A Class-Aware Representation Refinement Framework for Graph Classification

Anonymous authors
Paper under double-blind review

Abstract

Graph Neural Networks (GNNs) are widely used for graph representation learning. Despite its prevalence, GNN suffers from two drawbacks in the graph classification task, the neglect of graph-level relationships, and the generalization issue. Each graph is treated separately in GNN message passing/graph pooling, and existing methods to address overfitting operate on each individual graph. This makes the graph representations learnt less effective in the downstream classification. In this paper, we propose a Class-Aware Representation rEfinement (CARE) framework for the task of graph classification. CARE computes simple yet powerful class representations and injects them to steer the learning of graph representations towards better class separability. CARE is a plug-and-play framework that is highly flexible and able to incorporate arbitrary GNN backbones without significantly increasing the computational cost. We also theoretically prove that CARE has a better generalization upper bound than its GNN backbone through Vapnik-Chervonenkis (VC) dimension analysis. Our extensive experiments with 10 well-known GNN backbones on 9 benchmark datasets validate the superiority and effectiveness of CARE over its GNN counterparts.

1 Introduction

In recent years, Graph Neural Networks (GNNs) have attracted a surge of attention and been extensively used to learn graph representations for downstream tasks such as graph classification (Xu et al., 2018; Ying et al., 2018), link prediction (Zhang & Chen, 2018), graph clustering (Kipf & Welling, 2016b; Zhang et al., 2019a), etc. They have been applied to graph structured data from a wide range of application domains, including social networks (Kipf & Welling, 2016a; Hamilton et al., 2017; Fan et al., 2019), molecular data (Dai et al., 2016; Gilmer et al., 2017), recommendation system (Wu et al., 2020a; Xia et al., 2021), brain networks (Kim & Ye, 2020), and many more.

The GNN-based graph representation learning can be divided into two categories: (1) some of the methods (Duvenaud et al., 2015; Lee et al., 2019) use a down-sampling strategy (Wu et al., 2020b) to aggregate and transform node representations to graph representations; (2) while some other methods (Ying et al., 2018; Zhang et al., 2019b; Nouranizadeh et al., 2021) obtain the graph embedding by exploring the hierarchical structure of each input graph. These approaches suffer from two major drawbacks when applied to the downstream classification task: (1) Neglect of graph-level relationships; and (2) Generalization issue.

Neglect of graph-level relationships. Existing GNN architectures consider each input graph independently in their training processes. Input graphs are passed individually to GNN to yield node representations. In addition, the model also treats each graph separately in its design of loss. The relationships (similarity and/or discrepancy) among different input graphs are fully neglected. Though the model parameters are trained by the set of input graphs collectively, it is done through a long pathway from node representations to graph representations and finally to the loss. As a result, the effectiveness of the model will be significantly compromised when applied to the downstream classification. With molecular data, for instance, one would want the molecules from the same class to share similar representations. This is natural as molecules belonging to the same class often carry certain common substructures (e.g., the same set of functional groups). These substructures could be class-specific and serve as a perfect signature of a class. Without considering such graph-level

information, the graph representations learnt would be less effective in separating different graph classes.

Generalization issue. This is an inherent issue in GNN that the model tends to overfit when the network gets deeper or the hidden dimensionality gets larger (Song et al., 2021). Some methods have been proposed to alleviate this issue. Several graph-argumentation based methods improve the generalization ability by modifying input graphs (Papp et al., 2021), or generating new graphs for adversarial learning (Ding et al., 2018) and contrastive learning (You et al., 2020). Some other works (Byrd & Lipton, 2019; Lin et al., 2017) resample or reweight data instances to remit the overfitting problem. However, these methods operate on each individual graph and fail to explore the effectiveness of graph-level information in improving generalization. The discussion of related works is provided in Appendix A.

In this paper, we develop a Class-Aware Representation rEfinement (CARE) framework to address the two above-mentioned limitations of GNNs on the task of graph classification. Built upon the two existing steps of GNN models for node representation learning and graph pooling, we introduce a new block for extracting class-specific information, namely the Class-Aware Refiner. The idea of this refiner is simple but effective. Under the supervision of ground-truth labels, the refiner learns the class representation from a bag of subgraph representations (generated by graph pooling) from each class, and uses it to refine the representation of each graph within the same class. In this way, we inject the class-specific information to graph representations, with the hope that it can steer the graph representation learning to reflect class signatures. Inspired by the separability in clustering (Wen et al., 2016), we also propose a class loss that takes into account intra-class graph similarity and inter-class graph discrepancy. This class loss is combined with the classification loss to directly influence the training of class representations to gain better class separability. We further design two different architectures of CARE so that our framework can incorporate arbitrary GNN backbones. In terms of the model generalization, we analyze the Vapnik-Chervonenkis (VC) dimension (Vapnik & Chervonenkis, 2015) of CARE and provide theoretic guarantee that the upper bound of the VC dimension of CARE is lower than that of the GNN backbone. The better ability of CARE in alleviating overfitting is also evidenced in our experiments.

We empirically validate the graph classification performance of CARE on 9 benchmark datasets with 10 commonly used GNN backbones. The results demonstrate that CARE significantly outperforms its GNN backbones: it achieves up to 11% improvement in classification accuracy. We also perform a series of ablation studies to assess the effect of each component. We use a case study to showcase that the graph representations refined by CARE is able to achieve better class separability. Though CARE introduces an additional refiner to the backbone model, our results show that the consideration of graph-level information drives the model to converge with fewer training epochs. Consequently, the training time of CARE is comparable to or even shorter than its GNN counterparts.

The main contributions of this paper are summarized as follows:

- We propose a novel graph representation refinement framework CARE, which considers class-aware graph-level relationships. CARE is a flexible plug-and-play framework that can incorporate arbitrary GNNs without significantly increasing the computational cost.
- We provide theoretic support through VC dimension analysis that CARE has better generalization upper bound in comparison with its GNN backbone.
- We perform extensive experiments using 10 GNN backbones on 9 benchmark datasets to justify the superiority of CARE on graph classification task in terms of both effectiveness and efficiency.

2 Our Proposed Method

In this section, we first give the problem formulation of graph classification, followed by the introduction of our proposed CARE framework. We also discuss two architectures for applying CARE to existing GNNs. Finally, we provide theoretic support for the generalization of CARE.

2.1 Problem Formulation

We represent a graph as $G = (\boldsymbol{A}, \boldsymbol{X})$, where $\boldsymbol{A} \in \{0,1\}^{n \times n}$ is its adjacency matrix, and $\boldsymbol{X} \in \mathbb{R}^{n \times c}$ denotes the feature matrix with each node characterized by a feature vector of c dimensions. The node set of G is denoted by $\mathcal{V}_G$ and $|\mathcal{V}_G| = n$. We use $\boldsymbol{X}_v$ to denote the feature vector of a node $v \in \mathcal{V}_G$. Table 6 in Appendix summarizes the notations used throughout the paper.

Given a data set of labeled graphs $\mathcal{D} = (\mathcal{G}, \mathcal{Y}) = \{(G, y_G)\}$, where $y_G \in \mathcal{Y}$ is the corresponding label of graph $G \in \mathcal{G}$, the problem of graph classification aims to learn a predictive function f: $\mathcal{G} \rightarrow \mathcal{Y}$ that maps graphs to their labels.

2.2 Class-Aware Representation Refinement Framework

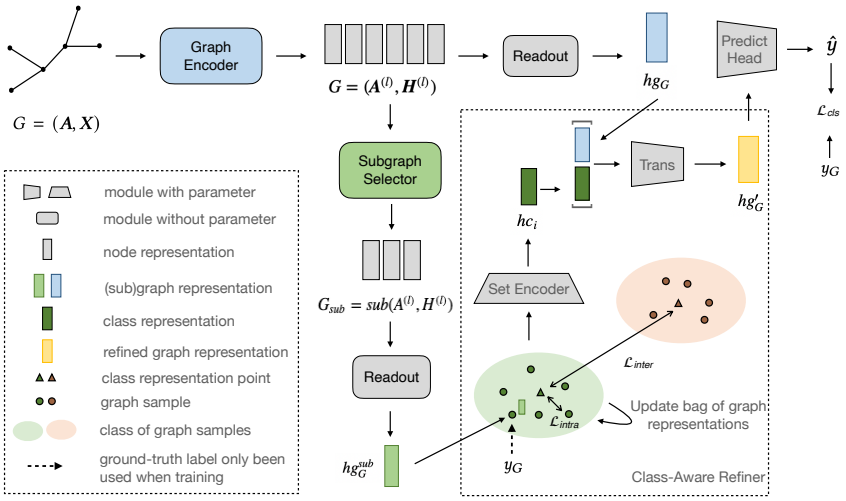

Figure 1: Framework of CARE.

We now describe our proposed Class-Aware Representation rEfinement framework (CARE), which aims to refine graph representations by considering the graph-level similarity. CARE contains four main components, including a graph encoder, a subgraph selector, a class-aware refiner and a class loss. The former two allow the flexible incorporation of existing GNN methods, while the latter two are newly proposed in our framework. Figure 1 depicts the CARE framework.

The remainder of this section describes the four components in detail. We first introduce the graph encoder to get the initial node/graph representation, and then describe the subgraph selector to extract an appropriate substructure for the subsequent class representation learning. The class-aware refiner learns class representations from different graph classes, which are used to refine graph representations. A class loss is proposed to further improve class separability. The two new components in CARE only contain a small number of parameters and are easy to plugin arbitrary GNN backbone.

Graph Encoder A graph encoder extracts the node representations $\boldsymbol{H}$ and the graph-level representation hg_G for graph G. CARE does not impose any constraint on the architecture of the graph encoder. Any message-passing GNN model could be applied here, which is formalized as $\boldsymbol{H}_v^{(l+1)} = \mathrm{UP}^{(l)}\left(\boldsymbol{H}_v^{(l)}, \mathrm{AGG}^{(l)}\left(\left\{\boldsymbol{H}_u^{(l)}\right\}_{\forall u \in \mathcal{N}(v)}\right)\right)$, where $\boldsymbol{H}^{(l+1)} \in \mathbb{R}^{n \times m}$ denotes the $(l+1)$-th layer node representation with m dimensions, UP and AGG are arbitrary differentiable update and aggregate functions, $\mathcal{N}(v)$ represents the neighbor node set of node $v \in \mathcal{V}_G$, and $\boldsymbol{H}_v^{(0)}$ is initialized as the input feature vector $\boldsymbol{X}_v$.

After a few message-passing layers, we can obtain a set of node representations. A READOUT function can be applied to produce the graph representation $\boldsymbol{hg}_G \in \mathbb{R}^m$ as $\boldsymbol{hg}_G = \mathrm{READOUT}\left(\{\boldsymbol{H}_v \mid v \in \mathcal{V}_G\}\right)$.

Subgraph Selector The class-aware refiner in CARE aims to maintain generic features for graph samples from different classes. However, the READOUT function treats all nodes equally without considering the class information. In fact, graphs in different classes are likely to have various substructures. To address this limitation, CARE introduces a subgraph selector $sub(\cdot)$ to filter nodes in the original graph, which is defined as $A^{(l+1)}, H^{(l+1)} = sub(A^{(l)}, H^{(l)})$.

Any graph pooling methods could be applied here to select subgraphs. Typical ones include node drop pooling methods (Lee et al., 2019) and node clustering pooling methods (Baek et al., 2021).

Class-Aware Refiner As existing GNN models ignore the relationships of graphs from different classes, a new component is designed in CARE to fill this gap. In the training process, the class-aware refiner utilizes the ground truth label of each training instance. It maintains a bag of encoded (sub)graph representations $\mathcal{B}_i$ and aggregates these representations to obtain a class representation hc_i for each class $i \in \mathcal{Y}$. The aggregation function is a universal Set Encoder, e.g., DeepSets (Zaheer et al., 2017) or PointNet (Qi et al., 2016). Herein, we apply DeepSets as in Eq. (1), in which $\rho(\cdot)$ is a multilayer perceptron (MLP) with a non-linear function ReLU, and $\phi(hg) = hg/|\mathcal{B}_i|$.

$$hc_i = \rho\left(\sum_{hg_G^{sub} \in \mathcal{B}_i} \phi(hg_G^{sub})\right), \tag{1}$$

where hg_G^{sub} is the subgraph representation of a graph $G \in \mathcal{G}$, obtained by passing the output of the subgraph selector through the READOUT function. The class representation hc_i is then used to refine graph representations for all graphs in the same class:

$$hg'_G = Trans([hg_G \mid hc_i]), \tag{2}$$

where $Trans(\cdot)$ is a transformation function and hg'_G is the refined graph representation. Herein, we set $Trans(\cdot) = \rho(\cdot)$.

In the validation and the test processes, the ground truth label y_G is not available for a validation/test graph G. However, we need a label for each validation/test graph to determine which class representation to be used to composite its refined representation. In this case, the Class-Aware Refiner will predict a pseudo label $\widetilde{y}_G$ for graph G by classifying it to the most similar class and use the corresponding class representation for graph representation refinement. We use the cosine similarity as a metric to quantify the similarity between a graph representation and a class representation. The pseudo label is obtained as $\widetilde{y}_G = \arg\max_{i \in \mathcal{Y}}(cos_sim(hg_G^{sub}, hc_i))$. Note that class representations are kept unchanged in the validation/test process. The training algorithm of the class-aware refiner is summarized in Algorithm 1.

Algorithm 1 Training of Class-Aware Refiner

Input: Subraph representation hg_G^{sub} of graph G (output by Subgraph Selector), graph representation hg_G, and ground-truth label y_G of G;
Output: Refined graph representation hg'_G, intra-class loss $\mathcal{L}_{intra}$ and inter-class loss $\mathcal{L}_{inter}$;

1: Use hg_G^{sub} to update the bag of (sub)graph representations $\mathcal{B}_i$ for $i = y_G$;
2: Calculate the class representation hc_i by Eq. (1);
3: Use hc_i and hg_G to obtain the refined graph representation hg'_G by Eq. (2);
4: Calculate the intra-class loss $\mathcal{L}_{intra}$ and inter-class loss $\mathcal{L}_{inter}$ by Eqs. (3) and (4);

Class Loss For graph classification task, it would be beneficial to exploit the graph similarity within the same class and the graph discrepancy between different classes. This is essential for making different classes more separable. Using the classification loss only fails to learn such graph-level relations. Therefore, a class loss $\mathcal{L}_{class}(\cdot)$ is proposed in CARE to enforce the intra-class similarity and the inter-class discrepancy. The former $\mathcal{L}_{intra}$ is defined as the similarity between each graph representation and its class representation, while the latter $\mathcal{L}_{inter}$ is defined as the similarity between different class representations:

$$\mathcal{L}_{intra} = \underset{i \in \mathcal{Y}}{AVG}(\underset{y_G = i}{AVG}(cos_sim(hc_i, hg_G^{sub}))), \tag{3}$$

$$\mathcal{L}_{inter} = \underset{i \in \mathcal{Y}}{AVG}(\underset{j > i, j \in \mathcal{Y}}{AVG}(cos_sim(hc_i, hc_j))), \tag{4}$$

We again use the cosine similarity as a metric. The class loss $\mathcal{L}_{class} = \exp(\mathcal{L}_{inter} - \mathcal{L}_{intra})$ is then defined as a function that maximizes $\mathcal{L}_{intra}$ and minimizes $\mathcal{L}_{inter}$, where $AVG(\cdot)$ is the average function.

The predicted class label is still supervised by a classification loss $\mathcal{L}_{cls}(\cdot)$. Herein, we apply the commonly used cross-entropy loss (Cox, 1958). The overall loss $\mathcal{L}$ of CARE is defined as:

$$\mathcal{L} = \mathcal{L}_{cls} + \lambda * \mathcal{L}_{class}, \tag{5}$$

where λ is a trade-off hyperparameter for balancing the classification loss and the class loss.

2.3 Model Architecture Variants

As GNNs can be categorized as hierarchical ones and non-hierarchical ones, we design two corresponding architectures that apply CARE to plug-and-play in different GNN backbones.

Global Architecture Several GNN models (e.g., GCN (Kipf & Welling, 2016a), GAT (Veličković et al., 2017) and GraphSAGE (Hamilton et al., 2017)) apply the readout function only at the end of graph convolution. The global architecture of CARE is designed to apply the Class-Aware Refiner only after the readout function. The outputs are then passed to a linear layer for graph classification.

Hierarchical Architecture Some other GNN models, such as GIN (Xu et al., 2018) and UGformer (Nguyen et al., 2019), have a readout function in each graph convolutional layer. The graph representations from each layer are taken into account when making the final prediction. The hierarchical architecture of CARE is designed to apply the class-aware refiner on each layer in order to cope with the hierarchical GNN backbones.

2.4 Generalization Analysis

In this section, we present the theoretical support that the proposed CARE has a better model generalization than its GNN backbone in the case of binary classification. We use the VC dimension to measure the capacity of a model. Based on the VC theory (Vapnik, 2000), reducing the VC dimension of a model has the effect of eliminating potential generalization errors.

Our analysis is grounded on the VC theory for neural nets (Bartlett & Maass, 2003): the VC dimension of a neural network is upper bounded by a function with respect to the number of model parameters t and the number of operations p. In the following, we first derive the computational complexity of the GNN backbone and CARE measured by the number of multiplications, based on which we obtain an upper bound of the VC dimension for each model. We then present a theorem that states that CARE has a lower VC dimension upper bound than its GNN backbone when the number of parameters is identical. In subsequent discussions, we use GCN as an example backbone. The conclusion generally applies to other backbones by plugging in their corresponding computational complexity. We present the theoretical results here and defer the detailed proofs to Appendix.

Computational Complexity of Models. CARE and its backbone GCN are both composed of GCN layers, an embedding layer, and several fully-connected layers.

[*Complexity of GCN Backbone*.] We denote the GCN layer in the GCN model as $gcn(\cdot)$ and its input/output dimensions as $h_{gcn_{in}}/h_{gcn_{out}}$. The layer mapping is given by $gcn(A, H) = \sigma_{gcn}(\hat{A}HW_{gcn})$, where σ_{gcn} is the activation function, $W_{gcn} \in \mathbb{R}^{h_{gcn_{in}} \times h_{gcn_{out}}}$ is the weight matrix, and $\hat{A}$ is the normalized adjacency matrix. The computational complexity of the GCN network measured by the multiplication number, denoted as $q_1(d)$, for d number of layers, is given by:

$$q_1(d) = \sum_{l=0}^{d} (n^2 h^l_{gcn_{in}} + n h^l_{gcn_{in}} h^l_{gcn_{out}}). \tag{6}$$

[*Complexity of CARE.*] The GCN-based CARE network with a hierarchical architecture is composed of the following:

- a GCN layer same as the GCN backbone, whose computational complexity is $q^l_{gcn} = n^2 h^l_{gcn_{in}} + n h^l_{gcn_{in}} h^l_{gcn_{out}}$.
- a subgraph selector (SAGPool), which contains a score layer (GCN with 1 dimensional output) and a top-k pooling algorithm. The complexity is $q^l_{subgraph} = n^2 h^l_{gcn_{out}} + n h^l_{gcn_{out}}$.
- a class-aware refiner contains a set encoder Eq. (1) and a transformation layer Eq. (2). The mapping of the fully-connection layer $fc(\cdot)$ from input H is given by $fc(H) = \sigma_{fc}(HW_{fc})$, where σ_{fc} is the activation function, $W_{fc} \in \mathbb{R}^{h_{fc_{in}} \times h_{fc_{out}}}$, and $h_{fc_{in}}/h_{fc_{out}}$ are the input/output dimensions. The complexities for the set encoder and the transformation layer are $q^l_{set} = n h^l_{set_{in}} h^l_{set_{out}}$ and $q^l_{trans} = n h^l_{trans_{in}} h^l_{trans_{out}}$, respectively.

Therefore, the computational complexity of the GCN-based CARE is given by:

$$q_2(d) = \sum_{l=0}^{d} (q^l_{gcn} + q^l_{subgraph} + q^l_{set} + q^l_{trans}). \tag{7}$$

VC Dimension Upper Bound. Inspired by the theoretical analysis in Kabkab et al. (2016) that derives an upper bound of the VC dimension for a CNN model, we extend its result to a GCN model, as given by the following lemma.

Lemma 1 Let $\mathcal{C}^d$ be the set of GCN models with d convolutional layers. Let $\mathcal{H}^d \triangleq \{h_c : I \to \{0,1\} | c \in \mathcal{C}^d\}$ be the set of boolean functions implementable by all GCNs in $\mathcal{C}^d$. The VC dimension of GCNs, as well as CARE, satisfies $VC_{dim}(\mathcal{H}^d) \leqslant \alpha(d \cdot q(d))^2$ for some constant α. Here, $q(d)$ is the computational complexity of the model under consideration.

VC Dimension Comparison We now compare the upper bounds of VC dimension on the GCN backbone and CARE, which is formalized by the following theorem.

Theorem 1. Assume that the number of parameters in a GCN backbone and CARE is identical. Let *upperVC(GCN)* and *upperVC(CARE)* be the upper bounds of VC dimension on the two models, respectively, which are given by Lemma 1. We have $upperVC(GCN) > upperVC(CARE)$.

Based on Theorem 1 and the VC theory, we conclude that our CARE has a better generalization potential than its GCN backbone.

3 EXPERIMENTS

Due to the space limitation, we provide the experiment setup in Appendix C. In this section, we first assess the performance of CARE in comparison with state-of-the-art GNN backbones. We then conduct ablation studies to analyze the effects of different components in CARE. A case study is presented to further investigate how CARE affects the separability of graphs from different classes. Finally, an evaluation of model efficiency is performed. The sensitivity test of hyperparameters in CARE is discussed in Appendix D.

3.1 PERFORMANCE COMPARISON WITH GNN BACKBONES

Effectiveness Analysis. We assess the graph classification performance on the first 8 datasets and the last dataset using two different metrics. The former is assessed by the classification accuracy and the latter by the ROC-AUC. This is because the OGBG-MOLHIV dataset has a severe class imbalance issue. The results on the first 8 datasets are reported in Table 1. Each row in the table

shows the performance of an original GNN backbone and the performance after applying CARE. Each column reports the results on a dataset. In total there are 80 backbone/dataset pairs and the best result in each pair is highlighted in bold. As shown in Table 1, CARE is a clear winner: it outperforms the GNN backbone in 74 out of 80 cases. CARE gains over 1% improvement in the absolute accuracy in 47 out of 74 winning cases, while it drops over 1% in accuracy in only 1 out of 6 losing cases. In particular, the improvement of CARE is up to 11.48%, which is achieved on the FRANKENSTEIN dataset with GraphSAGE as backbone. The same observation is made when testing on the OGBG-MOLHIV dataset. As shown in Table 2, CARE outperforms the GNN backbones in most cases with improvements up to 5.63%. To sum up, the results demonstrate that CARE is able to serve as a general framework to boost up the graph classification performance over state-of-the-art GNN models on various datasets. To match with the setting of Theorem 1, we also conduct experiments under the same parameter numbers in Appendix E. The experimental comparison with GIB (Yu et al., 2020) is provided in Appendix F.

Model		D&D	PROTEINS	MUTAG	NCI1	NCI109	FRANK	Tox21	ENZYMES
GraphSAGE	Original	72.18 ± 2.93	74.87 ± 3.38	75.48 ± 6.11	63.94 ± 2.53	65.46 ± 1.12	52.95 ± 4.01	88.36 ± 0.15	52.50 ± 5.69
	CARE	**73.26** ± 3.25	**75.92** ± 2.84	**81.97** ± 6.42	**75.23** ± 1.76	**73.58** ± 1.68	**64.43** ± 3.15	**90.14** ± 0.74	**53.67** ± 5.67
GCN	Original	71.02 ± 3.17	73.89 ± 2.85	77.52 ± 10.81	78.80 ± 2.01	75.06 ± 2.50	55.58 ± 0.11	88.14 ± 0.29	62.17 ± 6.33
	CARE	**72.15** ± 3.88	**75.01** ± 2.91	**79.30** ± 11.81	**79.66** ± 1.71	**75.75** ± 1.63	**59.15** ± 2.25	**90.31** ± 0.56	**65.00** ± 5.63
GIN	Original	73.10 ± 2.44	**72.41** ± 4.45	89.36 ± 4.71	81.96 ± 2.03	81.01 ± 1.84	67.23 ± 1.93	92.10 ± 0.59	62.79 ± 7.64
	CARE	**74.70** ± 3.37	72.32 ± 4.25	**90.44** ± 4.58	**82.34** ± 2.11	**82.15** ± 1.79	**67.33** ± 2.74	**92.43** ± 0.78	**68.17** ± 7.05
GAT	Original	74.25 ± 3.76	74.34 ± 2.09	77.56 ± 10.49	78.07 ± 1.94	74.34 ± 2.18	**62.85** ± 1.90	90.35 ± 0.71	67.67 ± 3.74
	CARE	**75.38** ± 2.93	**76.72** ± 1.74	**77.69** ± 8.99	**78.52** ± 2.12	**76.39** ± 2.76	62.57 ± 2.37	**90.76** ± 0.73	**69.17** ± 5.02
GXN	Original	67.62 ± 5.85	70.32 ± 3.03	83.22 ± 7.97	73.34 ± 2.54	72.18 ± 2.24	60.86 ± 2.17	89.93 ± 0.73	63.13 ± 4.68
	CARE	**69.24** ± 6.97	**72.70** ± 2.73	**84.24** ± 8.53	**74.75** ± 2.90	**73.78** ± 1.66	**62.64** ± 2.27	**90.16** ± 0.85	**64.44** ± 6.96
UGformer	Original	75.51 ± 3.92	70.17 ± 5.42	75.66 ± 8.67	**68.84** ± 1.54	66.37 ± 2.74	56.13 ± 2.51	88.06 ± 0.50	64.57 ± 4.53
	CARE	**76.23** ± 4.45	**71.84** ± 3.87	**77.66** ± 5.93	67.03 ± 1.69	**66.92** ± 1.58	**57.10** ± 2.27	**88.21** ± 0.24	**65.24** ± 5.91
SAGPool	Original	71.46 ± 3.60	74.12 ± 3.46	78.12 ± 8.35	78.34 ± 1.96	76.15 ± 2.25	59.07 ± 2.23	**90.78** ± 0.63	62.00 ± 4.76
	CARE	**73.28** ± 2.25	**74.75** ± 3.14	**79.81** ± 7.52	**78.91** ± 2.21	**76.44** ± 1.74	**59.67** ± 2.04	90.64 ± 0.38	**63.17** ± 4.37
DiffPool	Original	70.45 ± 2.54	72.18 ± 2.80	85.26 ± 4.79	79.78 ± 2.11	76.98 ± 1.88	**65.01** ± 3.17	91.02 ± 0.37	48.33 ± 6.67
	CARE	**72.90** ± 4.58	**72.57** ± 2.97	**86.33** ± 8.14	**80.47** ± 1.67	**78.49** ± 1.72	64.36 ± 3.43	**91.58** ± 0.65	**51.17** ± 6.75
HGPSLPool	Original	71.25 ± 3.25	73.06 ± 3.20	80.82 ± 6.63	79.26 ± 1.44	75.83 ± 1.98	60.82 ± 2.85	90.12 ± 0.47	63.33 ± 5.06
	CARE	**71.61** ± 3.36	**75.47** ± 3.98	**82.31** ± 6.91	**79.77** ± 1.97	**76.87** ± 1.94	**63.36** ± 1.73	**90.44** ± 0.69	**66.00** ± 4.48
MEWISPool	Original	**76.03** ± 2.59	68.10 ± 3.97	84.73 ± 4.73	74.21 ± 3.26	75.30 ± 1.45	64.63 ± 2.83	88.13 ± 0.05	53.66 ± 6.07
	CARE	75.18 ± 3.90	**69.64** ± 3.69	**85.39** ± 5.85	**76.48** ± 2.74	**75.34** ± 2.86	**65.39** ± 1.67	**88.65** ± 0.07	**55.67** ± 6.33

Table 1: Graph Classification Results (Average Accuracy ± Standard Deviation). Winner in each backbone/dataset pair is highlighted in bold.

	GraphSAGE	GCN	GIN	GAT	GXN
Original	70.37 ± 0.42	73.49 ± 1.99	65.11 ± 2.56	75.83 ± 1.78	69.15 ± 0.01
CARE	**74.33** ± 2.12	**74.29** ± 1.07	**65.42** ± 3.70	**76.89** ± 2.18	**69.17** ± 0.02
	UGformer	SAGPool	DiffPool	HGPSLPool	MEWISPool
Original	77.23 ± 3.54	73.80 ± 1.86	**71.63** ± 2.25	76.08 ± 2.86	**79.66** ± 1.71
CARE	**78.04** ± 3.19	**74.42** ± 1.53	70.21 ± 2.79	**77.23** ± 2.16	77.37 ± 1.05

Table 2: Graph Classification Results (Average ROC-AUC ± Standard Deviation) on OGBG-MOLHIV dataset. Winner in each backbone/dataset pair is highlighted in bold.

Generalization Performance. We also observe that CARE is able to alleviate the overfitting in GNN backbones. An example is shown in Figure 2, where we plot the accuracy curves on the PROTEINS dataset with GCN as the backbone. It shows that the test accuracy of GCN (in blue) exhibits a steep and continuous downward trend starting from epoch 120, while its corresponding training accuracy continues to climb up. This indicates an obvious overfit of GCN to the training data. After applying CARE on GCN (in red), the steep drop in the test accuracy vanishes, which demonstrates the ability of CARE in remitting overfitting.

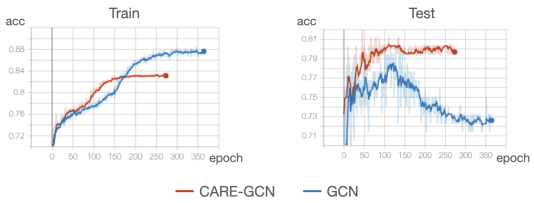

Figure 2: Accuracy curves of CARE-GCN and GCN on PROTEINS dataset.

Backbone	Subgraph Selector	D&D	PROTEINS	MUTAG	NCI1	NCI109
GraphSAGE	None	67.24 ± 4.64	75.01 ± 4.15	**82.95** ± 5.86	**77.66** ± 1.98	73.67 ± 1.28
	SAGPool	**73.26** ± 3.25	**75.92** ± 2.84	81.97 ± 6.42	75.23 ± 1.76	73.58 ± 1.68
	HGPSLPool	71.82 ± 3.99	75.28 ± 3.76	77.57 ± 7.33	75.84 ± 1.50	**74.36** ± 2.17
GCN	None	71.05 ± 3.89	73.39 ± 3.45	78.74 ± 9.67	79.15 ± 1.66	**76.30** ± 2.31
	SAGPool	**72.15** ± 3.88	**75.01** ± 2.91	**79.30** ± 11.81	**79.66** ± 1.71	75.75 ± 1.63
	HGPSLPool	68.75 ± 3.45	73.39 ± 3.45	77.66 ± 5.13	79.25 ± 1.90	75.35 ± 2.57
GIN	None	**75.64** ± 3.38	71.96 ± 5.49	88.80 ± 5.05	**82.85** ± 1.27	82.11 ± 1.60
	SAGPool	74.70 ± 3.37	72.32 ± 4.25	90.44 ± 4.58	82.34 ± 2.11	**82.15** ± 1.79
	HGPSLPool	75.06 ± 3.49	**72.86** ± 4.66	**90.47** ± 6.10	81.63 ± 1.80	81.05 ± 1.10
GAT	None	74.28 ± 2.43	75.74 ± 2.88	78.22 ± 6.77	75.30 ± 3.01	74.90 ± 2.24
	SAGPool	**75.38** ± 2.93	**76.72** ± 1.74	77.69 ± 8.99	**78.52** ± 2.12	**76.39** ± 2.76
	HGPSLPool	74.79 ± 2.73	75.46 ± 3.56	**79.33** ± 9.73	75.74 ± 2.32	74.15 ± 3.53

Table 3: Ablation Study on Different Subgraph Selectors. Winner is highlighted in bold.

3.2 Ablation Studies

We perform two ablation studies to show how different designs of the Subgraph Selector and the loss function $\mathcal{L}$ influence the performance of CARE.

Subgraph Selector We compare the performance of three CARE variants with different subgraph selectors on 5 datasets with 4 GNN backbones. The results are presented in Table 3. The first CARE variant, denoted as "None", uses the whole graph for class representation learning without selecting any subgraph. The other two models apply SAGPool and HGSPLPool respectively as the subgraph selector. The pooling ratio for both SAGPool and HGSPLPool is set to 0.5. We can see that using the subgraph selector achieves the best result in 15 out of 20 cases. When comparing the performance of using SAGPool and HGSPLPool, the former beats the latter in 14 out of 20 cases. Therefore, we choose SAGPool as the default subgraph selector.

Class Loss We investigate different designs of the loss function to study the impact of the class loss we proposed. We consider four designs of the overall loss function $\mathcal{L}$: a) cls: uses the classification loss $\mathcal{L}_{cls}$ only; b) $intra$: uses a combination of the classification loss and the intra-class loss as given by $\mathcal{L} = \mathcal{L}_{cls} - \lambda * \exp(\mathcal{L}_{intra})$; c) $inter$: uses a combination of the classification loss and the inter-class loss as given by $\mathcal{L} = \mathcal{L}_{cls} + \lambda * \exp(\mathcal{L}_{inter})$; and d) $combine$, the overall loss function in Eq. (5). The results in Table 4 show that the proposed loss function performs the best among all designs. This demonstrates the effectiveness of the proposed class loss that takes into account the intra-class similarity and inter-class discrepancy.

Backbone	Loss	D&D	PROTEINS	MUTAG
Graph SAGE	*cls*	72.24 ± 2.72	75.82 ± 3.34	75.50 ± 6.05
	intra	70.12 ± 3.27	75.83 ± 2.83	77.02 ± 10.45
	inter	70.30 ± 3.61	75.02 ± 3.55	81.87 ± 7.67
	combine	**73.26** ± 3.25	**75.92** ± 2.84	**81.97** ± 6.42
GCN	*cls*	71.39 ± 2.81	74.21 ±2.74	77.11 ± 9.48
	intra	71.89 ± 5.35	74.65 ± 4.10	78.22 ± 7.17
	inter	71.31 ± 2.06	74.20 ± 3.88	78.18 ± 10.41
	combine	**72.15** ± 3.88	**75.01** ± 2.91	**79.30** ± 11.81

Table 4: Ablation Study on Design of Loss Function in terms of Classification Accuracy. Winner in each backbone/dataset pair is highlighted in bold.

3.3 Case Study for Class Separability

We design a case study to further investigate the effect of CARE, in particular its class-aware components, in refining graph representations for the classification task. The idea is to study how CARE affects the separability of graph classes in the training process. Is it able to direct the graph representation learning to move towards better class separability? In order to answer this question, we use four class separability metrics as follows. For all metrics, the larger their values, the better the class separability is. We refer the reader to Appendix H for their formal definitions.

Silhouette Coefficient (Rousseeuw, 1987) It measures how similar a sample is to its own class (cohesion) compared to those from other classes (separation). Its value ranges from -1 to 1. **Separability Index** (Thornton, 1998) It computes the fraction of samples that have a nearest neighbour with the same class label. Its value ranges from 0 to 1. **Hypothesis Margin** (Gilad-Bachrach et al., 2004) It measures the distance between a sample's nearest neighbor from the same class (near-hit) and the nearest neighbor from the opposing class (near-miss) and averages over all samples. **Centroid**

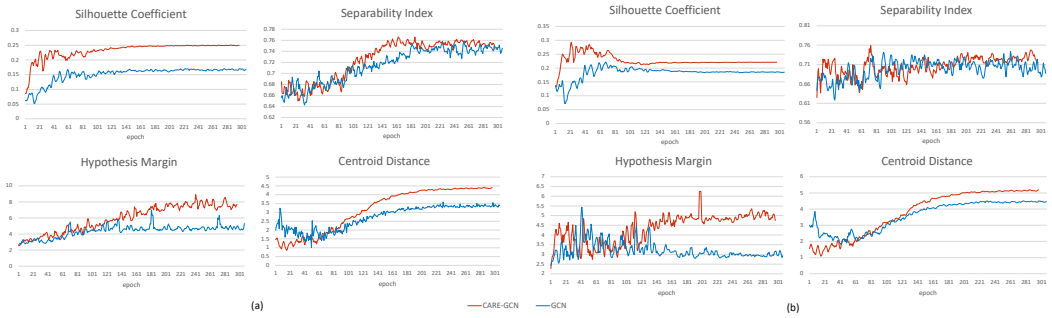

Figure 3: (a) Class Separability on PROTEINS with GCN Backbone (Training Set). (b) Class Separability on PROTEINS with GCN Backbone (Test Set). The results were obtained by passing the test data once at the end of each training epoch. Note that this process doesn't affect the training in any way as the model parameters/loss are not updated when passing the test data.

Distance It sums up the distances between the centroids for all pairs of classes, where the centroid of a class is the mean of all samples in the class.

Figure 3 reports the results on the PROTEINS dataset with GCN as the backbone. We compute the four metrics on the graph representations produced by CARE and GCN on the training data. CARE uses the refined graph representations, while GCN uses the original ones. It can be seen that the training curves of CARE exhibit an upward trend under all the four separability metrics. At the time when models converge, CARE outperforms GCN in all metrics. In particular, CARE achieves 49.26% improvement on Silhouette Coefficient, 1.96% on Separability Index, 45.21% on Hypothesis Margin, and 30.51% on centroid distance. A visualization of the graph representations in each model is shown in Figure 4. Graph representations are passed into T-SNE (Van der Maaten & Hinton, 2008) for dimensionality reduction and colored by their class labels. This demonstrates that CARE is indeed able to steer the graph representation learning towards better class separability, which is also reflected by its superior classification performance over GNN backbones. Similar conclusions can be drawn from the results on the test data, which indicates that the class separability of CARE generalizes well to the test data.

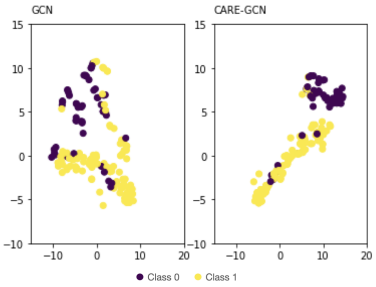

Figure 4: Visualization of Graph Representations Produced by GCN and CARE-GCN on PROTEINS dataset.

3.4 Time Efficiency

CARE, when applied to a GNN backbone, introduces an additional refiner and the class loss. A natural question arises: will CARE significantly sacrifice the efficiency of its GNN backbone for better classification performance? This subsection aims to answer this question. Table 5 in Appendix reports the number of epochs and the total time needed (including training, validation and test) for CARE and the backbones GraphSAGE and GCN. It can be seen that CARE takes less number of epochs to converge than its GNN counterpart in all cases. Consequently, the running time of CARE is shorter than (4 out of 6 cases) or comparable to its backbones. The results demonstrate that CARE is able to work on top of existing GNN models with superior effectiveness and improved/comparable efficiency, making it a practical choice in real applications.

4 Conclusions

In this paper, we proposed CARE, a novel graph representation refinement framework for GNN-based graph classification. It fills the gap that existing GNNs fail to consider graph-level relationships in model design, and meanwhile improves the model generalization as proven theoretically

and evidenced empirically. Its plug-in-play nature makes it a powerful framework to build upon arbitrary GNN models and boost up their classification performance without sacrificing efficiency.

REFERENCES

Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*, volume 4. AMLBook New York, 2012.

Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. *arXiv preprint arXiv:2102.11533*, 2021.

Peter L Bartlett and Wolfgang Maass. Vapnik-chervonenkis dimension of neural nets. *The handbook of brain theory and neural networks*, pp. 1188–1192, 2003.

Jonathon Byrd and Zachary Lipton. What is the effect of importance weighting in deep learning? In *International Conference on Machine Learning*, pp. 872–881. PMLR, 2019.

David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pp. 2702–2711. PMLR, 2016.

Ming Ding, Jie Tang, and Jie Zhang. Semi-supervised learning on graphs with generative adversarial nets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 913–922, 2018.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pp. 417–426, 2019.

Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Transactions on Knowledge and Data Engineering*, 33(6):2493–2504, 2019.

Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. Margin based feature selection-theory and algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 43, 2004.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

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. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Maya Kabkab, Emily Hand, and Rama Chellappa. On the size of convolutional neural networks and generalization performance. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 3572–3577. IEEE, 2016.

Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. URL `http://graphkernels.cs.tu-dortmund.de`.

Byung-Hoon Kim and Jong Chul Ye. Understanding graph isomorphism network for rs-fmri functional connectivity analysis. *Frontiers in neuroscience*, pp. 630, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pp. 3734–3743. PMLR, 2019.

Maosen Li, Siheng Chen, Ya Zhang, and Ivor W Tsang. Graph cross networks with vertex infomax pooling. *arXiv preprint arXiv:2010.01804*, 2020.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. Universal graph transformer self-attention networks. *arXiv preprint arXiv:1909.11855*, 2019.

Amirhossein Nouranizadeh, Mohammadjavad Matinkia, Mohammad Rahmati, and Reza Safabakhsh. Maximum entropy weighted independent set pooling for graph neural networks. *arXiv preprint arXiv:2107.01410*, 2021.

Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *computer vision and pattern recognition*, 2016.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Grover: Self-supervised message passing transformer on large-scale molecular data. *Advances in Neural Information Processing Systems*, 2020.

Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

Shuhao Shi, Kai Qiao, Shuai Yang, Linyuan Wang, Jian Chen, and Bin Yan. Boosting-gnn: Boosting algorithm for graph networks on imbalanced node classification. *Frontiers in neurorobotics*, pp. 154, 2021.

Xiang Song, Runjie Ma, Jiahang Li, Muhan Zhang, and David Paul Wipf. Network in graph neural network. *arXiv preprint arXiv:2111.11638*, 2021.

Qingyun Sun, Jianxin Li, Hao Peng, Jia Wu, Yuanxing Ning, Philip S Yu, and Lifang He. Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism. In *Proceedings of the Web Conference 2021*, pp. 2081–2091, 2021.

Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko. Bootstrapped representation learning on graphs. *arXiv preprint arXiv:2102.06514*, 2021.

Chris Thornton. Separability is a learner's best friend. In *4th Neural Computation and Psychology Workshop, London, 9–11 April 1997*, pp. 40–46. Springer, 1998.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pp. 11–30. Springer, 2015.

VN Vapnik. The nature of statistical learning theory (information science and statistics) springer-verlag. *New York*, 2000.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, et al. Deep graph library: Towards efficient and scalable deep learning on graphs. 2019.

Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968.

Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *European conference on computer vision*, pp. 499–515. Springer, 2016.

Shiwen Wu, Fei Sun, Wentao Zhang, and Bin Cui. Graph neural networks in recommender systems: a survey. *arXiv preprint arXiv:2011.02260*, 2020a.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020b.

Wenwen Xia, Yuchen Li, Jianwei Tian, and Shenghong Li. Forecasting interaction order on temporal graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1884–1893, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

Mingqi Yang, Yanming Shen, Heng Qi, and Baocai Yin. Soft-mask: Adaptive substructure extractions for graph neural networks. In *Proceedings of the Web Conference 2021*, pp. 2058–2068, 2021.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33:5812–5823, 2020.

Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. Graph information bottleneck for subgraph recognition. *arXiv preprint arXiv:2010.05563*, 2020.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. Attributed graph clustering via adaptive graph convolution. *arXiv preprint arXiv:1906.01210*, 2019a.

Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*, 2019b.

Qian Zhou, Bo Sun, Yunsheng Song, and Shuang Li. K-means clustering based undersampling for lower back pain data. In *Proceedings of the 2020 3rd International Conference on Big Data Technologies*, pp. 53–57, 2020.

Model		D&D		PROTEINS		MUTAG	
		Epoch # ± s.d.	Time	Epoch # ± s.d.	Time	Epoch # ± s.d.	Time
GraphSAGE	Original	500.6 ± 123.2	1.205	320.5 ± 53.2	1.209	384.1 ± 101.3	0.180
	CARE	293.5 ± 12.0	**1.142**	282.0 ± 51.5	**0.911**	302.2 ± 34.1	**0.159**
GCN	Original	267.4 ± 3.4	**0.692**	365.0 ± 27.9	0.670	352.4 ± 69.9	**0.132**
	CARE	264.1 ± 5.2	0.848	306.5 ± 17.2	**0.665**	332.4 ± 57.3	0.143

Table 5: Time Efficiency of CARE and Backbones . Total time (h) was recorded for a single run (including training, validation, and test) with batch size 20 and 10-fold CV. Best time in each backbone/dataset pair is highlighted in bold.

Notation	Description
G	An input graph
$\boldsymbol{A}$	Adjacency matrix of G
$\boldsymbol{X}$	Node feature matrix of G
$\mathcal{V}_G$	Node set of G
v	A node in G
n	Number of nodes in G
c	Dimensionality of node feature vector $\boldsymbol{X_v}$
$\mathcal{D}$	Input dataset
$\mathcal{G}$	Input graph set
$\mathcal{Y}$	Input label set
$\mathcal{B}_i$	Bag of (sub)graph representations of class i
y_G	Label of G
$\boldsymbol{H}$	Node representations
l	Number of layers in GNN
hg_G	Whole graph representation of G
$\boldsymbol{H}_v$	Node representation of v
m	Dimensionality of node representation $\boldsymbol{H}_v$
$\boldsymbol{G}_{sub}$	Subgraph of G
$\boldsymbol{A}_{sub}$	Adjacency matrix of G_{sub}
$\boldsymbol{H}_{sub}$	Node representations of G_{sub}
i	i-th class in $\mathcal{Y}$
hc_i	Class representation of class i
hg'_G	Refined representation of G
$\hat{y}$	Predicted class label

Table 6: Notation Table

A Related Works

A.1 Graph Neural Networks

Kipf and Welling (Kipf & Welling, 2016a) were the first to introduce the convolution operation to graphs. Later on, Hamilton et al. Hamilton et al. (2017) proposed to use sampling and aggregation to learn node representations. The attention mechanism was introduced to graph convolution in Veličković et al. (2017) to yield node representations by unequally considering messages from different neighbors. The Graph Isomorphism Network (GIN) was proposed in Xu et al. (2018), which has been shown to be as powerful as the Weisfeiler-Lehman (WL) test (Weisfeiler & Leman, 1968). Recently, the transformer architecture (Vaswani et al., 2017) has been introduced to GNN

(Nguyen et al., 2019; Rong et al., 2020), which considers the relatedness between nodes in the node representation learning.

For GNN-based graph classification, graph pooling is commonly applied as the readout function to generate graph representations. Existing pooling methods can be categorised under node drop pooling (Duvenaud et al., 2015; Zhang et al., 2018; Lee et al., 2019) and node clustering pooling (Ying et al., 2018; Zhang et al., 2019b; Li et al., 2020; Nouranizadeh et al., 2021). Node drop pooling uses learnable scoring functions to drop nodes with lower scores while node clustering pooling casts the graph pooling problem into the node clustering problem (Baek et al., 2021). In a recent development, Yang et al. (Yang et al., 2021) leverage the mask mechanism to select the subgraph by considering the consistency over samples. Both categories of the pooling methods focus on node level manipulations and neglect the graph-level information.

A.2 Methods to Treat GNN Overfitting

Expressiveness of GNN could be improved by increasing the number of model parameters through e.g., expanding the hidden dimension of the GNN layer or adding more layers. However, this process could detriment the performance and induce overfitting (Song et al., 2021).

A typical method to treat GNN overfitting is via data augmentation. Several works (Papp et al., 2021; Rong et al., 2019) used node/edge dropping augmentations. Ding et al. (2018) proposed a generator-classifier network under the adversarial learning setting to generate fake nodes. Feng et al. (2019) performed adversarial perturbations to node features. Recently, the paradigm of contrastive learning has been introduced to GNN to perform graph contrastive learning (You et al., 2020; Thakoor et al., 2021). SUGAR (Sun et al., 2021) generates subgraphs and uses these subgraphs for reconstruction, which still follows the graph contrastive learning paradigm. All these methods perform augmentation on individual graphs and again neglect the graph-level information.

Several resampling (Byrd & Lipton, 2019; Zhou et al., 2020) and reweighting (Lin et al., 2017; Shi et al., 2021) methods have also been proposed to prevent GNN from overfitting. In general, these methods design algorithms to control the influence of each sample on the model, while the sample relations are not taken into account.

To the best of our knowledge, our work is the first to explore the use of class-aware graph-level relationships to alleviate the overfitting in GNNs.

B Theoretical Proofs

B.1 Proof Sketch of Lemma 1

Our proof follows the same flow as Lemma 1 in Kabkab et al. (2016).

A parametrized class of functions with parameters in $\mathbb{R}^t$ that is computable in no more than p operations has a VC dimension which is $O(t^2p^2)$ (Bartlett & Maass, 2003). t in GCN and CARE can be formulated as:

$$t_{GCN} = \sum_{l=0}^{d} h^l_{gcn_{in}} h^l_{gcn_{out}}; \tag{8}$$

$$t_{CARE} = \sum_{l=0}^{d} (h^l_{gcn_{in}} h^l_{gcn_{out}} + h^l_{gcn_{out}} + h^l_{set_{in}} h^l_{set_{out}} + h^l_{trans_{in}} h^l_{trans_{out}}). \tag{9}$$

By plugging in the number of multiplications $q_1(d)$ and $q_2(d)$ given by Eqs. (6) and (7), together with the above equations on the number of parameters t, into $O(t^2p^2)$, we complete the proof of Lemma 1 for both GCN and CARE.

B.2 PROOF OF THEOREM 1

We compare the VC dimension upper bounds of a GCN layer and a GCN-based CARE layer under the identical number of parameters. According to Section 2.2 of Abu-Mostafa et al. (2012), the VC dimension provides a loose generalization bound for models and can be used as a guideline for generalization comparison - models with a lower upper bound tend to have better generalization capability. The number of parameters in a GCN layer t_1 and that in a GCN-based CARE layer t_2 are formulated as:

$$t_1 = h_{gcn_{in}} h_{gcn_{out}}, \tag{10}$$

$$t_2 = h_{gcn_{in}} h_{gcn_{out}} + h_{gcn_{out}} + h_{set_{in}} h_{set_{out}} + h_{trans_{in}} h_{trans_{out}}. \tag{11}$$

In our setting, we choose a basic hidden dimension h_1 and h_2 for GCN and CARE respectively. We set each layer to be an integer multiple of the basic hidden dimension. Thus, $t_1 = h_1^2$ and $t_2 = h_2^2 + h_2 + h_2^2 + 2h_2^2 = 4h_2^2 + h_2$, respectively.

Note that $h_{trans_{in}} = h_{set_{out}} + h_{gcn_{out}}$ as we concatenate the class representation with the subgraph representation.

Similarly, the computational complexities q_1 and q_2 can be rewritten as:

$$q_1(d) = \sum_{l=0}^{d} (nh_1^2 + n^2 h_1), \tag{12}$$

$$q_2(d) = \sum_{l=0}^{d} (4nh_2^2 + (2n^2 + n)h_2). \tag{13}$$

When $d = 1$, the complexity can be written as:

$$q_1(1) = nh_1^2 + n^2 h_1, \tag{14}$$

$$q_2(1) = 4nh_2^2 + (2n^2 + n)h_2. \tag{15}$$

Under the identical number of parameters, we let $t_1 = t_2$, and have $h_1 = \sqrt{4h_2^2 + h_2}$. Thus,

$$q_1(1) = 4nh_2^2 + nh_2 + n^2\sqrt{4h_2^2 + h_2}. \tag{16}$$

The difference between $q_1(1)$ and $q_2(1)$ satisfies:

$$q_1(1) - q_2(1) = n^2(\sqrt{4h_2^2 + h_2} - 2h_2). \tag{17}$$

Because $\sqrt{4h_2^2 + h_2} - 2h_2 > 0$, we have:

$$q_1(1) > q_2(1). \tag{18}$$

According to our setting, the input and output feature map sizes of all layers is identical, which means that the 'n' in each layer's complexity equation are identical. Thus, we extend Eq. (18) to the full model and have:

$$q_1(d) > q_2(d). \tag{19}$$

With Eq. (19) and Lemma 1, we complete the proof of Theorem 1.

C EXPERIMENTAL SETUP

In this section, we present the experimental setup, including datasets, GNN backbones and detailed model implementation.

C.1 DATASETS

Dataset	Graph#	Class#	Avg Node#	Avg Edge#
D&D	1178	2	284.32	715.66
PROTEINS	1113	2	39.06	72.82
MUTAG	188	2	17.93	19.79
NCI1	4110	2	29.87	32.30
NCI109	4127	2	29.68	32.13
FRANKENSTEIN	4337	2	16.90	17.88
Tox21	8169	2	18.09	18.50
ENZYMES	600	6	32.63	62.14
OGBG-MOLHIV	41127	2	25.50	27.50

Table 7: Statistics of Datasets.

Nine commonly used benchmark datasets were tested in our experiments. Eight of them were selected from TUDataset (Kersting et al., 2016) and include DD, PROTEINS, MUTAG, NCI1, NCI109, FRANKENSTEIN (FRANK), Tox21 and ENZYMES. The last dataset OGBG-MOLHIV was selected from Open Graph Benchmark (Hu et al., 2020) and consists of 41K+ graphs. The statistics of the datasets are summarized in Table 7.

C.2 GNN BACKBONES

We test the effectiveness of CARE on a wide range of GNN backbones, including GCN (Kipf & Welling, 2016a), GraphSAGE (Hamilton et al., 2017), GIN (Xu et al., 2018), GAT (Veličković et al., 2017), UGformer (Nguyen et al., 2019) , GXN (Li et al., 2020), SAGPool (Lee et al., 2019), DiffPool (Ying et al., 2018), HGPSLPool (Zhang et al., 2019b) and MEWISPool (Nouranizadeh et al., 2021). We apply CARE on each of them and compare the performance with the original backbone model. Among the 10 models selected, GIN and UGformer are hierarchical ones. We thus apply the hierarchical architecture CARE on them. The global architecture CARE is applied to the rest models. A brief introduction of each model is provided as follow:

GCN (Kipf & Welling, 2016a) is a mean pooling baseline with graph convolution network as a message-passing layer.

GAT (Veličković et al., 2017) is a mean pooling baseline, which adopts an attention mechanism to learn the relative weights between the node and its neighbors.

GraphSAGE (Hamilton et al., 2017) is a mean pooling baseline, which adopts sampling to obtain a fixed number of neighbors for each node.

GXN (Li et al., 2020) is a sort pooling (Zhang et al., 2018) baseline, which uses vertex infomax pooling to select nodes that can maximally express their corresponding neighborhoods.

SAGPool (Lee et al., 2019) is a graph pooling method that uses graph convolution in graph pooling to consider both node features and graph structure.

DiffPool (Ying et al., 2018) is an end-to-end trainable graph pooling method that can produce hierarchical representations for graphs.

HGPSLPool (Zhang et al., 2019b) is a pooling method that introduces a structure learning mechanism to refine graph structure after pooling.

MEWISPool (Nouranizadeh et al., 2021) is a pooling method, which introduces Shannon capacity to maximize the mutual information between the input graph and the output graph.

GIN (Xu et al., 2018) is a sum pooling baseline that uses a learnable parameter to adjust the weight of the central node, thus improving the message-passing network's ability to distinguish different graph structures.

UGformer (Nguyen et al., 2019) is a sum pooling baseline which identifies meta-paths to transform the graph structure for node representation learning and adopts Transformer for aggregation.

C.3 Implementation Details

The default number of graph convolutional layers in both CARE and GNN backbones is 4. We use SAGPool with a pooling ratio of 0.5 as the default subgraph selector in CARE. Notice that we did not apply any subgraph selector on GNNs that are already equipped with their own pooling methods for substructure extraction. This includes SAGPool, DiffPool, HGPSLPool and MEWISPool. The trade-off hyperparameter λ in Eq. (5) is set to 1 by default. The whole network is trained in an end-to-end manner using the Adam optimizer (Kingma & Ba, 2014). We use the early stopping criterion, i.e., we stop the training once there is no further improvement on the validation loss during 25 epochs. The learning rate is initialized to 10^{-4} and the maximum number of epochs is set to 1000. We set the hidden size to 146 and batch size to 20 for all models. The only exception is DiffPool when tested on the D&D dataset. Since the D&D dataset has a large number of nodes (see Table 7), the hidden size and batch size are set to 32 and 6 to achieve an acceptable number of parameters in DiffPool.

For TUdataset, we split it into 8:1:1 for training, validation and test. For all experiments of CARE and GNN backbones, we evaluate each model with the same random seed for 10-fold cross-validation. We use the scaffold splits for the OGBG-MOLHIV dataset and report the average ROC-AUC with 10 random seeds. All the codes were implemented using PyTorch (Paszke et al., 2017) and Deep Graph Library (Wang et al., 2019) packages. The experiments were conducted in a Linux server with Intel(R) Core(TM) i9-10940X CPU (3.30GHz), GeForce GTX 3090 GPU, and 125GB RAM.

D Hyperparameter Analysis

In this section, we study the sensitivity of two important hyperparameters in CARE, the trade-off parameter λ in the loss function and the number of layers. We test on three datasets using GIN as backbone for this set of experiments.

Trade-off Parameter λ. This hyperparameter is used in the overall loss function $\mathcal{L}$ (Eq. (5)) to trade-off between the classification loss and the class loss. We tune the value of λ from 0.01 to 10^5. The results are presented in Table 8. It shows that the choice of λ affects the performance marginally and there doesn't exist a value that works best for all datasets. In practice, we could use the validation set to find the best value of λ.

λ	D&D	PROTEINS	MUTAG
0.01	74.20 ± 3.65	71.78 ± 4.84	89.91 ± 4.35
0.1	74.11 ± 4.38	71.87 ± 4.99	**90.47** ± 5.11
1	**74.70** ± 3.37	72.32 ± 4.25	90.44 ± 4.58
10	74.45 ± 3.12	72.14 ± 4.71	89.88 ± 4.39
100	74.28 ± 3.19	**72.59** ± 4.56	89.39 ± 4.68
10^5	74.20 ± 3.33	72.05 ± 4.85	89.36 ± 4.08

Table 8: Results when Tuning λ.

Number of Layers. The depth of the neural network can certainly affect the model performance. We adjust the number of layers to investigate whether CARE can adapt to different depths of neural networks. We vary the number of layers from 2 to 5, and report the results in Table 9. For each dataset, we underline the best result among all the numbers of layers tested.

As shown in Table 9, CARE consistently outperforms GIN at different number of layers, except for 4 layers on PROTEINS where the performance difference is marginal at 0.09%. The best results

Layer#	module	D&D	PROTEINS	MUTAG
2	GIN	74.11 ± 3.42	72.42 ± 2.06	89.91 ± 4.35
	CARE	**76.40** ± 2.14	**73.22** ± 2.78	**90.47** ± 5.11
3	GIN	74.53 ± 3.36	70.79 ± 5.18	87.78 ± 4.07
	CARE	**75.13** ± 3.39	**71.69** ± 4.65	**90.47** ± 5.11
4	GIN	73.10 ± 2.44	**72.41** ± 4.45	89.36 ± 4.71
	CARE	**74.70** ± 3.37	72.32 ± 4.25	**90.44** ± 4.58
5	GIN	73.93 ± 2.62	70.71 ± 4.00	91.49 ± 4.83
	CARE	**74.70** ± 3.50	**72.69** ± 3.24	**91.52** ± 5.39

Table 9: Results when Tuning Number of Layers.

are achieved with 2 layers on D&D and PROTEINS and with 5 layers on MUTAG. Therefore, the number of layers should also be selected through the validation process for different datasets.

E Effectiveness Analysis under the same Parameter Number

CARE is proposed as a plug-and-play framework. However, in addition to directly plugging it in a GNN backbone without changing the number of model parameters in the backbone (as what we have done in experiments in the submitted version), it could also be used in a way that the resultant CARE after plug-in has a comparable number of parameters to the original GNN backbone before plug-in. This can be achieved by adjusting the number of parameters in the GNN backbone at the time when CARE is plugged in. To demonstrate this, we conduct a new experiments to match with the setting of Theorem 1. For each GNN backbone, we first set its number of parameters to 100K. For CARE, we adjust the hidden dimension of each GNN backbone to which CARE is applied such that the number of parameters of CARE is also 100K. As shown in the table below, CARE still outperforms its GNN backbone in 8 out of 9 cases. The results demonstrate that CARE can boost up the graph classification performance without introducing additional parameters.

		DD	PROTEINS	MUTAG
GraphSAGE	original	72.18 ± 2.93	74.87 ± 3.38	75.48 ± 6.11
	CARE	**72.22** ± 3.10	**75.74** ± 1.68	**76.08** ± 10.83
GCN	original	71.02 ± 3.17	73.89 ± 2.85	77.52 ± 10.81
	CARE	**71.73** ± 4.12	**74.91** ± 3.59	**79.27** ± 4.31
GIN	original	73.10 ± 2.44	72.41 ± 4.45	89.36 ± 4.71
	CARE	**73.19** ± 4.44	**70.43** ± 4.69	**89.70** ± 5.53

Table 10: Graph Classification Results (Average Accuracy ± Standard Deviation) under the same parameters setting. The parameter numbers of all models are 100K. Winner in each backbone/dataset pair is highlighted in bold.

F Comparison with GIB

GIB proposes a novel objective to recognize the maximally informative subgraph. It designs a bi-level optimization scheme and a connectivity loss to optimize the GIB objective. As both GIB and our work consider graph-level similarity, we add it as a new baseline in our experiments. The results of GIB are taken from its original paper. As shown in the table below, as a plugin component, CARE outperforms GIB in 7 out of 12 cases.

G Ablation Study on Similarity Metric for Class Loss

We study the effect of the similarity metric in computing the class loss. Besides the cosine similarity, L2 distance is also commonly used to measure the dissimilarity between two vectors. We use L2 distance in place of the cosine similarity in Eqs. (3) and (4) to define the intra-class and inter-class losses. As L2 distance measures the dissimilarity, we take an inverse of the class loss when L2

		DD	PROTEINS	MUTAG
GraphSAGE	original in GIB paper	72.9 ± 4.1	72.1 ± 4.2	74.3 ± 7.7
	GIB	**78.1 ± 4.2**	73.4 ± 4.3	76.0 ± 7.4
	original in our paper	72.2 ± 2.9	74.9 ± 3.4	75.5 ± 6.1
	CARE	73.3 ± 3.3	**75.9 ± 2.8**	**82.0 ± 6.42**
GCN	original in GIB paper	72.5 ± 4.6	71.9 ± 4.1	74.3 ± 11.0
	GIB	**76.5 ± 5.0**	74.8 ± 4.6	77.6 ± 7.5
	original in our paper	71.0 ± 3.2	73.9 ± 2.9	77.5 ± 10.8
	CARE	72.2 ± 3.9	**75.0 ± 2.9**	**79.3 ± 11.8**
GIN	original in GIB paper	73.0 ± 3.3	70.7 ± 4.8	82.5 ± 6.8
	GIB	**74.7 ± 3.9**	**74.9 ± 5.0**	83.9 ± 6.4
	original in our paper	73.1 ± 2.4	72.4 ± 4.5	89.4 ± 4.7
	CARE	**74.7 ± 3.4**	72.3 ± 4.3	**90.4 ± 4.6**
GAT	original in GIB paper	69.5 ± 4.5	71.4 ± 4.0	73.8 ± 7.4
	GIB	**76.9 ± 4.0**	73.7 ± 4.4	74.9 ± 9.7
	original in our paper	74.3 ± 3.8	74.3 ± 2.1	77.6 ± 10.5
	CARE	75.4 ± 2.9	**76.7 ± 1.7**	**77.7 ± 9.0**

Table 11: Graph Classification Results (Average Accuracy ± Standard Deviation) comparing with GIB. Winner in each GIB/CARE pair is highlighted in bold.

distance is used. We then compare the performance of CARE under these two different metrics. Table 12 shows the results under GCN as the GNN backbone. It shows that CARE with the cosine similarity is more powerful than that with the L2 distance.

$$\mathcal{L}_{intra} = norm(\sum_{i=1}^{|\mathcal{Y}|} norm(\sum_{y_G=i} dis(hc_i, hg_G^{sub}))), \tag{20}$$

$$\mathcal{L}_{inter} = norm(\sum_{i=1}^{|\mathcal{Y}|} norm(\sum_{j=i}^{|\mathcal{Y}|} dis(hc_i, hc_j))), \tag{21}$$

$$\mathcal{L}_{class} = \exp(\mathcal{L}_{intra} - \mathcal{L}_{inter}). \tag{22}$$

	D&D	PROTEINS	MUTAG
-	71.39 ± 2.81	74.21 ± 2.74	77.11 ± 9.48
L2	71.91 ± 4.97	74.65 ± 3.54	77.69 ± 8.35
KL	71.85 ± 4.49	74.38 ± 3.16	78.74 ± 5.19
cos_sim	**72.15** ± 3.88	**75.01** ± 2.91	**79.30** ± 11.81

Table 12: Ablation Study on Similarity Metric for Class Loss.

H Class Separability Metrics in Case Study

H.1 Silhouette Coefficient

The Silhouette of a sample x_i is defined as:

$$sil(x_i) = \frac{b^i - a^i}{max(a^i, b^i)}, \tag{23}$$

$$Silhouette = \underset{x_i}{AVERAGE}(\{sil(x_i)\}), \tag{24}$$

where a^i denotes the average distance between x_i and all other samples in the same class, and b^i denotes the smallest mean distance from x_i to all samples in any other class.

H.2 SEPARABILITY INDEX

The Separability Index SI is defined as:

$$K(x_i, x_j) = \begin{cases} 1, & \text{if } y_i = y_j, y_i \in \mathcal{Y}, y_j \in \mathcal{Y} \\ 0, & \text{otherwise} \end{cases}, \tag{25}$$

$$x_i' = \underset{x_j \neq x_i}{\arg\min}(\|x_i - x_j\|), \tag{26}$$

$$SI = \frac{\sum_{x_i} K(x_i, x_i')}{m}, \tag{27}$$

where m is the total number of samples, x_i denotes the i-th sample, y_i denotes its corresponding class label, and $\mathcal{Y}$ denotes the set of classes. The nearest neighbour distance function $\|\cdot\|$ is assumed to utilise a suitable metric, e.g., a Manhalobis metric for symbolic data or a Euclidean metric for spatial data.

H.3 HYPOTHESIS MARGIN

The Hypothesis Margin (HM) is defined as:

$$hm(x_i) = \frac{\|x_i - \textbf{nearmiss}(x_i)\|}{\|x_i - \textbf{nearhit}(x_i)\|}, \tag{28}$$

$$HM = \underset{x_i}{AVERAGE}(\{hm(x_i)\}), \tag{29}$$

where **nearhit**(x_i) and **nearmiss**(x_i) denote the nearest sample to x_i with the same and different label, respectively. $\|\cdot\|$ denotes the L2 distance. Note that a chosen set of features affects the margin through the distance measure.

H.4 CENTROID DISTANCE

The Centroid Distance (CD) is defined as:

$$c_i = AVERAGE(\{x\}_{y_x = i}), \tag{30}$$

$$CD = \sum_{i=1}^{|\mathcal{Y}|} \sum_{j=i}^{|\mathcal{Y}|} \|c_i - c_j\|, \tag{31}$$

where y_x denotes the class label of a sample x, $\mathcal{Y}$ denotes the set of classes and $\|\cdot\|$ denotes the L2 distance.