (vanilla)	81.9(64)	72.8(64)	80.2(64)
SGC++ (OD)	82.8(16)	73.4(16)	80.7(4)
SGC++ (DS)	83.3(128)	**73.9**(128)	**80.7**(64)

Table 2: Test accuracy (%) on citation network datasets. The scores in the table are obtained by averaging over 10 runs for each experiment. The numbers in brackets of SGC++ mean the numbers of hops k we use in the experiment.

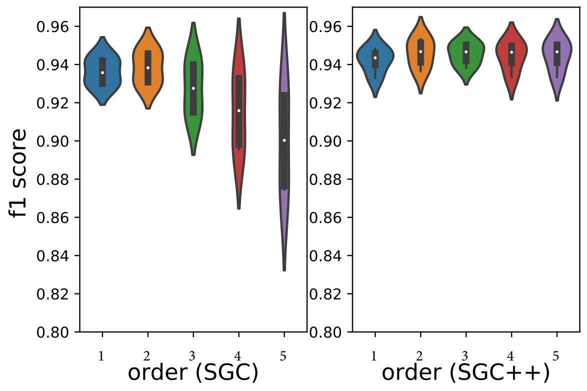


Figure 3: Violin plot with various orders of SGC and SGC++ on Reddit dataset

4.1.2 On Large-scale datasets - Reddit, ogbn-arxiv.

We further fairly validate SGC++ on 2 large-scale datasets: ogbn-arxiv and Reddit.

Dataset and Experimental Setting The details of the two datasets are listed in Table. 1. We follow the generally used data split of training/validation/testing [39]. For SGC++, we use the Graph Order Differentiable Selection method, and the tuning of hyperparameter

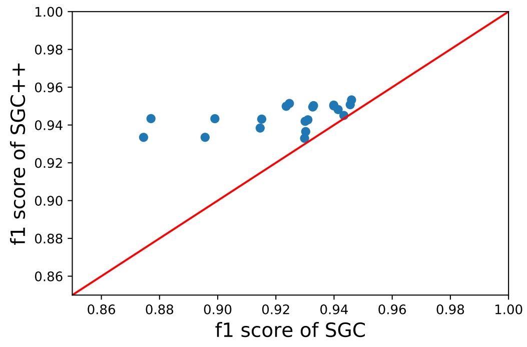


Figure 4: f1 score of all experiments of SGC and SGC++ with the same setting on Reddit dataset. The red line denote y=x. The blue point above the points denotes SGC++ performs better than SGC with same hyperparameter setting.

Method	Test Acc.	Validation Acc.
GCN	71.74	73.00
GraphSAGE	71.49	72.77
SIGN	71.95	73.23
DAGNN	72.09	72.90
DeeperGCN	71.92	72.62
SGC++	72.04(8)	73.09(8)

Table 3: Validation and Test accuracy on ogb-arxiv

mask is based on training data instead of validation data. We use exactly the same setting on SGC and SGC++, where optimizer is Adam, learning rate is from {0.01, 0.1}, weight decay is {5e-4, 5e-3}, order r is from {2,3,4,5,6} and training epochs is 100. Each configuration runs for 10 times with random initialization and we take the average. We in total have run 5x2x2x10 (order x learning rate x weight decay x repeat) experiments for SGC and SGC++ respectively.

Performance Comparison On Reddit, the best testing F1 score for SGC++ is **95.3%** (#hop $k = 2$), while the best testing F1 score for SGC is **94.6%** (#hop $k = 2$). We outperform SGC by **0.7%**. We further plot all the results of different orders in violin figure in Fig 3. We find that SGC++ is consistently better than SGC in any order of {1,2,3,4,5}. And the variance of SGC actually is larger than SGC++ especially in higher order. In Fig 4, we further plot the scores of all the experiments in scatter figure. Every spot denotes one setting of hyperparameter on SGC and SGC++, where x-axis denotes SGC f1 test score and y-axis denotes SGC++ f1 test score. We find that under all the different settings, SGC++ always outperforms SGC. For ogbn-arxiv dataset, in Table. 3, SGC++ also achieves compatible results against the other GNN models that require heavier computation. We can not compare with SGC because SGC didn't report their result on that dataset.

Method	Cora	Citeseer.	Pubmed.
GCN	85.77	73.68	88.13
GAT	86.37	74.32	87.62
Geom-GCN-I	85.19	77.99	90.05
Geom-GCN-P	84.93	75.14	88.09
Geom-GCN-S	85.27	74.71	84.75
APPNP	87.87	76.53	89.40
JKNet	85.25(16)	75.85(8)	88.94(64)
GCNII	88.49(64)	77.08(64)	89.57(64)
SGC++	88.10(64)	76.84(32)	89.18(64)

Table 4: Test accuracy (%) averaged over 10 runs on full-supervised citation networks.

4.2 Full-Supervised Node Classification

Dataset and Experimental Setting In full-supervised node classification, following [26] and [3], we randomly split nodes of each class into 60%, 20%, and 20% for training, validation, and testing, and measure the performance of all models on the testing sets over 10 random splits. Three datasets are used: Cora, Citeseer, and Pubmed. We choose Adam optimizer to update SGC++ with the learning rate to be 0.1 with 5e-5 as the weight decay.

Performance Comparison Experiments and comparisons are shown in Table 4. The most recent models are presented for comprehensive comparison, which includes GCN [13], GAT [32], Geom-GCNs [26], APPNP [14], JKNet [42], GCNII [3]. The numbers in brackets of SGC++ mean the numbers of hops k we use in the experiment. The numbers in brackets of GCNII and JKNet denote the number of layers they have. From Table 4, we can find SGC++ can achieve comparable performance compared with state-of-the-art models. Differently, SGC++ is much more efficient than the listed methods in terms of both training time and the number of parameters. For detailed efficiency analysis, we will introduce it in Sec. 4.4.

4.3 Fashion Clustering

We extend our empirical evaluation to a downstream application in vision field: fashion clustering. The reason we choose this task is the potential graph structure in the data. Recent state-of-the-art models [44] successfully exploit the graph structure by adding GCNs in their models.

Dataset and Experimental Setting The task aims to solve the clustering problem on clothes images. The dataset used in our experiment is DeepFashion [23], a large-scale clothes dataset of over 800K images. We exactly follow the experimental configuration of [44], a state-of-the-art method applying GCNs to model the latent graph structure behind data. More specifically, 25, 752 images from 3, 997 categories are for training and the other 26, 960 images with 3, 984 categories are for testing. For model structure, we directly substitute the GCN part in [44] with the proposed SGC++ without any further modification.

Metrics for Clustering Fashion clustering is commonly evaluated by two metrics [44]: *Pairwise F-score* and *BCubed F-score*. *Pairwise F-score* accentuates on large clusters while *BCubed F-score*

Methods	F_P	F_B
K-means [24]	32.86	53.77
HAC [29]	22.54	48.77
DBSCAN [6]	25.07	53.23
MeanShift [4]	31.61	56.73
Spectral [9]	29.02	46.4
ARO [25]	26.03	53.01
CDP [48]	28.28	57.83
L-GCN [37]	28.85	58.91
LTC [45]	29.14	59.11
GCN-V [44]	33.07	57.26
SGC++-V	**33.58**	**59.13**

Table 5: Results on Deepfashion clustering. Two metrics are used to evaluate the model's performance. F_P means *Pairwise F-score*, and F_B means *BCubed F-score*.

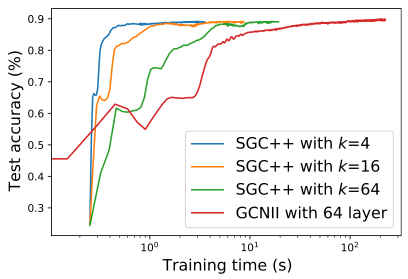
weights clusters conforming to their cluster size. Both metrics are the harmonic mean of precision and recall respectively..

Performance Comparison In Table 5, "GCN-V" denotes the model from [44], where they apply an original GCN on top of affinity to estimate the confidence of vertices. We replace the GCN with proposed SGC++ and obtain the model named "SGC++-V". Besides, several recent methods are also added for comparison. Compared with GCN-V, SGC++-V can bring 0.51% and 1.87% improvement on *Pairwise F-score* and *BCubed F-score* by simply changing GCN to SGC++ . This experiment demonstrates the generalization ability of SGC++ on downstream vision tasks where there is potential graph structure in data.

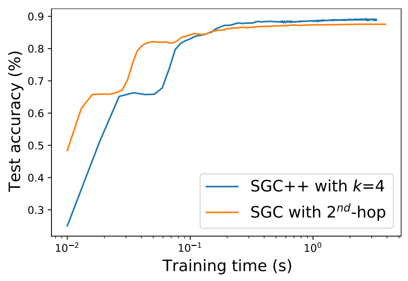
4.4 Efficiency Analysis

In this section, we evaluate the efficiency of proposed SGC++ on both training time and the number of parameters in the network. For a fair comparison, all experiments are done on Pubmed dataset under the full-supervised setting. The training time is measured on an NVIDIA GTX 2080 Ti GPU.

Training Time In Figure 5, we plot the test accuracy over training time of state-of-the-arts graph networks. We select GCNII and SGC for comparison because GCNII is the existing best performed model and SGC is the most efficient model. Especially, SGC and SGC++ take into account the precomputation time to process the original feature to propagated feature as [39]. We set an early stopping schema to stop the training when the validation loss doesn't further drop in 100 epochs. From subfigure (a) of Figure 5, we find SGC++ with different number of hops k is always much faster to converge with comparable results as GCNIIs. And an interesting phenomenon is that with larger k, SGC++ converges slower. The reason might be that too many hops need to be tuned. From subfigure (b) of Figure 5, we find that even compared with SGC, SGC++ converges as fast as SGC and outperforms SGC significantly. However, for the time of one feedforward and one backward computation on the entire Pubmed graph, SGC costs 3×10^{-3}s and SGC++ costs 4.2×10^{-3}s (40% longer). That means, for the time of training a certain number of epochs, SGC++ is slower than SGC because of the added attentive



(a) GCNII versus SGC++



(b) SGC versus SGC++

Figure 5: x axis is the training time in log scale. y axis is the testing accuracy. Experiments are done on Pubmed dataset. The reason why SGC++ and SGC do not start from 0s in the figure is that we include the precomputation time of them.

weights and weighted sum operation. Combining above finding and subfigure (b), we think the added additional parameters and operations help a lot in faster convergence and better performance. In Table 6, we further report the quantitative measurement of training time needed for converge.

Inference Time We further measure the inference time on the larger Reddit dataset. The inference time of SGC and SGC++ with k = 2 on Reddit test data are 5.18e-2s and 5.99e-2s, which means SGC++ is only 15% slower than SGC.

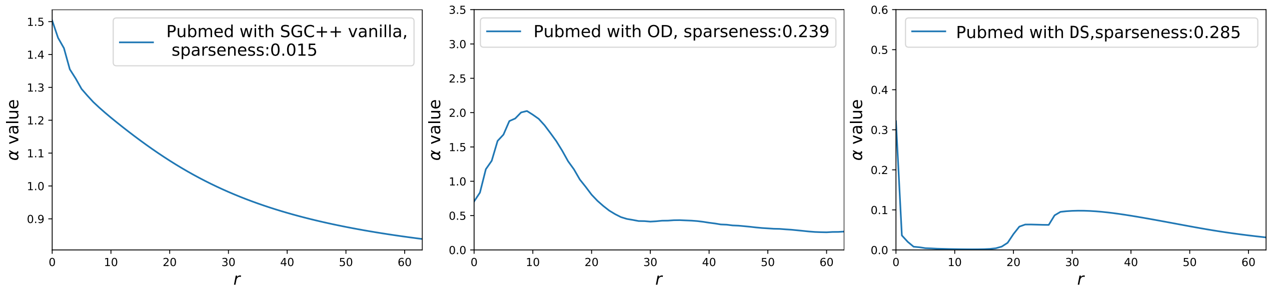
Number of Parameters In Table 6, we present the number of parameters used in SGC, GCNII and SGC++. Compared with SGC, the lightest model, SGC++ only adds a very small amount of additional parameters. Both SGC++ and SGC bring much fewer parameters than GCNII.

Computation Memory SGC++'s propagated feature can be precomputed as in SGC. The increased memory consumption brought by more $\hat{A}$, compared with single $\hat{A}$ in SGC is relatively small regarding other components&operations.

In summary, through above analysis, we find that SGC performs impressively well in terms of both efficiency and effectiveness.

4.5 Ablation on Number of Hops k

Another natural question toward SGC++ would be how the number of hops k would influence the performance. So we further carry on ablation experiments with the DS method on a wide range of k from $\{8, 16, 32, 64, 128\}$. We run experiments of each k over 30 runs with the learning rate as 0.4 and weight decay as 5e-5. Θ is randomly initialized in each running. We show the experimental

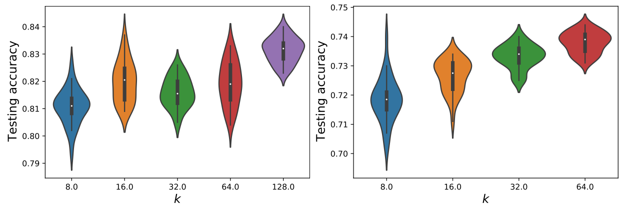


(a) SGC++ with Vanilla (b) SGC++ with OD (c) SGC++ with DS

Figure 6: The learned attentive weights α of SGC++ (vanilla), SGC++ (OD) and SGC++ (DS), with $k = 64$. We can clearly observe that they have different distributions.

	# parameters	converge time (s)
SGC ($k = 4$)	1503	3.87
SGC++ ($k = 4$)	1512	3.28
SGC++ ($k = 64$)	1632	3.8
GCNII (64 layer)	294.4k	226.5

Table 6: The number of parameters and training time needed to converge for SGC, SGC++ and GCNII models. Training time is measure on Pubmed dataset.



(a) Cora (b) Citeseer

Figure 7: The violin plot of test accuracy of SGC++ (DS) on Cora and Citeseer with increasing k. Each experiment is run 30 times and here we draw the obtained mean and variance.

results on both Cora and Citeseer datasets. We draw the mean and variance of each experiment in a violin plot Figure. 7. From the figure, we can find that with k increasing, the testing accuracy is increasing and the variance tends to be smaller. That means a bigger k will bring stronger performance and more robustness, thanks to the contribution of higher-order connection information.

4.6 Visualization of Learned Attentive Weights

Since attentive weights in SGC++ have been empirically proved to be effective in our experiments, it is interesting to dive deeper to have a look at the learned attentive weights for different hops. Here we visualize the optimized attentive weights α of both SGC++ with Graph Order Descending and SGC++ with Graph Order Differentiable Selection on Pubmed dataset in Figure 6.

We can clearly see that the attentive weights generated by DS method tend to be sparse, which might be contributed by the additional softmax mask and iterative updating schema. To further quantitatively compare the difference between vanilla, OD, and DS, we measure the vectors' sparseness [10] based on the relationship between the L1 norm and the L2 norm. The bigger sparseness means the vector is sparser.

$$\text{sparseness}\ (\alpha) = \frac{\sqrt{n} - \left(\sum |\alpha_i|\right) / \sqrt{\sum \alpha_i^2}}{\sqrt{n} - 1} \tag{9}$$

where n is the number of items in x, and $n = k + 1$ here.

The computed sparseness for the vanilla version, OD method, and DS method are 0.015, 0.239, and 0.285 respectively. So, we can see optimized α of DS is sparser than OD's and vanilla's visually and numerically. Intuitively, sparsity indicates DS has the ability to choose few appropriate hops from numerous hops. About the deeper relationship between sparsity and performance, we leave it for further research.

5 CONCLUSION

In this paper, we propose a simplified attentive graph convolutional network on graph-structured data. SGC++. SGC++ can incorporate adjacency information of multiple hops with a set of learnable attentive weights to adjust the significance of different hops. It also inherits the efficiency from simple graph convolution (SGC). Furthermore, Graph Order Descending algorithm and Graph Order Differentiable Selection algorithm are proposed to effectively optimize the attentive weights especially for higher-order hops. We have conducted experiments on citation network node classification and visual fashion clustering task to evaluate the performance of the proposed SCG++ model. Experimental results and comparisons with the state-of-the-art methods demonstrate the effectiveness and efficiency of the proposed SGC++ model.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*. PMLR, 21–29.
[2] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. arXiv:1801.10247 [cs.LG]

[3] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*. PMLR, 1725–1735.

[4] Yizong Cheng. 1995. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence* 17, 8 (1995), 790–799.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375* (2016).

[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*, Vol. 96. 226–231.

[7] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. 2019. Hypergraph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3558–3565.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[9] Jeffrey Ho, Ming-Husang Yang, Jongwoo Lim, Kuang-Chih Lee, and David Kriegman. 2003. Clustering appearances of objects under varying illumination conditions. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, Vol. 1. IEEE, I–I.

[10] Patrik O Hoyer. 2004. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research* 5, 9 (2004).

[11] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

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

[14] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* (2018).

[15] Loic Landrieu and Martin Simonovsky. 2018. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4558–4567.

[16] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9267–9276.

[17] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. 2020. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739* (2020).

[18] Yikang Li, Wanli Ouyang, Bolei Zhou, Jianping Shi, Chao Zhang, and Xiaogang Wang. 2018. Factorizable net: an efficient subgraph-based framework for scene graph generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 335–351.

[19] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2017. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926* (2017).

[20] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S. Zemel. 2019. LanczosNet: Multi-Scale Deep Graph Convolutional Networks. arXiv:1901.01484 [cs.LG]

[21] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[22] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 338–348.

[23] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. 2016. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1096–1104.

[24] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[25] Charles Otto, Dayong Wang, and Anil K Jain. 2017. Clustering millions of faces by identity. *IEEE transactions on pattern analysis and machine intelligence* 40, 2 (2017), 289–303.

[26] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. arXiv:2002.05287 [cs.LG]

[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[28] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198* (2020).

[29] Robin Sibson. 1973. SLINK: an optimally efficient algorithm for the single-link cluster method. *The computer journal* 16, 1 (1973), 30–34.

[30] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3693–3702.

[31] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 817–826.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[33] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax.. In *ICLR (Poster)*.

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

[35] Guan'an Wang, Shuo Yang, Huanyu Liu, Zhicheng Wang, Yang Yang, Shuliang Wang, Gang Yu, Erjin Zhou, and Jian Sun. 2020. High-order information matters: Learning relation and topology for occluded person re-identification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6449–6458.

[36] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)* 38, 5 (2019), 1–12.

[37] Zhongdao Wang, Liang Zheng, Yali Li, and Shengjin Wang. 2019. Linkage based face clustering via graph convolution network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1117–1125.

[38] Xin Wei, Ruixuan Yu, and Jian Sun. 2020. View-GCN: View-based graph convolutional network for 3D shape analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1850–1859.

[39] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

[40] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020).

[41] Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. 2019. Graph wavelet neural network. *arXiv preprint arXiv:1904.07785* (2019).

[42] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*. PMLR, 5453–5462.

[43] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. 2018. Graph r-cnn for scene graph generation. In *Proceedings of the European conference on computer vision (ECCV)*. 670–685.

[44] Lei Yang, Dapeng Chen, Xiaohang Zhan, Rui Zhao, Chen Change Loy, and Dahua Lin. 2020. Learning to cluster faces via confidence and connectivity estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13369–13378.

[45] Lei Yang, Xiaohang Zhan, Dapeng Chen, Junjie Yan, Chen Change Loy, and Dahua Lin. 2019. Learning to cluster faces on an affinity graph. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2298–2306.

[46] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in neural information processing systems* 32 (2019).

[47] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. 2018. Neural motifs: Scene graph parsing with global context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5831–5840.

[48] Xiaohang Zhan, Ziwei Liu, Junjie Yan, Dahua Lin, and Chen Change Loy. 2018. Consensus-driven propagation in massive unlabeled data for face recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 568–583.

[49] Long Zhao, Xi Peng, Yu Tian, Mubbasir Kapadia, and Dimitris N Metaxas. 2019. Semantic graph convolutional networks for 3d human pose regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3425–3435.

A APPENDIX

A.1 Re-production

Besides the link to the anonymous repository, we also uploaded the code to supplementary material with result log.