# Boosting-GNN: Boosting Algorithm for Graph Networks on Imbalanced Node Classification

*Shuhao Shi, Kai Qiao, Shuai Yang, Linyuan Wang, Jian Chen and Bin Yan\**

*Henan Key Laboratory of Imaging and Intelligence Processing, PLA Strategy Support Force Information Engineering University, Zhengzhou, China*

The graph neural network (GNN) has been widely used for graph data representation. However, the existing researches only consider the ideal balanced dataset, and the imbalanced dataset is rarely considered. Traditional methods such as resampling, reweighting, and synthetic samples that deal with imbalanced datasets are no longer applicable in GNN. This study proposes an ensemble model called Boosting-GNN, which uses GNNs as the base classifiers during boosting. In Boosting-GNN, higher weights are set for the training samples that are not correctly classified by the previous classifiers, thus achieving higher classification accuracy and better reliability. Besides, transfer learning is used to reduce computational cost and increase fitting ability. Experimental results indicate that the proposed Boosting-GNN model achieves better performance than graph convolutional network (GCN), GraphSAGE, graph attention network (GAT), simplifying graph convolutional networks (SGC), multi-scale graph convolution networks (N-GCN), and most advanced reweighting and resampling methods on synthetic imbalanced datasets, with an average performance improvement of 4.5%.

Keywords: graph neural network, imbalanced datasets, ensemble learning, adaboost, node classification

## 1. INTRODUCTION

Convolutional neural networks (CNNs) have been widely used in image recognition (Russakovsky et al., 2015; He et al., 2016), object detection (Lin et al., 2014), speech recognition (Yu et al., 2016), visual coding and decoding (Huang et al., 2021a,b). However, traditional CNNs can only handle data in the Euclidean space. It cannot effectively address graphs that are prevalent in real life. Graph neural networks (GNNs) can effectively construct deep learning models on graphs. In addition to homogeneous graphs, heterogeneous GNN (Wang et al., 2019; Li et al., 2021; Peng et al., 2021) can effectively handle more comprehensive information and semantically richer heterogeneous graphs.

The graph convolutional network (GCN) (Kipf and Welling, 2016) has achieved remarkable success in multiple graph data-related tasks, including recommendation systems (Chen et al., 2020; Yu and Qin, 2020), molecular recognition (Zitnik and Leskovec, 2017), traffic forecast (Bai et al., 2020), and point cloud segmentation (Li et al., 2019). GCN is based on the neighborhood aggregation scheme, which generates node embedding by combining information from neighborhoods. GCN achieves superior performance in solving node classification problems compared with conventional methods, but it is adversely affected by datasets imbalance. However, existing studies on GCNs all aim at balanced datasets, and the problem of imbalanced datasets have not been considered.

In the field of machine learning, the processing of imbalanced data sets is always challenging (Carlson et al., 2010; Taherkhani et al., 2020). The data distribution of an imbalanced dataset makes the fitting ability of the model insufficient because it is difficult for the model to learn useful information from unevenly distributed datasets (Japkowicz and Stephen, 2002). A balanced dataset consists of almost the same number of training samples in each class. In reality, it is impractical to obtain the same number of training samples for different classes because the data in different classes are generally not uniformly distributed (Japkowicz and Stephen, 2002; Han et al., 2005). The imbalance of the training dataset is caused by many possible factors, such as deviation sampling and measurement errors. Samples may be collected from narrow geographical areas in a specific time period and in different areas at different times, exhibiting a completely different sample distribution. The datasets widely used in deep learning research, e.g., IMAGENET large scale visual recognition challenge (ImageNet ILSVRC 2012) (Russakovsky et al., 2015), microsoft common objects in context (MS COCO) (Lin et al., 2014), and Places Database (Zhou et al., 2018), balanced datasets, where the amount of data in different classes is basically the same. Recently, more and more imbalanced datasets reflecting real-world data characteristics have been built and released, e.g., iNaturalist (Cui et al., 2018), a dataset for large vocabulary instance segmentation (LVIS) (Gupta et al., 2019), and a large-scale retail product checkout dataset (RPC) (Wei et al., 2019). It is difficult for traditional pattern recognition methods to achieve excellent results on imbalanced datasets, so methods that can deal with imbalanced datasets efficiently are urgently needed.

For imbalanced datasets, additional processing is needed to reduce the adverse effects (Japkowicz and Stephen, 2002). The existing machine learning methods mainly rely on resampling, data synthesis, and reweighting. 1) Resampling samples the original data by undersampling and oversampling. Undersampling removes part of data in the majority class so that the majority class can match with the minority class in terms of the amount of data. Oversampling copies the data in the minority class. 2) Data synthesis, i.e., synthetic minority oversampling technique (SMOTE) (Chawla et al., 2002) and its improved version (Han et al., 2005; Ramentol et al., 2011; Douzas and Bação, 2019) as well as other synthesis methods (He et al., 2008), synthesize the new sample artificially by analyzing the samples in the minority class. 3) Reweighting assigns different weights to different samples in the loss function to improve the model's performance of the model on imbalanced datasets.

In the GNN, the existing processing methods for imbalanced datasets in machine learning are not applicable. 1) The data distribution problem of imbalanced datasets cannot be overcome by resampling. The use of oversampling may introduce many repeated samples to the model, which reduces the training speed and leads to overfitting easily. In the case of undersampling, valuable samples that are important to feature learning may be discarded, making it difficult for the model to learn the actual data distribution. 2) The use of the data synthesis method or oversampling method loses the relationship between the newly generated samples and the original samples in

the GNN, which affects the aggregation process of nodes. 3) Reweighting, e.g., Focal Loss (Lin et al., 2017), and CB Focal Loss (Cui et al., 2019), can solve the problem of the imbalanced dataset in GCN to some extent, but it does not consider the relationship between training samples, and fails to achieve satisfactory performance in dealing with imbalanced datasets.

Ensemble learning methods are more effective in improving the classification performance of imbalanced data than data sampling techniques (Khoshgoftaar et al., 2015). It is challenging for a single model to classify rare and few samples on an imbalanced dataset accurately, thus, the overall performance is limited. Ensemble learning is a process of aggregating multiple base classifiers to improve the generalization ability of classifiers. Briefly, ensemble learning uses multiple weak classifiers to make classification on the dataset. In traditional machine learning, ensemble learning is used to improve the classification accuracy of multi-class imbalanced data (Chawla et al., 2003; Seiffert et al., 2010; Galar et al., 2013; Blaszczynski and Stefanowski, 2015; Nanni et al., 2015; Hai-xiang et al., 2016). In CNNs, some models adopt ensemble learning to deal with imbalanced datasets. Enhanced-random-feature-subspace-based ensemble CNN (Lv et al., 2021) adaptively resamples the training set in iterations to get multiple classifiers and forms a cascade ensemble model. AdaBoost-CNN (Taherkhani et al., 2020) integrates AdaBoost with a CNN to improve accuracy on imbalanced data.

Inspired by ensemble learning, an ensemble GNN classifier that can deal with the imbalanced dataset is proposed in this study. The adaptive boosting (AdaBoost) algorithm is combined with GNN to train the GNN classifier by serialization, and the samples are reweighted according to the calculation results. Based on this, the proposed classifier improved the classification performance on the imbalanced dataset. The main contributions of this study are as follows:

- This article uses the ensemble learning to study the imbalanced dataset problem in GNN for the first time. An Boosting-GNN model is proposed to deal with imbalanced datasets in semi-supervised nodes classification. A transfer learning strategy is also applied to speed up the training of the Boosting-GNN model.
- Four imbalanced datasets are constructed to evaluate the performance of the Boosting-GNN. Boosting-GNN uses GCN, GAT, and GraphSAGE as base classifiers, improving the classification accuracy on imbalanced datasets.
- The robustness of Boosting-GNN under feature noise perturbations is discussed, and it is discovered that ensemble learning can significantly improve the robustness of GNNs.

The rest of this article is organized as follows. Section 2 introduces the related work of dealing with imbalanced data sets and the application of ensemble learning in deep learning. In section 3, the principle of the proposed Boosting-GNN is discussed. Then, four datasets and a proposed method for performance evaluation are described, and the experimental results are discussed in section 4. Finally, section 5 concludes the article.

## 2. RELATED WORKS

Due to the prevalence of imbalanced data in practical applications, the problem of imbalanced data sets has attracted more and more attention. Recent researches are mainly conducted in the following four directions:

### 2.1. Resampling

Resampling can be specifically divided into two types: 1) Oversampling by copying data in minority classes (Buda et al., 2018; Byrd and Lipton, 2019). After oversampling, some samples are repeated in the dataset, leading to a less robust model and worse generalization performance on imbalanced data. 2) Undersampling by selecting data in the majority classes (Buda et al., 2018; Byrd and Lipton, 2019). Undersampling may cause information loss in majority classes. The model only learns a part of the overall pattern, leading to underfitting (Shen and Lin, 2016). $K$-means and stratified random sampling (KSS) (Zhou et al., 2020) performs undersampling after $K$-means clustering for majority classes, and achieves good results.

### 2.2. Synthetic Samples

The data synthesis methods generate samples similar to samples of minority classes in the original set. The representative method is SMOTE (Chawla et al., 2002), and the operations of this method are as follows. For each sample in a small sample set, an arbitrary sample is selected from its $K$-nearest neighbors. Then, a random point on the line between the sample and the selected sample is taken as a new sample. However, the overlapping degree will be increased by synthesizing the same number of new samples for each minority class. The Borderline-SMOTE (Han et al., 2005) synthesizes new samples similar to the samples on the classification boundary. Preprocessing method combining SMOTE and RST (SMOTE-RSB*) (Ramentol et al., 2011) exploits the synthetic minority oversampling technique and the editing technique based on the rough set theory. Geometric SMOTE (G-SMOTE) (Douzas and Bação, 2019) generates a synthesized sample for each of the selected instances in a geometric region of the input space. Adaptive synthetic sampling (ADASYN) (He et al., 2008) algorithm synthesizes different number of new samples for different minority classes samples.

### 2.3. Reweighting

Reweighting typically assigns different weights to different samples in the loss function. In general, reweighting assigns large weights to training samples in minority classes (Wang et al., 2017). Besides, finer control of loss can be achieved at the sample level. For example, Focal Loss (Lin et al., 2017) designed a weight adjustment scheme to improve the classification performance of imbalanced dataset. CB Focal Loss (Cui et al., 2019) introduced a weight factor inversely proportional to the number of effective samples to rebalance the loss, reaching the most advanced level in the imbalanced dataset.

### 2.4. Ensemble Classifiers

Ensemble classifiers are more effective than sampling methods to deal with the imbalance problem (Khoshgoftaar et al., 2015). In GNN models, AdaGCN (Sun et al., 2021) integrates Adaboost and GCN layers to get deeper network models. Different from AdaGCN, Boosting-GNN uses GNN as a sub-classifier of Boosting algorithm to improve the performance on imbalanced datasets. To our knowledge, we are the first to use ensemble learning to solve the classification on graph imbalanced datasets.

In addition, there are transfer learning, domain adaptation, and other methods to deal with imbalance problems. The method based on transfer learning solves the problem by transferring the characteristics learned from majority classes to minority classes (Yin et al., 2019). Domain adaptive method processes different types of data and learns how to reweight adaptively (Zou et al., 2018). These methods are beyond the scope of this article.

## 3. THE PROPOSED METHOD

### 3.1. GCN Model

Given an input undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ and $\mathcal{E}$, respectively, denote the set of $N$ nodes and the set of $e$ edges. The corresponding adjacency matrix $A \in \mathbb{R}^{N \times N}$ is an $N \times N$ sparse matrix. The entry $(i, j)$ in the adjacency matrix is equal to 1 if there is an edge between $i$ and $j$, and 0, otherwise. The degree matrix $D$ is a diagonal matrix where each entry on the diagonal indicates the degree of a vertex, which can be computed as $d_i = \sum_j a_{ij}$. Each node is associated with an $F$-dimensional feature vector, and $X \in \mathbb{R}^{N \times F}$ denotes the feature matrix for all nodes. GCN model of semi-supervised classification has two layers (Kipf and Welling, 2016), and every layer computes the transformation:

$$H^{(l+1)} = \sigma(Z^{(l+1)}), Z^{(l+1)} = \tilde{A}H^{(l)}W^{(l)} \tag{1}$$

where $\tilde{A}$ is normalized adjacency obtained by $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. $W^{(l)}$ is the trainable weights of the layer. $\sigma(\cdot)$ denotes an activation function (usually ReLU), and $H^{(l)} \in \mathbb{R}^{N \times d_l}$ is the input activation matrix of the $l$th hidden layer, where each row is a $d_l$-dimensional vector for node representation. The initial node representations are the original input features:

$$H^{(0)} = X \tag{2}$$

A two-layer GCN model can be defined in terms of vertex features $X$ and $\hat{A}$ as:

$$\text{GCN}_{2-\text{layer}}(\hat{A}, X; \theta) = softmax(\hat{A} \cdot \sigma(\hat{A}XW^{(0)})W^{(1)}) \tag{3}$$

The GCN is trained by the back propagation learning algorithm. The last layer uses the *softmax* function for classification, the cross-entropy loss over all labeled examples are evaluated:

$$\mathcal{L} = -\sum_{|\mathcal{Y}_L|} \sum_{i \in \mathcal{Y}_L} loss(y_i, z_i^L) \tag{4}$$

Formally, given a dataset with $n$ entities $(X, Y) = \left\{(x_i, y_i)\right\}_{i=1}^N$, where $x_i$ represents the word embedding for entity $i$, and $y_i \in \{1, \cdots\cdots, C\}$ represents the label for $x_i$. Multiple weak classifiers are combined with AdaBoost algorithm to make a single strong classifier.

## 3.2. Proposed Algorithm

Since ensemble learning is an effective method to deal with imbalanced datasets, Boosting-GNN adopts the Adaboost algorithm proposed by Hastie et al. (2009) to design an ensemble strategy for GCNs, which can train the GCNs sequentially. In Boosting-GNN, the weight of each training sample is assigned according to the degree to which the sample was not correctly trained in the previous classifier.

### 3.2.1. Aggregation

Boosting-GNN aggregates GNN through the Adaboost algorithm to improve the performance on imbalanced datasets. First, the overall formula of Boosting-GNN can be expressed as:

$$F_M(x) = \sum_{m=1}^{M} \alpha_m * G_m(x; \theta_m) \qquad (5)$$

where $F_M(x)$ is the ensemble classifier obtained after $M$ rounds of training, and $x$ denotes samples. A new GNN classifier $G_m(x; \theta_m)$ is trained in each round, and $\theta_m$ is the optimal parameter learned by the base classifier. The weight of the classifier $\alpha_m$ denotes the importance of classifier, and it could be obtained according to the error of the classifier. Based on (5), Formula (6) can be obtained:

$$F_m(x) = F_{m-1}(x) + \alpha_m * G_m(x; \theta_m) \qquad (6)$$

$F_{m-1}(x)$ is the weighted aggregation of the previously trained base classifier. In each iteration, a new base classifier $G_m(x; \theta_m)$ and its weights $\alpha_m$ are solved. Boosting-GNN uses an exponential loss function:

$$L(y, F(x)) = e^{-y * F(x)} \qquad (7)$$

According to the meaning of the loss function, if the classification is correct, the exponent part is a negative number, otherwise, it is a positive number. As for training the base classifier, the training dataset is $T = \left\{ (x_i, y_i)_{i=1}^{N} \right\}$, $x_i$ is the feature vector of the $i$th node; $y_i$ is the category label of the $i$th node, and $y_i \in \{1, \ldots, C\}$, where $C$ is the total number of classes.

### 3.2.2. Reweight Samples

Assume that during the first training, the samples are evenly distributed and all weights are the same. The data weights are initialized by $D_1 = \{w_1^1, w_2^1, \ldots, w_N^1\}$, where $w_i^1 = 1/N, i = 1, \ldots, N$, and $N$ is the number of samples. Training $M$ networks in sequence on the training set, the expected loss $\varepsilon_m$ at the $m$th iteration is:

$$\varepsilon_m = \sum_{y_i \neq G_m(x_i; \theta_m)} w_i^m = \sum_{i=1}^{N} w_i^m \mathbb{I}(y_i \neq G_m(x_i; \theta_m)) \qquad (8)$$

where $\mathbb{I}$ is the indicator function. When the input is true, the function value is 1; otherwise, the function value is 0. $\varepsilon_m$ is the sum of the weights of all misclassified samples. $\alpha_m$ can be treated as a hyper-parameter to be tuned manually, or as a model parameter to be optimized automatically. In our model, to keep it simple, $\alpha_m$ is assigned according to $\varepsilon_m$.

$$\alpha_m = \frac{1}{2} \ln \frac{1 - \varepsilon_m}{\varepsilon_m} \qquad (9)$$

$\alpha_m$ decreases as $\varepsilon_m$ increases. The first GNN is trained on all the training samples with the same weight of $1/N$, indicating the same importance for all samples. After the $M$ estimators are trained, the output of GNN can be obtained, which is a $C$-dimensional vector. The vector contains the predicted values of $C$ classes, which indicate the confidence of belonging to the corresponding class. For the $m$th GNN input sample $x_i$, the output vector is $p^m(x_i)$. $p_k^m(x_i)$ is the $k$th element of $p^m(x_i)$, where $k = 1, 2, \cdots, C$.

$$w_i^{m+1} = w_i^m e^{\left( -a \frac{C-1}{C} y_i \log(p^m(x_i)) \right)} \qquad (10)$$

$w_i^m$ is the weight of the $i$th training sample of the $m$th GNN. $y_i$ is the one-hot label vector encoded according to the $i$th training sample. Formula (10) is obtained based on Adaboost's Samme.r algorithm (Hastie et al., 2009), which is used to update the weight of the sample. If the output vector of the misclassified sample is not related to the output label, a large value is obtained for the exponential term, and the misclassified sample will be assigned a larger weight in the next GNN classifier. Similarly, a correctly classified sample will be assigned a smaller weight in the next GNN classifier. In summary, the weight vector $D$ is updated so that the weight of the correctly classified samples is reduced and the weight of the misclassified samples is increased.

After the weights of all training samples for the current GNN are updated, they are normalized by the sum of weights of all samples. When the classifier $F_m(x)$ is trained, the weight distribution of the training dataset is updated for the next iteration. When the subsequent GNN-based classifier is trained, the GNN training does not start from a random initial condition. Instead, the parameters learned from the previous GNN are transferred to the $(m + 1)$th GNN, so GNN is fine-tuned based on the previous GNN parameters. The use of transfer learning can reduced the number of training epochs and make the model fit faster.

Moreover, due to the change of weight, the subsequent GNN focuses on untrained samples. The subsequent GNN performs training from scratch on a small number of training samples, which easily causes overfit. For a large number of training samples, the expected label output $p^m(x_i)$ by the GNN after training has a strong correlation with the real label $y_i$. For the subsequent GNN classifier, the trained samples have a smaller weight than the sample without previous GNN training.

### 3.2.3. Testing With Boosting-GNN

After training the $M$ base classifiers, Equation (11) can be used to predict the category of the input sample. The outputs of $M$ base classifiers are summed. In the summed probability vector, the category with the highest confidence is regarded as the predicted category.

$$Q(x) = \underset{k}{argmax} \sum_{m=1}^{M} h_k^m(x) \qquad (11)$$

$h_k^m$ is the classification result of the $k$th sample made by the $m$th basis classifier, $k = 1, 2, \cdots, C$, which can be calculated from the
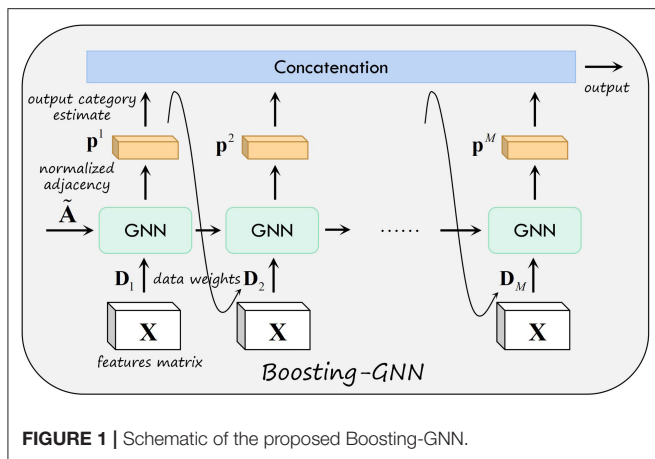
FIGURE 1 | Schematic of the proposed Boosting-GNN.

Equation (12).

$$h_k^m = (C-1) \cdot \left( \log\left(p_k^m(x)\right) - \frac{1}{C} \sum_{i=1}^{C} \log\left(p_i^m(x)\right) \right) \quad (12)$$

Where $p_i^m(x)$ is the $k$th element of the output vector of the $m$th GCN classifier for the input $x$. **Figure 1** shows the schematic of the proposed Boosting-GNN. The first GNN is first trained with the initial weight $D_1$. Then, based on the output of the first GNN, the data weight $D_2$ used to update the second GNN are obtained. In addition, the parameters learned from the first GNN are transferred to the second GNN. After the $m$th base classifier is trained in order, all base classifiers are aggregated to obtain the final Boosting-GNN classifier.

The pseudo-code for an Boosting-GNN is exhibited in **Algorithm 1**. In each iteration of sequential learning, the classifiers are first trained with corresponding training data and weights. Then, according to the training results of the classifiers, the data weights are updated for the next iteration. Both operations are performed until $M$ base classifiers are trained.

## 4. EXPERIMENTS AND ANALYSIS

### 4.1. Experimental Settings

The proposed ensemble model is evaluated on three well-known citation network datasets prepared by Kipf and Welling (2016): Cora, Citeseer, and Pubmed (Sen et al., 2008). These datasets are chosen because they are available online and are used by our baselines. In addition, experiments are also conducted on the Never-Ending Language Learning (NELL) dataset (Carlson et al., 2010). As a bipartite graph dataset extracted from a knowledge graph, NELL has a larger scale than the citation datasets, and it has 210 node classes.

### 4.1.1. Citation Networks

The nodes in the citation datasets represent articles in different fields, and the labels of nodes represent the corresponding journal where the articles were published. The edges between two nodes represent the reference relationship between articles. If an edge

---

**Algorithm 1** Framework of the Boosting-GNN algorithm.

**Input:** Training set $T = \{(x_1, y_1), \ldots, (x_N, y_N)\}$;
**Output:** Ensemble of classifiers $F_M(x)$;
1:  Initialization: $w_i^1 = 1/N$ for all $1 \leq i \leq N$
2:  **for** $m = 1, 2, \cdots, N$ **do**;
3:      **if** $m = 1$ **then**
4:          Train GNN classifier with weighted sample set $\{T, D_1\}$;
5:      **else**
6:          Transfer the learning parameters of the $(m-1)$th GNN to the $m$th GNN classifier;
7:          Train the $m$th GNN classifier with weighted sample set;
8:      **end if**
9:      Calculate the output category estimated for the $C$ classes of the $m$th GNN classifier $p_k^m(x)$, where $k = 1, 2, \cdots, C$;
10:      Calculate the training error $\varepsilon_m$ of the $m$th classifier according to (8);
11:      Assign the weight $\alpha_m$ to the classifier based on $\varepsilon_m$ using (9);
12:      Update the sample weight $D_{m+1}$ according to $p_k^m(x)$, and normalize the sample weight $D_{m+1}$;
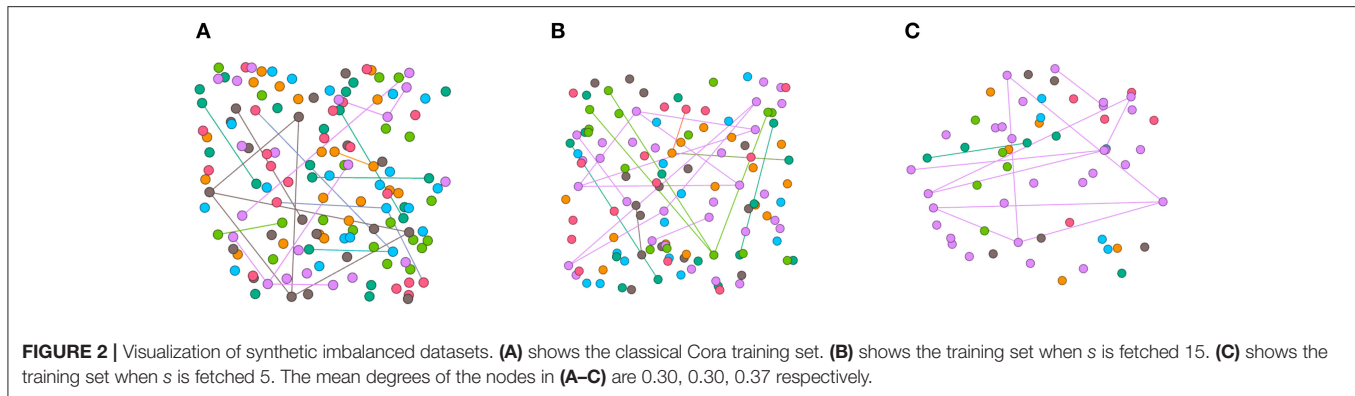13: **end for**

TABLE 1 | Datasets used for experiments.

| Dataset | Cora | Citeseer | Pubmed | NELL |
|---|---|---|---|---|
| Vertices | 2,708 | 3,327 | 19,717 | 65,755 |
| Edges | 5,429 | 4,732 | 44,338 | 266,144 |
| Classes | 7 | 6 | 3 | 210 |
| Features | 1,433 | 3,703 | 500 | 5,414 |

exists between the nodes, there is a reference relationship between the articles. Each node has a one-hot vector corresponding to the keywords of the article. The task of categorization is to classify the domain of unlabeled articles based on a subset of tagged nodes and references to all articles.

### 4.1.2. Never-Ending Language Learning

The pre-processing schemes described in Yang et al. (2016) are adopted in this study. Each relationship is represented as a triplet $(e_1, r, e_2)$, where $e_1$, $r$, and $e_2$, respectively, represent the head entity, the relationship, and the tail entity. Each entity $E$ is regraded as a node in the graph, and each relationship $r$ consists of two nodes $r_1$ and $r_2$ in the graph. For each $(e_1, r, e_2)$, two edges $(e_1, r_1)$ and $(e_2, r_2)$ are added to the graph. A binary, symmetric adjacency matrix from this graph is constructed by setting entries $A_{ij} = 1$, if one or more edges are present between nodes $i$ and $j$ (Kipf and Welling, 2016). All entity nodes are described by sparse feature vectors with the dimension of 5,414. **Table 1** summarizes the statistics of these datasets.

**FIGURE 2 |** Visualization of synthetic imbalanced datasets. **(A)** shows the classical Cora training set. **(B)** shows the training set when $s$ is fetched 15. **(C)** shows the training set when $s$ is fetched 5. The mean degrees of the nodes in **(A–C)** are 0.30, 0.30, 0.37 respectively.

### 4.1.3. Synthetic Imbalanced Datasets

Different synthetic imbalanced datasets are constructed based on the datasets mentioned above. According to the Pareto Principle that 80% of the consequences come from 20% of the causes, one of the classes is randomly selected as the majority category for simplicity. The remaining classes are regraded as minority classes. In Kipf and Welling (2016), 20 samples of each class were selected as the training set, and to keep the number of training samples broadly consistent, the datasets are described in Equation (13).

$$n_i = \begin{cases} 30 & i = c \\ s & i \neq c \end{cases} \quad (13)$$

$n_i$ is the number of samples in category $i$, $c$ is the randomly selected category, $C$ is the number of classes in the dataset, and $s$ is the number of samples in the minority category. By changing $s$, the number of minority category samples is altered, thus changing the degree of imbalance in the training set. For example, in the Cora dataset, there are seven classes of samples. So, the number of samples in one class is fixed to 30, and the number of samples in the other six classes is changed. Each time the training is conducted, a certain number of samples are randomly selected to form the training set. The test set is divided following the method in Kipf and Welling (2016) to evaluate the performance of different models.

Synthetic imbalanced datasets are constructed by node dropping. Given the graph $\mathcal{G}$, node dropping will randomly discard vertices along with their connections until the number of different classes of nodes matches the setting. In node dropping, the dropping probability of each node follows a uniform distribution. We visualize the synthetic datasets in **Figure 2** and use different colors to represent different categories of nodes. Due to the sparsity of the adjacency matrix of the graph data set, imbalanced sampling of the graph data does not reduce the average degree of the nodes. Although disconnect parts of the graph, missing part of vertices does not affect the semantic meaning of $\mathcal{G}$.

### 4.1.4. Parameter Settings

In Boosting-GNN, five GNN base classifiers are used. Boosting-GNN, respectively, uses GCN, GraphSAGE, and GAT as the base classifiers. All networks are composed of two layers, and all models are trained for a maximum of 100 epochs (training iterations) using Adam optimizer. For Cora, Citeseer, and Pubmed datasets, the number of hidden units is 16, and L2 regularization is 5e-4. For NELL, the number of hidden units is 128, and L2 regularization is 1e-5.

The following sets of hyperparameters are used for Boosting-GNN: For Boosting-GCN, the activation function is ReLU. The learning rates on Cora, Citeseer, Pubmed, and NELL are 1e-2, 1e-2, 1e-2, 5e-3, respectively. For Boosting-GraphSAGE, the activation function is ReLU. The sampled sizes (S1 = 25, S2 = 10) is used for each layer. The learning rates on Cora, Citeseer, Pubmed, and NELL are 1e-3, 1e-3, 5e-4, 1e-4, respectively. For Boosting-GAT, the first-layer activation function is *ELU* and the second-layer activation function is *softmax*. The number of attention heads $K$ is 8. The learning rates on Cora, Citeseer, Pubmed and NELL are 1e-3, 1e-3, 1e-3, 5e-4, respectively.

For GCN, GraphSAGE, GAT, SGC, N-GCN, and other algorithms, the models are trained for a total of 500 epochs. The highest accuracy is taken as the result of a single experiment, and the mean accuracy of 10 runs with random sample split initializations is taken as the final result. A different random seed is used for every run (i.e., removing different nodes), but the 10 random seeds are the same across models. All the experiments are conducted on a machine equipped with two NVIDIA Tesla V100 GPU (32 GB memory), 20-core Intel Xeon CPU (2.20 GHz), and 192 GB of RAM.

## 4.2. Baseline Methods

The performance of the proposed method is evaluated and compared to that of three groups of methods:

### 4.2.1. GCN Methods

In experiments, our Boosting-GNN model is compared with the following representative baselines:

- Graph convolutional network (Kipf and Welling, 2016) produces node embedding vectors by truncating the Chebyshev polynomial to the first-order neighborhoods.
- GAT (Velickovic et al., 2018) generates node embedding vectors for each node by introducing an attention mechanism when computing node and its neighboring nodes.

TABLE 2 | Summary of results in terms of classification accuracy (in percentage).

| Model | Cora | Citeseer | Pubmed | NELL |
|---|---|---|---|---|
| GCN | 65.6 ± 0.8 | 62.2 ± 0.5 | 71.8 ± 0.6 | 68.5 ± 1.4 |
| GraphSAGE | 66.3 ± 0.8 | 59.7 ± 0.6 | 69.7 ± 0.6 | 69.6 ± 1.3 |
| GAT | 67.4 ± 0.7 | 60.3 ± 0.6 | 66.2 ± 0.7 | 70.3 ± 1.6 |
| N-GCN | 67.3 ± 0.6 | 65.4 ± 0.3 | 72.3 ± 0.3 | 73.3 ± 1.2 |
| SGC | 69.7 ± 0.8 | 59.4 ± 0.5 | 66.9 ± 0.5 | 67.1 ± 1.4 |
| GCN-FL | 67.8 ± 1.2 | 65.1 ± 0.8 | 72.4 ± 0.8 | 71.2 ± 1.2 |
| GraphSAGE-FL | 66.5 ± 1.2 | 59.5 ± 0.8 | 69.7 ± 1.3 | 72.1 ± 1.1 |
| GAT-FL | 67.4 ± 1.3 | 61.3 ± 0.7 | 69.2 ± 1.2 | 72.6 ± 1.0 |
| GCN-CB | 70.6 ± 0.9 | 65.1 ± 0.6 | 72.3 ± 0.8 | 72.9 ± 1.4 |
| GraphSAGE-CB | 66.3 ± 0.9 | 59.7 ± 0.9 | 70.1 ± 0.9 | 69.8 ± 1.4 |
| GAT-CB | 67.6 ± 1.0 | 60.3 ± 1.0 | 69.3 ± 0.9 | 73.4 ± 1.5 |
| GCN-RS | 70.4 ± 1.0 | 61.8 ± 1.1 | 70.4 ± 1.1 | 68.9 ± 2.1 |
| Boosting-GCN | 73.2 ± 0.7 | **65.7 ± 0.7** | **73.1 ± 0.7** | 74.9 ± 1.0 |
| Boosting-GraphSAGE | 72.4 ± 1.0 | 63.2 ± 1.0 | 70.4 ± 1.1 | 75.3 ± 1.2 |
| Boosting-GAT | **73.5 ± 0.5** | 64.3 ± 0.8 | 69.7 ± 0.7 | **75.5 ± 1.0** |

*The highest performance of models is highlighted in boldface.*

- GraphSAGE (Hamilton et al., 2017) generates the embedding vector of the target vertex by learning a function that aggregates neighboring vertices. The default settings of sampled sizes (S1 = 25, S2 = 10) is used for each layer in GraphSAGE.
- SGC (Wu et al., 2019) reduces model complexity by eliminating the non-linearity between GCN layers, transforming a non-linear GCN into a simple linear model that is more efficient than GCNs and other GNN models for many tasks.
- N-GCN (Abu-El-Haija et al., 2019) obtains the feature representation of nodes by convolving in the neighborhood of nodes at different scales and then fusing all the convolution results. These methods can be regarded as ensemble models.

### 4.2.2. Resampling Method
The KSS (Zhou et al., 2020) method is used for performance comparison. KSS is a kind of $K$-means clustering method based on undersampling and achieves state-of-the-art performance on an imbalanced medical dataset.

### 4.2.3. Reweighting Method
Boosting-GNN is compared with GCN, GraphSAGE, and GAT. These classic models use Focal Loss (Lin et al., 2017) and CB-Focal (Cui et al., 2019), and achieve good classification accuracy on imbalanced datasets.

## 4.3. Node Classification Accuracy
Our method is implemented in Keras. For the other methods, the code from the Github pages introduced in the original articles is used. For synthetic imbalanced datasets, $s$ is set to 10. The classification accuracy of GCN, GraphSAGE, GAT, SGC, N-GCN, and Boosting-GNN method is listed in **Table 2**.

Results in **Table 2** show that Boosting-GNN outperforms the classic GNN models and state-of-the-art methods for processing imbalanced datasets. The N-GCN obtains a feature representation of the nodes by convolving around the nodes at different scales and then fusing all the convolution results, which can slightly improve the classification compared to the GCN. Resampling method and Reweighting method can improve the accuracy of GNN on imbalanced datasets, but the improvement is very limited. Since RS is not suitable for graph data, RE is slightly better than RS. Boosting-GNN can significantly improve the classification accuracy of GNN, with an average increase of 6.6, 3.7, 1.8, and 5.8% compared with the original GNN model in Cora, Citeseer, Pubmed, and NELL datasets, respectively.

Implementation details are as follows: Following the method in Kipf and Welling (2016), 500 nodes are used as the validation set and 1,000 nodes as the test set. Besides, for a fair performance comparison, the same training procedure is used for all the models.

## 4.4. Effect of Different Levels of Imbalance in the Training Data
The level of imbalance in the training data is changed by gradually increasing $s$ from 1 to 10. The evaluation results of Boosting-GNN, GCN, GraphSAGE, and GAT are compared, which are shown in **Figure 3**.

Results in **Figure 3** show that classification accuracy of different models varies with $s$. The shadows indicate the range of fluctuations in the experimental results. When $s$ is relatively small, the degree of imbalance in the training data is large. In this case, the classification accuracy of Boosting-GNN is higher than that of GCN, GraphSAGE, and GAT. As $s$ decreases, the performance advantage of Boosting-GNN increases gradually. Experimental results show that when the sample imbalance is large, aggregation can significantly reduce the adverse effects caused by sample imbalance and improve the classification accuracy. On the Cora dataset, the accuracy of Boosting-GCN, Boosting-GraphSAGE, Boosting-GAT exceeds that of GCN, GraphSAGE, and GAT by 10.3, 8.0, and 6.1% respectively at most.

## 4.5. Impact of Numbers of Base Classifiers
The number of base classifiers is changed to evaluate the classification accuracy on imbalanced datasets with different base classifiers. We compare the classification results of Boosting-GCN and GCN, and the experimental results are listed in **Table 3**.

The experimental results show that aggregation can contribute to performance improvements. As the number of base classifiers increases, the performance improvement is more and more significant. As the number of base classifiers increases from 3 to 11, the number of base classifiers is odd. The data of Cora, Pubmed, and Citeseer are verified, and the division of train set and test set is the same as that of Section 4.3. Ten experiments are conducted, and each base classifier are trained with 100 epochs and 200
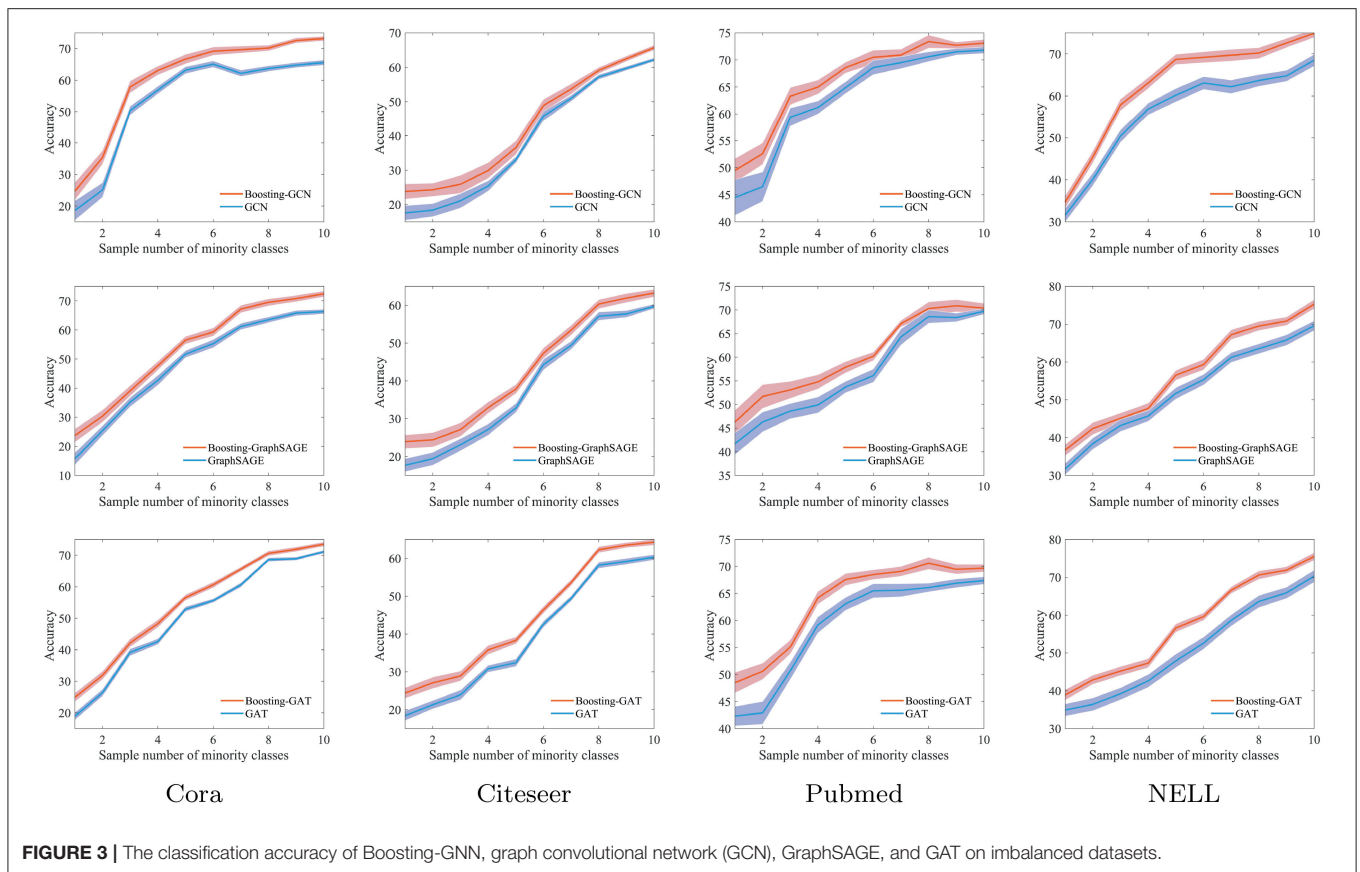
**FIGURE 3 |** The classification accuracy of Boosting-GNN, graph convolutional network (GCN), GraphSAGE, and GAT on imbalanced datasets.

**TABLE 3 |** Results of Boosting-GCN with varying numbers of base classifiers in terms of accuracy (in percentage).

| Numbers of base classifiers | epoch:100 | | | epoch:200 | | |
|---|---|---|---|---|---|---|
| | **Cora** | **Citeseer** | **Pubmed** | **Cora** | **Citeseer** | **Pubmed** |
| 3 | **75.7 ± 2.4** | 65.5 ± 2.5 | 63.9 ± 2.4 | 75.4 ± 2.1 | 65.6 ± 1.1 | 72.0 ± 0.8 |
| 5 | 73.2 ± 0.7 | **65.7 ± 0.7** | 73.1±0.7 | **75.6 ± 2.3** | **65.9 ± 0.5** | 73.1 ± 1.1 |
| 7 | 73.5 ± 1.4 | 64.5 ± 0.5 | **73.5 ± 1.4** | 74.1 ± 2.7 | 64.7 ± 0.4 | **73.5 ± 0.8** |
| 9 | 72.0 ± 0.5 | 63.6 ± 0.5 | 72.0 ± 0.5 | 73.9 ± 2.0 | 64.2 ± 0.3 | 72.6 ± 1.1 |
| 11 | 73.0 ± 0.7 | 64.5 ± 0.6 | 73.0 ± 0.7 | 74.1 ± 2.3 | 65.1 ± 0.3 | 71.5 ± 0.7 |

*The highest performance of models is highlighted in boldface.*

epochs. The training samples are randomly selected for each experiment.

To sum up, when the number of base classifiers is small, the classification accuracy increases with the number of base classifiers. When the number of base classifiers reaches a certain degree, the accuracy decreases due to overfitting.

## 4.6. Tolerance to Feature Noise

The proposed method is tested under feature noise perturbations by removing node features randomly (Abu-El-Haija et al., 2019). This test is practical, because, in the Citation networks datasets, features could be missing as the authors

article might forget to include relevant terms in the article abstract. By removing different features from a node, the performance of Boosting-GNN, GCN, GraghSAGE, and GAT is compared.

**Figure 4** shows the performance of different methods when features are removed. As the number of removed features is increased, Boosting-GNN achieves better performance than GCN, GraghSAGE, and GAT. The greater the proportion of features removed, the greater the performance advantage of Boosting-GNN over other models. This suggests that our approach can restore the deleted features to some extent by pulling in the features directly from nearby and distant neighbors.
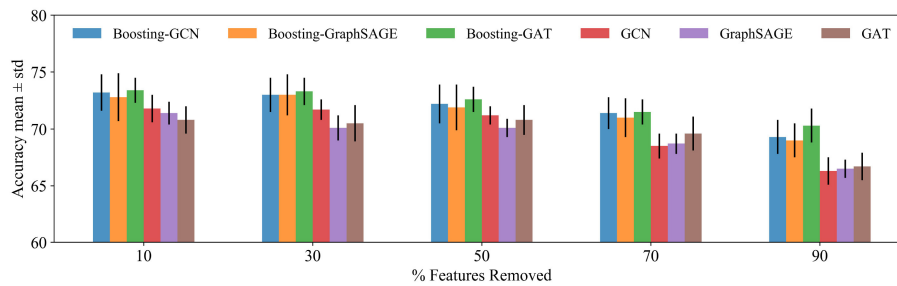
**FIGURE 4 |** Classification accuracy for the Cora dataset. The features are removed randomly, and the result of 10 runs is averaged. A different random seed is used for every run (i.e., removing different features from each node), but the same 10 random seeds are used across models.
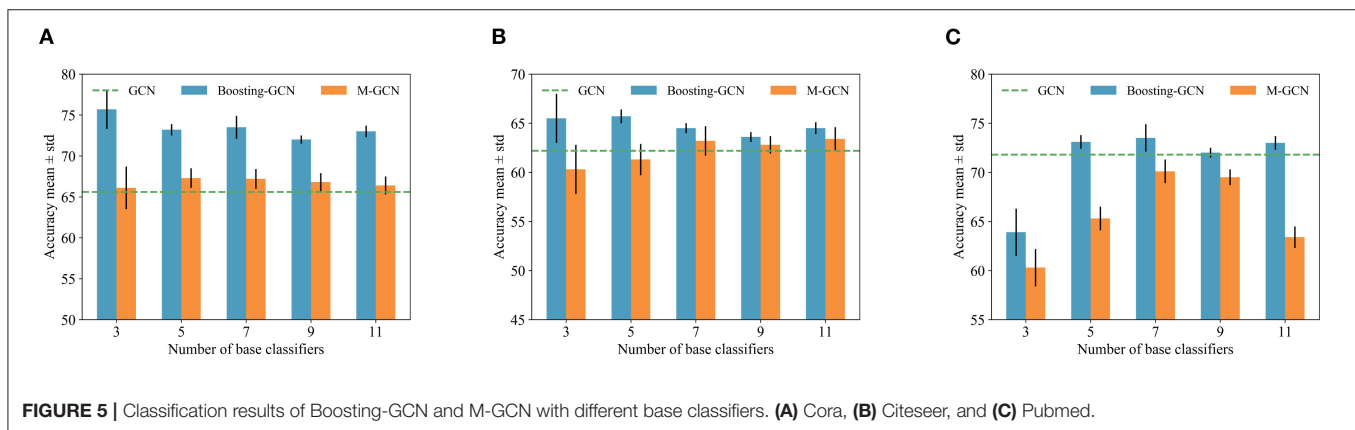


**FIGURE 5 |** Classification results of Boosting-GCN and M-GCN with different base classifiers. **(A)** Cora, **(B)** Citeseer, and **(C)** Pubmed.

## 4.7. Why Ensemble Method Useful?

This section analyzes why the ensemble learning approach works on imbalanced datasets and the advantages of Boosting-GNN over traditional GNN. The process of ensemble learning can be divided into two steps:

1) Generating multiple base classifiers for integration. Our model could adjust the weight of samples, adopt specific strategies to reconstruct the dataset, and assign smaller weights to the determined samples and larger weights to the uncertain samples. It makes subsequent base classifiers focus more on samples that are difficult to be classified. In general, the samples of minority classes in imbalanced datasets are more likely to be misclassified. By changing the weights of these samples, subsequent base classifiers can focus more on these samples.

2) Combining the results of the base classifiers. The weight of the classifier is obtained according to the error of the classifier. The base classifier with high classification accuracy has greater weight and a greater influence on the final combined classifier. In contrast, the base classifier with low classification accuracy has less weight and impact on the final combined classifier.

We independently trained $M$ GCNs using the same strategy described in Equation (11) and named this method M-GCN. We compare Boosting-GNN with M-GCN, which is trained according to the hard voting frameworks. Using the synthetic imbalanced datasets in Section 4.3, we changed $M$ and conducted several experiments. Ten runs with different random seeds were conducted to calculate the mean and SD. The experimental results are shown in **Figure 5**, and the classification results of GCN are represented by dotted lines. By effectively setting the number of base classifiers, Boosting-GCN significantly improves classification accuracy compared with M-GCN and GCN.

Next, in order to study the misclassification of samples, we observed the confusion matrix. To increase the imbalance, $s$ is set to 5. The last class is selected as the majority class, and the other classes are selected as the minority classes for convenience. Ten experiments are conducted, and the confusion matrix of the average experimental results is shown in **Figure 6**. Compared with the confusion matrix of the classification performed by GCN, Boosting-GCN achieves a better classification performance.

Due to the sample imbalance, the classifier tends to divide the samples into the majority class, which is reflected by the fact that the last column of the confusion matrix usually has the maximum value (with the brightest color). Compared with GNN, Boosting-GNN improves the performance to a certain extent, especially on the Cora dataset. Based on the aggregation of base estimators, the values on the diagonal of the confusion matrix increase, and the values in the last column of the confusion matrix decrease.
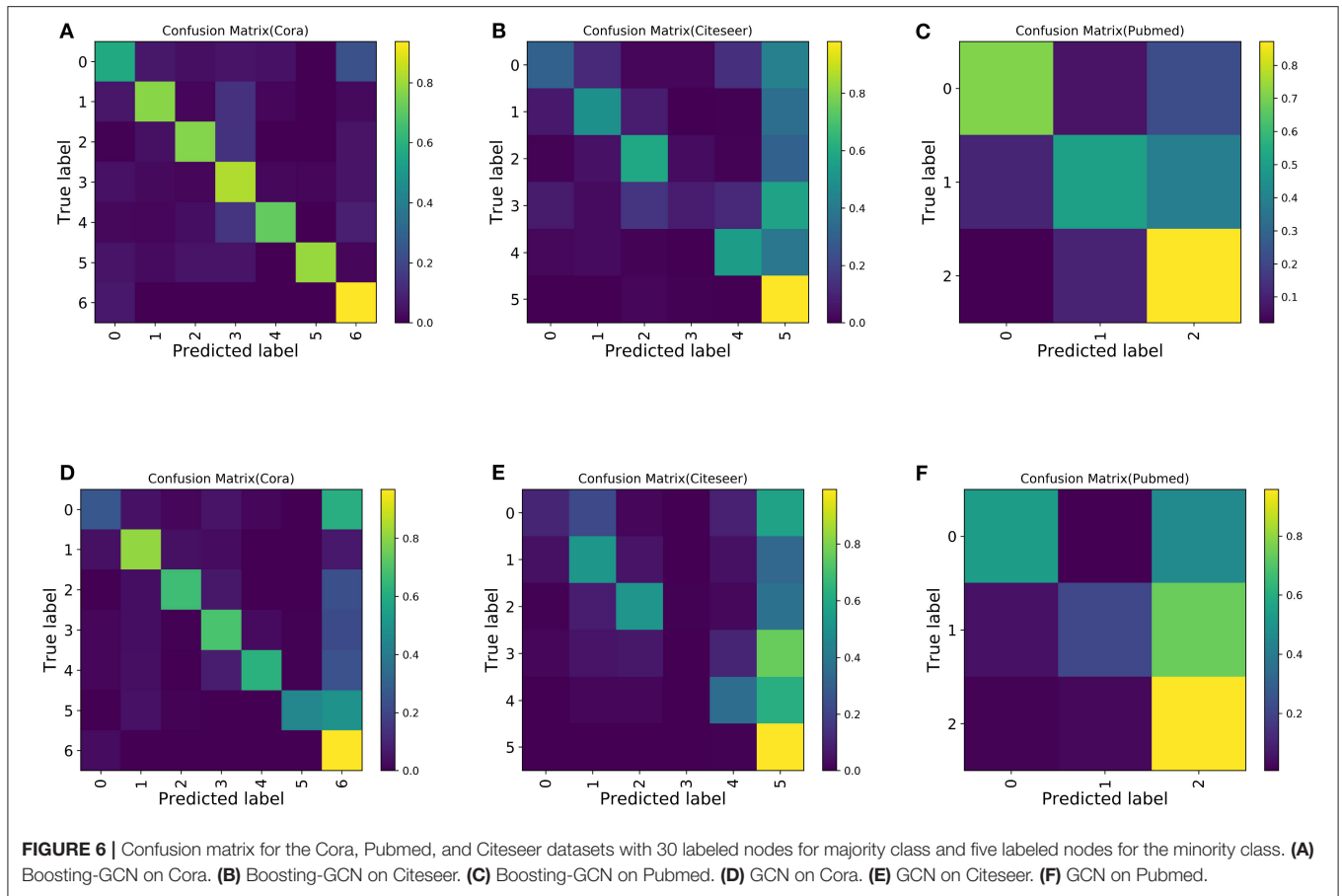
**FIGURE 6 |** Confusion matrix for the Cora, Pubmed, and Citeseer datasets with 30 labeled nodes for majority class and five labeled nodes for the minority class. **(A)** Boosting-GCN on Cora. **(B)** Boosting-GCN on Citeseer. **(C)** Boosting-GCN on Pubmed. **(D)** GCN on Cora. **(E)** GCN on Citeseer. **(F)** GCN on Pubmed.

**TABLE 4 |** Comparison of running time when using different number of GCN base classifiers.

| Method | 5-classifier | 7-classifier | 9-classifier |
|---|---|---|---|
| M-GCN | 28.76 s | 39.52 s | 51.04 s |
| Boosting-GCN-t | 10.44 s | 13.43 s | 18.03 s |
| Boosting-GCN-w/o | 18.36 s | 27.64 s | 34.83 s |

*We use Cora and train each base classifier for 100 epochs.*

In summary, Boosting-GNN integrates multiple GNN classifiers to reduce the effect of overfitting to a certain degree. Moreover, Boosting-GNN reduces the deviation caused by a single classifier and achieves better robustness. Boosting-GNN is an improvement of traditional GNN and makes AdaBoost compatible with GNN. Boosting-GNN achieves higher classification accuracy than a single GNN on imbalanced datasets with the same number of learning epochs.

## 4.8. Analysis of Training Time

In this section, we conduct a time-consuming analysis of the experiment. We measure the training time on an NVIDIA Tesla V100 GPU. The time required to train the original GCN model for 100 epochs is 6.11s. The time consumed by M-GCN and Boosting-GCN is shown in the **Table 4**. Boosting-GCN-t and

Boosting-GCN-w/o denote Boosting-GCN with transfer learning and Boosting-GCN without migration learning, respectively.

Compared to GCN, Boosting-GCN consumes exponentially more time. However, Boosting-GCN reduces the training time by about 50% compared to M-GCN. The application of transfer learning can significantly reduce the time consumed, and models can achieve similar accuracy.

## 5. CONCLUSION

A multi-class AdaBoost for GNN, called Boosting-GNN, is proposed in this article. In Boosting-GNN, several GNNs are used as base estimators, which are trained sequentially. Also, the errors of a previous GNN are used to update the weights of samples for the next GNN to improve performance. The weights of training samples are incorporated in to the cross-entropy error function in the GNN back propagation learning algorithm. The appliance of transfer learning can significantly reduce the time consumed for computation. The performance of the proposed Boosting-GNN for processing imbalanced data is tested. The experimental results show that Boosting-GNN achieves better performance than state-of-the-arts on synthetic imbalanced datasets, with an average performance improvement of 4.5%.

# DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: GitHub, https://github.com/tkipf/gcn/tree/master/gcn/data.

# AUTHOR CONTRIBUTIONS

SS performed the data analyses and wrote the manuscript. KQ and SY designed the algorithm. LW and JC analyzed the data.

BY did supervision and project administration. All authors listed have made a substantial, direct, and intellectual contribution to the work and approved it for publication.

# FUNDING

# REFERENCES

Abu-El-Haija, S., Kapoor, A., Perozzi, B., and Lee, J. (2019). "N-gcn: Multi-scale graph convolution for semi-supervised node classification," in *UAI* (Tel Aviv-Yafo).

Bai, L., Yao, L., Li, C., Wang, X., and Wang, C. (2020). Adaptive graph convolutional recurrent network for traffic forecasting. *ArXiv, abs/2007.02842.*

Blaszczynski, J., and Stefanowski, J. (2015). Neighbourhood sampling in bagging for imbalanced data. *Neurocomputing* 150, 529–542. doi: 10.1016/j.neucom.2014.07.064

Buda, M., Maki, A., and Mazurowski, M. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Netw.* 106, 249–259. doi: 10.1016/j.neunet.2018.07.011

Byrd, J., and Lipton, Z. C. (2019). "What is the effect of importance weighting in deep learning?" in *ICML* (Long Beach, CA).

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E. R., and Mitchell, T. M. (2010). "Toward an architecture for never-ending language learning," in *AAAI* (Atlanta, GA).

Chawla, N. V., Bowyer, K., Hall, L., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* 16, 321–357. doi: 10.1613/jair.953

Chawla, N. V., Lazarevic, A., Hall, L., and Bowyer, K. (2003). "Smoteboost: Improving prediction of the minority class in boosting," in *Knowledge Discovery in Databases: PKDD 2003. PKDD 2003. Lecture Notes in Computer Science, Vol. 2838*, eds N. Lavra, D. Gamberger, L. Todorovski, and H. Blockeel (Berlin; Heidelberg: Springer).

Chen, L., Wu, L., Hong, R., Zhang, K., and Wang, M. (2020). "Revisiting graph based collaborative filtering: a linear residual graph convolutional network approach," in *Proceedings of the AAAI Conference on Artificial Intelligence* (New York, NY), 27–34.

Cui, Y., Jia, M., Lin, T.-Y., Song, Y., and Belongie, S. J. (2019). "Class-balanced loss based on effective number of samples," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Long Beach, CA: IEEE), 9260–9269.

Cui, Y., Song, Y., Sun, C., Howard, A., and Belongie, S. J. (2018). "Large scale fine-grained categorization and domain-specific transfer learning," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT: IEEE), 4109–4118.

Douzas, G., and Baç ao, F. (2019). Geometric smote a geometrically enhanced drop-in replacement for smote. *Inf. Sci.* 501, 118–135. doi: 10.1016/j.ins.2019.06.007

Galar, M., Fernández, A., Tartas, E. B., and Herrera, F. (2013). Eusboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. *Pattern Recognit.* 46, 3460–3471. doi: 10.1016/j.patcog.2013.05.006

Gupta, A., Dollár, P., and Girshick, R. B. (2019). "Lvis: a dataset for large vocabulary instance segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Long Beach, CA: IEEE), 5351–5359.

Hai-xiang, G., Yi-jing, L., Ya-nan, L., Xiao, L., and Jin-ling, L. (2016). Bpso-adaboost-knn ensemble learning algorithm for multi-class imbalanced data classification. *Eng. Appl. Artif. Intell.* 49, 176–193. doi: 10.1016/j.engappai.2015.09.011

Hamilton, W. L., Ying, Z., and Leskovec, J. (2017). "Inductive representation learning on large graphs," in *NIPS* (Long Beach, CA).

Han, H., Wang, W., and Mao, B. (2005). "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science, Vol. 3644*, eds D. S. Huang, X. P. Zhang, and G. B. Huang (Berlin; Heidelberg: Springer).

Hastie, T., Rosset, S., Zhu, J., and Zou, H. (2009). Multi-class adaboost. *Stat. Interface.* 2, 349–360. doi: 10.4310/SII.2009.v2.n3.a8

He, H., Bai, Y., Garcia, E. A., and Li, S. (2008). "Adasyn: adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (Hong Kong: IEEE), 1322–1328.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Las Vegas, NV: IEEE), 770–778.

Huang, W., Yan, H., Cheng, K., Wang, C., Li, J., Wang, Y., et al. (2021a). A neural decoding algorithm that generates language from visual activity evoked by natural images. *Neural Netw.* 144, 90–100. doi: 10.1016/j.neunet.2021.08.006

Huang, W., Yan, H., Cheng, K., Wang, Y., Wang, C., Li, J., et al. (2021b). A dual–channel language decoding from brain activity with progressive transfer training. *Hum. Brain Mapp.* 42, 5089–5100. doi: 10.1002/hbm.25603

Japkowicz, N., and Stephen, S. (2002). The class imbalance problem: a systematic study. *Intell. Data Anal.* 6, 429–449. doi: 10.3233/IDA-2002-6504

Khoshgoftaar, T., Fazelpour, A., Dittman, D., and Napolitano, A. (2015). "Ensemble vs. data sampling: which option is best suited to improve classification performance of imbalanced bioinformatics data?" in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)* (Vietri sul Mare, SA), 705–712.

Kipf, T. N., and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *ICLR, abs/1609.02907.*

Li, G., Müller, M., Thabet, A. K., and Ghanem, B. (2019). "Deepgcns: can gcns go as deep as cnns?" in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (Seoul), 9266–9275.

Li, J., Peng, H., Cao, Y., Dou, Y., Zhang, H., Yu, P. S., et al. (2021). Higher-order attribute-enhancing heterogeneous graph neural networks. *ArXiv, abs/2104.07892.* doi: 10.1109/TKDE.2021.3074654

Lin, T.-Y., Goyal, P., Girshick, R. B., He, K., and Dollár, P. (2017). "Focal loss for dense object detection," in *2017 IEEE International Conference on Computer Vision (ICCV)* (Venice: IEEE), 2999–3007.

Lin, T.-Y., Maire, M., Belongie, S. J., Hays, J., Perona, P., Ramanan, D., et al. (2014). Microsoft coco: common objects in context. *ECCV, abs/1405.0312.* doi: 10.1007/978-3-319-10602-1_48

Lv, Q., Feng, W., Quan, Y., Dauphin, G., Gao, L., and dao Xing, M. (2021). Enhanced-random-feature-subspace-based ensemble cnn for the imbalanced hyperspectral image classification. *IEEE J. Select. Top. Appl. Earth Observat. Remote Sens.* 14, 3988–3999. doi: 10.1109/JSTARS.2021.3069013

Nanni, L., Fantozzi, C., and Lazzarini, N. (2015). Coupling different methods for overcoming the class imbalance problem. *Neurocomputing* 158, 48–61. doi: 10.1016/j.neucom.2015.01.068

Peng, H., Zhang, R., Dou, Y., Yang, R., Zhang, J., and Yu, P. S. (2021). Reinforced neighborhood selection guided multi-relational graph neural networks. *ArXiv, abs*/2104.07886.

Ramentol, E., Mota, Y., Bello, R., and Herrera, F. (2011). Smote-rsb*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using smote and rough sets theory. *Knowl. Inf. Syst.* 33, 245–265. doi: 10.1007/s10115-011-0465-6

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.* 115, 211–252. doi: 10.1007/s11263-015-0816-y

Seiffert, C., Khoshgoftaar, T., Hulse, J., and Napolitano, A. (2010). Rusboost: a hybrid approach to alleviating class imbalance. *IEEE Trans. Syst. Man Cybern. A Syst. Hum.* 40, 185–197. doi: 10.1109/TSMCA.2009.2029559

Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI Mag.* 29, 93–106. doi: 10.1609/aimag.v29i3.2157

Shen, L., and Lin, Z. (2016). "Relay backpropagation for effective learning of deep convolutional neural networks," in *Computer Vision–ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, Vol. 9911*, eds B. Leibe, J. Matas, N. Sebe, and M. Welling (Cham: Springer), 467–482.

Sun, K., Lin, Z., and Zhu, Z. (2021). Adagcn: adaboosting graph convolutional networks into deep models. *ArXiv, abs*/1908.05081.

Taherkhani, A., Cosma, G., and Mcginnity, T. (2020). Adaboost-cnn: An adaptive boosting algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning. *Neurocomputing* 404, 351–366. doi: 10.1016/j.neucom.2020.03.064

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. *ICLR, abs*/1710.10903.

Wang, X., Ji, H., Shi, C., Wang, B., Cui, P., Yu, P., et al. (2019). "Heterogeneous graph attention network," in *The World Wide Web Conference* (San Francisco, CA).

Wang, Y.-X., Ramanan, D., and Hebert, M. (2017). "Learning to model the tail," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)* (Long Beach, CA).

Wei, X.-S., Cui, Q., Yang, L., Wang, P., and Liu, L. (2019). Rpc: a large-scale retail product checkout dataset. *ArXiv, abs*/1901.07249.

Wu, F., Zhang, T., de Souza, A. H., Fifty, C., Yu, T., and Weinberger, K. Q. (2019). Simplifying graph convolutional networks. *ArXiv, abs*/1902.07153.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. *ICML, abs*/1603.08861.

Yin, X., Yu, X., Sohn, K., Liu, X., and Chandraker, M. (2019). "Feature transfer learning for face recognition with under-represented data," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Long Beach, CA), 5697–5706.

Yu, D., Xiong, W., Droppo, J., Stolcke, A., Ye, G., Li, J., et al. (2016). Deep convolutional neural networks with layer-wise context expansion and attention. *Proc. Interspeech* 2016, 17–21. doi: 10.21437/Interspeech.2016-251

Yu, W., and Qin, Z. (2020). Graph convolutional network for recommendation with low-pass collaborative filters. *ArXiv, abs*/2006.15516.

Zhou, B., Lapedriza, À., Khosla, A., Oliva, A., and Torralba, A. (2018). Places: a 10 million image database for scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 40, 1452–1464. doi: 10.1109/TPAMI.2017.2723009

Zhou, Q., Sun, B., Song, Y., and Li, S. (2020). "K-means clustering based undersampling for lower back pain data," in *Proceedings of the 2020 3rd International Conference on Big Data Technologies* (Qingdao).

Zitnik, M., and Leskovec, J. (2017). Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33, i190–i198. doi: 10.1093/bioinformatics/btx252

Zou, Y., Yu, Z., Kumar, B. V., and Wang, J. (2018). "Unsupervised domain adaptation for semantic segmentation via class-balanced self-training," in *ECCV* (Munich).