AutoGCL: Automated Graph Contrastive Learning via Learnable View Generators

Yihang Yin[1], Qingzhong Wang[2], Siyu Huang[3], Haoyi Xiong[2], Xiang Zhang[4]
[1]Nanyang Technological University
[2]Baidu Research
[3]Harvard University
[4]The Pennsylvania State University
yyin009@e.ntu.edu.sg, wangqingzhong@baidu.com, huang@seas.harvard.edu, xionghaoyi@baidu.com, xzz89@psu.edu

Abstract

Contrastive learning has been widely applied to graph representation learning, where the view generators play a vital role in generating effective contrastive samples. Most of the existing contrastive learning methods employ pre-defined view generation methods, *e.g.*, node drop or edge perturbation, which usually cannot adapt to input data or preserve the original semantic structures well. To address this issue, we propose a novel framework named *Automated Graph Contrastive Learning* (AutoGCL) in this paper. Specifically, AutoGCL employs a set of learnable graph view generators orchestrated by an auto augmentation strategy, where every graph view generator learns a probability distribution of graphs conditioned by the input. While the graph view generators in AutoGCL preserve the most representative structures of the original graph in generation of every contrastive sample, the auto augmentation learns policies to introduce adequate augmentation variances in the whole contrastive learning procedure. Furthermore, AutoGCL adopts a joint training strategy to train the learnable view generators, the graph encoder, and the classifier in an end-to-end manner, resulting in topological heterogeneity yet semantic similarity in the generation of contrastive samples. Extensive experiments on semi-supervised learning, unsupervised learning, and transfer learning demonstrate the superiority of our AutoGCL framework over the state-of-the-arts in graph contrastive learning. In addition, the visualization results further confirm that the learnable view generators can deliver more compact and semantically meaningful contrastive samples compared against the existing view generation methods. Our code is available at https://github.com/Somedaywilldo/AutoGCL.

Introduction

Graph neural networks (GNNs) (Kipf and Welling 2016a; Veličković et al. 2017; Hamilton, Ying, and Leskovec 2017; Xu et al. 2018) are gaining increasing attention in the realm of graph representation learning. By generally following a recursive neighborhood aggregation scheme, GNNs have shown impressive representational power in various domains, such as point clouds (Shi and Rajkumar 2020), social networks (Fan et al. 2019), chemical analysis (De Cao and Kipf 2018), and so on. Most of the existing GNN models are trained in an end-to-end supervised fashion, which relies on a high volume of fine-annotated data. However, labeling graph data requests a huge amount of effort from professional annotators with domain knowledge. To alleviate this issue, GAE (Kipf and Welling 2016b) and GraphSAGE (Hamilton, Ying, and Leskovec 2017) have been proposed to exploit a naive unsupervised pretraining strategy that reconstructs the vertex adjacency information. Some recent works (Hu et al. 2019; You et al. 2020b) introduce self-supervised pretraining strategies to GNNs which further improve the generalization performance.

More recently, with developments of contrastive multi-view learning in computer vision (Tian, Krishnan, and Isola 2019; He et al. 2020; Chen et al. 2020a) and natural language processing (Logeswaran and Lee 2018; Yang et al. 2019), some self-supervised pretraining approaches perform as good as (or even better than) supervised methods. In general, contrastive methods generate training views using data augmentations, where views of the same (positive pairs) input are concentrated in the representation space with views of different inputs (negative pairs) pushed apart. To work on graphs, DGI (Veličković et al. 2018) has been proposed to treat both graph-level and node-level representations of the same graph as positive pairs, pursuing consistent representations from local and global features. CMRLG (Hassani and Khasahmadi 2020) achieves a similar goal by grouping adjacency matrix (local features) and its diffusion matrix (global features) as positive pairs. GCA (Zhu et al. 2020b) generates the positive view pairs through sub-graph sampling with the structure priors with node attributes randomly masked. GraphCL (You et al. 2020a) offers even more strategies for augmentations, such as node dropping and edge perturbation. While above attempts incorporate contrastive learning into graphs, they usually fail to generate views with respect to the semantic of original graphs or adapt augmentation policies to specific graph learning tasks.

Blessed by the invariance of image semantics under various transformation, image data augmentation has been widely used (Cubuk et al. 2019) to generative contrastive views. However, the use of graph data augmentation might be ineffective here, as transformations on a graph might severely disrupt its semantics and properties for learning. In the meanwhile, InfoMin (Tian et al. 2020) improves contrastive learning for vision tasks and proposes to replace image data augmentation with a flow-based generative model

Property	CMRLG	GRACE	GraphCL	GCA	JOAO	AD-GCL	Ours
Topological	✓	✓	✓	✓	✓	✓	✓
Node Feature	-	✓	✓	✓	✓	-	✓
Label-preserving	-	-	-	-	-	-	✓
Adaptive	-	-	-	✓	✓	✓	✓
Variance	-	✓	✓	✓	✓	✓	✓
Differentiable	-	-	-	-	-	✓	✓
Efficient BP	-	-	-	-	-	✓	✓

Table 1: An overview of graph augmentation methods.

for contrastive views generation. Thus, learning a probability distribution of contrastive views conditioned by an input graph might be an alternative to simple data augmentation for graph contrastive learning but still requests non-trivial efforts, as the performance and scalability of common graph generative models are poor in real-world scenarios.

In this work, we propose a learnable graph view generation method, namely AutoGCL, to address above issues via learning a probability distribution over node-level augmentations. While the conventional pre-defined view generation methods, such as random dropout or graph node masking, may inevitably change the semantic labels of graphs and finally hurt contrastive learning, AutoGCL adapts to the input graph such that it can well preserve the semantic labels of the graph. In addition, thanks to the gumbel-softmax trick (Jang, Gu, and Poole 2016), AutoGCL is end-to-end differentiable yet providing sufficient variances for contrastive samples generation. We further propose a joint training strategy to train the learnable view generators, the graph encoders, and the classifier in an end-to-end manner. The strategy includes the view similarity loss, the contrastive loss, and the classification loss. It makes the proposed view generators generate augmented graphs that have similar semantic information but with different topological properties. In Table 1, we summarize the properties of existing graph augmentation methods, where AutoGCL dominates in the comparisons.

We conduct extensive graph classification experiments using semi-supervised learning, unsupervised learning, and transfer learning tasks to evaluate the effectiveness of AutoGCL. The results show that AutoGCL improves the state-of-the-art graph contrastive learning performances on most of the datasets. In addition, we visualize the generated graphs on MNIST-Superpixel dataset (Monti et al. 2017) and reveal that AutoGCL could better preserve semantic structures of the input data than existing pre-defined view generators. Our contributions can be summarized as follows.

- We propose a graph contrastive learning framework with learnable graph view generators embedded into an auto augmentation strategy. To the best of our knowledge, this is the first work to build learnable generative node-wise augmentation policies for graph contrastive learning.
- We propose a joint training strategy for training the graph view generators, the graph encoder, and the graph classifier under the context of graph contrastive learning in an end-to-end manner.
- We extensively evaluate the proposed method on a variety of graph classification datasets with semi-supervised, unsupervised, and transfer learning settings. The t-SNE and view visualization results also demonstrate the effectiveness of our method.

Related Work

Graph Neural Networks

Denote a graph as $g = (V, E)$ where the node features are $\boldsymbol{x}_v$ for $v \in V$. In this paper, we focus on the graph classification task using Graph Neural Networks (GNNs). GNNs generate node-level embedding $\boldsymbol{h}_v$ through aggregating the node features $\boldsymbol{x}_v$ of its neighbors. Each layer of GNNs serves as an iteration of aggregation, such that the node embedding after the k-th layers aggregates the information within its k-hop neighborhood. The k-th layer of GNNs can be formulated as

$$\boldsymbol{a}_v^{(k)} = \text{AGGREGATE}^{(k)}(\{\boldsymbol{h}_u^{(k-1)} : u \in \mathcal{N}(v)\}) \quad (1)$$

$$\boldsymbol{h}_v^{(k)} = \text{COMBINE}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) \quad (2)$$

For the downstream tasks such as graph classification, the graph-level representation $\boldsymbol{z}_g$ is obtained via the READOUT function and MLP layers as

$$F(g) = \text{READOUT}(\{\boldsymbol{h}_n^{(k)} : v_n \in \mathcal{V}\}) \quad (3)$$

$$\boldsymbol{z}_g = \text{MLP}(F(g)) \quad (4)$$

In this work we follow the existing graph contrastive learning literature to employ two state-of-the-art GNNs, *i.e.*, GIN (Xu et al. 2018) and ResGCN (Chen, Bian, and Sun 2019), as our backbone GNNs.

Pre-training Graph Neural Networks

Pre-training GNNs on graph datasets still remains a challenging task, since the semantics of graphs are not straightforward, and the annotation of graphs (proteins, chemicals, etc.) usually requires professional domain knowledge. It is very costly to collect large-scale and fine-annotated graph datasets like ImageNet (2012). An alternative way is to pre-train the GNNs in an unsupervised manner. The GAE (2016b) first explored the unsupervised GNN pre-training by reconstructing the graph topological structure. GraphSAGE (2017) proposed an inductive way of unsupervised node embedding by learning the neighborhood aggregation function. Pretrain-GNN (Hu et al. 2019) conducted the first systematic large-scale investigation of strategies for pre-training GNNs under the transfer learning setting. It proposed self-supervised pre-training strategies to learn both the local and global features of graphs. However, the benefits of graph transfer learning may be limited and lead to negative transfer (Rosenstein et al. 2005), as the graphs from different domains actually differ a lot in their structures, scales and node/edge attributes. Therefore, many of the following works started to explore an alternative approach, *i.e.*, the contrastive learning, for GNNs pre-training.

Contrastive Learning

In recent years, contrastive learning (CL) has received considerable attention among the self-supervised learning approaches, and a series of CL methods including SimCLR (Chen et al. 2020a) and MoCo-v2 (Chen et al. 2020b) even outperform supervised baselines. Through minimizing the contrastive loss (2006), the views generated from the same input (*i.e.*, positive pairs) are pulled close in the representation space, while the views of different inputs (*i.e.*, negative pairs) are pushed apart. Most of the existing CL methods (He et al. 2020; Zbontar et al. 2021; Chen et al. 2020a;

Grill et al. 2020) generate views using data augmentation, which is still challenging and under-explored for the graph data. Instead of data augmentation, DGI (Veličković et al. 2018) treated the graph-level and node-level representations of the same graph as positive view pairs. CMRLG (Hassani and Khasahmadi 2020) achieved an analogical goal by treating the adjacency matrix and the diffusion matrix as positive pairs. More recently, GraphCL (You et al. 2020a) employed four types of graph augmentations, including node dropping, edge perturbation, sub-graph sampling, and node attribute masking, enabling the most diverse augmentations by far for graph view generation. GCA (Zhu et al. 2020b) used sub-graph sampling and node attribute masking as augmentations and introduced a prior augmentation probability based on the node centrality measures, enabling more adaptiveness than GraphCL (You et al. 2020a), but the prior is not learnable.

Learnable Data Augmentation

As mentioned above, data augmentation is a significant component of CL. The existing literature (Chen et al. 2020a; You et al. 2020a) has revealed that the optimal augmentation policies are task-dependent and the choice of augmentations makes a considerable difference to the CL performance. The researchers have explored to automatically discover the optimal policy for image augmentations in the computer vision field. For instance, AutoAugment (Cubuk et al. 2019) firstly optimized the combination of augmentation functions through reinforcement learning. Faster-AA (Hataya et al. 2020) and DADA (Li et al. 2020) proposed a differentiable augmentation optimization framework following the DARTS (Liu, Simonyan, and Yang 2018) style.

However, the learnable data augmentation methods are barely explored for CL until the InfoMin framework (Tian et al. 2020), which claims that good views of CL should maintain the label information as well as minimizing the mutual information of positive view pairs. InfoMin employs a flow-based generative model as the view generator for data augmentation and trains the view generator in a semi-supervised manner. However, transferring this idea to graph is a non-trivial task since current graph generative models are either of limited generation qualities (Kipf and Welling 2016b) or designed for specific tasks such as the molecular data (De Cao and Kipf 2018; Madhawa et al. 2019; Wang et al. 2021). To make graph augmentations adaptive to different tasks, JOAO (You et al. 2021) learns the sampling distribution of pre-defined augmentations in a Bayesian manner, but the augmentations themselves are still not learnable. AD-GCL (Suresh et al. 2021) first proposed a learnable edge dropping augmentation and employs adversarial training strategy, but node-level augmentations are not considered, and the strategy will not ensure to generate label-preserving augmentations.

In this work we build a learnable graph view generator that learns a probability distribution over the node-level augmentations. Compared to the existing graph CL methods, our method well preserves the semantic structures of original graphs. Moreover, it is end-to-end differentiable and can be efficiently trained.

Methodology

What Makes a Good Graph View Generator?

Our goal is to design a learnable graph view generator that learns to generate the augmented graph view in data-driven manner. Although various graph data augmentation methods have been proposed, there is less discussion on what makes a good graph view generator? From our perspective, an ideal graph view generator for data augmentation and contrastive learning should satisfy the following properties: (1) It supports both the augmentations of the graph **topology** and the **node feature**. (2) It is **label-preserving**, *i.e.*, the augmented graph should maintain the semantic information in the original graph. (3) It is **adaptive** to different data distributions and scalable to large graphs. (4) It provides **sufficient variances** for contrastive multi-view pre-training. (5) It is **end-to-end differentiable** and **efficient** enough for fast gradient computation via **back-propagation (BP)**.

Here we provide an overview of the augmentation methods proposed in existing literature of graph contrastive learning in Table 1. CMRLG (Hassani and Khasahmadi 2020) applies diffusion kernel to get different topological structures. GRACE (Zhu et al. 2020a) uses random edge dropping and node attribute masking. GCA (Zhu et al. 2020b) uses node dropping and node attribute masking along with a structural prior. GraphCL (You et al. 2020a) proposes the most flexible set of graph data augmentations so far, including node dropping, edge perturbation, sub-graph, and attribute masking. We provide a detailed ablation study and analysis of GraphCL in the supplementary. JOAO (You et al. 2021) optimizes the augmentation sampling policy of GraphCL in a Bayesian manner. AD-GCL (Suresh et al. 2021) designs a learnable edge dropping augmentation.

In this work, we propose a learnable view generator to address all the above issues. Our view generator includes both augmentations of node dropping and attribute masking, but it is much more flexible since both two augmentations can be simultaneously employed in a node-wise manner, without the need of tuning the *"aug ratio"*. Besides the concern of model performance, another reason for not incorporating edge perturbation in our view generator is, the generation of edges through the learnable methods (*e.g.*, VGAE (Kipf and Welling 2016b)) requires to predict the full adjacency matrix that contains $O(N^2)$ elements, which is a heavy burden for back-propagation when dealing with large-scale graphs.

Learnable Graph View Generator

Fig. 1 illustrates the scheme of our proposed learnable graph view generator. We use GIN (Xu et al. 2018) layers to get the node embedding from the node attribute. For each node, we use the embedded node feature to predict the probability of selecting a certain augment operation. The augmentation pool for each node is drop, keep, and mask. We employ the gumbel-softamx (Jang, Gu, and Poole 2016) to sample from these probabilities then assign an augmentation operation to each node. Formally, if we use k GIN layers as the embedding layer, we denote $\boldsymbol{h}_v^{(k)}$ as the hidden state of node v at the k-th layer and $\boldsymbol{a}_v^{(k)}$ as the embedding of node v after the k-th layer. For node v, we have the node feature $\boldsymbol{x}_v$, the

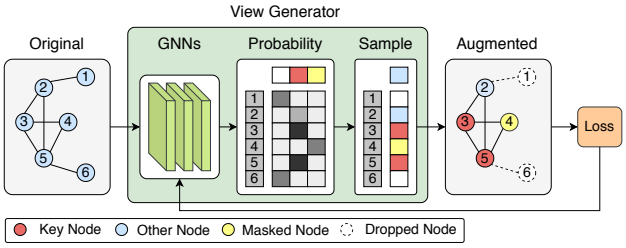

Figure 1: The architecture of our learnable graph view generator. The GNN layers embed the original graph to generate a distribution for each node. The augmentation choice of each node is sampled from it using the gumbel-softmax.

augmentation choice f_v, and the function $\text{Aug}(\boldsymbol{x}, f)$ for applying the augmentation. Then the augmented feature $\boldsymbol{x}_v^{'}$ of node v is obtained via

$$\boldsymbol{h}_v^{(k-1)} = \text{COMBINE}^{(k)}(\boldsymbol{h}_v^{(k-2)}, \boldsymbol{a}_v^{(k-1)}) \tag{5}$$

$$\boldsymbol{a}_v^{(k)} = \text{AGGREGATE}^{(k)}(\{\boldsymbol{h}_u^{(k-1)} : u \in \mathcal{N}(v)\}) \tag{6}$$

$$f_v = \text{GumbelSoftmax}(\boldsymbol{a}_v^{(k)}) \tag{7}$$

$$\boldsymbol{x}_v^{'} = \text{Aug}(\boldsymbol{x}_v, f_v) \tag{8}$$

The dimension of the last layer k is set as the same number of possible augmentations for each node. $\boldsymbol{a}_v^{(k)}$ denotes the probability distribution for selecting each kind of augmentation. f_v is a one-hot vector sampled from this distribution via gumbel-softmax and it is differentiable due to the reparameterization trick. The augmentation applying function $\text{Aug}(\boldsymbol{x}_v, f_v)$ combines the node attribute $\boldsymbol{x}_v$ and f_v using differentiable operations (*e.g.* multiplication), so the gradients of the weights of the view generator are kept in the augmented node features and can be computed using back-propagation. For the augmented graph, the edge table is updated using f_v for all $v \in V$, where the edges connected to any dropped nodes are removed. As the edge table is only the guidance for node feature aggregation and it does not participate in the gradient computation, it does not need to be updated in a differentiable manner. Therefore, our view generator is end-to-end differentiable. The GIN embedding layers and the gumbel-softmax can be efficiently scaled up for larger graph datasets and more augmentation choices.

Contrastive Pre-training Strategy

Since the contrastive learning requires multiple views to form a positive view pair, we have two view generators and one classifier for our framework. According to InfoMin principle (Tian et al. 2020), a good positive view pair for contrastive learning should maximize the label-related information as well as minimizing the mutual information (similarity) between them. To achieve that, our framework uses two separate graph view generators and trains them and the classifier in a joint manner.

Loss Function Definition Here we define three loss functions, contrastive loss $\mathcal{L}_{\text{cl}}$, similarity loss $\mathcal{L}_{\text{sim}}$, and classification loss $\mathcal{L}_{\text{cls}}$. For contrastive loss, we follow the previous works (Chen et al. 2020a; You et al. 2020a) and use the normalized temperature-scaled cross entropy loss (NT-XEnt) (Sohn 2016). Define the similarity function $\text{sim}(\boldsymbol{z}_1, \boldsymbol{z}_2)$ as

$$\text{sim}(\boldsymbol{z}_1, \boldsymbol{z}_2) = \frac{\boldsymbol{z}_1 \cdot \boldsymbol{z}_2}{\|\boldsymbol{z}_1\|_2 \cdot \|\boldsymbol{z}_2\|_2} \tag{9}$$

Suppose we have a data batch made up of N graphs. We pass the batch to the two view generators to obtain $2N$ graph views. We regard the two augmented views from the same input graph as the positive view pair. We use $\mathbb{1}_{[k \neq i]} \in \{0, 1\}$ to denote the indicator function. We denote the contrastive loss function for a positive pair of samples (i, j) as $\ell(i, j)$, the contrastive loss of this data batch as $\mathcal{L}_{\text{cl}}$, the temperature parameter as τ, then we have

$$\ell_{(i,j)} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)} \tag{10}$$

$$\mathcal{L}_{\text{cl}} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)] \tag{11}$$

The similarity loss is used to minimize the mutual information between the views generated by the two view generators. During the view generation process, we have a sampled state matrix S indicting each node's corresponding augmentation operation (see Fig. 1). For a graph G, we denote the sampled augmentation choice matrix of each view generator as A_1, A_2, then we formulate the similarity loss $\mathcal{L}_{\text{sim}}$ as

$$\mathcal{L}_{\text{sim}} = \text{sim}(A_1, A_2) \tag{12}$$

Finally, for the classification loss, we directly use the cross entropy loss (ℓ_{cls}). For a graph sample g with class label y, we denote the augmented view as g_1 and g_2 and the classifier as F. Then the classification loss $\mathcal{L}_{\text{cls}}$ is formulated as

$$\mathcal{L}_{\text{cls}} = \ell_{\text{cls}}(F(g), y) + \ell_{\text{cls}}(F(g_1), y) + \ell_{\text{cls}}(F(g_2), y) \tag{13}$$

$\mathcal{L}_{\text{cls}}$ is employed in the semi-supervised pre-training task to encourage the view generator to generate label-preserving augmentations.

Naive Training Strategy For unsupervised learning and transfer learning tasks, we use a naive training strategy (naive-strategy). Since we do not know the label in the pre-training stage, the $\mathcal{L}_{\text{sim}}$ is not used because it does not make sense to just encourage the views to be different without keeping the label-related information. This could lead to generating useless or even harmful view samples. We just train the view generators and the classifier jointly to minimize the $\mathcal{L}_{\text{cl}}$ in the pre-training stage.

Also, we note that the quality of the generated views will not be as good as the original data. During the $\mathcal{L}_{\text{cl}}$ minimization, instead of just minimizing the $\mathcal{L}_{\text{cl}}$ between two augmented views like GraphCL (You et al. 2020a), we also make use of the original data. By pulling the original data and the augmented views close in the embedding space, the view generator are encouraged to preserve the label-related information. The details are described in Algorithm 1.

Joint Training Strategy For semi-supervised learning tasks, we proposed a joint training strategy, performs contrastive training and supervised training alternately. This strategy generates label-preserving augmentation and outperforms the naive-strategy. During the unsupervised training stage, we fix the view generators, and train the classifer

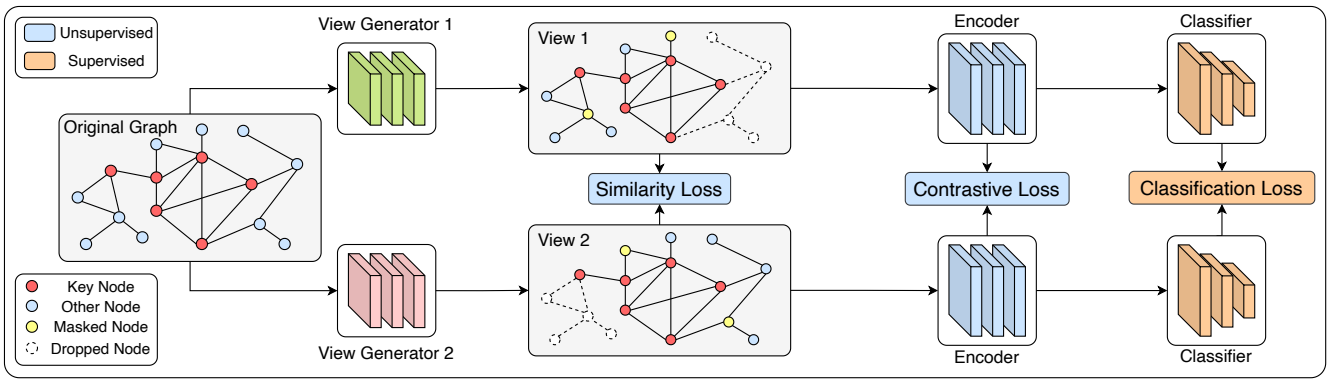

Figure 2: The proposed AutoGCL framework is composed of three parts: (1) two *view generators* that generate different views of the original graph, (2) a *graph encoder* that extracts the features of graphs and (3) a *classifier* that provides the graph outputs.

Algorithm 1: Naive training strategy (naive-strategy).

Initialize weights of G_1, G_2, F.
while not reached maximum epochs **do**
  for mini-batch x from unlabeled data **do**
    Get augmentation $x_1 = G_1(x), x_2 = G_2(x)$
    Sample two views v_1, v_2 from $\{x, x_1, x_2\}$
    $\mathcal{L} = \mathcal{L}_{\text{cl}}(v_1, v_2)$
    Update the weights of G_1, G_2, F to minimize $\mathcal{L}$
while not reached maximum epochs **do**
  for mini-batch x from labeled data **do**
    $\mathcal{L} = \mathcal{L}_{\text{cls}}(x)$
    Update the weights of F to minimize $\mathcal{L}$

Algorithm 2: Joint training strategy (joint-strategy).

Initialize weights of G_1, G_2, F.
while not reached maximum epochs **do**
  for mini-batch x from unlabeled data **do**
    Fix the weights of G_1, G_2
    Get augmentation $x_1 = G_1(x), x_2 = G_2(x)$
    Sample two views v_1, v_2 from $\{x, x_1, x_2\}$
    $\mathcal{L} = \mathcal{L}_{\text{cl}}(v_1, v_2)$
    Update the weights of F to minimize $\mathcal{L}$
  for mini-batch x from labeled data **do**
    Get augmentation $x_1 = G_1(x), x_2 = G_2(x)$
    $\mathcal{L} = \mathcal{L}_{\text{cls}}(x, x_1, x_2) + \lambda \cdot \mathcal{L}_{\text{sim}}(x_1, x_2)$
    Update the weights of G_1, G_2, F to minimize $\mathcal{L}$

by contrastive learning using unlabeled data. During the supervised training stage, we jointly train the view generator with the classifier using labeled data. By simultaneously optimizing $\mathcal{L}_{\text{sim}}$ and $\mathcal{L}_{\text{cls}}$, the two view generators are encouraged to generated label-preserving augmentations, yet being different enough from each other. The unsupervised training stage and supervised training stage are repeated alternately. This is very different from previous graph contrastive learning methods. Previous work like GraphCL (2020a) use the pre-training/fine-tuning strategy, which first minimizes the contrastive loss ($\mathcal{L}_{\text{cl}}$) until convergence using the unlabeled data and then fine-tunes it with the labeled data.

However, we found that for graph contrastive learning, the pre-training/fine-tuning strategy are more likely to cause over-fitting in the fine-tuning stage. And minimizing the $\mathcal{L}_{\text{cl}}$ too much may have negative effect for the fine-tuning stage. We speculate that minimizing the $\mathcal{L}_{\text{cl}}$ too much will push data points near the decision boundary to be too closed to each other, thus become more difficult the classifer to separate them. Because no matter how well we train the GNN classifer, there are still mis-classified samples due to the natural overlaps between the data distribution of different classes. But in the contrastive pre-training state, the classifer is not aware of whether the samples being pulled together are really from the same class.

Therefore, we propose a new semi-supervised training strategy, namely the joint-strategy by alternately minimizing the $\mathcal{L}_{\text{cl}}$ and $\mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{cls}}$. Minimizing $\mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{cls}}$ is inspired by InfoMin (Tian et al. 2020), so as to make the two view generator to keep label-related information while having less mutual information. However, since we only have a small portion of labeled data to train our view generator, it is still beneficial to use the original data just like the naive-strategy. Interestingly, since we need to minimize $\mathcal{L}_{\text{cls}}$ and $\mathcal{L}_{\text{sim}}$ simultaneously, a weight λ can be applied to better balance the optimization, but actually we found setting $\lambda = 1$ works pretty well during the experiments. The detailed training strategy is described in Algorithm 2. And the overview of our whole framework is shown in Fig. 2.

Experiment

Comparison with State-of-the-Art Methods

Unsupervised Learning For the unsupervised graph classification task, we contrastively train a representation model using unlabeled data, then fix the representation model and train the classifier using labeled data. Following GraphCL (You et al. 2020a), we use a 5-layer GIN with a hidden size of 128 as our representation model, and use an SVM as our classifier. We train the GIN with a batch size of 128 and a learning rate of 0.001. There are 30 epochs of contrastive pre-training under the naive-strategy. We perform a 10-fold cross validation on every dataset. For each fold, we employ 90% of the total data as the unlabeled data for contrastive pre-training, and 10% as the labeled testing data. We repeat every experiment for 5 times using different random seeds.

We compare with the kernel-based methods like graphlet

Model	MUTAG	PROTEINS	DD	NCI1	COLLAB	IMDB-B	REDDIT-B	REDDIT-M-5K
GL	81.66±2.11	-	-	-	-	65.87±0.98	77.34±0.18	41.01±0.17
WL	80.72±3.00	72.92±0.56	-	80.01±0.50	-	72.30±3.44	68.82±0.41	46.06±0.21
DGK	87.44±2.72	73.30±0.82	-	80.31±0.46	-	66.96±0.56	78.04±0.39	41.27±0.18
node2vec	72.63±10.20	57.49±3.57	-	54.89±1.61	-	-	-	-
sub2vec	61.05±15.80	53.03±5.55	-	52.84±1.47	-	55.26±1.54	71.48±0.41	36.68±0.42
graph2vec	83.15±9.25	73.30±2.05	-	73.22±1.81	-	71.10±0.54	75.78±1.03	47.86±0.26
InfoGraph	**89.01±1.13**	74.44±0.31	72.85±1.78	76.20±1.06	70.65±1.13	73.03±0.87	82.50±1.42	53.46±1.03
GraphCL	86.80±1.34	74.39±0.45	**78.62±0.40**	77.87±0.41	71.36±1.15	71.14±0.44	**89.53±0.84**	55.99±0.28
JOAOv2	-	71.25±0.85	66.91±1.75	72.99±0.75	70.40±2.21	71.60±0.86	78.35±1.38	45.57±2.86
AD-GCL	-	73.59±0.65	74.49±0.52	69.67±0.51	**73.32±0.61**	71.57±1.01	85.52±0.79	53.00±0.82
Ours	88.64±1.08	**75.80±0.36**	77.57±0.60	**82.00±0.29**	70.12±0.68	**73.30±0.40**	88.58±1.49	**56.75±0.18**

Table 2: Comparison with the existing methods for unsupervised learning. The bold numbers represent the best performance and the underlined numbers represent the second best performance.

Model	BBBP	Tox21	ToxCast	SIDER	ClinTox	MUV	HIV	BACE
No Pretrain	65.8±4.5	74.0±0.8	63.4±0.6	57.3±1.6	58.0±4.4	71.8±2.5	75.3±1.9	70.1±5.4
Infomax	68.8±0.8	75.3±0.5	62.7±0.4	58.4±0.8	69.9±3.0	75.3±2.5	76.0±0.7	75.9±1.6
EdgePred	67.3±2.4	76.0±0.6	64.1±0.6	60.4±0.7	64.1±3.7	74.1±2.1	76.3±1.0	79.9±0.9
AttrMasking	64.3±2.8	**76.7±0.4**	**64.2±0.5**	61.0±0.7	71.8±4.1	74.7±1.4	77.2±1.1	79.3±1.6
ContextPred	68.0±2.0	75.7±0.7	63.9±0.6	60.9±0.6	65.9±3.8	75.8±1.7	77.3±1.0	79.6±1.2
GraphCL	69.68±0.67	73.87±0.66	62.40±0.57	60.53±0.88	75.99±2.65	69.80±2.66	**78.47±1.22**	75.38±1.44
JOAOv2	71.39±0.92	74.27±0.62	63.16±0.45	60.49±0.74	80.97±1.64	73.67±1.00	77.51±1.17	75.49±1.27
AD-GCL	70.01±1.07	76.54±0.82	63.07±0.72	**63.28±0.79**	79.78±3.52	72.30±1.61	78.28±0.97	78.51±0.80
Ours	**73.36±0.77**	75.69±0.29	63.47±0.38	62.51±0.63	**80.99±3.38**	**75.83±1.30**	78.35±0.64	**83.26±1.13**

Table 3: Comparison with the existing methods for transfer learning. The bold numbers represent the best performance and the underlined numbers represent the second best performance.

kernel (GL) (2009), Weisfeiler-Lehman sub-tree kernel (WL) (2011) and deep graph kernel (DGK) (2015), and other unsupervised graph representation methods like node2vec (Grover and Leskovec 2016), sub2vec (Adhikari et al. 2018), graph2vec (Narayanan et al. 2017) also the contrastive learning methods like InfoGraph (Sun et al. 2019), GraphCL (You et al. 2020a), JOAO (You et al. 2021) and AD-GCL (Suresh et al. 2021). Table 2 show the comparison among different models for unsupervised learning. Our proposed model achieves the best results on PROTEINS, NCI1, IMDB-binary, and REDDIT-Multi-5K datasets and the second best performances on MUTAG, DD, and REDDIT-binary datasets, outperforming current state-of-the-art contrastive learning methods GraphCL, JOAO and AD-GCL.

Transfer Learning We also evaluate the transfer learning performance of the proposed method. A strong baseline method for graph transfer learning is Pretrain-GNN (Hu et al. 2019). The network backbone of Pretrain-GNN, GraphCL, JOAO, AD-GCL and our method is a variant of GIN (Xu et al. 2018), which incorporates the edge attribute. We perform 100 epochs of supervised pre-training on the pre-processed ChEMBL dataset ((Mayr et al. 2018; Gaulton et al. 2012)), which contains 456K molecules with 1,310 kinds of diverse and extensive biochemical assays.

We perform 30 epochs of fine-tuning on the 8 chemistry evaluation subsets. We use a hidden size of 300 for the classifier, a hidden size of 128 for the view generator. We train the model using a batch size of 256 and a learning rate of 0.001. The results in Table 3 are the mean±std of the ROC-AUC scores from 10 reps. Infomax, EdgePred, AttrMasking, ContextPred are the manually designed pre-training strategies from Pretrain-GNN (Hu et al. 2019).

Table 3 presents the comparison among different methods. Our proposed method achieves the best performance on most dataset, such as BBBP, ClinTox, MUV, and BACE, and compared with the current SoTA model AD-GCL (Suresh et al. 2021), our method performs considerably better, for example, on BACE dataset, the accuracy raises from 78.51±0.80 to 83.26±1.13. Considering all datasets, the average gain of using our proposed method is around 1.5%. Interestingly, AttrMasking achieves the best performance on Tox21 and ToxCast, which is slightly better than our method. One possible reason is that attributes are important for classification in Tox21 and ToxCast datasets.

Semi-Supervised Learning We perform semi-supervised graph classification task on TUDataset (Morris et al. 2020). For our view generator, we use a 5-layer GIN with a hidden size of 128 as the embedding model. We use ResGCN (2019) with a hidden size of 128 as the classifier. For GraphCL, we use the default augmentation policy *random4*, which randomly selects two augmentations from node dropout, edge perturbation, subgraph, and attribute masking for every mini-batch. All augmentation ratios are set to 0.2, which is also the default setting in GraphCL.

We employ a 10-fold cross validation on each dataset. For each fold, we use 80% of the total data as the unlabeled data, 10% as labeled training data, and 10% as labeled testing data. For the augmentation only (Aug Only) experiments, we only perform 30 epochs of supervised training with augmentations using labeled data. For the contrastive learning experiments of GraphCL and our naive-strategy, we perform 30 epochs of contrastive pre-training followed by 30 epochs of supervised training. For our joint-strategy, there is 30 joint epochs of contrastive training and supervised training.

Table 4 compares the performances obtained by different training strategies: augmentation only (Aug only), naive-

Model	PROTEINS	DD	NCI1	COLLAB	GITHUB	IMDB-B	REDDIT-B	REDDIT-M-5K
Full Data	78.25±1.61	80.73±3.78	83.65±1.16	83.44±0.77	66.89±1.04	76.60±4.20	89.95±2.06	55.59±2.24
10% Data	69.72±6.71	74.36±5.86	75.16±2.07	74.34±2.00	61.05±1.57	64.80±4.92	76.75±5.60	49.71±3.20
10% GCA	73.85±5.56	76.74±4.09	68.73±2.36	74.32±2.30	59.24±3.21	**73.70±4.88**	77.15±6.96	32.95±10.89
10% GraphCL Aug Only	70.71±5.63	76.48±4.12	70.97±2.08	73.56±2.52	59.80±1.94	71.10±5.11	76.45±4.83	47.33±4.02
10% GraphCL CL	74.21±4.50	76.65±5.12	73.16±2.90	75.50±2.15	63.51±1.02	68.10±5.15	78.05±2.65	48.09±1.74
10% JOAOv2	73.31±0.48	75.81±0.73	74.86±0.39	75.53±0.18	**66.66±0.60**	-	88.79±0.65	52.71±0.28
10% AD-GCL	73.96±0.47	**77.91±0.73**	**75.18±0.31**	75.82±0.26	-	-	**90.10±0.15**	**53.49±0.28**
10% Our Aug Only	75.49±5.15	77.16±4.53	73.33±2.86	75.92±1.93	60.65±1.04	71.90±2.88	79.65±2.84	47.97±2.22
10% Our CL Naive	74.57±3.29	75.55±4.76	73.22±2.48	76.60±2.15	60.95±1.32	71.00±2.91	79.10±4.38	46.71±2.64
10% Our CL Joint ($\mathcal{L}_{\text{cls}}$)	74.66±2.58	76.57±5.08	71.78±1.61	75.38±2.15	60.39±1.50	70.60±4.17	78.90±3.11	46.89±3.13
10% Our CL Joint ($\mathcal{L}_{\text{cls}}$+$\mathcal{L}_{\text{sim}}$)	75.12±3.35	76.23±3.57	72.55±2.72	75.60±2.08	60.18±1.75	71.70±3.86	79.25±2.88	47.51±2.51
10% Our CL Joint ($\mathcal{L}_{\text{cl}}$ +$\mathcal{L}_{\text{cls}}$)	74.75±3.35	76.82±3.85	73.07±2.31	76.18±2.46	61.75±1.30	71.50±5.32	78.35±4.21	47.73±2.69
10% **Our CL Joint** ($\mathcal{L}_{\text{cl}}$ +$\mathcal{L}_{\text{cls}}$+$\mathcal{L}_{\text{sim}}$)	**75.65±2.40**	77.50±4.41	73.75±2.25	**77.16±1.48**	62.46±1.51	71.90±4.79	79.80±3.47	49.91±2.70

Table 4: Comparison with existing methods and different strategies for semi-supervised learning. The bold numbers represent the best performance and the underlined numbers represent the second best performance.

strategy (CL naive) and joint-strategy (CL joint). We also conducted an ablation study of our joint loss function. The proposed CL joint approach achieves relatively high accuracy on most datasets, *e.g.*, on PROTEINS and COLLAB datasets, using joint strategy obtains the best performance. For other datasets, using joint strategy could also achieve the second best performances. Looking at the comparison among Aug only, CL naive and CL joint, CL joint is superior to the other two approaches, in particular to CL naive.

Effectiveness of Learnable View Generators

In this section, we demonstrate the superiority of learnable graph augmentation policies over the fixed ones. Since the graph datasets are usually difficult to be manually classified and visualized, we trained a view generator on MNIST-Superpixel dataset (Monti et al. 2017) to verify that our graph view generator is able to effectively capture the semantic information in graphs than GraphCL (You et al. 2020a), since MNIST-Superpixel graphs have clear semantics which does not require any domin knowledge. The visualization result is shown in Fig. 3.

Here we jointly trained the view generators with the classifier until the test accuracy (evaluated on generated views) reached 90%. Since our only topological augmentation is node dropping. So we compared the view of GraphCL's node dropping augmentation, and use the default setting aug_ratio $= 0.2$. Fig. 3 shows that, our view generator are more likely to keep key nodes in the original graph, preserving its semantic feature, yet providing enough variance for contrastive learning. Details and more visualization examples are shown in the supplementary.

Analysis for Joint Training Strategy

We compared the naive-strategy with the joint-strategy. We trained on COLLAB (2015) dataset, which have 5000 social network graphs of 3 classes, the average nodes and edges are 74.49 and 2457.78. Here we use 5-layer GIN (Xu et al. 2018) as the backbone for both the view generator and the classifier. For naive-strategy, there is 30 epochs of contrastive pretrain using 80% unlabeled data and 30% of fine-tuning using 10% of data. For joint-strategy, there is 30 epochs of joint training. Our results show that the joint strategy considerably alleviate the over-fitting effect, and our label-preserving view generator is very effective. We also visualize the learning curves and the process for learning the embedding for each strategy using t-SNE (2008) in the supplementary. We found that joint-strategy leads to better representation much faster since labeled data is used for supervision, also this supervision signal could benefit view generator learning.

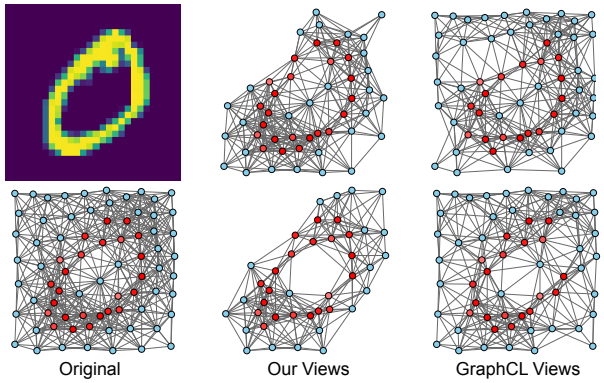

Figure 3: View visualization on the MNIST-Superpixel dataset. Redness reflects the magnitude of node attribute.

Conclusion

In this paper, we presented a learnable data augmentation approach to graph contrastive learning, where we employed GIN to generate different views of the original graphs. To preserve the semantic label of the input graph, we developed a joint learning strategy, which alternately optimizes view generators, graph encoders and the classifier. We also conducted extensive experiments on a number of datasets and tasks, such as semi-supervised learning, unsupervised learning and transfer learning, where results demonstrate the advantage of our proposed method outperforming counterparts on most datasets and tasks. In addition, we visualized the generated graph views, which could preserve discriminative structures of input graphs, benefiting classification. Finally, the t-SNE visualization illustrated that the proposed joint training strategy could be a better choice for semi-supervised graph representation learning.

Acknowledgements

We appreciate the efforts made by AC, SPCs, PCs and Reviewers for improving the manuscript. This work was done in part, when Mr. Yihang Yin did his research internship at Baidu Research. Q. Wang and H. Xiong were supported in part by National Key R&D Program of China (No. 2018YFB1402600).

References

Adhikari, B.; Zhang, Y.; Ramakrishnan, N.; and Prakash, B. A. 2018. Sub2vec: Feature learning for subgraphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 170–182. Springer.

Chen, T.; Bian, S.; and Sun, Y. 2019. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv preprint arXiv:1905.04579*.

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G. 2020a. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, 1597–1607. PMLR.

Chen, X.; Fan, H.; Girshick, R.; and He, K. 2020b. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*.

Cubuk, E. D.; Zoph, B.; Mane, D.; Vasudevan, V.; and Le, Q. V. 2019. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 113–123.

De Cao, N.; and Kipf, T. 2018. MolGAN: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*.

Fan, W.; Ma, Y.; Li, Q.; He, Y.; Zhao, E.; Tang, J.; and Yin, D. 2019. Graph neural networks for social recommendation. In *The World Wide Web Conference*, 417–426.

Gaulton, A.; Bellis, L. J.; Bento, A. P.; Chambers, J.; Davies, M.; Hersey, A.; Light, Y.; McGlinchey, S.; Michalovich, D.; Al-Lazikani, B.; et al. 2012. ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40(D1): D1100–D1107.

Grill, J.-B.; Strub, F.; Altché, F.; Tallec, C.; Richemond, P. H.; Buchatskaya, E.; Doersch, C.; Pires, B. A.; Guo, Z. D.; Azar, M. G.; et al. 2020. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864.

Hadsell, R.; Chopra, S.; and LeCun, Y. 2006. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, 1735–1742. IEEE.

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

Hassani, K.; and Khasahmadi, A. H. 2020. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, 4116–4126. PMLR.

Hataya, R.; Zdenek, J.; Yoshizoe, K.; and Nakayama, H. 2020. Faster autoaugment: Learning augmentation strategies using backpropagation. In *European Conference on Computer Vision*, 1–16. Springer.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9729–9738.

Hu, W.; Liu, B.; Gomes, J.; Zitnik, M.; Liang, P.; Pande, V.; and Leskovec, J. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*.

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

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

Kipf, T. N.; and Welling, M. 2016b. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25: 1097–1105.

Li, Y.; Hu, G.; Wang, Y.; Hospedales, T.; Robertson, N. M.; and Yang, Y. 2020. Differentiable Automatic Data Augmentation. In *European Conference on Computer Vision*, 580–595. Springer.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Logeswaran, L.; and Lee, H. 2018. An efficient framework for learning sentence representations. *arXiv preprint arXiv:1803.02893*.

Madhawa, K.; Ishiguro, K.; Nakago, K.; and Abe, M. 2019. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*.

Mayr, A.; Klambauer, G.; Unterthiner, T.; Steijaert, M.; Wegner, J. K.; Ceulemans, H.; Clevert, D.-A.; and Hochreiter, S. 2018. Large-scale comparison of machine learning methods for drug target prediction on ChEMBL. *Chemical science*, 9(24): 5441–5451.

Monti, F.; Boscaini, D.; Masci, J.; Rodola, E.; Svoboda, J.; and Bronstein, M. M. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 5115–5124.

Morris, C.; Kriege, N. M.; Bause, F.; Kersting, K.; Mutzel, P.; and Neumann, M. 2020. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*.

Narayanan, A.; Chandramohan, M.; Venkatesan, R.; Chen, L.; Liu, Y.; and Jaiswal, S. 2017. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*.

Rosenstein, M. T.; Marx, Z.; Kaelbling, L. P.; and Dietterich, T. G. 2005. To transfer or not to transfer. In *NIPS 2005 workshop on transfer learning*, volume 898, 1–4.

Rossi, R. A.; and Ahmed, N. K. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*.

Shervashidze, N.; Schweitzer, P.; Van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9).

Shervashidze, N.; Vishwanathan, S.; Petri, T.; Mehlhorn, K.; and Borgwardt, K. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, 488–495. PMLR.

Shi, W.; and Rajkumar, R. 2020. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 1711–1719.

Sohn, K. 2016. Improved deep metric learning with multi-class n-pair loss objective. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 1857–1865.

Sun, F.-Y.; Hoffmann, J.; Verma, V.; and Tang, J. 2019. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*.

Suresh, S.; Li, P.; Hao, C.; and Neville, J. 2021. Adversarial Graph Augmentation to Improve Graph Contrastive Learning. *arXiv preprint arXiv:2106.05819*.

Tian, Y.; Krishnan, D.; and Isola, P. 2019. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*.

Tian, Y.; Sun, C.; Poole, B.; Krishnan, D.; Schmid, C.; and Isola, P. 2020. What makes for good views for contrastive learning. *arXiv preprint arXiv:2005.10243*.

Van der Maaten, L.; and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11).

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

Veličković, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2018. Deep graph infomax. *arXiv preprint arXiv:1809.10341*.

Wang, Y.; Min, Y.; Shao, E.; and Wu, J. 2021. Molecular Graph Contrastive Learning with Parameterized Explainable Augmentations. In *2021 IEEE International conference on bioinformatics and biomedicine (BIBM)*, 1558–1563. IEEE.

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

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.

Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R.; and Le, Q. V. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

You, Y.; Chen, T.; Shen, Y.; and Wang, Z. 2021. Graph Contrastive Learning Automated. *arXiv preprint arXiv:2106.07594*.

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

You, Y.; Chen, T.; Wang, Z.; and Shen, Y. 2020b. When does self-supervision help graph convolutional networks? In *International Conference on Machine Learning*, 10871–10880. PMLR.

Zbontar, J.; Jing, L.; Misra, I.; LeCun, Y.; and Deny, S. 2021. Barlow twins: Self-supervised learning via redundancy reduction. *arXiv preprint arXiv:2103.03230*.

Zhu, Y.; Xu, Y.; Yu, F.; Liu, Q.; Wu, S.; and Wang, L. 2020a. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*.

Zhu, Y.; Xu, Y.; Yu, F.; Liu, Q.; Wu, S.; and Wang, L. 2020b. Graph Contrastive Learning with Adaptive Augmentation. *arXiv preprint arXiv:2010.14945*.