Graph Transplant: Node Saliency-Guided Graph Mixup with Local Structure Preservation

Joonhyung Park[*1]**, Hajin Shim**[*1]**, Eunho Yang** [1,2]

[1]Korea Advanced Institute of Science and Technology (KAIST)
[2]AITRICS
{deepjoon, shimazing, eunhoy}@kaist.ac.kr

Abstract

Graph-structured datasets usually have irregular graph sizes and connectivities, rendering the use of recent data augmentation techniques, such as Mixup, difficult. To tackle this challenge, we present the first Mixup-like graph augmentation method at the graph-level called *Graph Transplant*, which mixes irregular graphs in data space. To be well defined on various scales of the graph, our method identifies the substructure as a mix unit that can preserve the local information. Since the mixup-based methods without special consideration of the context are prone to generate noisy samples, our method explicitly employs the node saliency information to select meaningful subgraphs and adaptively determine the labels. We extensively validate our method with diverse GNN architectures on multiple graph classification benchmark datasets from a wide range of graph domains of different sizes. Experimental results show the consistent superiority of our method over other basic data augmentation baselines. We also demonstrate that *Graph Transplant* enhances the performance in terms of robustness and model calibration.

1 Introduction

Graph classification plays a critical role in a variety of fields from chemistry to social data analysis. In recent years, attempts to use graph neural networks (GNNs) to solve graph classification tasks have been in the spotlight (Gilmer et al. 2017; Simonovsky and Komodakis 2017; Ying et al. 2018; Errica et al. 2019; Meng and Zhang 2019) , as deep models successfully learn from unstructured data in various domains such as computer vision (CV) (He et al. 2016) or natural language processing (NLP) (Vaswani et al. 2017). However, the challenge is that, like most deep models, in order for GNNs to successfully generalize even for unseen graph data, we are often required so much data that it is not practical.

To train these data-hungry neural networks more cost-efficiently, data augmentation techniques are commonly employed. Existing graph augmentation methods largely rely on injecting structural noise such as edge manipulation (Rong et al. 2019; Zhou, Shen, and Xuan 2020) or node deletion (You et al. 2020) without considering the semantics of each graph data so that they can mislead the network

due to their context-agnostic nature. Furthermore, most of the proposed graph augmentation strategies focus on node classification tasks (Verma et al. 2019b; Rong et al. 2019), which is difficult to apply directly to graph classification.

As another attractive option, we consider Mixup (Zhang et al. 2017) which has become one of the standard data augmentations with its excellent performance in image classification (Yun et al. 2019), sentence classification (Guo, Mao, and Zhang 2019a), model robustness (Hendrycks et al. 2019), etc. However, since existing Mixup variants (Verma et al. 2019a; Yun et al. 2019; Guo, Mao, and Zhang 2019b; Kim et al. 2021) are somewhat specific to image data or require data instances to lie in the same dimensional space, they are inappropriate for the case of graph data where there is usually no node correspondence, and even the number of nodes may differ across graph instances.

In this paper, we propose *Graph Transplant*, a novel Mixup-based data augmentation for graph classification tasks that can be generally used for multiple-domain datasets regardless of graph geometries. To make sense of mixing graph instances with no node correspondence and the different number of nodes, *Graph Transplant* selects a subgraph from each of the destination and source instances and transplants the subgraph of the source onto the destination whose subgraph is removed. However, transplanting a subgraph from one instance to another may not properly align the final product with the corresponding interpolated label if the original charge of the subgraph is not reflected in the label. To address this issue, we propose to utilize the node saliency for selecting subgraphs and assigning labels. Specifically for our purpose, we define the node saliency as ℓ_2 norm of the gradient of classification loss with respect to the output of the graph convolutional layer. It can be regarded as a measure of node's contribution when the model classifies the graph. Furthermore, the label for the mixed graph is determined as the ratio of the saliency between the source and destination subgraphs. Instead of naïvely using the ratio of the number of nodes in subgraphs, saliency-based label mixing describes the mixed graphs properly by weighting nodes according to their importance. Another non-trivial challenge in mixing graphs is to determine the connectivity between subgraphs obtained from two different instances. To this end, we suggest the two variants: 1) random connection under the constraint on the original node degree and 2) edge

prediction based on node features.

In summary, our contribution is threefold:

- We propose *Graph Transplant*, the first input-level Mixup-based graph augmentation, that can mix two dissimilar-structured graphs by replacing the destination subgraph with the source subgraph while preserving the local structure.

- In contrast to random augmentations which risk producing noisy samples, we present a novel strategy to generate properly mixed graphs with adaptively assigned labels that adheres to the corresponding labels by explicitly leveraging the node saliency.

- Through extensive experiments, we show that our *Graph Transplant* is an effective algorithm that can bring overall improvement across the multiple fields of graphs analysis for diverse architectures in terms of the classification performance, robustness, and model calibration.

2 Related Works

Data augmentation for graph-structured data Most of the data augmentation methods for graph-structured data are biased to node classification tasks. In addition, many of them largely rely on edge perturbation. GraphSage (Hamilton, Ying, and Leskovec 2017) samples a different subset of neighbors uniformly for every iteration. DropEdge (Rong et al. 2019) randomly removes a fraction of edges before each training epoch. AdaEdge (Chen et al. 2020) iteratively adds or removes edges between nodes that are predicted to have the same labels with high confidence. Zhao et al. (2020) proposed to manipulate edges with a separate edge prediction module in training and inference stages. Another approach, GraphMix (Verma et al. 2019b), applies manifold Mixup (Verma et al. 2019a) to graph node classification which jointly trains a Fully Connected Network (FCN) and a GNN with shared parameters.

Data augmentation for graph classification tasks, which is our main concern, is relatively less explored. You et al. (2020) created new graphs with four general data augmentations: node dropping, edge perturbation, attribute masking, and subgraph extraction for contrastive learning. These baselines do succeed in enlarging the dataset size but take the risk of creating misleading data since the characteristics of each graph are not considered. In this sense, a filtering method for augmented data is proposed (Zhou, Shen, and Xuan 2020) that compares the prediction value of augmented and validation data after augmenting with edge perturbation to decide if the augmented data are taken or not. Wang et al. (2021) also proposed Mixup-like augmentations for graph and node classification, respectively. For graph classification, they suggested to mixup final graph representation vectors after a readout layer, which can be viewed as a simple extension of manifold Mixup to graph data.

Mixup approach The recently proposed Mixup (Zhang et al. 2017) uses a pair of input samples to generate augmented data via convex combination. This approach demonstrates its effectiveness in that mixed data allows the model to learn a more general decision boundary and improve

its generalization performance for unseen data. Moreover, Manifold Mixup (Verma et al. 2019a) expands the Mixup to the latent feature space. Especially, a novel extension of Mixup called CutMix (Yun et al. 2019) generates a virtual image by cutting an image patch from one image and pasting it to the other, and mixes their labels according to the area each occupies. This method effectively compensates the information loss issue in Cutout (Devries and Taylor 2017), while taking advantage of the regional dropout enforcing the model to attend on the entire object. While Mixup and its variants lead the data augmentation field not only in CV but also in NLP (Guo, Mao, and Zhang 2019a) and ASR (Medennikov et al. 2018), the mixing data scheme is not yet transplanted to the domain of graphs. Since the nodes of the two different graphs do not match and the connectivity of the newly mixed graph must be determined, mixing two graphs is non-trivial and becomes a difficult challenge.

Saliency Interpreting the model predictions has emerged as an important topic in the field of deep learning. Specifically, many researchers exploit the gradient of output with respect to input features to compute the saliency map of the image. In the domain of graphs, there are also attempts to identify the salient sub-structures or node features that primarily affect the model prediction (Ying et al. 2019; Neil et al. 2018). The simple approach using the gradient of the loss with respect to the node features may somewhat be less accurate than other more advanced interpretability methods such as (Ying et al. 2019), but it does not require any network modifications as (Selvaraju et al. 2017) nor the training of additional networks (Zhou et al. 2016).

3 Preliminary

Graph classification Let us define $G = (V, E)$ to be an undirected graph where V is the set of nodes, E is the set of edges or paired nodes. $X \in \mathbb{R}^{|V| \times d}$ is the feature matrix whose i-th entry is a d-dimensional node feature vector for node i. $\mathcal{N}(v)$ is the set of nodes $\{u \in V | \{u, v\} \in E\}$ that are connected to v by edges. For the graph C-class classification task, each data point (G, y) consists of a graph G and its corresponding label $y \in \{1, \ldots, C\}$. The goal is to train a GNN model which takes the graph G and predicts the probability for each class.

Graph neural networks for graph classification In this paper, we consider different members of message-passing GNN framework with the following four differentiable components: 1) message function m_l, 2) permutation invariant message aggregation function ϕ_l such as element-wise mean, sum or max, 3) node update function h_l, and 4) readout function γ that permutation-invariantly combines node embeddings together, at the end to obtain graph representation (Kipf and Welling 2016; Veličković et al. 2017; Xu et al. 2018). Let $x_v^{(l)}$ be the latent vector of node v at layer l. To simplify notations with the recursive definition of GNNs, we use $x_v^{(0)}$ to denote the input node feature. At each GNN layer, node features are updated as $x_v^{(l+1)} = h_l(x_v^{(l)}, \phi_i(\{m_l(x_v^{(l)}, x_u^{(l)}, e_{v,u}) | u \in \mathcal{N}(v)\}))$ and after the

last L-th layer, all node feature vectors are combined by the readout function: $x_G = \gamma(x_0^{(L)}, ..., x_{|V|}^{(L)})$. By passing x_G to a classifier f, we obtain the prediction $\hat{y} = f(x_G)$ for graph G, where $\hat{y}_c = P(y = c|G)$. As a representative example, a layer of Graph Convolutional Network (GCN) is defined as $x_v^{(l+1)} = \Theta \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{e_{v,u}}{\sqrt{\hat{d}_u \hat{d}_v}} x_u^{(l)}$, where $\hat{d}_v = 1 + \sum_{u \in \mathcal{N}(v)} e_{v,u}$ with the edge weight $e_{v,u}$.

Mixup and its variants Mixup is an interpolation-based regularization technique that generates virtual data by combining pairs of examples and their labels for classification problems. Let $x \in \mathcal{X}$ be an input vector and $y \in \mathcal{Y}$ be an one-hot encoded C-class label from data distribution $\mathcal{D}$. Mixup based augmentation methods optimize the following loss $\mathcal{L}$ given a mixing ratio distribution q:

$$\min_{\theta} . \mathbb{E}_{(x_i,y_i),(x_j,y_j) \sim \mathcal{D}, \lambda \sim q}[\mathcal{L}(g(x_i, x_j; \lambda), \ell(y_i, y_j; \lambda))],$$

where g and ℓ are the data and label Mixup functions, respectively. In case of the vanilla Mixup (Zhang et al. 2017), these functions are defined as $g(x_i, x_j; \lambda) = (1-\lambda)x_i + \lambda x_j$ and $\ell(y_i, y_j) = (1-\lambda)y_i + \lambda y_j$ with the mixing parameter λ sampled from beta distribution $\text{Beta}(\alpha, \alpha)$. Another example, CutMix (Yun et al. 2019), defines mixing function as $g(x_i, x_j; B) = (1 - \mathbb{1}_B) \odot x_i + \mathbb{1}_B \odot x_j$, where $\odot$ denotes the element-wise Hadamard product for a binary mask $\mathbb{1}_B$ with $B = [a, a + w] \times [b, b + h]$ and $\lambda = \frac{wh}{WH}$, where W and H are the width and height of images.

4   Graph Transplant

Now we introduce our effective augmentation strategy for graph classification, *Graph Transplant*, which generates semantically meaningful mixed graphs considering the context of graph instances. First, we define the node saliency for GNNs that encodes the importance of each node in classifying the graph property (Section 4.1). Armed with this notion of node saliency, we devise two main components of *Graph Transplant*: graph mixing that preserves the local graph structure (Section 4.2) and the label mixing that adaptively assigns the supervisions based on saliency of each subgraph to appropriately describe the mixed graphs (Section 4.3). The details are described below.

4.1   Node Saliency for GNNs

Graph Transplant does not rely on a specific method of calculating the importance of a subgraph and can be combined with a variety of approaches. However, unlike the general purpose of explainability, which finds important regions given a trained model, our goal is to dynamically find and mix up subgraphs while GNN is being trained. Hence, in this paper, we preferentially consider an approach that can simply and quickly find the importance of a node using node features and gradients. Among them, in particular, motivated by the works only using gradient values for image classification (Simonyan, Vedaldi, and Zisserman 2013; Shrikumar, Greenside, and Kundaje 2017; Kim, Choo, and Song 2020), we devise the following definition of the node saliency.

Given the latent node feature matrix $X^{(l)}$ obtained from the GNN l-th layer, we first compute the gradient of classification loss $\mathcal{L}(G, y = c) = -\log \hat{y}_c$ with respect to $X^{(l)}$, $\frac{\partial \mathcal{L}}{\partial X^{(l)}} \in \mathbb{R}^{|V| \times d}$. It is worth noting that the gradient can be obtained almost without cost as it is a byproduct of training step on the original graphs. Then, the saliency s_v of node v is derived as the ℓ_2 norm of the gradient corresponding to node v: $s_v = \left\| \left[\frac{\partial \mathcal{L}}{\partial X^{(l)}} \right]_{v,:} \right\|_2$. The choice of computing gradients at the l-th layer is to capture the high-level spatial information in l-hop neighborhood for each node (Bengio, Courville, and Vincent 2013; Selvaraju et al. 2017). The final node saliency vector $\mathcal{S}_G = [s_1, ..., s_{|V|}]^T$ is therefore obtained without modifying the network architecture.

4.2   Graph Mixing

Salient subgraph selection For every iteration, we consider the batch of N graphs $(G_1, ..., G_N)$. We pair two graphs G_{π_i} and G_i where π is a shuffled index of $\text{range}(N) = [1, ..., N]$. From now on, we drop the index i for brevity, and G_π and G will be referred to as a source and a destination graph, respectively.

To construct the subgraph $\bar{G}_\pi = (\bar{V}_\pi, \bar{E}_\pi)$ from the source G_π, we first choose salient $R\%$ of nodes $\hat{V}_\pi$, to locate the important part containing the class semantics. The node saliency represents the importance of the l-hop subgraph by its definition. To avoid repeatedly selecting the same nodes for the whole training phase, we stochastically sample $\hat{V}_\pi$, from the top-$2R\%$ salient nodes uniformly at random. Given the set of selected anchors $\hat{V}_\pi$, we extract the *partial K-hop* subgraphs $\bar{G}_\pi$ by randomly taking $p\%$ of adjacent neighbors for each moving step. p is sampled from $\text{Beta}(\alpha, \alpha) * 100$ for each training iteration (See Algorithm 1). The main reason for choosing the partial subgraph is for the case of when the graph is dense, the K-hop subgraph may coincide with the whole graph G. For each iteration, K is stochastically sampled from a discrete space $\mathcal{K}$.

On the destination side, the subgraph is removed from the original graph to 'transplant' the source-side subgraph $\bar{G}_\pi$. The nodes $\bar{V}$ obtained by Algorithm 1 are removed from destination graph G. In contrast to the source-side, anchor nodes $\hat{V}$ are *randomly* selected at $R\%$ in the destination graph G for the diversity of augmented graphs. The place where these nodes fall off is where the subgraph from the source, $\bar{G}_\pi$, will be transplanted. The remaining destination subgraph induced by $V \setminus \bar{V}$ is denoted by $\bar{G} = (V \setminus \bar{V}, \bar{E})$.

Algorithm 1: `Partial K-hop`

Input: number of hops K, original graph G, subnodes $\hat{V} \subset V$, sampling ratio p
Output: $\bar{V}$
Initialize: $\bar{V} \leftarrow \hat{V}, H \leftarrow \hat{V},$
  1: **for** $k = 1$ to K **do**
  2:      $H \leftarrow$ randomly chosen $p \%$ of nodes in $\mathcal{N}(H)$
  3:      $\bar{V} \leftarrow \bar{V} \cup H$
  4: **end for**

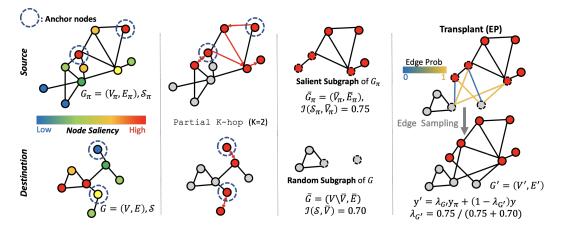
Figure 1: Overview of *Graph Transplant*. First, we compute the node saliency $\mathcal{S}_\pi, \mathcal{S}$ of source/destination graphs G_π, G. Then, we extract partial K-hop subgraphs from each graph anchored by salient nodes $\hat{V}_\pi$ and random nodes $\hat{V}$, respectively. Mixed label y' is adaptively determined based on $\mathcal{S}_\pi, \mathcal{S}$. Finally, we transplant the source-side subgraph $\bar{G}_\pi$ to the destination subgraph.

Algorithm 2: Graph Transplant$_{EP}$

Input: graph pair (G_π, G), node saliency $\mathcal{S}_{G_\pi}$, number of hops K, central nodes sampling ratio R, edge prediction module ψ

Output: mixed graph $G' = (V', E')$

1: $\hat{V}_\pi \leftarrow R\%$ salient nodes from G_π given $\mathcal{S}_{G_\pi}$ // Source
2: $\bar{V}_\pi \leftarrow$ Partial K-hop $(K, G_\pi, \hat{V}_\pi)$
3: $\bar{G}_\pi \leftarrow$ Induced Subgraph $(G_\pi, \bar{V}_\pi)$
4: $\hat{V} \leftarrow R\%$ random nodes from G // Destination
5: $\bar{V} \leftarrow$ Partial K-hop $(K, G, \hat{V})$
6: $\bar{G} \leftarrow$ Induced Subgraph $(G, V \setminus \bar{V})$
   /* Merge */
7: $V' \leftarrow \bar{V}_\pi \cup (V \setminus \bar{V})$
8: $E' \leftarrow \bar{E}_\pi \cup \bar{E}$
   /* Edge prediction */
9: **for** $u \in \bar{V}_\pi, v \in V \setminus \bar{V}$ **do**
10:   **if** $\deg_{G_\pi}(u) > \deg_{\bar{G}_\pi}(u), \deg_G(v) > \deg_{\bar{G}}(v)$ **then**
11:     $\bar{e}_{u,v} \sim$ Bernoulli $\left(\frac{1}{2}\psi(u,v) + \frac{1}{2}\psi(v,u)\right)$
12:   **else**
13:     $\bar{e}_{u,v} \leftarrow 0$
14:   **end if**
15: **end for**
16: $E' \leftarrow E' \cup \{\{u,v\} | u \in \bar{V}_\pi, v \in V \setminus \bar{V}, \bar{e}_{u,v} = 1\}$

Transplant After extracting the subgraphs $\bar{G}_\pi$ and $\bar{G}$, we transplant $\bar{G}_\pi$ into $\bar{G}$ by adding new edges between $\bar{V}_\pi$ and $V \setminus \bar{V}$. Formally, we construct the combined graph $G' = (V', E')$ with $V' = \bar{V}_\pi \cup (V \setminus \bar{V})$ and need to determine the connectivity between $\bar{V}_\pi$ and $(V \setminus \bar{V})$. We only consider the nodes losing edges by subgraph extraction to preserve the local information and the inner-structure intact. Toward this, let us define $\deg_G(v)$ as the degree of node v in graph G. Then, the set of candidate nodes of the source and the destination graphs where the factitious edges can be attached is $U_\pi = \{v \in \bar{V}_\pi | \deg_{G_\pi}(v) > \deg_{\bar{G}_\pi}(v)\}$, $U = \{v \in$

$(V \setminus \bar{V}) | \deg_G(v) > \deg_{\bar{G}}(v)\}$ respectively, and the total amount of change in degree is $D_\pi = \sum_{v \in U_\pi}(\deg_{G_\pi}(v) - \deg_{\bar{G}_\pi}(v))$, $D = \sum_{v \in U}(\deg_G(v) - \deg_{\bar{G}}(v))$.

To decide the connectivity between $v_\pi \in U_\pi$ and $v \in U$, we propose the following two different approaches. The first one is to sample $\lfloor(D + D_\pi)/2\rfloor$ node pairs $(u, v) \in U_\pi \times U$ uniformly at random. This straightforward and efficient method, which we call *degree preservation* abbreviated as DP, preserves the expected degree of the nodes in the original graphs. This makes the distribution of the generated graphs not too away from the original one while producing diverse samples. The other is to introduce a differentiable edge prediction module in the training phase to take into account the features of node pairs for connectivity. We name it *edge prediction* or EP. The edge prediction module ψ takes a concatenated pair of node features (r_{v_π}, r_v) and predicts the probability of edge existence $\psi(r_{v_\pi}, r_v) = P(\{v_\pi, v\} \in E')$. As a slight abuse of notation, we use $\psi(u, v)$ to denote the edge predictor consuming representations of node pair (u, v). For undirected graphs, we consider the symmetric form and take the average $\hat{e}_{u,v} = \frac{1}{2}(\psi(u,v) + \psi(v,u))$. For inputs of ψ, we utilize latent node features $x_{v_\pi}^{(l)}$ and $x_v^{(l)}$ after l-th GNN layer to leverage the local information together. We sample the new edge weight $\bar{e}_{v_\pi,v}$ from Bernoulli$(\hat{e}_{v_\pi,v})$. To make the sampled edge weight $\bar{e}$ differentiable, we resort to the straight-through Gumbel-Softmax estimator (Jang, Gu, and Poole 2016) in the training phase. Finally, a new edge set E' is set to $\bar{E}_\pi \cup \bar{E} \cup \{\{v_\pi, v\} | v_\pi \in \bar{V}_\pi, v \in V, \bar{e}_{v_\pi,v} = 1\}$. Edge prediction module ψ is trained by both supervised learning with the original graphs and in an end-to-end manner with virtual graphs (see Lines 4-5, 13 of Algorithm 3). In Supplementary, we provide the accuracy learning curve of the edge prediction module for node pairs of original graphs.

4.3 Adaptive Label Mixing

In the previous section, our method generates virtual graph data that captures the graph semantics while preserving the

---

Algorithm 3: Train with `Graph Transplant`$_{EP}$

Input: dataset $\mathcal{D} = \{(G, y)\}$, GNN model θ, central nodes sampling ratio R, edge prediction module ψ, discrete K-hop space $\mathcal{K}$, learning rate η

1: **while** not satisfying early stopping criteria **do**
2:     Sample batch $\mathcal{B} = \{(G_i, y_i)\}_i^{|\mathcal{B}|} \sim \mathcal{D}$
3:     Backpropagate to get $\nabla_\theta \mathcal{L} = \nabla_\theta \frac{1}{|\mathcal{B}|} \sum_i \mathcal{L}(G_i, y_i)$
      and node saliency $\{\mathcal{S}_{G_i}\}_i^{|\mathcal{B}|}$ given θ
      /* Update the edge prediction module */
4:     Sample pairs of node and their connection
      $\mathcal{E} = \{((x_u^{(l)}, x_v^{(l)}), \mathbb{1}_{\{u,v\} \in \cup_i E_i})\}$
5:     Update the edge predictor ψ with $\mathcal{E}$
      /* Do *Graph Transplant* */
6:     Pair graphs by shuffling batch $\{(G_{\pi_i}, G_i)\}_i^{|\mathcal{B}|}$
7:     $K \leftarrow$ random.choice$(\mathcal{K})$
8:     **for** $i = 1$ to $|\mathcal{B}|$ **do**
9:       $G_i' \leftarrow$ `Graph Transplant`$_{EP}((G_{\pi_i}, G_i),$
        $\mathcal{S}_{G_{\pi_i}}, K, R, \psi)$
10:       $y_i' \leftarrow \ell(y_{\pi_i}, y_i)$
11:     **end for**
      /* Update */
12:     $\theta \leftarrow \theta - \eta(\nabla_\theta \mathcal{L} + \nabla_\theta \frac{1}{|\mathcal{B}|} \sum_i \mathcal{L}_{Aug}(G_i', y_i'))$
13:     $\psi \leftarrow \psi - \eta \nabla_\psi \frac{1}{|\mathcal{B}|} \sum_i \mathcal{L}_{Aug}(G_i', y_i')$
14: **end while**

---

original local information. However, assigning the appropriate label to the generated graph is necessary to prevent the network from being misled by inappropriate supervision from noisy data. For example, suppose that the core nodes that mainly determine the property of destination graph G are condensed in a small region. If the nodes in this small region are eliminated via our subgraph selection method described above, the original property of G is mostly lost, and hence simply assigning the label in proportion to the number of nodes will not explain the generated graph properly. To tackle this issue, we propose a simple but effective label mixing strategy for *Graph Transplant*.

We define the importance of source subgraph $\bar{G}_\pi$ (and the remaining part of destination graph $\bar{G}$) as the total saliency of the nodes constituting the subgraph in the original full graph. This adaptive importance function $\mathcal{I}(\mathcal{S}, V)$ can be formulated as $\mathcal{I}(\mathcal{S}, V) = \frac{\sum_{v \in V} s_v}{\|\mathcal{S}\|_1}$. The label weight $\lambda_{G'}$ for generated graph G' is derived according to the relative importance of the parts being joined from the source $\mathcal{I}(\mathcal{S}_{G_\pi}, \bar{V}_\pi)$ and the destination $\mathcal{I}(\mathcal{S}_G, V \setminus \bar{V})$, so that the label for the mixed instance $\ell(y_\pi, y)$ is defined as $\lambda_{G'} y_\pi + (1 - \lambda_{G'})y$ where $\lambda_{G'} = \frac{\mathcal{I}(\mathcal{S}_{G_\pi}, \bar{V}_\pi)}{\mathcal{I}(\mathcal{S}_{G_\pi}, \bar{V}_\pi) + \mathcal{I}(\mathcal{S}_G, V \setminus \bar{V})}$.

Our final objective is

$$\min_{\theta, \psi} \mathbb{E}_{(G_\pi, y_\pi), (G, y) \sim \mathcal{D}}[\mathcal{L}(G, y) + \mathcal{L}_{Aug}(G', y')], \quad (1)$$

where $\mathcal{L}_{Aug}(G', y')$ is defined as $\lambda_{G'} \mathcal{L}(G', y_\pi) + (1 - \lambda_{G'})\mathcal{L}(G', y)$. The whole learning process with *Graph Transplant* is described in Algorithm 3.

5 Experiment

We perform a series of experiments on multiple domain datasets to verify our method *Graph Transplant*. Since this is the first input-level Mixup-based work on graph-structured data themselves, we compare the performance against Manifold Mixup (Wang et al. 2021) that mixes up the hidden representations after the pooling layer, and other stochastic graph augmentations (Section 5.1). Although theoretically *Graph Transplant* can be unified to any GNN architecture without network modification, we conduct the experiments with four different GNN architectures: GCN (Kipf and Welling 2016), GCS (GCN with learnable Skip connection), GAT (Veličković et al. 2017), GIN (Xu et al. 2018). We also conduct ablation studies (Section 5.4) to explore the contribution of each component of our method - node saliency, saliency-based label mixing, and locality preservation. Finally, we conduct the experiments of model calibration and adversarial attack (Section 5.5, 5.6).

5.1 Baselines

We now introduce the graph augmentation baselines. In addition to graph augmentation works, we also adopt general graph augmentations - DropN, PermE, MaskN, and SubG - from the recent graph contrastive learning work called GraphCL (You et al. 2020) by assigning original labels to the augmented graphs. Note that for all baselines, we train the models with both original and augmented graphs for fair comparisons. We defer the details and priors of baselines to Supplementary. Manifold Mixup (Wang et al. 2021) mixes the graph latent vector after the readout function γ and their labels. M-Evolve (Zhou, Shen, and Xuan 2020) first perturbs the edges with a heuristically designed open-triad edge swapping method. Then, noisy-augmented graphs are filtered out based on the validation set. We conduct augmentation and filtration for every training step and the filtration criterion is updated after each training epoch. DropN (Node dropping) randomly drops a certain portion of nodes. PermE (Edge perturbation) randomly adds and removes a certain ratio of edges. In line with Dropout (Srivastava et al. 2014), MaskN (Attribute masking) stochastically masks a certain ratio of node features. Using the random walk, SubG (Subgraph) extracts the subgraph from the original graph.

5.2 Experimental Settings

Datasets To demonstrate that *Graph Transplant* brings consistent improvement across various domains and dataset size , we conduct the experiment on 8 benchmark datasets: COLLAB (Yanardag and Vishwanathan 2015) for social networks dataset, ENZYMES (Schomburg et al. 2004), obgb-ppa (Hu et al. 2020) for bioinformatics dataset, COIL-DEL (Riesen and Bunke 2008) for computer vision dataset, and NCI1 (Wale, Watson, and Karypis 2008), Mutagenicity (Kazius, McGuire, and Bursi 2005), NCI-H23, MOLT-4, P388 (Yan et al. 2008) for molecules datasets.

Evaluation protocol We evaluate the model with 5-fold cross validation. For small-scale datasets, the experiments are repeated three times so the total of 15 different train/-validation/test stratified splits with the ratio of 3:1:1. We

Method	Dataset (*Small-scale*)					Dataset (*Medium-scale*)		
	COLLAB	NCI1	ENZYMES	Mutagenicity	COIL-DEL	NCI-H23	MOLT-4	P388
Vanilla (**GCS**)	$0.814_{\pm.010}$	$0.784_{\pm.025}$	$0.639_{\pm.040}$	$0.819_{\pm.009}$	$0.688_{\pm.019}$	$0.876_{\pm.009}$	$0.833_{\pm.011}$	$0.905_{\pm.009}$
Manifold Mixup	$0.814_{\pm.010}$	$0.792_{\pm.018}$	$0.664_{\pm.053}$	$0.822_{\pm.009}$	$0.756_{\pm.029}$	$0.884_{\pm.013}$	$\mathbf{0.842}_{\pm.011}$	$0.902_{\pm.021}$
M-Evolve	$0.817_{\pm.011}$	$0.775_{\pm.023}$	$0.666_{\pm.059}$	$0.821_{\pm.009}$	$0.703_{\pm.013}$	$0.875_{\pm.009}$	$0.829_{\pm.011}$	$0.904_{\pm.011}$
DropN	$0.816_{\pm.008}$	$0.789_{\pm0.031}$	$0.686_{\pm.056}$	$0.820_{\pm.018}$	$0.860_{\pm.021}$	$0.878_{\pm.012}$	$0.834_{\pm.011}$	$0.911_{\pm.013}$
PermE	$0.816_{\pm.006}$	$0.785_{\pm.023}$	$0.675_{\pm.052}$	$0.820_{\pm.015}$	$0.796_{\pm.016}$	$0.884_{\pm.004}$	$0.840_{\pm.010}$	$0.910_{\pm.014}$
MaskN	-	$0.790_{\pm.018}$	$0.678_{\pm.046}$	$0.819_{\pm.008}$	$0.791_{\pm.021}$	$0.877_{\pm.006}$	$0.838_{\pm.007}$	$0.907_{\pm.010}$
SubG	$0.812_{\pm.010}$	$0.784_{\pm.021}$	$0.675_{\pm.057}$	$0.821_{\pm.011}$	$0.819_{\pm.025}$	$0.877_{\pm.012}$	$0.836_{\pm.011}$	$0.911_{\pm.014}$
Graph Transplant$_{DP}$	$0.820_{\pm.011}$	$\mathbf{0.815}_{\pm.011}$	$0.694_{\pm.065}$	$0.821_{\pm.010}$	$0.858_{\pm.022}$	$0.880_{\pm.016}$	$\mathbf{0.843}_{\pm.003}$	$\mathbf{0.921}_{\pm.010}$
Graph Transplant$_{EP}$	$\mathbf{0.828}_{\pm.007}$	$0.807_{\pm.013}$	$\mathbf{0.707}_{\pm.053}$	$\mathbf{0.829}_{\pm.010}$	$\mathbf{0.865}_{\pm.016}$	$\mathbf{0.890}_{\pm.018}$	$\mathbf{0.843}_{\pm.009}$	$0.919_{\pm.007}$
Vanilla (**GIN**)	$0.805_{\pm.007}$	$0.793_{\pm.013}$	$0.596_{\pm.030}$	$0.815_{\pm.010}$	$0.620_{\pm.013}$	$0.879_{\pm.006}$	$0.834_{\pm.015}$	$0.901_{\pm.012}$
Manifold Mixup	$0.809_{\pm.008}$	$0.797_{\pm.013}$	$0.618_{\pm.042}$	$0.822_{\pm.011}$	$0.654_{\pm.020}$	$0.873_{\pm.015}$	$0.824_{\pm.009}$	$0.902_{\pm.006}$
M-Evolve	$0.809_{\pm.007}$	$0.785_{\pm.015}$	$0.608_{\pm.031}$	$0.813_{\pm.011}$	$0.604_{\pm.017}$	$0.877_{\pm.005}$	$0.832_{\pm.016}$	$0.895_{\pm.013}$
DropN	$0.807_{\pm.014}$	$0.789_{\pm.022}$	$0.619_{\pm.045}$	$0.822_{\pm.010}$	$0.665_{\pm.030}$	$0.850_{\pm.030}$	$0.821_{\pm.009}$	$0.896_{\pm.012}$
PermE	$0.809_{\pm.008}$	$0.790_{\pm.018}$	$0.641_{\pm.038}$	$0.823_{\pm.008}$	$0.689_{\pm.023}$	$0.867_{\pm.012}$	$0.836_{\pm.011}$	$0.910_{\pm.018}$
MaskN	-	$0.794_{\pm.017}$	$0.624_{\pm.028}$	$0.820_{\pm.008}$	$0.632_{\pm.017}$	$0.877_{\pm.007}$	$0.831_{\pm.014}$	$0.898_{\pm.017}$
SubG	$0.805_{\pm.010}$	$0.789_{\pm.021}$	$0.631_{\pm.030}$	$0.820_{\pm.008}$	$0.667_{\pm.022}$	$0.878_{\pm.005}$	$0.834_{\pm.014}$	$0.900_{\pm.010}$
Graph Transplant$_{DP}$	$0.814_{\pm.012}$	$\mathbf{0.815}_{\pm.011}$	$\mathbf{0.653}_{\pm.038}$	$0.822_{\pm.012}$	$\mathbf{0.700}_{\pm.025}$	$\mathbf{0.898}_{\pm.018}$	$\mathbf{0.855}_{\pm.013}$	$0.914_{\pm.009}$
Graph Transplant$_{EP}$	$\mathbf{0.815}_{\pm.009}$	$0.811_{\pm.011}$	$0.649_{\pm.039}$	$\mathbf{0.828}_{\pm.009}$	$0.682_{\pm.025}$	$0.890_{\pm.012}$	$0.846_{\pm.011}$	$\mathbf{0.916}_{\pm.010}$

Table 1: (Left) We report the averaged accuracy and the standard deviation of 3 repeats of 5-fold cross validation on five graph classification benchmark datasets (*Small-scale*). Results of GCN and GAT are deferred to Supplementary. Note that MaskN can not be applied to COLLAB which does not have node features. (Right) Our method also enhances the classification performance on the larger-scale datasets (*Medium-scale*). We report the mean and standard deviation of AUROC for 5-fold.

carefully design the unified training pipeline that can guarantee sufficient convergence. For small and medium-scale datasets, we train the GNNs for 1000 epochs under the early stopping condition and terminate training when there is no further increase in validation accuracy for 1500 iterations. Similarly, the learning rate is decayed by 0.5 if there is no improvement in validation loss for 1000 iterations. Further details are provided in Supplementary.

5.3 Main Results

We test the *Graph Transplant* on the five benchmark datasets (Table 1, Left). Subscripts DP and EP denote transplant methods (See **Transplant** in Section 4.2). *Graph Transplant* achieves the best performances at the most cases. It is worth noting that, our method consistently exhibits superiority for datasets with different characteristics such as sparsity of connection, graph size, and the presence or absence of node features, while other baselines are jagged. Interestingly, our method with simple DP generally beats the other baselines and is even superior to our method transplanting with EP in some cases. We conjecture that it helps to allow the mixed graphs to be variety within the weak but reasonable constraint on the expected node degree. We obtain further gain in several cases by exploiting the edge predictor that utilizes the node features to determine the connectivity.

We also verify *Graph Transplant* with larger-scale datasets. We test on molecular datasets with 40k samples (Table 1, Right) with the two superior GNNs - GCS, GIN - and ogbg-ppa with 150k samples (Table 2) with GCS. For class-imbalanced medium datasets, we use AUROC metric.

Method	Open Graph Benchmark (OGB)
(**GCS**)	*ogbg-ppa*
Vanilla	$0.638_{\pm0.007}$
Manifold Mixup	$0.616_{\pm0.013}$
DropN	$0.660_{\pm0.005}$
PermE	$0.655_{\pm0.003}$
MaskN	$0.627_{\pm0.012}$
SubG	$0.661_{\pm0.005}$
Graph Transplant$_{EP}$	$\mathbf{0.680}_{\pm0.002}$

Table 2: The averaged accuracy with standard deviation for 3 runs of different data augmentation methods on *ogbg-ppa*.

5.4 Ablation Study

To validate each component of *Graph Transplant*, we conduct the following four ablations : **1)** Edge Prediction **2)** Label mixing strategies: Size-based vs Saliency-based **3)** Comparison of mixing units: K-hop subgraph vs scattered node **4)** Importance of saliency-based subgraph. In Figure 2, each color represents each variant. The results for other datasets and detailed descriptions are in Supplementary.

5.5 Robustness to Adversarial Examples

Recently, mixup-variants prove their effectiveness in terms of model robustness (Hendrycks et al. 2019; Pang, Xu, and Zhu 2019). We test if the GNNs trained with *GraphTransplant* can enhance the model robustness against adversarial attacks. We attack the GNNs using gradient-based white-box attack called *GradArgMax* (Dai et al. 2018) designed for graph-structured data. In Table 3, our method shows superior
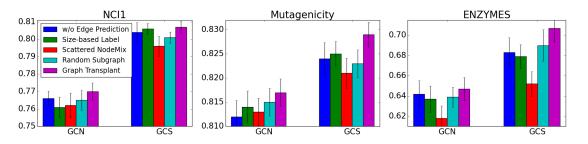
Figure 2: Ablation study. We draw a bar chart for each dataset, where bars are grouped by model architectures within a chart. The different ablation studies are distinguishable by colors. The rightmost bars (the best ones) are of our complete method.

defense performance compared to the other baselines. Interestingly, although the adversarial attack is based on edge modification, ours exhibits better performance than PermE. Detailed settings and extra results are in Supplementary.

5.6 Further Analysis

Model Calibration In image classification tasks, DNNs trained with mixup are known to significantly improves the model calibration. We evaluate if *Graph Transplant* can alleviate the over/under-confident prediction of GNNs. In Table 4, *Graph Transplant* has the lowest Expected Calibration Error (ECE) in most cases. This implies that the GNNs trained with *GraphTransplant* classifies the graphs with more accurate confidence, which is the better indicator of the actual likelihood of a correct prediction.

Qualitative analysis We analyze the node saliency of the molecule graphs in Mutagenicity to show that our method improves generalizability and leads the model to attend where it really matters to correctly classify molecules. The task is to decide whether the molecule causes DNA mutation or not, which is important to capture the interaction of functional groups in molecules (Luch 2005). For example, the first case in Figure 3 is of a mutagenic molecule in that it has two methyl groups attached to oxygen atoms, which are vulnerable to radical attack. Attacked by radical, the attached methyl groups are transferred to DNA base, so-called DNA methylation, entailing the change of DNA base property, resulting in mutation. While the model trained with *Graph Transplant* attends to the appropriate region colored by red,

Method	*GradArgmax* (White-Box Attack)		
(GCS)	ENZYMES	Mutagenicity	NCI1
Vanilla	0.543 ±0.030	0.579 ±0.013	0.596 ±0.027
Manifold Mixup	0.575 ±0.041	0.596 ±0.019	0.609 ±0.030
M-Evolve	0.593 ±0.030	0.603 ±0.042	0.628 ±0.028
DropN	0.617 ±0.030	0.617 ±0.030	0.631 ±0.028
PermE	0.623 ±0.071	0.645 ±0.031	0.602 ±0.042
MaskN	0.588±0.050	0.514 ±0.024	0.589 ±0.020
SubG	0.608 ±0.030	0.582 ±0.024	0.533 ±0.024
Graph Transplant$_{EP}$	**0.648** ±0.058	**0.672** ±0.010	**0.642** ±0.028

Table 3: Robustness against adversarial attack.

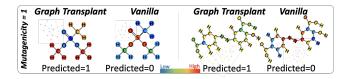| Method	Expected Calibration Error		
(GCS/GIN)	ENZYMES	Mutagenicity	COIL-DEL
Vanilla	0.299/0.361	0.068/0.093	0.088/0.021
Manifold Mixup	0.144/0.139	0.063/0.093	0.313/0.241
M-Evolve	0.285/0.319	0.056/0.069	0.065/0.099
DropN	0.252/0.327	0.051/0.080	0.051/0.327
PermE	0.242/0.301	0.052/0.069	0.062/0.327
MaskN	0.254/0.320	0.063/0.081	**0.046**/0.138
SubG	0.269/0.329	0.069/0.095	0.050/0.122
Graph Transplant$_{EP}$	**0.046/0.136**	**0.041/ 0.061**	0.051/**0.061**

Table 4: The model calibration on GCS and GIN.

Figure 3: Saliency from the trained models with *Graph Transplant* and vanilla. More cases are in Supplementary.

representing high node saliency, the vanilla model gives an incorrect answer with highlighting the unrelated spot. Next, the right molecule has 'OH' and 'COOH' functional groups which can be ionized, the resulting exposed electron could attack DNA base. The saliency of our method points out 'OH' the most. On the other hand, the vanilla model focuses on the core of symmetric structure locally and considers it stable so that it is non-mutagenic.

6 Conclusion

We proposed *Graph Transplant*, the first input-level Mixup strategy for graph-structured data, that generates mixed graphs via the guidance of the node saliency. To create semantically valuable graphs, *Graph Transplant* exploits the salient subgraphs as mixing units while preserving the local structure. Furthermore, supervisions for augmented graphs are determined in a context-aware manner to rectify the data-label mismatching problem. Through the extensive experiments, *Graph Transplant* demonstrated its effectiveness in that it outperforms the existing graph augmentations across the multiple domains of graphs and GNN architectures.

Acknowledgements

References

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828.

Chen, D.; Lin, Y.; Li, W.; Li, P.; Zhou, J.; and Sun, X. 2020. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 3438–3445.

Dai, H.; Li, H.; Tian, T.; Huang, X.; Wang, L.; Zhu, J.; and Song, L. 2018. Adversarial attack on graph structured data. In *International conference on machine learning*, 1115–1124. PMLR.

Devries, T.; and Taylor, G. W. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. *CoRR*.

Errica, F.; Podda, M.; Bacciu, D.; and Micheli, A. 2019. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 1263–1272. PMLR.

Guo, H.; Mao, Y.; and Zhang, R. 2019a. Augmenting data with mixup for sentence classification: An empirical study. *arXiv preprint arXiv:1905.08941*.

Guo, H.; Mao, Y.; and Zhang, R. 2019b. Mixup as locally linear out-of-manifold regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 3714–3722.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hendrycks, D.; Mu, N.; Cubuk, E. D.; Zoph, B.; Gilmer, J.; and Lakshminarayanan, B. 2019. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687*.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Kazius, J.; McGuire, R.; and Bursi, R. 2005. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of medicinal chemistry*, 48(1): 312–320.

Kim, J.-H.; Choo, W.; Jeong, H.; and Song, H. O. 2021. Comixup: Saliency guided joint mixup with supermodular diversity. *arXiv preprint arXiv:2102.03065*.

Kim, J.-H.; Choo, W.; and Song, H. O. 2020. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In *International Conference on Machine Learning*, 5275–5285. PMLR.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Luch, A. 2005. Nature and nurture–lessons from chemical carcinogenesis. *Nature Reviews Cancer*, 5(2): 113–125.

Medennikov, I.; Khokhlov, Y. Y.; Romanenko, A.; Popov, D.; Tomashenko, N. A.; Sorokin, I.; and Zatvornitskiy, A. 2018. An Investigation of Mixup Training Strategies for Acoustic Models in ASR. In *INTERSPEECH*, 2903–2907.

Meng, L.; and Zhang, J. 2019. Isonn: Isomorphic neural network for graph representation learning and classification. *arXiv preprint arXiv:1907.09495*.

Neil, D.; Briody, J.; Lacoste, A.; Sim, A.; Creed, P.; and Saffari, A. 2018. Interpretable graph convolutional neural networks for inference on noisy knowledge graphs. *arXiv preprint arXiv:1812.00279*.

Pang, T.; Xu, K.; and Zhu, J. 2019. Mixup inference: Better exploiting mixup to defend adversarial attacks. *arXiv preprint arXiv:1909.11515*.

Riesen, K.; and Bunke, H. 2008. IAM graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, 287–297. Springer.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2019. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*.

Schomburg, I.; Chang, A.; Ebeling, C.; Gremse, M.; Heldt, C.; Huhn, G.; and Schomburg, D. 2004. BRENDA, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1): D431–D433.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, 618–626.

Shrikumar, A.; Greenside, P.; and Kundaje, A. 2017. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, 3145–3153. PMLR.

Simonovsky, M.; and Komodakis, N. 2017. Dynamic edge-conditioned filters in convolutional neural networks on

graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3693–3702.

Simonyan, K.; Vedaldi, A.; and Zisserman, A. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Verma, V.; Lamb, A.; Beckham, C.; Najafi, A.; Mitliagkas, I.; Lopez-Paz, D.; and Bengio, Y. 2019a. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, 6438–6447. PMLR.

Verma, V.; Qu, M.; Lamb, A.; Bengio, Y.; Kannala, J.; and Tang, J. 2019b. Graphmix: Regularized training of graph neural networks for semi-supervised learning. *arXiv preprint arXiv:1909.11715*.

Wale, N.; Watson, I. A.; and Karypis, G. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3): 347–375.

Wang, Y.; Wang, W.; Liang, Y.; Cai, Y.; and Hooi, B. 2021. Mixup for Node and Graph Classification. In *Proceedings of the Web Conference 2021*, 3663–3674.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Yan, X.; Cheng, H.; Han, J.; and Yu, P. S. 2008. Mining Significant Graph Patterns by Leap Search. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, 433–444. Association for Computing Machinery.

Yanardag, P.; and Vishwanathan, S. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 1365–1374.

Ying, R.; Bourgeois, D.; You, J.; Zitnik, M.; and Leskovec, J. 2019. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32: 9240.

Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*.

You, Y.; Chen, T.; Sui, Y.; Chen, T.; Wang, Z.; and Shen, Y. 2020. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33.

Yun, S.; Han, D.; Oh, S. J.; Chun, S.; Choe, J.; and Yoo, Y. 2019. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 6023–6032.

Zhang, H.; Cisse, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*.

Zhao, T.; Liu, Y.; Neves, L.; Woodford, O.; Jiang, M.; and Shah, N. 2020. Data Augmentation for Graph Neural Networks. *arXiv preprint arXiv:2006.06830*.

Zhou, B.; Khosla, A.; Lapedriza, A.; Oliva, A.; and Torralba, A. 2016. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2921–2929.

Zhou, J.; Shen, J.; and Xuan, Q. 2020. Data Augmentation for Graph Classification. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2341–2344.