Imbalanced Graph Classification via Graph-of-Graph Neural Networks

Yu Wang
yu.wang.1@vanderbilt.edu
Vanderbilt University

Yuying Zhao
yuying.zhao@vanderbilt.edu
Vanderbilt University

Neil Shah
nshah@snap.com
Snap Research

Tyler Derr
tyler.derr@vanderbilt.edu
Vanderbilt University

ABSTRACT

Graph Neural Networks (GNNs) have achieved unprecedented success in identifying categorical labels of graphs. However, most existing graph classification problems with GNNs follow the protocol of balanced data splitting, which misaligns with many real-world scenarios in which some classes have much fewer labels than others. Directly training GNNs under this imbalanced scenario may lead to uninformative representations of graphs in minority classes, and compromise the overall classification performance, which signifies the importance of developing effective GNNs towards handling imbalanced graph classification. Existing methods are either tailored for non-graph structured data or designed specifically for imbalanced node classification while few focus on imbalanced graph classification. To this end, we introduce a novel framework, Graph-of-Graph Neural Networks (G^2GNN), which alleviates the graph imbalance issue by deriving extra supervision globally from neighboring graphs and locally from stochastic augmentations of graphs. Globally, we construct a graph of graphs (GoG) based on kernel similarity and perform GoG propagation to aggregate neighboring graph representations. Locally, we employ topological augmentation via masking node features or dropping edges with self-consistency regularization to generate stochastic augmentations of each graph that improve the model generalibility. Extensive graph classification experiments conducted on seven benchmark datasets demonstrate our proposed G^2GNN outperforms numerous baselines by roughly 5% in both F1-macro and F1-micro scores. Open-source code can be found at https://github.com/YuWVandy/G2GNN.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

Imbalanced graph classification, graph neural network, graph augmentations, graph of graphs

ACM Reference Format:
Yu Wang, Yuying Zhao, Neil Shah, and Tyler Derr. 2022. Imbalanced Graph Classification via Graph-of-Graph Neural Networks. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557356

1 INTRODUCTION

Employing graph representations for classification has recently attracted significant attention due to the emergence of Graph Neural Networks (GNNs) associated with its unprecedented power in expressing graph representations [42]. A typical GNN architecture for graph classification begins with an encoder that extracts node representations by propagating neighborhood information followed by pooling operations that integrate node representations into graph representations, which are then fed into a classifier to predict graph labels [8]. Although numerous GNN variants have been proposed by configuring different propagation and pooling schemes, most works are framed under the setting of balanced data-split where an equal number of labeled graphs are provided as the training data for each class [28]. However, collecting such balanced data tends to be time-intensive and resource-expensive, and thus are often impossible in reality [18].

In many real-world graph datasets, the distribution of graphs across classes varies from a slight bias to a severe imbalance where a large portion of classes contain a limited number of labeled graphs (minority classes) while few classes contain enough labeled graphs [6, 45] (majority classes). For example, despite the huge chemical space, few compounds are labeled active with the potential to interact with a target biomacromolecule; the remaining majority are labeled inactive [14, 21, 25]. Since most GNNs are designed and evaluated on balanced datasets, directly employing them on imbalanced datasets would compromise the overall classification performance. As one sub-branch of deep learning on graph-structured data, GNNs similarly inherit two severe problems from traditional deep learning on imbalanced datasets: inclination to learning towards majority classes [15] and poor generalization from given scarce training data to abounding unseen testing data [28, 48]. Aiming at these two challenges, traditional solutions include augmenting data via under- or over-sampling [4, 33], assigning weights to adjust the portion of training loss of different classes [31], and constructing synthetic training data via interpolation over minority instances to balance the training data [3]. However, these methods have been primarily designed on point-based data and their performance on graph-structured data is unclear.

CIKM '22, October 17–21, 2022, Atlanta, GA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557356

Imbalance on graph-structured data could lie either in the node or graph domain where nodes (graphs) in different classes have different amount of training data. Nearly all related GNN works focus on imbalanced node classification by either pre-training or adversarial training to reconstruct the graph topology [26, 38, 41, 51], while to the best of our knowledge, imbalanced graph classification with GNNs remains largely unexplored. On one hand, unlike node classification where we can derive extra supervision for minority nodes from their neighborhoods, graphs are individual instances that are isolated from each other and we cannot aggregate information directly from other graphs by propagation. On the other hand, compared with imbalance on regular grid or sequence data (e.g., images or text) where imbalance lies in feature or semantic domain, the imbalance of graph-structured data could also be attributed to the graph topology since unrepresentative topology presented by limited training graphs may ill-define minority classes that hardly generalize to the topology of diverse unseen testing graphs. To address the aforementioned challenges, we present Graph-of-Graph Neural Networks (G^{2}GNN), which consists of two essential components that seamlessly work together to derive supervision globally and locally. In summary, the main contributions are as follows:

- **Problem:** We investigate the problem of imbalanced graph classification, which is heavily unexplored in the GNN literature.
- **Algorithm:** We propose a novel framework G^{2}GNN for imbalanced graph classification, which derives extra supervision by globally aggregating from neighboring graphs and locally augmenting graphs with self-consistency regularization.
- **Experiments:** We perform extensive experiments on various real-world datasets to corroborate the effectiveness of G^{2}GNN on imbalanced graph classification.

We define imbalanced graph classification problem in section 2 and related work in section 3. The proposed framework, G^{2}GNN, is given in Section 4, consisting of global graph of graph construction/propagation and local graph augmentation. In Section 5, we conduct extensive experiments to validate the effectiveness of G^{2}GNN. Finally, we conclude and discuss future work in Section 6.

2 PROBLEM FORMULATION

Let $G = (\mathcal{V}^G, \mathcal{E}^G, \mathbf{X}^G)$ denote an attributed graph with node feature $\mathbf{X}^G \in \mathbb{R}^{|\mathcal{V}^G| \times d}$ and adjacency matrix $\mathbf{A}^G \in \mathbb{R}^{|\mathcal{V}^G| \times |\mathcal{V}^G|}$ where $\mathbf{A}^G_{ij} = 1$ if there is an edge between nodes v_i, v_j and vice versa. In graph classification, given a set of N graphs $\mathcal{G} = \{G_1, G_2, ..., G_N\}$ with each graph $G_i = (\mathcal{V}^{G_i}, \mathcal{E}^{G_i}, \mathbf{X}^{G_i})$ as defined above and their labels $\mathbf{Y} \in \mathbb{R}^{N \times C}$ where C is the total number of classes, we aim to learn graph representations $\mathbf{P} \in \mathbb{R}^{N \times d'}$ with $\mathbf{P}_i$ for each $G_i \in \mathcal{G}$ that is well-predictive of its one-hot encoded label $\mathbf{Y}_i$. The problem of imbalanced graph classification can be formalized as:

Problem 1. *Given a set of attributed graphs $\mathcal{G}$ with a subset of ℓ labeled graphs $\mathcal{G}^{\ell}$ that are imbalanced among different classes, we aim to learn a graph encoder and classifier $\mathcal{F} : \mathcal{F}(\mathbf{X}^{G_i}, \mathbf{A}^{G_i}) \rightarrow \mathbf{Y}_i$ that works well for graphs in both majority and minority classes.*

3 RELATED WORK

Graph Imbalance Problem. Graph imbalance exists in many real-world scenarios [51] where graph topology can be harnessed to derive extra supervision for learning graph/node representations. DR-GCN [26] handles multi-class imbalance by class-conditional adversarial training and latent distribution regularization. RECT [41] merges a GNN and proximity-based embeddings for the completely-imbalanced setting (i.e., some classes have no labeled nodes during training). GraphSMOTE [51] attempts to generate edges by pre-training an edge generator for isolated synthetic nodes generated from SMOTE [3]. Most recently, imGAGN [24] simulates both distributions of node attributes in minority classes and graph structures via generative adversarial model. However, all of these recent powerful deep learning works are proposed for node imbalance classification. Graph imbalance classification [23], remains largely unexplored, especially in GNN domain. Therefore, this work tackles this problem and different from previous work, we expect to leverage the graph topology via graph kernels to construct graph of graphs (GoG) and perform propagation on the constructed GoG.

Graph of Graphs. Graphs model entities by nodes and their relationships by edges. Sometimes, nodes at a higher level in a graph can be modeled as graphs at a lower level, which is termed as graph of graphs (GoG) [22]. This hierarchical relationship was initially used in [22] to rank nodes in a broader and finer context. Recently, [13] and [37] leverage Graph of Graphs (GoG) to perform link prediction between graphs and graph classification. However, both of them assume the GoG is provided in advance, e.g., [37] constructs edges between two molecule graphs based on their interactions and two drug graphs based on their side effects. Conversely, in this work, we construct a kNN GoG based on graph topological similarity and aggregate neighboring graph information by propagation on the constructed GoG.

Graph Augmentations. Recent years have witnessed successful applications of data augmentation in computer vision (CV) [27] and natural language processing (NLP) [9]. As its derivative in graph domain, graph augmentation enriches the training data [7, 49, 50] and therefore can be naturally leveraged to alleviate class imbalance. In this work, we augment graphs by randomly removing edges and masking node features [39, 44] to enhance the model generalizability and further employ self-consistency regularization to enforce the model to output low-entropy predictions [1].

4 THE PROPOSED FRAMEWORK

In this section, we introduce our proposed G^{2}GNN framework. Figure 1 presents an overview of G^{2}GNN, which is composed of two modules from global and local perspective. Globally, a graph kernel-based GoG construction is proposed to establish a k-nearest neighbor (kNN) graph and hence enable two-level propagation, where graph representations are first obtained via a GNN encoder and then neighboring graph representations are aggregated together through the GoG propagation on the established kNN GoG. Locally, we employ graph augmentation via masking node features or removing edges with self-consistency regularization to create novel supervision from stochastic augmentations of each individual graph. The GoG propagation serves as a global governance to retain the model discriminability by smoothing intra-class graphs while separating inter-class graphs. Meanwhile the topological augmentation behaves as a local explorer to enhance the model generalibility in discerning unseen non-training graphs. Next, we introduce details of each module.

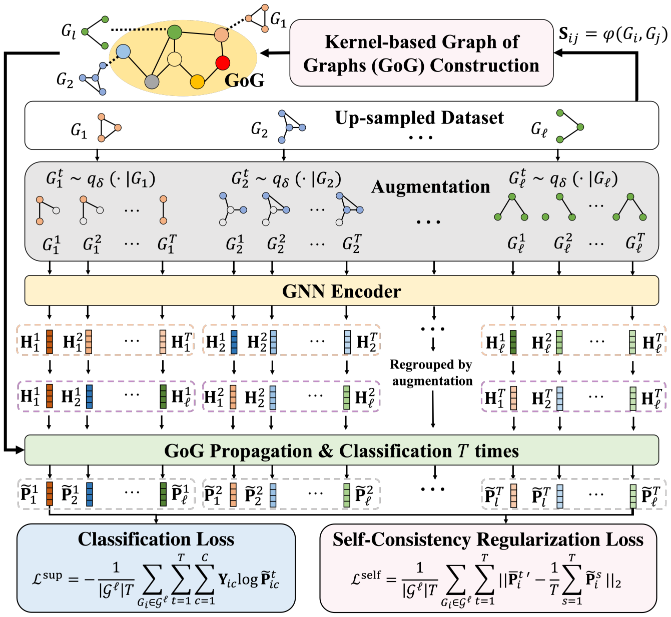

G_l
G_1
G_2
GoG
Kernel-based Graph of Graphs (GoG) Construction
$\mathbf{S}_{ij} = \varphi(G_i, G_j)$
Up-sampled Dataset
G_1
G_2
G_ℓ
$G_1^t \sim q_\delta(\cdot|G_1)$
$G_2^t \sim q_\delta(\cdot|G_2)$
Augmentation
$G_\ell^t \sim q_\delta(\cdot|G_\ell)$
G_1^1 G_1^2 ⋯ G_1^T
G_2^1 G_2^2 ⋯ G_2^T
G_ℓ^1 G_ℓ^2 ⋯ G_ℓ^T
GNN Encoder
$\mathbf{H}_1^1$ $\mathbf{H}_1^2$ ⋯ $\mathbf{H}_1^T$
$\mathbf{H}_2^1$ $\mathbf{H}_2^2$ ⋯ $\mathbf{H}_2^T$
$\mathbf{H}_\ell^1$ $\mathbf{H}_\ell^2$ ⋯ $\mathbf{H}_\ell^T$
Regrouped by augmentation
$\mathbf{H}_1^1$ $\mathbf{H}_2^1$ ⋯ $\mathbf{H}_\ell^1$
$\mathbf{H}_1^2$ $\mathbf{H}_2^2$ ⋯ $\mathbf{H}_\ell^2$
$\mathbf{H}_1^T$ $\mathbf{H}_2^T$ ⋯ $\mathbf{H}_\ell^T$
GoG Propagation & Classification T times
$\widetilde{\mathbf{P}}_1^1$ $\widetilde{\mathbf{P}}_2^1$ ⋯ $\widetilde{\mathbf{P}}_\ell^1$
$\widetilde{\mathbf{P}}_1^2$ $\widetilde{\mathbf{P}}_2^2$ ⋯ $\widetilde{\mathbf{P}}_\ell^2$
$\widetilde{\mathbf{P}}_1^T$ $\widetilde{\mathbf{P}}_2^T$ ⋯ $\widetilde{\mathbf{P}}_\ell^T$
Classification Loss
$\mathcal{L}^{\text{sup}} = -\frac{1}{|\mathcal{G}^\ell|T}\sum_{G_i \in \mathcal{G}^\ell}\sum_{t=1}^{T}\sum_{c=1}^{C}\mathbf{Y}_{ic}\log\widetilde{\mathbf{P}}_{ic}^t$
Self-Consistency Regularization Loss
$\mathcal{L}^{\text{self}} = \frac{1}{|\mathcal{G}^\ell|T}\sum_{G_i \in \mathcal{G}^\ell}\sum_{t=1}^{T}||\widetilde{\mathbf{P}}_i^{t\prime} - \frac{1}{T}\sum_{s=1}^{T}\widetilde{\mathbf{P}}_i^s||_2$

Figure 1: An overview of the Graph-of-Graph Neural Network (G^2GNN). To reduce imbalance effect on graph classification, we up-sample minority graphs, augment each graph T times followed by a GNN encoder to get their representations and regroup them according to their augmentation order, perform GoG propagation on constructed GoG T times with each time using all graph representations from that specific augmentation t, and finally forward the propagated representations through a classifier to compute classification loss and self-consistency regularization loss.

4.1 Global Imbalance Mitigation: Graph-of-Graph Construction and Propagation

Graph representations obtained by solely forwarding each graph through GNN encoders cannot be well-learned given scarce labeled training graphs in minority classes. Therefore, we construct a GoG to connect independent graphs and perform GoG propagation to aggregate the information of neighboring graphs. The intuition is that feature propagation and aggregation would mimic the way of SMOTE [3] and mixup [47], which are two of the most fundamental approaches handling the issue of class imbalance and poor generalizability. Aggregating representations of graphs of the same/different class/classes would simulate the interpolation of SMOTE/mixup with coefficients being determined by the specific graph convolution we use in propagation. In the following, we first introduce the basic GNN encoder to obtain graph representations, which will be later used for GoG propagation. Then we construct GoG and empirically demonstrate its high homophily, which naturally motivates the GoG propagation.

4.1.1 Basic GNN Encoder. In this work, we employ graph isomorphism network (GIN) as the encoder to learn graph representation given its distinguished discriminative power of different topology [42]. However, our framework holds for any other GNN encoder.

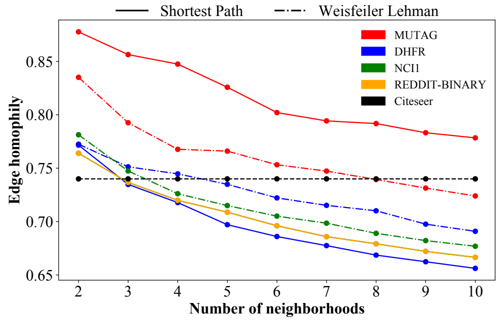


Figure 2: Edge homophily of constructed kNN GoGs.

One GIN layer is defined as:

$$\mathbf{X}^{G_i,l+1} = \text{MLP}^l((\mathbf{A}^{G_i} + (1+\epsilon)\mathbf{I})\mathbf{X}^{G_i,l}), \forall l \in \{1,2,...,L\} \quad (1)$$

where $\mathbf{X}^{G_i,l}$ is the intermediate node representation at layer l, $\mathbf{X}^{G_i,0} = \mathbf{X}^{G_i}$ is the initial node feature in the graph G_i, and MLP is a multi-layer perceptron at layer l. After L GIN convolutions, each node aggregates information from its neighborhoods up to L hops away and a readout function integrates node representations into the graph representation $\mathbf{H}_i$ for each graph G_i as:

$$\mathbf{H}_i = \text{READOUT}(\{\mathbf{X}_j^{G_i,L}|v_j \in \mathcal{V}^{G_i}\}) \quad (2)$$

Then we construct a kNN graph on top of each individual graph and perform GoG propagation to borrow neighboring graphs' information. Here we employ global-sum pooling as our READOUT function, which adds all nodes' representations to obtain the graph representation.

4.1.2 Graph of Graphs Construction. Given a set of graphs $\mathcal{G}$, we expect to construct a high-level graph where every graph $G_i \in \mathcal{G}$ is represented by a node and two graphs are linked by an edge if they are similar. In this work, we determine the graph similarity based on their topological similarity since graphs with similar topology typically possess similar functions or belong to the same class such as scaffold hopping [52] and enzyme identification [34]. Here we leverage the graph kernel to quantify topological similarity between pairs of graphs [43] and further use it to construct GoG. Denote the similarity matrix as $\mathbf{S} \in \mathbb{R}^{N \times N}$ where each entry $\mathbf{S}_{ij}$ measures the topological similarity between each pair of graphs (G_i, G_j) and is computed by the kernel function ϕ as:

$$\mathbf{S}_{ij} = \phi(G_i, G_j), \quad (3)$$

where multiple choices of the kernel function ϕ could be adopted here depending on specific types of topological similarity required by downstream tasks and in this work, we choose the Shortest Path Kernel due to its simplicity and effectiveness as demonstrated in Section 5. Then we construct a kNN graph $\mathcal{G}^{\text{kNN}}$ by connecting each graph G_i with its top-k similar graphs based on the similarity matrix $\mathbf{S}$ and then measure its edge homophily as:

$$\chi^{\mathcal{G}^{\text{kNN}}} = \frac{|\{(G_i, G_j) \in \mathcal{E}^{\mathcal{G}^{\text{kNN}}} : \mathbf{Y}_i = \mathbf{Y}_j\}|}{|\mathcal{E}^{\mathcal{G}^{\text{kNN}}}|}, \quad (4)$$

where high $\chi^{\mathcal{G}^{\text{kNN}}}$ means most edges connect graphs of the same class and by varying k, we end up with multiple $\mathcal{G}^{\text{kNN}}$ with different homophily level. Figure 2 visualizes the homophily of $\mathcal{G}^{\text{kNN}}$

constructed using Shortest-Path and Weisfeiler-Lehman kernels on three graph datasets populating in the literature [29, 44]. We can clearly see that edge homophily decreases as k increases because graphs with lower topological similarity have higher chance to be selected as neighborhoods while they likely belong to different classes from corresponding center graphs. However, edge homophily even when k is up to 5 is still in $[0.7, 0.8]$ and comparable to Citeseer dataset[1], which indicates that most edges in the constructed $\mathcal{G}^{\text{kNN}}$ connects graphs of the same class. Motivated by this observation, we perform GoG propagation on the generated kNN graph $\mathcal{G}^{\text{kNN}}$ to aggregate neighboring graph information.

4.1.3 Graph of Graphs Propagation. Denoting the adjacency matrix with added self-loops of the constructed graph $\mathcal{G}^{\text{kNN}}$ as $\hat{\mathbf{A}}^{\text{kNN}} = \mathbf{A}^{\text{kNN}} + \mathbf{I}$ and the corresponding degree matrix as $\hat{\mathbf{D}}^{\text{kNN}}$, the l^{th}-layer GoG propagation is formulated as:

$$\mathbf{P}^{l+1} = (\hat{\mathbf{D}}^{\text{kNN}})^{-1}\hat{\mathbf{A}}^{\text{kNN}}\mathbf{P}^{l}, l \in \{1, 2, ..., L\} \quad (5)$$

where $\mathbf{P}^0 = \mathbf{H}$ includes representations of all individual graphs $\mathbf{H}_i$ that are previously obtained from GIN followed by the graph pooling, as Eqs. (1)-(2). Note that here we do not differentiate between layers l, L used in GoG propagation here and layers used in GIN convolution in Section 4.1.1 since their difference is straightforward based on the context. After L layers propagation, the representation of a specific graph $\mathbf{P}_i^L$ aggregates information from neighboring graphs up to L hops away, which naturally smooths neighboring graphs and their labels by the following theorem [36]:

Theorem 4.1. *Suppose that the latent ground-truth mapping $\mathcal{M} : \mathbf{P}_i^l \rightarrow \mathbf{Y}_i$ from graph representations to graph labels is differentiable and satisfies $\mu-$Lipschitz constraints, i.e., $|\mathcal{M}(\mathbf{P}_i^l) - \mathcal{M}(\mathbf{P}_j^l)| \le \mu||\mathbf{P}_i^l - \mathbf{P}_j^l||_2$ for any pair of graphs G_i, G_j (μ is a constant), then the label smoothing is upper bounded by the feature smoothing among graph G_i and its neighboring graphs $\hat{\mathcal{N}}_i$ through* (6) *with an error $\boldsymbol{\epsilon}_i^l = \mathbf{P}_i^{l+1} - \mathbf{P}_i^l$:*

$$\underbrace{(\hat{\mathbf{d}}_i^{-1}\sum_{G_j \in \hat{\mathcal{N}}_i}\mathbf{Y}_j - \mathbf{Y}_i)}_{\text{Label smoothing}} - \underbrace{(\hat{\mathbf{d}}_i^{-1}\sum_{G_j \in \hat{\mathcal{N}}_i} o(||\mathbf{P}_j^l - \mathbf{P}_i^l||_2))}_{\text{Feature smoothing}} \le \mu\boldsymbol{\epsilon}_i^l. \quad (6)$$

Proof of Theorem 4.1 is provided in [36]. Specifically, $\boldsymbol{\epsilon}_i^l$ quantifies the difference of the graph G_i's representation between l^{th} and $(l+1)^{\text{th}}$ propagation, which decreases as propagation proceeds [19] and eventually converges after infinite propagation $\lim_{l\rightarrow\infty}\boldsymbol{\epsilon}_i^l = 0$ [20]. Treating each graph G_i as a node in $\mathcal{G}^{\text{kNN}}$ and its representation $\mathbf{P}_i^l$ gradually converges since $\lim_{l\rightarrow\infty}\epsilon_i^l = 0$. Such feature smoothing further leads to the label smoothing based on Theorem 4.1. Therefore propagating features according to (5) is equivalent to propagating labels among neighboring graphs, which derives extra information for imbalance classification. Given the high-homophily of the $\mathcal{G}^{\text{kNN}}$ in Figure 2, i.e, neighboring graphs tend to share the same class, the extra information derived from feature propagation (label propagation) is very likely beneficial to the performance of downstream classification.

4.2 Local Imbalance Mitigation: Self-consistency Regularization via Graph Augmentation

Even though feature propagation globally derives extra label information for graphs in minority classes from their neighboring graphs, training with limited graph instances still restricts the power of the model in recognizing numerous unseen non-training graphs. To retain the model generalibility, we further leverage two types of augmenting schemes, removing edges and masking node features [44], which are respectively introduced in the next.

4.2.1 Removing Edges: For each graph $G_i \in \mathcal{G}$, we randomly remove a subset of edges $\widehat{\mathcal{E}}^{G_i}$ from the original edge set $\mathcal{E}^{G_i}$ with probability: $P(e_{uv} \in \widehat{\mathcal{E}}^{G_i}) = 1 - \delta_{uv}^{G_i}$, where $\delta_{uv}^{G_i}$ could be uniform or adaptive for different edges. Since uniformly removing edges (i.e., $\delta_{uv}^{G_i} = \delta$) already enjoys a boost over baselines as shown in Section 5.2, we leave the adaptive one as future work.

4.2.2 Masking Node Features: Instead of directly removing nodes that may disconnect the original graph into several components, we retain the graph structure by simply zeroing entire features of some nodes following [10, 44]. Randomly masking entire features of some nodes enables each node to only aggregate information from a random subset of its neighborhoods multi-hops away, which reduces its dependency on particular neighborhoods. Compared with partially zeroing some feature channels, we empirically find that zeroing entire features generates more stochastic augmentations and achieves better performance. Formally, we randomly sample a binary mask $\eta_j^{G_i} \sim \text{Bernoulli}(1 - \delta_j^{G_i})$ for each node v_j in graph G_i and multiply it with the node feature, i.e., $\widehat{\mathbf{X}}_j^{G_i} = \eta_j^{G_i}\mathbf{X}_j^{G_i}$ [48].

For model simplicity, we unify the probability of removing edges and masking node features as a single augmentation ratio δ. Note that by using these augmentations after feature propagation, features of each node are stochastically mixed with its neighborhoods and create multiple augmented representations, which significantly increases the model generalibility if these augmented representations overlap with unseen non-training data. However, arbitrary modification of graph topology without any regularization could unintentionally introduce invalid or even abnormal topology. Therefore, we leverage self-consistency regularization to enforce the model to output low-entropy predictions [10].

4.2.3 Self-Consistency Regularization. Formally, given a set of T augmented variants of a graph G_i, $\widehat{\mathcal{G}}_i = \{G_i^1, G_i^2, ..., G_i^T | G_i^t \sim q_\delta(\cdot|G_i)\}$ where $q_\delta(\cdot|G_i)$ is the augmentation distribution conditioned on the original graph G_i parameterized by the augmentation ratio δ, we feed them through a graph encoder by Eq. (1)-(2) and the GoG propagation by Eq. (5) to obtain their representations $\{\mathbf{P}_i^1, \mathbf{P}_i^2, ..., \mathbf{P}_i^T\}$. More specifically, we forward the set of representations of all t^{th}-augmented graphs $\{\mathbf{H}_i^t | i \in \{1, 2, ..., |\mathcal{G}^{\ell}|\}\}$ through GoG propagation parallelly T times to obtain their representations $\{\mathbf{P}_i^t | i \in \{1, 2, ..., |\mathcal{G}^{\ell}|\}, t \in \{1, 2, ..., T\}\}$. Then we further apply the classifier to obtain their predicted label distributions $\{\widetilde{\mathbf{P}}_i^t = \sigma(g_{\boldsymbol{\theta}_g}(\mathbf{P}_i^t)) | i \in \{1, 2, ..., |\mathcal{G}^{\ell}|\}, t \in \{1, 2, ..., T\}\}$ where σ is the softmax normalization and $g_{\boldsymbol{\theta}_g}$ is a trainable classifier parametrized by $\boldsymbol{\theta}_g$. After that, we propose to optimize the consistency of predictions among T augmentations for each graph. We first calculate the center of label

[1] Citeseer: a well-known GNN node classification benchmark dataset [28].

distribution by taking the average of predicted distribution of all augmented variants for each specific graph G_i, i.e., $\hat{\mathbf{P}}_i = \frac{1}{T}\sum_{t=1}^{T} \widetilde{\mathbf{P}}_i^t$. Then we sharpen [1] this label distribution center:

$$\bar{\mathbf{P}}_{ij} = (\hat{\mathbf{P}}_{ij})^{\tau} / \sum_{c=1}^{C} (\hat{\mathbf{P}}_{ic})^{\tau}, \forall j \in \{1, 2, ..., C\}, i \in \{1, 2, ..., |\mathcal{G}^{\ell}|\} \quad (7)$$

where $\tau \in [0, 1]$ acts as the temperature to control the sharpness of the predicted label distribution and as $\tau \longrightarrow \infty$, the sharpened label distribution of each graph approaches a one-hot distribution and hence becomes more informative. Then the self-consistency regularization loss for the graph G_i is formulated as the average L_2 distance between the predicted distribution of each augmented graph $\widetilde{\mathbf{P}}_i^t$ and their sharpened average predicted distribution:

$$\mathcal{L}_i^{\text{self}} = \frac{1}{T} \sum_{t=1}^{T} ||\bar{\mathbf{P}}_i - \widetilde{\mathbf{P}}_i^t||_2. \quad (8)$$

Optimizing (8) requires the encoder and classifier to output similar predicted class distribution of different augmentations of each graph to the center one; this prevents the decision boundary of the whole model from passing through high-density regions of the marginal data distribution [12]. Also, as we increase τ, we can enforce the model to output low-entropy (high-confidence) predictions.

4.3 Objective Function and Prediction

The overall objective function of G^{2}GNN is formally defined as:

$$\mathcal{L} = \underbrace{-\frac{1}{|\mathcal{G}^{\ell}|T} \sum_{G_i \in \mathcal{G}^{\ell}} \sum_{t=1}^{T} \sum_{c=1}^{C} \mathrm{Y}_{ic} \log \widetilde{\mathbf{P}}_{ic}^t}_{\mathcal{L}^{\text{sup}}} + \underbrace{\frac{1}{|\mathcal{G}^{\ell}|T} \sum_{G_i \in \mathcal{G}^{\ell}} \sum_{t=1}^{T} ||\bar{\mathbf{P}}_i - \widetilde{\mathbf{P}}_i^t||_2}_{\mathcal{L}^{\text{self}}}, \quad (9)$$

where $\mathcal{L}^{\text{sup}}$ is the cross entropy loss over all training graphs in $\mathcal{G}^{\ell}$ with known label information as previously defined with C graph classes to be predicted, and $\mathcal{L}^{\text{self}}$ is the self-consistency regularization loss defined by Eq. (8) over all training graphs.

To predict classes of graphs in validation/testing set, instead of forwarding each individual unlabeled graph through the already-trained encoder f_{θ_f} and the classifier g_{θ_g} to predict its label, we first generate T augmented variants of each unlabeled graph $\widehat{\mathcal{G}}_i = \{G_i^1, G_i^2, ..., G_i^T | G_i^t \sim q_{\delta}(\cdot|G_i)\}, \forall G_i \in \mathcal{G}/\mathcal{G}^{\ell}$ following Section 4.2 and then collectively forward the group of augmented graphs through f_{θ_f}, GoG propagation and the classifier g_{θ_g} to obtain their predicted label distribution $\{\widetilde{\mathbf{P}}_i^1, \widetilde{\mathbf{P}}_i^2, ..., \widetilde{\mathbf{P}}_i^T\}$, then the final predicted distribution of graph $\mathcal{G}_i$ is averaged over all augmented variants as $\frac{1}{T}\sum_{t=1}^{T} \widetilde{\mathbf{P}_i}^t, \forall \mathcal{G}_i \in \mathcal{G}/\mathcal{G}^{\ell}$ and the final predicted class is the one that owns the highest class probability, i.e., $y_i = \underset{j \in \{1,2,...,C\}}{\arg\max} \frac{1}{T}\sum_{t=1}^{T} \widetilde{\mathbf{P}}_{ij}^t$.

4.4 Algorithm

In Algorithm 1, we present a holistic overview of the key stages in the proposed G^{2}GNN framework. Note that the GoG propagation and the graph augmentation with self-consistency regularization are both proposed to create more supervision from scarce minority training graphs, which can only handle the poor generalization problem. To avoid the problem of inclination to learning towards majority classes as mentioned in Section 1, we up-sample minority labeled graphs till the graphs in training and validation set are both balanced among different classes before starting the whole training processes as step 3 shows here. Balancing the labeled graphs in training set cannot only balance the training loss computed by Eq. (9) but also provide sufficient graphs from minority class to construct GoG. Otherwise given only few graphs in the minority class, the top-k similar graphs to one graph in minority class would be more likely come from majority class, which would further cause inter-class feature smoothing when performing GoG propagation and hence compromise the classification performance. Balancing the labeled graphs in validation set could avoid the imbalanced bias introduced in determining which model should be preserved for later evaluation. Note that in Table 2, we show that even equipping other baselines with up-sampling to remove the imbalanced training bias, G^{2}GNN still achieves better performance, which demonstrates that the performance improvement is not solely caused by the technique of up-sampling but also by the proposed GoG propagation and augmentation with self-consistency regularization.

Algorithm 1: The algorithm of G^{2}GNN

```
Input: The imbalanced set of labeled graphs $\mathcal{G}^{\ell}$, the kernel function $\phi$, the
       augmentation distribution $q_{\delta}$, the encoder $f_{\theta_f}$ and the classifier $g_{\theta_g}$
       with their learning rate $\alpha_f, \alpha_g$.
1  Compute pairwise similarity matrix S by Eq. (3) and construct $\mathcal{G}^{\text{kNN}}$
     following Section 4.1.2
2  Up-sample minority graphs in $\mathcal{G}^{\ell}$ for both training and validation sets
3  while not converged do
4      for mini-batch of graphs $\mathcal{G}^B = \{G_i | G_i \sim \mathcal{G}^{\ell}, i = \{1, 2, ..., |\mathcal{G}^B|\}\}$ do
5          Find top-$k$ similar graphs for each $G_i \in \mathcal{G}^B$ based on S and
             incorporate them into $\mathcal{G}^B$                        // Section 4.1.2
6          Obtain the subgraph $\mathcal{G}^{\text{kNN},B}$ from $\mathcal{G}^{\text{kNN}}$ induced by graphs in $\mathcal{G}^B$
7          For each $G_i \in \mathcal{G}^B$, generate $T$ augmented graphs
             $\widehat{\mathcal{G}}_i = \{G_i^1, G_i^2, ..., G_i^T | G_i^t \sim q_{\delta}(\cdot|G_i)\}$          // Section 4.2
8          Apply graph encoder $f_{\theta_f}$ by Eqs. (1)-(2), the GoG propagation by
             Eq. (5), and the classifier $g_{\theta_g}$ to predict graph class distribution
             $\{\widetilde{\mathbf{P}}_i^t | G_i \in \mathcal{G}^{\ell}, t \in T\}$                        // Section 4.1.3
9          Compute loss by Eq. (9) and update parameters
10         $\theta_g \leftarrow \theta_g - \alpha_g * \nabla_{\theta_g} \mathcal{L}, \theta_f \leftarrow \theta_f - \alpha_f * \nabla_{\theta_f} \mathcal{L}$ // Section 4.3
```

4.5 Complexity Analysis

Next, we compare our proposed G^{2}GNN with vanilla GNN-based encoders by analyzing the time and model compelxity. Since we employ shortest path kernel for all experiments in this work, we only analyze our models with this specific graph kernel.

In comparison to vanilla GNN-based encoders, additional computational requirements come from three components: kernel-based GoG construction and topological augmentation. In kernel-based GoG construction, applying shortest path kernel to calculate the similarity between every pair of graphs requires $O(n^3)$ [2] time and thus the total time complexity of this part is $O(\binom{|\mathcal{G}|}{2}\tilde{n}^3)$ ($\tilde{n} = \max_{G_i \in \mathcal{G}}(|\mathcal{V}^{G_i}|)$) due to the total $|\mathcal{G}|$ graphs. After computing the pairwise similarity, we can construct the GoG by naively thresholding out the top$-k$ similar graphs for each graph and the time complexity here is $O(|\mathcal{G}|k)$. By default $k \leq |\mathcal{G}|$, we directly have $O(|\mathcal{G}|k) < O(|\mathcal{G}|^2) = O(\binom{|\mathcal{G}|}{2}) < O(\binom{|\mathcal{G}|}{2}\tilde{n}^3)$ and hence the time complexity of the first module is $O(\binom{|\mathcal{G}|}{2}\tilde{n}^3)$. Despite the prohibitively heavy computation of $O(\binom{|\mathcal{G}|}{2}\tilde{n}^3)$, the whole module is a pre-procession computation once for all and we can further save the already computed similarity matrix S for future use, which

Table 1: Statistics of datasets

Networks	# Graphs	# Avg-Node	# Avg-Edge	# Attr	Time(s)*
PTC-MR [32]	344	14.29	14.69	18	0.257
NCI1 [35]	4110	29.87	32.30	37	11.21
MUTAG [5]	188	17.93	19.79	7	0.212
PROTEINS [43]	1113	39.06	72.82	3	11.36
D&D [30]	1178	284.32	715.66	89	574.71
DHFR [30]	756	42.43	44.54	3	3.70
REDDITB [43]	2000	429.63	497.75	\	3376

* The column 'time' represents the actual time used for applying Shortest Path kernel to compute S for each dataset.

therefore imposes no computational challenge. In topological augmentation, we augment graphs T times during each training epoch and each time we either go over all its edges or nodes, therefore the total time complexity of this module during each training epoch is $O(T\sum_{G_i\in\mathcal{G}^B}(|\mathcal{V}^{G_i}|+|\mathcal{E}^{G_i}|))$. Since augmenting graphs multiple times gains no further improvement than 2 [44], we fix T to be the constant 2 and therefore the total complexity of this part is linearly proportional to the size of each graph, which imposes no additional time compared with GNN encoders. Among the GoG propagation component, the most computational part comes from propagation in Eq. (5), which can be efficiently computed by applying power iteration from the edge view in $O(K|\mathcal{E}^{\mathcal{G}^{\text{kNN},B}}|)$ for each subgraph induced by graphs in batch $\mathcal{G}^B$. Based on experimental results in Figure 5a-5b, we usually choose k to be small to ensure the sparcity and the high homophily of GoG, then $O(K|\mathcal{E}^{\mathcal{G}^{\text{kNN},B}}|)$ can be neglected compared with applying GNN encoders to get representations of each graph, $O(K\sum_{G_i\in\mathcal{G}^B}|\mathcal{E}^{G_i}|)$. For the model complexity, besides the parameters of GNN encoders, G^{2}GNN adds no additional parameters and therefore its model complexity is exactly the same as traditional GNN encoders.

In summary, our model introduces no extra model complexity but $O(\binom{|\mathcal{G}|}{2}\tilde{n}^3)$ extra time complexity in the pre-procession stage. We further presents the actual time used for applying Shortest Path kernel to compute $\mathbf{S}$ in Table 1. It can be clearly see that similarity matrix $\mathbf{S}$ is calculated in a short time for each dataset other than D&D and REDDIT-B since graphs in these two dataset are on average denser than other datasets as shown in Table 1. However, we can simply pre-compute this $\mathbf{S}$ once for all and reuse it for G^{2}GNN. Moreover, we can make this computation feasible by either employing the fast shortest-path kernel computations by sampling-based approximation where we sample pairs of nodes and compute shortest paths between them [16].

5 EVALUATION

In this section, we evaluate the effectiveness of G^{2}GNN by conducting extensive imbalanced graph classification on multiple types of graph datasets with different levels of imbalance. We begin by introducing the experimental setup, including datasets, baselines, evaluation metrics, and parameter settings.

5.1 Experimental Setup

5.1.1 Dataset. We conduct experiments on seven widely-adopted real-world datasets [30, 43], which include: (1) Chemical compounds: PTC-MR, NCI1, and MUTAG. (2) Protein compounds: PROTEINS, D&D, and DHFR. (3) Social network: REDDIT-B. Details of these datasets can be found in Table 1.

5.1.2 Baselines. To evaluate the effectiveness of the proposed G^{2}GNN, we select three models designed for graph classification, which includes:

- **GIN** [42]: A basic supervised GNN model for graph classification due to its distinguished expressiveness of graph topology.
- **InfoGraph** [29]: An unsupervised GNN model for learning graph representations via maximizing mutual information between the original graph and its substructures of different scales.
- **GraphCL** [44]: Stepping further from InfoGraph, GraphCL proposes four strategies to augment graphs and learns graph representations by maximizing the mutual information between the original graph and its augmented variants.

Since imbalanced datasets naturally provide weak supervision on minority classes, unsupervised GNNs outweigh supervised counterparts and selecting them as baselines could more confidently justify the superiority of our model. All the above three baselines are proposed without consideration of imbalanced setting, therefore we further equip these three backbones with strategies designed specifically for handling imbalance issue, which includes:

- **Upsampling** (*us*): A classical approach that repeats samples from minority classes [17]. We implement this directly in the input space by duplicating minority graphs.
- **Reweighting** (*rw*): A general cost-sensitive approach introduced in [46] that assigns class-specific weights in computing the classification loss term in Eq. (9); we set the weights of each class as inverse ratio of the total training graphs to the number of training graphs in that class.
- **SMOTE** (*st*): Based on the ideas of SMOTE [3], synthetic minority samples are created by interpolating minority samples with their nearest neighbors within the same class based on the output of last GNN layer. Since directly interpolating in the topological space may generate invalid graph topology, here we first obtain graph representations by GNN-based encoders and interpolate minority graph representations in the embedding space to generate more minority training instances. Here, the nearest neighbors are computed according to Euclidean distance.

Equipping each of the above three backbones with up-sampling, re-weighting, and SMOTE strategies tailored specifically for imbalanced classification, we end up with 10 baselines. Specifically, we equip up-sampling and re-weighting with all three backbones and name each new baseline by combining the name of its backbone and the equipped strategy, e.g, GIN$_{us}$ represents the backbone GIN equipped with the up-sampling strategy. Since applying SMOTE empirically leads to similar or even worse performance gains, we only stack it on the GIN backbone.

5.1.3 Evaluation Metrics. Following existing work in imbalanced classification [51], we use two criterion: F1-macro and F1-micro to measure the performance of G^{2}GNN and other baselines. F1-macro computes the accuracy independently for each class and then takes the average (i.e., treating different classes equally). F1-micro computes accuracy over all testing examples at once, which may underweight the minority classes. Following [10], The whole GoG propagation is conducted in the transductive setting where representations of graphs in the training set could aggregate representations of graphs in the validation and testing sets while the classification loss is only evaluated on the given training labels.

Table 2: Graph classification performance on seven datasets. Note that the standard deviation is relatively higher since we focus on the imbalance problem and use 50 different data splits (i.e., having different training data distributions). G^{2}GNN$_e$ and G^{2}GNN$_n$ represent our proposed model using the removing edges and masking node features augmentation strategy, respectively. Red (blue) denotes the best (runner-up) model.

Model	**MUTAG** (5:45)		**PROTEINS** (30:270)		**D&D** (30:270)		**NCI1** (100:900)	
	F1-macro	F1-micro	F1-macro	F1-micro	F1-macro	F1-micro	F1-macro	F1-micro
GIN	52.50 ± 18.70	56.77 ± 14.14	25.33 ± 7.53	28.50 ± 5.82	9.99 ± 7.44	11.88 ± 9.49	18.24 ± 7.58	18.94 ± 7.12
GIN$_{us}$	78.03 ± 7.62	78.77 ± 7.67	65.64 ± 2.67	71.55 ± 3.19	41.15 ± 3.74	70.56 ± 10.28	59.19 ± 4.39	71.80 ± 7.02
GIN$_{rw}$	77.00 ± 9.59	77.68 ± 9.30	54.54 ± 6.29	55.77 ± 7.11	28.49 ± 5.92	40.79 ± 11.84	36.84 ± 8.46	39.19 ± 10.05
GIN$_{st}$	74.61 ± 9.66	75.11 ± 9.87	56.07 ± 7.95	57.85 ± 8.70	27.08 ± 8.63	39.01 ± 15.87	40.40 ± 9.63	44.48 ± 12.05
InfoGraph	69.11 ± 9.03	69.68 ± 7.77	35.91 ± 7.58	36.81 ± 6.51	21.41 ± 4.51	27.68 ± 7.52	33.09 ± 3.30	34.03 ± 3.68
InfoGraph$_{us}$	78.62 ± 6.84	79.09 ± 6.86	62.68 ± 2.70	66.02 ± 3.18	41.55 ± 2.32	71.34 ± 6.76	53.38 ± 1.88	62.20 ± 2.63
InfoGraph$_{rw}$	80.85 ± 7.75	81.68 ± 7.83	65.73 ± 3.10	69.60 ± 3.68	41.92 ± 2.28	72.43 ± 6.63	53.05 ± 1.12	62.45 ± 1.89
GraphCL	66.82 ± 11.56	67.77 ± 9.78	40.86 ± 6.94	41.24 ± 6.38	21.02 ± 3.05	26.80 ± 4.95	31.02 ± 2.69	31.62 ± 3.05
GraphCL$_{us}$	80.06 ± 7.79	80.45 ± 7.86	64.21 ± 2.53	65.76 ± 2.61	38.96 ± 3.01	64.23 ± 8.10	49.92 ± 2.15	58.29 ± 3.30
GraphCL$_{rw}$	80.20 ± 7.27	80.84 ± 7.43	63.46 ± 2.42	64.97 ± 2.41	40.29 ± 3.31	67.96 ± 8.98	50.05 ± 2.09	58.18 ± 3.08
G^{2}GNN$_e$	80.37 ± 6.73	81.25 ± 6.87	67.70 ± 2.96	73.10 ± 4.05	43.25 ± 3.91	77.03 ± 9.98	63.60 ± 1.57	72.97 ± 1.81
G^{2}GNN$_n$	83.01 ± 7.01	83.59 ± 7.14	67.39 ± 2.99	73.30 ± 4.19	43.93 ± 3.46	79.03 ± 10.78	64.78 ± 2.86	74.91 ± 2.14

Model	**PTC-MR** (9:81)		**DHFR** (12:108)		**REDDIT-B** (50:450)		**Ave. Rank**	
	F1-macro	F1-micro	F1-macro	F1-micro	F1-macro	F1-micro	F1-macro	F1-micro
GIN	17.74 ± 6.49	20.30 ± 6.06	35.96 ± 8.87	49.46 ± 4.90	33.19 ± 14.26	36.02 ± 17.38	12.00	12.00
GIN$_{us}$	44.78 ± 8.01	55.43 ± 14.25	55.96 ± 10.06	59.39 ± 6.52	66.71 ± 3.92	83.00 ± 5.18	5.00	4.43
GIN$_{rw}$	36.96 ± 14.08	43.09 ± 20.01	55.16 ± 9.47	57.78 ± 6.69	45.17 ± 8.46	51.92 ± 12.29	8.86	8.86
GIN$_{st}$	36.30 ± 11.45	40.04 ± 15.32	56.06 ± 9.60	58.48 ± 6.42	60.05 ± 4.14	73.59 ± 6.05	8.29	8.43
InfoGraph	25.85 ± 6.14	26.71 ± 6.50	50.62 ± 8.33	56.28 ± 4.58	57.67 ± 3.80	67.10 ± 4.91	10.00	10.14
InfoGraph$_{us}$	44.29 ± 4.69	48.91 ± 7.49	59.49 ± 5.20	61.62 ± 4.18	67.01 ± 3.34	78.68 ± 3.71	5.00	5.00
InfoGraph$_{rw}$	44.09 ± 5.62	49.17 ± 8.78	58.67 ± 5.82	60.24 ± 4.80	65.79 ± 3.38	77.35 ± 3.96	4.43	4.29
GraphCL	24.22 ± 6.21	25.16 ± 5.25	50.55 ± 10.01	56.31 ± 6.12	53.40 ± 4.06	62.19 ± 5.68	10.71	10.57
GraphCL$_{us}$	45.12 ± 7.33	53.50 ± 13.31	60.29 ± 9.04	61.71 ± 6.75	62.01 ± 3.97	75.84 ± 3.98	5.29	5.43
GraphCL$_{rw}$	44.75 ± 7.62	52.22 ± 13.24	60.87 ± 6.33	61.93 ± 5.15	62.79 ± 6.93	76.15 ± 9.15	5.00	5.29
G^{2}GNN$_e$	46.40 ± 7.73	56.61 ± 13.72	61.63 ± 10.02	63.61 ± 6.05	68.39 ± 2.97	86.35 ± 2.27	1.71	1.86
G^{2}GNN$_n$	46.61 ± 8.27	56.70 ± 14.81	59.72 ± 6.83	61.27 ± 5.40	67.52 ± 2.60	85.43 ± 1.80	1.71	1.71

5.1.4 Parameter Settings. We implement our proposed G^{2}GNN and some necessary baselines using Pytorch Geometric [11]. For InfoGraph[2] and GraphCL[3] we use the original authors' code with any necessary modifications. Aiming to provide a rigorous and fair comparison across models on each dataset, we tune hyperparameters for all models individually as: the weight decay $\in [0, 0.1]$, the encoder hidden units $\in \{128, 256\}$, the learning rate $\in \{0.001, 0.01\}$, the inter-network level propagation $L \in \{1, 2, 3\}$, the augmentation ratio $\delta \in \{0.05, 0.1, 0.2\}$, the number of neighboring graphs in constructing GoG $k \in \{2, 3, 4\}$, the augmentation number $T = 2$ and sharpening temperature $\tau = 0.5$. We employ Shortest Path Kernel to compute similarity matrix $\mathbf{S}$ and set the trainable classifier g as a 2-layer MLP. For REDDITB dataset, we use one-hot encoding of the node degree as the feature of each node following [29, 44]. For reproducibility, the code of the model with its corresponding hyperparameter configurations are publicly available[4].

5.2 Performance Comparison

In this subsection, we compare the performance of **G^{2}GNN$_e$** and **G^{2}GNN$_n$**, which represent the G^{2}GNN framework with the edge removal or node feature masking as augmentation, respectively, against the aforementioned baselines. Since class distributions of most datasets are not strictly imbalanced, we use an imitative imbalanced setting: we randomly set 25%/25% graphs as training/validation sets and among each of them, we choose one class as minority and reduce the graphs of this class in the training set (increase the other one) till the imbalance ratio reaches 1:9, which creates an extremely imbalanced scenario[5]. We average the performance per metric across 50 different data splits to avoid any bias from data splitting. Table 2 reports the mean and the standard deviation of the performance.

We observe from Table 2 that G^{2}GNN performs the best in all 7 datasets under both F1-macro and F1-micro. Moreover, edge removing (i.e., G^{2}GNN$_e$) benefits more on the social network (i.e., REDDIT-B) while node feature masking (i.e., G^{2}GNN$_n$) enhances more on biochemical molecules (e.g., MUTAG, D&D, NCI1 and PTC-MR), which conforms to [44] and is partially attributed to no node attributes presented in the social network. Models that are specifically designed for tackling the class imbalance issue generally perform better than the corresponding bare backbones without any strategy handling imbalance. The inferior performance of GIN$_{rw(st)}$ to GIN$_{us}$ is because we either set weights for adjusting training loss of different classes or generate synthetic samples based on training data at current batch. Since the number of training instances in each batch may not strictly follow the prescribed imbalance ratio, the batch-dependent weight or synthetic samples hardly guarantee the global balance. InfoGraph(GraphCL)-based variants do not suffer from the issue introduced by batch-training since once we obtain graph representations from pre-trained models by mutual information maximization, we feed them through downstream classifiers all at once without any involvement of batch process. Therefore, the performance of InfoGraph(GraphCL)$_{rw(st)}$ is comparable to InfoGraph(GraphCL)$_{us}$. We emphasize that the larger standard deviation in our setting is due to the significantly

[2] https://github.com/fanyun-sun/InfoGraph

[3] https://github.com/Shen-Lab/GraphCL

[4] Code for G^{2}GNN: https://github.com/submissionconff/G2GNN

[5] We select the amount of training and validation data as 25% to ensure the sufficiency of minority instances in both training and validation set given the imitative data distribution is at such a skewed level

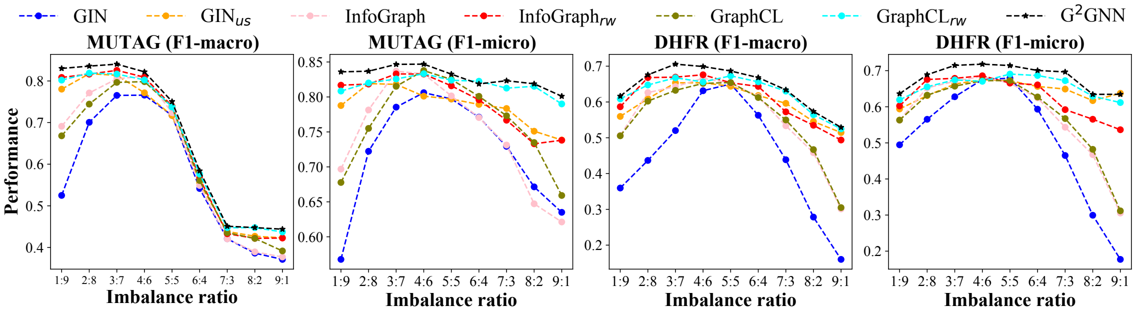


Figure 3: Graph classification results under different class imbalance ratios where 5:5 corresponds to balanced scenario while 1:9 and 9:1 correspond to highly imbalance scenario. Compared with GIN(blue), InfoGraph(pink), GraphCL(olive) designed not specifically for imbalanced scenario, our G^{2}GNN(black) model outperforms all of them in nearly all imbalance ratio settings and the margin further increases as the level of imbalance increases (i.e., deviates from the balanced scenario). Note that here we use the same amount of training and validation graphs (25%/25%) as used in Table 2.

different training data across different runs. We further argue that this standard deviation cannot be reduced by only increasing the number of runs due to the imbalance nature of the problem. However, the higher average performance of our model still signifies its superiority in handling a wide range of imbalanced data splittings.

5.3 Influence of Imbalance Ratio

We further compare the performance of our model with other baselines under different imbalance ratios. We vary the imbalance ratio from 1:9 to 9:1 by fixing the total number of training and validation graphs as 25%/25% of the whole dataset as before and gradually varying the number of graphs in different classes, which exhausts the imbalance scenarios from being balanced (5:5) to the extremely imbalanced (1:9 or 9:1) scenarios. Note that for clear comparison, we only visualize the performance of the best variant among each of three backbones in Figure 3. We can clearly see that the performance of all models first increases and then decreases as the imbalance ratio increases from 0.1 to 0.9, which demonstrates the detrimental effect of data imbalance on the model performance and such detrimental effect becomes even worse when the imbalance becomes more severe. Furthermore, the F1-macro score of our G^{2}GNN model clearly outperforms all other baselines on both MUTAG and DHFR under each imbalance ratio, which soundly justifies the superiority and robustness of our model in alleviating imbalance of different level. Different from supervision presented from given labeled data, the extra supervision derived by leveraging neighboring graphs' information via propagation and topological augmentation is weakly influenced by the amount of training data. Therefore, the margin achieved by our model further grows when imbalance ratio is either too low or too high compared with GIN, InfoGraph and GraphCL that are not designed specifically for handling the imbalance scenario since the extra supervision derived in our model stays the same while the basic supervision encoded in the training data decreases. Besides, our model also performs comparable or even slightly better then all other baselines under balanced scenario, which additionally signifies the potentiality of our model in balanced data-splitting. Among other baselines, GraphCL$_{rw}$ performs the best since it applies re-weight strategy to balance the training loss and further leverages the graph augmentation coupled with mutual information maximization to extract the most relevant information for downstream classification. An interesting observation is that the optimal performance is not always when the labeled data is strictly balanced, which reflects the uneven distribution of informatic supervision embedded across different classes.

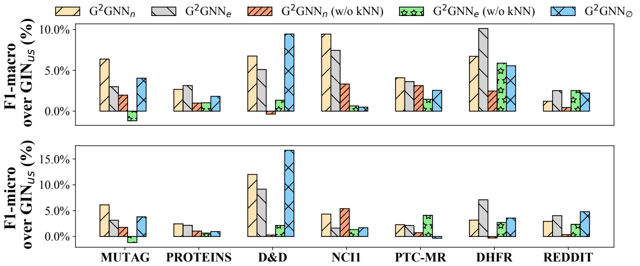
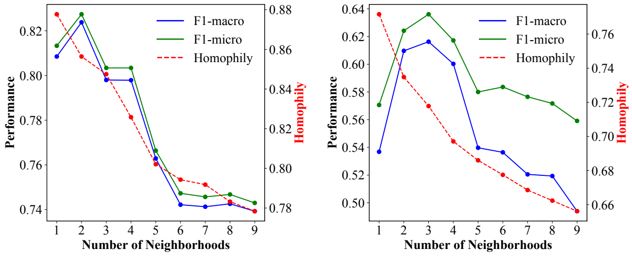


Figure 4: Ablation study results of G^{2}GNN where we report the improvement over GIN$_{us}$ due to its simplicity and effectiveness (seen in Table 2) for understanding relative improvements of each G^{2}GNN component.

5.4 Ablation Study

In this section, we conduct ablation study to fully understand the effect of each component in G^{2}GNN on alleviating the imbalance issue. In Figure 4, we present performance improvement over the baseline GIN$_{us}$ achieved by our proposed framework (**G^{2}GNN$_{e(n)}$**) along with variants that remove the GoG propagation (**G^{2}GNN$_{e(n)}$ (w/o kNN)**) and remove the topological augmentation (**G^{2}GNN$_\emptyset$**). **(1)** We notice that solely employing GoG propagation (**G^{2}GNN$_\emptyset$**) increases the performance on all datasets according to F1-macro, demonstrating the effectiveness of GoG propagation in alleviating imbalance issue. **(2)** Augmenting via removing edges hurts the performance on MUTAG. This is because the size of each graph in MUTAG is relatively small and thus removing edges may undermine crucial topological information related to downstream classification. **(3)** We observe that the proposed GoG propagation and graph augmentation generally achieve more performance boost on F1-macro than F1-micro. This is because the derived supervision significantly enhance the generalizability of training data in minority classes. However, for majority classes where majority training instances already guarantee high generalizability, the enhancement would be minor. **(4)** Combining GoG propagation and



(a) MUTAG **(b) DHFR**

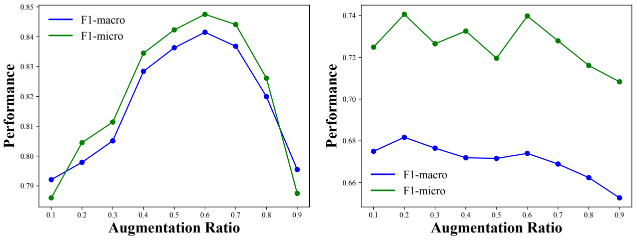
Figure 5: Relationship between neighborhood number, edge homophily, and performance on MUTAG and DHFR. The performance first increases and then decreases as the number of neighborhoods increases on $\mathcal{G}^{\text{kNN}}$. The reported result here is averaged over 20 runs.

graph augmentation together is better than only applying one of them in most cases, which indicates that the extra supervision derived by globally borrowing neighboring information and locally augmenting graphs are both beneficial to downstream tasks and not overlapped with each other as the accumulating benefit shown here. **(5)** On NCI1, despite the minor improvement of applying only one of the proposed two modules, combining them together leads to significant progress. This is because instead of propagating original graphs' representations, we leverage augmented graphs in GoG propagation and the derived local supervision is further enhanced by the global propagation to create more novel supervision and extremely enhance the model generalibility on minority classes.

5.5 Further Probe

5.5.1 Effect of Neighborhood Numbers. Here we investigate the influence of the number of neighboring graphs on the performance of G^{2}GNN$_n$ on MUTAG and DHFR. The experimental setting is the same as Section 5.1 except that we alter the k among $\{1, 2, ..., 9\}$. In Figure 5, we see that both of the F1-macro and F1-micro increase first as k increases to 2 on MUTAG and 3 on DHFR since higher k means more number of neighboring graphs sharing the same label, as the homophily level at this stage is generally higher given the red line, therefore we derive more beneficial supervision. However, as we further increases k to 6, the performance begins to decrease since most of added neighborhoods share different labels due to low homophily in this middle stage and hence provide adverse information that compromises classification. In the last stage when k proceeds to increase beyond 6, the performance gradually becomes stable, this is because directly linking each graph with its 6-top similar graphs leads to a very dense GoG and propagation on this dense GoG directly incorporates information from most of other graphs and therefore the neighboring information that each graph receives is too noisy and useless.

5.5.2 Effect of augmentation ratio. Then we investigate the effect of augmentation ratio δ among $\{0.1, 0.2, ..., 0.9\}$ on the performance of G^{2}GNN$_n$ on MUTAG and G^{2}GNN$_e$ on PROTEINS. We see that the F1-macro on both MUTAG and PROTEINS first increase and then decrease. This is because initially increasing augmentation ratio would generate abundant unseen graphs and enhance the model generalibility, which conforms to the advantageous of harder contrastive learning concluded in [44]. However, as we further increase the augmentation ratio, the performance decreases because graphs of one class maybe over-augmented, which destroys the latent relationship between graphs and its class or even mismatch graphs with other classes.



(a) MUTAG **(b) PROTEINS**

Figure 6: Relationship between augmentation ratio δ and performance on MUTAG and PROTEINS. The performance first increases and then decreases as augmentation ratio increases. The reported result here is averaged over 20 runs.

Table 3: Running time (in seconds) of different models.

Dataset	GIN	GIN$_{\text{us}}$	GIN$_{\text{rw}}$	G^{2}GNN$_\emptyset$	G^{2}GNN$_e$	G^{2}GNN$_n$
MUTAG	5.2	8.9	5.6	16.8	24.4	22.6
PROTEINS	24.3	40.7	25.3	111.1	155.7	153.0

5.5.3 Efficient Analysis. Furthermore, we compare the efficiency of each model in Table 3 where the running time is averaged across 10 times. Without equipping any imbalance-tailored operation, GIN achieves shortest running time. Equipping reweighting as GIN$_{\text{rw}}$ is faster than equipping upsampling as GIN$_{\text{us}}$ since upsampling increase the size of the dataset. Our proposed G^{2}GNN and its variants generally have longer running time due to topological augmentation and graph-level propagation.

6 CONCLUSION

In this paper, we focused on imbalanced graph classification, which widely exists in the real world while rarely explored in the literature. Noticing that unlike the node imbalance problem where we can propagate neighboring nodes' information to obtain extra supervision, graphs are isolated and have no connections with each other. Therefore, we employ a kernel-based Graph of Graph (GoG) construction to establish a kNN graph and devise a two-level propagation to derive extra supervision from neighboring graphs globally. By theoretically proving the feature smoothing is upper bounded by the label smoothing and empirically showing the high homophily on the constructed kNN GoG, we guarantee the derived supervision is beneficial for downstream classification. Moreover, we employ local augmentation and upsampling of minority graphs to enhance the model generalizability in discerning unseen non-training (especially minority) graphs. Experiments on 7 real-world datasets demonstrate the effectiveness of G^{2}GNN in relieving the graph imbalance issue. For future work, we plan to incorporate attention mechanism in the GoG propagation to adaptively aggregate neighboring graphs' information based on their topological similarity and further more work on the imbalance problem in link prediction, especially in recommendation systems [40].

REFERENCES

[1] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. 2019. Mixmatch: A holistic approach to semi-supervised learning. *arXiv preprint arXiv:1905.02249* (2019).

[2] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *ICDM*.

[3] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *JAIR* 16 (2002), 321–357.

[4] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. 2004. Special issue on learning from imbalanced data sets. *SIGKDD Explorations* 6, 1 (2004).

[5] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *J. Med. Chem.* 34, 2 (1991), 786–797.

[6] Kaize Ding, Jianling Wang, Jundong Li, Kai Shu, Chenghao Liu, and Huan Liu. 2020. Graph prototypical networks for few-shot learning on attributed networks. In *CIKM*. 295–304.

[7] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. 2022. Data augmentation for deep graph learning: A survey. *arXiv preprint arXiv:2202.08235* (2022).

[8] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. 2020. A Fair Comparison of Graph Neural Networks for Graph Classification. In *ICLR*.

[9] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. 2021. A survey of data augmentation approaches for nlp. *arXiv:2105.03075* (2021).

[10] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. 2020. Graph Random Neural Network for Semi-Supervised Learning on Graphs. *arXiv:2005.11079* (2020).

[11] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[12] Yves Grandvalet and Yoshua Bengio. 2004. Semi-supervised learning by entropy minimization. *Advances in neural information processing systems* 17 (2004).

[13] Shawn Gu, Meng Jiang, Pietro Hiram Guzzi, and Tijana Milenkovic. 2021. Modeling multi-scale data via a network of networks. *arXiv:2105.12226* (2021).

[14] Gabriel Idakwo, Sundar Thangapandian, Joseph Luttrell, Yan Li, Nan Wang, Zhaoxian Zhou, Huixiao Hong, Bei Yang, Chaoyang Zhang, and Ping Gong. 2020. Structure–activity relationship-based chemical classification of highly imbalanced Tox21 datasets. *Journal of cheminformatics* 12, 1 (2020), 1–19.

[15] Justin M Johnson and Taghi M Khoshgoftaar. 2019. Survey on deep learning with class imbalance. *Journal of Big Data* 6, 1 (2019), 1–54.

[16] Jonatan Kilhamn. 2015. *Fast shortest-path kernel computations using aproximate methods*. Master's thesis.

[17] Miroslav Kubat, Stan Matwin, et al. 1997. Addressing the curse of imbalanced training sets: one-sided selection. In *Icml*, Vol. 97. Citeseer, 179.

[18] Joffrey L Leevy, Taghi M Khoshgoftaar, Richard A Bauder, and Naeem Seliya. 2018. A survey on addressing high-class imbalance in big data. *Journal of Big Data* 5, 1 (2018), 1–30.

[19] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.

[20] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 338–348.

[21] Yunchao Liu, Yu Wang, Oanh T Vu, Rocco Moretti, Bobby Bodenheimer, Jens Meiler, and Tyler Derr. 2022. Interpretable Chirality-Aware Graph Neural Network for Quantitative Structure Activity Relationship Modeling in Drug Discovery. *bioRxiv* (2022).

[22] Jingchao Ni, Hanghang Tong, Wei Fan, and Xiang Zhang. 2014. Inside the atoms: ranking on a network of networks. In *KDD*. 1356–1365.

[23] Shirui Pan and Xingquan Zhu. 2013. Graph classification with imbalanced class distributions and noise. In *IJCAI*.

[24] Liang Qu, Huaisheng Zhu, Ruiqi Zheng, Yuhui Shi, and Hongzhi Yin. 2021. ImGAGN: Imbalanced Network Embedding via Generative Adversarial Graph Networks. *arXiv:2106.02817* (2021).

[25] Benedek Rozemberczki, Charles Tapley Hoyt, Anna Gogleva, Piotr Grabowski, Klas Karis, Andrej Lamov, Andriy Nikolov, Sebastian Nilsson, Michael Ughetto, Yu Wang, et al. 2022. ChemicalX: A Deep Learning Library for Drug Pair Scoring. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3819–3828.

[26] Min Shi, Yufei Tang, Xingquan Zhu, David Wilson, and Jianxun Liu. 2020. Multi-class imbalanced graph convolutional network learning. In *IJCAI*.

[27] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data* 6, 1 (2019), 1–48.

[28] Zixing Song, Xiangli Yang, Zenglin Xu, and Irwin King. 2021. Graph-based Semi-supervised Learning: A Comprehensive Review. *arXiv:2102.13303* (2021).

[29] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. [n.d.]. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization. In *ICLR 2020*.

[30] Jeffrey J Sutherland, Lee A O'brien, and Donald F Weaver. 2003. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *J Chem Inform Comput Sci* 43, 6 (2003).

[31] Nguyen Thai-Nghe, Zeno Gantner, and Lars Schmidt-Thieme. 2010. Cost-sensitive learning methods for imbalanced data. In *IJCNN*. IEEE.

[32] Hannu Toivonen, Ashwin Srinivasan, Ross D King, Stefan Kramer, and Christoph Helma. 2003. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics* 19, 10 (2003), 1183–1193.

[33] Jason Van Hulse, Taghi M Khoshgoftaar, and Amri Napolitano. 2007. Experimental perspectives on learning from imbalanced data. In *ICML*. 935–942.

[34] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. 2010. Graph kernels. *Journal of Machine Learning Research* 11 (2010), 1201–1242.

[35] Nikil Wale, Ian A Watson, and George Karypis. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *KAIS* 14, 3 (2008), 347–375.

[36] Hongwei Wang and Jure Leskovec. 2020. Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755* (2020).

[37] Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, and Xuemin Lin. 2020. Gognn: Graph of graphs neural network for predicting structured entity interactions. *arXiv:2005.05537* (2020).

[38] Yu Wang, Charu Aggarwal, and Tyler Derr. 2021. Distance-wise Prototypical Graph Neural Network in Node Imbalance Classification. *arXiv preprint arXiv:2110.12035* (2021).

[39] Yu Wang, Wei Jin, and Tyler Derr. 2022. Graph neural networks: Self-supervised learning. In *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer, 391–420.

[40] Yu Wang, Yuying Zhao, Yi Zhang, and Tyler Derr. 2022. Collaboration-Aware Graph Convolutional Networks for Recommendation Systems. *arXiv preprint arXiv:2207.06221* (2022).

[41] Zheng Wang, Xiaojun Ye, Chaokun Wang, Jian Cui, and Philip Yu. 2020. Network embedding with completely-imbalanced labels. *IEEE TKDE* (2020).

[42] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.

[43] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *KDD*. 1365–1374.

[44] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph Contrastive Learning with Augmentations. In *NeurIPS*.

[45] Jin-Zhu Yu, Mincheng Wu, Gisela Bichler, Felipe Aros-Vera, and Jianxi Gao. 2022. Reconstructing Sparse Illicit Supply Networks: A Case Study of Multiplex Drug Trafficking Networks. *arXiv preprint arXiv:2208.01739* (2022).

[46] Bo Yuan and Xiaoli Ma. 2012. Sampling+ reweighting: Boosting the performance of AdaBoost on imbalanced datasets. In *The 2012 international joint conference on neural networks (IJCNN)*. IEEE, 1–6.

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

[48] Tong Zhao, Tianwen Jiang, Neil Shah, and Meng Jiang. 2021. A Synergistic Approach for Graph Anomaly Detection With Pattern Mining and Feature Learning. *IEEE TNNLS* (2021).

[49] Tong Zhao, Gang Liu, Stephan Günnemann, and Meng Jiang. 2022. Graph Data Augmentation for Graph Machine Learning: A Survey. *arXiv preprint arXiv:2202.08871* (2022).

[50] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2021. Data augmentation for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11015–11023.

[51] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2021. GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks. In *WSDM*.

[52] Shuangjia Zheng, Zengrong Lei, Haitao Ai, Hongming Chen, Daiguo Deng, and Yuedong Yang. 2021. Deep scaffold hopping with multimodal transformer neural networks. *Journal of cheminformatics* 13, 1 (2021), 1–15.