Graph Convolutional Network with Node Addition and Edge Reweighting for Semi-Supervised Learning

Wen-Yu Lee
GREE, Inc.
Tokyo, Japan

ABSTRACT

Graph convolutional networks (GCNs) provide a promising way to explore datasets that have graph structures in nature. Recent works on GCNs focused on reweighting edges and adding missing edges for corrupted or incomplete graphs. Further, this paper presents a simple while effective extension of GCNs by node addition and edge reweighting. Node addition adds new nodes and edges as communication centers to the original graphs. Thereby nodes may share information together for efficient inference and noise reduction. Edge reweighting re-distributes the weights of edges and even removes noisy edges considering local structures of graphs for performance enhancement. Based on seven benchmark datasets, the experimental results demonstrate that the proposed approach can achieve better performance than three state-of-the-art approaches.

CCS CONCEPTS

• **Computing methodologies** → **Semi-supervised learning settings**; **Neural networks**; • **Mathematics of computing** → **Graph algorithms**.

KEYWORDS

graph convolutional network, semi-supervised learning, node addition, edge reweighting

1 INTRODUCTION

For a long time, convolutional neural networks (CNNs) have been widely used to various multimedia applications, such as image classification [1, 6], semantic segmentation [7, 11], and clothing recommendation [21]. Classical CNNs focus on the problems where a data instance can be represented in a regular grid structure [2], *e.g.*, an image. With a regular structure, filters can directly be applied to extract effective features for model generation. However, many problems involve irregular structures in nature, and these datasets are commonly modeled as irregular graph structures, such as social relation analysis and protein-protein interactions. As a result, generalized CNNs have been rapidly developed for irregular graph structures from single-relational and even multi-relational data instances, see, *e.g.*, [2–4, 10].

Earlier, Bruna *et al.* extended the classical convolution operator based on spectral representation of graphs for generalized CNNs [2]. Extending the work in [2], Defferrard *et al.* proposed a computationally efficient method to perform convolution operation on graphs [3]. Subsequently, Kipf and Welling considered the classical graph-based semi-supervised learning (SSL) problems, where the objective is to predict labels for unlabeled nodes based on labeled nodes, see *e.g.*, [22, 23]. They then developed graph convolutional networks (GCNs) for graph-based SSL problems [10]. Further, Veličković *et al.* added self-attention layers to reweight edges of graphs [18]. Jiang *et al.* then showed performance improvement by seamlessly combining Kipf and Welling's GCNs and their edge reweighting method [8]. Franceschi *et al.* proposed a framework, called LDS, that can simultaneously learn graph structures and GCN parameters [5]. Recently, Yu *et al.* indicated that LDS may not scale well to large graphs, and then presented graph-revised convolutional networks (GRCNs), which are capable of adding new edges and reweighting edges [20].

While most works resorted to edge-based refinement, this paper further explores the domain of node addition. As data instances may have similar features, node addition allows nodes with similar features to share information together and reduce noisy information. Moreover, this paper considers edge reweighting so as to determine proper weights for edges adjacent to the added nodes, and remove noisy edges. Overall, this paper presents a simple while effective extension of GCNs in [10] by node addition and edge reweighting, for graph-based SSL problems. Compared to [8, 18], node addition considers the addition of new nodes and new edges to original graphs, while the two works focused on reweighting the edges existing in the original graphs. Compared to [5], the proposed approach is capable of handling more large-scale datasets. In contrast to [20], the proposed approach further considers the addition of new nodes and removal of noisy edges. Overall, this paper has three main contributions as follows:

- To the best of our knowledge, this paper presents the first work on adding new nodes for graph-based CNNs on SSL problems.
- This paper presents a new method to reweight edges and even to remove noisy edges of a given graph. Besides, the method can benefit node addition on determining proper weights for edges adjacent to the new nodes.
- We conduct experiments on seven datasets to validate the effectiveness of the proposed approach on graph-based SSL problems.

The remainder of this paper is organized as follows. Section 2 reviews the GCN method for graph-based SSL problems. Section 3 details the proposed approach. Section 4 evaluates the performance of the proposed approach. Section 5 concludes this paper. For readability, Appendix A summarizes the symbols and notations used throughout this paper.

2 BACKGROUND

This section briefly reviews the SSL of using the GCN method in [10]. Let $G(V, E)$ be a graph with nodes V and edges $E \subseteq V \times V$. An adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$ provides a representation of whether pairs of nodes in G contain edges connecting them, where $|V|$ is the number of nodes of graph G. Typically, $A_{ij} = 1$ if the

nodes i and j are adjacent, and $A_{ij} = 0$ otherwise. We are given a feature matrix $X = (x_1, x_2, ..., x_n) \in \mathbb{R}^{n \times p}$ for n instances, where x_i is the feature vector of instance i, and p is the dimension of a feature vector. For SSL, each node of graph G is associated with the feature vector of an instance, and thus, $|V| = n$. The adjacency matrix, A, is associated with the relationship between pairs of the instances, *e.g.*, similarity. We let $Y \in \mathbb{R}^{n \times c}$ be a label matrix and L be the set of nodes with labels, where $Y_{ij} = 1$ if $i \in L$ and the label of x_i is j, and $Y_{ij} = 0$ otherwise.

Given X, A, Y, and L, as the inputs for SSL, an r-layer GCN method performs layer-wise propagation as follows.

$$H^{(u+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(u)}W^{(u)}), \tag{1}$$

where $H^{(0)} = X$, $u = 0, 1, ..., r-1$ is a layer index, $\tilde{A} = A + I$ means to add a self-loop of every node, $I \in \mathbb{R}^{n \times n}$ is the identify matrix, $\tilde{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with $\tilde{D}_{ii} = \sum_{j=1}^{n} \tilde{A}_{ij}$, $W^{(u)}$ is the weight matrix that is going to be learned for the u-th layer, and $\sigma(.)$ is an activation function, *e.g.*, ReLU(.). It is worth mentioning that $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is a symmetric normalization for $\tilde{A}$. If we let $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, then $\hat{A}_{ij}$ can be viewed as the weight of edge that connects the nodes associating to x_i and x_j of graph G. Note that the output of the propagation is defined as,

$$Z = \text{softmax}(H^{(r)}W^{(r)}), \tag{2}$$

where $Z \in \mathbb{R}^{n \times c}$. Row Z_i refers to the prediction of the node associating to x_i for the c classes. Finally, the GCN method defines the loss function as,

$$\zeta_{\text{pred}} = -\sum_{i \in L}\sum_{j=1}^{c}\left(Y_{ij}\ln Z_{ij}\right), \tag{3}$$

to measure how good or bad the model does.

3 PROPOSED APPROACH

Figure 1 outlines the proposed approach. We first group the given nodes into several clusters, followed by adding a new node for each cluster. In each time we add a new node into a cluster, we will also connect the new node to the nodes in the same cluster by adding new edges. More details about node addition will be presented in Section 3.1. After adding the new nodes, we will obtain a new graph. To combine edge reweighting with the GCN method, we add a parameter that will be learned for each node of the new graph, and then modify the GCN method partially. More details about edge reweighting will be presented in Section 3.2.

Note that we will follow the notations defined in Section 2 throughout this paper.

3.1 Node Addition

In real-world applications, inputs for SSL might be incomplete or contain noisy data. The goal of node addition is to reduce the influence of the imperfect inputs. More specifically, the motivation of node addition is twofold:

- It is expected that nodes with similar features can share information together and also reduce noise by averaging.
- Adding edges for isolated nodes or even ordinary nodes is capable of increasing the performance of SSL.

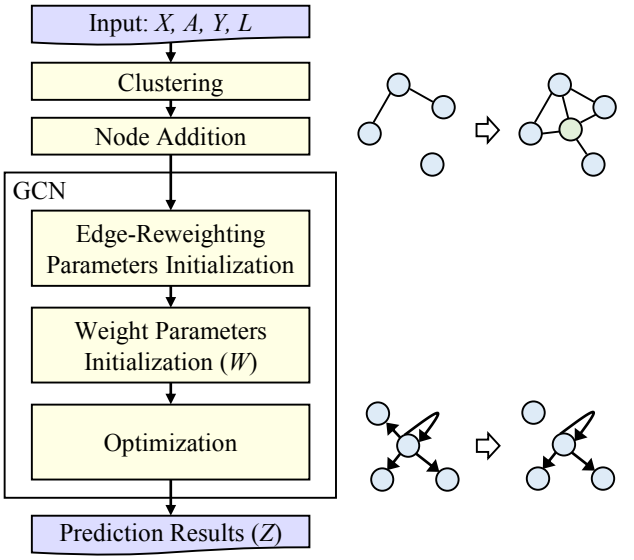

Figure 1: Overall flow of the proposed approach. Nodes with similar features are grouped into clusters. A new node will be added into a cluster as a communication center. Edge reweighting is then applied for GCN optimization.

Considering the two, we intend to provide a *communication center* that connects nodes with similar features, and helps information sharing.

For node addition, we group nodes of graph G into clusters, based on the features associated with the nodes, so that nodes with similar features can be assigned to the same cluster. For each cluster, we then add a node in it, where the feature of the new node is set to be the average of the features of the original nodes in the cluster. That is, the new node is placed on the center of the cluster. In each time we add a new node for a cluster, we also add an edge between the new node and each of the original nodes in the cluster. Figure 2 illustrates the idea. Consider seven nodes that are associated with seven instances, say $\{x_1, x_2, ..., x_7\}$, respectively. As shown in Figure 2(a), suppose that $\{x_1, x_2, x_3, x_4\}$ are grouped into one cluster, and $\{x_5, x_6, x_7\}$ are grouped into the other cluster. As we can see in Figure 2(b), a new node is then added at the center of each cluster. Finally, new edges are added so as to each new node with original nodes, as shown in Figure 2(c).

So far, we have built a *communication center* by adding a new node and some edges, for each cluster of nodes with similar features. Note that the number of nodes added and the number of edges added in graph G equal to the number of clusters and the total number of nodes in graph G, respectively. Because node addition changes graph G, we will use the superscript, $'$, for the notations associated with the new graph. For example, $G'(V', E')$ is the new graph, $X' \in \mathbb{R}^{n' \times p}$ is the new feature matrix, and $A' \in \mathbb{R}^{n' \times n'}$ is the new adjacency matrix, where n' is the number of nodes in graph G', and obviously $(n' - n)$ is the number of nodes added into G. Note that entries of A' are either zeros or one. Intuitively, no two features are exactly the same for most cases, and thus it is expected that weights of edges incident to the *communication centers* could

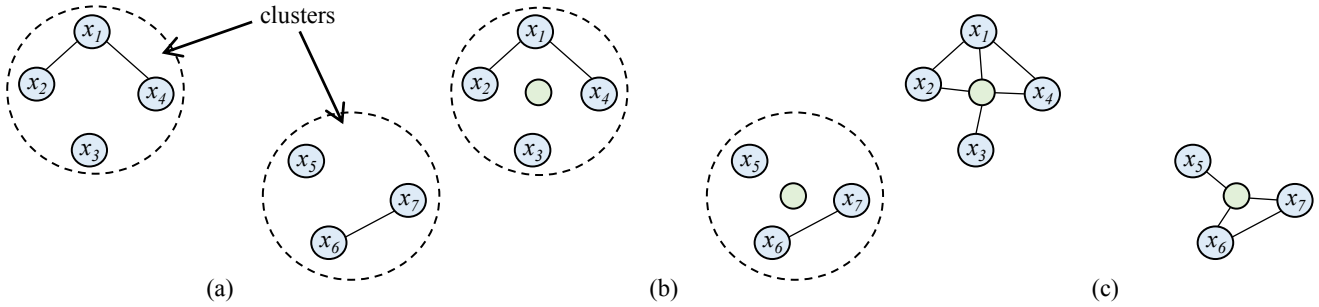

Figure 2: Illustration of node addition. (a) Suppose that nodes associated with $\{x_1, x_2, x_3, x_4\}$ are grouped together, and nodes associated with $\{x_5, x_6, x_7\}$ are grouped together. (b) For each cluster, a new node is added. (c) For each cluster, new edges are added. Each new node will be acted as a communication center of the nodes within the same cluster.

properly be assigned. The assignment of proper weights will be covered in the scope of our edge reweighting method. Later in Section 3.2, we will generate a weighted adjacency matrix from A', and then do edge reweighting.

For practical implementation, we use k-means clustering, and we apply a simple heuristic as follows. We randomly pick m ($m = 5$ in this paper) distinct numbers from 0 to $n/10$ as the candidates of the k value, where we set $n/10$ as a bound because we do not intend to generate small-size clusters. For each of the candidates of the k value, we will go through the flow mentioned in Figure 1. Eventually, we can get m results of accuracy for prediction. We will select the candidate which has the highest accuracy, and then select the two nearest candidates where the candidate with the highest accuracy are included between them.

For example, suppose candidates of the k values are $\{s_1, s_2, .., s_m\}$, where $s_1 < s_2 < ... < s_m$. If s_3 achieves the highest accuracy, s_2 and s_4 will be extracted. If s_1 achieves the highest accuracy, s_1 and s_2 will be extracted. With the two nearest candidates, we then perform a binary-search-like method, which aims to find a better k-value iteratively. In each time we get the middle value of the two nearest candidates, and try to remove half the search towards a higher accuracy than what the two can achieve. The procedure stops if the middle value results in a lower accuracy than we have ever seen or the middle value has been used as a candidate before.

3.2 Edge Reweighting

This section reviews layer-wise propagation first, and details the implementation of edge reweighting afterwards. Based on graph G', Eq. (1) can directly be rewritten as,

$$H'^{(u+1)} = \sigma(\tilde{D}'^{-\frac{1}{2}}\tilde{A}'\tilde{D}'^{-\frac{1}{2}}H'^{(u)}W'^{(u)}). \tag{4}$$

In Eq. (4), we can see that the operation, $\hat{A}' = \tilde{D}'^{-\frac{1}{2}}\tilde{A}'\tilde{D}'^{-\frac{1}{2}}$, normalizes $\tilde{A}'$ in a symmetric way. More specifically, the operation assigns a weight for each edge considering the degrees of nodes in graph G'. Generally, the operation is used when edges of graph G' are undirected, *i.e.*, A' is symmetric. $\hat{A}'$ will be symmetric if A' is symmetric. If edges of graph G' are directed, the operation can be replaced with $\hat{A}' = D'^{-1}\tilde{A}'$, so that each row in $\hat{A}'$ sums to one. That is, weights of edges pointing from each node sums to one. In many cases, $\hat{A}'$ will not be symmetric if edges of graph G' are directed.

Edge reweighting will view the given graph, *i.e.*, G', as a directed graph. Typically, an undirected graph can be converted into a directed graph by replacing the undirected edge between each pair of nodes with two directed edges in opposite direction. Practically, if edges of the initial graph, *i.e.*, graph G, are directed, we will use two directed edges to connect a new node to each of the original nodes in the same cluster for the stage of node addition. Then for Eq. (4), we will replace $\tilde{D}'^{-\frac{1}{2}}\tilde{A}'\tilde{D}'^{-\frac{1}{2}}$ with $D'^{-1}\tilde{A}'$. Note that if edges of the initial graph are undirected, we will not make any change for Eq. (4). It is worth mentioning that our edge reweighting method has two advantages. Firstly, we can consider only the edges pointing from (or to) a node in each time, and thus drastically reduce the complexity of reweighting. Secondly, our method can not only be applied to undirected graphs, but also directed graphs. Note that no matter whether edges of graph G (and thus, graph G') are directed or undirected, the methods introduced below can be applied and are exactly the same.

Given graph G', we create a vector $b = (b_1, b_2, ..., b_{n'})^T \in \mathbb{R}^{n'\times 1}$, as parameters to be learned. Each parameter is assigned to exactly a node of graph G'. The parameter of a node will be added to the weight of every edge pointing from the node. Formally, we generate an adjacency matrix, say $B \in \mathbb{R}^{n'\times n'}$, where

$$B_{ij} = \max(\hat{A}'_{ij} + b_i, 0), \forall i, (i, j) \in E'. \tag{5}$$

Note that b_i can be negative, and max(.) forces negative values to be zero. If B_{ij} equals zero, there is no edge pointing from node i to node j. That is, some edges can be removed if b_i is negative. We then do normalization by $\hat{B} = D'^{-1}B$ so that weights of edges points from each node sums to one. If b_i is positive and b_i is much greater than any of $\hat{A}'_{ij}$ with $(i, j) \in E'$, normalization can make all of the values of $\hat{B}_{ij}$ with $(i, j) \in E'$ be almost the same. That is, parameters $\{b_1, b_2, ..., b_{n'}\}$ can be used to reduce the difference of edge weights or remove some edges of graph G'.

Figure 3 gives an illustration. As shown in Figure 3(a), suppose there is a node associated with x'_1, and there are four edges pointing from the node. A parameter, say b_1, is added for edge reweighting. As can be seen in Figure 3(b), if b_1 is negative, and b_1 is greater than or equal to the weight of exactly one edge, the edge will be

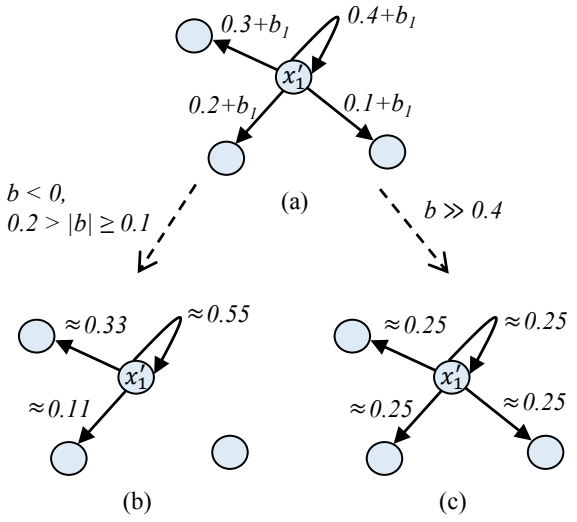

Figure 3: Illustration of edge reweighting. (a) There are four edges pointing from x_1' with weights 0.4, 0.3, 0.2, and 0.1, and a parameter, b_1, is used for edge reweighting. (b) An edge is removed because b_1 is negative. (c) The edge weights would be almost the same if b_1 is large enough.

removed from the graph. Note that normalization will make the sum of the weights of the remaining edges nearly to be one. If b_1 is positive, and b_1 is much greater than any weight of the edges, as in Figure 3(c), normalization will make the edge weights be almost the same.

The resultant layer-wise propagation is as follows:

$$H'^{(u+1)} = \sigma(\hat{B}H'^{(u)}W'^{(u)}). \tag{6}$$

Similar to [8, 13], we then define the loss function used to optimize the parameters as,

$$\zeta_{\text{graph}} = (1-\lambda)\sum_{i,j=1}^{n'}\left(\left\|x_i - x_j\right\|_2^2 \hat{B}_{ij}\right) + \lambda\sum_{i,j=1}^{n'}\hat{B}_{ij}^2, \tag{7}$$

where the former encourages nodes with larger distance in features to have smaller weights, the latter tries to remove noisy edges, and $1 \geq \lambda \geq 0$ is a constant used to control the relative importance between the two terms. Finally, the loss function of our approach is set to be $\zeta_{\text{pred}} + \beta\zeta_{\text{graph}}$, where $\beta \geq 0$ is also a constant used to control the relative importance. Empirically, λ and β are set to be 0.9 and 0.1, respectively.

4 EXPERIMENTS

We implemented the proposed approach based on PyTorch [14] and scikit-learn [15]. For comparative studies, we evaluated the performance of (1) the GCN model [10], (2) the GLCN model [8], (3) the GRCN model [20], and (4) our model. Note that all of the models used the same optimizer (*i.e*, Adam [9]), learning rates, weight decays, and the number of hidden units, based on the settings of GRCN. They were also based on PyTorch. For evaluation, we conducted experiments based on seven publicly available benchmark datasets. The statistics of the datasets are shown in Table 1.

Table 1: Statistics of benchmark datasets.

Dataset	#Nodes	#Features	#Edges	#Classes
Cora	2,708	1,433	5,278	7
CiteSeer	3,327	3,703	4,552	6
PubMed	19,717	500	44,324	3
Photo	7,650	745	119,081	8
CoraFull	19,793	8,710	63,421	70
Computers	13,752	767	245,861	10
CS	18,333	6,805	81,894	15

In Table 1, column "#Nodes" lists the number of nodes, "#Features" the dimension of each feature vector, "#Edges" the number of edges, and "#Classes" the number of classes for classification. Among the datasets, Cora [16], CiteSeer [16], and PubMed [12] are commonly used as benchmark datasets. The preparation of the datasets is the same as [19], where 20 instances of each class were used for training data. Overall, there were 500 and 1,000 instances used for validation data and testing data, respectively. As suggested in [20], Photo, CoraFull, Computers, and CS from [17] were used to check the scalability of our model on number of nodes, number of features, graph density, or number of classes. Same as [20], there were 20 and 30 instances of each class used for training data and validation data, respectively. For testing data, the classes and instances were first removed if the number of instances of a class is smaller than 50. We then used the rest of data as the testing data.

Table 2 summarizes the experimental results. For each dataset, we reported the average results over ten runs with random splits on training, validation, and testing data (the data numbers were kept the same). As can be seen, our model outperformed than other models in most datasets. Based on the results, we can see that adding edges is of benefit to high accuracy, because both GRCN and our model added edges on the original graphs while GCN and GLCN did not. It is interesting to note that GCN outperformed our model on the Cora dataset. We thought that the original graph from the dataset may be *good* enough. Although our model extended GCN, such an extension also enlarged the solution space for optimization. Our model may thus be trapped into a local optimum. Further, the results in Table 2 motivated us to develop (a) a dataset analyzer to predict if a technique (*e.g.*, node addition) could be helpful and (b) a partition approach that helps to apply our model on local graphs, for future studies.

5 CONCLUSION

This paper has presented a simple while effective extension of GCNs. The extension is not limited to undirected graphs; it can also be applied to directed graphs. Briefly, node addition provides communication centers for nodes to share information together. Edge reweighting not only reweights edges, but also removes noisy edges for high performance. Besides the directions mentioned above, future works include the addition of small graphs and dynamic graph modification for GCNs.

Table 2: Comparison the accuracy (%) of GCN, GLCN, GRCN, and our model, on the benchmark datasets, where the best results are marked in bold.

	Cora	CiteSeer	PubMed	Photo	CoraFull	Computers	CS
GCN	**82.13±0.78**	69.99±1.29	76.71±2.81	90.16±1.39	**60.57±0.29**	80.92±1.56	91.44±0.25
GLCN	82.07±0.68	68.83±1.77	76.81±2.47	89.72±1.24	59.93±0.66	79.86±2.20	90.26±0.34
GRCN	**82.13±0.87**	**71.65±1.36**	77.09±2.45	90.85±0.97	60.54±0.55	81.77±1.77	91.21±0.27
Ours	81.98±0.96	70.99±1.33	**77.25±2.47**	**91.70±0.72**	**60.57±0.29**	**82.07±1.35**	**92.56±0.28**

REFERENCES

[1] T. Agrawal, R. Gupta, and S. Narayanan. 2019. On evaluating CNN representations for low resource medical image classification. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*. 1363–1367.

[2] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. 2014. Spectral networks and deep locally connected networks on graphs. In *Proceedings of International Conference on Learning Representations*.

[3] M. Defferrard, X. Bresson, and P. Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of International Conference on Neural Information Processing Systems*. 3844–3852.

[4] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of International Conference on Neural Information Processing Systems*. 2224–2232.

[5] L. Franceschi, M. Niepert, M. Pontil, and X. He. 2019. Learning discrete structures for graph neural networks. In *arXiv preprint arXiv:1903.11960*.

[6] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[7] S. Jégou, M. Drozdzal, D. Vázquez, A. Romero, and Y. Bengio. 2017. The one hundred layers tiramisu: fully convolutional DenseNets for semantic segmentation. In *Proceedings of International Workshop on Computer Vision in Vehicle Technology*.

[8] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo. 2019. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11313–11320.

[9] D. P. Kingma and J. Ba. 2014. Adam: a method for stochastic optimization. In *arXiv preprint arXiv:1412.6980*.

[10] T. N. Kipf and M. Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of International Conference on Learning Representations*.

[11] J. Liu, Q. Zhou, Y. Qiang, B. Kang, X. Wu, and B. Zheng. 2020. FDDWNet: a lightweight convolutional neural network for real-time semantic segmentation. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*. 2373–2377.

[12] G. Namata, B. London, L. Getoor, and B. Huang. 2012. Query-driven active surveying for collective classification. In *Proceedings of International Workshop on Mining and Learning with Graphs*.

[13] F. Nie, X. Wang, and H. Huang. 2014. Clustering and projected clustering with adaptive neighbors. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 977–986.

[14] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. 2017. Automatic differentiation in PyTorch. In *Proceedings of NIPS Workshop on Autodiff*.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. 2011. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research* 12, 85 (2011), 2825–2830.

[16] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. 2008. Collective classification in network data. *AI Magazine* 29, 3 (2008), 93–106.

[17] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. 2018. Pitfalls of graph neural network evaluation. In *arXiv preprint arXiv:1811.05868*.

[18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. 2018. Graph attention networks. In *Proceedings of International Conference on Learning Representations*.

[19] Z. Yang, W. W. Cohen, and R. Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *arXiv preprint arXiv:1603.08861*.

[20] D. Yu, R. Zhang, Z. Jiang, Y. Wu, and Y. Yang. 2020. Graph-revised convolutional network. In *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.

[21] W. Yu, H. Zhang, X. He, X. Chen, L. Xiong, and Z. Qin. 2018. Aesthetic-based clothing recommendation. In *Proceedings of the World Wide Web Conference*. 649–658.

[22] D. Zhou, Q. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. 2003. Learning with local and global consistency. In *Proceedings of NIPS Foundation Advances in Neural Information Processing Systems*. 321–328.

[23] X. Zhu, Z. Ghahramani, and J. Lafferty. 2003. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of International Conference on Machine Learning*. 912–919.

A TABLE OF SYMBOLS AND NOTATIONS

Table 3 lists the symbols and notations used throughout this paper.

Table 3: Symbols and notations used throughout the paper.

Symbol/Notation	Description
A	An adjacency matrix of a graph
β	A constant controls the relative importance between two loss functions
b_i	A parameter associated with node x'_i
B	An adjacency matrix of a graph with parameters of edge reweighting
c	The total number of classes for a classification problem
D	A diagonal matrix of node degrees
E	The edge set of a graph
G	A graph with a set of nodes and a set of edges
$H^{(u)}$	The hidden layer matrix of layer u
i	An index
I	An identity matrix
j	An index
k	The number of centers for k-means clustering
λ	A constant controls the importance between two loss values
L	The set of nodes with labels
m	The number of distinct numbers for k-means clustering
n	The total number of instances
p	The dimension of the feature vector of an instance
r	The total number of layers of a GCN
s_i	The value of the i-th candidate number for k-means clustering
u	A layer index of a GCN
V	The node set of graph G
$W^{(u)}$	A weight matrix to be learned for layer u
x_i	The feature vector of instance i
X	A feature matrix of a set of instances
Y	A label matrix that indicates labels of instances
Z	A matrix of GCN prediction result of instance labels
ζ_{pred}	A loss function for GCN prediction
ζ_{graph}	A loss function for graph refinement by edge reweighting
$\tilde{.}$	A matrix with self-loops added
$\sigma(.)$	An activation function
$\hat{.}$	A normalized matrix
$.'$	A matrix or a number associated with the graph after node addition
$\|.\|$	The number of elements of a set
$\|\|.\|\|_2$	The length of a vector