Label Contrastive Coding based Graph Neural Network for Graph Classification

Yuxiang Ren[1⋆], Jiyang Bai[2⋆], and Jiawei Zhang[1]

[1] IFM Lab, Department of Computer Science, Florida State University, FL, USA
yuxiang@ifmlab.org, jiawei@ifmlab.org
[2] Department of Computer Science, Florida State University, FL, USA
bai@cs.fsu.edu

Abstract. Graph classification is a critical research problem in many applications from different domains. In order to learn a graph classification model, the most widely used supervision component is an output layer together with classification loss (e.g.,cross-entropy loss together with softmax or margin loss). In fact, the discriminative information among instances are more fine-grained, which can benefit graph classification tasks. In this paper, we propose the novel **L**abel **C**ontrastive Coding based **G**raph **N**eural **N**etwork (LCGNN) to utilize label information more effectively and comprehensively. LCGNN still uses the classification loss to ensure the discriminability of classes. Meanwhile, LCGNN leverages the proposed *Label Contrastive Loss* derived from self-supervised learning to encourage instance-level intra-class compactness and inter-class separability. To power the contrastive learning, LCGNN introduces a dynamic label memory bank and a momentum updated encoder. Our extensive evaluations with eight benchmark graph datasets demonstrate that LCGNN can outperform state-of-the-art graph classification models. Experimental results also verify that LCGNN can achieve competitive performance with less training data because LCGNN exploits label information comprehensively.

1 Introduction

Applications in many domains in the real world exhibit the favorable property of graph data structure, such as social networks [15], financial platforms [20] and bioinformatics [5]. Graph classification aims to identify the class labels of graphs in the dataset, which is an important problem for numerous applications. For instance, in biology, a protein can be represented with a graph where each amino acid residue is a node, and the spatial relationships between residues (distances, angles) are the edges of a graph. Classification of graphs representing proteins can help predict protein interfaces [5].

Recently, graph neural networks (GNNs) have achieved outstanding performance on graph classification tasks [29, 33]. GNNs aims to transform nodes to low-dimensional dense embeddings that preserve graph structural information

⋆ Two authors contributed equally to this work

and attributes [34]. When applying GNNs to graph classification, the standard method is to generate embeddings for all nodes in the graph and then summarize all these node embeddings to a representation of the entire graph, such as using a simple summation or neural network running on the set of node embeddings [31]. For the representation of the entire graph, a supervision component is usually utilized to achieve the purpose of graph classification. A final output layer together with classification loss (e.g.,cross-entropy loss together with softmax or margin loss) is the most commonly used supervision component in many existing GNNs [29, 28, 32, 6]. This supervision component focuses on the discriminability of class but ignores the instance-level discriminative representations. A recent trend towards learning stronger representations to serve classification tasks is to reinforce the model with discriminative information as more as possible [4]. To be explicit, graph representations, which consider both intra-class compactness and inter-class separability [14], are more potent on the graph classification tasks.

Inspired by the idea of recent self-supervised learning [3] and contrastive learning [7, 18], the contrastive loss [17] is able to extract extra discriminative information to improve the model's performance. The recent works [8, 18, 35] of using contrast loss for representation learning are mainly carried out under the setting of unsupervised learning. These contrastive learning models treat each instance as a distinct class of its own. Meanwhile, discriminating these instances is their learning objective [7]. The series of contrastive learning have been verified effective in learning more fine-grained instance-level features in the computer vision [26] domain. Thus we plan to utilize the contrastive learning on graph classification tasks to make up for the shortcomings of supervision components, that is, ignoring the discriminative information on the instance-level. However, when applying contrastive learning, the inherent large intra-class variations may import noise to graph classification tasks [14]. Besides, existing contrastive learning based GNNs (e.g., GCC [18]) detach the model pre-training and fine-tuning steps. Compared with end-to-end GNNs, the learned graph representations via contrastive learning can hardly be used in the downstream application tasks directly, like graph classification.

To cope with the task of graph classification, we propose the label contrastive coding based graph neural network (LCGNN), which employs *Label Contrastive Loss* to encourage instance-level intra-class compactness and inter-class separability simultaneously. Unlike existing contrastive learning using a single positive instance, the label contrastive coding imports label information and treats instances with the same label as multiple positive instances. In this way, the instances with the same label can be pulled closer, while the instances with different labels will be pushed away from each other. Intra-class compactness and inter-class separability are taken into consideration simultaneously. The label contrastive coding can be regarded as training an encoder for a dictionary look-up task [7]. In order to build an extensive and consistent dictionary, we propose a dynamic label memory bank and a momentum-updated graph encoder inspired by the mechanism [7]. At the same time, LCGNN also uses *Classification Loss*

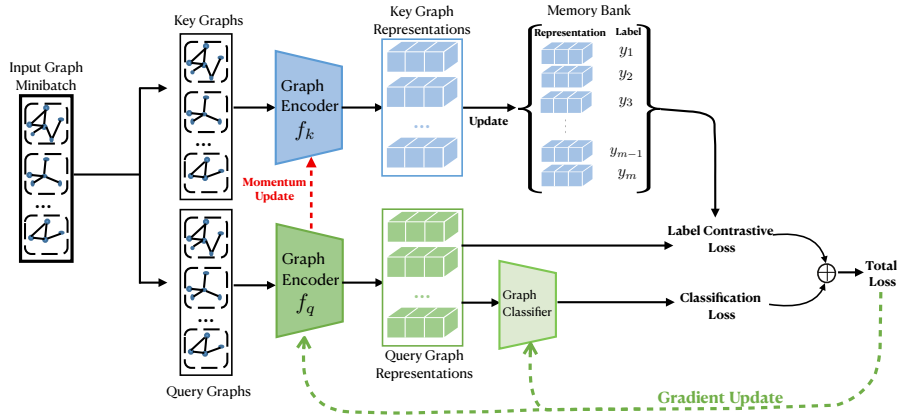


Fig. 1. The high-level structure of LCGNN. LCGNN trains the graph encoder f_q and the graph classifier using a mixed loss. *Label Contrastive Loss* and *Classification Loss* constitute the mixed loss. *Classification Loss* used in LCGNN is cross-entropy loss. *Label Contrastive Loss* is calculated by a dictionary look-up task. The query is each graph of the input graph minibatch, and the dictionary is a memory bank that can continuously update the label-known graph representations. The graph representation in the memory bank is updated by the graph encoder f_k, which is momentum-updated. After training, the learned graph encoder f_q, and the graph classifier can serve for graph classification tasks.

to ensure the discriminability of classes. LCGNN can utilize label information more effectively and comprehensively from instance-level and class-level, allowing using fewer label data to achieve comparative performance, which can be considered as a kind of label augmentation in essence. We validate the performance of LCGNN on graph classification tasks over eight benchmark graph datasets. LCGNN achieves SOTA performance in seven of the graph datasets. What is more, LCGNN outperforms the baseline methods when using less training data, which verifies its ability to learn from label information more comprehensively.

The contributions of our work are summarized as follows:

- We propose a novel label contrastive coding based graph neural network (LCGNN) to reinforce supervised GNNs with more discriminative information.
- The *Label Contrastive Loss* extends the contrastive learning to the supervised setting, where the label information can be imported to ensure intra-class compactness and inter-class separability.
- The momentum-updated graph encoder and the dynamic label memory bank are proposed to support our supervised contrastive learning.
- We conduct extensive experiments on eight benchmark graph datasets. LCGNN not only achieves SOTA performance on multiple datasets but also can offer comparable results with fewer labeled training data.

2 Related Works

Graph Classification Several different techniques have been proposed to solve the graph classification problem. One important category is the kernel-based

method, which learns a graph kernel to measure similarity among graphs to differentiate graph labels [25]. The Weisfeiler-Lehman subtree kernel (WL) [21], Multiscale Laplacian graph kernels (MLG) [13], and Graphlets kernel(GK) [22] are all representative graph kernels. Another critical category is the deep-learning-based method. Deep Graph Kernel (DGK) [30], Anonymous Walk Embeddings (AWE), and Graph2vec [16] all employ the deep-learning framework to extract the graph embeddings for graph classification tasks. With the rise of graph neural networks (GNNs), many GNNs are also used for graph classification tasks by learning the representation of graphs, which will be introduced below.

Graph Neural Network The graph neural network learns the low-dimensional graph representations through a recursive neighborhood aggregation scheme [29]. The derived graph representations can be used to serve various downstream tasks, such as graph classification and top-k similarity search. According to the learning method, the current GNN that can serve graph classification can be divided into end-to-end models and pre-train models. The end-to-end models are usually under supervised or semi-supervised settings, with the goal of optimizing classification loss or mutual information, mainly including GIN [29], CapsGNN [28], DGCNN [32] and InfoGraph [23].The pre-trained GNNs use certain pre-training tasks [9] to learn the graph's general representation under the unsupervised setting. In order to perform graph classification tasks, a part of label data will be used to fine-tuning the models [18].

Contrastive Learning Contrastive learning has been widely used for unsupervised learning by training an encoder that can capture similarity from data. The contrastive loss is normally a scoring function that increases the score on the single matched instance and decreases the score on multiple unmatched instances [17, 26]. In the graph domain, DGI [24] is the first GNN model utilizing the idea of contrastive learning, where the mutual information between nodes and the graph representation is defined as the contrastive metric. HDGI [19] extends the mechanism to heterogeneous graphs. InfoGraph [23] performs contrastive learning in semi-supervised graph-level representation learning. When faced with the task of supervised learning, such as graph classification, we also need to use the advantage of contrastive learning to capture similarity. GCC [18] utilizes contrastive learning to pre-train a model that can serve for the downstream graph classification task by fine-tuning. Compared to them, our method is an end-to-end model and performs label contrastive coding to encourage instance-level intra-class compactness and inter-class separability.

3 Proposed Method

In this section, we introduce the label contrastive coding based graph neural network (LCGNN). Before introducing LCGNN, we provide the preliminaries about graph classification first.

3.1 Preliminaries

The goal of graph classification is to predict class labels of graphs based on the graph structural information and node contents. Formally, we denote it as follows:

Graph Classification Given a set of labeled graphs $\mathbb{G}_L = \{(\mathcal{G}_1, y_1), (\mathcal{G}_2, y_2), \dots\}$ and $y_i \in \mathbb{Y}$ is the corresponding label of $\mathcal{G}_i$. The task is to learn a classification function $f : \mathcal{G} \longrightarrow \mathbb{Y}$ to make predictions for unseen graphs $\mathbb{G}_U$.

3.2 LCGNN Architecture Overview

A learning process illustration of the proposed LCGNN is shown in Figure 1. Usually, for the input graph, we need to extract the latent features that can serve the graph classification through a high-performance graph encoder. In order to cooperate with the proposed mixed loss (*Label Contrastive Loss* & *Classification Loss*), LCGNN contains two graph encoder f_k and f_q, which serve for encoding input key graphs and query graphs respectively. *Label Contrastive Loss* encourages instance-level intra-class compactness and inter-class separability simultaneously by keeping intermediate discriminative representations, while *Classification Loss* ensures the class-level discriminability. A dynamic memory bank containing key graph representations and corresponding labels works for label contrastive loss calculation. A graph classifier takes the representations from the graph encoder f_q as its input to predict the graph labels. In the following parts, we will elaborate on each component and the learning process of LCGNN in detail.

3.3 Label Contrastive Coding

Existing contrastive learning has been proved a success in training an encoder that can capture the universal structural information behind graph data [18]. In the graph classification task, we focus on classification-related structural patterns compared with the universal structural patterns. Therefore, our proposed label contrastive coding learns to discriminate between instances with different class labels instead of treating each instance as a distinct class of itself and contrasting to other distinct classes.

Contrastive learning can be considered as learning an encoder for a dictionary look-up task [7]. We can describe the contrastive learning as follows. Given an encoded query $\mathbf{q}$ and a dictionary containing m encoded keys $\{\mathbf{k}_1, \mathbf{k}_2, \dots, \mathbf{k}_m\}$, there is only a single positive key $\mathbf{k}_+$ (normally encoded from the same instance as $\mathbf{q}$). The loss of this contrastive learning is low when $\mathbf{q}$ is similar to the positive key $\mathbf{k}_+$ while dissimilar to negative keys for $\mathbf{q}$ (all other keys in the dictionary). A widely used contrastive loss is InfoNCE [17] like:

$$\mathcal{L} = -\log \frac{\exp(\mathbf{q} \cdot \mathbf{k}_+/\tau)}{\sum_{i=1}^{m} \exp(\mathbf{q} \cdot \mathbf{k}_i/\tau)} \tag{1}$$

Here, τ is the temperature hyper-parameter [26]. Essentially, the loss of InfoNCE is a classification loss aiming to classify $\mathbf{q}$ from $m = 1$ classes to the same class as $\mathbf{k}_+$.

However, when facing graph classification tasks, the class labels have been determined, and we hope to import known label information in the training data to assist contrastive learning to serve the graph classification task. In this way, we design the label contrastive coding.

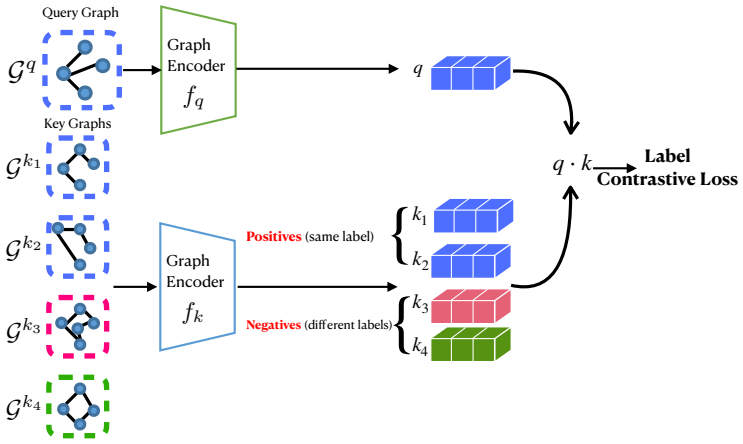


Fig. 2. *Label Contrastive Loss.* The query graph $\mathcal{G}^q$ and key graphs $\mathcal{G}^k$ are encoded by f_q and f_k to low-dimensional representations $\mathbf{q}$ and $\mathbf{k}$ respectively. $\mathbf{k}_1$ and $\mathbf{k}_2$ having the same label as $\mathbf{q}$ are denoted as positive keys. $\mathbf{k}_3$ and $\mathbf{k}_4$ are negative keys due to different labels. The label contrastive loss encourage the model to distinguish the similar pair $(\mathcal{G}^q, \mathcal{G}^{k_1})$ and $(\mathcal{G}^q, \mathcal{G}^{k_2})$ from dissimilar instance pairs, e.g., $(\mathcal{G}^q, \mathcal{G}^{k_3})$.

Define similar and dissimilar In the graph classification task, we seek that instances with the same label can be pulled closer, while instances with different labels will be pushed away from each other. Therefore, in the label contrastive coding, we consider two instances with the same label as a similar pair while treating the pair consisting of different label instances as dissimilar.

Label contrative loss Still from a dictionary look-up perspective, given an labeled encoded query $(\mathbf{q}, y)$, and a dictionary of m encoded labeled keys $\{(\mathbf{k}_1, y_1), (\mathbf{k}_2, y_2), \ldots, (\mathbf{k}_m, y_m)\}$, the positive keys $\mathbf{k}_+$ in label contrastive coding are the keys $\mathbf{k}_i$ where $y_i = y$. The label contrastive coding looks up the positive keys $\mathbf{k}_i$ that the query $\mathbf{q}$ matches in the dictionary. For the encoded query $(\mathbf{q}, y)$, its label contrastive loss $\mathcal{L}_{LC}$ is calculated by

$$\mathcal{L}_{LC}(\mathbf{q}, y) = -\log \frac{\sum_{i=1}^{m} \mathbb{1}_{y_i=y} \cdot \exp(\mathbf{q} \cdot \mathbf{k}_i / \tau)}{\sum_{i=1}^{m} \exp(\mathbf{q} \cdot \mathbf{k}_i / \tau)} \tag{2}$$

Here, $\mathbb{1}_{statement} \in \{0, 1\}$ is a binary indicator that returns 1 if the statement is true. We illustrate the label contrastive loss in Figure 2 for reference. In LCGNN, key graph representations are stored in a dynamic memory bank. For the sake of brevity, we have not shown in Figure 2. We introduce the dynamic memory bank and the updating process next.

The dynamic memory bank In label contrastive coding, the m-size dictionary is necessary. We use a dynamic memory bank to work as a dictionary. In order to fully utilize label information, the size of the memory bank is equal to the size of the set of labeled graphs $\mathbb{G}_L$, i.e., $m = |\mathbb{G}_L|$. The memory bank contains both the encoded low-dimensional key graph representations along with

the corresponding labels, i.e., $\{(\mathbf{k}_1, y_1), (\mathbf{k}_2, y_2), \ldots, (\mathbf{k}_{|\mathbb{G}_L|}, y_{|\mathbb{G}_L|})\}$. Based on the conclusions in MoCo [7], the key graph representations should be kept as consistent as possible when the graph encoder f_k encoder evolves during training. Therefore, in each training epoch, newly encoded key graphs will dynamically replace the old version in the memory bank.

3.4 Graph Encoder Design

For given graphs $\mathcal{G}^q$ and $\mathcal{G}^k$, LCGNN empolys two graph encoders f_q and f_k to encode them to low-dimensional representations.

$$\begin{aligned} \mathbf{q} &= f_q(\mathcal{G}^q) \\ \mathbf{k} &= f_k(\mathcal{G}^k) \end{aligned} \tag{3}$$

In LCGNN, f_q and f_k have the same structure. Graph neural network has proven its powerful ability to encode graph structure data [27]. Many potential graph neural networks can work as the graph encoder in LCGNN.

Two kinds of encoders are considered in LCGNN. The first is Graph Isomorphism Network (GIN) [29]. GIN uses multi-layer perceptrons (MLPs) to conceive aggregation scheme and updates node representations as:

$$h_v^k = \mathrm{MLP}^{(k)}\left((1+\epsilon^{(k)}) + \sum\nolimits_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right) \tag{4}$$

where ϵ is a learnable parameter or a fixed scalar, and k represents k-th layer. Given embeddings of individual nodes, the readout function is proposed by GIN to produce the representation $\boldsymbol{g}$ of the entire graph $\mathcal{G}$ for graph classification tasks:

$$\boldsymbol{g} = \Big\Vert_{k=1}^{K} \left(\mathrm{SUM}(\{h_v^k | v \in \mathcal{G}\})\right) \tag{5}$$

Here, $\|$ is the concatenation operator.

The second encoder we consider is Hierarchical Graph Pooling with Structure Learning (HGP-SL) [33]. HGP-SL incorporates graph pooling and structure learning into a unified module to generate hierarchical representations of graphs. HGP-SL proposes a graph pooling operation to identify a subset of informative nodes to form a new but smaller graph. The details about the Manhattan distance-based pooling operation can be referenced to [33]. For graph $\mathcal{G}$, HGP-SL repeats the graph convolution and pooling operations for several times and achieves multiple subgraphs in different layers: $\mathbf{H}^1, \mathbf{H}^2, \ldots, \mathbf{H}^K$. HGP-SL uses the concatenation of mean-pooling and max-pooling to aggregate all the node representations in the subgraph as follows:

$$\mathbf{r}^k = \mathcal{R}(\mathbf{H}^k) = \sigma\left(\frac{1}{n^k}\sum_{p=1}^{n^k} \mathbf{H}^k(p,:) \,\|\, \max_{q=1}^{d} \mathbf{H}^k(:,1)\right) \tag{6}$$

where σ is a nonlinear activation function. n^k is the node number in the k-th layer subgraph. In order to achieve the final representation $\boldsymbol{g}$ of the entire graph $\mathcal{G}$, another readout function is utilized to combine subgraphs in different layers.

$$\boldsymbol{g} = \mathrm{SUM}(\mathbf{r}^k | k = 1, 2, \ldots, K) \tag{7}$$

In the experiment section, we will show the performance along with the analysis of using GIN and HGP-SL as graph encoders in LCGNN.

3.5 LCGNN Learning

The training process illustration is provided in Figure 1. During the training process, the input of LCGNN is a batch of labeled graphs $\mathbb{G}_b \subset \mathbb{G}_L$. For each mini-batch iteration, the set of key graphs and the set of query graphs are the same as $\mathbb{G}_b$. The graph encoder f_q and f_k will be initialized with the same parameters ($\theta_q = \theta_k$). The memory bank's size is equal to the size of the set of labeled graphs $\mathbb{G}_L$. The labeled graph $\mathcal{G}^i$ with the label y_i is assigned with a random representation to initialize the memory bank. The set of key graphs will be encoded by f_k to low-dimensional key graph representations $\mathbb{K}$, which will replace the corresponding representations in the memory bank. The set of query graphs are encoded by f_q to query graph representations $\mathbb{Q}$, whereas $\mathbb{Q}$ is also the input of the graph classifier. In LCGNN, a logistic regression layer serves as the graph classifier. Based on the output of the graph classifier, *Classification Loss* can be calculated by:

$$\mathcal{L}_{Cla} = -\frac{1}{|\mathbb{Q}|} \sum_{\mathbf{q}_i \in \mathbb{Q}} \sum_{j \in \mathbb{Y}} \mathbb{1}_{\mathbf{q}_i, j} \log(p_{\mathbf{q}_i, j}) \tag{8}$$

where $\mathbb{1}$ is a binary indicator (0 or 1) that indicates whether label j is the correct classification for the encoded query graph $\mathbf{q}_i$. Besides, $p_{\mathbf{q}_i, j}$ is the predicted probability.

$\mathbb{Q}$ and the memory bank work together to implement the label contrastive coding described in previous parts. Based on the Equ. 2, *Label Contrastive Loss* of the mini-batch $\mathbb{G}_b$ is:

$$\mathcal{L}_{LC} = -\frac{1}{|\mathbb{Q}|} \sum_{\mathbf{q}_i \in \mathbb{Q}} \mathcal{L}_{LC}(\mathbf{q}_i, y_{\mathbf{q}_i}) \tag{9}$$

In order to train the model by utilizing label information more effectively and comprehensively, we try to minimize the following mixed loss combining both the *Label Contrastive Loss* and the *Classification Loss*:

$$\mathcal{L}_{total} = \mathcal{L}_{Cla} + \beta \, \mathcal{L}_{LC} \tag{10}$$

Here, the hyper-parameter β controls the relative weight between the label contrastive loss and the classification loss. The motivation behind $\mathcal{L}_{total}$ is that $\mathcal{L}_{LC}$ encourages instance-level intra-class compactness and inter-class separability while $\mathcal{L}_{Cla}$ ensures the discriminability of classes. The graph encoder f_q, and the graph classifier can be updated end-to-end by back-propagation according to the loss $\mathcal{L}_{total}$. The parameters θ_k of f_k follows a momentum-based update

mechanism as MoCo [7] instead of the back-propagation way. Specifically, the momentum updating process is:

$$\theta_k \longleftarrow \alpha\theta_k + (1-\alpha)\theta_q \tag{11}$$

where $\alpha \in [0, 1)$ is the momentum weight to control the speed of f_k evolving. We use this momentum-based update mechanism not only to reduce the overhead of backpropagation but also to keep the key graph representations in the memory bank as consistent as possible despite the encoder's evolution.

After completing the model training, the learned graph encoder f_q along with the graph classifier can be used to perform graph classification tasks for the unlabeled graphs $\mathbb{G}_U$.

4 Experiments

4.1 Experiment Settings

Datasets We test our algorithms on 8 widely used datasets. Three of them are social networks benchmark datasets: IMDB-B, IMDB-M, and COLLAB; the rest five datasets: MUTAG, PROTEINS, PTC, NCI1, and D&D, belong to biological graphs datasets [30, 28, 29]. Each dataset of these contains many graphs, and each graph is assigned with a label. The statistics of these datasets are summarized in Table 1. What should be mentioned is that the biological graphs have categorical node attributes, while social graphs do not come with node attributes. In this paper, for the encoders requiring node attributes as input, we follow [29] to use one-hot encodings of node degrees as the node attributes on datasets without node features.

Table 1. Datasets in the Experiments

Datasets	# graphs	Avg # nodes	Avg # edges	# classes
IMDB-B	1000	19.77	96.53	2
IMDB-M	1500	13.00	65.94	3
COLLAB	5000	74.49	2457.78	3
MUTAG	188	17.93	19.79	2
PROTEINS	1113	39.06	72.82	2
PTC	344	25.56	25.56	2
NCI1	4110	29.87	32.30	2
D&D	1178	284.32	715.66	2

Methods Compared We select three categories of models as our comparison methods:

- Kernel-based method: Weisfeiler-Lehman subtree kernel (**WL**) [21], **AWE** [10], and Deep Graph Kernel (**DGK**) [30]: They first decompose graphs into sub-components based on the kernel definition, then learn graph embeddings in a feature-based manner. For graph classification tasks, a machine learning model (i.e., SVM) will be used to perform the classification with learned graph embeddings.

- Graph embedding-based methods: **Sub2vec** [1], **Graph2vec** [16]: They extend document embedding neural networks to learn representations of entire graphs. A machine learning model (i.e., SVM) work on the classification tasks with learned graph representations.
- Graph neural network methods: **GraphSAGE** [6], **GCN** [12], **DCNN** [2]: They are designed to learn meaningful node level representations. A readout function is empolyed to summarize the node representations to the graph representation for graph-level classification tasks; **DGCNN** [32], **Caps-GNN** [28], **HGP-SL** [33], **GIN** [29], **InfoGraph** [23]: They are GNN-based algorithms with the pooling operator for graph representation learning. Then a classification layer will work as the last layer to implement graph classification; **GCC** [18]: It follows pre-training and fine-tuning paradigm for graph representation learning. A linear classifier is used to support the fine-tuning targeing graph classification; $\mathbf{LCGNN_{GIN}}$, $\mathbf{LCGNN_{HGP\text{-}SL}}$: They are two variants of the proposed LCGNN. LCGNN_{GIN} uses GIN [29] as the graph encoders, and LCGNN_{HGP-SL} sets the graph encoders as HGP-SL [33].

Experiment Configurations We adopt two graph model structures: GIN [29] and HGP-SL [33] as the graph encoders. For LCGNN with different encoders, we follow the model configurations from the initial papers as the default settings. For the LCGNN structure, we choose the hidden representation dimension as 64 and 128 for two respective encoders; the contrastive loss weight $\beta \in \{0.1, 0.6, \ldots, 1.0\}$; the momentum term $\alpha \in [0.0, 1.0)$; the temperature $\tau = 0.07$. For the graph classification tasks, to evaluate the proposed LCGNN we adopt the procedure in [29, 28] to perform 10-fold cross-validation on the aforementioned datasets. For the training process of LCGNN, we select the Adam [11] as the optimizer, and tune the hyperparameters for each dataset as: (1) the batch size $\in \{32, 128, 512\}$; (2) the learning rate $\in \{0.01, 0.001\}$; (3) the dropout rate $\in \{0.0, 0.5\}$; (4) number of training epochs 1000 and select the epoch as the same with [29]. We run the experiments on the Server with 3 GTX- 1080 ti GPUs, and all codes are implemented in Python3. Code and supplementary materials are available at: *Anonymous link*[3].

4.2 Experimental Results and Analysis

Overall evaluation We present the main experimental results in Table 2. For the graph datasets that comparison methods do not have the results in the original papers, we denote it as "−". From the table, we can observe that LCGNN outperforms all comparison methods on 7 out of the total 8 datasets. The improvement is especially evident on datasets such as IMBD-B and D&D, which can be up to about 1.0%. At the same time, we can find that LCGNN using different graph encoders have achieved SOTA performance on different datasets (LCGNN_{GIN} in 3 datasets; LCGNN_{HGP-SL} in 4 datasets). The results also show that for different data sets, the selection of graph encoders has a critical impact on performance Nonetheless LCGNN generally outperforms all other baselines methods.

[3] https://www.dropbox.com/sh/kc7xf42kz4lqx9a/AAC9wKim768TBNocN1JNPudFa?dl=0

Table 2. Test Sets Classification Accuracy on All Datasets. We use **bold** to denote the best result on each dataset.

Categories	**Methods**	IMDB-B	IMDB-M	COLLAB	MUTAG	PROTEINS	PTC	NCI1	D&D
Kernels	WL	73.4 ± 4.6	49.3 ± 4.8	79.0 ± 1.8	82.1 ± 0.4	76.2 ± 4.0	–	76.7 ± 2.0	76.4 ± 2.4
	AWE	74.5 ± 5.9	51.5 ± 3.6	73.9 ± 1.9	87.9 ± 9.8	–	–	–	71.5 ± 4.0
	DGK	67.0 ± 0.6	44.6 ± 0.5	73.1 ± 0.3	87.4 ± 2.7	75.7 ± 0.5	60.1 ± 2.5	80.3 ± 0.5	73.5 ± 1.0
Graph Embedding	Graph2vec	71.1 ± 0.5	50.4 ± 0.9	–	83.2 ± 9.3	73.3 ± 1.8	60.2 ± 6.9	73.2 ± 1.8	–
	Sub2vec	55.2 ± 1.5	36.7 ± 0.8	–	61.0 ± 15.8	–	60.0 ± 6.4	–	–
GNNs	DCNN	72.4 ± 3.6	49.9 ± 5.0	79.7 ± 1.7	79.8 ± 13.9	65.9 ± 2.7	–	74.7 ± 1.3	–
	GCN	73.3 ± 5.3	51.2 ± 5.1	$\mathbf{80.1 \pm 1.9}$	87.2 ± 5.1	75.2 ± 3.6	–	76.3 ± 1.8	73.3 ± 4.5
	GraphSAGE	72.4 ± 3.6	49.9 ± 5.0	79.7 ± 1.7	79.8 ± 13.9	65.9 ± 2.7		74.7 ± 1.3	–
	DGCNN	70.0 ± 0.9	47.8 ± 0.9	73.8 ± 0.5	85.8 ± 1.7	75.5 ± 0.9	58.6 ± 2.5	74.4 ± 0.5	79.4 ± 0.9
	CapsGNN	73.1 ± 4.8	50.3 ± 2.7	79.6 ± 0.9	86.7 ± 6.9	76.3 ± 3.6	–	78.4 ± 1.6	75.4 ± 4.2
	HGP-SL	–	–	–	82.2 ± 0.6	84.9 ± 1.6	–	78.5 ± 0.8	81.0 ± 1.3
	GIN	75.1 ± 5.1	52.3 ± 2.8	80.2 ± 1.9	89.4 ± 5.6	76.2 ± 2.8	64.6 ± 7.0	82.7 ± 1.7	–
	InfoGraph	73.0 ± 0.9	49.7 ± 0.5	–	89.0 ± 1.1	–	61.7 ± 1.4	–	–
	GCC	73.8	50.3	81.1	–	–	–	–	–
Proposed	LCGNN_{GIN}	$\mathbf{76.1 \pm 6.9}$	$\mathbf{52.4 \pm 6.7}$	72.3 ± 6.3	89.9 ± 4.8	76.9 ± 6.8	64.7 ± 2.0	$\mathbf{82.9 \pm 3.6}$	77.4 ± 1.2
	LCGNN_{HGP-SL}	75.4 ± 1.5	46.5 ± 1.3	77.5 ± 1.2	$\mathbf{90.5 \pm 2.3}$	$\mathbf{85.2 \pm 2.4}$	$\mathbf{65.9 \pm 2.8}$	78.8 ± 4.4	$\mathbf{81.8 \pm 3.6}$

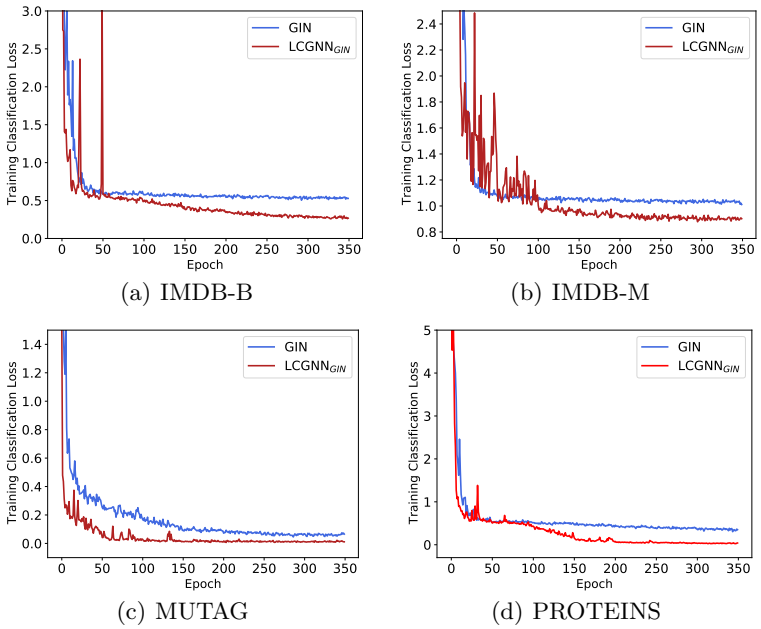
We also note that, compared to the baseline methods GIN and HGP-SL, LCGNN$_{GIN}$ and LCGNN$_{HGP-SL}$ can acquire better results when adopting them as corresponding encoders. For the COLLAB dataset results, LCGNN actually achieves much higher performance compared with the result we get when running GIN source code (71.7 ± 3.5). However, the result reported by the original paper [29] is 80.1 ± 1.9, which we also report in Table 2. To further evaluate the advantages of LCGNN and highlight the effectiveness of *Label Contrastive Loss*, we compare the classification loss during the training processes and show the curves of GIN and LCGNN in Figure 3. From the figure, we can see that not only LCGNN has a faster convergence rate, but also can finally converge to lower classification loss. The classification loss comparison results on other datasets are also consistent, but we did not show them all due to the space limitation. Thus we can conclude that with the support of label contrastive coding, LCGNN has better potential on graph classification tasks.

Besides, through the comparison between GCC and LCGNN, we can find that for the task of graph classification, The proposed label contrastive coding shows more advantages than the contrastive coding in GCC. We believe that the contrastive coding in GCC mainly focuses on learning universal representations. The label contrastive coding in LCGNN has a stronger orientation for representation learning, that is, extracting features that significantly affect on the intra-class compactness and inter-class separability.

Table 3. Experiments with Less Labeled Training Data

Datasets	Methods	Training Ratio				
		60%	70%	80%	90%	100%
IMDB-B	GIN	61.8	65.4	69.2	70.5	75.1
	LCGNN$_{GIN}$	**66.3**	**70.8**	**71.3**	**72.2**	**76.1**
IMDB-M	GIN	40.5	41.4	41.8	46.0	52.3
	LCGNN$_{GIN}$	**43.4**	**42.8**	**43.6**	**48.1**	**52.4**

Performance with less labeled data To validate our claim that LCGNN can utilize label information more comprehensively and use fewer label data to achieve comparative performance, we conduct experiments with less training data. For each fold of cross-validation, we extract only part of the training set (e.g., 60% of the data in the training set) as the training data and maintain the test set as the same. We present the results in Table 3. In Table 3, the training ratio denotes how much data in the training set is extracted as the training data. When the training ratio is 100%, it means using the full training set in each fold. From the results, it is obvious that LCGNN$_{GIN}$ can always outperform the baseline GIN when using less training data. What's more, in many cases when LCGNN$_{GIN}$ with less training data (e.g., 70% training data for LCGNN$_{GIN}$ while 80% for GIN; 60% for LCGNN$_{GIN}$ while 70% for GIN), LCGNN$_{GIN}$ still obtains more competitive results than GIN. The experimental results demonstrate that LCGNN can utilize the same amount of training data more comprehensively and efficiently. The capability also makes LCGNN possible to learn with less training data to obtain a better performance than comparison methods when they need more training data.



(a) IMDB-B (b) IMDB-M

(c) MUTAG (d) PROTEINS

Fig. 3. Training *Classification Loss* versus Training Epoch

The effectiveness of the label contrastive coding In order to further verify the effectiveness of the proposed label contrastive coding on the task of graph classification, we conduct comparison experiments between LCGNN$_{GIN}$ and LCGNN+InfoNCE. Here, LCGNN+InfoNCE replaces the label contrastive loss in LCGNN$_{GIN}$ with InfoNCE loss [17] but keeps other parts the same. We present the results in Figure 4. The experimental results show that the performance of LCGNN$_{GIN}$ on all data sets exceeds LCGNN+InfoNCE, which also demonstrates that the label contrastive coding can effectively utilize label information to improve model performance. In addition, we observe that the performance of LCGNN+InfoNCE is even worse than GIN. It verifies that the inherent large intra-class variations may import noise to graph classification tasks if we treat the intra-class instances as distinct classes like the existing comparative learning.

Table 4. LCGNN$_{GIN}$ with Different Contrastive Loss Weight β

Datasets	Contrastive Loss Weight β							
	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
IMDB-B	73.8	75.1	**76.1**	75.5	76.0	75.4	75.7	75.7
IMDB-M	50.5	51.2	**52.4**	51.9	51.7	51.5	51.5	51.6

Hyper-parameter β analysis We consider the influence of label contrastive loss weight term β and conduct experiments with different values. The results is exhibited in Table 4. We select β from $\{0.1, 0.2, \ldots, 1.0\}$, and find the trend of using a relatively larger β inducing better results. Thus in the experiment, we empirically select from $\beta \in \{0.5, 0.6, \ldots, 1.0\}$ to achieve the best performance. Nevertheless, we also observed that when β gradually increases, the performance does not continue to increase. Our analysis is that when the label contrastive

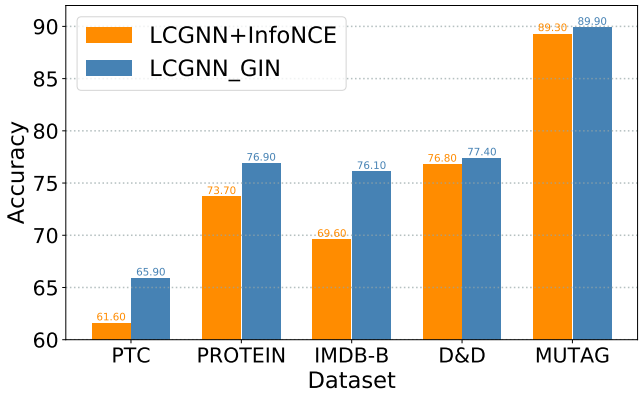


Fig. 4. The effectiveness of *Label Contrastive Loss*

loss weight is too high, the learning of the model places too much emphasis on instance-level contrast. More fine-grained discriminative features on the instance-level will reduce the generalization performance of the model on the test set.

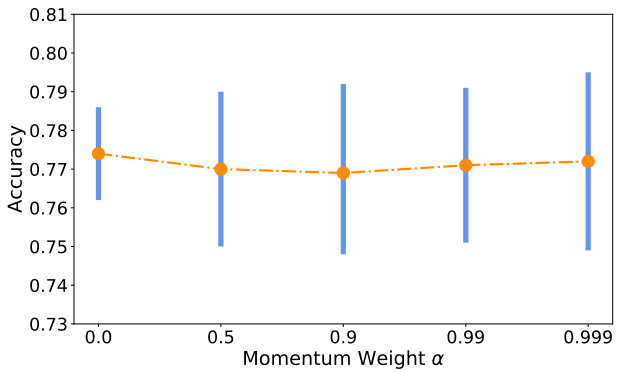


Fig. 5. LCGNN with Different Momentum Weight

Momentum ablation The momentum term plays an important role in contrastive learning problems. In our experiments, we also try different momentum weight α when running LCGNN$_{GIN}$ on D&D and show the results in Figure 5. Unlike [7], LCGNN$_{GIN}$ also achieves good performance when $\alpha = 0$. The main reason should be that the D&D is not extremely large, which makes it easy for representations to ensure consistency during encoder evolving. Furthermore, in this set of experiments, the momentum term did not show much impact on Accuracy, that is, the model performance is relatively stable, which should be caused by the moderate-sized dataset as well.

5 Conclusion

In this paper, we have introduced a novel label contrastive coding based graph neural network, LCGNN, which works on graph classification tasks. We extend the existing contrastive learning to the supervised setting and define the label contrastive coding. The label contrastive coding treats instances with the same

label as multiple positive instances, which is different from the single positive instance in unsupervised contrastive learning. The label contrastive coding can pull the same label instances closer and push the instances with different labels away from each other. We demonstrate the effectiveness of LCGNN on graph classification tasks over eight benchmark graph datasets. The experimental results show that LCGNN achieves SOTA performance in 7 datasets. Besides, LCGNN can take advantage of label information more comprehensively. LCGNN outperforms the baseline method when using less training data, which verifies this advantage.

References

1. Adhikari, B., Zhang, Y., Ramakrishnan, N., Prakash, B.A.: Sub2vec: Feature learning for subgraphs. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 170–182. Springer (2018)
2. Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. In: Advances in neural information processing systems. pp. 1993–2001 (2016)
3. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. arXiv preprint arXiv:2002.05709 (2020)
4. Elsayed, G., Krishnan, D., Mobahi, H., Regan, K., Bengio, S.: Large margin deep networks for classification. In: Advances in neural information processing systems. pp. 842–852 (2018)
5. Fout, A., Byrd, J., Shariat, B., Ben-Hur, A.: Protein interface prediction using graph convolutional networks. In: Advances in neural information processing systems. pp. 6530–6539 (2017)
6. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in neural information processing systems. pp. 1024–1034 (2017)
7. He, K., Fan, H., Wu, Y., Xie, S., Girshick, R.: Momentum contrast for unsupervised visual representation learning. arXiv pp. arXiv–1911 (2019)
8. Hjelm, R.D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., Bengio, Y.: Learning deep representations by mutual information estimation and maximization. arXiv preprint arXiv:1808.06670 (2018)
9. Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., Leskovec, J.: Strategies for pre-training graph neural networks. arXiv preprint arXiv:1905.12265 (2019)
10. Ivanov, S., Burnaev, E.: Anonymous walk embeddings. In: International Conference on Machine Learning. pp. 2186–2195 (2018)
11. Kingma, D.P., Ba, J.L.: Adam: A method for stochastic optimization. In: International Conference on Learning Representaion (2015)
12. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representaion (2017)
13. Kondor, R., Pan, H.: The multiscale laplacian graph kernel. In: Advances in Neural Information Processing Systems. pp. 2990–2998 (2016)
14. Liu, W., Wen, Y., Yu, Z., Yang, M.: Large-margin softmax loss for convolutional neural networks. In: ICML. vol. 2, p. 7 (2016)
15. Meng, L., Ren, Y., Zhang, J., Ye, F., Philip, S.Y.: Deep heterogeneous social network alignment. In: 2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI). pp. 43–52. IEEE (2019)

16. Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., Jaiswal, S.: graph2vec: Learning distributed representations of graphs. arXiv preprint arXiv:1707.05005 (2017)
17. Oord, A.v.d., Li, Y., Vinyals, O.: Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748 (2018)
18. Qiu, J., Chen, Q., Dong, Y., Zhang, J., Yang, H., Ding, M., Wang, K., Tang, J.: Gcc: Graph contrastive coding for graph neural network pre-training. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1150–1160 (2020)
19. Ren, Y., Liu, B., Huang, C., Dai, P., Bo, L., Zhang, J.: Heterogeneous deep graph infomax. arXiv preprint arXiv:1911.08538 (2019)
20. Ren, Y., Zhu, H., Zhang, J., Dai, P., Bo, L.: Ensemfdet: An ensemble approach to fraud detection based on bipartite graph. arXiv preprint arXiv:1912.11113 (2019)
21. Shervashidze, N., Schweitzer, P., Van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**(9) (2011)
22. Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., Borgwardt, K.: Efficient graphlet kernels for large graph comparison. In: Artificial Intelligence and Statistics. pp. 488–495 (2009)
23. Sun, F.Y., Hoffmann, J., Verma, V., Tang, J.: Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. arXiv preprint arXiv:1908.01000 (2019)
24. Veličković, P., Fedus, W., Hamilton, W.L., Liò, P., Bengio, Y., Hjelm, R.D.: Deep graph infomax. arXiv preprint arXiv:1809.10341 (2018)
25. Vishwanathan, S., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. The Journal of Machine Learning Research **11**, 1201–1242 (2010)
26. Wu, Z., Xiong, Y., Yu, S., Lin, D.: Unsupervised feature learning via non-parametric instance-level discrimination. arXiv preprint arXiv:1805.01978 (2018)
27. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems (2020)
28. Xinyi, Z., Chen, L.: Capsule graph neural network. In: International conference on learning representations (2018)
29. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? arXiv preprint arXiv:1810.00826 (2018)
30. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1365–1374 (2015)
31. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: Advances in neural information processing systems. pp. 4800–4810 (2018)
32. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
33. Zhang, Z., Bu, J., Ester, M., Zhang, J., Yao, C., Yu, Z., Wang, C.: Hierarchical graph pooling with structure learning. arXiv preprint arXiv:1911.05954 (2019)
34. Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., Wang, L.: Deep graph contrastive representation learning. arXiv preprint arXiv:2006.04131 (2020)
35. Zhuang, C., Zhai, A.L., Yamins, D.: Local aggregation for unsupervised learning of visual embeddings. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 6002–6012 (2019)