Mixup for Node and Graph Classification

Yiwei Wang  
National University of Singapore  
wangyw_seu@foxmail.com

Wei Wang  
National University of Singapore  
wangwei@comp.nus.edu.sg

Yuxuan Liang  
National University of Singapore  
yuxliang@outlook.com

Yujun Cai  
Nanyang Technological University  
yujun001@e.ntu.edu.sg

Bryan Hooi  
National University of Singapore  
bhooi@comp.nus.edu.sg

ABSTRACT
Mixup is an advanced data augmentation method for training neural network based image classifiers, which interpolates both features and labels of a pair of images to produce synthetic samples. However, devising the Mixup methods for graph learning is challenging due to the irregularity and connectivity of graph data. In this paper, we propose the Mixup methods for two fundamental tasks in graph learning: node and graph classification. To interpolate the irregular graph topology, we propose the two-stage Mixup framework, which uses each node’s neighbors’ representations before Mixup for graph convolutions. For graph classification, we interpolate complex and diverse graphs in the semantic space. Qualitatively, our Mixup methods enable GNNs to learn more discriminative features and reduce over-fitting. Quantitative results show that our method yields consistent gains in terms of test accuracy and F1-micro scores on standard datasets, for both node and graph classification. Overall, our method effectively regularizes popular graph neural networks for better generalization without increasing their time complexity.

CCS CONCEPTS
• Computing methodologies → Supervised learning by classification; Neural networks; Regularization.

KEYWORDS
data augmentation, node classification, graph classification

1 INTRODUCTION
Graph neural networks (GNNs) have achieved state-of-the-art performance on graph learning tasks, including node classification [27], [65], and graph classification [16], [60]. GNNs are capable of making predictions based on complex graph structures, thanks to their advanced representational power. However, the increased representational capacity comes with higher model complexity, which can induce over-fitting and weaken the generalization ability of GNNs. In this case, a trained GNN may capture random error or noise instead of the underlying data distribution [66], which is not what we expect.

To combat the over-fitting of neural networks, data augmentation has been demonstrated to be effective [38]. For node classification specifically, [40] proposes a data augmentation method named DropEdge. DropEdge follows the Vicinal Risk Minimization (VRM) principle [7] to define a vicinity around each node through randomly removing edges. Then, it draws additional virtual examples from the vicinity distribution to enlarge the support of the training distribution. In other words, it assumes that nodes have their class labels unchanged after the edge removals. However, whether this assumption holds is dataset-dependent and thus requires expert knowledge for usage. Furthermore, although DropEdge models the vicinity for the nodes sharing the same class, it does not describe the vicinity relation across samples of different classes.

Motivated by the above issues, we aim to design Mixup [67] methods for graph learning. Mixup is a recently proposed data augmentation method for image classification. Through linearly interpolating pixels of random image pairs and their training targets, Mixup generates synthetic images for training (see Fig. 1). Mixup does not need the ground-truth labels to be unchanged with the augmented features. In contrast, it incorporates the prior knowledge that interpolations of features should lead to interpolations of the associated targets [67]. Thus, Mixup extends the training distribution by constructing virtual training samples across all classes. From this vantage, Mixup acts as an effective regularization strategy for training image classifiers, which smoothens decision boundaries and improves the arrangements of hidden representations [52].

Although Mixup is effective in augmenting the image data, designing Mixup methods for graph learning is challenging. The challenges are rooted in the irregularity and connectivity of graph data. GNNs learn nodes’ representations via the ‘message passing’ mechanism, which aggregates the representations between each node and its neighbors at each layer [58]. As a result, the
representation of a node relies on the nodes and edges inside its receptive field [58], all of which act as its features. Thus, to mix a pair of nodes, we need to mix their receptive field subgraphs, which consist of nodes and topology. However, unlike image pixels, nodes are not placed on a regular grid but are instead unordered, which makes it difficult to pair the nodes in different (sub)graphs for Mixup. Besides, the interpolation is not well-defined for graph topology, which is necessary for Mixup. Furthermore, due to the connectivity between nodes, the use of Mixup on different node pairs can interfere with one another, which can cause conflicts and perturb the mixed features.

In our work, we propose Mixup methods for two fundamental tasks in graph learning: node and graph classification. For the former, we randomly pair nodes and aim to mix their receptive field subgraphs. We propose the two-branch Mixup graph convolution to interpolate the irregular graph topology. At each layer, we conduct the graph convolutions following the paired nodes’ topology separately in two branches and then interpolate the aggregated representations from the two branches before the next layer. In this way, the receptive field subgraphs of the paired nodes contribute to the final prediction together. To resolve the conflicts between the results of Mixup on different node pairs, we propose the two-stage Mixup framework. In the first stage, we perform a feed-forward as in the original GNNs to obtain nodes’ representations without Mixup. Then in the second stage, we conduct Mixup but use each node’s neighbors’ representations obtained from stage one to perform the graph convolutions. As a result, each node’s representations after Mixup do not interfere with the ‘message passing’ for other nodes. For graph classification, we mix the paired graphs in semantic space.

Our Mixup methods can be incorporated into popular GNNs thanks to their succinct design. We evaluate our methods on node classification using the Citeseer, Cora, Pubmed [33], Flickr [35], Yelp, and Amazon [65] datasets, and on graph classification using the standard chemical [15] and social [62] datasets. Qualitatively, our methods enable GNNs to learn more discriminative representations and effectively reduce over-fitting. We also observe quantitative improvements in terms of the test accuracy and F1-micro scores, which are higher than those achieved by the existing data augmentation strategies designed for specific domains [40]. Overall, our Mixup methods effectively regularize GNN models for better generalization without increasing their time complexity.

2 RELATED WORK

Node Classification Graph neural networks are the state-of-the-art solution for node classification [59], [71]. The first work that proposes the convolution operation on graph data is [5]. More recently, [27] made breakthrough advancements in the task of node classification. As a result, the model proposed in [27] is generally denoted as the vanilla GCN or GCN (Graph Convolutional Network). After [27], numerous methods are proposed for better performance on the graph learning [47], [58], [13], [57], [56], [1], [64], [39]. There are two main lines of research in this field.

The first line is to propose new GNN architectures to improve the model capacity [23], [49], [68]. For example, LGCN [18] ranks a node’s neighbors based on node features. It assembles a feature matrix that consists of its neighborhood and sorts this feature matrix along each column. [72] utilizes the positive pointwise mutual information (PPMI) matrix to capture nodes co-occurrence information through random walks sampled from a graph. [28] combines PageRank with GNNs to enable efficient information propagation. [51] alternatively drives local network embeddings to capture global structural information by maximizing local mutual information. [6] proposes a non-uniform graph convolutional strategy, which learns different convolutional kernel weights for different neighboring nodes according to their semantic meanings. [55] proposes the low-pass ‘message passing’ for robust graph neural networks, inhibiting the adversarial signals propagated through edges.

Another line is to propose new mini-batch training techniques for GNNs to enhance their scalability without the loss of effectiveness [22], [65]. GraphSAGE [22] performs uniform node sampling on the previous layer neighbors. It enforces a pre-defined budget
on the sample size, so as to bound the mini-batch computation complexity. [8] further restricts neighborhood size by requiring only two support nodes in the previous layer. Instead of sampling layers, ClusterGCN [10] and GraphSAINT [65] build mini-batches from subgraphs, so as to avoid the ‘neighbor explosion’ problem.

Our work is orthogonal to the above two lines in the sense that it does not alter the GNN architecture, or introduce a mini-batch technique. Instead, we propose a new method that can regularize GNN models to enhance their effectiveness by augmenting the graph data. DropEdge [40] is a pioneering work for data augmentation on graphs. DropEdge assumes the class labels of nodes are unchanged after the edge removals and thus requires domain knowledge for usage. In contrast, our mixup does not need the ground-truth labels to be unchanged given the augmented features and extends the training distribution by incorporating the prior knowledge that interpolations of features should lead to that of the associated targets [67]. We find that the favorable characteristics of model regularization provided by our Mixup methods lead to more accurate predictions.

**Graph Classification.** Early solutions to graph classification include graph kernels. The pioneering work [24] decomposes graphs into small subgraphs and computes kernel functions based on their pair-wise similarities. Subsequent work proposes various subgraphs, such as paths [3], and subtrees [44], [36]. More recently, many efforts have been made to design graph neural networks (GNNs) for graph classification [42], [32], [37], [19], [63], [69], [60]. Some work proposes the graph pooling methods to summarize the node representations [60], [53], [30], [26], [25], [17], [12]. The authors of [29] provide a unified view of local pooling and node attention mechanisms, and study the ability of pooling methods to generalize to larger and noisy graphs. In [9], the authors report that linear convolutional filters followed by nonlinear set functions achieve competitive performances. These work focuses on developing GNN architectures of higher complexity to improve their fitting capacity. In contrast, our framework is orthogonal to them in the sense that we propose a new data augmentation method that enhances a GNN model by interpolating the graphs from all classes to enlarge the support for training distribution.

**Data Augmentation.** Data Augmentation plays a central role in training neural networks. It operates on the input data and improves the performance significantly. For example, in image classification, DA strategies such as horizontal flips, random erasing [70], Hide-and-Seek [46], and Cutout [14] have been shown to improve performance. On MNIST, elastic distortions across scale, position, and orientation have been applied to achieve impressive results [41], [11], [45], [54]. Mixup [67], [52] is a particularly effective augmentation method for image classification, where the neural network is trained on convex combinations of images and their corresponding labels. We devise the Mixup methods for graph learning, for which we propose the two-branch graph convolution and the two-stage Mixup framework to handle the irregularity and connectivity of graph data. Different from existing data augmentation techniques designed for the graph data [40], [57], [58], which require the ground-truth labels to be unchanged after data augmentation, our method is dataset independent and do not require domain knowledge for usage. Our Mixup methods model the vicinity relations across nodes or graphs of different classes, which enables GNNs to learn better arrangements of representations.

### 3 METHODOLOGY

We interpolate a pair of nodes/graphs as well as their ground-truth labels to produce a novel and synthetic sample for training. To mix the graph topology, which is highly irregular, we propose the two-branch Mixup graph convolution (see Fig. 2(b)). Besides, to coordinate the Mixup of different nodes in the same mini-batch, we design a two-stage framework that utilizes the representations learned before Mixup (see Fig. 4). Last but not least, we interpolate the diverse and complicated graphs in the semantic embedding space for graph classification. We discuss the details of our Mixup methods for node and graph classification next.

#### 3.1 Background and Motivation

Mixup is first proposed in [67] for image classification. Consider a pair of samples \((x_i, y_i)\) and \((x_j, y_j)\), where \(x\) denotes the input feature, and \(y\) the one-hot class label. Mixup produces the synthetic sample as (see Fig. 1):

\[
\tilde{x} = \lambda x_i + (1 - \lambda) x_j, \quad (1)
\]

\[
\tilde{y} = \lambda y_i + (1 - \lambda) y_j, \quad (2)
\]

where \(\lambda \in [0, 1]\). In this way, Mixup extends the training distribution by incorporating the prior knowledge that interpolations of features should lead to interpolations of the associated labels [67]. Implementation of Mixup randomly picks one image and then pairs it up with another image drawn from the same mini-batch.

In our work, we focus on two fundamental tasks in graph learning: node and graph classification, the former of which aims to learn a mapping function that maps every node to a predicted class label, while the latter maps every graph to a label. We define a graph as \(G = (\mathcal{V}, \mathcal{E})\), where \(\mathcal{V}\) denotes the set of nodes, and \(\mathcal{E}\) is the set of edges. The input attribute vector of node \(i\) is \(x_i\), and the neighborhood of node \(i\) is \(N(i) = \{j \in \mathcal{V} | (i, j) \in \mathcal{E}\}\). Graph neural networks (GNNs) are the state-of-the-art solution for both...
node and graph classification [27], [60]. Typically, GNNs obtain the nodes’ representations $h^{(l)}_i$ at layer $l$ through the ‘message passing’ mechanism:

$$h^{(l)}_i = \text{AGGREGATE}\left(h^{(l-1)}_i, \{h^{(l-1)}_j \mid j \in \mathcal{N}(i)\}, W^{(l)}\right),$$

where $W^{(l)}$ denotes the trainable weights at layer $l$, and AGGREGATE is an aggregation function defined by the specific GNN model [60]. $h^{(0)}_i = x_i$ holds at the input layer. For node classification, GNNs learn the high-level semantic representations by stacking $L$ layers and minimizing the classification loss, e.g., cross-entropy [2], over the final-layer predictions, as presented in Fig. 4(a). For graph classification, GNNs summarize nodes’ representations into a single graph embedding through a ‘readout’ function:

$$h_G = \text{READOUT}\left(\{h^{(L)}_i \mid i \in \mathcal{V}\}\right),$$

where READOUT can be a simple permutation invariant function such as summation or a more sophisticated graph pooling function [63], [69].

Designing Mixup for graph learning is challenging due to the irregularity and connectivity of graph data. The classical Mixup in Eq. (1) is defined over the assumption that the input features $x$ follow the format of plain vectors, which does not fit the graph data. This motivates us to design the Mixup methods that offer effective regularization for graph learning and easy to implement alongside existing GNN models.

### 3.2 Mixup for Node Classification

We describe the ‘message passing’ of a GNN layer in Eq. (3) and Fig. (2) [58]. In principle, a GNN layer updates node $i$’s representations by aggregating the representations of itself and its neighbors. By stacking $L$ layers, GNNs make the final-layer prediction of node $i$ based on its $L$-hop neighborhood, which is known as node $i$’s
Algorithm 1 Two-Stage Mixup for Node Classification

Input: Graph $G = (\mathcal{V}, \mathcal{E})$ of a mini-batch, with node attributes $(x_i, i \in \mathcal{V})$, a GNN model with the aggregation function $\text{AGGREGATE}(\cdot)$, hyper-parameter $\alpha$ for the distribution of $\lambda$, the ground truth labels $(y_i, i \in \mathcal{V})$.

Output: The trained parameters of GNN: $\{W^{(l)}\}_{l}$.

1: for $i \leftarrow 1$ to $\#\mathcal{V}$ do
2: $h_i^{(0)} = x_i$
3: end for

4: for $l \leftarrow 1$ to $L - 1$ do
5: for $i \leftarrow 1$ to $\#\mathcal{V}$ do
6: $h_i^{(l)} = \text{AGGREGATE}\left(h_i^{(l-1)}, \left\{ h_{k}^{(l-1)} | k \in \mathcal{N}(i) \right\}, W^{(l)} \right)$
7: end for
8: end for
9: for $i \leftarrow 1$ to $\#\mathcal{V}$ do
10: Sample $j$ from $\mathcal{V}$
11: $\lambda \leftarrow \text{Beta}(\alpha, \alpha)$
12: $\tilde{x}_{ij} = \lambda x_i + (1 - \lambda) x_j$
13: $\tilde{y}_{ij} = \lambda y_i + (1 - \lambda) y_j$
14: $\tilde{h}_{ij}^{(0)} = \tilde{x}_{ij}$
15: for $l \leftarrow 1$ to $L$ do
16: $\tilde{h}_{ij}^{(l)} = \text{AGGREGATE}\left(\tilde{h}_{ij}^{(l-1)}, \left\{ h_{k}^{(l-1)} | k \in \mathcal{N}(i) \right\}, W^{(l)} \right)$
17: $\check{h}_{ij, j}^{(l)} = \text{AGGREGATE}\left(\tilde{h}_{ij}^{(l-1)}, \left\{ h_{k}^{(l-1)} | k \in \mathcal{N}(j) \right\}, W^{(l)} \right)$
18: $\check{h}_{ij}^{(l)} = \lambda \check{h}_{ij, j}^{(l)} + (1 - \lambda) \tilde{h}_{ij}^{(l)}$
19: end for
20: end for
21: Calculate classification loss $\mathcal{L}$ on $\{\check{h}_{ij}^{(l)}, \check{y}_{ij} | i \in \mathcal{V}\}$.
22: Back-propagation on $\{W^{(l)}\}$ for minimizing $\mathcal{L}$.

3.3 Mixup for Graph Classification

Graph neural networks utilize a READOUT function to summarize the node-level embeddings into a graph embedding. GNNs embed the complex and irregular graph structures into the embedding vectors of fixed dimension. We conduct Mixup for graph classification in the embedding space (see Fig. 5). In detail, given the graphs $G_1$ and $G_2$ with the embeddings $h_{G_1}$, $h_{G_2}$ and the labels $y_{G_1}$, $y_{G_2}$ respectively, we mix them as:

$$\check{h}_{G_{1}, G_{2}} = \lambda h_{G_1} + (1 - \lambda) h_{G_2},$$

$$\check{y}_{G_{1}, G_{2}} = \lambda y_{G_1} + (1 - \lambda) y_{G_2}.$$ (8) (9)

Finally, the interpolated graph-level embedding $\check{h}_{G_{1}, G_{2}}$ will be passed to a multi-layer perceptron followed by a softmax layer to produce the predicted distribution for the targeted classes.

3.4 Discussion

Mixup has been successfully applied to the tasks on image and text data, e.g., the classification of images [67] and sentences [20]. However, the graph data significantly differs from the above two kinds of data. First, in a graph, the nodes are connected, while images or sentences are isolated. Second, both the images and sentences are well-structured, the former of which has a two-dimensional grid and the latter of which is a one-dimensional sequence. However, graphs hold complicated and irregular structures. These differences pose serious challenges for Mixup. When mixing the input features, we must consider not only the node attributes but also the graph...
topology, for which the interpolation is not well-defined. Therefore, we propose the two-branch Mixup graph convolutions to handle this problem. In this way, we do not mix the topology directly, but mix the aggregated messages from different topology across GNN layers. In addition to this, due to the connectivity between different nodes and the ‘message passing’ mechanism, we need to resolve the conflicts between the Mixup of different nodes, as visualized in Fig. 3. This motivates us to propose the two-stage Mixup framework for node classification, where each node’s representation after Mixup does not interfere with the ‘message passing’ for other nodes. In this way, each node’s feature is not perturbed by the Mixup happening on its neighbors.

4 COMPLEXITY ANALYSIS

With Mixup, we train GNNs in the end-to-end style. First, since our Mixup method for graph classification does not induce extra computation, its complexity is the same as the original GNN model. Second, we analyze the time complexity of our two-stage Mixup framework for node classification. Given the dimension of node representations on layer $l$ being $d_l$, the time complexity of GCN is $O \left( \sum_{l=1}^{L} d_l + \sum_{l=1}^{L} d_{l-1} d_l \right)$ [27]. In our method, the time complexity of the first stage is $O \left( \sum_{l=1}^{L} d_l + \sum_{l=1}^{L} d_{l-1} d_l \right)$. In the second stage, we have $O \left( \sum_{l=1}^{L} d_l + \sum_{l=1}^{L} d_{l-1} d_l \right)$. Taking all the computation into consideration, we have the complexity of $O \left( \sum_{l=1}^{L} d_l + \sum_{l=1}^{L} d_{l-1} d_l \right)$, which is as same as the original GCN. For other kinds of GNNs, the analysis is similar to the above. Indeed, our first stage is as same as the original GNN without the final layer computation, while each layer in the second stage contributes the same complexity as that of the original GNN. Thus, our Mixup method improves the effectiveness of GNNs without increasing their time complexity.

5 EXPERIMENTS

In this section, we present the performance of various GNN models trained with our Mixup methods. For node classification, we report the experimental results under both the transductive and inductive settings. For graph classification, we report the test accuracy on both chemical and social graphs. After that, we adjust the volume of labeled data to evaluate the generalization of GNNs with and without our Mixup. In addition, we visualize the learned representations of GNNs trained with Mixup compared with the GNNs without Mixup. Last but not least, we conduct ablation studies to show the sensitivity of GNNs’ performance with respect to the hyper-parameters of our Mixup methods.

For node classification, we use the standard benchmark datasets: Cora, Citeseer, Cora, Pubmed [33], Flickr [35], Yelp, and Amazon [65] for evaluation. The first three are citation networks, where each node is a document and each edge is a citation link. In Flickr, each node represents one image. And an edge is built between two images if they share some common properties (e.g., same geographic location, same gallery, etc.). The Yelp dataset contains a social network, where an edge indicates that the connected users are friends. For the Amazon dataset, a node is a product on the Amazon website and an edge between two products is created if the products are bought by the same customer. Each of them contains an unweighted adjacency matrix and bag-of-words features. The statistics of these datasets are summarized in Table 1.

We use the standard benchmark datasets: D&D [15], NCI1, PROTEINS [4], COLLAB, IMDB-M, REDDIT-5K [62] for the evaluation of graph classification. The first three are chemical datasets, where the nodes have categorical input features. The last three are social datasets that do not have node attributes. We follow [60], [69] to use node degrees as attributes. The statistics of these datasets are summarized in Table 2.

For the hyper-parameters of the baseline methods, e.g., the number of hidden units, the optimizer, the learning rate, we set them as
We conduct the experiments for 100 trials with random weight initialization. We conduct the experiments under both transductive and inductive settings. We have access to the attributes of all nodes but only the labels of nodes in the training set for training. In the inductive setting, both the attributes and labels of the nodes in the validation/testing set are unavailable during training.

In the transductive node classification, we take the popular GNN models of GCN [27], GAT [50], LGCN [18], JKNet [61], GMNN [39], ResGCN [31], and the regularization method DropEdge [40] as the baseline methods for comparison. We split nodes in each graph into 60%, 20%, 20% for training, validation, and testing. We make 10 random splits and conduct the experiments for 100 trials with random weight initialization for each split. We vary the number of layers from 1 to 30 for each model and choose the best performing model with respect to the validation set. The results are reported in Table 3. We observe that our two-stage Mixup method outperforms all the baseline methods.

In the inductive settings, we use the datasets Flickr, Yelp, Amazon with the fixed partition [65] for evaluation. These datasets are too large to be handled well by the full-batch implementations of GCN architectures. Hence, we use more scalable GraphSAGE [22] and GraphSAINT [65] as the baselines for comparison. We vary the number of layers of each method from 1 to 30 for each model and choose the best performing model with respect to the validation set. We conduct the experiments for 100 trials with random weight initialization. The results are reported in Table 4. GraphSAGE-mean/LSTM/pool denotes that GraphSAGE uses mean, LSTM, and max-pooling as the aggregator respectively. And GraphSAINT-GCN/GAT/JKNet means that GraphSAINT takes GCN, GAT, and JKNet as the backbone respectively. We implement our two-stage Mixup method with GraphSAGE-mean and GraphSAINT-GCN to study whether Mixup can improve the performance of GCNs under the inductive setting. We observe that our two-stage Mixup improves the test F1-micro scores of GraphSAGE-mean by 3.0% on Flickr, 1.9% on Yelp, 2.0% on Amazon, and GraphSAINT-GCN by 2.5% on Flickr, 1.5% on Yelp, and 0.6% on Amazon respectively. As a result, our two-stage Mixup method enhances them to outperform the baseline methods.

Given the same GCN architecture, our Mixup method consistently produces larger improvements than DropEdge. DropEdge assumes the class labels of nodes kept unchanged after the edge removals, which is dataset-dependent. DropEdge does not model the vicinity relation across examples belonging to different classes [7]. In contrast, our mixup performs the data augmentation in a dataset independent manner and extends the training distribution by incorporating the prior knowledge that linear interpolations of features should lead to that of the associated targets, which has been demonstrated to induce better representation arrangements, and higher generalization ability [67]. Overall, the results above validate that our approach is effective in improving the performance of the popular GCN models under both transductive and inductive settings.

### 5.1 Node Classification

We conduct the experiments under both transductive and inductive settings for a comprehensive evaluation. In the transductive setting, we have access to the attributes of all nodes but only the labels of nodes in the training set for training. In the inductive setting, both the attributes and labels of the nodes in the validation/testing set are unavailable during training.

In the transductive node classification, we take the popular GNN models of GCN [27], GAT [50], LGCN [18], JKNet [61], GMNN [39], ResGCN [31], and the regularization method DropEdge [40] as the baseline methods for comparison. We split nodes in each graph into 60%, 20%, 20% for training, validation, and testing. We make 10 random splits and conduct the experiments for 100 trials with random weight initialization for each split. We vary the number of layers from 1 to 30 for each model and choose the best performing model with respect to the validation set. The results are reported in Table 3. We observe that our two-stage Mixup method improves the test accuracy of GCN by 2.1% on Citeseer, 1.9% on Cora, 1.7% on Pubmed, and improves JKNet by 2.6% on Citeseer, 1.5% on Cora, 1.7% on Pubmed respectively. As a result, our two-stage Mixup method enhances them to outperform all the baseline methods.

We study whether Mixup can improve the performance of GCNs under the inductive setting. We observe that our two-stage Mixup improves the test F1-micro scores of GraphSAGE-mean by 3.0% on Flickr, 1.9% on Yelp, 2.0% on Amazon, and GraphSAINT-GCN by 2.5% on Flickr, 1.5% on Yelp, and 0.6% on Amazon respectively. As a result, our two-stage Mixup method enhances them to outperform the baseline methods.

### 5.2 Graph Classification

For graph classification, we follow [16] and [60] to use the 10-fold cross-validation scheme for a fair comparison and evaluation. For each training fold, as suggested by [16], we conduct an inner hold-out technique with a 90%/10% training/validation split, i.e., we train fifty times on a training fold holding out a random fraction (10%) of the data to perform early stopping. These fifty separate trials are needed to smooth the effect of unfavorable random weight initialization on test performances. The final test fold score is obtained as the mean of these fifty runs.

### Table 3: Test Accuracy (%) of transductive node classification.

<table>
<thead>
<tr>
<th>Method</th>
<th>Citeseer</th>
<th>Cora</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN [27]</td>
<td>77.1±1.4</td>
<td>88.3±0.8</td>
<td>86.4±1.1</td>
</tr>
<tr>
<td>GAT [50]</td>
<td>76.3±0.8</td>
<td>87.6±0.5</td>
<td>85.7±0.7</td>
</tr>
<tr>
<td>JKNet [61]</td>
<td>78.1±0.9</td>
<td>89.1±1.2</td>
<td>86.9±1.3</td>
</tr>
<tr>
<td>LGCN [18]</td>
<td>77.5±1.1</td>
<td>89.0±1.2</td>
<td>86.5±0.6</td>
</tr>
<tr>
<td>GMNN [39]</td>
<td>77.4±1.5</td>
<td>88.7±0.8</td>
<td>86.7±1.0</td>
</tr>
<tr>
<td>ResGCN [31]</td>
<td>77.9±0.8</td>
<td>88.1±0.6</td>
<td>87.1±1.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Citeseer</th>
<th>Cora</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>DropEdge [40] + GCN</td>
<td>78.1±1.1</td>
<td>89.2±0.7</td>
<td>87.3±0.6</td>
</tr>
<tr>
<td>DropEdge [40] + JKNet</td>
<td>79.3±0.7</td>
<td>89.9±0.8</td>
<td>87.6±0.9</td>
</tr>
<tr>
<td>Mixup + GCN</td>
<td>78.7±0.9</td>
<td>90.0±0.7</td>
<td>87.9±0.8</td>
</tr>
<tr>
<td>Mixup + JKNet</td>
<td><strong>80.1±0.8</strong></td>
<td><strong>90.4±0.9</strong></td>
<td><strong>88.3±0.6</strong></td>
</tr>
</tbody>
</table>

### Table 4: Test F1-micro score (%) of inductive node classification.

<table>
<thead>
<tr>
<th>Method</th>
<th>Flickr</th>
<th>Yelp</th>
<th>Amazon</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphSAGE-mean [22]</td>
<td>50.1±1.1</td>
<td>63.4±0.6</td>
<td>75.8±0.2</td>
</tr>
<tr>
<td>GraphSAGE-LSTM [22]</td>
<td>50.3±1.3</td>
<td>63.2±0.8</td>
<td>75.7±0.1</td>
</tr>
<tr>
<td>GraphSAINT-GCN [65]</td>
<td>51.1±0.2</td>
<td>65.3±0.3</td>
<td>81.5±0.1</td>
</tr>
<tr>
<td>DropEdge [40] + GraphSAGE-mean</td>
<td>51.6±0.8</td>
<td>64.6±0.6</td>
<td>77.3±0.1</td>
</tr>
<tr>
<td>DropEdge [40] + GraphSAINT</td>
<td>51.7±0.6</td>
<td>65.8±0.7</td>
<td>81.8±0.2</td>
</tr>
<tr>
<td>Mixup + GraphSAGE-mean</td>
<td>52.4±0.4</td>
<td>66.3±0.4</td>
<td>82.0±0.1</td>
</tr>
</tbody>
</table>

**suggested by their authors.** For the hyper-parameters of our Mixup methods, we set α = 1 for the distribution of Mixup weights by default.
Table 5: Test Accuracy (%) of graph classification. We perform 10-fold cross-validation to evaluate model performance, and report the mean and standard derivations over 10 folds. We highlight best performances in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>D&amp;D</th>
<th>NCI1</th>
<th>PROTEINS</th>
<th>COLLAB</th>
<th>IMDB-M</th>
<th>REDDIT-5K</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPHLET [44]</td>
<td>72.1 ± 3.7</td>
<td>64.3 ± 2.2</td>
<td>70.1 ± 4.1</td>
<td>61.7 ± 2.2</td>
<td>42.6 ± 2.7</td>
<td>36.2 ± 1.8</td>
</tr>
<tr>
<td>WL [43]</td>
<td>73.2 ± 1.8</td>
<td>76.3 ± 1.9</td>
<td>72.3 ± 3.4</td>
<td>70.4 ± 1.8</td>
<td>45.4 ± 2.9</td>
<td>49.4 ± 2.1</td>
</tr>
<tr>
<td>GCN [27]</td>
<td>74.2 ± 3.1</td>
<td>76.8 ± 2.1</td>
<td>73.3 ± 3.6</td>
<td>74.3 ± 2.0</td>
<td>48.2 ± 3.1</td>
<td>53.7 ± 1.7</td>
</tr>
<tr>
<td>DGCNN [69]</td>
<td>76.7 ± 4.1</td>
<td>76.5 ± 1.9</td>
<td>72.9 ± 3.5</td>
<td>71.1 ± 1.7</td>
<td>45.6 ± 3.4</td>
<td>49.8 ± 1.9</td>
</tr>
<tr>
<td>DiffPool [63]</td>
<td>75.2 ± 3.8</td>
<td>76.8 ± 2.0</td>
<td>73.6 ± 3.6</td>
<td>68.9 ± 2.2</td>
<td>45.7 ± 3.4</td>
<td>53.6 ± 1.4</td>
</tr>
<tr>
<td>EigenPool [34]</td>
<td>75.9 ± 3.9</td>
<td>78.7 ± 1.9</td>
<td>74.1 ± 3.1</td>
<td>70.8 ± 1.9</td>
<td>47.2 ± 3.0</td>
<td>54.5 ± 1.7</td>
</tr>
<tr>
<td>GIN [60]</td>
<td>75.4 ± 2.6</td>
<td>79.7 ± 1.8</td>
<td>73.5 ± 3.8</td>
<td>75.5 ± 2.3</td>
<td>48.5 ± 3.3</td>
<td>56.1 ± 1.6</td>
</tr>
<tr>
<td>Mixup + GIN</td>
<td>75.4 ± 2.8</td>
<td>77.7 ± 2.1</td>
<td>74.1 ± 3.5</td>
<td>75.4 ± 2.2</td>
<td>48.8 ± 3.5</td>
<td>54.6 ± 1.8</td>
</tr>
<tr>
<td>Mixup + GIN</td>
<td>76.8 ± 2.9</td>
<td>81.0 ± 1.9</td>
<td>74.3 ± 3.5</td>
<td>77.0 ± 2.2</td>
<td>49.9 ± 3.2</td>
<td>57.8 ± 1.7</td>
</tr>
</tbody>
</table>

Table 6: Results of node classification averaged over the 20 random splits of the varied ratio \( r \) of training nodes, in terms of test accuracy (%). We highlight the best performance in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>Citeseer</th>
<th>Cora</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r = 30% )</td>
<td>( r = 40% )</td>
<td>( r = 50% )</td>
<td>( r = 30% )</td>
</tr>
<tr>
<td>GCN [27]</td>
<td>74.7 ± 2.5</td>
<td>75.2 ± 1.8</td>
<td>76.3 ± 1.6</td>
</tr>
<tr>
<td>Mixup + GCN</td>
<td>76.9 ± 2.1</td>
<td>77.1 ± 1.5</td>
<td>78.1 ± 1.3</td>
</tr>
<tr>
<td>JKNet [61]</td>
<td>75.6 ± 1.9</td>
<td>76.0 ± 1.4</td>
<td>77.1 ± 1.1</td>
</tr>
<tr>
<td>Mixup + JKNet</td>
<td>78.0 ± 1.7</td>
<td>78.3 ± 1.2</td>
<td>79.2 ± 1.0</td>
</tr>
</tbody>
</table>

Table 7: Results of graph classification averaged over the 20 random splits of the varied ratio \( r \) of labeled examples, in terms of test accuracy (%). We highlight the best performance in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>NCI1</th>
<th>PROTEINS</th>
<th>COLLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r = 60% )</td>
<td>( r = 70% )</td>
<td>( r = 80% )</td>
<td>( r = 60% )</td>
</tr>
<tr>
<td>GCN [27]</td>
<td>68.4 ± 2.4</td>
<td>70.1 ± 2.2</td>
<td>72.9 ± 2.2</td>
</tr>
<tr>
<td>Mixup + GCN</td>
<td>72.0 ± 2.3</td>
<td>72.9 ± 2.3</td>
<td>74.7 ± 2.0</td>
</tr>
<tr>
<td>GIN [60]</td>
<td>71.1 ± 2.2</td>
<td>73.0 ± 2.1</td>
<td>75.5 ± 2.0</td>
</tr>
<tr>
<td>Mixup + GIN</td>
<td>\textbf{74.7±2.0}</td>
<td>\textbf{75.4±2.0}</td>
<td>\textbf{77.1±2.1}</td>
</tr>
</tbody>
</table>

We use popular graph classification models as the baselines: GRAPHLET [44] and Weisfeiler-Lehman Kernel (WL) [43] are classical graph kernel methods, while GCN [27], DGCNN [69], DiffPool [63], EigenPool [34], and GIN [60] are the GNNs with state-of-the-art performance in graph classification. We report the average and standard deviation of test accuracy across the 10 folds within the cross-validation on the chemical and social datasets in Table 5 respectively. On the chemical datasets, we observe that our Mixup method improves the test accuracy of GCN by 1.6% on D&D, 1.2% on NCI1, 1.1% on PROTEINS respectively, and enhances GIN by 1.9% on D&D, 1.6% on NCI1, 1.1% on PROTEINS. On the social datasets, our Mixup method improves GCN by more than 1% and enhances GIN by at least 2% on the COLLAB, IMDB-M, and REDDIT-5K datasets in terms of test accuracy. Overall, Mixup achieves substantial improvements for GCN and GIN on both the chemical and social datasets. As a result, Mixup enhances GCN and GIN to outperform all the baseline methods.

Taking a closer look, we observe that the graph kernel methods, GRAPHLET and WL, generally present worse performance than the GNN methods. This demonstrates the stronger fitting capacity of the advanced neural network models. Mixup generally achieves higher improvements on GIN than that on GCN. The reason is that GIN is a more advanced GNN model proposed for graph classification than GCN. However, the increased learning power of GIN comes with higher risks of over-fitting. Our Mixup method effectively regularizes them by interpolating the graph representations to expand the training set, which reduces their over-fitting tendencies successfully.

5.3 Training Set Sizing

Over-fitting tends to be more severe when training on smaller datasets. By conducting experiments using a restricted fraction of the available training data, we show that our Mixup method has more significant improvements for smaller training sets.
First, we conduct the experiments on node classification. We randomly select $r \in \{30\%, 40\%, 50\%\}$ nodes from the whole set to form the training set, and randomly take half of the left nodes as the validation set, with the other half being the testing set. The results are reported in Table 6. Empirically, our Mixup method enhances the performance of GCN and JKNet for different sizes of the training set. In principle, with fewer labeled nodes, i.e., smaller $r$, our Mixup method gives larger accuracy improvements, because over-fitting is more serious when the training data is limited, where the regularization offered by our Mixup is essential to offer better generalization.

Next, we conduct the experiments on graph classification. We randomly select $r \in \{60\%, 70\%, 80\%\}$ graphs from the whole set to form the labeled data, with the left graphs being the test graphs, to form a split. For each labeled set, following [16], we conduct an inner holdout technique with a 90%/10% training/validation split. In other words, we train fifty times on a labeled set holding out a random fraction (10%) of the data to perform early stopping. We conduct the experiments on 20 random splits and report the mean and standard derivations over all the splits in Table 7. Empirically, Mixup enhances the performance of GCN and GIN for different sizes of the training set. In principle, with fewer labeled nodes, i.e., smaller $r$, our method gives larger accuracy improvements, which demonstrates the necessity of the regularization given by our Mixup especially when the labeled data is limited.

5.4 Visualization of Mixup

We study the effects of our Mixup method on GCN models during training. We depict the test loss at each training epoch in Fig. 7 on the Cora and Citeseer datasets. As we can see, for both GCNs with and without Mixup, their test loss decreases initially. However, our Mixup method significantly reduces the increase in test loss at later iterations and helps GCN models to converge to a lower test loss. This demonstrates that our Mixup method is able to effectively regularize GCNs to reduce over-fitting.

Fig. 6 presents the final-layer representations obtained by GCN and GCN with our Mixup on the Cora dataset. It is shown that the hidden layers supported by Mixup learn more discriminative representations, thanks to the regularization given by our Mixup.

Figure 6: The learned representations of the nodes in the Cora dataset (visualized by t-SNE [48]). Colors denote the ground-truth class labels. The node representations of same classes given by GCN with our Mixup are concentrated more than those given by GCN.

Figure 7: The training curves of GCN with and without our Mixup methods.

Table 8: Test Accuracy (%) of GCN on the Pubmed dataset and F1-micro score (%) of GraphSAINT-GCN on the Yelp dataset of node classification with and without our two-stage framework.

<table>
<thead>
<tr>
<th>Method</th>
<th>two stages</th>
<th>Pubmed</th>
<th>Yelp</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN [27]</td>
<td>-</td>
<td>86.4±1.1</td>
<td>0</td>
<td>65.3±0.3</td>
</tr>
<tr>
<td>Mixup + GCN</td>
<td>w/o</td>
<td>85.8±1.3</td>
<td>-0.6</td>
<td>64.2±0.6</td>
</tr>
<tr>
<td></td>
<td>w/</td>
<td>87.9±0.8</td>
<td>+1.5</td>
<td>66.3±0.4</td>
</tr>
</tbody>
</table>

These highly discriminative representations potentially help to produce better class predictions than less discriminative ones.

5.5 Ablation Study

We conduct a number of ablations to analyze our Mixup methods. First, we investigate the effects of our two-stage framework. Using our two-stage Mixup method, we can optionally not use the hidden representations of neighbors from the first stage during the ‘message passing’. This will enable each node’s hidden representations after Mixup to contribute to the ‘message passing’ for other nodes, which is likely to be the unwanted inference. In that case, we only need to conduct the second stage of our Mixup method without the first stage. Thus, we call this simpler version as the
Mixup method without our two stages. We compare the test accuracy of GCN trained with our Mixup with and without the two stages in Table 8. Mixup without our two stages does not provide improvements, and even causes a decrease in performance, while our two-stage Mixup method achieves consistent enhancements on the test accuracy. The reason is that, without our two stages, the Mixup happening on different nodes affects each other through the ‘message passing’ across GNN layers, which alters the learned representations of nodes and causes inconsistency between the mixed features and labels. As a result, the GNN models are not trained effectively to offer satisfactory performance. On the other hand, with our two-stage Mixup, we utilize each node’s neighbors’ representations without Mixup (given by the first stage) to process the ‘message passing’. In this way, our method prevents the Mixup for different nodes from affecting each other, and the GNN models are thus trained to effectively model the vicinity relation across nodes of different classes.

Last but not least, we evaluate how sensitive our Mixup method is to the selection of hyper-parameter value: $\alpha$, which controls the distribution from which we randomly select the Mixup weights. We present the experimental results on node and graph classification with different $\alpha$ in Table 9 and 10 respectively. As we can see, the performance of both GCN and GIN with Mixup is relatively smooth when parameters are within certain ranges, while extremely large or small values of $\alpha$ result in low performances, which should be avoided in practice. Thus, empirically, we choose $\alpha = 1$ as the default setting in our experiments and show that we achieve satisfactory performance with it.

### 6 Conclusion

Inspired by the success of Mixup, an advanced data augmentation method through sample interpolation for image classification, we explore to propose the Mixup methods for graph learning. In particular, we propose the two-branch Mixup graph convolution method and the two-stage Mixup framework to deal with the irregularity and connectivity of graph data, which is distinct to image data and poses serious challenges for Mixup. For the Mixup on graph classification, we interpolate the complex and diverse graphs in the semantic space. Empirical results show that our Mixup methods act as a dataset independent regularizer to offer better generalization for the popular GNN models on node and graph classification. Future work includes devising Mixup methods for other graph learning tasks beyond supervised learning, such as unsupervised, semi-supervised, and reinforcement learning. Extending our Mixup methods to feature-label extrapolation for more robust GNNs is worth exploration.

### Acknowledgments

This paper is supported by NUS ODPRT Grant R252-000-A81-133 and Singapore Ministry of Education Academic Research Fund Tier 3 under MOE official grant number MOE2017-T3-1-007.

### References


---

### Table 9: Test Accuracy (%) of node classification given by GCN and GCN with our Mixup method of different $\alpha$ values.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\alpha$</th>
<th>Citeseer</th>
<th>Cora</th>
<th>Flickr</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN [27]</td>
<td>-</td>
<td>77.1±1.4</td>
<td>88.3±0.8</td>
<td>51.1±0.2</td>
</tr>
<tr>
<td>Mixup + GCN</td>
<td>0.2</td>
<td>78.1±0.9</td>
<td>89.2±0.8</td>
<td>52.0±0.3</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>78.4±0.8</td>
<td>89.5±0.7</td>
<td>52.1±0.3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>78.7±0.9</td>
<td>90.0±0.7</td>
<td>52.4±0.4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>78.6±1.0</td>
<td>89.8±0.8</td>
<td>52.8±0.5</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>78.4±1.2</td>
<td>89.4±1.1</td>
<td>52.7±0.4</td>
</tr>
</tbody>
</table>

### Table 10: Test Accuracy (%) of graph classification given by GIN and GIN with our Mixup method of different $\alpha$ values.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\alpha$</th>
<th>D&amp;D</th>
<th>PROTEINS</th>
<th>IMDB-M</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIN [60]</td>
<td>-</td>
<td>75.4±2.6</td>
<td>73.5±3.8</td>
<td>48.5±3.3</td>
</tr>
<tr>
<td>Mixup + GIN</td>
<td>0.2</td>
<td>76.1±2.7</td>
<td>74.1±3.6</td>
<td>49.0±3.3</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>76.5±2.8</td>
<td>74.4±3.4</td>
<td>49.6±3.1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>76.8±2.7</td>
<td>74.3±3.5</td>
<td>49.9±3.2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>76.3±2.5</td>
<td>74.0±3.7</td>
<td>49.8±3.0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>76.0±2.6</td>
<td>73.7±3.8</td>
<td>49.5±3.1</td>
</tr>
</tbody>
</table>


