Contrastive Graph Few-Shot Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Prevailing deep graph learning models often suffer from label sparsity issue. Although many graph few-shot learning (GFL) methods have been developed to avoid the performance degradation in face of limited annotated data, they excessively rely on labeled data, where the distribution shift in the test phase might result in impaired generalization ability. Additionally, they lack a general purpose as their designs are coupled with task or data-specific characteristics. To this end, we propose a general and effective **C**ontrastive **G**raph **F**ew-shot **L**earning framework (CGFL). CGFL leverages a self-distilled contrastive learning procedure to boost GFL. Specifically, our model firstly pre-trains a graph encoder with contrastive learning using unlabeled data. Later, the trained encoder is frozen as a teacher model to distill a student model with a contrastive loss. The distilled model is finally fed to GFL. CGFL learns data representation in a self-supervised manner, thus mitigating the distribution shift impact for better generalization and making model task and data-independent for a general graph mining purpose. Furthermore, we introduce an information-based method to quantitatively measure the capability of CGFL. Comprehensive experiments demonstrate that CGFL outperforms state-of-the-art baselines on several graph mining tasks in the few-shot scenario. We also provide quantitative measurement of CGFL's success.

1 Introduction

Deep graph learning, e.g., graph neural networks (GNNs), has recently attracted tremendous attention due to its remarkable performance in various application domains, such as social/information systems [23, 14], molecular chemistry/biology [21, 15], and recommendation [51, 6]. The success of GNNs often relies on massive annotated samples, which contradicts the fact that it is expensive to collect sufficient labels. This motivates the graph few-shot learning (GFL) study to tackle performance degradation in the face of limited labeled data.

Previous GFL models are built on meta-learning (or few-shot learning) techniques, either metric-based approaches [43, 37] or optimization-based algorithms [8]. They aim to quickly learn an effective GNN adapted to new tasks with few labeled samples. GFL has been applied to a variety of graph mining tasks, including node classification [55, 19], relation prediction [47, 27, 54], and graph classification [1, 29]. Despite substantial progress, most previous GFL models still have the following limitations: *(i) Impaired generalization.* Existing GFL methods excessively rely on labeled data and attempt to inherit a strong inductive bias for new tasks in the test phase. However, a distribution shift exists between non-overlapping meta-training data and meta-testing data. Without supervision signals from ground-truth labels, GFL may not learn an effective GNN for novel classes of test data. This gap limits the meta-trained GNN's generalization and transferability. *(ii) Constrained design.* Most of the current GFL methods lack a general purpose as they possess the premise that the designated task is universally the same prior across different graph tasks or datasets, which in fact is not always guaranteed. For example, GSM [1] needs to manually define a superclass of graphs, which cannot

Submitted to NeurIPS 2022 GLFrontiers Workshop. Do not distribute.

expand to node-level tasks. The task or data-specific design limits the GFL's utility for different graph mining tasks.

The above challenges call for a new generic GFL framework that can learn a generalizable, transferable, and effective GNN for various graph mining tasks with few labels. Fortunately, contrastive learning has emerged to alleviate the dependence on labeled data, and learn label-irrelevant but transferable representations from unsupervised pretext tasks for vision, language, and graphs [3, 10, 53, 38]. Thus, the natural idea is to leverage contrastive learning to boost GFL.

In this work, we are motivated to develop a general and effective **C**ontrastive **G**raph **F**ew-shot **L**earning framework (CGFL) with contrastive learning. To be specific, the proposed framework firstly pre-trains a GNN by minimizing the contrastive loss between two views' embeddings generated in two augmented graphs. Later, we introduce a self-distillation step to bring additional elevation: the pre-trained GNN is frozen as a teacher model and kept in the contrastive framework to distill a randomly initialized student model by minimizing the agreement of two views' embeddings generated by two models. Both pre-training and the distillation steps can work at the meta-training and meta-testing phases without requiring labeled data. Finally, the distilled student model is taken as the initialized model fed to GFL for few-shot graph mining tasks. CGFL pre-trains GNN self-supervised, thus mitigating the negative impact of distribution shift. The learned graph representation is transferable and discriminable for new tasks in the test data. Besides, our simple and generic framework of CGFL is applicable for different graph mining tasks. Furthermore, to quantitatively measure the capability of CGFL, we introduce information-based method to measure the quality of learned node (or graph) embeddings on each layer of the model: we allocate each node a learnable variable as a noise and train these variables to maximize the entropy while keeping the change of output as small as possible.

To summarize, our contributions in this work are:

- We develop a general and effective framework named CGFL to leverage a self-distilled contrastive learning procedure to boost GFL. CGFL mitigates distribution shift impact and has the task and data-independent capacity for a general graph mining purpose.
- We introduce an information-based method to quantitatively measure the capability of CGFL by measuring the quality of learned node (or graph) embeddings. To the best of our knowledge, this is the first study to explore GFL model measurement.
- Comprehensive experiments on multiple graph datasets demonstrate that CGFL outperforms state-of-the-art methods for both node classification and graph classification tasks in the few-shot scenario. Additional measurement results further show that CGFL learns better node (or graph) embeddings than baseline methods.

2 Related Work

Few-Shot Learning on Graphs. Many GFL models have been proposed to solve various graph mining problems in face of label sparsity issue, such as node classification [55, 50, 4, 19], relation prediction [47, 27, 2, 54], and graph classification [1, 29, 13, 44]. They are built on meta-learning (or few-shot learning) techniques that can be categorized into two major groups: (1) metric-based approaches [43, 37]; (2) optimization-based algorithms [8]. For the first group, they learn effective similarity metric between few-shot support data and query data. For example, GPN [4] conducts node informativeness propagation to build weighted class prototypes for a distance-based node classifier. The second group proposes to learn well-initialized GNN parameters that can be fast adapted to new graph tasks with few labeled data. For instance, G-Meta [19] builds local subgraphs to extract subgraph-specific information and optimizes GNN via MAML [8]. Unlike prior efforts that rely on labeled data and have the task and data-specific design, we aim to build a novel framework that explores unlabeled data and has a generic design for a general graph mining purpose.

Self-Supervised Learning on Graphs. Recently, self-supervised graph learning (SGL) has attracted significant attention due to its effectiveness in pre-training GNN and competitive performance in various graph mining applications. Previous SGL models can be categorized into two major groups: generative or contrastive, according to their learning tasks [25, 38]. The generative models learn graph representation by recovering feature or structural information on the graph. The task can solely recover adjacency matrix alone [52] or along with the node features [18]. As for the contrastive methods, they firstly define the node context which can be node-level or graph-level instances. Then, they perform contrastive learning by either maximizing the mutual information between the node-context pairs [16, 42, 39] or by discriminating context instances [32, 56]. In addition to

above strategy, recently random propagation applies graph augmentation [33] for semi-supervised learning [7]. Motivated by the success of SGL, we propose to leverage it to boost GFL.

3 Preliminary

GNNs. A graph is represented as $G = (V, E, X)$, where V is the set of nodes, $E \subseteq V \times V$ is the set of edges, and X is the set of node attributes. GNNs [14, 48] learn compact representations (embeddings) by considering both graph structure E and node attributes X. To be specific, let $f_\theta(\cdot)$ denote a GNN encoder with parameter θ, the updated embedding of node v at the l-th layer of GNN can be formulated as:

$$h_v^{(l)} = \mathcal{M}(h_v^{(l-1)}, \{h_u^{(l-1)} \mid \forall u \in \mathcal{N}_v\}; \theta), \tag{1}$$

where $\mathcal{N}_v$ denotes the neighbor set of v; $\mathcal{M}(\cdot)$ is the message passing function for neighbor information aggregation, such as a mean pooling layer followed by a fully-connected (FC) layer; $h_v^{(0)}$ is initialized with node attribute X_v. The whole graph embedding can be computed over all nodes' embeddings as:

$$h_G^{(l)} = \text{READOUT}\{h_v^{(l)} \mid \forall v \in V\}, \tag{2}$$

where the READOUT function can be a simple permutation invariant function such as summation.

GFL Setting and Problem. Let $\mathcal{C}_{base}$ and $\mathcal{C}_{novel}$ denote the base classes set and novel (new) classes set in training data $\mathcal{T}_{train}$ and testing data $\mathcal{T}_{test}$, respectively. Similar to the general meta-learning problem [8], the purpose of graph few-shot learning (GFL) is to train a GNN encoder $f_\theta(\cdot)$ over $\mathcal{C}_{base}$, such that the trained GNN encoder can be quickly adapted to $\mathcal{C}_{novel}$ with few labels per class. Note that there is no overlapping between base classes and novel classes, i.e., $\mathcal{C}_{base} \cap \mathcal{C}_{novel} = \varnothing$. In K-shot setting, during the meta-training phase, a batch of classes (tasks) is randomly sampled from $\mathcal{C}_{base}$, where K labeled instances per class are sampled to form the support set $\mathcal{S}$ for model training and the remaining instances are taken as the query set $\mathcal{Q}$ for model evaluation. After sufficient training, the model is further transferred to the meta-testing phase to conduct N-way classification over $\mathcal{C}_{novel}$ (N is the number of novel classes), where each class is only with K labeled instances. GFL applies to different graph mining problems, depending on the class meaning. Each class corresponds to a node label for the node classification problem or corresponds to a graph label for the graph classification problem. In this work, we will study both node classification and graph classification problems under few-shot setting, which are formally defined as follows:

Problem 1 ***Few-Shot Node Classification***. *Given a graph $G = (V, E, X)$ and labeled nodes of $\mathcal{C}_{base}$, the problem is to learn a GNN $f_\theta(\cdot)$ to classify nodes of $\mathcal{C}_{novel}$, where each class in $\mathcal{C}_{novel}$ only has few labeled nodes.*

Problem 2 ***Few-Shot Graph Classification***. *Given a set of graphs $\mathcal{G}$ and labeled graphs of $\mathcal{C}_{base}$, the problem is to learn a GNN $f_\theta(\cdot)$ to classify graphs of $\mathcal{C}_{novel}$, where each class in $\mathcal{C}_{novel}$ only has few labeled graphs.*

Unlike previous studies that rely on labeled data of $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$ for GFL model training and adaption, we consider both unlabeled graph information and labeled data to learn GFL model for solving the above problems.

4 Methodology

Figure 1 illustrates the proposed CGFL framework, which includes two phases: self-distilled graph contrastive learning and graph few-shot learning (GFL). In the first phase (Figure 1(a)), the framework pre-trains a GNN encoder with contrastive learning, then introduces knowledge distillation to elevate the pre-trained GNN in a self-supervised manner. The distilled GNN is finally fed to the GFL phase (Figure 1(b)) for few-shot graph mining tasks. In addition to the proposed framework, we introduce an information-based method to measure the superiority of CGFL quantitatively.

4.1 Self-Distilled Graph Contrastive Learning

GNN Contrastive Pre-training. In the first phase, we firstly introduce contrastive learning to pre-train GNN. Inspired by the representation bootstrapping technique [11], our method learns node

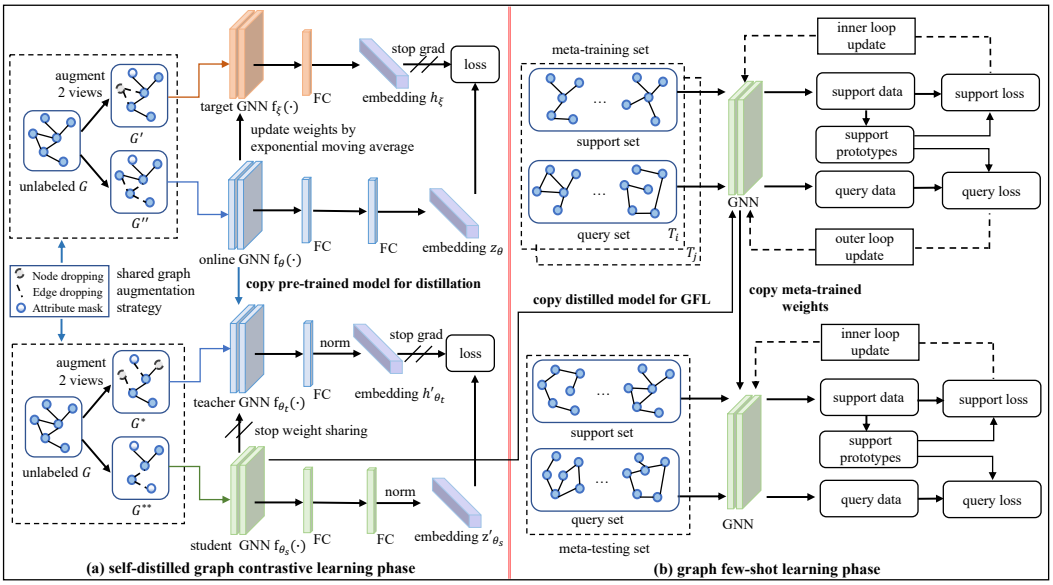


Figure 1: The overall framework of CGFL: (a) self-distilled graph contrastive learning phase which pre-trains a GNN encoder with contrastive learning and further evaluates the model with knowledge distillation in self-supervised manner; (b) graph few-shot learning phase which takes the distilled student network as the initialized model and employs meta-learning algorithm for model optimization.

(or graph) representation by discriminating context instances. Specifically, two GNN encoders: an online GNN $f_\theta(\cdot)$ and a target GNN $f_\xi(\cdot)$, are introduced to encode two randomly augmented views of a given graph. The online GNN is supervised under the target GNN's output, while the target GNN is updated by the online GNN's exponential moving average. The contrastive pre-training step is shown in Figure 1(a).

Graph Augmentation: The given graph G is processed with randomly data augmentations to generate a contrastive pair (G', G'') as the input for two GNN branches (online branch and target branch) of the following GNN training. In this work, we apply a combination of stochastic node feature masking, edge removing, and node dropping with constant probabilities for graph augmentation.

GNN Update: With the generated graph pair (G', G''), the online GNN $f_\theta(\cdot)$ and the target GNN $f_\xi(\cdot)$ are respectively utilized to process G' and G'' for node (or graph) embeddings generation. Both GNNs have the same architecture, while a two-layer FC (one-layer FC) is attached after online GNN (target GNN) to refine embedding. The reason that two branches have different FC layers is to prevent the prediction of the online model from being exactly the same as the output of the target model, thus avoiding the learned representation collapse. Later, to enforce online GNN's embeddings z_θ approximate the target GNN's embeddings h_ξ, the mean squared error between them is formulated as the objective function:

$$\mathcal{L}_{\theta,\xi} = \|z_\theta - h_\xi\|_2^2 = 2 - 2 \cdot \frac{z_\theta, h_\xi}{\|z_\theta\|_2 \cdot \|h_\xi\|_2}. \tag{3}$$

The parameters θ of online GNN are updated with Adam optimizer [22]:

$$\theta \leftarrow \text{Adam}(\theta, \nabla_\theta \mathcal{L}_{\theta,\xi}, \eta), \tag{4}$$

where η is the learning rate. The target GNN provides the regression target to supervise the online GNN, and its parameters ξ are updated as exponential moving average (EMA) of the online GNN parameters θ. More precisely, ξ is updated as follows:

$$\xi \leftarrow \tau\xi + (1-\tau)\theta, \tag{5}$$

where $\tau \in [0, 1]$ is the decay rate. Note that the target GNN stops the backpropagation from $\mathcal{L}_{\theta,\xi}$, and it is only updated by EMA.

Contrastive Distillation. With the pre-trained GNN $f_\theta(\cdot)$ obtained in the previous step, we introduce a self-distillation step to elevate $f_\theta(\cdot)$. This is inspired by the Born-again strategy [9], which implies a well-trained teacher can boost a random initialized identical student. The distillation step adopts a

similar contrastive framework as the previous step, as shown in Figure 1(a). Specifically, we load the pre-trained GNN $f_\theta(\cdot)$ and take it as the teacher model $f_{\theta_t}(\cdot)$. The teacher model is frozen and applied to distill a student model $f_{\theta_s}(\cdot)$. Two augmented views (G^*, G^{**}) of a graph G are generated and fed to $f_{\theta_t}(\cdot)$ and $f_{\theta_s}(\cdot)$, respectively. Later, the student's normalized output is forced to approximate the teacher's normalized output as follows:

$$\mathcal{L}_{\theta_s} = \|z'_{\theta_s} - h'_{\theta_t}\|_2^2 = 2 - 2 \cdot \frac{z'_{\theta_s}, h'_{\theta_t}}{\|z'_{\theta_s}\|_2 \cdot \|h'_{\theta_t}\|_2}, \tag{6}$$

$$z'_{\theta_s} = \frac{z_{\theta_s}}{\|z_{\theta_s}\|_2},\ h'_{\theta_t} = \frac{h_{\theta_t}}{\|h_{\theta_t}\|_2}, \tag{7}$$

where z_{θ_s} and h_{θ_t} are teacher's output embeddings and student's output embeddings, respectively. The student model is updated as follows:

$$\theta_s \leftarrow \text{Adam}(\theta_s, \nabla_{\theta_s}\mathcal{L}_{\theta_s}, \eta). \tag{8}$$

Different from EMA (Eqn. 5) for target GNN update in contrastive pre-training, the teacher model is frozen and can be seen as a special case of EMA:

$$\theta_t \leftarrow \tau\theta_t + (1-\tau)\theta_s,\ \tau = 1. \tag{9}$$

4.2 Graph Few-Shot Learning

In GFL phase, we take the distilled student GNN $f_{\theta_s}(\cdot)$ generated in the former phase as the initialized GNN model and employ the optimization-based algorithm, i.e., model-agnostic meta-learning (MAML) [8], to train the model for few-shot graph mining tasks. During meta-training, for task T_i, the task specific parameters $\theta'_{s,i}$ is computed using a number of gradient descent updates over the support set $\mathcal{S}_i$ of T_i (i.e., inner-loop):

$$\theta'_{s,i} \leftarrow \theta_s - \alpha\nabla_{\theta_s}\mathcal{L}_{\mathcal{T}_i}^{\mathcal{S}_i}(\theta_s), \tag{10}$$

where α is the learning step size, $\mathcal{L}_{\mathcal{T}_i}^{\mathcal{S}_i}$ denotes the downstream task loss over $\mathcal{S}_i$. In this work, we employ prototypical loss [37] for node or graph classification, which uses embeddings extracted from the support set by a neural network as the class prototype, and the query set is classified according to the distance between its embeddings and prototypes. The task-specific parameter $\theta'_{s,i}$ is further utilized to compute the loss over query set $\mathcal{Q}_i$ of T_i: $\mathcal{L}_{\mathcal{T}_i}^{\mathcal{Q}_i}(f_{\theta'_{s,i}})$. Later, $\mathcal{L}_{\mathcal{T}_i}^{\mathcal{Q}_i}(f_{\theta'_{s,i}})$ of a batch of randomly sampled tasks are summed up to update the model parameters θ_s (i.e., outer-loop):

$$\theta_s \leftarrow \theta_s - \beta\nabla_{\theta_s}\sum_i \mathcal{L}_{\mathcal{T}_i}^{\mathcal{Q}_i}(f_{\theta'_{s,i}}), \tag{11}$$

where β is the learning step size. During meta-testing, the same procedure above is applied using the final meta-updated parameter θ_s^* for novel tasks (without outer-loop). In particular, θ_s^* is learned from knowledge across meta-training tasks and is the optimal parameter to quickly adapt to novel tasks. Note that the GFL algorithm can be applied to different graph mining problems by changing the task meaning, i.e., each task corresponds to a node class (or graph class) for node classification (or graph classification).

4.3 Quantitative Measurement of GFL

The previous GFL studies target developing better methods in performance while none of them has thought about model capability measurement. In light of this, we extend the measurement of neural network model [28] to graph data, and quantitatively show why different GFL models can learn node (or graph) representations at different extents. The proposed method provides a measurement of information encoded in GNN for the input graph. Specifically, let Z denote GNN hidden state of a GFL model, the information of the input graph G encoded by Z can be measured by mutual information $MI(G;Z)$:

$$MI(G;Z) = H(G) - H(G|Z), \tag{12}$$

where $H(\cdot)$ denotes the entropy; $H(G)$ is a constant; $H(G|Z)$ represents the amount of discarded information after G is processed by GNN and encoded by Z. We can compute $H(G|Z)$ by decomposing it into the node level:

$$H(G|Z) = \int_{\mathbf{z}\in Z} p(\mathbf{z})H(G|\mathbf{z})d\mathbf{z}, \tag{13}$$

where $\mathbf{z} = f(x)$ denotes GNN hidden state corresponding to attribute x of a node. $H(G|\mathbf{z})$ reflects how much information from x is discarded by $\mathbf{z}$ during the forward propagation. To disentangle information components of individual nodes from the whole graph, we assume that each node is independent of each other and have:

$$H(G|\mathbf{z}) = \sum_i H(x_i|\mathbf{z}), \tag{14}$$

where x_i denotes a random variable of i-th node attribute in the graph. Then, we introduce a noise perturbation-based method to approximate $H(x_i|\mathbf{z})$. Specifically, let $\widetilde{x}_i = x_i + \epsilon_i$ ($\epsilon_i \sim \mathcal{N}(0, \mathbf{\Sigma}_i = \sigma_i^2\mathbf{I})$) and we aim to optimize the following loss function:

$$\mathcal{L}(\sigma) = \mathbb{E}_\epsilon \|f(\widetilde{x}_i) - \mathbf{z}\|^2 - \beta \sum_{i=1}^{n} H(\widetilde{x}_i|\mathbf{z})|_{\epsilon_i \sim \mathcal{N}(0,\sigma_i^2\mathbf{I})}, \tag{15}$$

where $\sigma = [\sigma_1, \sigma_2, \cdots, \sigma_n]$ are learnable parameters; β is trade-off weight. In particular, the first term of the above objective minimizes difference between the encoded embedding of noisy input and the hidden state while the second term encourages a high conditional entropy according to the maximum entropy principle. In this way, we have $p(\widetilde{x}_i|\mathbf{z}) = p(\epsilon_i)$ and $H(x_i|\mathbf{z})$ is approximated by $H(\widetilde{x}_i|\mathbf{z})$:

$$H(\widetilde{x}_i|\mathbf{z}) = p(\widetilde{x}_i|\mathbf{z}) \log p(\widetilde{x}_i|\mathbf{z}) \propto \log \sigma_i + C, \tag{16}$$

where $C = \frac{1}{2}\log(2\pi e)$. By taking Eqn. 16 into Eqn. 15, we can optimize $\mathcal{L}(\sigma)$ with Adam optimizer and obtain the optimal σ for computing the overall discarded information $H(G|Z)$. Furthermore, we can explain GFL model's capability by comparing discarded information of different models. Ideally, we expect the GFL model to encode valid node (or graph) embeddings as much as possible and therefore discard information as small as possible.

5 Experiments

We conduct extensive experiments on multiple graph datasets to evaluate the model performance compared with state-of-the-art models. We first describe experimental settings and then discuss the performance comparison of different models. Finally, discarded information is computed to show the capabilities of different GFL models. More experimental results are provided in the Appendix.

5.1 Experimental Setup

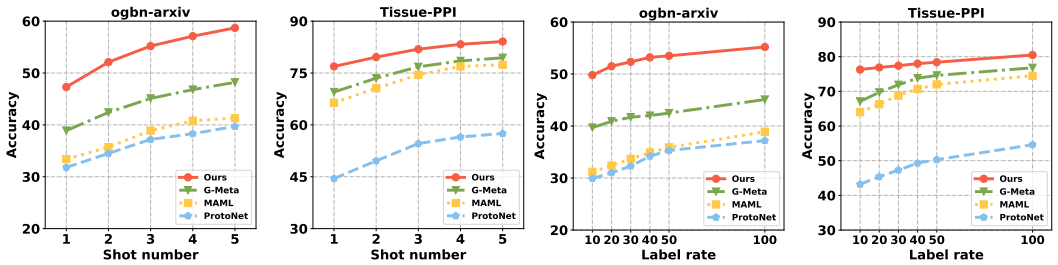
Datasets. We use multiple graph datasets to conduct experiments: for the node classification task, we use ogbn-arxiv [17], Tissue-PPI [14], Fold-PPI [57], Cora [34], and Citeseer [34]; for the graph classification task, we use datasets in [1], i.e., Letter-High, Triangles, Reddit-12K, and Enzymes. The detailed information of datasets is illustrated in the Section A of Appendix.

Baseline Methods. We employ a variety of baseline methods for model comparison in two tasks. For few-shot node classification, we use node2vec [12], DeepWalk [31], Meta-GNN [55], FS-GIN [48], FS-SGC [45], No-Finetune [41], Finetune [41], KNN [41], ProtoNet [37], MAML [8], and G-Meta [19]. For few-shot graph classification, we utilize WL [35], Graphlet [36], AWE [20], Graph2Vec [30], Diffpool [24], CapsGNN [46], GIN [48], GIN-KNN [48], GSM-GCN [1], GSM-GAT [1], and AS-MAML [29]. Details of baselines are illustrated in the Section B of Appendix.

Experimental Settings. In CGFL, we use GCN [23] as the GNN backbone for the node classification task, including two-layer graph convolution and one-layer FC. The graph classification task adds an extra average pooling operation as a readout layer. In pretraining phase, we pre-train GNN on the unlabeled graph dataset with contrastive learning. For graph data augmentation, the node drop rate is 15%, the edge removing rate is 15%; and feature masking rate is 20%. The mini-batch size is set to 2,048, and the learning rate is set to 0.05 with a decay factor = 0.9. Meanwhile, the τ in exponential moving average is 0.999. We consider both inductive and transductive settings. For inductive setting (**CGFL-I**), we only use unlabeled data in the training set; for transductive setting (**CGFL-T**), we use unlabeled data in both training data and testing data. Additionally, for pretrained models without knowledge distillation elevation, we refer to them as **Teacher-I** and **Teacher-T** for two settings, respectively. In GFL phase, we employ MAML to fine-tune the model. We implement CGFL by PyTorch and train it on NVIDIA V100 GPUs. The code is provided in the appendix.

Table 1: Few-shot node classification results. The best results are highlighted in bold while the best baseline results are underlined. -I and -T denote inductive and transductive settings of CGFL, respectively. Teacher indicates the CGFL model without knowledge distillation.

Method	ogbn-arxiv		Tissue-PPI		Fold-PPI		Cora		Citeseer	
	3-shot	5-shot	3-shot	5-shot	3-shot	5-shot	3-shot	5-shot	3-shot	5-shot
node2vec [12]	28.9±4.0	29.5±3.7	48.5±3.3	49.3±3.9	36.6±3.7	37.4±1.9	25.7±1.3	26.9±3.0	20.0±2.5	21.7±2.9
DeepWalk [31]	30.3±2.1	31.5±3.4	46.2±4.8	47.4±3.6	35.0±4.4	36.3±3.2	25.6±0.8	26.7±2.0	21.2±0.6	22.6± 2.7
Meta-GNN [55]	27.3±1.2	30.2±3.6	50.8±8.1	53.5±1.5	30.8±5.4	33.5±2.1	76.8±0.9	79.2±1.9	69.4±1.4	72.6±1.9
FS-GIN [48]	33.6±4.2	36.8±2.5	49.2±2.4	51.5±3.0	36.7±2.1	39.1±1.4	53.5±1.6	56.2±2.8	50.2±2.6	53.2±3.8
FS-SGC [45]	34.7±0.5	37.3±1.0	49.8±3.8	52.3±2.2	38.0±1.6	40.9±3.9	57.2±2.1	60.3±1.2	52.0±2.1	54.4±2.5
No-Finetune [41]	36.4±1.4	38.8±2.0	51.6±0.6	55.0±2.1	37.6±1.7	39.9±3.6	61.2±1.2	64.5±1.3	54.9±1.7	58.3±2.5
Finetune [41]	35.9±1.0	38.6±2.5	52.1±1.3	54.3±2.4	37.0±2.2	40.0±2.6	63.5±0.8	65.7±2.1	57.8±1.8	59.0±2.9
KNN [41]	39.2±1.5	42.3±1.8	61.9±2.5	65.2±3.2	43.3±3.4	46.2±1.9	67.8±1.4	70.3±3.6	60.6±1.4	63.2±1.6
ProtoNet [37]	37.2±1.7	39.7±1.7	54.6±2.5	57.5±2.9	38.2±3.1	41.3±1.1	59.2±1.3	61.6±3.0	55.5±1.5	58.0±3.7
MAML [8]	38.9±2.1	41.3±2.4	74.5±5.1	77.4±2.7	48.2±6.2	51.3±3.3	65.7±0.9	68.8±1.1	63.1±1.6	65.7±1.7
GPN [5]	44.4±3.5	48.2±4.0	77.3±3.0	79.0±3.6	57.0±4.7	58.2±3.7	73.1±2.4	76.1±2.2	68.3±1.4	71.1±2.0
RALE [26]	45.1±2.7	47.8±1.5	76.6±3.3	79.2±3.3	57.8±4.5	58.8±3.3	72.2±2.0	75.6±1.9	69.9±2.3	71.3±2.2
G-Meta [19]	45.1±3.2	48.2±3.1	76.8±2.9	79.4±2.6	56.1±5.9	59.0±2.5	71.9±2.9	74.5±2.0	67.8±2.2	70.8±3.8
Teacher-I	52.0±2.0	54.9±1.6	77.9±2.6	80.8±1.7	58.8±3.4	61.3±3.6	78.2±1.3	80.9±1.9	70.0±1.3	72.7±1.8
CGFL-I	52.8±1.8	55.6±1.1	78.7±2.8	81.5±3.6	59.5±4.1	62.0±2.0	78.5±1.5	81.2±1.5	70.6±1.2	73.1±2.0
Teacher-T	54.3±2.7	58.4±3.9	79.8±3.1	82.9±1.3	63.0±3.6	65.6±2.2	80.0±2.7	82.6±1.9	72.1±1.1	75.0±1.5
CGFL-T	**55.2±2.5**	**58.7±2.7**	**80.9±3.0**	**84.1±3.2**	**66.9±3.4**	**69.0±2.7**	**80.7±1.9**	**83.5±3.0**	**72.6±1.6**	**75.3±2.0**



(a) Impact of shot number. (b) Impact of label rate.

Figure 2: Impact of shot number and label rate on node classification.

5.2 Few-Shot Node Classification

Overall Performance. The performances of all models for 3/5-shot node classification are reported in Table 1. According to this Table, we have several findings: (1) CGFL outperforms all baseline methods on all datasets, showing the superiority of our model for few-shot node classification; (2) The improvement of CGFL-I over baseline methods ranges from 1.1% to 50% (3-shot) and from 2.1% to 51% (5-shot). Simultaneously, this value of CGFL-T ranges from 4.8% to 55.0% (3-shot) and from 4.5% to 56.6% (5-shot). The significant improvement demonstrates the effectiveness of contrastive pre-training and self-distillation in learning rich node representations from unlabeled graph data and alleviating the label-hungry issue; (3) CGFL-T (or Teacher-T) is better than CGFL-I (or Teacher-I). It indicates that the model gets benefits from unlabeled data in the meta-testing set, thus improving generalization ability over testing data; (4) CGFL gains an additional boost compared with the pretrained teacher model without any label cost in distillation, showing the effectiveness of contrastive distillation.

Impact of Shot Number. In Figure 2a, we show the performance of our model (CGFL-T) under different shot numbers (1 to 5) compared with some selected baselines. It is easy to see that CGFL achieves better accuracy across different shot numbers. The result demonstrates our model's performance is robust for node classification. Note that we only show results on two datasets and the results on the other datasets (i.e., ogbn-arxiv and Tissue-PPI) are shown in the Section C of Appendix. The same goes for the following analyses.

Impact of Training Label Rate. We further evaluate CGFL's performance under different training label rates (10%, 20%, 30%, 40%, 50%, 100%) compared with baseline methods for 3-shot node classification, as shown in Figure 2b. It is easy to see that (1) CGFL has better results across different label rates; (2) CGFL achieves larger improvement over baseline models when label rate becomes lower (e.g., 10%), showing contrastive pre-training and self-distillation of CGFL lead to more significant improvement for label sparsity case.

Impact of Data Augmentation. As a key step for contrastive learning in CGFL, graph augmentation

Table 3: Few-shot graph classification results. The best results are highlighted in bold while the best baseline results are underlined. -I and -T denote inductive and transductive settings of CGFL, respectively. Teacher indicates the CGFL model without knowledge distillation.

Method	Letter-High		Triangles		Reddit-12K		Enzymes	
	5-shot	10-shot	5-shot	10-shot	5-shot	10-shot	5-shot	10-shot
WL [35]	65.27±7.67	68.39±4.69	51.25±4.02	53.26±2.95	40.26±5.17	42.57±3.69	55.78±4.72	58.47±3.84
Graphlet [36]	33.76±6.94	37.59±4.60	40.17±3.18	43.76±3.09	33.76±6.94	37.59±4.60	53.17±5.92	55.30±3.78
AWE [20]	40.60±3.91	42.20±2.87	39.36±3.85	42.58±3.11	30.24±2.34	33.44±2.04	43.75±1.85	45.58±2.11
Graph2Vec [30]	66.12±5.21	68.17±4.26	48.38±3.85	50.16±4.15	27.85±4.21	29.97±3.17	55.88±4.86	58.22±4.30
Diffpool [24]	58.69±6.39	61.59±5.21	64.17±5.87	67.12±4.29	35.24±5.69	37.43±3.94	45.64±4.56	49.64±4.23
CapsGNN [46]	56.60±7.86	60.67±5.24	65.40±6.13	68.37±3.67	36.58±4.28	39.16±3.73	52.67±5.51	55.31±4.23
GIN [48]	65.83±7.17	69.16±5.14	63.80±5.61	67.30±4.35	40.36±4.69	43.70±3.98	55.73±5.80	58.83±5.32
GIN-KNN [48]	63.52±7.27	65.66±8.69	58.34±3.91	61.55±3.19	41.31±2.84	43.58±2.80	57.24±7.06	59.34±5.24
GSM-GCN [1]	68.69±6.50	72.80±4.12	69.37±4.92	73.11±3.94	40.77±4.32	44.28±3.86	54.34±5.64	58.16±4.39
GSM-GAT [1]	69.91±5.90	73.28±3.46	71.40±4.34	75.60±3.67	41.59±4.12	45.67±3.68	55.42±5.74	60.64±3.84
AS-MAML [29]	70.23±1.53	73.19±1.17	71.56±1.17	75.56±2.39	41.90±1.65	45.66±1.11	56.03±1.85	60.79±2.74
Teacher-I	71.43±5.23	73.62±2.93	71.93±3.51	76.21±2.87	42.32±4.48	46.31±3.84	57.53±3.16	61.12±4.80
CGFL-I	72.12±4.88	74.08±3.30	72.34±3.42	76.91±2.98	42.85±4.62	46.89±4.96	57.94±2.85	61.66±4.61
Teacher-T	75.20±4.34	78.35±1.95	78.55±3.75	81.03±3.37	44.80±4.85	48.95±4.03	59.85±2.34	62.30±3.29
CGFL-T	**75.97±5.02**	**79.10±4.23**	**79.32±4.05**	**81.78±3.30**	**45.55±3.67**	**49.32±4.21**	**60.34±4.04**	**62.97±2.92**

plays an important role in affecting model performance. Here we conduct experiments to evaluate the model performance with different augmentation strategies. We consider three graph augmentations - node dropping (**ND**), feature masking (**FM**), edge removing (**ER**), and their combinations for CGFL phase and report their performances in Figure 3. It is easy to find that the combination of three augmentation strategies works better than a single augmentation or the combination of two augmentations. It demonstrates that various graph augmentations are able to generate sufficient contrastive pairs for a better model.

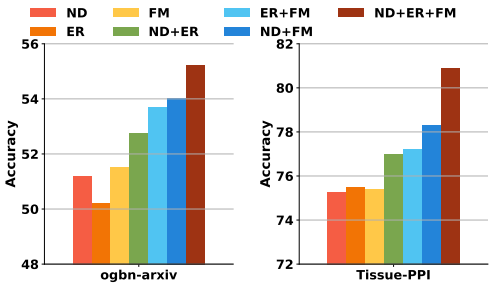


Figure 3: Impact of data aug. on node classification.

Table 2: Info. loss for node classification.

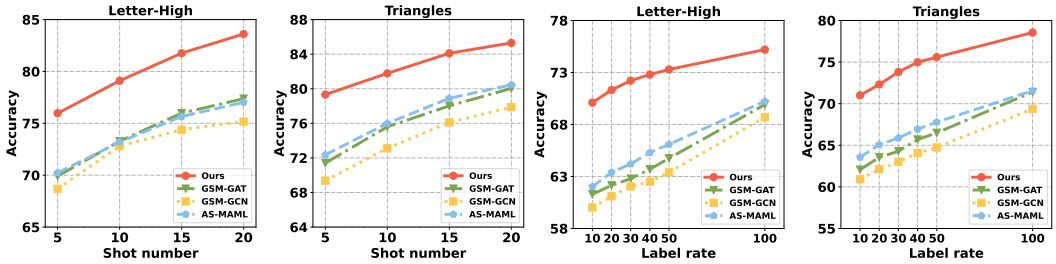
Method	Layer	ogbn-arxiv	Tissue-PPI
G-Meta	GNN-1	385.15	788.30
	GNN-2	383.39	789.78
	FC	374.42	753.85
CGFL-I	GNN-1	305.29	663.38
	GNN-2	300.29	650.59
	FC	290.99	612.98
CGFL-T	GNN-1	**276.30**	**533.86**
	GNN-2	**267.33**	**525.42**
	FC	**264.23**	**503.86**

Loss Information Comparison. In Section 4.3, we propose to compute graph information discarded in the GFL model. Here, we compare results between CGFL and a selected baseline method (G-Meta) in Table 2. From this table, the amount of discarded information in each layer of CGFL is smaller than baseline models. This may be because CGFL can learn more label-irrelevant information from unlabeled graph data, which somehow shows the superiority of CGFL in learning node embeddings for node classification.

5.3 Few-Shot Graph Classification

Overall Performance. The results of all models for 5/10-shot graph classification are reported in Table 3. Similar to the findings obtained from Table 1, according to this table: (1) CGFL outperforms all baseline models, showing its superiority for few-shot graph classification; (2) The improvement of CGFL-I over baseline models ranges from 0.95% to 38.36% (5-shot) and from 0.8% to 34.33% (10-shot). Meanwhile, this value of CGFL-T ranges from 5.65% to 46.21% (5-shot) and from 2.18% to 41.51% (10-shot). This demonstrates the effect of contrastive pre-training and self-distillation in learning rich graph embedding from unlabeled data; (3) CGFL-T (or Teacher-T) outperforms CGFL-I (or Teacher-I), showing that unlabeled data in testing set improves model's generalization; (4) CGFL is better than teacher model as contrastive distillation step further elevates the model.

Impact of Shot Number. In Figure 4a, we report our model's result under different shot numbers (5, 10, 15, 20) compared with some selected baselines. Similar to Figure 2a, CGFL consistently outperforms baseline methods across different shot numbers, showing the robustness of CGFL. Note that we only show results on two datasets (e.g., Letter-High and Triangles), and the results on the other datasets are shown in Section C of Appendix. The same goes for the following analyses.



(a) Impact of shot number.

(b) Impact of label rate.

Figure 4: Impact of shot number and label rate on graph classification.

Impact of Training Label Rate. In Figure 4b, we report our model's performance under different training label rates compared with baseline models for 5-shot graph classification. The phenomenon is similar to the node task: our model achieves better accuracy across different label rates; the performance gaps between CGFL and baseline methods are larger when the label rate is lower. Again, this shows the significance of CGFL when labeled data is limited.

Impact of Data Augmentation. We also study the impact of graph augmentation on few-shot graph classification task. According to Figure 5, we find that the combination of three augmentation strategies brings the best performance, showing the importance of sufficient contrastive pairs on model training.

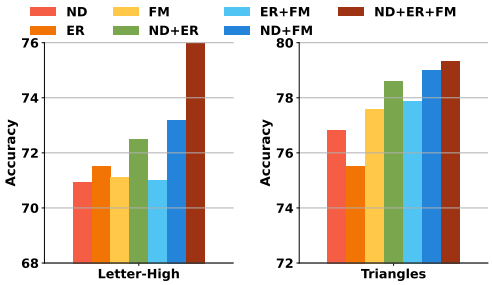


Figure 5: Impact of data aug. on graph classification.

Table 4: Info. loss for graph classification.

Method	Layer	Letter-High	Triangles
GSM-GAT	GNN-1	1286.67	1520.94
	GNN-2	1250.54	1487.42
	FC	1105.96	1364.11
CGFL-I	GNN-1	932.23	1276.77
	GNN-2	920.58	1219.91
	FC-3	892.03	1107.56
CGFL-T	GNN-1	**792.65**	**923.35**
	GNN-2	**780.59**	**894.09**
	FC	**765.73**	**885.35**

Loss Information Comparison. Finally, we compute the discarded graph information of our model and a selected baseline method (GSM-GAT), as shown in Table 4. Obviously, the amount of information discarded in CGFL is less than baseline models. It somehow shows the superiority of CGFL in learning graph embeddings for graph classification.

6 Conclusion

In this paper, to tackle the limitations of existing GFL models, we propose a general and effective framework named CGFL - **S**elf-Distilled **G**raph **F**ew-Shot **L**earning. CGFL leverages a self-distilled contrastive learning procedure to boost GFL, in which the GNN encoder is pre-trained with contrastive learning and further elevated with knowledge distillation in a self-supervised manner. Additionally, we introduce an information-based method to compute the amount of graph information discarded by the GFL model. Extensive experiments on multiple graph datasets demonstrate that CGFL outperforms state-of-the-art baseline methods for both node classification and graph classification tasks in the few-shot scenario. The discarded information value further shows the superiority of CGFL in learning node (or graph) embeddings.

Limitation: Our method has introduced a pretraining step that requires more computation cost. In the future, we plan to design more efficient contrastive pretraining technique which will further benefit GFL with less budget.

References

[1] Jatin Chauhan, Deepak Nathani, and Manohar Kaul. Few-shot learning on graphs via super-classes based on graph spectral measures. In *ICLR*, 2020.

[2] Mingyang Chen, Wen Zhang, Wei Zhang, Qiang Chen, and Huajun Chen. Meta relational learning for few-shot link prediction in knowledge graphs. In *EMNLP-IJCNLP*, 2019.

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

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

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

[6] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *WWW*, 2019.

[7] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. In *NeurIPS*, 2020.

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[9] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *ICML*, 2018.

[10] Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *EMNLP*, 2021.

[11] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. In *NeurIPS*, 2020.

[12] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

[13] Zhichun Guo, Chuxu Zhang, Wenhao Yu, John Herr, Olaf Wiest, Meng Jiang, and Nitesh V Chawla. Few-shot graph learning for molecular property prediction. In *WWW*, 2021.

[14] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[15] Zhongkai Hao, Chengqiang Lu, Zhenya Huang, Hao Wang, Zheyuan Hu, Qi Liu, Enhong Chen, and Cheekong Lee. Asgn: An active semi-supervised graph neural network for molecular property prediction. In *KDD*, 2020.

[16] Kaveh Hassani and Amir Hosein Khas Ahmadi. Contrastive multi-view representation learning on graphs. In *ICML*, 2020.

[17] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020.

[18] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *KDD*, 2020.

[19] Kexin Huang and Marinka Zitnik. Graph meta learning via local subgraphs. In *NeurIPS*, 2020.

[20] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *ICML*, 2018.

[21] Wengong Jin, Connor W Coley, Regina Barzilay, and Tommi Jaakkola. Predicting organic reaction outcomes with weisfeiler-lehman network. In *NeurIPS*, 2017.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[23] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[24] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *ICML*, 2019.

[25] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Zhaoyu Wang, Li Mian, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *arXiv preprint arXiv:2006.08218*, 2020.

[26] Zemin Liu, Yuan Fang, Chenghao Liu, and Steven CH Hoi. Relative and absolute location embedding for few-shot node classification on graph. In *AAAI*, 2021.

[27] Xin Lv, Yuxian Gu, Xu Han, Lei Hou, Juanzi Li, and Zhiyuan Liu. Adapting meta knowledge graph information for multi-hop reasoning over few-shot relations. In *EMNLP-IJCNLP*, 2019.

[28] Haotian Ma, Yinqing Zhang, Fan Zhou, and Quanshi Zhang. Quantifying layerwise information discarding of neural networks. *arXiv preprint arXiv:1906.04109*, 2019.

[29] Ning Ma, Jiajun Bu, Jieyu Yang, Zhen Zhang, Chengwei Yao, Zhi Yu, Sheng Zhou, and Xifeng Yan. Adaptive-step graph meta-learner for few-shot graph classification. In *CIKM*, 2020.

[30] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.

[32] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. GCC: graph contrastive coding for graph neural network pre-training. In *KDD*, 2020.

[33] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

[34] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 2008.

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

[36] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, 2009.

[37] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, 2017.

[38] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *NeuriPS*, 33:596–608, 2020.

[39] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *ICLR*, 2020.

[40] Shixiang Tang, Dapeng Chen, Lei Bai, Kaijian Liu, Yixiao Ge, and Wanli Ouyang. Mutual crf-gnn for few-shot learning. In *CVPR*, 2021.

[41] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[42] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *ICLR*, 2019.

[43] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NeurIPS*, 2016.

[44] Yaqing Wang, Abulikemu Abuduweili, Quanming Yao, and Dejing Dou. Property-aware relation networks for few-shot molecular property prediction. In *NeurIPS*, 2021.

[45] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

[46] Zhang Xinyi and Lihui Chen. Capsule graph neural network. In *ICLR*, 2018.

[47] Wenhan Xiong, Mo Yu, Shiyu Chang, Xiaoxiao Guo, and William Yang Wang. One-shot relational learning for knowledge graphs. In *EMNLP*, 2018.

[48] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[49] Ling Yang, Liangliang Li, Zilun Zhang, Xinyu Zhou, Erjin Zhou, and Yu Liu. Dpgn: Distribution propagation graph network for few-shot learning. In *CVPR*, 2020.

[50] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh Chawla, and Zhenhui Li. Graph few-shot learning via knowledge transfer. In *AAAI*, 2020.

[51] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018.

[52] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, 2018.

[53] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *NeurIPS*, 2020.

[54] Chuxu Zhang, Huaxiu Yao, Chao Huang, Meng Jiang, Zhenhui Li, and Nitesh V Chawla. Few-shot knowledge graph completion. In *AAAI*, 2020.

[55] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. Meta-gnn: On few-shot node classification in graph meta-learning. In *CIKM*, 2019.

[56] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph Contrastive Learning with Adaptive Augmentation. In *WWW*, 2021.

[57] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 2017.

A Dataset Details

For few-shot node classification, we use five different datasets: ogbn-arxiv [17], Tissue-PPI [14], Fold-PPI [57], Cora [34], and Citeseer [34] to perform extensive empirical evaluations of different models. These datasets vary from citation network to biochemical graph. The statistics of datasets are reported in Table 5.

Table 5: Statistics of datasets used in the node classification task.

Dataset	# Graph	# Node	# Edge	# Feat.	# Label
ogbn-arxiv	1	169,343	1,166,243	128	40
Tissue-PPI	24	51,194	1,350,412	50	10
Fold-PPI	144	274,606	3,666,563	512	29
Cora	1	2,708	10,556	1,433	7
Citeseer	1	3,327	9,228	3,703	6

For few-shot graph classification, we use four different datasets [1]: Reddit-12K, ENZYMES, Letter-High, and TRIANGLES, to perform extensive empirical evaluations of different models. These datasets vary from small average graph size (e.g., Letter-High) to large graph size (e.g., Reddit-12K). The statistics of datasets are reported in Table 6.

Table 6: Statistics of datasets used in the graph classification task.

Dataset	Class #		Graph #		
	Train	Test	Training	Validation	Test
Letter-High	11	4	1,330	320	600
Triangles	7	3	1,126	271	603
Reddit-12K	7	4	566	141	404
Enzymes	4	2	320	80	200

B Baseline Method Details

B.1 Node Classification

Graph embedding models:

node2vec [12]: We use node2vec to generate node embeddings, then employ a FC layer as a predictor to classify nodes. We use the code at this link[1].

DeepWalk [31]: Similar to node2vec, we use DeepWalk to generate node embeddings, then employ an FC layer as a predictor to classify nodes. We use the code at this link[2].

GNN-based models:

Meta-GNN [55]: It combines MAML and simple graph convolution (SGC) to learn node embeddings. We use the code at this link[3].

FS-GIN [48]: This method uses GIN to learn node embeddings and only uses few-shot nodes to propagate loss and enable training. We use the code for GIN backbone at this link[4].

FS-SGC [45]: This model is similar to FS-GIN while changing GIN to SGC as GNN backbone. We use the code of SGC at this link[5].

No-Finetune [19]: This method trains a GCN on the support set and uses the trained backbone to classify samples in the meta-testing set. We use the code of GCN at this link[6].

[1]https://shorturl.at/sEINW
[2]https://github.com/phanein/deepwalk
[3]https://github.com/ChengtaiCao/Meta-GNN
[4]https://github.com/weihua916/powerful-gnns
[5]https://github.com/Tiiiger/SGC
[6]https://shorturl.at/lwFPR

Finetune [41]: This method trains GCN on the meta-training set, and the model is fine-tuned on the meta-testing set. We use the code of GCN at this link[6].

KNN [41]: This method trains a GNN on meta-training set. Then, it uses the label of the K-closest examples in the support set for each query example. We use the related code at this link[7].

ProtoNet [41]: This method applies prototypical network on node embeddings processed by a neural network, which is trained under the standard meta-learning setting. We use the related code at this link[8].

MAML [8]: It is similar to ProtoNet but changes meta-learner from ProtoNet to MAML. We use the code at this link[9].

G-Meta [19]: This is state-of-the-art baseline for few-shot node classification. It uses GCN as GNN backbone to learn node embeddings based on local subgraphs. It further combines prototypical loss and MAML for model training. We use the code at this link[9].

B.2 Graph Classification

Graph embedding models:

WL [35]: It uses KNN search on the output embeddings of WL. We use the code of WL at this link[10].

Graphlet [36]: It uses Graphlet Kernel to decompose a graph and generates graph embeddings. We use the code of Graphlet at this link[11].

AWE [20]: It uses KNN search on the output embeddings of AWE. We use the code of AWE at this link[11].

Graph2Vec [30]: This method applies KNN search on the output embeddings of Graph2Vec. We use the code of Graph2Vec at this link[11].

GNN-based models:

Diffpool [24]: It uses Diffpool with supervised loss to generate graph embeddings. We use the code of Diffpool at this link[12].

CapsGNN [46]: This method applies CapsGNN to generate graph embeddings with supervised training. We use the code of CapsGNN backbone at this link[13].

GIN [48]: This model applies GIN to generate graph embeddings with supervised training. We use the code of GIN backbone at this link[4].

GIN-KNN [48]: Similarly, this model implements GIN to generate graph embeddings while it switches the MLP classifier to the KNN algorithm. We use the code of GIN backbone at this link[4].

GSM-GCN [1]: This is a state-of-the-art model (with GCN as backbone) for few-shot graph classification. We follow the default settings in the original paper and use the code at this link[14].

GSM-GAT [1]: This is a state-of-the-art model (with GAT as backbone) for few-shot graph classification. We follow the default settings in the original paper and use the code at this link[14].

AS-MAML [29]: It is a state-of-the-art model for few-shot graph classification. We follow the default settings in the original paper and use the code at this link[15].

[7] https://shorturl.at/etBO8
[8] https://shorturl.at/erQU9
[9] https://github.com/mims-harvard/G-Meta
[10] https://github.com/BorgwardtLab/P-WL
[11] https://github.com/paulmorio/geo2dr
[12] https://github.com/RexYing/diffpool
[13] https://github.com/benedekrozemberczki/CapsGNN
[14] https://github.com/chauhanjatin10/GraphsFewShot
[15] https://github.com/NingMa-AI/AS-MAML

C Additional Experiment Results

This section shows additional results on the other three datasets (i.e., Fold-PPI, Cora, and Citeseer) for few-shot node classification and the other two datasets (i.e., Reddit-12K and Enzymes) for few-shot graph classification.

C.1 Few-Shot Node Classification Results

Impact of Shot Number. Figure 6 shows our model's performance under different shot numbers (1 to 5) compared with some selected baselines. It is easy to see that CGFL consistently outperforms baseline methods across different shot numbers.

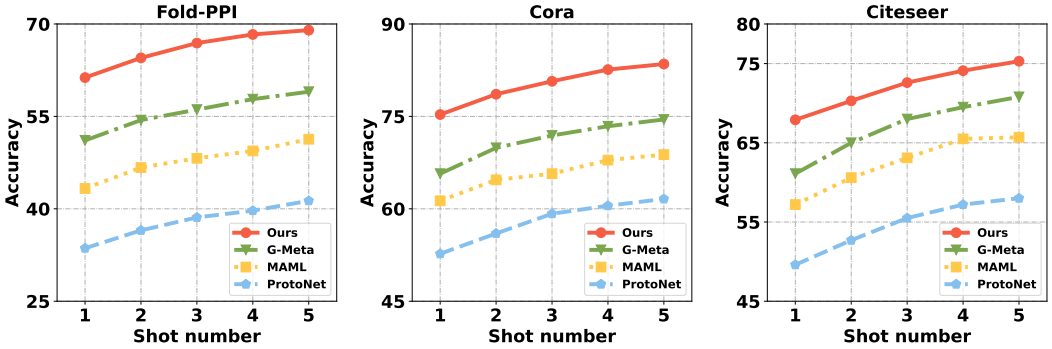


Figure 6: Impact of shot number on node classification.

Impact of Training Label Rate. Figure 7 reports our model's performance under different training label rates compared with baseline models for 3-shot node classification. Obviously, our model achieves better accuracy across different label rates.

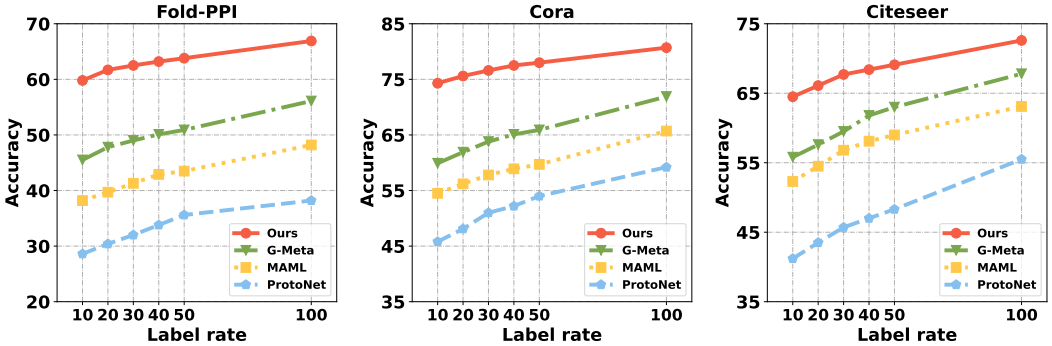


Figure 7: Impact of training label rate on node classification.

Impact of Data Augmentation. The impact of graph augmentation on node classification is shown in Figure 8. According to this figure, we can find that the combination of three augmentation strategies brings the best performance.

Discarded Information Comparison. The discarded graph information of different models are shown in Table 7. Obviously, the amount of information discarded in CGFL is less than baseline models.

C.2 Few-Shot Graph Classification Results

Impact of Shot Number. Figure 9 shows CGFL's performance under different shot numbers (5, 10, 15, 20) compared with some selected baselines. From this figure, our model consistently outperforms baseline methods across different shot numbers.

Impact of Training Label Rate. Figure 10 shows our CGFL's result under different training label rates compared with baseline models for 5-shot graph classification. Obviously, CGFL achieves better performance across different label rates.

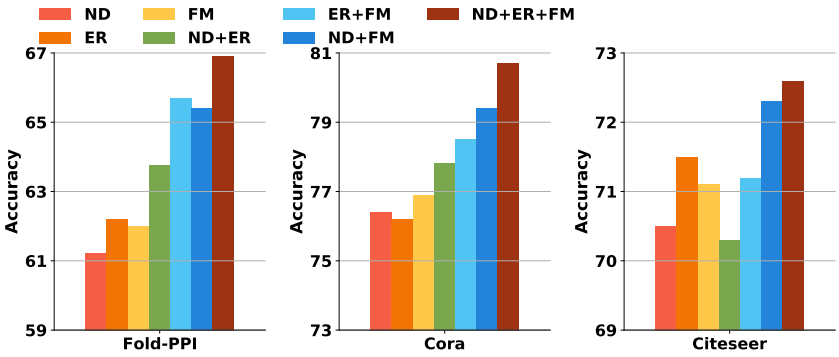


Figure 8: Impact of data augmentation on node classification.

Table 7: Discarded information for node classification.

Method	Layer	Discarded Information		
		Fold-PPI	Cora	Citeseer
Meta-GNN	GNN-1	618.52	361.70	134.20
	GNN-2	612.11	357.47	126.75
	FC	591.13	327.49	116.55
G-Meta	GNN-1	585.31	356.57	129.04
	GNN-2	592.38	355.63	119.72
	FC	553.25	318.87	100.10
CGFL-I	GNN-1	509.10	340.72	111.90
	GNN-2	511.10	326.35	105.01
	FC	461.52	316.21	92.95
CGFL-T	GNN-1	**426.23**	**332.39**	**104.04**
	GNN-2	**418.90**	**314.65**	**88.33**
	FC	**384.23**	**306.98**	**78.25**

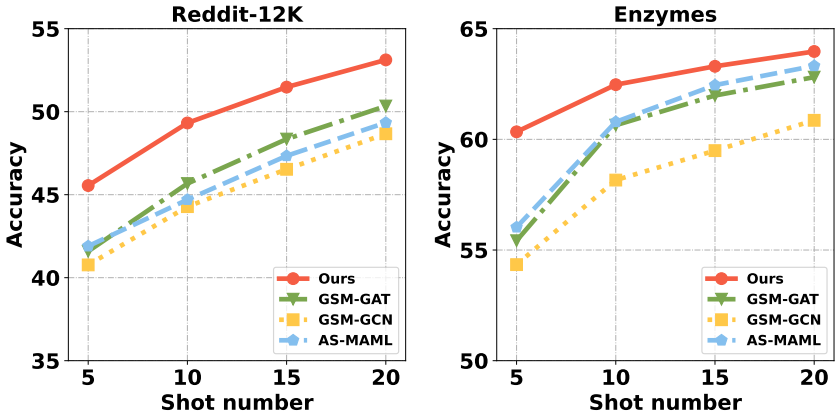


Figure 9: Impact of shot number on graph classification.

Impact of Data Augmentation. The impact of graph augmentation on graph classification is shown in Figure 11. From this figure, we can see that the combination of three augmentation strategies leads to the best result.

Discarded Information Comparison. The discarded graph information of different methods are reported in Table 8. According to this table, the amount of information discarded in CGFL is less than baseline models.

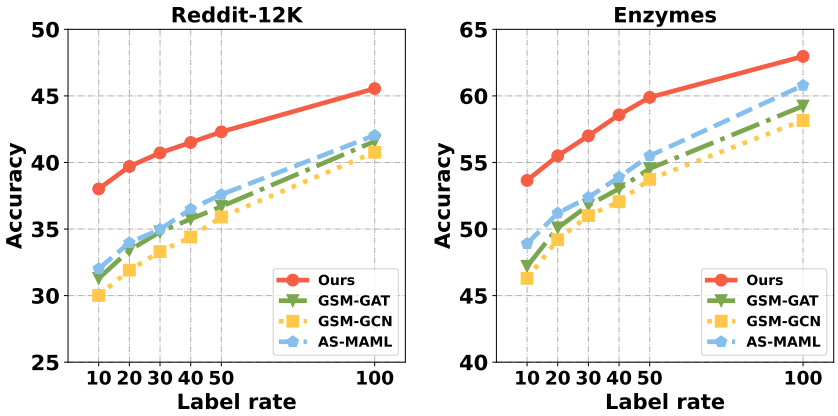


Figure 10: Impact of training label rate on graph classification.

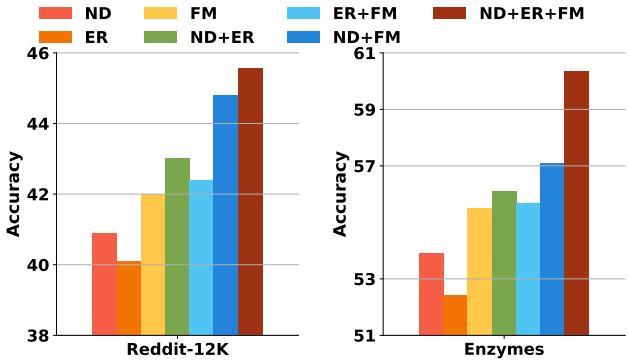


Figure 11: Impact of data augmentation on graph classification.

Table 8: Discarded information for graph classification.

Method	Layer	Discarded Information	
		Reddit-12K	Enzymes
AS-MAML	GNN-1	5455.46	2455.54
	GNN-2	5024.15	2243.25
	FC	4937.99	2125.91
GSM-GAT	GNN-1	5401.30	2273.80
	GNN-2	5183.87	2128.04
	FC	4936.76	1958.18
CGFL-I	GNN-1	5323.04	1864.98
	GNN-2	5098.54	1788.97
	FC	4802.03	1759.87
CGFL-T	GNN-1	**4823.91**	**1787.98**
	GNN-2	**4546.23**	**1698.35**
	FC	**4329.39**	**1585.33**

C.3 Apply to MAML++ and ALFA in Few-shot Learing Phase

We implement MAML++ and ALFA (extended from MAML) in our CGFL to replace MAML under transductive setting on ogbn-arxiv dataset. The results are reported in Table 9. Our aim is to design a contrastive pretraining framework which is plug-and-play for graph few-shot learning. So our CGFL is not limited to MAML. It is easy to be extended with more new methods (e.g., MAML++). Replacing MAML to MAML++ in CGFL brings better accuracy, which implies our framework is general and extendable.

Table 9: Comparison with finetuning by MAML++ and ALFA.

Model	3-shot	5-shot
MAML++	55.9±2.1	59.4±3.2
ALFA	54.8±3.6	58.8±3.7
Ours (with MAML)	55.2±2.5	58.7±2.7

C.4 Apply to Few-shot Image Classification

We have made an attempt to transfer our method to the few-shot image classification task. Specifically, we use ResNet-12 to get image features. The results of our method and a representative GNN-based method [49, 40] on inductive mini-ImageNet 5-way classification are reported in Table 10. It demonstrates that our method can outperform the selected baseline method and has good potential in few-shot image classification task.

Table 10: Comparison with GNN-based method for few-shot image dataset mini-ImageNet.

Model	1-shot	5-shot
DPGN [49]	57.04±0.52	72.83±0.74
MCGN [40]	57.89±0.87	73.58±0.87
Ours	60.02±0.54	76.45±0.88

D Clarification about Related Work

Both motivation and method design of our self-distillation strategy are different from Fixmatch [38]: (a) Our CGFL is motivated by graph few-shot learning problem where label domain distribution shift exists between training data and test data. Thus, we use contrastive self-supervised training to learn more label-irrelevant representation for alleviating label domain distribution shift issue. It is different from FixMatch's aim to learn from partially labeled images; (b) During contrastive pre-training and self-distillation procedures, our model learns representation without labeled graph data while Fixmatch requires partially labeled images during contrastive training.

E Implementation

Our code is provided in this anonymous repository[16].

[16]https://www.dropbox.com/sh/uv3vfc0zfp9yrgt/AACQ427aMkm7FTbsQWbj-q3pa?dl=0