TRAINING GRAPH NEURAL NETWORKS VIA SELF-SUPERVISED LEARNING: EXPERIMENTS AND ANALYSIS

Ho Ching-Ru*

*M.S. in Data Science Degree Program,
National Taiwan University and Academia Sinica.
E-mail: r09946006@ntu.edu.tw
Advisor: Yen Tso-Jung (Institute of Statistical Science, Academia Sinica)
Oral Exam Date: July 4th, 2022

Abstract

Supervised learning is a popular model training method. However, its success relies on the use of huge amounts of labeled data. Recent advances in self-supervised learning have provided researchers with a means to train models on data in which only a few labeled observations are required. Self-supervised learning is efficient because it can perform model training without requiring a large amount of preprocessed data. State-of-the-art self-supervised models can achieve, even exceed, the performance of supervised models.

Most studies on self-supervised learning have been conducted in the fields of computer vision and natural language processing. Meanwhile, self-supervised learning on graph data is still nascent. In this thesis, we explored self-supervised learning for training graph neural networks (GNNs). We conducted experiments by training GNN models on four molecular and bioinformatics datasets in different experimental settings. Furthermore, we provided possible explanations for the experiment results.

We found that models with a deeper encoder structure can obtain superior results. However, increasing the hidden dimension size when a model is trained on small or medium-size datasets can only result in little improvement. By contrast, different data augmentation methods and different types of models can yield different results on molecular and bioinformatics datasets.

Keywords: self-supervised learning, graph neural network, encoder training

1 Introduction

Rapid advances in machine learning have enabled computer models to solve tasks that previously necessitated extensive human labor. Most of such successes rely on supervised learning, a learning paradigm in which computer models are trained on labeled data. These labeled data guide these models to learn efficiently. However, labeling data is challenging and laborious. For example, to train a model to predict the solubility of a chemical compound, experts need some domain knowledge about chemistry to label data on molecular attributes before training. The cost of labeling data makes supervised learning less attractive and robust.

Recently, learning from unlabeled data has become popular in machine learning. One of such learning concepts is called self-supervised learning, which trains models by leveraging information from data themselves without guidance from labels or other external information. Models trained by self-supervised learning are often used to produce a data representation, i.e., features, for various downstream tasks such as prediction and classification.

Generally, self-supervised learning methods can be grouped into four approaches: contrastive learning, clustering learning, distillation learning, and redundancy reduction. In the fields of computer vision (CV) [1] and natural language processing (NLP), researchers have developed many self-supervised methods. These methods have produced models that outperform other supervised learning methods in many benchmark tasks.

However, despite its widespread use in computer vision and NLP, self-supervised learning remains a challenge in graph representation learning. One early attempt is the GraphCL [2] introduced by You et al. The GraphCL method trains graph neural networks (GNNs) under a contrastive learning framework. It has achieved remarkable results in several graph classification benchmarks.

In this thesis, we use three self-supervised learning approaches to train models on graph data: contrastive learning, distillation learning, and redundancy reduction. We will conduct simulation experiments by training our models on four real-world datasets under various experimental settings: the use of different data augmentation methods, different numbers of layers, various batch sizes, and so on. Overall, there are 144 experimental settings. Figure 1 shows the flow diagram of each experiment.

This thesis comprises five themed chapters. In the next chapter, we will review self-supervised learning and benchmark GNN models, which serve as encoders for extracting features from graph data. Then, we will describe three self-supervised learning methods in Chapter 3 that will be further used to train GNNs on un-

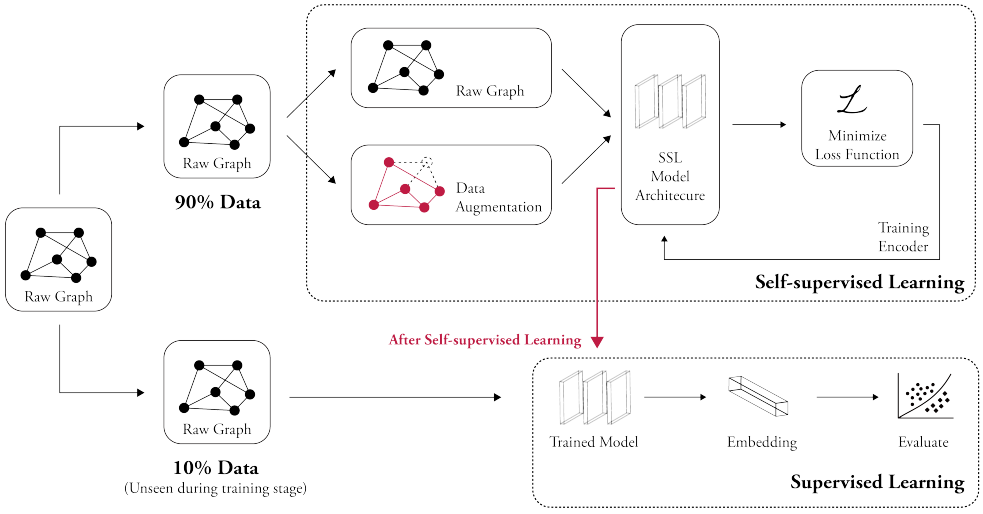

Figure 1: **Flow diagram of experiments.** The experiment can be roughly separated into two parts: 90% of the data are used to train an encoder via self-supervised learning, and the other 10% are used to evaluate the performance of the model via supervised learning.

labeled data. In Chapter 4, we will evaluate the performance of the three self-supervised learning methods by conducting simulation experiments, where several GNNs will be trained on real-world data under different experiment settings. Finally, we will briefly describe possible future research directions in the last chapter as a conclusion.

2 Literature Reviews

In this chapter, we review recent developments in self-supervised learning and graph neural networks.

2.1 Self-supervised Learning

Supervised learning is the most widely adopted ML approach for training deep neural networks (DNNs). However, it requires large amounts of labeled data for success. Due to the limited human resources and computing power, it is difficult to obtain large amounts of labeled data. Because of this difficulty, self-supervised learning has received significant attention in recent years.

Being a branch of unsupervised learning, self-supervised learning trains models by leveraging information from data themselves without guidance from labels or other external information. In image processing or NLP, there is a common situation in which there are large amounts of data but only a small fraction of them are labeled [3].

Several methods based on self-supervised learning have been developed in recent years. In 2018, a memory bank [4] structure was introduced into machine learning. Researchers use the convolutional neural network (CNN) backbone to generate high-dimension features of original images, store these features in a memory bank, and then use a non-parametric softmax classifier with NCE loss and proximal regularization to calculate the probability of prediction and train an encoder. This process is regarded as the basis of contrastive learning [5].

Based on the memory bank architecture, MoCo (*Momentum Contrast*) [6; 7] uses a specific momentum-updated encoder, a memory bank, and a dynamic queue used for generating negative samples. MoCo performs the learning procedure by comparing positive and negative sample pairs. MoCo can produce models that outperform supervised models in several ImageNet-related tasks.

SimCLR (*A Simple Framework for Contrastive Learning of Visual Representations*) [8; 9] is another self-supervised learning method based on the idea of contrastive learning. SimCLR first applies several data augmentation methods to an image and then inputs a data pair into an encoder to obtain the embedding. The embedding is then mapped to a latent space through a projector. In this latent space, positive and negative sample pairs are compared to train the encoder.

SimCLR can achieve better performance than supervised models in several ImageNet-related tasks. In addition, the authors have proposed several tips to improve the performance of self-supervised learning, including using a larger batch size, a

multilayer architecture as the projector, and different types of image data augmentation methods such as Gaussian deblur.

In MoCo and SimCLR, the distance between each pair of projections is calculated, and similar projections should have a closer distance than the other unrelated pairs. Another method for performing such comparison, known as clustering learning, is to let the encoder group closer projections as a cluster by itself.

DeepCluster [10] introduces a clustering module into the latent feature layer. Features generated from input images will be separated into various clusters. After the pseudo-labeling stage, each cluster will be regarded as a unique class. The model will train the encoder via backpropagation.

To prevent mapping all data points to the same cluster, SeLa (*Self Labelling*) [11] adds some constraints on a label by maximizing the information between the label and input data. Furthermore, SeLa uses the Sinkhorn–Knopp algorithm to speed up the self-labeling process and reduce the training time.

SwAV (*Swapping Assignments Between Views*) [12] computes encoded probability by matching projections to prototype clusters. SwAV adopts one's assignment to predict the projection of another by swapping cluster assignments between different images.

Contrastive and clustering learning are powerful self-supervised learning approaches. However, they require a large number of negative samples to train an encoder. Without negative samples, an encoder can only be trained with positive representation information, causing a trained model to easily converge to a trivial solution or suffer from gradient collapse.

To tackle this problem, BYOL (*Bootstrap Your Own Latent*) [13] connects a predictor to the projector in the MoCo architecture. The predictor learns by mapping projection from an online network (student encoder) to a target network (teacher encoder, similar to the momentum encoder in MoCo). In addition, only the online network is updated via backpropagation during training. The target network is updated using a stop-gradient mechanism and an exponential average of the online network. In this sense, BYOL can also be regarded as a special type of distillation learning approach.

Another distillation learning method is called Simsiam (*Simple Siamese Representation Learning*) [14]. Simsiam is inspired by BYOL, but unlike BYOL, the former simplifies the prototype of the online and target networks by removing the momentum-updated target encoder and connecting the target network directly to the online network.

Negative samples are not required in distillation learning methods such as BYOL and Simsiam. Instead, they used the predictor and stop-gradient mechanism to

train a powerful encoder.

As discussed above, the contrastive, clustering, and distillation learning approaches train an encoder from the sample projection. Barlow Twins [15], another self-supervised learning method, starts from another perspective. Inspired by neuroscientist H. Barlow's redundancy reduction principle, Barlow Twins uses the embedding without considering negative samples or momentum average. It focuses on training an encoder that can yield data representation without redundant components. Barlow Twins can also achieve good performance on ImageNet-related tasks.

Self-supervised learning has made remarkable progress in recent years. Table 1 shows the learning approaches and methods to which they belong.

Contrastive Learning	Memory Bank (Wu et al., 2018 [4]) Moco (He et al., v1:2019 [6]; v2:2020 [7]) SimCLR (Chen et al., 2020a & 2020b [8; 9])
Clustering Learning	DeepCluster (Caron et al., 2019 [10]) SeLa (Asano et al., 2020 [11]) SwAV (Caron et al., 2020 [12])
Distillation Learning	BYOL (Grill et al.,2020 [13]) Simsiam (Chen & He, 2020 [14])
Redundancy Reduction	Barlow Twins (Zbontar et al., 2021 [15])

Table 1: **Approaches of self-supervised learning.**

2.2 Graph Neural Networks (GNN)

A graph is a type of data structure for non-tablize information. It is commonly seen in knowledge graph [16], social network [17], recommendation system [18], combinatorial optimization [19], particle simulation [20], molecule discovery [21], and many other machine learning related tasks. Owing to its specific structure, we can use a neighborhood aggregation and message passing scheme to capture information within nodes' neighborhoods, which can preserve the relationship between each node and edge rather than use the table format [22]..

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes a set of nodes and $\mathcal{E}$ denotes a set of edges. Let $\mathcal{N}_u$ denote a node set of nodes adjacent to node $u \in \mathcal{V}$. We can obtain the $\mathbf{h}_u^{(k)}$, the feature vector of node u at the kth layer, via the following

operators:

$$\mathbf{h}_u^{(k)} = \text{COMBINE}^{(k)}\Big(\mathbf{h}_u^{(k-1)}, \mathbf{a}_u^{(k)}\Big), \tag{1}$$

where

$$\mathbf{a}_u^{(k)} = \text{AGGREGATE}^{(k)}\Big(\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}_u\}\Big). \tag{2}$$

During the iteration of neural networks, the AGGREGATE$(\cdot)$ operator generates aggregation message $\mathbf{a}_u^{(k)}$ based on the aggregated information of adjacent nodes set $\mathcal{N}_u$. Subsequently, the aggregation message of node u and the feature of previous $k-1$th layer $\mathbf{h}_u^{(k-1)}$ will be combined to generate the updated feature in the latest kth layer, $\mathbf{h}_u^{(k)}$, through the COMBINE$(\cdot)$ operator.

After generating each feature of nodes using a K-layer GNN, we can use a READOUT$(\cdot)$ operator to obtain the entire graph embedding $\mathbf{h}$ of $\mathcal{G}$:

$$\mathbf{h} = \text{READOUT}\Big(\{\mathbf{h}_u^{(K)}, \forall u \in \mathcal{V}\}\Big). \tag{3}$$

Based on the use of different COMBINE$(\cdot)$ and AGGREGATE$(\cdot)$ operators, several architectures for encoding graph data have been proposed. For example, GraphSAGE [23] uses element-wise max-pooling MAX$(\cdot)$ with a non-linear function $\sigma(\cdot)$, such as ReLu or sigmoid function, as the AGGREGATE$(\cdot)$ operator and concatenates the aggregation vector and $k-1$th feature vector to obtain the updating feature:

$$\mathbf{a}_u^{(k)} = \text{MAX}\Big(\Big\{\sigma(\Theta^{(k)} \cdot \mathbf{h}_v^{(k-1)}), \forall v \in \mathcal{N}_u\Big\}\Big), \tag{4}$$

$$\mathbf{h}_u^{(k)} = \text{CONCAT}\Big(\mathbf{h}_u^{(k-1)}, \mathbf{a}_u^{(k)}\Big). \tag{5}$$

where $\Theta^{(k)}$ is a learnable weight matrix.

By contrast, Graph Convolutional Networks (GCN) [24] use element-wise mean pooling MEAN$(\cdot)$ for information propagation, and the AGGREGATE$(\cdot)$ and COMBINE$(\cdot)$ functions are integrated as follows:

$$\mathbf{h}_u^{(k)} = \text{ReLu}\Big(\Theta^{(k)} \cdot \text{MEAN}\Big\{\mathbf{h}_v^{(k-1}, \forall v \in \mathcal{N}_u \cup \{u\}\Big\}\Big). \tag{6}$$

Furthermore, Graph Isomorphism Network (GIN) [25] use multilayer perceptrons (MLPs) for information propagation because they can represent a good function composition. The iteration function can be represented as

$$\mathbf{a}_u^{(k)} = \Big(\sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(k-1)} \Big), \tag{7}$$

$$\mathbf{h}_u^{(k)} = \mathrm{MLP}\Big((1 + \epsilon^{(k)}) \cdot \mathbf{h}_u^{(k-1)} + \mathbf{a}_u^{(k)} \Big), \tag{8}$$

$$\mathbf{h} = \mathrm{CONCAT}\Big(\mathbf{h}_u^{(K)}, \forall u \in \mathcal{V} \Big). \tag{9}$$

where ϵ can be a learnable parameter or a fixed scalar.

In this thesis, we will choose Graph Isomorphism Network as the main encoder for encoding graph data. The details are presented in Table 3.

2.3 Discussion

In this chapter, we have described the basic idea of self-supervised learning. Furthermore, we have reviewed several benchmark GNN models, including GraphSAGE, Graph Convolutional Networks, and Graph Isomorphism Networks.

In the next chapter, we will present more details on model training, including the datasets, data augmentation techniques, and the three self-supervised learning methods that we will use to train those GNNs on unlabeled data.

3 Methodology

Many factors can influence model performance, including encoder structure, depth and width of layers, and batch size. This thesis investigates the effects of these factors on model performance, particularly the performance of models trained on graph data via self-supervised learning.

Next, we introduce our experiment settings, including the datasets, data augmentation methods, hyperparameters in the training procedure, and self-supervised learning methods we use for model training.

3.1 Dataset

We mainly use the following datasets for model training: MUTAG, NCI1, PROTEINS, and DD. These datasets are collected by TUDataset [26] and are widely used in graph representation learning for evaluating model performance [27].

These datasets contain graph data falling into two categories: chemical molecules and bioinformatics. In MUTAG [28] and NCI1 [29], each graph is used to represent a chemical compound or molecule. Each node represents an atom, and each edge represents a chemical bond connecting two nodes. Both datasets are collected for binary graph classification tasks encoded by a one-hot encoding. The prediction task of MUTAG is to predict the mutagenicity of *Salmonella typhimurium*, and the task of NCI1 is to determine whether a chemical compound is positive or negative for cell lung cancer.

On the other hand, in PROTEINS [30] and DD [30], each graph is used to represent a protein structure. Each node represents an amino acid, and two nodes are connected by an edge if their distance is less than 6 angstroms apart. The prediction tasks of both datasets are to classify protein structures as enzymes or non-enzymes.

Table 2 provides the statistical details of these datasets.

3.2 Data Augmentation

Self-supervised learning heavily relies on data augmentation for model training [31; 32]. Designing a robust and useful augmentation method is an imperative task in the preprocessing stage. Unlike image data, which can be augmented using methods such as random cropping, blurring, or rotation, graph data are subject to different mechanisms for augmentation. Based on GraphCL, we adopt the following four methods for augmenting our graph data:

Dataset	Category	# Graphs	# Classes	Avg. Nodes	Avg. Edges
MUTAG [28]	Molecules	188	2	17.93	19.79
NCI1 [29]	Molecules	4110	2	29.87	32.30
PROTEINS [30]	Bioinformatics	1113	2	39.06	72.82
DD [30]	Bioinformatics	1178	2	284.32	715.66

Table 2: **Statistics of datasets.** Incidentally, these datasets are collected for graph classification tasks.

ATTRIBUTE MASKING. This operator replaces an original attribute with a random value generated by a normal distribution. Generally, some nodes have unique attributes. For example, an atom has its chemical property. We expect that a small change in a few nodes' attributes should not affect the information of the graph.

NODE DROPPING. This operator randomly drops some nodes and their linked edges that connect to other nodes. For example, if we set the augmentation ratio to 0.3, a total of 30% of the nodes and their connected edges will be dropped. Despite a few nodes being ignored, we expect the hidden information and features of the graph not to be significantly affected.

EDGE PERTURBATION. Similar to the NODE DROPPING operator, this operator randomly adds or deletes an edge based on a specific ratio. We expect that when a few edges between nodes in a graph are perturbed, the hidden property of the graph will be unaffected.

SUBGRAPH. This operator randomly samples a subgraph from the local part of the raw graph. In general, we expect the information in the graph to be preserved in its partial structure.

The illustrations for our data augmentation procedures are shown in Figure 2, where the red parts drawn in graphs are the output of data augmentation.

3.3 Experiment Factor

To compare the performance of different models trained on graph data, we conducted a series of experiments with varying experiment factors.

We adopted the three self-supervised learning methods, SimCLR, Simsiam, and Barlow Twins, along with the four data augmentations described in Section 3.2, to train our encoders. We set the batch size equal to 64 and 256 (for models

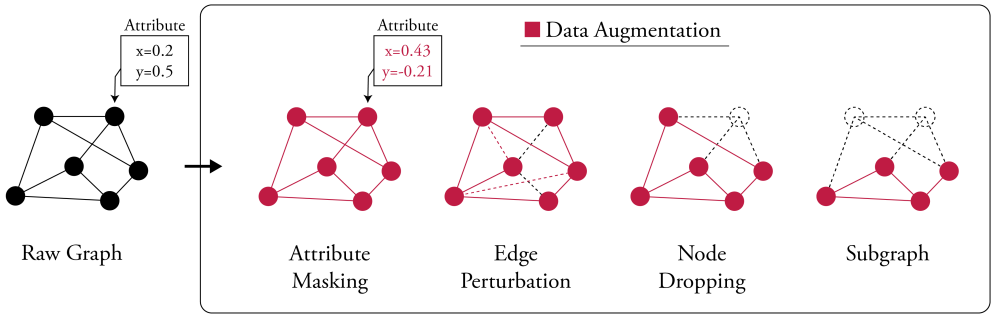

Figure 2: **Illustrations of augmentation operators.** Compared with the leftmost raw graph, the red parts drawn in the graphs are the output after data augmentation. The augmented ratio is set to 0.3.

trained on the DD dataset, we set the batch size equal to 64 and 128) and the dimension of the hidden space equal to 64 and 512.

In addition, because the depth of an encoder is a key factor in self-supervised learning, we adopted the monolayer, bilayer, and trilayer architectures for Graph Isomorphism Networks. Because of these experiment settings, there are 144 experiments for each of the four datasets.

These experiment factors are listed in Table 3. Each experiment was repeated five times to obtain an average result and standard deviation.

3.4 Contrastive Learning: SimCLR

Let $\mathbf{x}$ denote the raw data. In SimCLR, we can obtain $\mathbf{x}$'s embedding $\mathbf{h}$ using an encoder $\phi(\cdot)$ and obtain its projection $\mathbf{z}$ through a projector $\theta(\cdot)$. Similarly, we can obtain the augmented embedding $\mathbf{h}^*$ and augmented projection $\mathbf{z}^*$ using the augmented data $\mathbf{x}^*$, where $\mathbf{x}^* = \text{AugOperator}(\mathbf{x})$. Mathematically, we can express $\mathbf{h}$ and $\mathbf{z}$ by

$$\mathbf{h} = \phi(\mathbf{x}), \tag{10}$$

$$\mathbf{z} = \theta(\mathbf{h}) = \theta\Big(\phi(\mathbf{x})\Big), \tag{11}$$

respectively. According to empirical experiments and observation, a projector can

Self-Supervised Approach	• SimCLR (See Section 3.4 and Figure 3) • Simsiam (See Section 3.5 and Figure 4) • Barlow Twins (See Section 3.6 and Figure 5)
Data Augmentation	• ATTRIBUTE MASKING • EDGE PERTURBATION • NODE DROPPPING • SUBGRAPH • with ratio 0.3
Mini-batch Size	• for MUTAG, PROTEINS, NCCI1: 64, 256 • for DD: 64, 128
Hidden Dimension	64, 512
Encoder	• Encoder Type: Graph Isomorphism Network (GIN) • Number of Layer: 1 (monolayer), 2 (bilayer), 3 (trilayer)
Number of Projector Layer	3 (trilayer)
Learning Rate	• 0.01 • for Simsiam, with stop-gradient mechanism
Epoch	200
Data Proportion	• 90% used in self-supervised for training stage • 10% used in supervised for validation and testing stage

Table 3: **Detail of Experiment factors.** The use of different independent and control variables to observe the effect of each factor resulted in 144 experiments being conducted.

make representation more flexible and improve prediction effects.

The loss function used by SimCLR is the NT-Xent loss (normalized temperature-scaled cross entropy loss). For a positive pair of representations $(\mathbf{z}, \mathbf{z}^*)$, the loss function is defined as

$$Loss = -\log \frac{\exp\left(\mathbf{sim}(\mathbf{z}, \mathbf{z}^*)/\tau\right)}{\sum_{\mathbf{z}' \neq \mathbf{z}}^{N} \exp\left(\mathbf{sim}(\mathbf{z}, \mathbf{z}')/\tau\right)}, \tag{12}$$

where

$$\mathbf{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^\intercal \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2} \tag{13}$$

where τ denotes the trainable temperature parameter and $\mathbf{sim}(\mathbf{a}, \mathbf{b})$ denotes the l_2 normalized cosine similarity between two vectors $\mathbf{a}$ and $\mathbf{b}$. Throughout the experiments, we set $\tau = 0.2$. Notice that the negative representation $\mathbf{z}^{'}$ is generated from the other $N - 1$ samples, and the gradient is updated using both the raw and augmented data.

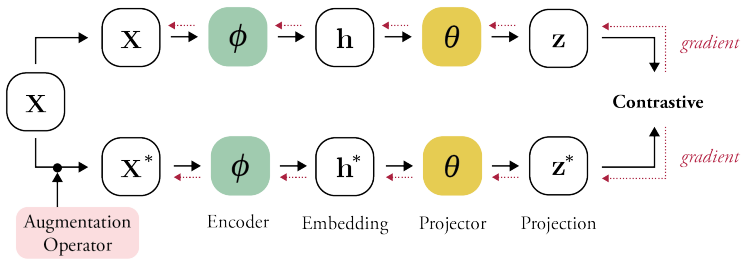

Figure 3: **Architecture of SimCLR models.** The encoder will learn from the projection of raw and augmented data via NT-Xent loss. According to empirical experiments and observation, a projector can make representation more flexible and improve prediction effects.

3.5 Distillation Learning: Simsiam

In Simsiam, we use an encoder $\phi(\cdot)$ and a projector $\theta(\cdot)$ to obtain the embedding and projection of the raw data $\mathbf{x}$ and the augmented data $\mathbf{x}^*$. Moreover, Simsiam introduces a predictor $\psi(\cdot)$ that generates the prediction $\mathbf{p}$ via

$$\mathbf{p} = \psi(\mathbf{z}) = \psi(\theta(\mathbf{h})) = \psi\Big(\theta(\phi(\mathbf{x}))\Big). \tag{14}$$

The loss function follows the cosine similarity between pairs $(\mathbf{z}^*, \mathbf{p})$ and $(\mathbf{z}, \mathbf{p}^*)$. It is defined by

$$\begin{aligned} \mathit{Loss} &= -\frac{1}{2}\sum_{\text{Dataset}}^{N}\Big(\mathbf{sim}(\mathbf{z}^*, \mathbf{p}) + \mathbf{sim}(\mathbf{z}, \mathbf{p}^*)\Big) \\ &= -\frac{1}{2}\sum_{\text{Dataset}}^{N}\Big(\frac{(\mathbf{z}^*)^\intercal \mathbf{p}}{\|\mathbf{z}^*\|_2\|\mathbf{p}\|_2} + \frac{\mathbf{z}^\intercal \mathbf{p}^*}{\|\mathbf{z}\|_2\|\mathbf{p}^*\|_2}\Big). \end{aligned} \tag{15}$$

To prevent gradient collapse, $\mathbf{z}$ and $\mathbf{z}^*$, the representations produced by the target network, are treated as constants in the training stage. They are subject to the stop-gradient operator when computing the gradient of the loss function. Only $\mathbf{p}$ and $\mathbf{p}^*$, the representations produced by the online network, are updated.

As shown in Figure 4, to calculate the similarity of $(\mathbf{z}^*, \mathbf{p})$, Simsiam feeds raw graphs to the online network and augmented graphs to the target network. Thus, only the gradient in the online network is updated via backpropagation. Similarly, when Simsiam calculates the similarity of $(\mathbf{z}, \mathbf{p}^*)$, it will send raw graphs to the target network and send augmented graphs to the online network.

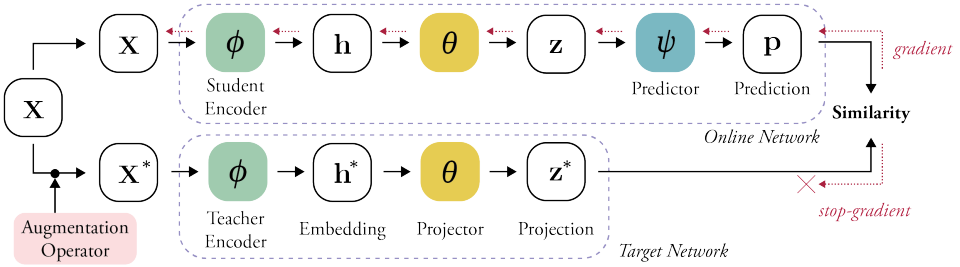

Figure 4: **Architecture of Simsiam models.** For example, to calculate the similarity of $(\mathbf{z}^*, \mathbf{p})$, Simsiam feeds raw graphs to the online network and augmented graphs to the target network. Only the gradient in the online network is updated via backpropagation.

3.6 Redundancy Reduction: Barlow Twins

Compared with Simsiam, which uses the projection–prediction pair $\mathbf{z}$ and $\mathbf{p}^*$, Barlow Twins directly uses the embedding pair for comparison. To calculate the loss function, Barlow Twins [33] computes the cross-correlation matrix $\mathcal{C}$ between the outputs of two identical networks along the batch dimension. The (i, j)th element of the correlation matrix $\mathcal{C}$ is defined by

$$\mathcal{C}_{ij} = \frac{\sum_b \left((\tilde{\mathbf{h}_{\mathbf{x}}})_{b,i}\right) \cdot \left((\tilde{\mathbf{h}^*_{\mathbf{x}}})_{b,j}\right)}{\sqrt{\sum_b \left((\tilde{\mathbf{h}_{\mathbf{x}}})_{b,i}\right)} \cdot \sqrt{\sum_b \left((\tilde{\mathbf{h}^*_{\mathbf{x}}})_{b,j}\right)}}. \tag{16}$$

where $\tilde{\mathbf{h}}$ denotes the normalized embedding of $\mathbf{h}$ and $\mathcal{C}_{ij}$ should be between 1 (perfect correlation) and -1 (perfect anticorrelation). If two identical networks are the same, known as perfect correlation, the correlation matrix $\mathcal{C}$ should be a diagonal matrix with 1, a.k.a. an identity matrix $\mathcal{I}$.

A good encoder should recognize augmentation data as having the same label as raw data. To train this encoder to distinguish a raw graph from an augmented graph and other irrelevant graphs, the loss function is divided into two parts, namely, the invariance term and redundancy reduction term, which are defined by

$$Loss = \underbrace{\sum_i (1 - \mathcal{C}_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\left(\sum_i \sum_{i \neq j} \mathcal{C}_{ij}^2\right)}_{\text{redundancy reduction term}}, \tag{17}$$

respectively. Here, λ is a positive constant used to trade off the importance of the invariance term and redundancy reduction term of the loss.

3.7 Discussion

In this chapter, we have described four benchmarks that will be used to train the GNNs. We also have described the data augmentation techniques used in model training. Furthermore, we have discussed the three self-supervised learning methods, SimCLR, Simsiam, and Barlow Twins, that will be used to train the GNNs on unlabeled data: contrastive learning, distillation learning, and dimension reduction approach, respectively.

In the next chapter, we will evaluate the performance of the three self-supervised learning methods by conducting simulation experiments on the four datasets.

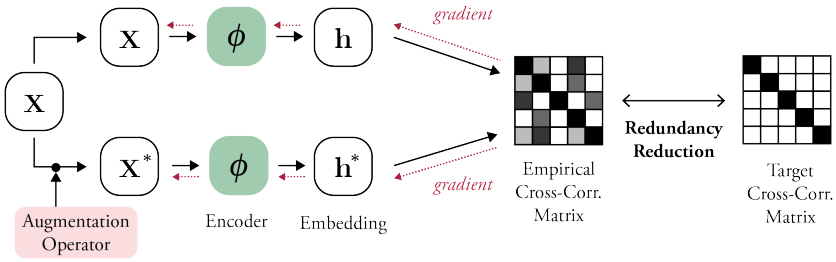

Figure 5: **Architecture of Barlow Twins models.** The target of an encoder is to classify the embeddings of raw and augmented data as the same label, where the correlation matrix $\mathcal{C}$ of embeddings should be an identity matrix $\mathcal{I}$.

4 Results and Discussion

In this chapter, we will discuss the results of the simulation experiments we described in Chapter 3. Table 4 shows the best results from the 144 experiments on the test dataset between these three learning approaches.

The complete experiment results and a link to the source code can be found in Appendix A.

Method	Molecular Dataset		Bioinformatics Dataset	
	MUTAG	NCI1	DD	PROTEINS
(1) 10% SimCLR	100.0±0.00	74.47± 1.45	84.36±4.02	76.88±2.45
(2) 10% Barlow Twins	94.00±3.54	72.12±0.82	79.94±2.99	83.20±1.31
(3) 10% Simsiam	95.00±0.00	73.70±0.38	78.21±2.76	75.29±2.10
(4) 10% baseline	-	73.72±0.24	73.56±0.41	70.40±1.54
(5) 10% Aug.	-	73.59±0.32	74.30±0.81	70.29±0.64
(6) 10% GAE	-	74.36±0.24	74.54±0.68	70.51±0.17
(7) 10% Infomax	-	74.86±0.26	75.78±0.34	72.27±0.40

Table 4: **A summary of experimental results for three learning approaches.** Each value in the table represents the best average of accuracy (unit: percentage) and the standard deviation in the testing stage. Results (1)–(3) are obtained from our experiments, and for Results (4)–(7), refer to You et al. [2]

4.1 Batch size's effects on SimCLR are not apparent

The authors of SimCLR [8] pointed out that contrastive learning benefits more from a larger batch size and longer training time. They used different batch sizes, 256, 512, 1024, 2048, 4096, and 8192, to train a ResNet-50 model. They found that these batch sizes are positively correlated with the model's performance, i.e., the larger the batch size, the better the performance of the model. This concept inspires subsequent self-supervised methods to attempt to use larger batch sizes for achieving better performance.

However, owing to computational power constraints, we only used 64 and 512 batch sizes (notice that DD uses 64 and 128). In addition, the sizes of the datasets we used are not large. The datasets contain less than 10,000 samples for model training. This makes the effect of the batch size less obvious in the SimCLR method.

In Figure 6, the results of the experiment are separated into two categories with two batch sizes. Each point represents the accuracy of supervised learning using different parameters. The performance of two batch sizes in SimCLR is compared, and the difference is not significant.

From the above results, we can infer that the advantage of large batch sizes in contrastive learning, especially in the SimCLR model, may only be appreciable when the dataset is large. If the dataset is of medium or small size, increasing the batch size may not improve model performance in self-supervised learning. Researchers could focus on other experiment factors.

4.2 Deeper encoders have better performance

Generally, models using a deep encoder are thought to have better performance than others using a shallow encoder. To judge this idea on graph datasets, we have tested three types of encoders, namely, monolayer, bilayer, and trilayer, during the training stage.

For an encoder with a monolayer, the MLP part consists of a linear layer, followed by a ReLU layer and another linear layer. We used a graph isomorphism network encoder to generate embeddings of samples. For bilayer and trilayer architectures, the MLP parts consist of several monolayers stacked and connected by batch normalization layers.

Figure 7 shows that the medians and quartiles of accuracy are higher for models with deeper encoders. Such superior performance is particularly obvious when models are trained on the MUTAG, NCI1, and PROTEINS datasets.

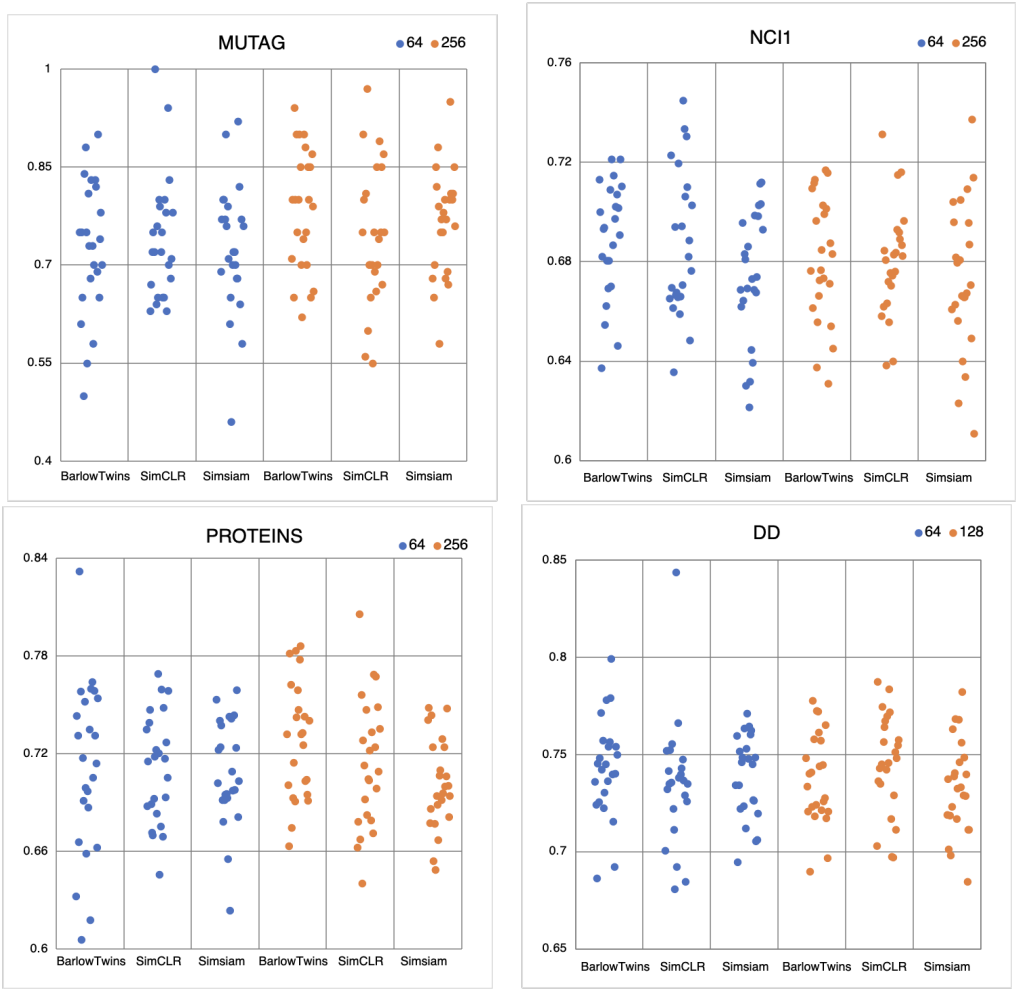

Figure 6: **Evaluation using different batch sizes.** In this figure, each colored dot represents the average accuracy of experiment after five times repetitions. By grouping experiments that use the same batch sizes, we use blue and orange colors to indicate the effect of different batch sizes in different types of models. For example, in the leftmost column of each figure, the blue dots represent the performance of a model that uses Barlow Twins and 64 mini-batches.

Following the experiment results, the hypothesis that the deeper encoder also benefits from the graph-structured model is verified through the analysis. Future self-supervised methods should concentrate on the deeper architecture.

4.3 Hidden dimension has little effect on model performance

In most state-of-the-art neural networks, especially in image processing, NLP, and classification tasks, the hidden dimension of each encoder layer influences model performance. The hidden dimension determines the size of the feature vector. As the hidden dimension increases, an encoder can capture more complex features as hidden states, resulting in a more comprehensive representation of data.

However, in our experiment, we tested 64 and 512 hidden dimensions for models trained on the MUTAG, NCI1, and PROTEINS and tested 64 and 128 for models trained on the DD dataset. Figure 8 shows that accuracy varies across different hidden dimensions. For models with 512 hidden dimensions, even if the vector size is eight times larger than those with 64 hidden dimensions, there is no significant improvement in model performance.

These surprising differences can be explained in part by the data structure of graph and image data. For image data, which consist of discrete pixels, with a larger hidden dimension, a neuron can capture more information via pixel clustering. However, graph data are more abstract. Nodes and edges cannot be separated or unified as pixel groups in image data. This difference makes adding hidden dimensions less effective in model training.

In summary, if we want to apply a self-supervised method to graph data, increasing the hidden dimension size may not be helpful. However, if the raw data can be discretization into small units, such as an image into pixels, or a sentence into n-gram, increasing the hidden dimension size is a reliable way to improve performance.

4.4 SimCLR performs better in molecular datasets

In our experiments, we used four datasets, of which two, MUTAG and NCI1, are molecular datasets. The other two datasets, PROTEINS and DD, are bioinformatics datasets. Figure 9 shows that models trained via SimCLR performed better than models trained via the other approaches on both molecular datasets. Conversely, models trained via SimCLR did not outperform models trained via Barlow Twins and Simsiam on bioinformatics datasets.

Reviewing these datasets will help us understand why performance varies. Al-

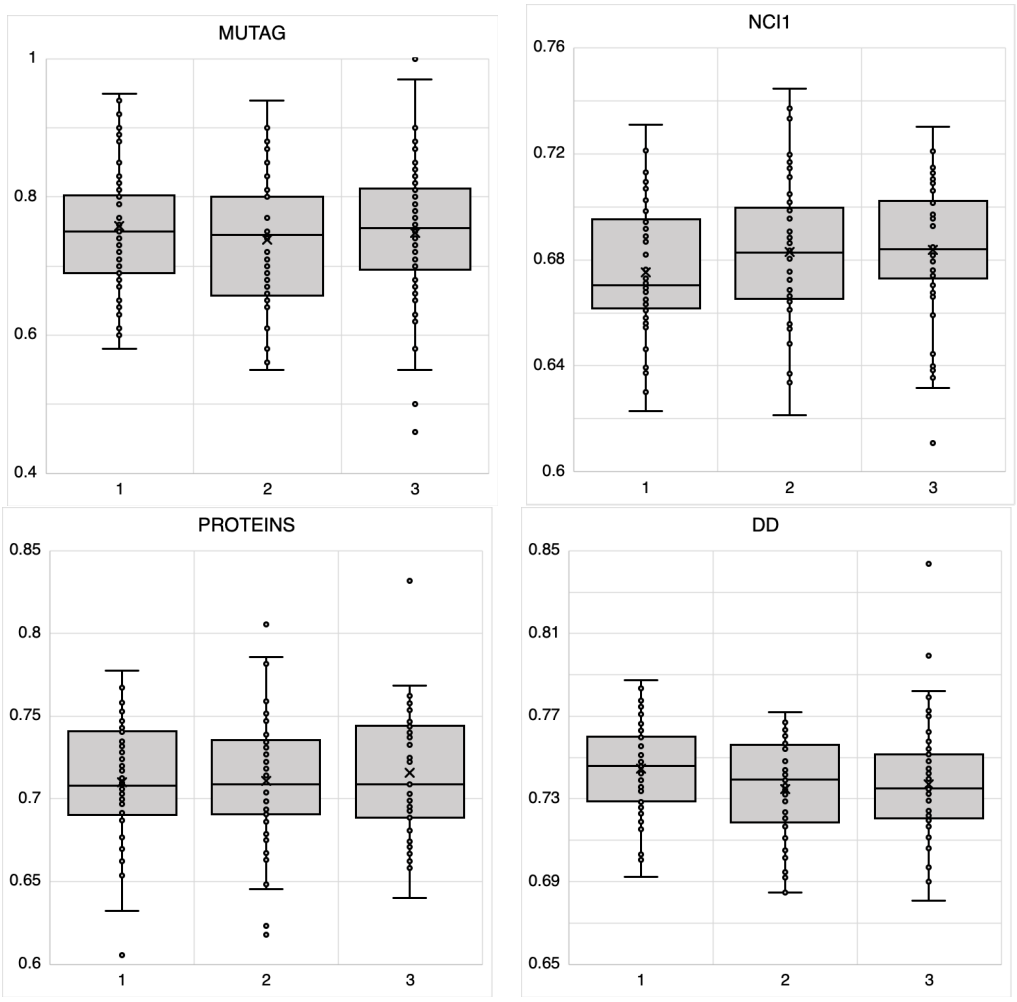

Figure 7: **Evaluation using different depths of an encoder layer.** In this figure, each gray dot represents the average accuracy of experiment after five times repetitions. By grouping experiments that use the same depths of the encoder layer, we can observe the minimum, median, maximum, and quartile performance under different encoder depths. For example, in the leftmost column of each figure, the gray nodes represent the performance of a model that uses used a monolayer encoder.

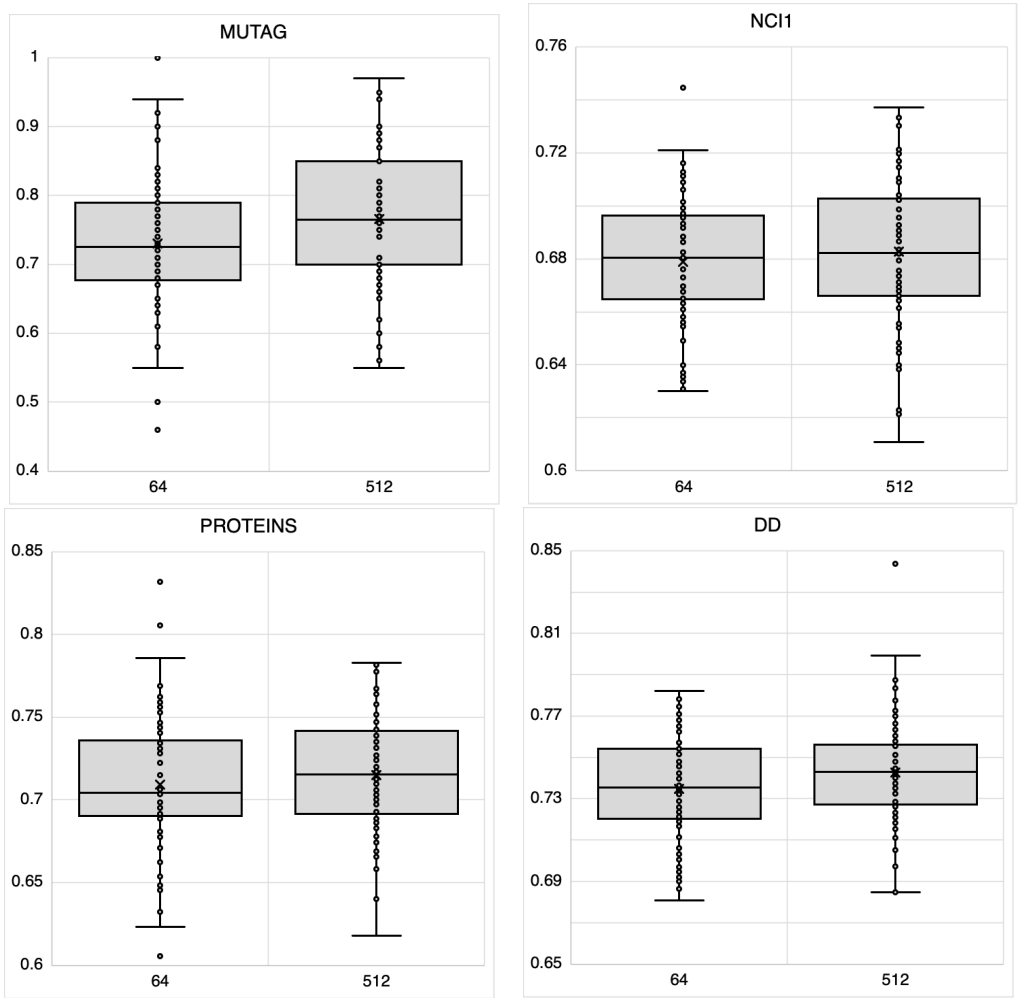

Figure 8: **Evaluation using different hidden dimensions.** In this figure, each gray dot represents the average accuracy of experiment after five times repetitions. By grouping experiments that use the same hidden dimensions, we can observe the minimum, median, maximum, and quartile performance under different hidden dimensions in the hidden layer. For example, in the leftmost column of each figure, the gray dots represent the performance of a model that uses 64 hidden dimensions.

though these data are graph-structured, there are fundamental differences between molecular and bioinformatics graph data. In molecular datasets, e.g., MUTAG and NCI1, each node represents an atom, and each edge represents a chemical bond connecting two nodes. In bioinformatics datasets, such as PROTEINS and DD, each node represents an amino acid, and two nodes are connected by an edge if they are less than 6 angstroms apart.

Both types of datasets have different types of properties, which can have complex effects on self-supervised models. Unlike Simsiam that only uses one projection, or Barlow Twins that uses embeddings directly, in the SimCLR architecture, raw data and augmented data are encoded into projections, which may affect performance.

The experimental results indicate that SimCLR is a suitable application model for molecular datasets; however, more empirical experiments are required to verify its robustness. Unfortunately, at present, owing to computational power constraints, this study lacks experimental logs of larger molecular datasets. A further study focusing on SimCLR or molecular graph-structured datasets should be conducted.

4.5 SUBGRAPH augmentation performs closer in bioinformatics dataset

Furthermore, there is an interesting discovery in our experiment results. The idea behind data augmentation, especially in the SUBGRAPH operator, assumes that the partial graph should retain the properties of the entire graph, even when some nodes or edges are added or removed.

From Figure 10 shows that models trained using the SUBGRAPH operator achieve similar performance on the DD and PROTEINS datasets. The variance in performance was lower than those on the molecular datasets.

As previously mentioned, molecular and bioinformatics datasets have different properties and targets. First, the goal of MUTAG is to train models to predict the mutagenicity of Salmonella typhimurium. Second, the goal of NCI1 is to train models to classify whether a chemical compound is positive or negative for cell lung cancer. Third, the goal of the PROTEINS and DD datasets is to train models to predict which proteins are enzymes or non-enzymes.

In a chemical molecule, each node and edge play a unique role. The functional group may be lost if we only capture a part of the graph. For example, the chemical formula of ethanol (a.k.a. alcohol) is C_2H_5OH, containing a hydroxyl group (-OH). If the augmented operator samples C_2H_4, it will be recognized as ethene, which has different chemical properties from ethanol. Because the

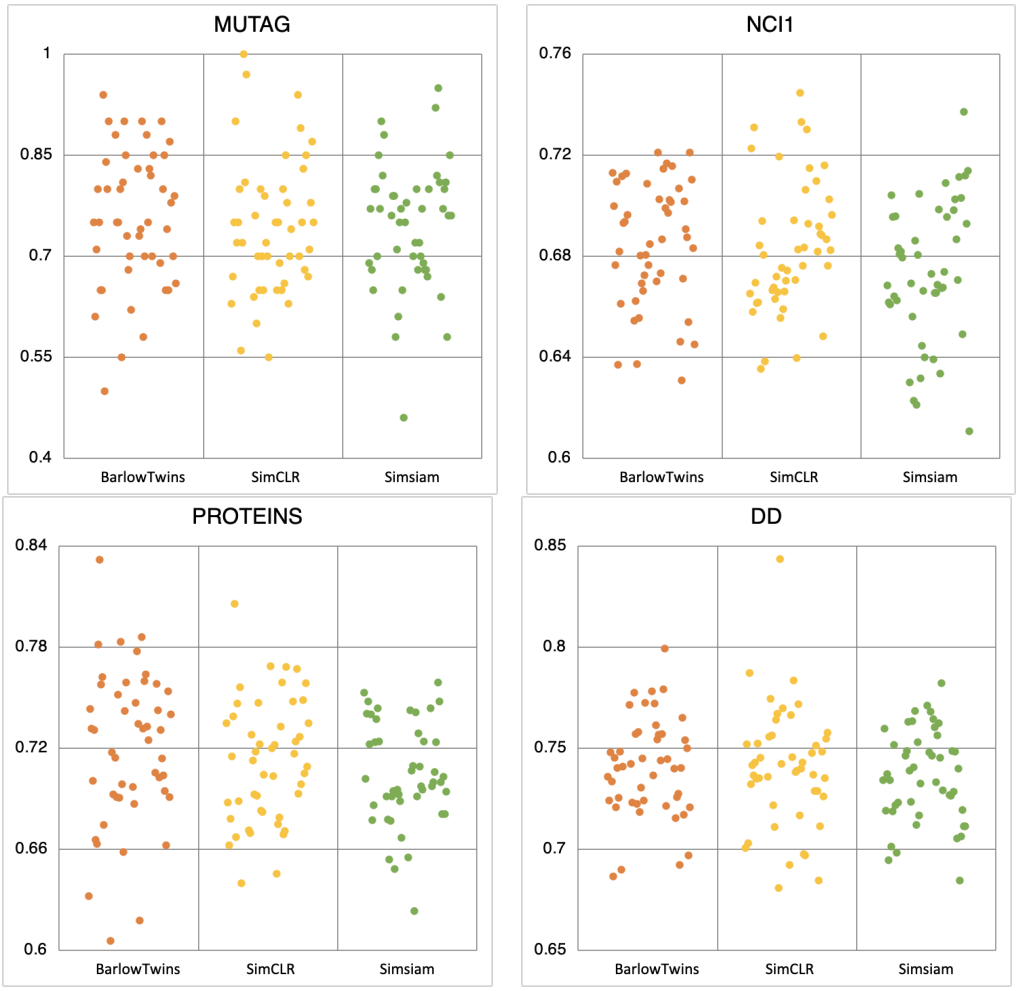

Figure 9: **Evaluation using different self-supervised methods.** In this figure, each colored dot represents the average accuracy of experiment. By grouping experiments that use the same approaches, we use orange, yellow, and green colors to indicate the evaluation via Barlow Twins, SimCLR, and Simsiam, respectively. For example, in the leftmost column of each figure, the orange dots represent the performance of a model that uses Barlow Twins.

augmented data have more diversity, the performance of the model will have a higher variance.

In contrast, a protein contains at least one long polypeptide, which is chained by amino acid. Proteins can be constructed in flexible ways [34]. Some proteins chained by different polypeptides can have similar chemical properties and functions. Therefore, even when we use the SUBGRAPH operator, the augmented data are similar to the raw data, reducing the variance between experiment results.

This finding, while preliminary, suggests that augmented data generated using the SUBGRAPH operator should be carefully used. To avoid a good experiment result producing trivial analysis, we should first observe the representation of each node and edge and consider their chemical and physic attributes before attempting to sample raw data as augmentation data.

4.6 Discussion

Most studies on self-supervised learning have only been conducted in image processing and NLP. In this chapter, we have revised the properties of graph data, analyzed the experimental results, and attempted to provide some possible explanations.

According to our findings, not all hyperparameters influence model performance significantly. Some of them only have a small effect when the dataset size is not sufficiently large. On the other hand, the fundamental differences between graphs and other data structures, along with the categories of graph datasets, prevent models from applying the same training strategies.

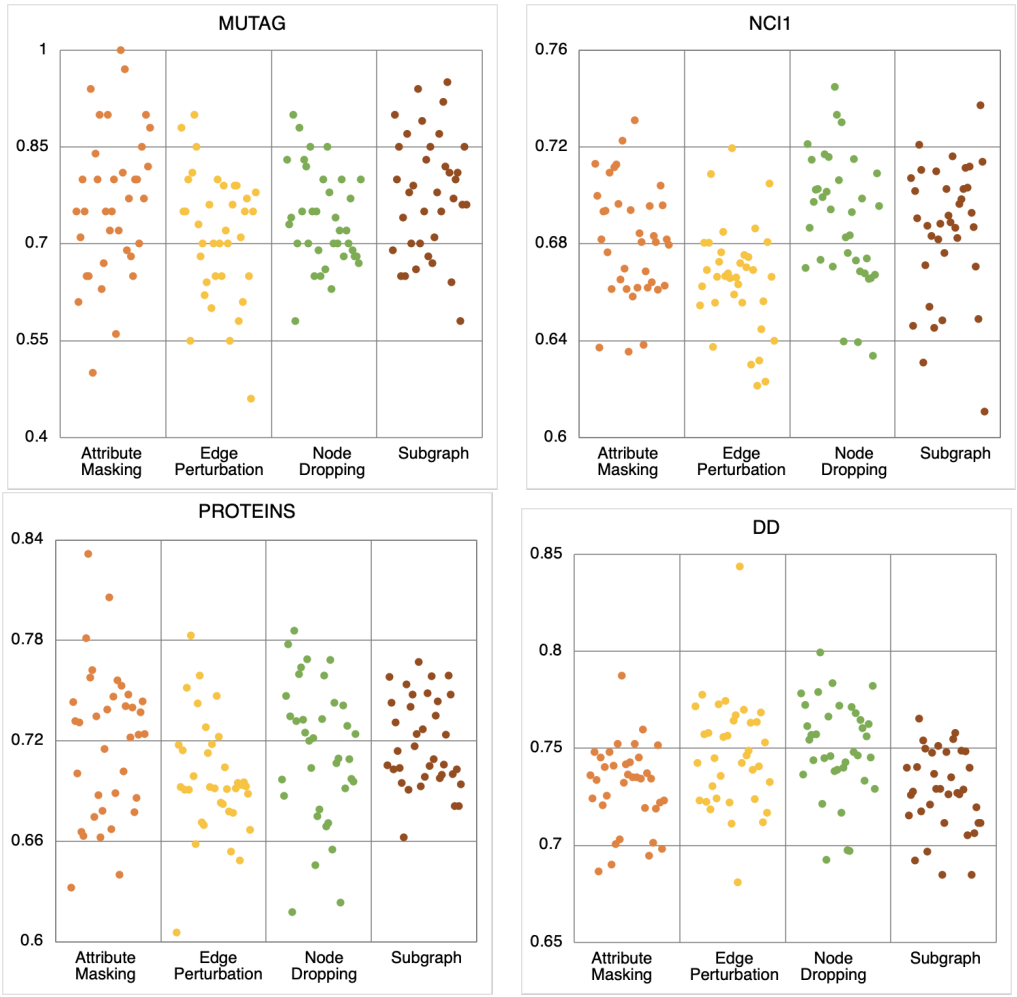

Figure 10: **SUBGRAPH augmentation performs better on bioinformatics datasets.** In this figure, each colored dot represents the average accuracy of experiment. By grouping experiments that use the same data augmentation method, we use orange, yellow, green, and brown colors to indicate the evaluation by ATTRIBUTE MASKING, EDGE PERTUBATION, NODE DROPPING, and SUBGRAPH, respectively. For example, in the leftmost column of each figure, the orange dots represent the performance of a model that uses ATTRIBUTE MASKING to generate augmented data.

5 Conclusion

We have systematically investigated three self-supervised learning methods by employing them to train DNNs on graph data. We have also discussed the simulation experiments in detail, providing explanations for the experiment results so that subsequent researchers may use them to propose better methods and ideas.

First, increasing the batch size in the training stage when the dataset is medium or small size is not helpful, but deepening the layer of the encoder seems to be a favorable means to improve model performance. Second, graphs have their special structure. Applying previous procedures used to improve the performance of models trained on image datasets, e.g., increasing the hidden dimension size, to every similar scenario may not be inappropriate. Finally, because of the characteristic differences between molecular and bioinformatics data, different optimal hyperparameters should be used when training models on these two data types.

At present, we only focus on small-scale datasets containing less than ten thousand samples. However, real-world data generated by commercial systems may have millions of nodes or 10 billion edges, which are orders of magnitude larger than the datasets used in this thesis. Owing to resource constraints and computational limitations, we cannot provide a comprehensive review of larger graph datasets.

There is abundant space for advancement in large-scale datasets [35] and other categories, such as social networks or recommendation systems [36]. Furthermore, future research should explore new data augmentation methods [37; 38], as well as other key factors in or approaches [39] to self-supervised learning, to find a robust and efficient method for improving model performance.

6 Appendix: Experiment Result

In the Table mentioned below, red colored values represent the best models in the training and test stages.

The source code and full experiment logs aref available on Github:

https://github.com/rutopio/Training-Graph-Neural-Networks-via-Self-Supervised-Learning

6.1 Performance on MUTAG dataset

Table 5 shows the experiment results of models trained on MUTAG dataset. Noticed that MUTAG is a relatively small dataset, which only has 188 graphs, so 100% accuracy is possible.

The best setting in the training stage is the one that uses SimCLR with Attribute Masking, batch size set to 64 hidden dimension set to 512, and using a trilayer MLP as the encoder.

The best setting in the test stage is the one that uses SimCLR with Attribute Masking, batch size set to 64, hidden dimension set to 512, and using a trilayer MLP as the encoder.

Model	Batch Size	Hidden Dimension	# Layer of Encoder	Node Droppping		Edge Pertubation		Attribute Masking		Subgraph	
				Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.
SimCLR	64	64	1	72.000±2.739	75.000±6.124	76.000±2.236	64.000±5.477	75.000±0.000	63.000±4.472	79.000±2.236	94.000±2.236
			2	75.000±0.000	65.000±0.000	65.000±0.000	65.000±0.000	78.000±4.472	72.000±4.472	84.000±2.236	83.000±4.472
			3	74.000±7.416	78.000±7.583	84.000±5.477	79.000±6.519	87.000±2.739	72.000±2.739	77.000±2.739	71.000±2.236
		512	1	80.000±0.000	65.000±0.000	78.000±6.708	76.000±2.236	75.000±0.000	67.000±2.739	85.000±0.000	70.000±0.000
			2	81.000±4.183	80.000±0.000	78.000±2.739	80.000±0.000	85.000±0.000	75.000±0.000	88.000±2.739	68.000±5.701
			3	77.000±2.739	63.000±4.472	80.000±0.000	72.000±2.739	100.000±0.000	100.000±0.000	85.000±0.000	78.000±5.701
	256	64	1	75.000±0.000	75.000±0.000	75.000±0.000	60.000±0.000	69.000±5.477	75.000±0.000	78.000±4.472	89.000±2.236
			2	76.000±2.236	66.000±2.236	71.000±2.236	70.000±0.000	81.000±4.183	80.000±0.000	76.000±2.236	85.000±0.000
			3	73.000±2.739	70.000±0.000	75.000±0.000	70.000±0.000	89.000±4.183	81.000±8.216	78.000±6.708	87.000±2.739
		512	1	80.000±0.000	69.000±2.236	65.000±0.000	70.000±0.000	77.000±2.739	90.000±0.000	85.000±0.000	75.000±0.000
			2	71.000±2.236	85.000±0.000	72.000±2.739	65.000±0.000	82.000±2.739	56.000±2.236	85.000±0.000	67.000±5.701
			3	76.000±2.236	74.000±2.236	75.000±0.000	55.000±0.000	99.000±2.236	97.000±6.708	85.000±0.000	75.000±0.000
Simsiam	64	64	1	80.000±0.000	70.000±0.000	76.000±2.236	76.000±2.236	74.000±5.477	69.000±4.183	83.000±2.739	92.000±2.739
			2	75.000±0.000	72.000±2.739	73.000±2.739	71.000±4.183	81.000±4.183	80.000±5.000	78.000±4.472	64.000±6.519
			3	80.000±3.536	68.000±2.739	79.000±5.477	65.000±0.000	84.000±4.183	77.000±2.739	72.000±6.708	58.000±4.472
		512	1	82.000±2.739	72.000±2.739	76.000±2.236	79.000±2.236	80.000±5.000	77.000±4.472	87.000±4.472	82.000±4.472
			2	75.000±0.000	70.000±0.000	76.000±2.236	61.000±2.236	83.000±2.739	80.000±3.536	78.000±4.472	77.000±7.583
			3	73.000±2.739	68.000±2.739	80.000±0.000	46.000±5.477	90.000±3.536	90.000±5.000	80.000±11.726	76.000±6.519
	256	64	1	74.000±4.183	80.000±0.000	72.000±4.472	79.000±2.236	75.000±0.000	68.000±4.472	79.000±5.477	95.000±0.000
			2	76.000±2.236	77.000±2.739	77.000±6.708	75.000±0.000	72.000±4.472	70.000±0.000	80.000±3.536	80.000±0.000
			3	74.000±2.236	67.000±2.739	75.000±0.000	75.000±0.000	82.000±2.739	82.000±2.739	81.000±2.236	85.000±0.000
		512	1	83.000±2.739	68.000±2.739	77.000±2.739	58.000±5.701	79.000±2.236	65.000±0.000	84.000±2.236	81.000±5.477
			2	79.000±6.519	69.000±9.618	77.000±4.472	77.000±4.472	78.000±5.701	85.000±0.000	81.000±6.519	81.000±2.236
			3	72.000±10.368	80.000±0.000	76.000±4.183	78.000±2.739	82.000±4.472	88.000±10.954	77.000±8.367	76.000±5.477
Barlow Twins	64	64	1	75.000±0.000	83.000±4.472	75.000±0.000	88.000±2.739	75.000±0.000	75.000±0.000	83.000±2.739	69.000±2.236
			2	80.000±0.000	58.000±2.739	73.000±4.472	55.000±0.000	80.000±3.536	75.000±0.000	80.000±5.000	65.000±0.000
			3	76.000±2.236	83.000±4.472	75.000±0.000	73.000±2.739	74.000±4.183	50.000±6.124	82.000±4.472	78.000±5.701
		512	1	80.000±0.000	73.000±2.739	85.000±0.000	75.000±0.000	71.000±5.477	61.000±2.236	90.000±0.000	90.000±0.000
			2	80.000±0.000	70.000±0.000	71.000±8.944	81.000±2.236	80.000±0.000	65.000±0.000	82.000±4.472	74.000±4.183
			3	86.000±2.236	82.000±4.472	84.000±2.236	68.000±6.708	82.000±5.701	84.000±2.236	86.000±4.183	70.000±7.071
	512	64	1	80.000±0.000	74.000±2.236	80.000±0.000	75.000±0.000	75.000±0.000	71.000±5.477	81.000±4.183	80.000±0.000
			2	76.000±2.236	88.000±2.739	75.000±0.000	90.000±0.000	75.000±0.000	65.000±0.000	83.000±4.472	65.000±3.536
			3	75.000±0.000	70.000±0.000	75.000±0.000	70.000±0.000	80.000±0.000	80.000±0.000	83.000±9.747	79.000±11.937
		512	1	81.000±4.183	90.000±0.000	77.000±2.739	80.000±0.000	75.000±0.000	80.000±0.000	85.000±0.000	85.000±0.000
			2	82.000±2.739	75.000±7.071	80.000±0.000	85.000±0.000	80.000±3.536	94.000±2.236	82.000±4.472	87.000±5.701
			3	83.000±2.739	85.000±0.000	80.000±3.536	62.000±4.472	91.000±4.183	90.000±3.536	85.000±0.000	66.000±4.183

Table 5: **Performance on MUTAG dataset.** The red colored values represent the best models in the training and test stages.

6.2 Performance on NCI1 dataset

Table 6 shows the experiment results of the models trained on NCI1 dataset.

The best setting in training stage is the one that uses SimCLR with Node Dropping, batch size set to 256, hidden dimension set to 512, and using a bilayer MLP as the encoder.

The best setting in test stage is the one that uses Barlow Twins with Attribute Masking, batch size set to 64, hidden dimension set to 64, and using a bilayer MLP as the encoder.

Model	Batch Size	Hidden Dimension	# Layer of Encoder	Node Droppping		Edge Pertubation		Attribute Masking		Subgraph	
				Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.
SimCLR	64	64	1	70.029±0.645	69.419±0.642	66.303±1.191	66.661±1.272	68.585±1.024	66.511±1.979	69.195±1.690	70.991±2.569
			2	71.452±0.673	74.470±1.453	67.541±0.714	66.574±0.505	68.463±0.657	66.961±0.884	70.756±1.055	68.839±1.382
			3	69.926±2.605	70.619±0.464	69.081±1.132	65.900±0.555	67.925±2.632	63.544±1.949	70.315±1.988	67.614±1.160
		512	1	66.995±1.143	67.052±2.307	65.343±0.692	66.771±1.121	68.959±1.484	72.261±0.572	69.923±0.782	68.190±1.158
			2	72.490±1.781	73.324±1.363	67.355±0.858	71.959±0.582	68.638±1.971	66.139±1.196	69.459±1.499	64.835±1.182
			3	71.783±1.552	73.026±1.266	70.509±0.578	66.606±0.915	70.317±0.712	69.395±2.789	68.876±1.016	70.261±0.717
	256	64	1	67.784±0.588	63.973±0.895	64.143±0.562	66.319±1.431	68.652±1.258	65.804±0.942	68.671±1.453	69.168±0.529
			2	68.554±1.137	67.611±0.262	67.415±1.054	65.566±0.499	66.164±0.989	66.179±0.733	70.369±1.032	71.603±1.485
			3	68.970±1.175	69.296±0.871	69.547±1.105	67.038±1.340	66.199±2.323	68.062±2.264	68.520±1.648	68.235±1.778
		512	1	66.913±0.621	68.268±1.073	66.462±1.127	67.194±0.314	68.351±0.601	73.107±0.886	70.062±1.458	68.893±0.880
			2	73.481±1.255	68.350±0.713	69.366±1.477	67.545±0.703	67.606±1.114	68.439±1.123	68.959±1.347	68.660±1.474
			3	71.152±0.662	71.489±0.550	70.078±0.775	67.443±0.630	67.779±0.471	63.833±0.657	69.290±1.676	69.639±0.499
Simsiam	64	64	1	65.804±1.483	67.304±1.548	64.193±0.656	63.013±0.545	65.778±1.437	66.854±1.338	67.703±1.756	69.839±1.062
			2	67.733±2.321	66.869±1.405	67.560±1.721	68.619±0.770	66.834±2.385	69.552±1.370	72.328±1.555	71.131±1.084
			3	67.934±1.523	66.765±1.226	67.544±1.397	63.170±1.175	66.784±0.477	68.311±1.442	72.484±1.256	71.186±2.633
		512	1	69.180±1.108	63.923±0.703	65.588±1.240	66.926±1.408	66.846±1.453	66.184±0.542	69.603±0.752	70.266±0.700
			2	67.763±1.538	69.863±0.460	66.952±1.212	62.129±1.662	66.901±1.460	66.422±0.495	71.433±2.326	70.315±1.545
			3	69.753±1.638	67.390±0.623	67.778±1.010	64.460±2.762	69.166±1.473	68.079±0.738	70.736±1.711	69.282±0.976
	256	64	1	67.242±0.938	66.557±1.131	65.250±1.808	65.609±1.776	66.339±0.890	66.087±1.165	66.626±0.674	68.684±1.034
			2	66.908±1.018	63.367±0.813	68.898±1.590	68.056±1.834	68.012±1.602	69.595±1.614	69.553±1.235	64.901±1.042
			3	67.309±1.175	70.902±0.779	68.563±2.455	66.619±1.035	69.799±2.421	68.180±1.456	70.986±0.676	71.386±1.127
		512	1	65.630±1.265	66.563±1.380	66.639±0.788	62.293±1.328	68.962±1.965	70.410±0.889	67.400±1.328	67.052±0.950
			2	67.930±1.086	66.729±1.665	67.091±0.630	70.474±1.065	66.877±2.049	66.267±1.686	71.700±1.019	73.707±0.382
			3	66.388±1.030	69.563±1.598	67.501±1.691	63.994±1.316	68.757±0.915	67.950±1.069	66.768±8.380	61.077±6.460
Barlow Twins	64	64	1	67.203±1.475	66.999±0.507	67.345±0.777	65.449±1.610	68.846±0.912	71.301±0.844	70.657±0.703	70.697±1.124
			2	67.641±0.715	68.667±1.165	68.843±0.667	68.035±1.100	67.315±2.203	63.702±0.881	68.499±1.449	70.163±1.540
			3	70.206±0.634	69.719±1.234	69.355±1.903	68.046±0.834	70.423±2.016	69.325±2.875	68.962±1.249	72.102±1.176
		512	1	68.772±1.239	72.116±0.819	67.489±1.057	66.225±0.550	70.146±0.775	69.981±1.416	66.331±1.576	64.616±1.351
			2	72.721±0.870	71.466±1.418	71.621±0.332	66.926±0.734	69.398±1.974	68.187±1.781	68.207±1.088	69.063±0.717
			3	72.811±0.739	70.220±1.881	71.194±0.875	70.879±1.290	70.415±1.174	69.361±0.486	69.684±1.819	71.035±0.565
	512	64	1	66.669±1.130	70.260±1.854	66.735±0.967	63.740±0.622	67.295±1.016	67.641±0.830	66.267±1.980	63.077±2.069
			2	68.411±1.960	69.911±0.962	68.144±0.568	66.625±1.121	68.695±2.220	66.128±0.892	67.678±1.353	68.751±2.064
			3	70.367±0.879	70.141±0.935	68.005±1.215	67.649±1.504	70.747±0.965	71.287±1.254	69.396±0.633	68.314±1.065
		512	1	67.559±0.470	67.336±0.529	65.880±0.204	65.562±0.215	67.689±1.016	70.947±0.503	65.118±0.326	67.111±0.794
			2	72.823±1.047	71.685±1.517	68.948±0.534	67.251±1.059	68.800±1.568	71.157±1.728	66.589±0.857	65.391±0.986
			3	71.030±0.669	71.575±0.960	71.468±0.370	68.475±1.044	70.350±1.513	69.649±1.087	67.527±1.321	64.516±0.527

Table 6: **Performance on NCI1 dataset.** The red colored values represent the best models in the training and test stages.

6.3 Performance on PROTEIS dataset

Table 7 shows the experiment results of the models trained on PROTEINS dataset.

The best setting in the training stage is the one that uses Barlow Twins with Attribute Masking, batch size set to 64, hidden dimension set to 64, and using a monolayer MLP as the encoder.

The best setting in the test stage is the one that uses Barlow Twins with Attribute Masking, batch size set to 64, hidden dimension set to 64, and using a trilayer

MLP as the encoder.

Model	Batch Size	Hidden Dimension	# Layer of Encoder	Node Droppping		Edge Pertubation		Attribute Masking		Subgraph	
				Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.
SimCLR	64	64	1	73.045±0.753	76.879±2.449	69.818±1.197	67.121±1.588	73.515±1.185	73.470±1.318	73.652±1.992	74.788±0.813
			2	73.697±1.467	64.561±3.432	71.909±1.787	69.242±0.748	71.318±1.308	71.515±2.127	71.030±2.058	69.288±1.919
			3	74.652±1.364	75.924±1.781	74.242±1.113	72.242±1.185	72.530±2.353	74.667±1.185	72.515±3.665	70.515±2.050
		512	1	73.939±1.113	72.015±1.010	71.288±0.643	66.970±1.090	71.394±1.197	68.773±2.156	73.470±0.813	71.697±1.000
			2	75.985±0.909	67.500±1.014	72.515±1.947	71.818±1.494	74.773±0.643	73.894±0.993	75.030±0.407	72.697±2.352
			3	72.970±1.137	66.894±1.163	69.485±1.037	68.303±0.410	73.621±2.914	68.879±3.939	74.712±0.729	75.864±0.822
	256	64	1	69.515±1.084	70.364±1.365	70.333±1.699	72.803±2.353	72.106±0.498	66.242±1.494	73.000±0.761	72.394±0.853
			2	69.682±1.276	67.894±0.743	68.712±0.909	69.182±1.220	67.030±2.100	80.561±0.996	72.636±1.885	69.864±0.966
			3	73.545±1.037	67.106±2.221	72.924±0.810	68.212±0.761	71.121±1.174	75.606±1.506	74.697±1.730	70.894±0.839
		512	1	72.864±1.521	72.182±0.697	74.742±0.822	71.258±0.993	73.424±0.498	67.818±1.725	73.015±0.743	76.712±0.996
			2	72.591±0.416	73.288±0.712	71.606±1.084	74.697±1.014	70.909±0.850	66.742±1.067	74.591±1.359	74.864±1.705
			3	69.288±1.364	76.848±0.617	71.561±0.975	70.424±2.108	74.106±0.507	64.000±1.420	76.136±0.000	73.500±2.183
Simsiam	64	64	1	70.909±1.822	65.515±1.379	72.212±1.797	69.136±1.898	73.470±2.100	75.288±2.095	72.803±0.969	74.364±1.767
			2	71.379±1.018	62.333±1.692	70.727±2.013	69.152±1.235	71.227±2.730	72.227±0.399	73.864±1.179	72.364±1.681
			3	73.288±2.795	70.894±1.391	68.864±1.148	69.530±1.274	70.182±0.801	72.379±0.761	73.212±1.571	68.106±1.387
		512	1	70.258±1.034	74.273±1.574	67.545±2.181	67.788±1.640	73.394±1.379	70.182±1.010	75.182±1.185	69.758±0.644
			2	69.182±1.543	74.136±1.757	70.545±1.867	69.470±0.494	69.788±1.116	74.000±0.743	72.197±1.701	75.894±1.259
			3	72.348±0.895	69.727±0.761	67.848±1.631	69.273±1.449	75.091±1.856	73.712±2.391	71.985±1.858	70.303±1.004
	256	64	1	72.288±0.891	70.652±1.027	68.015±1.518	65.379±1.575	73.909±1.247	74.061±1.037	74.106±1.286	69.985±2.036
			2	71.348±0.646	69.152±1.860	72.303±3.308	64.848±0.969	73.409±1.163	67.742±1.654	74.379±1.220	74.773±1.015
			3	70.530±2.414	69.561±0.775	70.424±3.570	68.848±2.528	70.121±2.414	74.364±2.462	73.939±0.727	68.091±2.919
		512	1	70.182±1.629	70.955±2.531	71.258±1.099	67.682±0.762	70.364±1.233	74.788±0.813	72.530±0.761	70.591±0.753
			2	72.697±3.196	72.894±1.831	71.636±1.876	69.379±1.448	73.606±1.310	68.606±2.592	71.955±1.869	70.000±1.685
			3	73.061±1.667	72.394±3.281	69.348±1.584	66.682±1.736	73.682±1.627	72.409±1.898	73.258±2.033	69.409±2.129
Barlow Twins	64	64	1	74.212±0.407	69.697±1.113	71.121±0.498	60.545±1.220	76.742±1.119	63.212±0.910	71.652±1.037	70.530±0.854
			2	71.636±1.352	73.455±0.757	75.076±0.643	69.091±0.763	72.000±0.801	73.091±2.058	70.212±0.465	73.106±0.643
			3	73.273±1.681	75.970±0.616	73.576±1.785	69.879±1.494	73.727±1.849	83.197±1.314	71.591±1.765	66.242±2.008
		512	1	72.636±0.498	68.697±1.692	72.318±0.456	71.742±0.805	73.182±0.909	74.318±1.802	73.000±0.996	75.833±1.286
			2	70.773±0.743	61.788±0.975	71.939±0.478	75.182±0.493	73.364±1.185	66.561±1.749	75.258±0.984	71.409±1.219
			3	69.712±0.910	76.394±1.347	70.742±1.293	65.833±1.014	73.121±1.122	75.788±1.852	74.106±1.370	75.379±1.684
	512	64	1	72.788±0.474	74.697±0.643	69.061±0.513	69.258±1.058	72.045±0.000	73.182±1.453	70.576±1.494	74.273±1.542
			2	74.136±0.891	78.591±0.996	75.258±0.761	69.076±0.813	68.015±1.318	66.318±1.185	75.061±0.891	70.379±0.643
			3	72.545±1.669	73.273±1.114	74.500±0.812	74.227±0.498	74.015±1.290	76.242±1.971	72.697±1.346	69.091±1.134
		512	1	73.985±0.996	77.758±2.073	70.970±0.813	71.424±1.332	73.485±0.784	70.061±1.326	72.061±1.037	70.288±1.885
			2	68.667±1.681	73.167±0.853	72.227±1.197	78.303±1.660	71.621±0.407	78.152±0.478	71.727±0.498	69.470±0.643
			3	72.045±2.132	72.500±0.784	71.530±1.741	75.909±3.083	72.470±0.941	67.439±1.451	70.773±2.065	74.030±1.180

Table 7: **Performance on PROTEINS dataset.** The red colored values represent the best models in the training and test stages.

6.4 Performance on DD dataset

Table 8 shows the experiment results of the models trained on DD datasets.

The best setting in the training stage is the one that uses SimCLR with Attribute Masking, batch size set to 64, hidden dimension set to 512, and using a monolayer MLP as encoder.

The best setting in the test stage is the one that uses SimCLR with Edge Pertubation, batch size set to 64, hidden dimension set to 512, and using a trilayer MLP as encoder.

Model	Batch Size	Hidden Dimension	# Layer of Encoder	Node Droppping		Edge Pertubation		Attribute Masking		Subgraph	
				Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.
SimCLR	64	64	1	74.576±1.412	69.242±1.318	74.985±1.270	73.576±1.464	73.379±1.038	70.061±1.270	72.788±1.610	74.758±2.331
			2	75.455±1.318	73.818±0.788	75.818±1.712	72.197±1.409	73.470±0.755	73.227±1.583	73.500±2.242	72.894±2.324
			3	77.076±1.168	73.985±0.712	75.227±1.120	68.076±5.247	74.424±1.909	73.485±3.399	76.258±1.932	72.606±3.293
		512	1	76.515±1.800	76.636±1.651	75.742±0.773	75.561±1.706	77.515±1.799	75.212±4.819	75.864±1.366	73.682±1.972
			2	76.682±1.345	73.894±0.740	75.545±1.025	71.121±2.823	75.303±1.253	74.167±4.285	75.470±1.676	68.470±2.970
			3	73.303±2.174	74.288±1.510	74.167±1.094	84.364±4.023	75.470±1.653	75.227±1.846	76.530±1.526	73.515±3.762
	128	64	1	76.121±1.090	74.576±3.677	77.227±2.210	77.455±1.556	72.106±1.776	70.318±5.482	74.621±0.569	72.894±2.533
			2	74.788±1.603	77.182±1.863	74.864±1.159	76.424±2.003	73.000±0.689	73.652±3.136	75.924±1.021	71.136±5.300
			3	75.848±0.873	69.758±2.437	75.576±0.893	74.242±1.117	72.576±1.763	73.515±3.799	76.288±1.581	75.485±7.635
		512	1	75.773±0.957	78.348±1.769	76.955±1.072	75.636±1.500	72.273±1.976	78.727±3.594	75.500±1.456	75.121±4.102
			2	74.985±2.109	71.682±0.729	74.970±1.414	76.712±1.505	76.939±2.851	74.288±2.357	75.152±1.633	74.818±1.668
			3	76.545±1.504	69.712±3.905	74.182±0.729	76.985±3.077	73.682±4.102	74.515±2.036	74.364±1.287	75.773±2.661
Simsiam	64	64	1	74.667±1.580	77.106±4.267	75.076±2.108	74.621±2.409	73.712±0.978	73.439±1.260	67.703±1.756	69.839±1.062
			2	75.242±3.170	76.455±4.297	75.091±2.269	72.364±3.635	73.182±1.134	69.470±1.521	72.328±1.555	71.131±1.084
			3	75.379±2.137	76.258±3.353	75.470±1.642	71.197±1.380	72.045±1.671	75.152±1.386	72.484±1.256	71.186±2.633
		512	1	76.212±2.400	74.788±2.553	74.803±2.076	74.864±3.788	74.212±1.332	75.970±2.971	69.603±0.752	70.266±0.700
			2	75.258±1.551	76.030±4.115	76.106±1.557	76.333±3.068	72.924±1.262	73.439±3.464	71.433±2.326	70.315±1.545
			3	75.727±2.294	74.515±1.639	75.833±2.084	75.288±3.106	71.530±2.684	72.197±2.876	70.736±1.711	69.282±0.976
	128	64	1	74.591±1.531	76.803±2.619	76.788±1.106	76.318±0.792	72.394±0.743	71.909±2.660	66.626±0.674	68.684±1.034
			2	75.379±1.131	73.333±2.062	74.697±2.026	74.061±2.565	72.379±1.588	70.136±5.839	69.553±1.235	64.901±1.042
			3	76.212±0.842	78.212±2.756	71.727±2.488	71.682±6.549	73.652±1.093	69.818±4.433	70.986±0.676	71.386±1.127
		512	1	74.864±2.867	74.621±1.121	75.909±2.635	73.894±1.916	74.591±2.163	73.727±1.548	67.400±1.328	67.052±0.950
			2	76.894±1.033	75.621±3.495	76.364±2.411	76.833±4.807	74.485±1.215	71.879±2.989	71.700±1.019	73.707±0.382
			3	75.924±2.575	72.909±2.720	74.985±3.224	73.258±2.703	74.955±3.303	72.318±3.345	66.768±8.380	61.077±6.460
Barlow Twins	64	64	1	75.212±1.929	77.818±4.042	75.318±0.811	77.152±0.649	74.045±0.716	73.606±2.382	72.697±2.193	74.000±4.773
			2	77.136±2.400	75.424±5.550	75.439±1.451	75.712±3.359	72.485±1.479	68.652±4.021	72.848±2.392	69.212±1.520
			3	77.455±1.797	77.909±2.332	72.985±1.815	73.061±4.373	72.848±1.560	72.561±2.334	71.803±1.617	75.409±5.511
		512	1	76.379±0.626	73.652±1.997	75.712±1.019	74.227±3.366	74.773±1.586	72.424±4.891	71.894±1.652	71.545±4.248
			2	74.348±1.809	75.667±4.428	74.939±1.363	72.242±4.101	74.773±1.396	74.530±5.156	73.424±0.775	74.015±2.086
			3	74.424±0.668	79.939±5.046	73.500±2.951	74.500±4.521	73.576±1.222	74.818±6.909	73.970±0.930	74.985±1.478
	128	64	1	74.894±1.606	77.227±3.591	75.121±1.017	72.303±1.654	73.848±2.228	74.803±4.171	75.894±1.615	72.591±1.989
			2	76.485±0.694	74.394±2.989	74.909±1.569	75.803±2.738	72.348±0.948	72.076±3.104	74.167±2.025	76.515±2.734
			3	75.424±1.078	72.136±4.707	73.939±1.708	72.424±3.512	71.667±0.847	68.985±1.789	73.197±1.707	69.682±2.111
		512	1	74.727±1.858	76.152±4.943	74.273±2.157	77.758±2.156	74.909±0.406	73.364±1.387	72.758±1.822	72.758±2.091
			2	76.197±1.250	75.712±2.535	73.652±1.064	71.833±2.745	74.833±2.291	74.015±3.088	73.591±1.498	71.727±3.109
			3	74.712±1.368	74.470±3.525	73.682±0.975	77.258±2.469	75.318±2.128	74.091±5.902	73.318±2.219	72.091±7.857

Table 8: **Performance on DD dataset.** The red colored values represent the best models in the training and test stages.

References

[1] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey,"IEEE transactions on pattern analysis and machine intelligence, vol. 43, no. 11, pp. 4037–4058, 2020. 1

[2] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations,"Advances in Neural Information Processing Systems, vol. 33, pp. 5812–5823, 2020. 1, 4

[3] Y. Zhu, Y. Xu, Q. Liu, and S. Wu, "An empirical study of graph contrastive learning,"arXiv preprint arXiv:2109.01116, 2021. 2.1

[4] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," inProceedings of the IEEE conference on computer vision and pattern recognition, pp. 3733–3742, 2018. 2.1, **??**

[5] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," inInternational Conference on Machine Learning, pp. 4116–4126, PMLR, 2020. 2.1

[6] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," inProceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 9729–9738, 2020. 2.1, **??**

[7] X. Chen, H. Fan, R. Girshick, and K. He, "Improved baselines with momentum contrastive learning,"arXiv preprint arXiv:2003.04297, 2020. 2.1, **??**

[8] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," inInternational conference on machine learning, pp. 1597–1607, PMLR, 2020. 2.1, **??**, 4.1

[9] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton, "Big self-supervised models are strong semi-supervised learners,"arXiv preprint arXiv:2006.10029, 2020. 2.1, **??**

[10] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," inProceedings of the European conference on computer vision (ECCV), pp. 132–149, 2018. 2.1, **??**

[11] Y. M. Asano, C. Rupprecht, and A. Vedaldi, "Self-labelling via simultaneous clustering and representation learning,"arXiv preprint arXiv:1911.05371, 2019. 2.1, **??**

[12] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments,"Advances in Neural Information Processing Systems, vol. 33, pp. 9912–9924, 2020. 2.1, **??**

[13] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar,et al., "Bootstrap your own latent-a new approach to self-supervised learning,"Advances in Neural Information Processing Systems, vol. 33, pp. 21271–21284, 2020. 2.1, **??**

[14] X. Chen and K. He, "Exploring simple siamese representation learning," inProceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 15750–15758, June 2021. 2.1, **??**

[15] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," 2021. 2.1, **??**

[16] N. Park, A. Kan, X. L. Dong, T. Zhao, and C. Faloutsos, "Estimating node importance in knowledge graphs using graph neural networks," inProceedings

of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 596–606, 2019. 2.2

[17] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," inThe world wide web conference, pp. 417–426, 2019. 2.2

[18] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," inProceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 974–983, 2018. 2.2

[19] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks,"Advances in Neural Information Processing Systems, vol. 32, 2019. 2.2

[20] J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics,"Machine Learning: Science and Technology, vol. 2, no. 2, p. 021001, 2020. 2.2

[21] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann,et al., "A deep learning approach to antibiotic discovery,"Cell, vol. 180, no. 4, pp. 688–702, 2020. 2.2

[22] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks,"IEEE transactions on neural networks and learning systems, vol. 32, no. 1, pp. 4–24, 2020. 2.2

[23] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs,"Advances in neural information processing systems, vol. 30, 2017. 2.2

[24] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks,"arXiv preprint arXiv:1609.02907, 2016. 2.2

[25] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?,"arXiv preprint arXiv:1810.00826, 2018. 2.2

[26] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs,"arXiv preprint arXiv:2007.08663, 2020. 3.1

[27] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks,"arXiv preprint arXiv:2003.00982, 2020. 3.1

[28] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity,"Journal of medicinal chemistry, vol. 34, no. 2, pp. 786–797, 1991. 3.1, **??**

[29] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification,"Knowledge and Information Systems, vol. 14, no. 3, pp. 347–375, 2008. 3.1, **??**

[30] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels,"Bioinformatics, vol. 21, no. suppl_1, pp. i47–i56, 2005. 3.1, **??**, **??**

[31] K. Ding, Z. Xu, H. Tong, and H. Liu, "Data augmentation for deep graph learning: A survey,"arXiv preprint arXiv:2202.08235, 2022. 3.2

[32] T. Zhao, G. Liu, S. Günnemann, and M. Jiang, "Graph data augmentation for graph machine learning: A survey,"arXiv preprint arXiv:2202.08871, 2022. 3.2

[33] P. Bielak, T. Kajdanowicz, and N. V. Chawla, "Graph barlow twins: A self-supervised representation learning framework for graphs,"arXiv preprint arXiv:2106.02466, 2021. 3.6

[34] J. Huan, W. Wang, A. Washington, J. Prins, R. Shah, and A. Tropsha, "Accurate classification of protein structural families using coherent subgraph analysis," inBiocomputing 2004, pp. 411–422, World Scientific, 2003. 4.5

[35] M. Choi, W. Shin, Y. Lu, and S. Kim, "Triangular contrastive learning on molecular graphs,"arXiv preprint arXiv:2205.13279, 2022. 5

[36] X. Ma, Z. Gao, Q. Hu, and M. AbdelHady, "Hcl: Hybrid contrastive learning for graph-based recommendation," 5

[37] Y. Yang, R. Miao, Y. Wang, and X. Wang, "Contrastive graph convolutional networks with adaptive augmentation for text classification,"Information Processing & Management, vol. 59, no. 4, p. 102946, 2022. 5

[38] Y. Shen, J. Yan, C.-W. Ju, J. Yi, Z. Lin, and H. Guan, "Improving subgraph representation learning via multi-view augmentation,"arXiv preprint arXiv:2205.13038, 2022. 5

[39] Z. Hou, X. Liu, Y. Dong, C. Wang, J. Tang,et al., "Graphmae: Self-supervised masked graph autoencoders,"arXiv preprint arXiv:2205.10803, 2022. 5