# An Empirical Study of Graph Contrastive Learning
## *Supplementary Material*

**Yanqiao Zhu**[1,2] **Yichen Xu**[3] **Qiang Liu**[1,2] **Shu Wu**[1,2*]
[1]Center for Research on Intelligent Perception and Computing
National Laboratory of Pattern Recognition
Institute of Automation, Chinese Academy of Sciences
[2]School of Artificial Intelligence, University of Chinese Academy of Sciences
[3]School of Computer Science, Beijing University of Posts and Telecommunications
yanqiao.zhu@cripac.ia.ac.cn linyxus@bupt.edu.cn {qiang.liu,shu.wu}@nlpr.ia.ac.cn

## A Reproducibility of Experiments

### A.1 Brief Introduction to PyGCL

PyGCL is a PyTorch-based battery-included toolkit for implementing Graph Contrastive Learning (GCL) models. We use it extensively in our benchmarking study to implement and execute all experiments. PyGCL provide encapsulated implementations of four main components of GCL algorithms, corresponding to our proposed design dimensions in this work:

- **Graph augmentation** transforms input graphs into congruent graph views.
- **Contrasting architectures and modes** generate positive and negative pairs according to node and graph embeddings.
- **Contrastive objectives** compute the likelihood scores for positive and negative pairs.
- **Negative mining strategies** improve the negative sample set by considering the relative similarity (i.e. hardness) of negative samples.

PyGCL also implements utility functions for model training, performance evaluation, and experiment management.

Table S1: Graph augmentation schemes and their corresponding classes in PyGCL.

| Augmentation | Class name |
|---|---|
| Edge Adding (EA) | `EdgeAdding` |
| Edge Removing (ER) | `EdgeRemoving` |
| Node Feature Masking (FM) | `FeatureMasking` |
| Node Feature Dropout (FD) | `FeatureDropout` |
| Edge Attribute Masking (EAM) | `EdgeAttrMasking` |
| Edge Attribute Dropout (EAD) | `EdgeAttrDropout` |
| Personalized PageRank (PPR) | `PPRDiffusion` |
| Markov Diffusion Kernel (MDK) | `MarkovDiffusion` |
| Node Dropping (ND) | `NodeDropping` |
| Subgraphs induced by Random Walks (RWS) | `RWSampling` |

*To whom correspondence should be addressed.

35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks.

### A.1.1 Package Reference

**Graph augmentation.** In `GCL.augmentor`, PyGCL provides the `Augmentor` base class, which offers a unified interface for graph augmentation functions. A list of graph augmentation functions and their corresponding class names in PyGCL in given in Table S1. Due to complexity issues, Edge Flipping (EF) is implemented as a composition of Edge Adding (EA) and Edge Removing (ER).

PyGCL supports composing arbitrary numbers of augmentation functions together. The `Compose` class can be used for jointly using a list of augmentation functions consecutively. Additionally, the `RandomChoice` class can be used to randomly draw a few augmentation functions each time.

**Contrasting architectures and modes.** Existing GCL architectures could be grouped into two lines: negative-sample-based methods and negative-sample-free ones.

- **Negative-sample-based approaches** can either have one single branch or two branches of graph encoders. In single-branch contrasting, we only need to construct one graph view and perform contrastive learning within this view. In dual-branch models, we generate two graph views and perform contrastive learning within and across views.
- **Negative-sample-free approaches** eschew the need of explicit negative samples. Currently, PyGCL supports the bootstrap-style contrastive learning as well contrastive learning by variance reduction within embeddings (such as Barlow Twins and VICReg).

Internally, PyGCL calls `DefaultSampler` classes in `GCL.model` that receive embeddings and produce positive/negative masks. PyGCL implements three contrasting modes: (a) Local-Local (L–L), (b) Global-Global (G–G), and (c) Global-Local (G–L) modes. Methods of L–L and G–G modes contrast embeddings at the same scale and the latter G–L one performs cross-scale contrasting. All supported contrasting architectures and modes and their implementation in PyGCL are summarized in Tables S2 and S3.

**Contrastive objectives.** In `GCL.loss`, PyGCL provides implementation for the following contrastive objectives: InfoNCE, Jensen-Shannon Divergence (JSD), Triplet Margin (TM), Bootstrapping Latent (BL), Barlow Twins (BT), and VICReg, as summarized in Table S4. All these objectives are able to contrast any arbitrary positive and negative pairs, except for Barlow Twins and VICReg losses that perform contrastive learning within embeddings.

Table S2: Graph contrasting architectures and its corresponding classes in PyGCL.

| Contrasting architectures | Supported contrastive modes | Need negative samples | Class name |
|---|---|---|---|
| Single-branch contrasting | G–L only | ✓ | `SingleBranchContrast` |
| Dual-branch contrasting | L–L, G–G, and G–L | ✓ | `DualBranchContrast` |
| Bootstrap contrasting | L–L, G–G, and G–L | ✗ | `BootstrapContrast` |
| Within-embedding contrasting | L–L and G–G | ✗ | `WithinEmbedContrast` |

Table S3: Graph contrasting modes and their corresponding classes in PyGCL.

| Contrasting modes | Class name |
|---|---|
| Same-scale contrasting (L–L and G–G) | `SameScaleSampler` |
| Cross-scale contrasting (G–L) | `CrossScaleSampler` |

Table S4: Contrastive objectives and their corresponding classes in PyGCL.

| Augmentation | Class name |
|---|---|
| InfoNCE loss | `InfoNCELoss` |
| Jensen-Shannon Divergence (JSD) loss | `JSDLoss` |
| Triplet Margin (TM) loss | `TripletLoss` |
| Bootstrapping Latent (BL) loss | `BootstrapLoss` |
| Barlow Twins (BT) loss | `BTLoss` |
| VICReg loss | `VICRegLoss` |

Table S5: Negative mining strategies and its corresponding implementation in PyGCL.

| Negative mining strategy | Class name |
|---|---|
| Hard negative mixing | `GCL.models.HardMixing` |
| Conditional negative sampling | `GCL.models.Ring` |
| Debiased contrastive objective | `GCL.losses.DebiasedInfoNCE`, `GCL.losses.DebiasedJSD` |
| Hardness-biased negative sampling | `GCL.losses.HardnessInfoNCE`, `GCL.losses.HardnessJSD` |

Table S6: Evaluators and their corresponding implementation in PyGCL.

| Evaluator | Class name |
|---|---|
| Trainable Logistic regression | `LRTrainableEvaluator` |
| Logistic regression based on scikit-learn | `LRSklearnEvaluator` |
| Support vector machine based on scikit-learn | `SVMEvaluator` |
| Random forest based on scikit-learn | `RFEvaluator` |

**Negative mining strategies.** PyGCL implements several negative sampling strategies: Hard Negative Mixing (HNM), Conditional Negative Sampling (CNS), Debiased Contrastive Learning (DCL) objective, and Hardness-Biased Negative Mining (HBNM), as summarized in Table S5.

The former two models serve as an additional sampling step similar to existing `Sampler` instances and can be used in conjunction with any objectives. The last two objectives are only compatible with the InfoNCE and JSD losses.

**Utilities for evaluating embeddings.** PyGCL further provides a variety of evaluator functions to evaluate the embedding quality in `GCL.eval`. Specifically, we provide two kinds of evaluator base classes: `BaseTrainableEvaluator` for trainable (e.g., logistic regression) evaluation and `BaseSKLearnEvaluator` for evaluation based on Scikit-learn [1]. These two base classes share almost identical call signatures. After feeding the evaluator with embeddings and ground-truths, evaluation results with metrics as keys and mean and standard deviation as values will be returned. Users can define their own evaluation by inheriting one of the two base classes. All available evaluation classes are summarized as in Table S6.

In addition, we provide two functions to generate dataset splits: `random_split` (for generating split indices for training, test, and validation sets) and `from_PyG_split` (for converting from predefined PyTorch-Geometric [2] split indices of training, test, and validation sets).

### A.1.2 Example Usage

We showcase an example of implementing existing work using PyGCL: we can implement GRACE [3] with very few lines of code. The code snippet is shown as in Listing S1.

Listing S1: Implementing GRACE [3] with PyGCL.

```
import GCL.loss as L
import GCL.augmentor as A
from GCL.model import DualBranchContrast
from GCL.eval import random_split, LRTrainableEvaluator

# Graph augmentator
aug1 = A.Compose([A.EdgeRemoving(pe=0.3), A.FeatureMasking(pf=0.3)])
aug2 = A.Compose([A.EdgeRemoving(pe=0.3), A.FeatureMasking(pf=0.3)])

# Graph encoder
gconv = GConv(input_dim=dataset.num_features, hidden_dim=32,
    activation=torch.nn.ReLU, num_layers=2)
encoder_model = Encoder(encoder=gconv, augmentor=(aug1, aug2),
    hidden_dim=32, proj_dim=32)

```

```
# Contrasting model
contrast_model = DualBranchContrast(loss=L.InfoNCE(tau=0.2),
    mode='L2L', intraview_negs=True)
z, z1, z2 = contrast_model(data.x, data.edge_index, data.edge_attr)
h1, h2 = [contrast_model.project(x) for x in [z1, z2]]
loss = contrast_model(h1, h2)

# Evaluator
split = random_split(num_samples=z.size(0), num_splits=10,
    train_ratio=0.1, test_ratio=0.8)
evaluator = LRTrainableEvaluator(
    input_dim=z.size(1), num_classes=data.y.max().item() + 1,
    metrics={'micro_f1': partial(f1_score, average='micro'), 'macro_f1':
        partial(f1_score, average='macro')},
    split=split, device=data.x.device, test_metric='micro_f1')
test_result = evaluator(z, data.y)
```

### A.2 Instructions for Reproducing Experimental Results

We list instructions for reproducing the experiments in this paper. We provide trial scripts in the experiment that can run GCL tasks on different configurations. These scripts are hosted along the PyGCL project, yet on a different repository. You can access them at https://github.com/GraphCL/BenchmarkingGCL.

- Experiments investigating the impact of data augmentation (Table 3, Figures 3 and 4) can be reproduced by executing `trial.py`, passing the configuration file to the `--config` argument and change the data augmentation schemes with the `--augmentor1:scheme` and `--augmentor2:scheme` arguments. For example, the result on the Wiki dataset with Edge Removing (ER) augmentation can be reproduced by executing `python trial.py --config params/wikics.json --augmentor1:scheme ER --augmentor2:scheme ER`.
- Sensitivity analysis with varied topology augmentation probabilities (Figure 2) can be reproduced using `trial.py` similarly by specifying the base configuration and modifying data augmentation schemes via command-line arguments. The probability of a specific augmentation can be modified through the `--augmentor1:ER:prob` and `--augmentor2:ER:prob` argument, taking Edge Removing (ER) as an example.
- Experiments on different contrastive objectives and contrasting modes can be reproduced using the `trial.py` and `BGRL_trial.py` scripts with arguments specified. To specify objectives and modes, we can change the `--obj:loss` and `--mode` arguments respectively. Specially, experiments involving the Bootstrapping Latent (BL) architecture should use the dedicated `BGRL_trial.py` script, while Barlow Twins (BT) and VICReg can be run with `trial.py` by passing `bt` and `vicreg` to the `--obj:loss` argument respectively.
- Sensitivity analysis of the temperature $\tau$ in InfoNCE objective (Figure 5) can be reproduced by running `trial.py` with different $\tau$ values filling in the `--obj:infonce:tau` argument.
- Performance of various negative mining strategies (Figure 6) can be reproduced by running `trial.py` and specifying the negative mining strategy in `--obj:loss`.
- Ablation studies on batch normalization in Bootstrapping Latent objective (Table S10) can be reproduced with the `BGRL_trial.py` script. The type of the encoder, projector and predictor normalizations can be specified in the `--obj:bl:encoder_norm`, `--obj:bl:projector_norm`, and `--obj:bl:predictor_norm` arguments.

## B Implementation Details and Experimental Protocols

**Implementation details.** Unlike GRACE [3], for all objectives we include only inter-view negative samples. In every experiment, we use grid search to find the optimal embedding dimension among [64, 128, 256, 512], learning rate among [0.0001, 0.001, 0.01, 0.1], the number of GNN layers among [2, 3, 4], and weight decay parameter among $[10^{-5}, 10^{-6}, 10^{-7}, 5 \times 10^{-8}, 10^{-8}]$. To ensure

Table S7: Summary of large-scale datasets.

| Dataset | Domain | Task | #Graphs | Avg. #nodes | Avg. #edges | #Features | Metric |
|---|---|---|---|---|---|---|---|
| ogbg-molhiv | Molecules | Binary graph classification | 41,127 | 25.5 | 27.5 | 9 (nodes) 3 (edges) | ROC-AUC |
| PCQM4M-10K | | Graph regression | 10,000 | 14.1 | 29.1 | | MAE |

convincing experiments and observations, we first perform an exhaustive search over the entire design space. Then, we select and report representative results to reveal common, useful practices. Then, with the aim of conducting controlled experiments, we fix as many variables, e.g., the GNN encoder architecture, embedding dimensions, the number of epochs, and activation functions, as possible for every dataset. In particular, when examining contrastive objectives and contrasting modes, we slightly fine-tune the hyperparameter of learning rate and objective-specific parameters (e.g., $\tau$ in InfoNCE), since different objective functions give contrastive scores in different magnitudes.

The temperature $\tau$ in the InfoNCE loss is chosen from 0.1 to 0.9. The batch size for graph datasets is chosen between [32, 64, 128, 256, 512]. We also apply the early stopping strategy with a window size of 50 and the model with the lowest loss will be used in evaluation. All experiments are executed on GeForce RTX 3090 GPUs with 24GB memory. All models are implemented with PyTorch 1.9 [4] and PyTorch Geometric 1.7.0 [2]. All datasets can be accessed at PyTorch-Geometric [2] and TUDataset [5].

**Evaluation protocols.** We mainly evaluate models with different design considerations on two benchmark tasks: (1) unsupervised node classification and (2) unsupervised graph classification. For all experiments, we follow the linear evaluation scheme used in Veličković et al. [6], where the models are first trained in an unsupervised manner, and then the frozen embeddings are fed into a $\ell_2$-regularized logistic regression classifier to fit the labeled data. Following previous work, we run the model with ten random splits (10% for training, 10% for validation, and the remaining 80% data for testing) and report the averaged accuracies (%) as well as standard deviation scores.

# C Additional Experiments

## C.1 Large-Scale Evaluation

Besides standard classification tasks, we further evaluate GCL on large-scale datasets from the Open Graph Benchmarks [7, 8].

**Summary of datasets, tasks, and metrics.** We use two additional datasets ogbg-molhiv for binary graph classification and a downsampled version of PCQM4M-LSC for graph regression. In the two datasets, each graph represents a molecule, where nodes and edges correspond to atoms and chemical bonds, respectively. Details of statistics of the two datasets are summarized in Table S7.

- For ogbg-molhiv, the task is to predict a certain molecular property, measured in terms of Receiver Operating Characteristic Area Under Curve (ROC-AUC) scores. We follow the official scaffold splitting where structurally different molecules are separated into different subsets.
- For PCQM4M-LSC, we randomly subsample 10K graphs according to PubChem ID (CID) and denote the resulting dataset as PCQM4M-10K. The regression task is to predict HOMO-LUMO energy gap in electronvolt (eV) given 2D molecular graphs. We report the model performance in terms of Mean Absolute Error (MAE).

Since there are edge features associated with each graph, we use the GINE model proposed in Hu et al. [9] as the encoder. For edge features, we examine two extra augmentation schemes Edge Attribute Masking (EAM) and Edge Attribute Dropout (EAD), similar to node-level FM and FD (denoted as NFM and NFD in the performance table). All other evaluation protocols remain the same as previously stated.

**Experiments on different augmentation schemes.** In Table S8, we report the performance on the two large-scale datasets with different data augmentation schemes while keeping the contrasting mode to global-global and the objective to InfoNCE. We have findings consistent with those in

Table S8: Performance (ogbg-molhiv in ROC-AUC and PCQM4M-10K in MAE) with different augmentation schemes.

| Augmentation | ogbg-molhiv | PCQM4M-10K |
|---|---|---|
| None | 55.86±2.02 | 0.604142±0.046415 |
| ER | 63.21±2.69 | 0.545553±0.011173 |
| ND | **65.18±2.53** | **0.528665±0.011708** |
| RWS | 63.36±3.75 | 0.545109±0.007593 |
| NFD | 57.82±1.67 | 0.573832±0.012382 |
| NFM | 56.79±1.86 | 0.568972±0.007222 |
| EAM | 56.88±3.90 | 0.573943±0.008932 |
| EAD | 56.78±2.38 | 0.579321±0.018473 |

Table S9: Performance (ogbg-molhiv in ROC-AUC and PCQM4M-10K in MAE) with different contrasting modes and contrastive objectives. The best performing results for objectives (row-wise) and contrasting modes (column-wise) are highlighted in boldface and underline respectively.

| Obj. | ogbg-molhiv | | | PCQM4M-10K | | |
|---|---|---|---|---|---|---|
| | L–L | G–L | G–G | L–L | G–L | G–G |
| InfoNCE | **65.22±2.92** | 62.08±1.87 | **64.12±2.07** | 0.537680±0.023940 | **0.568529±0.012488** | **0.532407±0.015661** |
| JSD | 62.02±2.98 | **62.98±2.42** | 61.51±2.01 | 0.587797±0.017347 | 0.578025±0.022175 | 0.565639±0.030184 |
| TM | 60.56±3.12 | 60.12±2.32 | 60.46±0.65 | **0.532207±0.026155** | 0.595626±0.074961 | 0.574360±0.011049 |

Observation 1 that (1) both ER and ND have lead to competitive performance consistently on the two datasets; (2) RWS is inferior to the other two topology augmentation schemes due to the limited size of graphs, as pointed out in Observation 1; (3) using feature augmentation (NFM, NFD, EAM, and EAD) alone cannot achieve satisfying performance. We suspect that for these two molecule datasets, structural information plays a more important role in GCL.

**Experiments on contrasting modes and contrastive objectives.** Then, we examine different contrasting modes and contrastive objectives, where the results are shown in Table S9. Data augmentation schemes are set to the combination of ER and NFM in all variants. From the table, we observe trends consistent to those in Table 4b that the global-global mode yields competitive performance on graph tasks and InfoNCE outperforms other objectives under most settings.

### C.2 Ablation Studies on Batch Normalization of the Bootstrapping Latent Loss

Previous work [10] empirically demonstrates that Batch Normalization (BN) compensates for improper initialization for contrastive learning models, instead of introducing implicit negative samples. To further demonstrate this phenomenon in GCL, we perform ablation studies by using BN or not in three critical components in the BL loss, i.e. the GNN encoder, the projector, and the predictor, on the node classification task. The results in Table S10 show that using BN in the GNN encoder *solely* is almost sufficient to obtain promising performance.

Table S10: Ablation studies on batch normalization of the bootstrapping latent loss. ✓ denotes having an extra BN layer in corresponding component.

| Encoder | Projector | Predictor | Wiki | CS | Physics | Computer |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | 80.61±0.04 | 93.29±0.07 | 95.31±0.01 | 89.81±0.07 |
| ✓ | ✓ | ✗ | 79.93±0.06 | 93.08±0.05 | 95.24±0.01 | 88.64±0.04 |
| ✓ | ✗ | ✓ | 80.01±0.03 | 92.93±0.10 | 95.00±0.04 | 87.42±0.15 |
| ✓ | ✗ | ✗ | 79.54±0.08 | 92.87±0.04 | 95.11±0.08 | 87.97±0.20 |
| ✗ | ✓ | ✓ | 79.42±0.09 | 93.53±0.03 | 95.23±0.05 | 87.45±0.11 |
| ✗ | ✓ | ✗ | 79.46±0.03 | 92.82±0.06 | 95.15±0.02 | 88.32±0.08 |
| ✗ | ✗ | ✓ | 79.96±0.04 | 78.32±0.69 | 85.91±3.32 | 57.80±0.33 |
| ✗ | ✗ | ✗ | 78.73±0.20 | 74.87±0.18 | 58.50±0.31 | 62.88±0.08 |

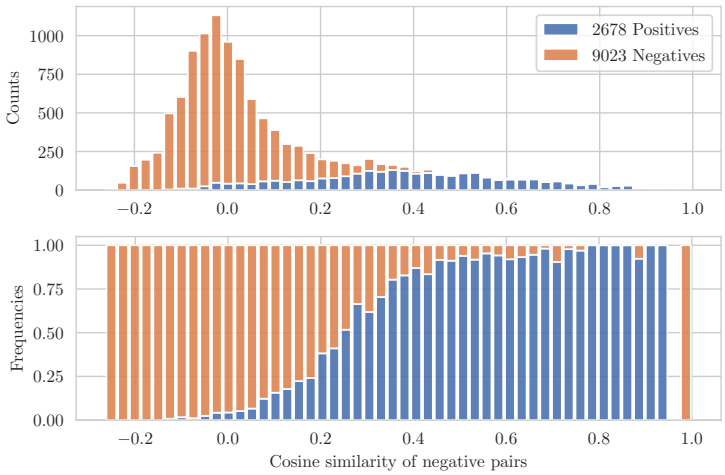


Figure S1: A histogram of negatives and their semantic similarity scores with an anchor node from the Wiki dataset. With the similarity to the anchor node increasing, there are more positive samples (i.e. false negatives), leading to wrong selection of hard negatives.

## D Discussions on Negative Mining Strategies for GCL

The studied negative mining schemes, originally designed for grid data, measures the relative hardness of negative pairs using dot-product of embeddings. Our observation in Section 3.3 is that adopting these hard negative mining schemes naïvely for graph-structured data may end up selecting hard but false negative samples, possibly due to the smoothing nature of GNNs.

To see this clearly, in Figure S1, we present a histogram of negatives and their semantic similarity scores with a randomly selected anchor node from the Wiki dataset. It is evident from the figure that with the similarity increasing, there are more positive samples (i.e. false negatives) as shown in blue, possibly leading to wrong selection of hard negatives. Furthermore, at the beginning of training, node embeddings suffer from poor quality, which may be another obstacle of selecting true hard negative samples. Therefore, by selecting hard negative samples merely according to similarity measure of embeddings, these hard negatives are potentially *positive samples*, which produces adverse learning signals to the contrastive objective.

## E Background and Related Work

Recently, Self-Supervised Learning (SSL) has shown its great capability in alleviating the label scarcity problem when applying machine learning models. As a subfield of SSL, Contrastive Learning (CL) has attained increasing popularity due to its simplicity and promising empirical performance. In essence, CL aims to learn discriminative representations by repulsing negative pairs and attracting positive pairs. Initially proposed for learning visual representations, visual CL work usually generates multiple views of the input images using data augmentation [11–14] at first. Under this multiview setting, congruent samples corresponding to one specific image in these views are usually considered as positive samples and other samples within the same batch [15, 16] or in an extra memory banks [17, 18] are used as negatives.

Graph Contrastive Learning (GCL) adapts the idea of CL to the graph domain. However, due to the complex, irregular structure of graph data, how to design strategies for constructing positive and negative samples for GCL is more challenging than CL for visual data or natural language data. Prior GCL work proposes data augmentation techniques for graph-structured data, explores different contrasting modes to build contrastive pairs, and examines various contrastive objectives that score positive and negative pairs. Regarding graph augmentation, many previous studies propose data augmentation techniques for general graph-structured data [19–21]. For GCL, You et al. [22] study four different data augmentation including node dropping, edge perturbation, subgraph sampling, and feature masking; MVGRL [23] employs graph diffusion to generate graph views with more global

information; GCA [24] proposes adaptive augmentation techniques to further consider important topology and attribute information. Many other methods explore various contrasting modes using different parts of a graph. For example, GRACE [3] contrasts node-node pairs, GraphCL [22] considers graph-graph pairs, while DGI [25], InfoGraph [26], and MVGRL [23] constructs graph-node contrasting pairs. Although there has been several survey papers on self-supervised graph representation learning [27–29], to the best of our knowledge, none of existing work provides rigorous empirical evidence on the impact of each component involved in GCL.

# F Details of Design Dimensions

## F.1 Data Augmentation

### F.1.1 Topology Augmentation

Topology augmentation perturbs the structural space of the graph by modifying its adjacency matrix $\boldsymbol{A}$. In this work, we consider the following types of topology augmentation:

- **Edge perturbation** randomly adds and/or removes a portion of edges in the original graph. Formally, we sample a random masking matrix $\widetilde{\boldsymbol{R}} \in \{0,1\}^{N\times N}$, where each entry is drawn from a Bernoulli distribution $\widetilde{\boldsymbol{R}}_{ij} \sim \mathrm{Bern}(p_r)$. Here $p_r$ is the probability for adding/removing an edge. The resulting adjacency matrix can be computed as

$$\widetilde{\boldsymbol{A}} = \boldsymbol{A} \odot \widetilde{\boldsymbol{R}}, \tag{S1}$$

  where $\odot$ is an element-wise operator. In this work, we consider three variants of edge perturbation, including **Edge Removing (ER)**, **Edge Adding (EA)**, and **Edge Flipping (EF)**, corresponding to $\odot$ being instantiated as add, subtract, and exclusive or operations, respectively.
- **Node perturbation** considers topology transformation in a node-wise manner. Because of the transductive nature of most GNN models, we mainly consider **Node Dropping (ND)** in this category. Similarly to ER, we assign each node with a probability $p_d$ being dropped. ND is equivalent to masking out all adjacent edges to the dropped node in the adjacency matrix.
- **Subgraph sampling** modifies the graph structure at the subgraph level. In this work, we are primarily concerned with **Subgraphs induced by Random Walks (RWS)**. Starting from a node, we sample a random walk that has a probability $p_{ij}$ to travel from node $v_i$ to $v_j$ and a probability $p_e$ to return to the start node [30]. Then, nodes appearing in this walk sequence are selected to construct a subgraph.
- **Diffusion** enriches the structure with more global information by adding new edges and changing edge weights. The resulting adjacency matrix takes a general form as

$$\widetilde{\boldsymbol{A}} = \sum_{k=0}^{\infty} \Theta_k \boldsymbol{T}^k, \tag{S2}$$

  where $\Theta_k$ is the weighting coefficient controlling the amount of information at order $k$ with $\sum_{k=0}^{\infty} \Theta_k = 1$ and $\boldsymbol{T}$ is a generalized transition matrix computed from $\boldsymbol{A}$. We point out that the diffusion operator usually converts the original graph into a dense one, bringing heavy computation to graph convolutions. Therefore, in this paper, to ensure sparsity, we consider two sparse diffusion transformations: **Personalized PageRank (PPR)** [31] followed by hard thresholding as sparsification [32] and **Markov Diffusion Kernels (MDK)** [33, 34].

### F.1.2 Feature Augmentation

Feature augmentation modifies to the attribute matrix $\boldsymbol{X}$. In this work, we concern the following two types of feature transformation functions.

- **Feature Masking (FM)** randomly masks a fraction of dimensions with zeros in node features:

$$\widetilde{\boldsymbol{X}} = [\boldsymbol{x}_1 \circ \widetilde{\boldsymbol{m}};\ \ \boldsymbol{x}_2 \circ \widetilde{\boldsymbol{m}};\ \ \cdots;\ \ \boldsymbol{x}_N \circ \widetilde{\boldsymbol{m}}]^\top, \tag{S3}$$

  where $\circ$ is Hadamard product and $\boldsymbol{m} \in \{0,1\}^F$ is a random vector with each entry drawn from a Bernoulli distribution with a probability $(1 - p_m)$.

- Instead of masking node entries in a column-wise manner, we can also apply element-wise dropout [35] to the node feature matrix. In this **Feature Dropout (FD)** scheme, each entry has a probability $p_f$ of being randomly masked with zero.

### F.2 Contrasting Modes

For an anchor instance, contrasting modes determine the positive and negative sets at different granularities of the graph. In mainstream work, three contrasting modes are widely employed.

- **Local-local CL** targets at contrasting between node-level representations in the two views. For a node embedding $\boldsymbol{v}_i$ being the anchor, the positive sample is its congruent counterpart in another view $\boldsymbol{u}_i$; embeddings other than $\boldsymbol{u}_i$ are then naturally selected as negatives.
- **Global-local CL** enforces the compatibility between node- and graph-level embeddings. Specifically, for every global embedding $\boldsymbol{s}$ being the anchor instance, its the positive sample is all its node embedding $\boldsymbol{v}_i$ within the graph. The global-local scheme shall be considered as a proxy for local-local CL, provided that the readout function $r$ is expressive enough [25, 36]. Note that when only one graph is provided, we need an explicit corruption function (e.g., random shuffling) to construct negative samples from original node embeddings [23, 25].
- **Global-global CL** further achieves consistency between the graph embeddings of the two augmented views from the same graph. For a graph embedding $\boldsymbol{s}_1$, the positive sample is the embedding $\boldsymbol{s}_2$ of the other augmented view. In this case, other graph embeddings in the batch are considered as negative samples. This scheme can be applied to datasets with multiple graphs.

### F.3 Contrastive Objectives

Contrastive objectives are used to train the encoder to maximize the agreement between positive samples and the discrepancy between negatives. We consider the following objective functions in this work.

- **Information Noice Contrastive Estimation (InfoNCE)** [37, 38] gives a lower bound of Mutual Information (MI) up to a constant, defined as

$$\mathcal{J}_{\text{InfoNCE}}(\boldsymbol{v}_i) = -\frac{1}{P}\sum_{\boldsymbol{p}_j \in \mathcal{P}(\boldsymbol{v}_i)} \log \frac{e^{\theta(\boldsymbol{v}_i, \boldsymbol{p}_j)/\tau}}{e^{\theta(\boldsymbol{v}_i, \boldsymbol{p}_j)/\tau} + \sum_{\boldsymbol{q}_j \in \mathcal{Q}(\boldsymbol{v}_i)} e^{\theta(\boldsymbol{v}_i, \boldsymbol{q}_j)/\tau}}. \tag{S4}$$

  Here $\theta(\cdot, \cdot)$ is a critic function measuring the similarity between two embeddings. Most work implements it with an additional projection head on embeddings followed by simple cosine similarity, i.e.

$$\theta(\boldsymbol{u}, \boldsymbol{v}) = \frac{g(\boldsymbol{u})^\top g(\boldsymbol{v})}{\|g(\boldsymbol{u})\| \|g(\boldsymbol{v})\|},$$

  where $g(\cdot)$ is a multilayer perceptron.

- **Jensen-Shannon Divergence (JSD)** computes the JS-divergence between the joint distribution and the product of marginals:

$$\mathcal{J}_{\text{JSD}}(\boldsymbol{v}_i) = \frac{1}{P}\sum_{\boldsymbol{p}_j \in \mathcal{P}(\boldsymbol{v}_i)} \log d(\boldsymbol{v}_i, \boldsymbol{p}_j) + \frac{1}{Q}\sum_{\boldsymbol{q}_j \in \mathcal{Q}(\boldsymbol{v}_i)} \log(1 - d(\boldsymbol{v}_i, \boldsymbol{q}_j)), \tag{S5}$$

  where $d(\cdot, \cdot)$ is a discriminator function, usually computing the inner product of node representations with a sigmoid activation:

$$d(\boldsymbol{u}, \boldsymbol{v}) = \sigma(g(\boldsymbol{u})^\top g(\boldsymbol{v})).$$

  We kindly note that Hjelm et al. [39] employ a softplus version of the JS divergence:

$$\mathcal{J}_{\text{SP-JSD}}(\boldsymbol{v}_i) = -\frac{1}{P}\sum_{\boldsymbol{p}_j \in \mathcal{P}(\boldsymbol{v}_i)} \text{sp}(-d(\boldsymbol{v}_i, \boldsymbol{p}_j)) - \frac{1}{Q}\sum_{\boldsymbol{q}_j \in \mathcal{Q}(\boldsymbol{v}_i)} \text{sp}(d(\boldsymbol{v}_i, \boldsymbol{q}_j)), \tag{S6}$$

  where $\text{sp}(x) = \log(1 + e^x)$. We empirically find that SP-JSD performs similarly to JSD. Therefore, we stick to SP-JSD in all experiments.

- **Triplet Margin loss (TM)** directly enforces the *relative* distance between positive and negative pairs:

$$\mathcal{J}_{\text{TM}}(\boldsymbol{v}_i) = \max\Big\{\frac{1}{P}\sum_{\boldsymbol{p}_j \in \mathcal{P}(\boldsymbol{v}_i)} \|\boldsymbol{v}_i - \boldsymbol{p}_j\| - \frac{1}{Q}\sum_{\boldsymbol{q}_j \in \mathcal{Q}(\boldsymbol{v}_i)} \|\boldsymbol{v}_i - \boldsymbol{q}_j\| + \epsilon, 0\Big\}, \tag{S7}$$

  where $\epsilon$ is the margin constant. TM is widely studied in metric learning literature [40].

Conceptually, these objective functions are related to the InfoMax principle [41], which aims to maximize the MI between representations of the same node in the two views. To be specific, InfoNCE and JSD are proved to be lower bounds of MI [42–44]; TM is also known to increase MI between positive representations but their relationship between MI maximization is not theoretically guaranteed. We further note that the InfoMax interpretation of these objectives may not be consistent with its behavior in practice [45] and many recent studies provide theoretical understanding behind their success [46, 47].

In addition, we explore the following three contrastive objectives that rely on no explicit construction of negative samples:

- **Bootstrapping Latent loss (BL)** simply minimizes cosine similarity consistency between positive node embeddings without explicit negative samples [10, 48–50]:

$$\mathcal{J}_{\text{BL}}(\boldsymbol{v}_i) = -\frac{q(\boldsymbol{v}_i)^\top \boldsymbol{v}_i'}{\|q(\boldsymbol{v}_i)\|\|\boldsymbol{v}_i'\|}. \tag{S8}$$

  Here $\boldsymbol{v}_i$ is the embedding from an online encoder on $v_i$ while $\boldsymbol{v}_i'$ is the embedding obtained from an offline encoder on $P(v_i)$, and $q(\cdot)$ is a predictor attempting to predict $\boldsymbol{v}_i'$ from $\boldsymbol{v}_i$. We follow previous work [49] that symmetrize the architecture by also employing the online encoder on $P(v_i)$ and predicts an offline representation on $v_i$.

  Note that it is easy to see that directly optimizing Eq. (S8) will result in a trivial solution that $q(\boldsymbol{v}_i) = \boldsymbol{v}_i'$. To avoid model collapsing, additional requirements are imposed such as asymmetric dual encoders, updating the offline encoder with exponential moving average [48], and batch normalization [51].

- **Barlow Twins loss (BT)** proposes to encourages similar representations between augmented views of a sample, while minimizing the redundancy *within* the latent representation vector [52–54]:

$$\mathcal{J}_{\text{BT}} = \sum_i (1 - \boldsymbol{C}_{ii})^2 + \lambda \sum_i \sum_{j \neq i} \boldsymbol{C}_{ij}^2, \tag{S9}$$

  where $\boldsymbol{C}$ is the correlation matrix cross representations resulting from two augmented views and $\lambda$ is a trade-off hyperparameter controlling the on- and off-diagonal terms.

- **VICReg loss** further combines variance and covariance regularization terms with the Barlow Twins loss [55]:

$$\mathcal{J}_{\text{VICReg}} = \lambda s(\boldsymbol{V}, \boldsymbol{U}) + \mu(v(\boldsymbol{U}) + v(\boldsymbol{V})) + \gamma(c(\boldsymbol{U}) + c(\boldsymbol{V})), \tag{S10}$$

  where $s(\cdot, \cdot)$ measures mean-squared Euclidean distance between each pair of node representations, $v(\cdot)$ is a hinge loss on the standard deviation of projections along the batch dimension, and $c(\cdot)$ defines a covariance term similar to BT. $\lambda$, $\mu$, and $\gamma$ control the importance of each term. Compared to the BT loss, VICReg is shown to be insensitive to any normalization tricks and much stabler than BT.

The latter BT and VICReg losses theoretically relate to the information bottleneck principle [53, 54], which learns representations being invariant to data augmentation while retaining informative about the sample itself [56, 57].

### F.4 Negative Mining Strategies

Notwithstanding subtle differences in prior arts, existing work presumes embeddings of nodes or graphs other than the anchor instance to be dissimilar to the anchor and thus considers them as negatives. Hence it is natural to see that large batch/sampling sizes are needed for effective CL [16, 17], since more negatives usually introduce more informative training signals.

In order to enrich the learning process with informative negative samples, many visual CL methods [58–63] advocate the explicit use of negative mining in the embedding space. Some methods develop debasing terms to select truly negative samples so as to avoid contrasting same-label instances; some other propose to upweight hard negative samples (points that are difficult to distinguish from an anchor) and remove easy ones that are less informative to improve the discriminative power of the GCL model. In our benchmarking study, we consider the following four negative mining techniques:

- **Debiased Contrastive Learning (DCL)** [60] develops a debiased InfoNCE objective to correct for the sampling of same-label data points (false negatives), motivated by the observation that sampling negative examples from truly different labels improves performance. Specifically, due to unavailability of ground-truth labels, DCL decomposes the data distribution $p(\boldsymbol{x})$ to positive and negative distributions and estimates the negative $p(\boldsymbol{x}^-)$ from the positive distribution $p(\boldsymbol{x}^+)$ with an class prior $\tau^+$, which essentially reweights positive and negative terms in the denominator of InfoNCE. In our experiments, we set $\tau^+ = 0.1$ following its original implementation.
- **Hardness-Biased Negative Mining (HBNM)** [62] improves DCL by further introducing an exponential distribution $q$ with a weighting hyperparameter (to denote the hardness level) $\beta$ so that it allows the model to concentrate the distribution of negative samples around those having high similarity with the anchor. DCL could be regarded as a special case of HBNM when $\beta = 0$.
- **Hard Negative Mixing (HNM)** proposes to upweight hard negative samples by mixing up other hard samples [61, 64, 65]. Here we first select $2S$ hardest samples of the top-$K$ similar to the anchor sample, denoted as $\{\boldsymbol{r}_i\}_{i=1}^{2S}$. Thereafter, we synthesize $S$ samples for each view. Taking the first view as an example, its synthesized hard negative samples are computed by

$$\widetilde{\boldsymbol{v}}_i = \alpha_i \boldsymbol{v}_i + (1 - \alpha_i)\boldsymbol{r}_{i+s}, \tag{S11}$$

  where $\alpha_i$ is sampled from a Beta distribution $\alpha_i \sim \mathrm{Beta}(1, 1)$.
- **Conditional Negative Sampling (CNS)** [63] proposes a ring-like negative sampling strategy to choose semi-hard negatives (that are not so hard and not so easy to an anchor sample) to yield strong representations. To be specific, CNS defines a lower and upper percentile $l$ and $u$ of pairwise distances to construct a support negative example set $S_B$. Then, CNS constructs a conditional distribution for negative examples based on $S_B$ such that negative samples are not too easy nor too hard for the anchor sample.

It should be noted that the above negative mining strategies, originally designed for grid data, all measure the relative similarity of positive/negative pairs using dot-product of embeddings.

## References


[1] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vander-Plas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.

[2] Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *RLGM@ICLR*, 2019.

[3] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep Graph Contrastive Representation Learning. In *GRL+@ICML*, 2020.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, pages 8024–8035, 2019.

[5] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: a Collection of Benchmark Datasets for Learning with Graphs. In *GRL+@ICML*, 2020.

[6] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *ICLR*, 2018.

[7] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *NeurIPS*, pages 22118–22133, 2020.

[8] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs. *arXiv.org*, March 2021.

[9] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for Pre-training Graph Neural Networks. In *ICLR*, 2020.

[10] Pierre H. Richemond, Jean-Bastien Grill, Florent Altché, Corentin Tallec, Florian Strub, Andrew Brock, Samuel Smith, Soham De, Razvan Pascanu, Bilal Piot, and Michal Valko. BYOL Works Even Without Batch Statistics. *arXiv.org*, October 2020.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. In *CVPR*, pages 1–9, 2015.

[13] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Colorization as a Proxy Task for Visual Understanding. In *CVPR*, pages 840–849, 2017.

[14] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised Representation Learning by Predicting Image Rotations. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

[15] Philip Bachman, R. Devon Hjelm, and William Buchwalter. Learning Representations by Maximizing Mutual Information Across Views. In *NeurIPS*, pages 15509–15519, 2019.

[16] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In *ICML*, pages 1597–1607, 2020.

[17] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. In *CVPR*, pages 9726–9735, 2020.

[18] Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. Unsupervised Feature Learning via Non-Parametric Instance Discrimination. In *CVPR*, pages 3733–3742, 2018.

[19] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Juncheng Liu, and Bryan Hooi. NodeAug: Semi-Supervised Node Classification with Data Augmentation. In *KDD*, pages 207–217, 2020.

[20] Xiaojie Guo, Yuanqi Du, and Liang Zhao. Deep Generative Models for Spatial Networks. In *KDD*, pages 505–515, 2021.

[21] Yuanqi Du, Shiyu Wang, Xiaojie Guo, Hengning Cao, Shujie Hu, Junji Jiang, Aishwarya Varala, Abhinav Angirekula, and Liang Zhao. GraphGT: Machine Learning Datasets for Deep Graph Generation and Transformation. In *NeurIPS*, 2021.

[22] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph Contrastive Learning with Augmentations. In *NeurIPS*, pages 5812–5823, 2020.

[23] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive Multi-View Representation Learning on Graphs. In *ICML*, pages 3451–3461, 2020.

[24] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph Contrastive Learning with Adaptive Augmentation. In *WWW*, pages 2069–2080, 2021.

[25] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep Graph Infomax. In *ICLR*, 2019.

[26] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization. In *ICLR*, 2020.

[27] Yaochen Xie, Zhao Xu, Zhengyang Wang, and Shuiwang Ji. Self-Supervised Learning of Graph Neural Networks: A Unified Review. *arXiv.org*, February 2021.

[28] Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S. Yu. Graph Self-Supervised Learning: A Survey. *arXiv.org*, February 2021.

[29] Lirong Wu, Haitao Lin, Zhangyang Gao, Cheng Tan, and Stan Z. Li. Self-supervised on Graphs: Contrastive, Generative, or Predictive. *arXiv.org*, May 2021.

[30] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast Random Walk with Restart and Its Applications. In *ICDM*, pages 613–622, 2006.

[31] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab, November 1999.

[32] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion Improves Graph Learning. In *NeurIPS*, pages 13333–13345, 2019.

[33] François Fouss, Kevin Françoisse, Luh Yen, Alain Pirotte, and Marco Saerens. An Experimental Investigation of Kernels on Graphs for Collaborative Recommendation and Semisupervised Classification. *Neural Networks*, 31:53–72, 2012.

[34] Hao Zhu and Piotr Koniusz. Simple Spectral Graph Convolution. In *ICLR*, 2021.

[35] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks From Overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

[36] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *ICLR*, 2019.

[37] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding. *arXiv.org*, 2018.

[38] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised Contrastive Learning. In *NeurIPS*, pages 18661–18673, 2020.

[39] R. Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Philip Bachman, Adam Trischler, and Yoshua Bengio. Learning Deep Representations by Mutual Information Estimation and Maximization. In *ICLR*, 2019.

[40] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. In *CVPR*, pages 815–823, 2015.

[41] Ralph Linsker. Self-Organization in a Perceptual Network. *IEEE Computer*, 21(3):105–117, 1988.

[42] Michael Gutmann and Aapo Hyvärinen. Noise-Contrastive Estimation: a New Estimation Principle for Unnormalized Statistical Models. In *AISTATS*, pages 297–304, 2010.

[43] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization. In *NIPS*, pages 271–279, 2016.

[44] Ben Poole, Sherjil Ozair, Aäron van den Oord, Alexander A. Alemi, and George Tucker. On Variational Bounds of Mutual Information. In *ICML*, pages 5171–5180, 2019.

[45] Michael Tschannen, Josip Djolonga, Paul K. Rubenstein, Sylvain Gelly, and Mario Lucic. On Mutual Information Maximization for Representation Learning. In *ICLR*, 2020.

[46] Tongzhou Wang and Phillip Isola. Understanding Contrastive Representation Learning through Alignment and Uniformity on the Hypersphere. In *ICML*, pages 9929–9939, 2020.

[47] Yuandong Tian, Xinlei Chen, and Surya Ganguli. Understanding Self-Supervised Learning Dynamics without Contrastive Pairs. In *ICML*, 2021.

[48] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning. In *NeurIPS*, pages 21271–21284, 2020.

[49] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko. Bootstrapped Representation Learning on Graphs. *arXiv.org*, February 2021.

[50] Zekarias T. Kefato and Sarunas Girdzijauskas. Self-Supervised Graph Neural Networks Without Explicit Negative Sampling. In *SSL@WWW*, 2021.

[51] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, pages 448–456, 2015.

[52] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. In *ICML*, 2021.

[53] Yao-Hung Hubert Tsai, Shaojie Bai, Louis-Philippe Morency, and Ruslan R. Salakhutdinov. A Note on Connecting Barlow Twins with Negative-Sample-Free Contrastive Learning. *arXiv.org*, April 2021.

[54] Piotr Bielak, Tomasz Kajdanowicz, and Nitesh V. Chawla. Graph Barlow Twins: A Self-Supervised Representation Learning Framework for Graphs. *arXiv.org*, June 2021.

[55] Adrien Bardes, Jean Ponce, and Yann LeCun. VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning. *arXiv.org*, May 2021.

[56] Naftali Tishby, Fernando C. Pereira, and William Bialek. The Information Bottleneck Method. *arXiv.org*, April 2000.

[57] Naftali Tishby and Noga Zaslavsky. Deep Learning and the Information Bottleneck Principle. In *ITW*, pages 1–5, 2015.

[58] Jovana Mitrovic, Brian McWilliams, and Melanie Rey. Less Can Be More in Contrastive Learning. In *ICBINB@NeurIPS*, pages 70–75, 2020.

[59] Tiffany Tianhui Cai, Jonathan Frankle, David J. Schwab, and Ari S Morcos. Are All Negatives Created Equal in Contrastive Instance Discrimination? *arXiv.org*, October 2020.

[60] Ching-Yao Chuang, Joshua Robinson, Lin Yen-Chen, Antonio Torralba, and Stefanie Jegelka. Debiased Contrastive Learning. In *NeurIPS*, pages 8765–8775, 2020.

[61] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. Hard Negative Mixing for Contrastive Learning. In *NeurIPS*, pages 21798–21809, 2020.

[62] Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive Learning with Hard Negative Samples. In *ICLR*, 2021.

[63] Mike Wu, Milan Mosse, Chengxu Zhuang, Daniel Yamins, and Noah Goodman. Conditional Negative Sampling for Contrastive Learning of Visual Representations. In *ICLR*, 2021.

[64] Kibok Lee, Yian Zhu, Kihyuk Sohn, Chun-Liang Li, Jinwoo Shin, and Honglak Lee. i-Mix: A Strategy for Regularizing Contrastive Representation Learning. *arXiv.org*, October 2020.

[65] Vikas Verma, Minh-Thang Luong, Kenji Kawaguchi, Hieu Pham, and Quoc V. Le. Towards Domain-Agnostic Contrastive Learning. *arXiv.org*, November 2020.