Mixup for Node and Graph Classification

Yiwei Wang
National University of Singapore
Singapore
wangyw_seu@foxmail.com

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

Yuxuan Liang
National University of Singapore
Singapore
yuxliang@outlook.com

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

Bryan Hooi
National University of Singapore
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-branch graph convolution to mix the receptive field subgraphs for the paired nodes. Mixup on different node pairs can interfere with the mixed features for each other due to the connectivity between nodes. To block this interference, 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

ACM Reference Format:
Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2021. Mixup for Node and Graph Classification. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3449796

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449796

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

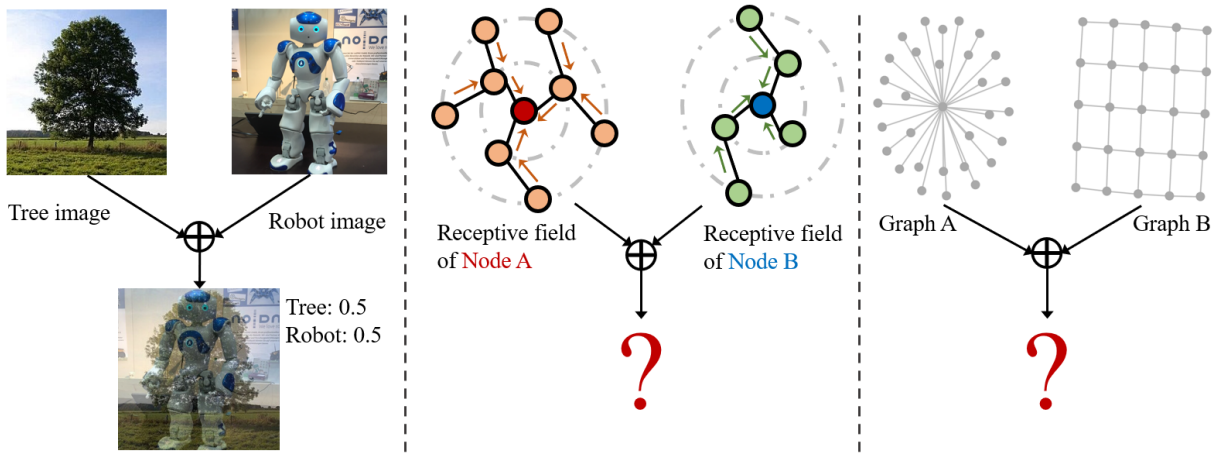


Figure 1: (left) For image classification, the existing Mixup generates synthetic images by interpolating both image pixels and labels. (middle) For node classification, to mix a pair of nodes A (red) and B (blue), we need to mix their receptive field subgraphs. (right) For graph classification, we need to mix the nodes and graph topology of a pair of graphs.

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.

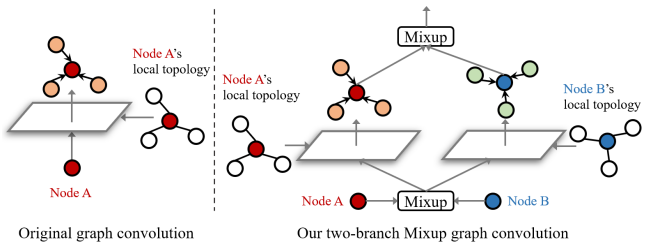


Figure 2: (left) Typically, a GNN layer updates a node's (red) representation by aggregating the representations of its neighbors and itself. (right) We propose the two-branch graph convolution to mix both nodes' attributes and their topology. For a pair of nodes (red and blue) to be mixed, we mix their attributes first. Then at each layer, we conduct the graph convolutions in two branches corresponding to the graph topology of the paired nodes (red and blue) separately, and mix the aggregated representations from two branches before the next layer.

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):

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j, \tag{1}$$

$$\tilde{y} = \lambda y_i + (1-\lambda)y_j, \tag{2}$$

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 $\mathbf{x}_i$, and the neighborhood of node i is $\mathcal{N}(i) = \{j \in \mathcal{V} | (i, j) \in \mathcal{E}\}$. Graph neural networks (GNNs) are the state-of-the-art solution for both

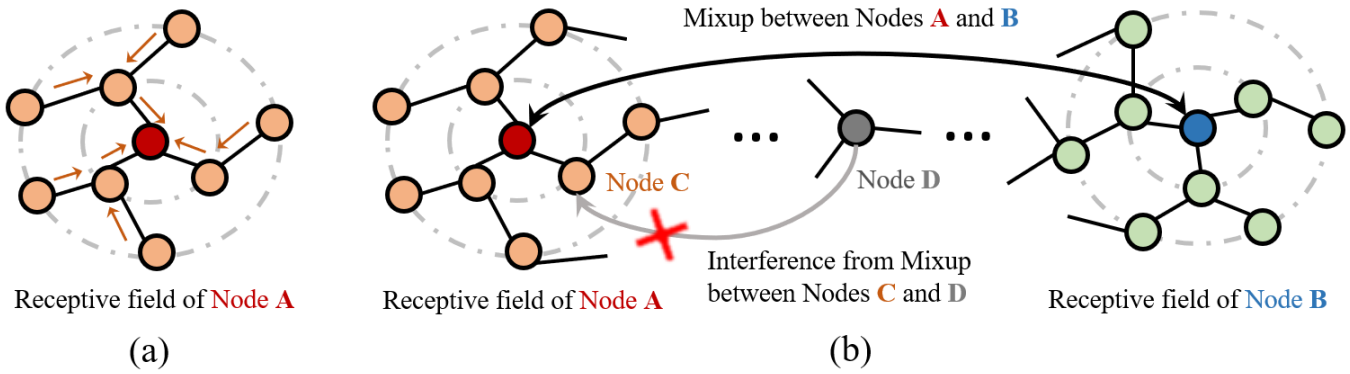


Figure 3: (a) A GNN model predicts the class of node A (red) by aggregating the nodes (orange) inside node A's receptive field. (b) To Mixup nodes A (red) and B (blue), we should mix the features inside A and B's receptive fields. However, if we conduct Mixup for Node C (orange) and Node D (grey) at the same time, the mixed input features from nodes A and B are perturbed by interference from Node D through Node C, which should be blocked.

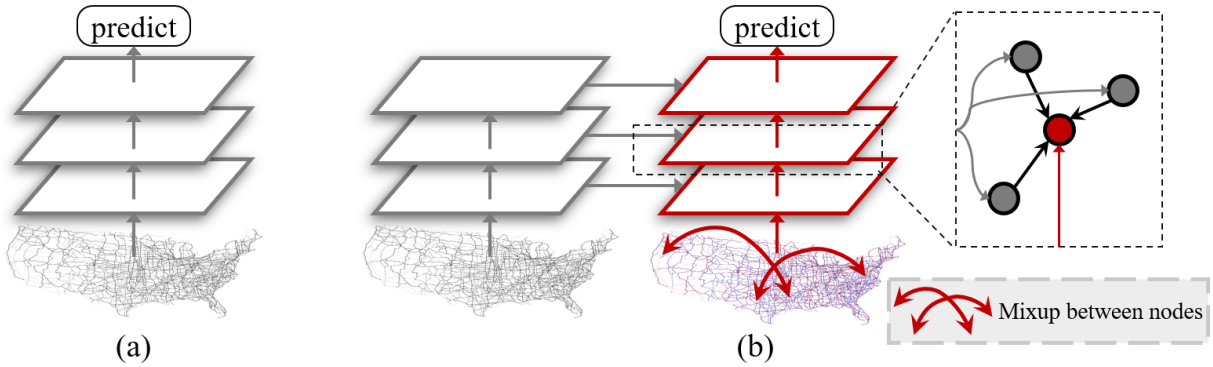


Figure 4: (a) Existing GNNs classify all nodes in a mini-batch graph at the same time. (b) We propose the two-stage Mixup method to resolve conflicts between the Mixup on different node pairs. At stage one, we perform the feed-forward as in existing GNNs without Mixup. Then at stage two, we randomly pair the nodes in the mini-batch graph and mix their input attributes. Next, we perform our two-branch Mixup graph convolutions (see Fig. 2) for the paired nodes at each layer, where we use each node's neighbors' representations obtained from stage one. This ensures that each node's representations after Mixup do not interfere with the 'message passing' for other nodes.

node and graph classification [27], [60]. Typically, GNNs obtain the nodes' representations $\mathbf{h}_i^{(l)}$ at layer l through the 'message passing' mechanism:

$$\mathbf{h}_i^{(l)} = \text{AGGREGATE}\left(\mathbf{h}_i^{(l-1)}, \left\{\mathbf{h}_j^{(l-1)} | j \in \mathcal{N}(i)\right\}, \mathbf{W}^{(l)}\right), \quad (3)$$

where $\mathbf{W}^{(l)}$ denotes the trainable weights at layer l, and AGGREGATE is an aggregation function defined by the specific GNN model [60]. $\mathbf{h}_i^{(0)} = \mathbf{x}_i$ 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:

$$\mathbf{h}_G = \text{READOUT}\left(\left\{\mathbf{h}_i^{(L)} | i \in \mathcal{V}\right\}\right), \quad (4)$$

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 $\{\mathbf{x}_i | i \in \mathcal{V}\}$, a GNN model with the aggregation function AGGREGATE$(\cdot)$, hyper-parameter α for the distribution of λ, the ground truth labels $\{\mathbf{y}_i | i \in \mathcal{V}\}$.

Output: The trained parameters of GNN: $\left\{\mathbf{W}^{(l)}\right\}_l$.

for $i \leftarrow 1$ to $\#\mathcal{V}$ **do**
  $\mathbf{h}_i^{(0)} \leftarrow \mathbf{x}_i$
end for
for $l \leftarrow 1$ to $L-1$ **do**
  for $i \leftarrow 1$ to $\#\mathcal{V}$ **do**
    $\mathbf{h}_i^{(l)} \leftarrow \text{AGGREGATE}\left(\mathbf{h}_i^{(l-1)}, \left\{\mathbf{h}_j^{(l-1)} | j \in \mathcal{N}(i)\right\}, \mathbf{W}^{(l)}\right)$
  end for
end for
for $i \leftarrow 1$ to $\#\mathcal{V}$ **do**
  Sample j from $\mathcal{V}$
  $\lambda \leftarrow \text{Beta}(\alpha, \alpha)$
  $\tilde{\mathbf{x}}_{ij} \leftarrow \lambda \mathbf{x}_i + (1-\lambda)\mathbf{x}_j$
  $\tilde{\mathbf{y}}_{ij} \leftarrow \lambda \mathbf{y}_i + (1-\lambda)\mathbf{y}_j$
  $\tilde{\mathbf{h}}_{ij}^{(0)} \leftarrow \tilde{\mathbf{x}}_{ij}$
  for $l \leftarrow 1$ to L **do**
    $\tilde{\mathbf{h}}_{ij,i}^{(l)} \leftarrow \text{AGGREGATE}\left(\tilde{\mathbf{h}}_{ij}^{(l-1)}, \left\{\mathbf{h}_k^{(l-1)} | k \in \mathcal{N}(i)\right\}, \mathbf{W}^{(l)}\right)$
    $\tilde{\mathbf{h}}_{ij,j}^{(l)} \leftarrow \text{AGGREGATE}\left(\tilde{\mathbf{h}}_{ij}^{(l-1)}, \left\{\mathbf{h}_k^{(l-1)} | k \in \mathcal{N}(j)\right\}, \mathbf{W}^{(l)}\right)$
    $\tilde{\mathbf{h}}_{ij}^{(l)} \leftarrow \lambda \tilde{\mathbf{h}}_{ij,i}^{(l)} + (1-\lambda)\tilde{\mathbf{h}}_{ij,j}^{(l)}$
  end for
end for
Calculate classification loss $\mathcal{L}$ on $\left\{\tilde{\mathbf{h}}_{ij}^{(L)}, \tilde{\mathbf{y}}_{ij} | i \in \mathcal{V}\right\}$.
Back-propagation on $\left\{\mathbf{W}^{(l)}\right\}_l$ for minimizing $\mathcal{L}$.

receptive field [58]. In other words, to interpolate the paired nodes i and j, we need to mix their receptive field subgraphs. To achieve this, we propose the two-branch Mixup graph convolution as shown in Fig. 2, where we mix the node attributes of nodes i and j before the input layer:

$$\tilde{\mathbf{x}}_{ij} = \lambda \mathbf{x}_i + (1-\lambda)\mathbf{x}_j, \tag{5}$$

Next, we conduct the graph convolutions based on nodes i and j's topologies separately at each layer:

$$\begin{aligned}\tilde{\mathbf{h}}_{ij,i}^{(l)} &= \text{AGGREGATE}\left(\tilde{\mathbf{h}}_{ij}^{(l-1)}, \left\{\mathbf{h}_k^{(l-1)} | k \in \mathcal{N}(i)\right\}, \mathbf{W}^{(l)}\right), \\ \tilde{\mathbf{h}}_{ij,j}^{(l)} &= \text{AGGREGATE}\left(\tilde{\mathbf{h}}_{ij}^{(l-1)}, \left\{\mathbf{h}_k^{(l-1)} | k \in \mathcal{N}(j)\right\}, \mathbf{W}^{(l)}\right),\end{aligned} \tag{6}$$

and mix the aggregated features from the two topologies together before the next layer:

$$\tilde{\mathbf{h}}_{ij}^{(l)} = \lambda \tilde{\mathbf{h}}_{ij,i}^{(l)} + (1-\lambda)\tilde{\mathbf{h}}_{ij,j}^{(l)}, \tag{7}$$

where $\tilde{\mathbf{h}}_{ij}^{(0)} = \tilde{\mathbf{x}}_{ij}$ holds.

Here, how to compute the node i's neighbors' representations $\left\{\mathbf{h}_k^{(l-1)} | k \in \mathcal{N}(i)\right\}$ in Eq. (6) is an issue. If we follow the same implementation as the classical Mixup [67], i.e., we randomly pair the nodes in the mini-batch to conduct Mixup and conduct the feed-forward for all nodes synchronously, we can only have $\mathbf{h}_k^{(l-1)} = \tilde{\mathbf{h}}_{km}^{(l-1)}$, where m is the node paired with node i's neigbhor k. This causes conflicts, because node m interferes with the 'message passing' for node i (see Eq. (6)) through the Mixup between m and k, but node m is likely to be outside the receptive field of node i. An example is shown in Fig. 3. Specifically, when mixing nodes i and j, node i's neighbors can be mixed with a node outside node i and j's receptive fields, which adds unwanted external noise to perturb the input features: here, by 'external noise', we mean any perturbation to the input features that do not arise from the receptive fields of nodes i and j.

To address the above problem, i.e., we propose the two-stage Mixup framework as shown in Fig. 4(b). In the first stage, we conduct the feed-forward to GNNs for the mini-batch graph to obtain the nodes' hidden representations without Mixup. Next, in the second stage, we randomly pair the nodes in the mini-batch to conduct the Mixup of node attributes. Then, we conduct our two-branch Mixup graph convolutions for the paired nodes as shown in Fig. 2(b). Note that at each layer in the second stage, we use the neighbors' representations without Mixup, which are obtained from the first stage, to conduct the graph convolutions (see Eq. (6)). In this way, each node's representations after Mixup do not interfere with the 'message passing' for other nodes. With our two-stage framework, we effectively prevent the input features from being perturbed by the nodes outside the receptive fields. We sample the Mixup weight λ from the distribution Beta(α, α) with a hyperparameter α [21]. Our Mixup method for node classification is summarized in Alg. 1.

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 $\mathbf{h}_{G_1}$, $\mathbf{h}_{G_2}$ and the labels $\mathbf{y}_{G_1}$, $\mathbf{y}_{G_2}$ respectively, we mix them as:

$$\tilde{\mathbf{h}}_{G_1G_2} = \lambda \mathbf{h}_{G_1} + (1-\lambda)\mathbf{h}_{G_2}, \tag{8}$$

$$\tilde{\mathbf{y}}_{G_1G_2} = \lambda \mathbf{y}_{G_1} + (1-\lambda)\mathbf{y}_{G_2}. \tag{9}$$

Finally, the interpolated graph-level embedding $\tilde{\mathbf{h}}_{G_1G_2}$ will be passed to a multi-layer perception 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

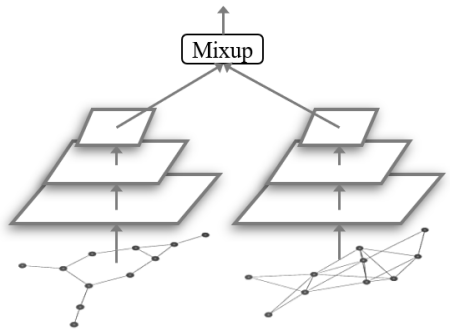


Figure 5: We mix the graph-level representations for the Mixup on graph classification, which encodes both nodes' attributes and graph topology.

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 $O\left(\#\mathcal{E}\sum_{l=1}^{L} d_l + \#\mathcal{V}\sum_{l=1}^{L} d_{l-1}d_l\right)$ [27]. In our method, the time complexity of the first stage is $O\left(\#\mathcal{E}\sum_{l=1}^{L-1} d_l + \#\mathcal{V}\sum_{l=1}^{L-1} d_{l-1}d_l\right)$. In the second stage, we have $O\left(\#\mathcal{E}\sum_{l=1}^{L} d_l + \#\mathcal{V}\sum_{l=1}^{L} d_{l-1}d_l\right)$. Taking all the computation into consideration, we have the complexity of $O\left(\#\mathcal{E}\sum_{l=1}^{L} d_l + \#\mathcal{V}\sum_{l=1}^{L} d_{l-1}d_l\right)$, 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.

Table 1: Statistics of the datasets for node classification. 'm' stands for multi-label classification, while 's' for single-label.

Dataset	#Nodes	#Edges	#Classes	#Attributes
Cora	2,708	5,429	7 (s)	1,433
Citeseer	3,327	4,732	6 (s)	3,703
Pubmed	19,717	44,338	3 (s)	500
Flickr	89,250	899,756	7 (s)	500
Yelp	716,847	6,977,410	100 (m)	300
Amazon	1,598,960	132,169,734	107 (m)	200

Table 2: Statistics of the datasets for graph classification. #Nodes and #Edges denotes the average number of nodes and edges per graph respectively.

Dataset	#Graphs	#Nodes	#Edges	#Classes
D&D	1,178	284.32	715.66	2
NCI1	4,110	29.87	32.30	2
PROTEINS	1,113	39.06	72.82	2
COLLAB	5,000	74.49	2457.78	3
IMDB-M	1,500	13.00	65.94	3
REDDIT-5K	4,999	508.52	594.87	5

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

Table 3: Test Accuracy (%) of transductive node classification. We conduct 100 trials with random weight initialization. The mean and standard deviations are reported.

Method	Citeseer	Cora	Pubmed
GCN [27]	77.1±1.4	88.3±0.8	86.4±1.1
GAT [50]	76.3±0.8	87.6±0.5	85.7±0.7
JKNet [61]	78.1±0.9	89.1±1.2	86.9±1.3
LGCN [18]	77.5±1.1	89.0±1.2	86.5±0.6
GMNN [39]	77.4±1.5	88.7±0.8	86.7±1.0
ResGCN [31]	77.9±0.8	88.1±0.6	87.1±1.2
DropEdge [40] + GCN	78.1±1.1	89.2±0.7	87.3±0.6
DropEdge [40] + JKNet	79.3±0.7	89.9±0.8	87.6±0.9
Mixup + GCN	78.7±0.9	90.0±0.7	87.9±0.8
Mixup + JKNet	**80.1±0.8**	**90.4±0.9**	**88.3±0.6**

Table 4: Test F1-micro score (%) of inductive node classification. We report mean and standard deviations of 100 trials with random weight initialization. We implement DropEdge and our Mixup method with GraphSAGE-mean and GraphSAINT-GCN.

Method	Flickr	Yelp	Amazon
GraphSAGE-mean [22]	50.1±1.1	63.4±0.6	75.8±0.2
GraphSAGE-LSTM [22]	50.3±1.3	63.2±0.8	75.7±0.1
GraphSAGE-pool [22]	50.0±0.8	63.1±0.5	75.5±0.2
DropEdge [40] + GraphSAGE	50.8±0.9	64.1±0.8	76.4±0.1
Mixup + GraphSAGE	**51.6±0.8**	**64.6±0.6**	**77.3±0.1**
GraphSAINT-GCN [65]	51.1±0.2	65.3±0.3	81.5±0.1
GraphSAINT-GAT [65]	50.5±0.1	65.1±0.2	81.5±0.1
GraphSAINT-JKNet [65]	51.3±0.5	65.3±0.4	81.6±0.1
DropEdge [40] + GraphSAINT	51.7±0.6	65.8±0.7	81.8±0.2
Mixup + GraphSAINT	**52.4±0.4**	**66.3±0.4**	**82.0±0.1**

suggested by their authors. For the hyper-parameters of our Mixup methods, we set $\alpha = 1$ for the distribution of Mixup weights by default.

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 number 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, 1.6% on Pubmed respectively. As a result, our two-stage Mixup method enhances GCN and JKNet to outperform 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.

5.2 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 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.

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

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.

Method	Citeseer			Cora			Pubmed		
	$r = 30\%$	$r = 40\%$	$r = 50\%$	$r = 30\%$	$r = 40\%$	$r = 50\%$	$r = 30\%$	$r = 40\%$	$r = 50\%$
GCN [27]	74.7 ± 2.5	75.2 ± 1.8	76.3 ± 1.6	86.3 ± 1.9	86.8 ± 1.4	87.5 ± 1.0	85.1 ± 2.3	85.4 ± 1.4	85.8 ± 1.2
Mixup + GCN	76.9 ± 2.1	77.1 ± 1.5	78.1 ± 1.3	88.5 ± 1.4	88.9 ± 1.0	89.4 ± 0.9	87.0 ± 1.6	87.2 ± 1.1	87.5 ± 1.0
JKNet [61]	75.6 ± 1.9	76.0 ± 1.4	77.1 ± 1.1	86.7 ± 2.1	87.4 ± 1.5	88.2 ± 1.3	85.3 ± 2.2	85.9 ± 1.6	86.4 ± 1.4
Mixup + JKNet	**78.0 ± 1.7**	**78.3 ± 1.2**	**79.2 ± 1.0**	**88.6 ± 2.0**	**89.1 ± 1.5**	**89.7 ± 1.2**	**87.2 ± 1.9**	**87.5 ± 1.3**	**87.9 ± 0.9**

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.

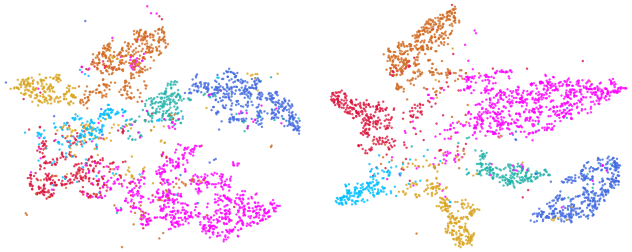
Method	NCI1			PROTEINS			COLLAB		
	$r = 60\%$	$r = 70\%$	$r = 80\%$	$r = 60\%$	$r = 70\%$	$r = 80\%$	$r = 60\%$	$r = 70\%$	$r = 80\%$
GCN [27]	68.4 ± 2.4	70.1 ± 2.2	72.9 ± 2.2	65.8 ± 4.0	67.7 ± 4.0	70.1 ± 3.9	67.7 ± 2.3	69.1 ± 2.4	71.2 ± 2.2
Mixup + GCN	72.0 ± 2.3	72.9 ± 2.3	74.7 ± 2.0	**69.2 ± 3.9**	**70.1 ± 3.8**	71.4 ± 3.8	70.8 ± 2.4	71.6 ± 2.3	73.0 ± 2.2
GIN [60]	71.1 ± 2.2	73.0 ± 2.1	75.5 ± 2.0	65.2 ± 4.1	67.1 ± 3.8	69.8 ± 3.7	68.0 ± 2.5	69.6 ± 2.5	71.8 ± 2.3
Mixup + GIN	**74.7 ± 2.0**	**75.4 ± 2.0**	**77.1 ± 2.1**	69.1 ± 3.9	69.8 ± 3.9	**71.5 ± 3.7**	**70.9 ± 2.5**	**71.8 ± 2.4**	**73.9 ± 2.2**

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.



(a) GCN (b) GCN + 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.

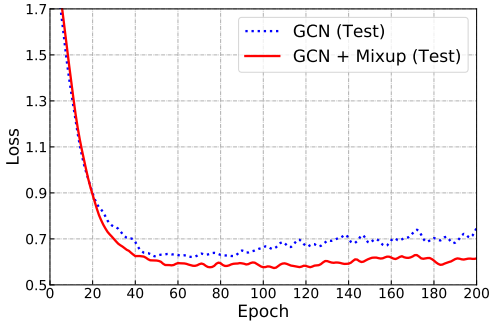
First, we conduct the experiments on node classification. We randomly select $r \in \{30\%, 40\%, 50\%\}$ nodes from the whole set to form the training set, and randomly take half of the left nodes as the validation set, with the other half being the testing set. The results are reported in Table 6. 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.

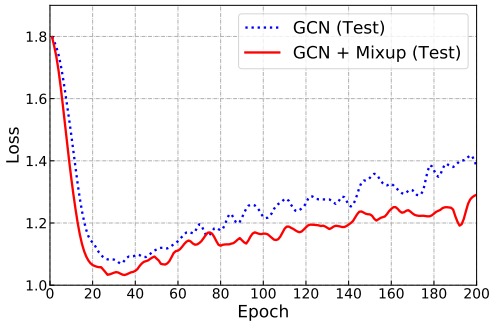
5.4 Visualization of Mixup

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.



(a) Cora



(b) Citeseer

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.

Method	two stages	Pubmed	Δ	Yelp	Δ
GCN [27]	-	86.4±1.1	0	65.3±0.3	0
Mixup + GCN	w/o	85.8±1.3	-0.6	64.2±0.6	-1.1
	w/	87.9±0.8	+1.5	66.3±0.4	+1.0

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

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

Method	α	Citeseer	Cora	Flickr
GCN [27]	-	77.1±1.4	88.3±0.8	51.1±0.2
Mixup + GCN	0.2	78.1±0.9	89.2±0.8	52.0±0.3
	0.5	78.4±0.8	89.5±0.7	52.1±0.3
	1	**78.7±0.9**	**90.0±0.7**	52.4±0.4
	2	78.6±1.0	89.8±0.8	**52.8±0.5**
	5	78.4±1.2	89.4±1.1	52.7±0.4

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

Method	α	D&D	PROTEINS	IMDB-M
GIN [60]	-	75.4±2.6	73.5±3.8	48.5±3.3
Mixup + GIN	0.2	76.1±2.7	74.1±3.6	49.0±3.3
	0.5	76.5±2.8	**74.4±3.4**	49.6±3.1
	1	**76.8±2.7**	74.3±3.5	**49.9±3.2**
	2	76.3±2.5	74.0±3.7	49.8±3.0
	5	76.0±2.6	73.7±3.8	49.5±3.1

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 affect each other, and the GNN models are thus trained to effectively model the vicinity relation across nodes of different classes [7].

Last but not least, we evaluate how sensitive our Mixup method is to the selection of hyper-parameter value: α, which controls the distribution from which we randomly select the Mixup weights. We present the experimental results on node and graph classification with different α 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 α 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 MOEs official grant number MOE2017-T3-1-007.

REFERENCES

[1] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. Midas: microcluster-based detector of anomalies in edge streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3242–3249.
[2] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.
[3] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE, 8–pp.
[4] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.
[5] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).
[6] Yujun Cai, Liuhao Ge, Jun Liu, Jianfei Cai, Tat-Jen Cham, Junsong Yuan, and Nadia Magnenat Thalmann. 2019. Exploiting spatial-temporal relationships for 3d pose estimation via graph convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 2272–2281.
[7] Olivier Chapelle, Jason Weston, Léon Bottou, and Vladimir Vapnik. 2001. Vicinal risk minimization. In *Advances in neural information processing systems*. 416–422.
[8] Jianfei Chen, Jun Zhu, and Le Song. 2017. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568* (2017).
[9] Ting Chen, Song Bian, and Yizhou Sun. 2019. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv preprint arXiv:1905.04579* (2019).
[10] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 257–266.
[11] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 3642–3649.
[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375* (2016).
[13] Ailin Deng and Bryan Hooi. 2021. Graph Neural Network-Based Anomaly Detection in Multivariate Time Series. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
[14] Terrance DeVries and Graham W Taylor. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. *arXiv preprint arXiv:1708.04552* (2017).
[15] Paul D Dobson and Andrew J Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology* 330, 4 (2003), 771–783.

[16] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. 2019. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893* (2019).

[17] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. In *Proceedings of the 36th International Conference on Machine Learning*.

[18] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1416–1424.

[19] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212* (2017).

[20] Hongyu Guo, Yongyi Mao, and Richong Zhang. 2019. Augmenting data with mixup for sentence classification: An empirical study. *arXiv preprint arXiv:1905.08941* (2019).

[21] Arjun K Gupta and Saralees Nadarajah. 2004. *Handbook of beta distribution and its applications*. CRC press.

[22] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.

[23] Lu Haonan, Seth H Huang, Tian Ye, and Guo Xiuyan. 2019. Graph Star Net for Generalized Multi-Task Learning. *arXiv preprint arXiv:1906.12330* (2019).

[24] David Haussler. 1999. *Convolution kernels on discrete structures*. Technical Report. Technical report, Department of Computer Science, University of California ....

[25] Jingjia Huang, Zhangheng Li, Nannan Li, Shan Liu, and Ge Li. 2019. Attpool: Towards hierarchical feature representation in graph convolutional networks via attention mechanism. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6480–6489.

[26] Amir Hosein Khasahmadi, Kaveh Hassani, Parsa Moradi, Leo Lee, and Quaid Morris. 2020. Memory-based graph networks. *arXiv preprint arXiv:2002.09518* (2020).

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

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

[29] Boris Knyazev, Graham W Taylor, and Mohamed R Amer. 2019. Understanding attention and generalization in graph neural networks. *arXiv preprint arXiv:1905.02850* (2019).

[30] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. In *36th International Conference on Machine Learning, ICML 2019*. International Machine Learning Society (IMLS), 6661–6670.

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

[32] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[33] Ben London and Lise Getoor. 2014. Collective Classification of Network Data. *Data Classification: Algorithms and Applications* 399 (2014).

[34] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 723–731.

[35] Julian McAuley and Jure Leskovec. 2012. Image labeling on a network: using social-network metadata for image classification. In *European conference on computer vision*. Springer, 828–841.

[36] Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. 2016. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning* 102, 2 (2016), 209–245.

[37] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*. 2014–2023.

[38] Luis Perez and Jason Wang. 2017. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621* (2017).

[39] Meng Qu, Yoshua Bengio, and Jian Tang. 2019. Gmnn: Graph markov neural networks. *arXiv preprint arXiv:1905.06214* (2019).

[40] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*.

[41] Ikuro Sato, Hiroki Nishimura, and Kensuke Yokoi. 2015. Apac: Augmented pattern classification with neural networks. *arXiv preprint arXiv:1505.03229* (2015).

[42] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.

[43] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, 9 (2011).

[44] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*. 488–495.

[45] Patrice Y Simard, David Steinkraus, John C Platt, et al. 2003. Best practices for convolutional neural networks applied to visual document analysis.. In *Icdar*, Vol. 3.

[46] Krishna Kumar Singh, Hao Yu, Aron Sarmasi, Gautam Pradeep, and Yong Jae Lee. 2018. Hide-and-Seek: A Data Augmentation Technique for Weakly-Supervised Localization and Beyond. *arXiv preprint arXiv:1811.02545* (2018).

[47] Zekun Tong, Yuxuan Liang, Changsheng Sun, David S Rosenblum, and Andrew Lim. 2020. Directed graph convolutional network. *arXiv preprint arXiv:2004.13970* (2020).

[48] Laurens Van Der Maaten. 2014. Accelerating t-SNE using tree-based algorithms. *The Journal of Machine Learning Research* 15, 1 (2014), 3221–3245.

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

[50] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[51] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep graph infomax. *arXiv preprint arXiv:1809.10341* (2018).

[52] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. 2019. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*. PMLR, 6438–6447.

[53] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2015. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391* (2015).

[54] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. In *International conference on machine learning*. 1058–1066.

[55] Yiwei Wang, Shenghua Liu, Minji Yoon, Hemank Lamba, Wei Wang, Christos Faloutsos, and Bryan Hooi. 2020. Provably Robust Node Classification via Low-Pass Message Passing. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 621–630.

[56] Yiwei Wang, Wei Wang, Yujun Ca, Bryan Hooi, and Beng Chin Ooi. 2020. Detecting Implementation Bugs in Graph Convolutional Network based Node Classifiers. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 313–324.

[57] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2020. GraphCrop: Subgraph Cropping for Graph Classification. *arXiv preprint arXiv:2009.10564* (2020).

[58] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Juncheng Liu, and Bryan Hooi. 2020. NodeAug: Semi-Supervised Node Classification with Data Augmentation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 207–217.

[59] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A Comprehensive Survey on Graph Neural Networks. *arXiv preprint arXiv:1901.00596* (2019).

[60] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[61] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536* (2018).

[62] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1365–1374.

[63] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*. 4800–4810.

[64] Yuxuan Liang Yujun Cai Yiwei Wang, Wei Wang and Bryan Hooi. 2020. Progressive Supervision for Node Classification. In *Proceedings of ECML-PKDD*, Vol. 2020. 2020.

[65] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

[66] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2016. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530* (2016).

[67] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017).

[68] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. 2018. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294* (2018).

[69] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[70] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2017. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896* (2017).
[71] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).
[72] Chenyi Zhuang and Qiang Ma. 2018. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*. 499–508.