Evaluating Deep Graph Network Performance by Augmenting Node Features with Structural Features

Mohamad Abushofa[1], Amir Atapour-Abarghouei[2], Matthew Forshaw[1], and A. Stephen McGough[1]

[1] Newcastle University, School of Computing, Newcastle upon Tyne, United Kingdom
m.e.a.abushofa2@ncl.ac.uk, matthew.forshaw@ncl.ac.uk, stephen.mcgough@newcastle.ac.uk

[2] Durham University, Department of Computer Science, Durham, United Kingdom
amir.atapour-abarghouei@durham.ac.uk

Abstract. In machine learning, features play a vital role in modeling and understanding the data; their quality and representation essentially determine how accurate the results are. The problem is compounded in the graph-based learning paradigm when one considers how complex and interconnected the data is. However, to achieve more accurate results, augmenting graph data poses specific challenges in the field of graph learning. Feature augmentation is a critical aspect of enhancing data. Moreover, some datasets have limited features and some real datasets do not have features. In this paper, we present our approach termed Feat-Aug which is an extension of our previous work on non-parametric approaches. The aim of this work is to augment node features in graphs on parametric approaches such as Graph Neural Networks (GNNs) with the objective of improving performance in node classification tasks. Our approach combines real features, such as a bag of words in citation networks, which are typically associated with nodes, with structural features extracted at the node level, such as node degree and clustering coefficient. To further enhance these features, we leverage deep learning models to incorporate additional node-level features. The final modified features are the result of the combination of both real and structural features. To evaluate the effectiveness of the approach, we carried out extensive experiments with several real datasets. Moreover, our method consistently outperforms or achieves comparable results to Graph Neural Networks (GNNs) baselines and their variations, such as popular graph neural network models. Crucially, our approach deals with the problem of insufficient real features in certain datasets. This study is a major progression in the field through an effective node classification model. By integrating both real and structural features, our approach holds promise to improve the performance of node classification models.

Keywords: structural features · node features · concatenated features · GNN models.

1 Introduction

Graph machine learning tasks, such as drug discovery [1], recommendation systems [2], and traffic prediction [3], have been greatly enhanced by Graph Neural

Networks (GNN) and their variants. The essence of GNNs is in the message-passing mechanism, where features from adjacent nodes are incorporated into the features of a given node [4]. Although GNNs have been quite successful in recent years, their model inference performance is improved by the augmentations to nodes and graphs, as a result of modified features. The features augmentation strategies have great success in classic machine learning domains such as computer vision [5] [6] and natural language processing [7]. Convolutional neural networks use this method, and the augmentation effects have been particularly fruitful when applied to image data [8]. Unlike graph data, where each node has connections to other nodes, the image pixels do not have any connections. Several data augmentation in image classification tasks independently modify image features, including RGB values and local neighbourhoods in the images [9]. Topology-level[10] and feature-level augmentation [11] [12] [13] are the two approaches for node and graph augmentations. At the topology level, a new graph topology is generated through augmentation techniques that modify the adjacency matrix. In contrast, adversarial training improves generalisation by perturbing node features during node-level augmentation techniques [14] [12] [13]. Some methods rely on changing the edges by removing or modifying them according to the information within the graph itself [15]. Additionally, some researchers have found great success in data augmentation by using extra topological features to build a dual graph similarity matrix of the graph [16]. Adding more edges between nodes with the same label increases the probability that the central node will benefit from the propagation process, which is another strategy used by researchers [16].

Numerous real-world issues, particularly in achieving high performance in applications, such as social network analysis, protein-protein interaction prediction, and network traffic categorization, involve graph machine learning tasks including graph classification, and node classification. Therefore, to improve the performance we should consider graph augmentation by supplementing the node features. In this paper, we looked at how to combine structural and real features at the node level to enhance the node features within the network.
In graph natural networks, nodes have linked properties like age names in social networks [17] and bags of words in citation networks [18]. On the other hand, the structural features are taken from the graph structure and provide the graph with a unique signature. The node degree, node centrality, cluster coefficient, and average node degree are a few examples of structural node properties. A mix-up technique described by Wang et al. [19] involves altering the data to create new features.

To the best of our knowledge, while some previous data augmentations based on GNNs, such as the baseline Graph Convolutional Networks (GCN) [20], GraphSAGE [1], and Graph Isomorphism Network (GIN) [21], and their variants, may have implicitly used some structural features alongside real features, our method is the first to explicitly and systematically concatenate these features. This explicit approach enhances interpretability and potentially improves performance by clearly leveraging the strengths of both feature types in the context of node classification using GNNs. The implicit methods may lose their potential advantages in graph machine learning applications as they disregard the explicit evaluation of structural features by not systematically concatenating them with the real features. Therefore, they have not assessed the explicit strength of the structural features themselves and how graph deep learning learns from concatenated features. In contrast, we address this gap by explicitly taking into account both structural and actual features.

The proposed method demonstrates its effectiveness in comparison to state-of-the-art methods. Combining real and extracted characteristics can enhance our understanding of the significance of structural features for graph machine-learning tasks.

Therefore, we are taking the structural features out of the graph and concatenating them with the actual ones. Additionally, our method is different from the earlier methods found in GNNs and their variations since previous methods disregard the structural features and concentrate primarily on actual features.

To facilitate the reproducibility of our results, we have made the source code for our method publicly available. This allows other researchers to easily access and build upon our work. Overall, our contributions in this paper can be summarized as follows:

- We propose a novel method for node augmentation that combines aggregated original features and local structural features as input to the graph deep learning model.
- We identify and utilize a set of relevant structural features for node feature concatenation.
- We conduct extensive experiments to evaluate the performance of our approach on node classification.
- Our experimental results show that our method outperforms state-of-the-art approaches on node classification, demonstrating the effectiveness of our approach.
- We make the source code for our method publicly available to enable better reproducibility of the results[3].
- Our approach demonstrates that the improved performance is due to the capability of a graph deep learning model to capture and learn combined features, which benefits data with insufficient features.

2 Literature Review

Since the advent of graph neural networks, there has been a significant shift in how we handle node feature representations. Initially, the methods for embedding lookups, which are crucial for interpreting node features, were limited in their effectiveness [4]. However, GNNs have introduced advanced techniques for managing these node properties. Specifically, GNNs utilize aggregators — sophisticated mechanisms that compile and integrate features from neighboring nodes.
This aggregation process is central to GNNs' ability to effectively capture the complex relationships and interconnectedness inherent in graph data. By leveraging these aggregators, GNNs can construct a more comprehensive and nuanced representation of each node, taking into account not just node attributes but also the influence of its network context [22]. Also, it has succeeded in inductive learning [1] [23]. The graph convolution methods, which are based on

[3] https://github.com/mohamad-1977/Fet-Aug

spatial-based methods, have demonstrated significant success compared to the spectral GNNs [1] [2], which are based on convolutions using spectral graph theory [24], they are also expensive to employ as they process the whole adjacency matrix [25] [20] [26].

Researchers have been seeking ways to make machine-learning models capable of improving the generalisation and efficiency of graph machine-learning tasks such as node classifications, link prediction, and graph classifications. Recently, they achieved significant success in augmentations techniques in many domains such as computer vision [5] and natural language processing [7].

Augmentation in graph machine learning tasks is also a popular topic. The graph augmentations have two approaches: topology-level augmentation and feature-level augmentation.

Works done by Chen et al. [27], and Rong et al. [28], focused on graph topological augmentation, which involves adding or removing edges or nodes and consequently changing the graph structures. To improve the generalization and performance of the models, numerous studies, like work done by Yoo et al. [29], focus on random edge insertion or deletion. The work of Yoo et al. [30] performs node splitting or merging, and You et al. has implemented the subgraph sampling [29]. These methods make variations in graph topology and have effectively leveraged the relationships in graph data and improved performance.

Alternatively, node-level augmentation approaches work on altering node features or edge features in the graph. Works done by Yang et al. [31] and by Feng et al. [12] use the perturbations of the node features based on adversarial training. These methods attempt to increase the diversity of node feature representations, which then increases the performance of the models.

Node level and topology augmentation have been showing improved outcomes in many graph machine learning tasks, for instance, recommendation systems [32], drug discovery [33], and traffic prediction [34]. However, all aforementioned methods neglect the structural features and just focus on node-associated features. Thus, this paper proposes a novel approach by incorporating both real and structural features at the node level and evaluating GNNs baseline models. Our approach advances how structural features enhance deep graph learning.

3 Motivation and Requirements

3.1 Motivation

Graph neural networks have gained considerable interest in recent years for their capacity to capture complex connections among nodes in a graph, making them well-suited for node classification. Nevertheless, to qualify this achievement of GNN, the structure depends on the presence of node features as illustrated in Eq. 1, that are rich and high-dimensional. However, in the most realistic graphs, the quantity and quality of available features are limited and some of the real data has no features [35], which weakens the performance of GNNs. In addition, all previous works based on GNN baselines, GCN [20], GraphSAGE [1], Graph Isomorphism Network [21], and their variants, neglected to integrate structural features by concatenating them with real features, which can give more information about the nodes. There have been lots of attempts to augment graph data as we mentioned in the literature. One of these approaches is the Mixup introduced by Wang et al. [19] that consists of perturbing the data to create new or enhanced features. While this method has also demonstrated some good progress; this and all methods in the litterateurs are hampered by just perturbing the data and do not take the structural features into account, To overcome this drawback we suggest a different way of node augmentation by performing feature augmentation(Feat-Aug) that combines real and structural features, creating more meaningful node embedding. The proposed approach is motivated by the Mixup technique yet varies in that, it includes a combination of actual and extracted features, instead of perturbing the data.

$$h_i^{(l)} \leftarrow \text{AGGREGATE}\left(h_i^{(l-1)}, h_j^{(l-1)} : j \in N(i)\right) \oplus W^{(l)} \tag{1}$$

3.2 Requirements

Incorporation of the suggested method of feature augmentation has to satisfy several requirements.

- A graph must be available for analysis, represented in a format that can be processed by a GNN model (e.g. Networkx)
- Real features must be available for the nodes in the graph. These may include domain-specific knowledge about the nodes, such as their function or role in a particular system.
- It must be possible to compute the structural features for nodes in the graph.
- A GNN model must be available for training and testing. This model should be capable of processing node features and generating node embedding.
- A node classification task must be defined, along with an objective function for measuring the performance of the model.
- A method for sampling and interpolating nodes, such as the mixup technique used in [19], must be implemented to generate the augmented node features.

4 Methodology

In our approach to enhancing node features, we drew inspiration from concepts explored in our prior research, as detailed in [36], which involved modifying node features through non-parametric approaches in graph classification. In contrast to this, our proposal focuses on parametric approaches (graph deep learning), specifically graph neural networks baseline and its variants. By incorporating structural features derived from the node-level, we aim to uncover structural properties inherent in the graph data. Leveraging these structural features could potentially enhance the accuracy of classification models across a diverse range of tests. We further strengthen this method by using deep graph learning models to reinforce structural features with more node-level attributes as shown in figure 1. Our approach performance is evaluated by applying the modified features on graph neural network baselines, such as GCN [20] and GraphSAGE [1] and recent graph augmentation based on GCN, Mix-up [19], in which data is perturbed to produce new features. However, in our work, we do not alter anything within the data; our work combines structural and real features at the node level to produce a modified feature. Our approach involves the following these steps:

1. Mining the node features at local-level features.
2. Create the modified features by combining both extracted features and node-associated features.
3. Use the modified attributes as inputs to graph deep learning models.
4. Employ deep learning models to evaluate the performance with the modified features.

Algorithm 1 Feat-Aug

```
Input: Graph $G$
Output: Node embedding
 1: // Extract features from nodes (see Algorithm 2)
 2: $FG \leftarrow \text{ExtractedFeatures}(G)$
 3: $C_{mf} \leftarrow \{\text{original features}\} \cup \{\text{structural features}\}$
 4: for each node $v \in G$ do
 5:     $h_v^{(0)} \leftarrow x_v^{(0)} \,||\, f_v^{(0)}$
 6:     for $k = 1$ to $K$ do
 7:         $\text{msg}_v^{(k)} \leftarrow \text{aggregate}\left(\left\{(h_v^{(k-1)}, h_u^{(k-1)}) \mid u \in \text{neighbors}(v)\right\}\right)$
 8:         $h_v^{(k)} \leftarrow \text{combine}\left(h_v^{(k-1)}, \text{msg}_v^{(k)}\right)$
 9:     end for
10:     $L \leftarrow \text{loss}(h_v^{(k)}, y)$
11:     $h_v^{(t)} \leftarrow h_v^{(t-1)} + \text{gradient}(L, h_v^{(t)})$
12: end for
13: return node embedding
```

Algorithm 2 Extracted Features

```
Require: Graph $G$
1: $FG \leftarrow []$                                   ▷ Initialize feature matrix for $G$
2: for each node $v$ in $G$ do                          ▷ Extract features for each node $v$ in $G$
3:     $\underline{features} \leftarrow [\delta(v), C(v), K(v), c(N(v)), \overline{d(N(v))}, |E_{\text{ego}}(v)|,$
4:     $\overline{c(N(v))}, |E_{\text{oego}}(v)|, |N(\text{ego}(v))|]$
5:     $FG \leftarrow FG \cup \{features\}$
6: end for
7: return $FG$                                         ▷ Return a matrix of node × features
```

Algorithm 1 presents our approach framework, which consists of three distinct steps: feature extraction (Algorithm 2), concatenating real features with extracted features (Algorithm 3), feature normalization, and evaluating the node classification performance.

Feature Extraction To encode the local features of the graph (see Algorithm 2), several node-level metrics are extracted. While there are many available metrics for node features, each one has distinct characteristics in terms of its sensitivity to topology and its execution time [36].

Algorithm 3 Concatenated Features

```
1: Input: Graph $G$, feature matrix $X$ (Real features matrix), additional features
   $\{F(v)\}$ for each node $v$
2: Output: $M$ concatenated features matrix
3: Initialize $M$ with dimensions $(\text{num\_nodes}, \dim(X) + \dim(\{F(v)\}))$
4: for each node $v \in G$ do
5:     $concatenated\_features \leftarrow \text{concatenate}(X[v], \{F(v)\})$
6:     $M[v] \leftarrow concatenated\_features$
7: end for
8: Return: $M$                    ▷ Return a matrix $M$ with dimensions
   $(\text{num\_nodes}, \dim(X) + \dim(\{F(v)\}))$
```

Our approach extracts 9 properties from every vertex in a graph. Through experimentation, we have determined that the following 9 feature metrics give the best balance between classification and run-time. Alternatively, if there are other significant attributes of a graph, different metrics could be employed. A value is extracted for each node for each of the 9 node properties given below $v \in V$:

1. **Node degree** δ: The degree of a node in a network refers to the number of connections it has with other nodes.
2. **Clustering coefficient** c: the clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together.
$$c_u = \frac{|(v_1, v_2) \in E : v_1, v_2 \in N|}{(2^d u)}.$$
The numerator represents the count of edges that connect neighbouring nodes of node u (where we use $N(u) = \{v \in V : (u, v) \in E\}$ to denote the node neighbourhood). The denominator determines the total number of node pairings in nearby of node u. [4]
3. **the average degree of neighbours** k: The average degree of each node nearby is calculated.
$$k_{nn,i} = \frac{1}{|N(i)|} \sum_{j \in N(i)} k_j$$
where $N(i)$ represents the set of neighbouring nodes of node i, and k_j denotes the degree of node j which is a member of $N(i)$ [36].
4. **Average clustering of neighbourhood**: The average clustering score of each vertex is calculated by taking the mean of the local clustering scores in its neighbourhood.
5. **The average number of node i's two-hops:** it is two-hop away neighbors, denoted as $\overline{dN(i)}$, is computed as $\frac{1}{d_i} \sum_{j \in N(i)} d_j$.
6. **The average clustering coefficient of** $N(I)$**:** The average clustering coefficient of $N(i)$, represented as $\overline{cN(i)}$, is computed as $\frac{1}{d_i} \sum_{j \in N(i)} c_j$.
The variable c_j represents the local clustering score calculated in step two [36].
7. **The number of edges in node** i**'s egonet:** The number of edges in node i's egonet is denoted as $|E_{ego(i)}|$, and ego(i) returns the egonet of node i [37].

Dataset	**#Nodes**	**#Edges**	**#Classes**	**#Features**
Citeseer	3,327	4,732	6 (s)	3,703
Cora	2,708	5,429	7 (s)	1,433
Pubmed	19,717	44,338	3 (s)	500
Amazon (Photo)	7,650	143,662	8 (s)	745
Amazon (Computers)	13,752	287,209	10 (s)	767
Flickr	1,598,960	132,169,734	107 (s)	200

Table 1: Data statistics for node classification datasets. The letter 's' represents single-label classification. [19]

where the egonet of a node includes the node itself, its immediate neighbors, and all the edges among them.

8. **The number of outgoing edges from** $ego(I)$**:** The number of outgoing edges from $ego(i)$ is denoted as $|E_{o,ego(i)}|$ [37].
9. **The number of neighbors of** $ego(I)$**:** The number of neighbors of $ego(i)$ is denoted as $|N(ego(i))|$[37].

Concatenating Features As described in Algorithm 3, the proposed approach to feature augmentation involves extracting structural features, defined previously, from the node level and concatenating them with real features associated with the nodes as shown in Eq.2. where f_i represents extracted features and h_i represents the real features at level 0.

Features Normalization A critical step in this preprocessing pipeline involved normalizing the data features. This normalisation technique was applied to mitigate the potential effects of varying scales and magnitudes across different features. By standardising the feature values to a common scale, we aimed to create a more uniform and interpretable representation space for the model. This process not only facilitated comparisons between different features but also prevented certain features from dominating the learning process because of their larger scales. Furthermore, normalization promoted fairer contributions from all features and improved model robustness and generalization by reducing sensitivity to the absolute magnitudes of feature values. Overall, normalisation of the data features was integral to enhancing the stability, reliability, and performance of our model.

4.1 Node Classification Method

Concatenated Features with the GNN Baseline After the three steps mentioned above, the goal of this process is to use these features to generate node embeddings that capture the relationships between nodes in the graph, and then use these embeddings for node classification, as shown in Algorithm 1. Therefore, we incorporate the concatenated features into works such as GNN baselines GCN[20], GraphSAGE [1], GraphSAINT[23], and recent node augmentation Mixup [19]. In the GNN baseline, the "message passing" process of a GNN layer, as described in Eq. (1) involves updating the representation of node i by aggregating the representations of itself and its neighbors. By stacking L layers, GNNs enable the final-layer prediction of node i to be based on its L-hop neighborhood. This is commonly referred to as the embedding of node i. Thus, in our method, the hidden states for each node in the graph are initialized by setting $h^{(0)}$ to the node attributes $x^{(0)}$, and then structural features are added to the initial features to produce the modified features $h^{(0)}$, as described in the previous sections. Then, for each layer l in the GNN model, from 1 to l - 1, update the hidden states of the nodes using the AGGREGATE function and the hidden states of the neighbors of each node.

Concatenated Features with Mixup we incorporate the concatenation data into the work of [19], as shown in Algorithm 4, in which the hidden states for each node in the graph are initialized by setting $h(0)$ to the node attributes x_i. For each layer l in the GNN model, from 1 to L - 1, update the hidden states of the nodes using the AGGREGATE function and the hidden states of the neighbors of each node. For each node in the graph, sample another node j from the set of all nodes. Sample a value for λ from a Beta distribution with parameter α. Interpolate the attributes and labels of the two nodes using λ, creating new attributes $\tilde{x_i}$ and $\tilde{y_i}$ for the interpolated node. Initialize the hidden state of the interpolated node $\tilde{h(0)}$ using the interpolated attributes $\tilde{x_i}$. For each layer l in the GNN model, from 1 to L, update the hidden states of the interpolated node using the AGGREGATE function and the hidden states of the neighbors of the interpolated node. Calculate the classification loss L on the interpolated hidden states and labels. Use backpropagation to minimize L and update the model parameters $W(l)$.

$$h^{(l)} = f_i \,||\, {h_i}^{(l)} \tag{2}$$

5 Experimental Setup and Results

This section evaluates the proposed method using comprehensive experiments. Our proposed approach aims to improve the features of nodes in a graph, enabling it to effectively perform node classification in both transductive and inductive settings. The transductive setting means that features at all nodes are available during training, except labels of test nodes, whereas in the inductive setting, none of the feature vectors are available for validation or testing.

5.1 Transductive Setting

A comprehensive evaluation was conducted to assess the effectiveness of our method. Many GNN baselines were compared with our approach, including the Mixup method [19] as well as widely-used GNN models such as GCN [20], GAT [38], and LGCN [39]. We employ the datasets mentioned in Table 1, to evaluate the efficacy of these node classification models. The datasets can be accessed by the public on the Torch Geometric website [40].

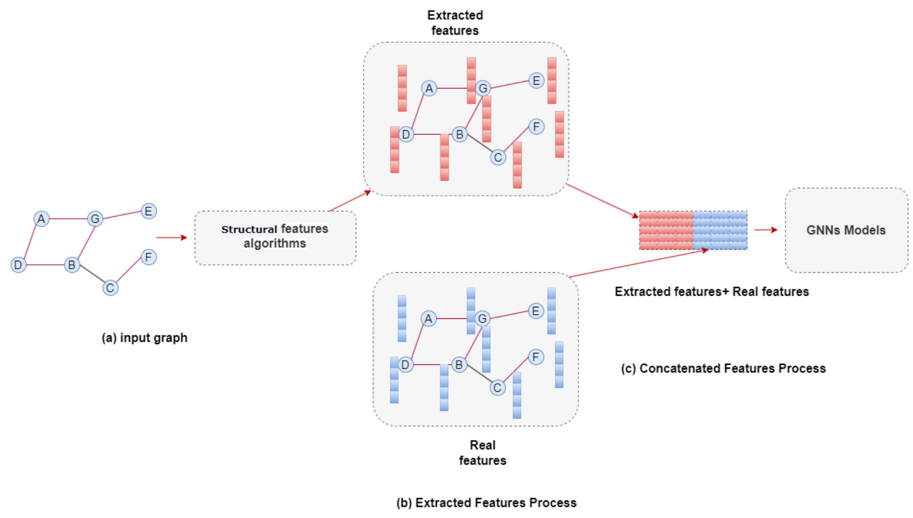

Extracted features
Structural features algorithms
GNNs Models
Extracted features+ Real features
(a) input graph
(c) Concatenated Features Process
Real features
(b) Extracted Features Process

Fig. 1: Architecture Diagram of GNN Models with Concatenated Features, This figure depicts the architecture diagram of GNN models with concatenated features. (a) Input: A labeled graph is provided as input to the extracted features algorithms.(b) Extracted Features Process: The extracted features algorithms process the input graph to extract relevant features. (c) Concatenated Features Process: Once the extracted features process is complete, the features are concatenated to form a feature matrix. Forwarding Concatenated Features to GNN Models: The concatenated feature matrix is then forwarded to the GNN models for further processing and analysis.

We divide the nodes in each graph into three sets: 60%, 20%, and 20% for training, validation, and testing, respectively. We conduct 50 trials of experiments, initializing the weights randomly for each split. Notably, the extracted features have an impact on the GNN baseline methods, as shown in Table 3..

Results Table 3 shows the results of our proposed approach against the baselines. We use the same experimental setup as in the original Mixup paper [19] to compare our proposed method to this approach. Our proposed method overcomes all methods on the dataset. Our approach achieves, on the Cora dataset, an improvement of 1% over the Mixup [19], and nearly 2% on the CiteSeer and Pumbed dataset; these results show how the structural features impact the graph deep learning models performance.

5.2 Inductive Setting

The datasets utilised for evaluation are Flickr, Amazon(Computer), and Amazon(Photo) [41]. The size of these datasets exceeds the capacity of full-batch implementations of GCN architectures to effectively handle them. Therefore, to compare, we employ the more scalable GraphSAGE [1] and GraphSAINT [23] as the baselines. These methods have similar goals to our proposed work but differ in their approach to augmenting the features of the nodes. The research

Algorithm 4 Feat-Aug

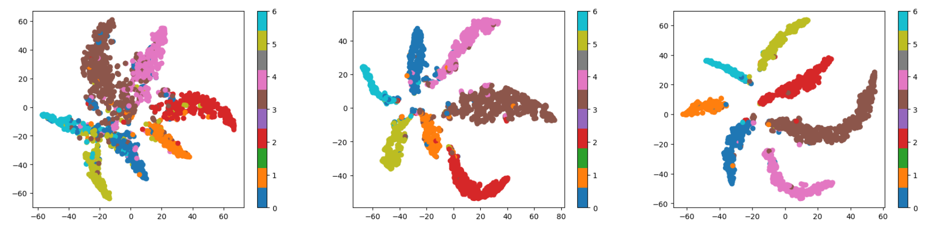
```
Input: Graph $G$
Output: Node embedding
 1: // Extract features from nodes (see Algorithm 2)
 2: $FG \leftarrow$ ExtractedFeatures($G$)
 3: $C_{mf} \leftarrow \{$original features$\} \cup \{$structural features$\}$
 4: for each node $v \in G$ do
 5:     $h_v^{(0)} \leftarrow x_v^{(0)} + f_v^{(0)}$
 6: end for
 7: for $l \leftarrow 1$ to $L-1$ do
 8:     for each node $i \in G$ do
 9:         $h_i^{(l)} \leftarrow \mathrm{AGGREGATE}(h_i^{(l-1)}, \{h_j^{(l-1)}\}, \{W_{ij}^{(l)}\})$
10:     end for
11: end for
12: for each node $i \in G$ do
13:     Sample node $j$ from $G$
14:     $\lambda \leftarrow \mathrm{Beta}(\alpha, \alpha)$
15:     $\widetilde{x}_{ij} \leftarrow \lambda x_i + (1-\lambda)x_j$
16:     $\widetilde{y}_{ij} \leftarrow \lambda y_i + (1-\lambda)y_j$
17:     $\widetilde{h}_{ij}^{(0)} \leftarrow \widetilde{x}_{ij}$
18:     for $l \leftarrow 1$ to $L$ do
19:         $\widetilde{h}_{ij,i}^{(l)} \leftarrow \mathrm{AGGREGATE}(\widetilde{h}_{ij,i}^{(l-1)}, \{h_k^{(l-1)}\}, \{W_{ik}^{(l)}\})$
20:         $\widetilde{h}_{ij,j}^{(l)} \leftarrow \mathrm{AGGREGATE}(\widetilde{h}_{ij,j}^{(l-1)}, \{h_k^{(l-1)}\}, \{W_{jk}^{(l)}\})$
21:         $\widetilde{h}_{ij}^{(l)} \leftarrow \lambda \widetilde{h}_{ij,i}^{(l)} + (1-\lambda)\widetilde{h}_{ij,j}^{(l)}$
22:     end for
23: end for
24: Calculate classification loss $L$ on $\{\widetilde{h}_{ij}^{(L)}, \widetilde{y}_{ij} \mid i \in G\}$
25: Back-propagation on $\{W^{(l)} \mid l \in \{1, 2, \ldots, L\}\}$ for minimizing $L$
```

experiments were conducted for a total of 100 trials, utilising random weight initialization. The findings have been documented in Table 2. GraphSAGE utilises mean, LSTM, and max-pooling as the aggregators, respectively.

Results We use GraphSAGE-mean and GraphSAINT-GCN as the implementations for our suggested method to investigate whether Feat-AUG can enhance the performance of GNNs in the inductive setting. Concatenating features improves the test F1-micro scores of GraphSAGE-mean by 1.9% on Flickr and yields a 1% improvement over DropEdge + GraphSAGE on the same data. Additionally, it improves the scores by 12% on Amazon (Computers) and 4% on Amazon(Photo) compared to GraphSAGE. Our method + GraphSAGE increases the scores by nearly 10% on Amazon (Computers) and by 5% on Amazon (Photo) compared to DropEdge + GraphSAGE. Furthermore, our method + GraphSAINT achieves nearly similar results on the Flickr dataset compared to GraphSAINT, and on Amazon Computers and Photo, it improves the results by 6% and 2% respectively. Consequently, our proposed method improves them to surpass the performance of the standard approaches.

Table 2: Classification accuracy comparison for inductive setting

Method	Flickr	Amazon (Computer)	Amazon (Photo)
	Accuracy	Accuracy	Accuracy
GraphSAGE-mean[1]	50.1 ± 1.1	80.91 ± 0.83	87.25 ± 0.50
GraphSAGE-LSTM[1]	50.3 ± 1.3	80.25 ± 0.43	87.1 ± 0.30
GraphSAGE-pool[1]	50.0 ± 0.8	80.76 ± 0.23	87.34 ± 0.30
DropEdge[28] + GraphSAGE	50.8 ± 0.9	82.11 ± 0.30	87.64 ± 0.40
GraphSAINT[23]	51.1 ± 0.2	77.4 ± 0.1	87.5 ± 0.3
Ours + GraphSAGE	**52.1** $\pm$ 0.2	**93.2** $\pm$ 0.1	**92.4** $\pm$ 0.2
Ours + GraphSAINT	**51.2** $\pm$ 0.2	**83.4** $\pm$ 0.1	**89.4** $\pm$ 0.2



(a) GCN (b) Mixup (c) Ours

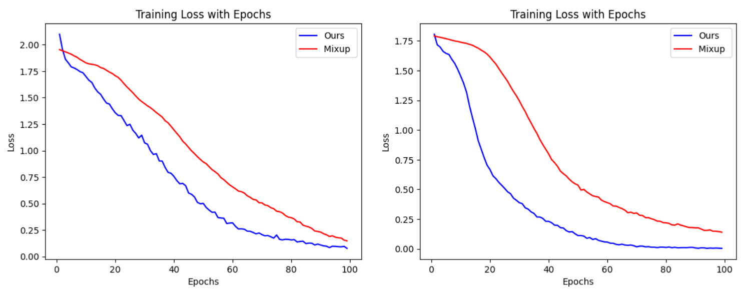
Fig. 2: A comparison of Mixup model and Ours Model, showcasing their performance differences in the final representation in the Cora dataset (visualized by t-SNE[42])

Table 3: Classification accuracy comparison for the Transductive setup. The experimental setup of Mixup is followed to enable a fair comparison.

Method	Citeseer	Cora	Pubmed
	Accuracy	Accuracy	Accuracy
GCN [20]	70.3	81.5	79.0
GAT [38]	72.5 ± 0.7	83.0 ± 0.7	79.0 ± 0.3
LGCN [39]	71.1 ± 0.5	82.2 ± 0.5	79.0 ± 0.2
GMNN [43]	73.6	83.7	81.9
Mixup + GCN [19]	78.7 ± 0.9	90.0 ± 0.7	87.9 ± 0.8
Ours + Mixup + GCN	**80.4** $\pm$ 1.4	**91.4** $\pm$ 0.1	**88.4** $\pm$ 1.1

5.3 Discussion

Our experiments demonstrate that our approach in node classification tasks on different datasets, including Cora, Citeseer, and PubMed for transductive settings, as well as Flickr, Amazon (computer), and Amazon (Photo) for induc-



(a) Cora (b) CiteSeer

Fig. 3: Comparison of loss functions between our proposed model and the Mixup model, illustrating the superior compression achieved by our model.

tive settings, has significant benefits. This indicates that leveraging structural features can greatly enhance the baseline of graph deep learning and its variants.

In addition, previous methods have neglected the use of structural features and focused only on real features associated with graphs. Therefore, by combining the extracted features with real features, the proposed method has the potential for further research to capture the benefits of structural features and leverage them in GNN (Graph Neural Networks) variants.

Furthermore, the interpreter of our proposed method for its significant achievements is the capability of a graph deep learning model to capture and learn these combined features. The modified features lead to precise node representations, thereby improving node classification performance.

Moreover, the simplicity of the architectures is the main advantage of our method. With this approach, we can achieve state-of-the-art performance, such as GCN [20], while maintaining low memory and computation costs.

5.4 Model Visualization

Fig. 2 presents the last node representations of the GCN, Mixup, and our proposed models on the Cora dataset. From the figures, we notice the clear node representations and discriminative between classes in the cora dataset, indicating that our proposed method is superior to other methods. Fig. 3 gives the visualization of the node representations of our proposed method and the Mixup. This figure provides some insight into the training process of both methods and provides a clear comparison, demonstrating how our approach leads to faster convergence.

6 Conclusion

In this paper, we have introduced a novel approach for augmenting node features in graphs to enhance performance in node classification tasks. Our method combines real features, such as a bag of words in citation networks, with structural features, like node degree and clustering coefficient, extracted at the node level. Additionally, we utilize deep learning models to further enrich these structural features, creating a comprehensive node representation. We have conducted extensive experiments on a variety of real-world datasets, including Cora, Citeseer, and Pubmed, to evaluate the effectiveness of our approach. The results demonstrated significant improvements over traditional methods that rely solely on real features, demonstrating the benefits of integrating both real and structural features. This approach not only addresses the issue of limited real features in some datasets but also highlights the potential of combining different types of node features to achieve more accurate and robust node classification.

Our approach represents a step forward in the field, as previous methods for augmenting graph features have largely focused on the use of real features and have neglected the potential benefits of incorporating structural features. By considering both real and structural features when augmenting the data, we are able to provide a more comprehensive representation of the nodes within the graph, leading to improved performance in node classification tasks. Future work will see an investigation of the potentials of our proposed approach for other graph-based tasks, such as graph generation and link prediction. We also plan to further explore the use of deep learning models to incorporate additional node-level features and evaluate the impact on the performance of the model. We believe that these efforts will help to facilitate further progress in the field and contribute to the development of more effective graph classification models.

References

1. W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *Advances in neural information processing systems*, vol. 30, 2017.
2. R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, “Graph convolutional neural networks for web-scale recommender systems,” in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 974–983.
3. S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, “Attention based spatial-temporal graph convolutional networks for traffic flow forecasting,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 922–929.
4. W. L. Hamilton, *Graph representation learning.* Morgan & Claypool Publishers, 2020.
5. E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, “Autoaugment: Learning augmentation strategies from data,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 113–123.
6. A. Atapour-Abarghouei, S. Bonner, and A. S. McGough, “A king’s ransom for encryption: Ransomware classification using augmented one-shot learning and bayesian approximation,” in *2019 IEEE International Conference on Big Data (Big Data).* IEEE, 2019, pp. 1601–1606.
7. S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy, “A survey of data augmentation approaches for nlp,” *arXiv preprint arXiv:2105.03075*, 2021.
8. Q. Xie, Z. Dai, E. Hovy, T. Luong, and Q. Le, “Unsupervised data augmentation for consistency training,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 6256–6268, 2020.
9. Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi, “Nodeaug: Semi-supervised node classification with data augmentation,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 207–217.

10. M. Adjeisah, X. Zhu, H. Xu, and T. A. Ayall, "Towards data augmentation in graph neural network: An overview and evaluation," *Computer Science Review*, vol. 47, p. 100527, 2023.
11. Z. Deng, Y. Dong, and J. Zhu, "Batch virtual adversarial training for graph convolutional networks," *arXiv preprint arXiv:1902.09192*, 2019.
12. F. Feng, X. He, J. Tang, and T.-S. Chua, "Graph adversarial training: Dynamically regularizing based on graph structure," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 6, pp. 2493–2504, 2019.
13. K. Kong, G. Li, M. Ding, Z. Wu, C. Zhu, B. Ghanem, G. Taylor, and T. Goldstein, "Robust optimization as data augmentation for large-scale graphs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 60–69.
14. S. Suresh, P. Li, C. Hao, and J. Neville, "Adversarial graph augmentation to improve graph contrastive learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 920–15 933, 2021.
15. X. Liu, D. Sun, and W. Wei, "A graph data augmentation strategy with entropy preservation," *arXiv preprint arXiv:2107.06048*, 2021.
16. R. Song, F. Giunchiglia, K. Zhao, and H. Xu, "Topological regularization for graph neural networks augmentation," *arXiv preprint arXiv:2104.02478*, 2021.
17. P. Chunaev, "Community detection in node-attributed social networks: a survey," *Computer Science Review*, vol. 37, p. 100286, 2020.
18. K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li, "Attention-based graph neural network for semi-supervised learning," *arXiv preprint arXiv:1803.03735*, 2018.
19. Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Mixup for node and graph classification," in *Proceedings of the Web Conference 2021*, 2021, pp. 3663–3674.
20. T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
21. K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.
22. J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.
23. H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," *arXiv preprint arXiv:1907.04931*, 2019.
24. J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.
25. M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in neural information processing systems*, vol. 29, 2016.
26. J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.
27. D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 3438–3445.
28. Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," *arXiv preprint arXiv:1907.10903*, 2019.
29. Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *Advances in neural information processing systems*, vol. 33, pp. 5812–5823, 2020.

30. J. Yoo, S. Shim, and U. Kang, "Model-agnostic augmentation for accurate graph classification," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1281–1291.
31. L. Yang, L. Zhang, and W. Yang, "Graph adversarial self-supervised learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 14 887–14 899, 2021.
32. M. Jing, Y. Zhu, T. Zang, J. Yu, and F. Tang, "Graph contrastive learning with adaptive augmentation for recommendation," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.* Springer, 2022, pp. 590–605.
33. Z. Zhong and D. Mottin, "Knowledge-augmented graph machine learning for drug discovery: From precision to interpretability," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 5841–5842.
34. J. Zhu, Q. Wang, C. Tao, H. Deng, L. Zhao, and H. Li, "Ast-gcn: Attribute-augmented spatiotemporal graph convolutional network for traffic forecasting," *IEEE Access*, vol. 9, pp. 35 973–35 983, 2021.
35. K. Ding, Z. Xu, H. Tong, and H. Liu, "Data augmentation for deep graph learning: A survey," *ACM SIGKDD Explorations Newsletter*, vol. 24, no. 2, pp. 61–77, 2022.
36. M. E. Abushofa, A. Atapour Abarghouei, M. Forshaw, and A. S. Mcgough, "Fegr: Feature enhanced graph representation method for graph classification," in *Proceedings of the International Conference on Advances in Social Networks Analysis and Mining*, 2023, pp. 371–378.
37. M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "Netsimile: A scalable approach to size-independent network similarity," *arXiv preprint arXiv:1209.2684*, 2012.
38. P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *stat*, vol. 1050, p. 20, 2017.
39. H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 1416–1424.
40. C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," *arXiv preprint arXiv:2007.08663*, 2020.
41. Y. Xue, Y. Liao, X. Chen, and J. Zhao, "Node augmentation methods for graph neural network based object classification," in *2021 2nd International Conference on Computing and Data Science (CDS).* IEEE, 2021, pp. 556–561.
42. L. Van Der Maaten, "Accelerating t-sne using tree-based algorithms," *The journal of machine learning research*, vol. 15, no. 1, pp. 3221–3245, 2014.
43. M. Qu, Y. Bengio, and J. Tang, "Gmnn: Graph markov neural networks," in *International conference on machine learning.* PMLR, 2019, pp. 5241–5250.