# Contrastive Self-supervised Learning for Graph Classification

**Jiaqi Zeng**[1] and **Pengtao Xie**[2*]
[1] Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
[2] Department of Electrical and Computer Engineering, University of California San Diego, USA
Gabyyyyyy@sjtu.edu.cn, p1xie@eng.ucsd.edu

## Abstract

Graph classification is a widely studied problem and has broad applications. In many real-world problems, the number of labeled graphs available for training classification models is limited, which renders these models prone to overfitting. To address this problem, we propose two approaches based on contrastive self-supervised learning (CSSL) to alleviate overfitting. In the first approach, we use CSSL to pretrain graph encoders on widely-available unlabeled graphs without relying on human-provided labels, then finetune the pretrained encoders on labeled graphs. In the second approach, we develop a regularizer based on CSSL, and solve the supervised classification task and the unsupervised CSSL task simultaneously. To perform CSSL on graphs, given a collection of original graphs, we perform data augmentation to create augmented graphs out of the original graphs. An augmented graph is created by consecutively applying a sequence of graph alteration operations. A contrastive loss is defined to learn graph encoders by judging whether two augmented graphs are from the same original graph. Experiments on various graph classification datasets demonstrate the effectiveness of our proposed methods. The code is available at https://github.com/UCSD-AI4H/GraphSSL.

## Introduction

Graph classification (Zhang et al. 2019; Di et al. 2019) is a widely studied problem in machine learning and data mining and finds broad applications. For example, given a molecule graph of a protein, judge whether this protein is non-enzyme. Given a chemical compound graph, judge whether the compound is mutagen or non-mutagen. In many real-world graph classification problems, the number of graphs available for training is oftentimes limited. For instance, it is difficult to obtain a lot of protein graphs in many biomedical studies due to the financial cost. It is well known that when the amount of training data is limited, the model tends to overfit to the training data and perform less well on test data.

To address the overfitting problem in graph classification, we propose two approaches: CSSL-Pretrain and CSSL-Reg, both based on contrastive self-supervised learning (CSSL) (He et al. 2019; Chen et al. 2020a,b). In CSSL-Pretrain, we use CSSL to pretrain graph encoders on widely-available unlabeled graphs without relying on human-provided labels, then finetune the pretrained encoders on labeled graphs. In CSSL-Reg, we develop a regularizer based on CSSL, and solve the supervised classification task and the unsupervised CSSL task simultaneously. Self-supervised learning (SSL) (Gidaris, Singh, and Komodakis 2018; Pathak et al. 2016; Zhang, Isola, and Efros 2016) is an unsupervised learning approach which defines auxiliary tasks on input data without using any human-provided labels and learns data representations by solving these auxiliary tasks. Contrastive SSL (He et al. 2019; Chen et al. 2020b,a) creates augmentations of original data examples and defines an auxiliary task which judges whether two augmented data examples originate from the same original data example. Recently, several self-supervised learning approaches (Peng et al. 2020; Qiu et al. 2020; Sun et al. 2019) are proposed for representation learning on graphs. These approaches focus on learning representations of local elements in graphs, such as nodes and subgraphs. In contrast, our method focuses on learning graph-level representations that are more suitable for tasks like graph classification.

To perform CSSL on graphs, we first create augmented graphs from the original graphs, based on four basic graph alteration operations including edge deletion, edge addition, node deletion, and node addition. To create an augmented graph, we apply a sequence of graph alteration operations consecutively: the operation at step $t$ is applied to the intermediate graph generated after applying the operation at step $t-1$. Given the augmented graphs, we define a CSSL task to distinguish whether two augmented graphs are created from the same original graph. In CSSL-Pretrain, we first pretrain a graph encoder by solving the graph CSSL task, then use this pretrained encoder as initialization and continue to finetune it by minimizing the graph classification loss. CSSL-Pretrain learns powerful graph representations on unlabeled graphs (which are widely available) in an unsupervised way without relying on human-provided labels. Since these representations are learned without using labels, they are less likely to be overfitted to the labels in the small-sized training dataset and hence help to reduce overfitting. In CSSL-Reg, the CSSL loss serves as a regularization term and is optimized jointly with the classification loss. CSSL-Reg enforces the graph encoder to jointly solve two tasks: an un-

[*] Corresponding Author

[*] Corresponding Author

supervised CSSL task and a supervised graph classification task. Due to the presence of the CSSL task, the model is less likely to be biased to the classification task defined on the small-sized training data. We perform experiments on five datasets. Our proposed CSSL-Pretrain and CSSL-Reg outperform baseline approaches, which demonstrate the effectiveness of our methods in alleviating overfitting.

The major contributions of this paper are as follows:

- We propose CSSL-Pretrain, which is an unsupervised pretraining method of graph encoders based on contrastive self-supervised learning, to learn graph representations that are resilient to overfitting.
- We propose CSSL-Reg, which is a data-dependent regularizer based on CSSL, to reduce the risk that the graph encoder is biased to the data-deficient classification task on the small-sized training data.
- Experiments on various datasets demonstrate the effectiveness of our approaches.

The rest of the paper is organized as follows. Section 2 reviews related works. Section 3 and 4 present the methods and experiments respectively. Section 5 concludes the paper and discusses future works.

# Related Works

## Graph Representation Learning

In graph applications, learning useful representations of nodes, edges, and the entire graph is crucial for downstream applications such as graph classification, node classification, graph completion, etc. Classic approaches for graph representation learning can be categorized as: (1) embedding methods: for example, DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) leveraged truncated random walk to learn node embeddings, LINE (Tang et al. 2015) used edge sampling to learn node embeddings in large-scale graphs, HARP (Chen et al. 2017) utilized hierarchical representation learning to capture global structures in graphs; (2) matrix-factorization-based methods: for example, NetMF (Qiu et al. 2018) discovered a theoretical connection between DeepWalk's implicit matrix and graph Laplacians and proposed an embedding approach based on this connection, HOPE (Ou et al. 2016) proposed an asymmetric transitivity preserving graph representation learning method for directed graphs.

Recently, graph neural networks (GNNs) have achieved remarkable performance for graph modeling. GNN-based approaches can be classified into two categories: spectral approaches and message-passing approaches. The spectral approaches generally use graph spectral theory to design parameterized filters. Based on Fourier transform on graphs, Bruna et al. (2013) defined convolution operations for graphs. To reduce the heavy computational cost of graph convolution, Defferrard, Bresson, and Vandergheynst (2016) utilized fast localized spectral filtering. Graph convolution network (GCN) (Kipf and Welling 2016) truncated the Chebyshev polynomial to the first-order approximation of the localized spectral filters. The message-passing approaches basically aggregate the neighbours' information through convolution operations. GAT (Veličković et al. 2017) leveraged attention mechanisms to aggregate the neighbours' information with different weights. GraphSAGE (Hamilton, Ying, and Leskovec 2017) generalized representation learning to unseen nodes using neighbours' information. Graph pooling methods such as DiffPool (Ying et al. 2018) and HGP-SL (Zhang et al. 2019) were developed to aggregate node-level representations into graph-level representations.

## Contrastive Self-supervised Learning

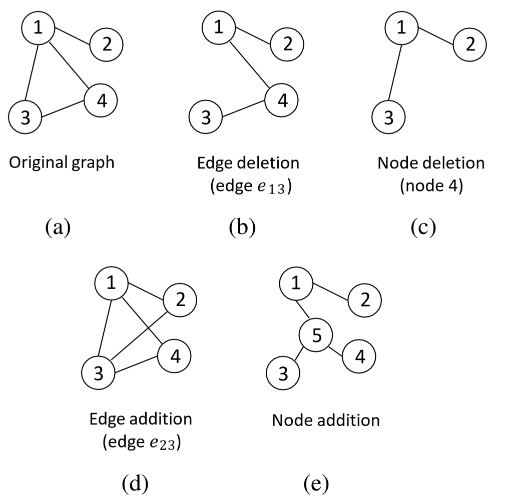
Contrastive self-supervised learning (He et al. 2019; Chen et al. 2020a) has arisen much research interest recently and has been widely applied for image classification (He et al. 2019; Chen et al. 2020a), text classification (Zhou, Li, and Xie 2021; Fang et al. 2020), visual question answering (He et al. 2020a), etc. MoCo (He et al. 2019) and SimCLR (Chen et al. 2020a) learned image encoders by predicting whether two augmented images were created from the same original image. Hénaff et al. (2019) studied data-efficient image recognition based on contrastive predictive coding (Oord, Li, and Vinyals 2018), which predicted the future in latent space by using powerful autoregressive models. Srinivas, Laskin, and Abbeel (2020) proposed to learn contrastive unsupervised representations for reinforcement learning. Khosla et al. (2020) investigated supervised contrastive learning, where clusters of points belonging to the same class were pulled together in embedding space, while clusters of samples from different classes were pushed apart. Klein and Nabi (2020) proposed a contrastive self-supervised learning approach for commonsense reasoning. He et al. (2020b); Yang et al. (2020) proposed a Self-Trans approach which applied contrastive self-supervised learning on top of networks pretrained by transfer learning.

## Self-supervised Learning on Graphs

Recently, several self-supervised learning approaches are proposed for representation learning on graphs. Peng et al. (2020) learned node representations by randomly selecting pairs of nodes in a graph and training a neural net to predict the contextual position of one node relative to the other. GCC (Qiu et al. 2020) defined the pre-training task as subgraph instance discrimination in and across networks and leveraged contrastive learning to learn structural representations. InfoGraph (Sun et al. 2019) defined SSL tasks which maximize mutual information between graph representations and sub-structural representations. These approaches focused on learning representations of local elements in graphs, such as nodes and subgraphs. In contrast, our method focuses on learning graph-level representations that are more suitable for tasks like graph classification.

# Methods

To alleviate overfitting in graph classification, we propose two methods based on contrastive self-supervised learning (CSSL): CSSL-Pretrain and CSSL-Reg. In CSSL-Pretrain, we use CSSL to pretrain the graph encoder. In CSSL-Reg, we use the CSSL task to regularize the graph encoder.



(a) (b) (c)

(d) (e)

Figure 1: Graph alteration operations.

## Contrastive Self-supervised Learning on Graphs

In this section, we discuss how to perform contrastive self-supervised learning on graphs, which is the basis of CSSL-Pretrain and CSSL-Reg. Self-supervised learning (SSL) (Gidaris, Singh, and Komodakis 2018; Pathak et al. 2016; Zhang, Isola, and Efros 2016) is a learning paradigm that aims to capture the intrinsic patterns and properties of input data without using human-provided labels. The basic idea of SSL is to construct some auxiliary tasks solely based on the input data itself without using human-annotated labels and make the network to learn meaningful representations by performing the auxiliary tasks well, such as rotation prediction (Gidaris, Singh, and Komodakis 2018), image inpainting (Pathak et al. 2016), automatic colorization (Zhang, Isola, and Efros 2016), context prediction (Nathan Mundhenk, Ho, and Chen 2018), etc. The auxiliary tasks in SSL can be constructed using many different mechanisms. Recently, a contrastive mechanism (Hadsell, Chopra, and LeCun 2006) has gained increasing attention and demonstrated promising results in several studies (He et al. 2019; Chen et al. 2020b). The basic idea of contrastive SSL is: generate augmented examples of original data examples, create a predictive task that predicts whether two augmented examples are from the same original data example or not, and learn the representation network by solving this task.

To perform CSSL on graphs, given a collection of original graphs, we perform graph augmentation to generate augmented graphs from the original graphs, then learn a network to predict whether two augmented graphs originate from the same original graph or not. To perform graph augmentation, we use four types of basic graph alteration operations, as illustrated in Figure 1. The four types of operations include:

- **Edge deletion**: randomly select an edge and remove it from the graph. For example, in Figure 1(b), we randomly select an edge (which is the one between node 1 and 3), and delete it.

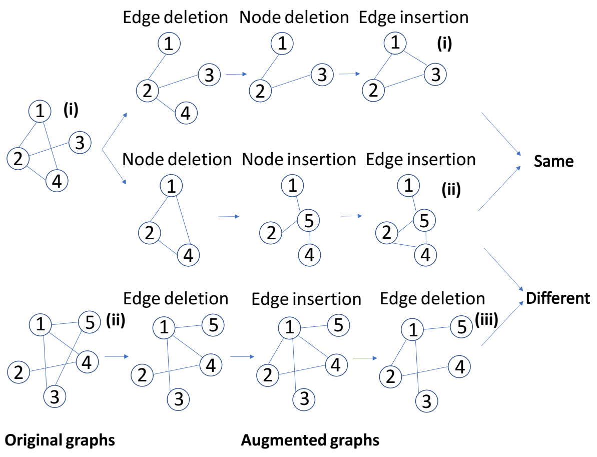


Figure 2: Illustration of contrastive SSL on graphs.

- **Node deletion**: randomly select a node and remove it from the graph; remove all edges connecting to this node. For example, in Figure 1(c), we randomly select a node (which is 4), delete this node and all edges connected with node 4.
- **Edge addition**: randomly select two nodes, if they are not directly connected but there is a path between them, add an edge between these two nodes. For example, in Figure 1(d), node 2 and 3 are not directly connected, but there is a path between them ($2 \rightarrow 1 \rightarrow 3$). We connect these two nodes with an edge.
- **Node addition**: randomly select a strongly-connected subgraph $S$, remove all edges in $S$, add a node $n$, and add an edge between $n$ and each node in $S$. For example, in Figure 1(e), node 1, 3, 4 form a complete subgraph. We insert a new node 5, connect node 1, 3, 4 to node 5, and remove the edges among node 1, 3, 4.

Given an original graph $G$, to create an augmentation of $G$, we apply a sequence of graph alteration operations consecutively. At step 1, we randomly sample an operation $o_1(\cdot)$ that is applicable to $G$, perform this operation and get an altered graph $G_1 = o_1(G)$. At step 2, we randomly sample another operation $o_2(\cdot)$ that is applicable to $G_1$, perform this operation and get $G_2 = o_2(G_1)$. This procedure continues until the maximum number of steps is reached. At each step $t$, an applicable operation is randomly sampled and applied to the intermediate graph $G_{t-1}$ generated at step $t-1$.

Next, we define the contrastive learning loss on augmented graphs. If two augmented graphs are created from the same original graph, they are labeled as being similar; otherwise, they are labeled as dissimilar. Augmented graphs created from different original graphs (OGs) could be the same. Though it is possible that augmented graphs created from different original graphs (OGs) could be the same, the probability is very low since augmentation operations are applied randomly. We learn a network to fit these similar/dissimilar binary labels. The network consists of two modules: a graph embedding module $f(\cdot)$ which extracts the latent representation $\mathbf{h} = f(\mathbf{x})$ of a graph $\mathbf{x}$ and a multi-

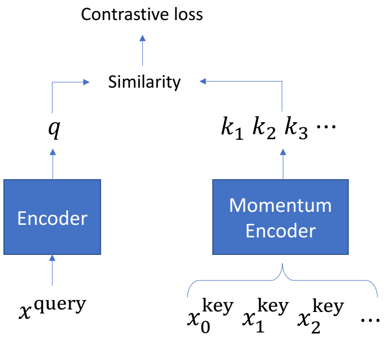


Figure 3: Illustration of MoCo.

layer perceptron $g(\cdot)$ which takes $\mathbf{h}$ as input and generates another latent representation $\mathbf{z} = g(\mathbf{h})$ used for predicting whether two graphs are similar. Given a similar pair $(\mathbf{x}_i, \mathbf{x}_j)$ and a set of graphs $\{\mathbf{x}_k\}$ that are dissimilar from $\mathbf{x}_i$, a contrastive loss (Hadsell, Chopra, and LeCun 2006; Chen et al. 2020a) can be defined as follows:

$$-\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau) + \sum_k \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \quad (1)$$

where $\text{sim}(\cdot, \cdot)$ denotes cosine similarity between two vectors and $\tau$ is a temperature parameter.

Figure 2 presents an illustrative example. Three augmented graphs (AGs) are created from two original graphs (OGs): AG (i) and (ii) are from OG (i); AG (iii) is from OG (ii). To create AG (i), three random alteration operations are performed consecutively, including edge deletion, node deletion, and edge addition. Each operation is applied to the intermediate graph resulting from the last operation. AG (ii) is created by applying node deletion, node addition, and edge addition. AG (iii) is created by applying edge deletion, edge addition, and edge deletion. AG (i) and (ii) are labeled as "similar" since they originate from the same original graph. AG (ii) and (iii) are labeled as "dissimilar" since they are created from different original graphs.

We use MoCo (He et al. 2019) to perform efficient optimization of the loss in Eq.(1), based on a queue that is independent of minibatch size. This queue contains a dynamic set of augmented graphs (called keys). In each iteration, the latest minibatch of graphs are added into the queue; meanwhile, the oldest minibatch is removed from the queue. In this way, the queue is decoupled from minibatch size. Figure 3 shows the architecture of MoCo. The keys are encoded using a momentum encoder. Given an augmented graph (called a query) in the current minibatch and a key in the queue, they are considered as a positive pair if they originate from the same graph, and a negative pair if otherwise. A similarity score is calculated between the encoding of the query and the encoding of each key. Contrastive losses are defined on the similarity scores and binary labels.

## CSSL-based Pretraining

Having presented CSSL on graphs, we study two approaches of using graph CSSL for alleviating overfitting in graph classification. The first approach is to use graph CSSL to pretrain a graph encoder and use this pretrained encoder to initialize the graph classification model. We call this approach CSSL-Pretrain. Given a collection of unlabeled graphs, we define a graph CSSL task on these graphs, then perform this task using a network consisting of a graph encoder and a CSSL-specific prediction head. The head is a multi-layer perceptron which takes graph representations generated by the graph encoder as inputs and predicts whether two augmented graphs are similar. After training, the CSSL-specific prediction head is discarded. Next, we finetune the pretrained graph encoder in the graph classification task. The graph classification network consists of a graph encoder and a classification head. The classification head takes the graph representations generated by the encoder as inputs and predicts the class label. We use the encoder pretrained by CSSL to initialize the encoder in the classification model and continue to train it on the original graphs and their class labels.

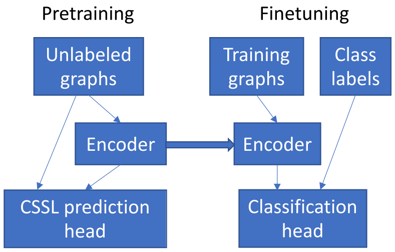


Figure 4: Illustration of CSSL-Pretrain.

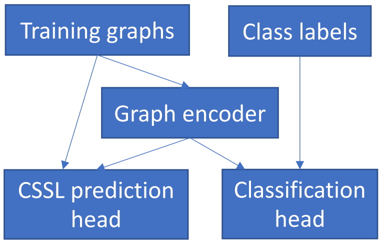


Figure 5: Illustration of CSSL-Reg.

## CSSL-based Regularization

The second approach we propose is CSSL-Reg, where we use the graph CSSL task to regularize the graph classification model. Given the training graphs, we encode them using a graph encoder. Then on top of the graph encodings, two tasks are defined. One is the classification task, which takes the encoding of a graph as input and predicts the class label of this graph. The prediction is conducted using a classification head. The other task is graph CSSL. Given the augmented graphs stemming from the training graphs, CSSL predicts whether two augmented graphs are from the same original graph. The loss of the CSSL task serves as a data-dependent regularizer to alleviate overfitting. The CSSL task has a predictive head. The two tasks share the same graph encoder. Formally, CSSL-Reg solves

| Dataset | PT* | D&D | NCI1 | NCI109 | Mut** |
|---|---|---|---|---|---|
| # classes | 2 | 2 | 2 | 2 | 2 |
| # train | 890 | 942 | 3288 | 3301 | 3469 |
| # validation | 111 | 117 | 411 | 412 | 433 |
| # test | 112 | 119 | 411 | 414 | 435 |
| Avg. # nodes | 39.1 | 284.3 | 29.9 | 29.7 | 30.3 |
| Avg. # edges | 72.8 | 715.7 | 32.3 | 32.1 | 30.8 |

Table 1: Statistics of datasets. *PT denotes PROTEINS. **Mut denotes Mutagenicity.

the following optimization problem:

$$\mathcal{L}^{(c)}(D, L; \mathbf{W}^{(e)}, \mathbf{W}^{(c)}) + \lambda \mathcal{L}^{(p)}(D, \mathbf{W}^{(e)}, \mathbf{W}^{(p)}) \quad (2)$$

where $D$ represents the training graphs and $L$ represents their labels. $\mathbf{W}^{(e)}$, $\mathbf{W}^{(c)}$, and $\mathbf{W}^{(p)}$ denote the graph encoder, classification head in the classification task, and prediction head in the CSSL task respectively. $\mathcal{L}^{(c)}$ denotes the classification loss and $\mathcal{L}^{(p)}$ denotes the CSSL loss. $\lambda$ is a tradeoff parameter.

## Graph Encoder

At the core of CSSL-Pretrain and CSSL-Reg is to better learn a graph encoder using CSSL. Our methods can be used to learn any graph encoder. In this work, we perform the study using the Hierarchical Graph Pooling with Structure Learning (HGP-SL) encoder (Zhang et al. 2019), while noting that other graph encoders are also applicable. HGP-SL is composed of interleaving layers of graph convolution and graph pooling. Graph convolution learns multiple layers of latent embeddings of each node in the graph by leveraging the embeddings of neighboring nodes. The graph pooling operation selects a subset of informative nodes to form a subgraph. A node is considered less informative if its representation can be well reconstructed by those of its neighbors. Given the structure of the pooled subgraph, HGP-SL performs structure learning to refine the structure of the subgraph. HGP-SL calculates the similarity of two nodes in the subgraph and connects them if the similarity score is large enough. Given the refined subgraph, graph convolution and pooling are conducted again. The layers of convolution, pooling, and structure refinement repeat multiple times. A readout function is used to aggregate representations of individual nodes into a single representation of the graph. A multi-layer perceptron serves as the classification head to predict the class label from the graph-level representation.

# Experiments

## Dataset

We used 5 graph classification datasets[1] in the experiments. Each data example consists of a graph and a class label. In PROTEINS and D&D, each graph represents a protein. A binary label is associated with each graph, representing whether the protein is a non-enzyme. NCI1 and NCI109 contain graphs representing chemical compounds with labels denoting whether they can inhibit the growth of cancer cells. The graphs in Mutagenicity represent chemical compounds. Each graph is labeled as mutagen or non-mutagen. We randomly split each dataset into three parts: 80% for training, 10% for validation, and 10% for testing. Pretraining is only performed on training datasets. The random split is repeated for 10 times and the average performance with standard deviation is reported. The statistics of these datasets are summarized in Table 1.

## Experimental Setup

**CSSL-Pretrain** For CSSL pretraining, the queue size in MoCo is set as 1024 for the D&D and PROTEINS dataset, and 4096 for the NCI1, NCI109, and Mutagenicity dataset. The MoCo momentum is set as 0.999 and the temperature $\tau$ is set as 0.07. The initial learning rate is searched in $\{1e^{-3}, 1e^{-4}, 1e^{-5}\}$ and decayed with the cosine decay schedule (Loshchilov and Hutter 2016). We find it beneficial to utilize a small batch size (16 or 32), a small learning rate ($1e^{-5}$), and train for more epochs ($1k \sim 3k$).

For finetuning the classification model, we search the initial learning rate in $\{1e^{-2}, 1e^{-3}, 1e^{-4}\}$ and utilize the Adam optimizer (Kingma and Ba 2014) to optimize the model. Following (Zhang et al. 2019), we adopt early stopping based on the validation loss. Specifically, we stop training if the validation loss does not decrease for 100 consecutive epochs. We select the model with the smallest validation loss as the final model.

**CSSL-Reg** We search the regularization parameter $\lambda$ in $\{1, 0.1, 0.01, 0.001, 0.0001\}$. The Adam optimizer is used and the initial learning rate is searched in $\{1e^{-2}, 1e^{-3}, 1e^{-4}\}$. We set the queue size in Moco as 512 for the D&D and PROTEINS dataset, and 2048 for the NCI1, NCI109, and Mutagenicity dataset. The settings of batch size, patience for early stopping, MoCo momentum, and temperature $\tau$ are the same as those in CSSL-Pretrain.

**Graph Encoder** Following (Zhang et al. 2019), the dimension of node representation is set to 128. The number of HGP-SL layers is set as 3. The pooling ratio is searched in $[0.1, 0.9]$ and the dropout ratio is searched in $[0.0, 0.5]$.

## Baselines

We compare with the following categories of baselines.

- **Graph Kernel Methods.** This category of methods compares the similarity of two graphs in a kernel space and performs classification based on the similarity between graphs. We compare with three algorithms: GRAPHLET (Shervashidze et al. 2009), Shortest-Path (SP) Kernel (Borgwardt and Kriegel 2005), and Weisfeiler-Lehman (WL) Kernel (Shervashidze et al. 2011).
- **Graph Neural Networks.** GCN (Kipf and Welling 2016), GraphSAGE (Hamilton, Ying, and Leskovec 2017), and GAT (Veličković et al. 2017) are three GNN models designed for learning node representations in graphs. Node representations are aggregated into a representation of the

[1] Datasets are publicly available at https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

| Categories | Method | PROTEINS | D&D | NCI1 | NCI109 | Mutagenicity |
|---|---|---|---|---|---|---|
| Kernels | GRAPHLET | 72.23±4.49 | 72.54±3.83 | 62.48±2.11 | 60.96±2.37 | 56.65±1.74 |
| | SP | 75.71±2.73 | 78.72±3.89 | 67.44±2.76 | 67.72±2.28 | 71.63±2.19 |
| | WL | 76.16±3.99 | 76.44±2.35 | 76.65±1.99 | 76.19±2.45 | 80.32±1.71 |
| GNNs | GCN | 75.17±3.63 | 73.26±4.46 | 76.29±1.79 | 75.91±1.84 | 79.81±1.58 |
| | GraphSAGE | 74.01±4.27 | 75.78±3.91 | 74.73±1.34 | 74.17±2.89 | 78.75±1.18 |
| | GAT | 74.72±4.01 | 77.30±3.68 | 74.90±1.72 | 75.81±2.68 | 78.89±2.05 |
| Pooling | Set2Set | 79.33±0.84 | 70.83±0.84 | 69.62±1.32 | 73.66±1.69 | 80.84±0.67 |
| | DGCNN | 79.99±0.44 | 70.06±1.21 | 74.08±2.19 | 78.23±1.31 | 80.41±1.02 |
| | DiffPool | 79.90±2.95 | 78.61±1.32 | 77.73±0.83 | 77.13±1.49 | 80.78±1.12 |
| | EigenPool | 78.84±1.06 | 78.63±1.36 | 77.24±0.96 | 75.99±1.42 | 80.11±0.73 |
| | gPool | 80.71±1.75 | 77.02±1.32 | 76.25±1.39 | 76.61±1.39 | 80.30±1.54 |
| | SAGPool | 81.72±2.19 | 78.70±2.29 | 77.88±1.59 | 75.74±1.47 | 79.72±0.79 |
| | EdgePool | 82.38±0.82 | 79.20±2.61 | 76.56±1.01 | 79.02±1.89 | 81.41±0.88 |
| | HGP-SL | 84.91±1.62 | 80.96±1.26 | 78.45±0.77 | 80.67±1.16 | 82.15±0.58 |
| Self-supervised | InfoGraph | 75.18±0.51 | 74.24±0.86 | 70.93±1.78 | 75.70±1.51 | 72.32±1.70 |
| | GCC-freezing | 74.48±3.12 | 75.63±3.22 | 66.33± 2.65 | 66.18±3.83 | 68.11±2.78 |
| | GCC-finetuning | 69.49±1.42 | 75.46±2.44 | 71.00±1.78 | 69.90±1.04 | 74.43±1.35 |
| CSSL-Freeze | A1-specific | 84.64±0.96 | 78.74±0.92 | 72.60±1.43 | 76.40±0.54 | 77.03±0.66 |
| | A1-all | 78.57±1.64 | 75.96±1.60 | 72.02±1.32 | 75.19±1.00 | 77.08±0.63 |
| | A3-specific | 80.36±1.99 | 78.49±0.94 | 72.70±1.94 | 76.42±0.71 | 77.08±0.48 |
| | A3-all | 76.34±1.92 | 77.73±1.73 | 71.56±0.93 | 75.70±1.16 | 76.85±0.84 |
| CSSL-Pretrain | A1-specific | 85.71±0.69 | 82.02±1.42 | 78.62±0.63 | 80.72±1.06 | 82.00±0.63 |
| | A1-all | 81.79±1.50 | 80.84±1.24 | 78.03±1.14 | 77.51±1.37 | 82.23±0.73 |
| | A3-specific | 82.77±1.70 | 80.84±1.54 | 79.44±0.67 | 81.01±1.01 | 82.41±0.59 |
| | A3-all | 81.07±1.63 | 80.25±1.41 | 78.71±0.80 | 79.87±1.06 | **82.64±0.83** |
| CSSL-Reg | A1-specific | 84.11±0.87 | **82.18±1.34** | 80.07±0.60 | **81.16±1.42** | 82.07±0.65 |
| | A1-all | 83.57±1.07 | 80.50±1.34 | 79.32±0.75 | 77.80±1.46 | 80.83±1.66 |
| | A3-specific | **85.80±1.01** | 79.66±1.71 | **80.09±1.07** | 79.69±1.70 | 81.61±1.05 |
| | A3-all | 81.61±1.61 | 79.58±1.41 | 78.64±0.76 | 79.18±0.87 | 82.23±1.04 |

Table 2: Graph Classification Accuracy (%). "A1" denotes performing one random graph alteration operation to obtain an augmented graph and "A3" denotes performing three consecutive random alteration operations to obtain an augmented graph. "Specific" denotes using the training graphs in the target dataset to define CSSL losses and "all" denotes using training graphs in all the five datasets to define CSSL losses.

entire graph via a readout function and the graph representation is subsequently used for graph classification.

- **Graph Pooling Methods.** Approaches in this group combine graph neural networks with pooling mechanisms. We compare with eight pooling algorithms, including two global pooling algorithms: Set2Set (Vinyals, Bengio, and Kudlur 2015) and DGCNN (Zhang et al. 2018), and six hierarchical graph pooling methods: DiffPool (Ying et al. 2018), EigenPool (Ma et al. 2019), gPool (Gao and Ji 2019), SAGPool (Lee, Lee, and Kang 2019), EdgePool (Diehl 2019), and HGP-SL (Zhang et al. 2019).
- **Self-supervised Learning Methods.** We compare with InfoGraph (Sun et al. 2019) which maximizes the mutual information between the graph-level representation and the representations of substructures at different scales and GCC (Qiu et al. 2020) where the SSL task is subgraph instance discrimination. The data and protocol used for pretraining and finetuning in GCC and InfoGraph are the same as our methods.
- **CSSL-Freeze.** We compare with the following setting called CSSL-Freeze. Given a collection of unlabeled graphs, we train the graph encoder using CSSL. Then the graph encoder is directly plugged into the graph classification model without further finetuning. When training the graph classification model, only the classification head is trained and the weights of the graph encoder are frozen.

| Datasets | PT* | D&D | NCI1 | NCI109 | Mut** |
|---|---|---|---|---|---|
| HGP-SL | 7.4 | 15.6 | 7.8 | 3.6 | 5.2 |
| CSSL-Pretrain | 7.6 | 11.3 | 3.6 | 3.7 | 3.4 |
| CSSL-Reg | 8.3 | 2.6 | 4.1 | 1.8 | 3.0 |

Table 3: L1 difference between training accuracy and testing accuracy. *PT denotes PROTEINS. **Mut denotes Mutagenicity.

## Results

The performance on graph classification is reported in Table 2. From this table, we make the following observations. **First**, CSSL-Reg and CSSL-Pretrain outperform baseline approaches for graph classification. This demonstrates the effectiveness of our methods in alleviating overfitting. To further confirm this, we measure the difference between accuracy on the training set and test set in Table 3. A larger difference implies more overfitting: performing well on the training set and less well on the test set. As can be seen, in most cases, the train-test difference under CSSL-(Pretrain,Reg) is smaller than that under HGP-SL, which demonstrates that our approaches can better alleviate overfitting. CSSL-Pretrain leverages widely-available unlabeled graphs to learn better graph representations that are robust to overfitting. CSSL-Reg encourages the graph encoder to solve an additional task which reduces the risk of overfitting to the data-deficient classification task on the small-sized training data. **Second**, our methods outperform other self-supervised learning methods in the literature. This is because our methods learn a holistic representation of the entire graph by judging whether two augmented graphs originate from the same graph. To successfully make such a judgment, the encoder needs to capture the global features of the entire graph. However, in baseline SSL methods, self-supervision is performed locally at individual nodes, which loses the global picture on the entire graph. Therefore, the learned representations are not suitable for classifying the entire graph. **Third**, on 4 out of the 5 datasets, CSSL-Reg performs better than CSSL-Pretrain. In Table 2, the train-test difference under CSSL-Reg is smaller than that under CSSL-Pretrain, which implies that CSSL-Reg can better prevent overfitting. This is because in CSSL-Reg, the encoder is learned to perform the classification task and CSSL task simultaneously. Thus the encoder is not completely biased to the classification task. In CSSL-Pretrain, the encoder is first learned by performing the CSSL task, then finetuned by performing the classification task. There is a risk that after finetuning, the encoder is largely biased to the classification task on the small-sized training data, which leads to overfitting. **Fourth**, performing CSSL on all graphs in the five datasets yields worse accuracy than CSSL on a single target dataset. This is counter-intuitive because it is expected that more data helps to learn better representations in CSSL. One possible reason is that the five datasets have large domain discrepancy. Using graphs from different domains to pretrain the encoder may render the encoder biased to those domains and eventually generalizes less well on the target domain. **Fifth**, CSSL-Pretrain works better than CSSL-Freeze. This is because in CSSL-Pretrain, the encoder is finetuned using the class labels after pretrained using CSSL. The finetuning can make the encoder more discriminative and suitable for solving the classification problem. In CSSL-Freeze, the encoder is not finetuned. As a result, it may not be optimal for the classification task. **Six**, on 3 out of the 5 datasets, applying three consecutive random operations yields better results than applying one operation only. The reason is that applying three operations makes the augmented graphs more difficult to judge whether they are from the same original graph. Solving a more difficult task makes the learned representations more robust and effective.

Figure 6 shows how the classification accuracy varies as we increase the regularization parameter $\lambda$ in CSSL-Reg. As can be seen, starting from 0, when the regularizer parameter is increasing, the accuracy increases. This is because a larger $\lambda$ imposes a stronger regularization effect, which helps to reduce overfitting. However, if $\lambda$ becomes too large, the accuracy drops. This is because the regularization effect is too strong, which dominates the classification loss.

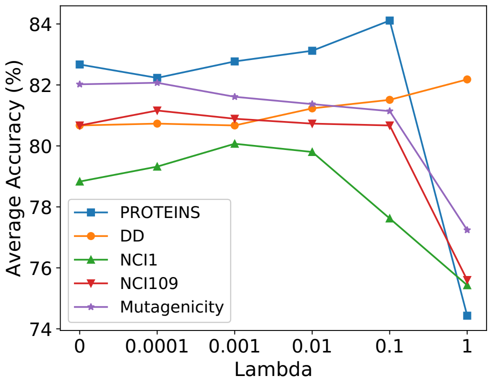


Figure 6: How the regularization parameter in CSSL-Reg affects graph classification accuracy.

| Datasets | PT* | D&D | NCI1 | NCI109 | Mut** |
|---|---|---|---|---|---|
| Edge Deletion | 78.4 | 80.3 | 77.5 | 79.1 | 77.6 |
| Node Deletion | 80.0 | 79.6 | 76.4 | 77.8 | 78.4 |
| Edge Addition | 78.8 | 80.1 | 76.0 | 75.9 | 81.7 |
| Node Addition | 77.8 | 79.8 | 78.4 | 76.8 | **82.1** |
| Random | **84.1** | **82.2** | **80.1** | **81.2** | **82.1** |

Table 4: Performance of CSSL-Reg with deterministic selection of graph alteration operation. *PT denotes PROTEINS. **Mut denotes Mutagenicity.

We also perform a study to verify the importance of randomly selecting graph alteration operations during graph augmentation. We compare with the following deterministic selection setting. For each type of operation including edge addition, edge deletion, node addition, and node deletion, we create augmented graphs by applying this operation once. Table 4 shows the average classification accuracy for each operation in CSSL-Reg. As can be seen, the performance of deterministic selection is worse than random selection. The reason is that augmented graphs created by randomly applying alteration operations are more difficult to judge whether they are from the same original graph. Solving a more challenging CSSL task can help to learn representations that are more effective and robust.

## Conclusions and Future Works

In this paper, we propose to use contrastive self-supervised learning to alleviate overfitting in graph classification problems. We propose two approaches based on CSSL. The first approach defines a CSSL task on widely-available unlabeled graphs and pretrains the graph encoder by solving the CSSL task. The second approach defines a regularizer based on CSSL and the graph encoder is trained to simultaneously minimize the classification loss and the regularizer. We demonstrate the effectiveness of our methods on various graph classification datasets.

For future works, we will develop other self-supervised learning methods on graphs, such as by predicting which augmented graph is closer to the original graph.

## Acknowledgments

This work is supported by a gift award from Tencent AI Lab. We would like to thank the anonymous reviewers for their constructive feedback which helps to improve this work.

## References

Borgwardt, K. M.; and Kriegel, H.-P. 2005. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* .

Chen, H.; Perozzi, B.; Hu, Y.; and Skiena, S. 2017. Harp: Hierarchical representation learning for networks. *arXiv preprint arXiv:1706.07845* .

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G. 2020a. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709* .

Chen, X.; Fan, H.; Girshick, R.; and He, K. 2020b. Improved Baselines with Momentum Contrastive Learning. *arXiv preprint arXiv:2003.04297* .

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, 3844–3852.

Di, X.; Yu, P.; Bu, R.; and Sun, M. 2019. Mutual Information Maximization in Graph Neural Networks. *arXiv preprint arXiv:1905.08509* .

Diehl, F. 2019. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990* .

Fang, H.; Wang, S.; Zhou, M.; Ding, J.; and Xie, P. 2020. CERT: Contrastive Self-supervised Learning for Language Understanding. *arXiv e-prints* arXiv–2005.

Gao, H.; and Ji, S. 2019. Graph u-nets. *arXiv preprint arXiv:1905.05178* .

Gidaris, S.; Singh, P.; and Komodakis, N. 2018. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728* .

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

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, 1024–1034.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. 2019. Momentum Contrast for Unsupervised Visual Representation Learning. *arXiv preprint arXiv:1911.05722* .

He, X.; Cai, Z.; Wei, W.; Zhang, Y.; Mou, L.; Xing, E.; and Xie, P. 2020a. Pathological Visual Question Answering. *arXiv preprint arXiv:2010.12435* .

He, X.; Yang, X.; Zhang, S.; Zhao, J.; Zhang, Y.; Xing, E.; and Xie, P. 2020b. Sample-efficient deep learning for covid-19 diagnosis based on ct scans. *MedRxiv* .

Hénaff, O. J.; Razavi, A.; Doersch, C.; Eslami, S.; and Oord, A. v. d. 2019. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272* .

Khosla, P.; Teterwak, P.; Wang, C.; Sarna, A.; Tian, Y.; Isola, P.; Maschinot, A.; Liu, C.; and Krishnan, D. 2020. Supervised Contrastive Learning. *arXiv preprint arXiv:2004.11362* .

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

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

Klein, T.; and Nabi, M. 2020. Contrastive Self-Supervised Learning for Commonsense Reasoning. *arXiv preprint arXiv:2005.00669* .

Lee, J.; Lee, I.; and Kang, J. 2019. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082* .

Loshchilov, I.; and Hutter, F. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* .

Ma, Y.; Wang, S.; Aggarwal, C. C.; and Tang, J. 2019. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 723–731.

Nathan Mundhenk, T.; Ho, D.; and Chen, B. Y. 2018. Improvements to context based self-supervised learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9339–9348.

Oord, A. v. d.; Li, Y.; and Vinyals, O. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* .

Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 1105–1114.

Pathak, D.; Krahenbuhl, P.; Donahue, J.; Darrell, T.; and Efros, A. A. 2016. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2536–2544.

Peng, Z.; Dong, Y.; Luo, M.; Wu, X.-M.; and Zheng, Q. 2020. Self-Supervised Graph Representation Learning via Global Context Prediction. *arXiv preprint arXiv:2003.01604* .

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710.

Qiu, J.; Chen, Q.; Dong, Y.; Zhang, J.; Yang, H.; Ding, M.; Wang, K.; and Tang, J. 2020. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. In *Proceed-*

*ings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1150–1160.

Qiu, J.; Dong, Y.; Ma, H.; Li, J.; Wang, K.; and Tang, J. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 459–467.

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

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

Srinivas, A.; Laskin, M.; and Abbeel, P. 2020. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136* .

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

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, 1067–1077.

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

Vinyals, O.; Bengio, S.; and Kudlur, M. 2015. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391* .

Yang, X.; He, X.; Liang, Y.; Yang, Y.; Zhang, S.; and Xie, P. 2020. Transfer Learning or Self-supervised Learning? A Tale of Two Pretraining Paradigms. *arXiv preprint arXiv:2007.04234* .

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, 4800–4810.

Zhang, M.; Cui, Z.; Neumann, M.; and Chen, Y. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Zhang, R.; Isola, P.; and Efros, A. A. 2016. Colorful image colorization. In *European conference on computer vision*, 649–666. Springer.

Zhang, Z.; Bu, J.; Ester, M.; Zhang, J.; Yao, C.; Yu, Z.; and Wang, C. 2019. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954* .

Zhou, M.; Li, Z.; and Xie, P. 2021. Self-supervised Regularization for Text Classification. *Transactions of the Association for Computational Linguistics* .