GRAPH-GRAPH SIMILARITY NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph classification aims to predict the class label for an entire graph. Recently, Graph Neural Networks (GNNs)-based approaches become an essential strand to learn low-dimensional continuous embeddings of the entire graphs for graph label prediction. While GNNs explicitly aggregate the neighborhood information and implicitly capture the topological structure for graph representation, they ignore the relationships among graphs. In this paper, we propose a Graph-Graph Similarity Network to tackle the graph classification problem by constructing a SuperGraph through learning the relationships among graphs. Each node in the SuperGraph represents an input graph, and the weights of edges denote the similarity between graphs. By this means, the graph classification is then transformed into a classical node classification problem. Specifically, we employ an Adversarial Autoencoder to align embeddings of all the graphs to a same distribution. After the alignment, we design the Graph-Graph Similarity Network to learn the similarity between graphs, which functions as the adjacency matrix of the SuperGraph. By running node classification algorithms on the SuperGraph, we can predict the labels of graphs. Experiments on five widely used benchmarks under a fair setting demonstrate the effectiveness of our method.

1 INTRODUCTION

Nowadays, there is an increasing number of applications on graph-structured data ranging from e-commence to biological molecules. In the graph structure, each node represents an entity, and edges linking two nodes denote the relationship between the two entities. For instance, in citation networks, each node denotes a paper, and edges indicate the citationships between papers. Another example is that in social networks, nodes and edges respectively represent users and their friendships. As graphs are always irregular with various number of nodes and edges, it is difficult to apply Convolutional Neural Networks (CNNs) (LeCun et al., 1995) to the graph domain directly. Encouraged by the success of CNNs, Graph Neural Networks (GNNs) (Scarselli et al., 2008) have recently become the most popular tool for graph-structured data by effectively combining node features and graph topology. The research on GNN is mainly developing into two lines. The first category is to infer labels of individual nodes by a given graph structure, and the second one is to predict the class labels of unseen graphs by a given set of graphs with different structures and sizes. In this paper, we focus on the second one, graph classification problem.

The majority of the GNN-based methods (Hamilton et al., 2017b; Zhou et al., 2018; Wu et al., 2020; Zhang et al., 2018a;b; Ying et al., 2018; Wang et al., 2018; Juan et al., 2019; Bui et al., 2018) mainly involve transforming, propagating, and aggregating node features across the graph, where they focus on the supervised or semi-supervised scenarios using the graph labels during the training process. When no class labels are available in graphs, some methods (Kipf & Welling, 2016b; Pan et al., 2018) have been proposed to learn the graph embeddings in an unsupervised way by adopting an autoencoder framework. Whether supervised or not, the unfixed structures and number of nodes bring in huge difficulties in graph classification. Firstly, the representations generated by convolution are high-level node representations. While the task is to predict labels for graphs, it is necessary to find a way to extract graph-level features. Secondly, the sizes and structures of graphs are different from each other. In order to generate representations for graphs by a shared model, how to align all the graphs to a same distribution is essential to be solved. Last but not least, the current graph classification methods seek the independent graph representation. Unfortunately, the relationships among different graphs are ignored.

In this paper, we propose a Graph-Graph Similarity Network to tackle the graph classification problem by constructing a SuperGraph from all the input graphs. While previous GNN-based approaches focus on generating graph-level representations to denote the identities of graphs, we present to find the relationship between graphs and make the prediction with this global information. Specifically, we first capture high-level node representations by Graph Convolutional Network. Then we utilize the Adversarial Autoencoder (Makhzani et al., 2015) to align all the graphs to a same space distribution. The max-pooling method is employed to down-sample and collapse node representations into graph representations. Next, we design a neural network for learning the similarity between graphs by the generated graph representations, which is the key part for the construction of the SuperGraph. On the SuperGraph, we run the K-Nearest Neighbors algorithm (Duda & Hart, 1973) and get the predictions for the graphs. In summary, our contributions are as follows:

- We address the problem of measuring the similarity between graphs. Comparing with the traditional metrics that are pre-defined and fixed, our method is more flexible and efficient with the guidance of the graph labels and network inference.
- We propose a novel Graph-Graph (G2G) model consisting of Graph Convolutional Network, Heterogeneous Space Alignment, Graph-Graph Similarity Network, and SuperGraph to learn effective graph-level representations under a common representation space and explore the complex relationships between graphs, converting the graph classification problem to the node classification problem, and finally predict labels for graphs.
- Extensive experimental results demonstrate that our G2G model outperforms other baseline models on all the five public datasets. We also provide similarity visualizations and an ablation study on G2G, which demonstrates the effectiveness of our Graph-Graph Similarity Network.

2 Related Works

In this section, we briefly review related work on graph classification and metric learning. Our work is also connected with Graph Convolutional Network and Generative Adversarial Network, whose related work is discussed comprehensively and thoroughly in their surveys (Wu et al., 2019; Pan et al., 2019). At the end of this part, we explain how our method is different from previous studies.

Graph Classification. Graph classification is the task of predicting the class labels of unseen graphs. Early popular approaches adopt graph kernels, which allow kernel-based methods such as Support Vector Machine (SVM) (Suykens & Vandewalle, 1999) to work directly on graphs for graph classification. By decomposing graphs into small substructures(e.g., shortest paths (Borgwardt & Kriegel, 2005), random walks (Vishwanathan et al., 2010), subtree structures (Shervashidze et al., 2011), or graphlets (Kriege & Mutzel, 2012)), the kernel function compares two graphs by comparing all the pair-wise substructures. However, graph kernels are restricted to substructures with few nodes because of the combinatorial complexity of substructure enumeration. Recently, Graph Neural Networks (GNNs) have been popular because they can extract features from graphs efficiently. GraphSAGE (Hamilton et al., 2017a) leverages node feature information to efficiently generate node embeddings for previously unseen data. Graph Attention Networks (GATs) (Veličković et al., 2017) utilize the attention mechanism to aggregate neighborhood representations with different weights. While the above methods are mainly designed for learning meaningful node representations, they face a computational challenge and are unable to generate graph-level representations. To solve this problem, other GNN-based approaches are often combined with pooling operations, which not only down-sample the nodes to generate smaller representations but also obtain a compact representation on the graph level. For example, DiffPool (Ying et al., 2018) proposes to softly assign nodes to a set of clusters on the basis of a supervised criterion. DGCNN (Zhang et al., 2018b) enables learning from global graph topology by sorting vertex features instead of summing them up.

Metric Learning. Many classification methods are based on calculating the difference or similarity between two data points. Traditional distance metrics are commonly used in machine learning, such as Euclidean distance (Lee et al., 2012), Cosine distance (Kim, 2014), and Wasserstein distance (Shen et al., 2018; Frogner et al., 2015). Differently, metric learning aims to learn distance functions, which are then used to support tasks like classification and clustering. Some methods (Lim & Lanckriet, 2014; McFee & Lanckriet, 2010) learn a linear transformation of the input data, while others (Bai et al., 2019; Li et al., 2019) employ deep neural networks for nonlinear metric learning, which is known as deep metric learning. Driven by loss functions, deep metric learning learns

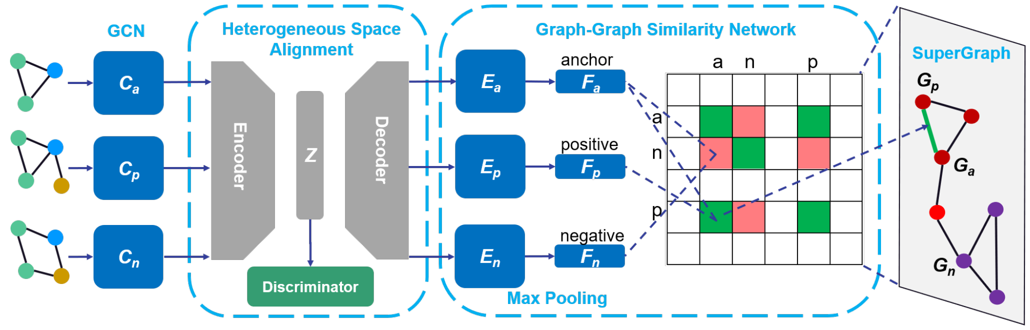

Figure 1: Graph-Graph (G2G) model overview. There are four components in the framework: Graph Convolutional Network (GCN), Heterogeneous Space Alignment, Graph-Graph Similarity Network, and SuperGraph. The input graphs are organized into triplets. For each triplet, an anchor graph, a positive graph, and a negative graph is included, where the positive graph has the same label with the anchor graph, and the negative has a different label with the anchor graph. Firstly, all the graphs are processed by a shared GCN to get informative embeddings. Then in the Heterogeneous Space Alignment, all the embeddings are aligned to a same distribution. After the alignment, a max-pooling layer is utilized to obtain the feature representation for each graph, and a Graph-Graph Similarity Network is designed to calculate the similarity between graphs. Finally, a SuperGraph is built by the feature representations and the similarity matrix. Then a node classification algorithm can be applied to the SuperGraph to predict the labels of the graphs.

feature embedding from the input data, which shows the importance of the loss function definition. Contrastive loss (Hadsell et al., 2006) is one of the most popular pairwise losses, which minimizes the distance between pairs of the same class (positive), and maximizes the distance between pairs of different classes (negative). Different from contrastive loss, triplet loss (Schroff et al., 2015) is defined based on an anchor sample, a positive sample, and a negative sample. Triplet loss defines the difference by relative similarity, and thus suits for more scenarios compared to the contrastive loss.

Our work is different from previous graph classification studies in the following aspects. Kernel-based methods cannot generate graph-level representations automatically and compute efficiently on large scale graphs, and GNN-based approaches ignore the relationship between graphs, while our work intends to involve both informative graph-level features and similarity between graphs when predicting class labels for graphs. In the metric learning aspect, traditional distance metrics are all pre-defined and fixed, so we adopt the deep metric learning method to acquire the distances by training a neural network together with zero-one loss and triplet loss.

3 Methodology

In this section, we first elaborate on the addressed graph classification problem, then introduce our proposed Graph-Graph (G2G) model, followed by the designated objective function in detail.

3.1 Problem Definition

A graph can be represented by $G = (V, X, A)$, where $V = \{v_1, v_2, ..., v_n\}$ is the set of vertexes, $X \in \mathbb{R}^{n \times d}$ denotes the features of each vertex, and $A \in \{0,1\}^{n \times n}$ represents the adjacency matrix. Given a set of labeled graphs $\mathcal{D} = \{(G_1, y_1), (G_2, y_2), ..., (G_N, y_N)\}$ where $y_i \in \mathcal{Y}$ is the corresponding label of graph $G_i \in \mathcal{G}$, the graph classification problem can be defined as to learn a mapping function $f : \mathcal{G} \to \mathcal{Y}$, which maps graphs to labels. To achieve the goal, we further define a procedure $g : \mathcal{G} \to (F, S)$ to convert graphs to finite-dimensional representations $F \in \mathbb{R}^{N \times D}$ and a similarity matrix $S \in \mathbb{R}^{N \times N}$ that denotes the similarity between graphs. N is the number of graphs in $\mathcal{G}$, and D is the dimension of the produced representations. A mapping function $h : (F, S) \to \mathcal{Y}$ is employed to predict labels for graphs.

In this graph classification problem, there are three challenges we have to deal with. The first one is how to get a fixed-length vector representation for each graph. A graph consists of vertexes, vertex features, and the relationship between vertexes. In order to apply standard machine learning methods for classification, it is necessary to find a way to extract useful feature vectors that contain both feature and relationship information of nodes from these input graphs. The second one is how to align the graphs to a same distribution. The existing GNN-based methods aggregates the node-level embeddings to present a graph, which cannot guarantee that all graph representations are in the same feature space due to the differences in sizes and structures of graphs. Thus it is essential to learn a common graph-level representation space. And the third one is how to calculate a meaningful similarity between graphs. With the vector representation of the graphs, Euclidean distance, cosine distance, Wasserstein distance, and graph alignment techniques can be used for graph similarity. However, these pre-defined metrics are inflexible to capture the similarity across multiple graphs.

3.2 Model

The goal of graph classification is to predict class labels associated with the graphs. While most Graph Neural Network (GNN)-based approaches generate graph-level representations for graphs by utilizing the node features and graph structure information, they ignore the connections between graphs. To exploit this relationship instead of only focusing on the identities of graphs, we present a model to calculate the similarity between graphs and build a SuperGraph by the similarity and graph level representations of graphs, which converts the graph classification problem to the node classification problem, and thus involves the relationship between graphs.

Specifically, to solve the graph classification problem and tackle the above challenges, we proposed the Graph-Graph (G2G) model illustrated in Figure 1. We first use Graph Convolutional Network to combine both node features and relationship between nodes of a graph. Then an Adversarial Autoencoder is employed to learn an invariant space for all the graph representations, which is essential for the similarity calculation. The generator produces the fake graph representation, while the discriminator judges whether the input graph representation is real or fake. The generator learns a universal graph space, and all the graph representations from GCN are fed into the unified generator to eliminate the different feature spaces issue. After the alignment, a pooling layer is added to get finite-dimensional vector representations for graphs with different number of nodes. With the features for graphs, a Graph-Graph Similarity Network trained by zero-one loss and triplet loss is designed to get the similarity between graphs, solving the problem of similarity calculation. The zero-one loss works as a guide for training, and the triplet loss makes the generated pair-wise similarity meaningful by encouraging each graph closer to graphs that belong to the same class with it than graphs of different classes. The feature representations of graphs, together with the similarity matrix, can formulate another graph, which is called the SuperGraph in this paper. Each node of the SuperGraph denotes an input graph, and the weights of edges in the SuperGraph is denoted by the similarity matrix. Then the graph classification problem is transformed into the node classification problem. Finally, a standard machine learning method for node classification can be utilized on the SuperGraph to predict the labels for the graphs, which captures not only the features of graphs but also the relationship between graphs. In general, Our model takes both graph structure similarity and graph label similarity into consideration. The graph structure similarity is captured by the GCN and the Heterogeneous Space Alignment part, and Supergraph takes the graph labels into account. The model takes graphs as the input and outputs the feature representations of graphs and the similarity between graphs in the first stage. Then for the second stage, we run a node classification method for node classification on the SuperGraph to get the predictions of graph labels.

3.3 Objective Function

Our model consists of four parts: Graph Convolutional Network (GCN), Heterogeneous Space Alignment, Graph-Graph Similarity Network, and SuperGraph. In this section, We use $\theta = \{\theta_G, \theta_Q, \theta_P, \theta_S\}$ to denote the trainable parameters set of four parts in the G2G model. Specifically, θ_G denotes the trainable parameters in the GCN part, θ_Q is the trainable parameters in the encoder, θ_P is the trainable parameters in the decoder, and θ_S represents the trainable parameters in the Graph-Graph Similarity Network. Our goal is to minimize the objective function through adjusting θ by the model and dataset. Each part is detailed as follows.

Graph Convolutional Network. We use GCN (Kipf & Welling, 2016a) to make use of both node features and the relationship of nodes in a graph. While the features of nodes are of the same dimension, a shared GCN can be employed for all the graphs. The Graph Convolutional Network

(GCN) embedding of the static inputs is as follows:

$$C_{\theta}^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}C_{\theta}^{(l)}\theta_{G}^{(l)}). \tag{1}$$

Here $\tilde{A} = A + I_n$ is the adjacency matrix A with added self-connections. $I_n \in \mathbb{R}^{n\times n}$ is the identity matrix, $\tilde{D}$ is the degree matrix of $\tilde{A}$, and $\theta_{G}^{(l)}$ is a layer-specific trainable weight matrix. $C^{(l)}$ denotes the matrix in the lth layer, and $C^{(0)} = X$. We employ $\sigma(\cdot) = ReLU(\cdot)$ as the activation function.

Heterogeneous Space Alignment. This part is designed as an Adversarial Autoencoder (Makhzani et al., 2015), which includes an encoder, a decoder, and a discriminator, aiming to match the aggregated posterior of the hidden codes with an arbitrary prior distribution using an adversarial training procedure. Specifically, let C be the input and z be the hidden code of the autoencoder. Denote by $q(z|C)$ and $q(C|z)$ an encoding distribution and the decoding distribution, respectively. Then the aggregated posterior distribution of $q(z)$ on the hidden codes can be computed as $q(z) = \int_C q(z|C)p(C)dC$, where $p(C)$ is the marginal distribution of inputs. Let $p(z)$ be the prior distribution one wants to impose on the codes. The network minimizes the reconstruction error for the autoencoder, and meanwhile guides $q(z)$ to match $p(z)$ through attaching an adversarial network on top of the hidden codes of the autoencoder.

Let E be the generated embeddings. Here we use $Q(X) = \sigma(X \times \theta_Q)$ to denote the encoder, $P(X) = \sigma(X \times \theta_P)$ the decoder. Then $E_\theta = P(Q(C_\theta))$. $D(\cdot)$ denotes the discriminator telling apart the true hidden codes sampled based on z. Then the reconstruction loss of the autoencoder is calculated by the following equation:

$$\mathcal{L}_{\mathcal{R};\theta} = \sum_{i}^{N} \|E_{i;\theta} - C_{i;\theta}\|_2^2. \tag{2}$$

We utilize the max-pooling method to get vector representations from the hidden layer of the autoencoder. The adversarial loss is as follows:

$$\mathcal{L}_{\mathcal{A};\theta} = \mathbb{E}_{z\sim p(z)}[\log D(z)] + \mathbb{E}_{C\sim p(C)}[\log(1 - D(\mathrm{maxpool}(Q(C_\theta))))]. \tag{3}$$

Graph-Graph Similarity Network. The features of graphs F is generated by a pooling layer. Here we also use the max-pooling method, that is $F_\theta = \mathrm{maxpool}(E_\theta)$. Then to get the similarity between graphs, we employ a neural network to simulate the calculation process, which is shown as follows:

$$S_\theta = \sigma(F_\theta \times \theta_S) \times \sigma(F_\theta \times \theta_S)^\top. \tag{4}$$

To guide the calculation of similarity matrix, we first build a Label matrix Y, where $Y_{i,j} = 1$ if $y_i \neq y_j$, and otherwise $Y_{i,j} = 0$. Then we utilize both zero-one loss and triplet loss (Schroff et al., 2015) as the constraints, which are designed as follows:

$$\mathcal{L}_{\mathcal{Z};\theta} = \sum_{i}^{N}\sum_{j}^{N}((1 - S_{i,j;\theta})^2 \cdot Y_{i,j} + S_{i,j;\theta}^2 \cdot (1 - Y_{i,j})), \tag{5}$$

$$\mathcal{L}_{\mathcal{T};\theta} = \sum_{(G_a,G_p,G_n)} \max(S_{a,n;\theta}^2 - S_{a,p;\theta}^2, \alpha), \tag{6}$$

where α is the margin of triplet loss, and (G_a, G_p, G_n) denotes a triplet of graphs. G_a is the anchor graph, G_p is the positive graph, and G_n is the negative graph. Here positive/negative means the samples that have the same/different labels with a chosen sample (also known as Anchor).

Combining Eq. (2), Eq. (3), Eq. (5) and Eq. (6), our overall objective function is shown as follows:

$$\min_\theta \mathcal{L}_{\mathcal{R};\theta} + \lambda_A\mathcal{L}_{\mathcal{A};\theta} + \lambda_Z\mathcal{L}_{\mathcal{Z};\theta} + \lambda_T\mathcal{L}_{\mathcal{T};\theta}, \tag{7}$$

where λ_A, λ_Z, and λ_T are hyperparameters controlling the weights of $\mathcal{L}_\mathcal{G}$, $\mathcal{L}_\mathcal{Z}$, and $\mathcal{L}_\mathcal{T}$ respectively. Then we utilize Stochastic Gradient Descent (Robbins & Monro, 1951) to optimize the objective function in the discriminator, and Adam (Kingma & Ba, 2014) to minimize other parts.

SuperGraph. With the generated features F and the similarity matrix S, we get a topological graph representation, which is called the SuperGraph in this paper. In the SuperGraph, each node corresponds to an input graph, and the weight of each edge denotes the relationship between the corresponding graphs of the two vertexes. With this SuperGraph, we turn the graph classification problem into the node classification problem, and many machine learning methods like graph kernels or GCNs can be applied on it without limitation. In our experiments, we employ the K-Nearest Neighbors algorithm (Duda & Hart, 1973) on the SuperGraph and get the predictions.

Table 1: Statistics of five datasets, MUTAG, ENZYMES, PROTEINS, D&D, and NCI1.

Dataset	# Graph	# Class	Avg. # Node	Avg. # Edge	# Node Label	Node Attr.
MUTAG	188	2	17.93	19.79	7	No
ENZYMES	600	6	32.63	62.14	3	Yes
PROTEINS	1113	2	39.06	72.82	3	Yes
D&D	1178	2	284.32	715.66	89	No
NCI1	4110	2	29.87	32.30	37	No

4 Experiments

In this section, we first describe the datasets and experimental setup, then evaluate our model. Finally, we provide exploitative experiments for ablation study and comparison with different distance metrics.

4.1 Dataset

We evaluate our model on five datasets, graphs in which are derived from small chemical compounds or protein molecules: MUTAG (Debnath et al., 1991), ENZYMES (Schomburg et al., 2004), PROTEINS (Borgwardt et al., 2005), D&D (Dobson & Doig, 2003), and NCI1 (Wale et al., 2008). Table 1 shows the statistics and properties of each dataset. In these datasets, each graph belongs to only one class, and all the nodes are labeled. For ENZYMES and PROTEINS, node attributes are provided, which are used as node features in our experiment. For the other three datasets, we utilize one-hot node labels as the node features. For all the datasets, we perform 10-fold cross validation (Cawley & Talbot, 2010) to evaluate model performance, and report the accuracy averaged over 10 folds. Specifically, we employ the splits of the datasets provided by Errica et al. (2019), who provides a grounding for rigorous evaluations of graph classification models. While they did not include a split for MUTAG, we generate a 10-fold split for MUTAG and test all the methods on it.

4.2 Baselines and Experimental Settings

We select five baseline methods that are based upon Graph Neural Networks: GraphSAGE (Hamilton et al., 2017a), GIN (Xu et al., 2018), ECC (Simonovsky & Komodakis, 2017), DGCNN (Zhang et al., 2018b), and DiffPool (Ying et al., 2018). GraphSAGE first performs sum, mean or max-pooling neighborhood aggregation. GIN extends it with arbitrary aggregation functions on multi-sets. ECC weights neighbor aggregation according to specific parameters learned for each edge label. DGCNN proposes a SortPooling layer, which performs pooling by sorting vertex features to a meaningful order. DiffPool learns a differentiable soft cluster assignment for nodes at each layer of a deep GNN to hierarchically pool graph nodes. All the five methods utilize the information within a graph to predict its label. The settings of each baseline method follow the work of Errica et al. (2019), and the displayed experimental results of the baseline methods in Tabel 2 on all the datasets except for MUTAG. For MUTAG, we run the tool provided by Errica et al. (2019) with their settings[1].

In experiments, we employ pseudo nodes to pad the input graphs to a same size. The features of pseudo nodes are zero-vectors, and the pseudo nodes do not link to any other nodes. All network parameters of our proposed Graph-Graph (G2G) model are initialized by xavier uniform (Glorot & Bengio, 2010). The hyperparameter settings are as follows. The weights of the adversarial loss λ_A, the zero-one loss λ_Z, and the triplet loss λ_T are separately set to 0.1, 1.0, and 1.0. The margin of triplet loss α is 0.5. We adopt a two-layer GCN and apply batch normalization after each layer of the model. We run the training stage 500 epochs for each dataset. The Adam optimizer with a learning rate of 0.0003 is utilized to minimize the loss for the G2G model, and the SGD optimizer with a learning rate of 0.0001 is to optimize the trainable parameters in the discriminator.

4.3 Results

The classification results are listed in Table 2. As we can see, our proposed G2G model achieves the best scores on all five datasets. Experiments demonstrate that our method consistently outperforms

[1]Since we follow the setting of Errica et al. (2019) for fair comparisons, the performance of the baseline methods on MUTAG might be different from the ones reported in the original papers

Table 2: Experimental results (% accuracy with standard deviation). The best scores are in **bold**.

Methods	MUTAG	ENZYMES	PROTEINS	D&D	NCI1
GraphSAGE	83.3 ± 9.0	58.2 ± 6.0	73.0 ± 4.5	72.9 ± 2.0	76.0 ± 1.8
GIN	83.9 ± 7.8	59.6 ± 4.5	73.3 ± 4.0	75.3 ± 2.9	80.0 ± 1.4
ECC	75.9 ± 6.4	29.5 ± 8.2	72.3 ± 3.4	72.6 ± 4.1	76.2 ± 1.4
DGCNN	80.3 ± 10.9	38.9 ± 5.7	72.9 ± 3.5	76.6 ± 4.3	76.4 ± 1.7
DiffPool	81.2 ± 9.3	59.5 ± 5.6	73.7 ± 3.5	75.0 ± 3.5	76.9 ± 1.9
G2G	$\mathbf{90.3 \pm 6.4}$	$\mathbf{68.0 \pm 3.6}$	$\mathbf{78.3 \pm 2.2}$	$\mathbf{77.0 \pm 4.8}$	$\mathbf{80.3 \pm 2.4}$

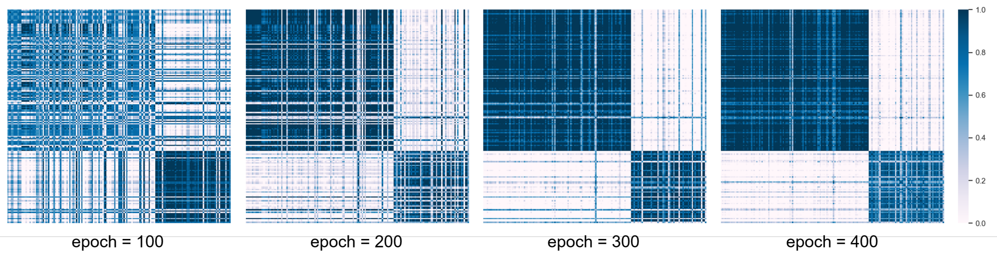

Figure 2: Similarity matrices of the training set of MUTAG along different iterations. The final similarity matrix of MUTAG with epoch=500 is in Figure 3.

baseline methods on all five datasets. Moreover, our results excel the second best method by nearly 5%, 10% and 5% on MUTAG, ENZYMES, and PROTEINS and pass all significance tests with p-value < 0.05. While on other two datasets, our results also pass the significance tests against other baseline methods, except for DGCNN on D&D and GIN on NCI1, which indicates that our model performs significantly better than the baseline models in most cases. The reason is that previous GNN-based graph classification methods build a classifier on top of the graph-level representations, while we transform it to label propagation on the SuperGraph. Instead of using independent graph information like other methods, our model incorporates the relationship between graphs by the generated pairwise similarity matrix. Together with the zero-one loss which drives the training process of the Graph-Graph Similarity Network, the triplet loss further promotes graphs closer to its corresponding positive graphs than to the negative ones under our learned distance metric.

To further analyze the performance of the G2G model, we visualize the similarity matrix of the SuperGraph generated by our model. Figure 2 displays the similarity matrices on one training set of MUTAG along iterations, where the darker color denotes the higher similarity. It is expected to see that the similarity matrix becomes more and more block-structured with more iterations, which indicates the effectiveness of the G2G model. Moreover, Figure 3 shows the similarity matrices on one training set of MUTAG, ENZYMES, D&D, and NCI1, and their similarity matrices on the corresponding testing sets. The classes can be easily identified on the training set, and the dark squares lie on the diagonal of the matrix, demonstrating the effectiveness of our designed objective function for training. Although the performance drops a little on the testing set, the dark squares are still recognizable on the diagonal on MUTAG, D&D, and NCI1. These demonstrate the generalization of our G2G model. The structure of ENZYMES is not so significant as on other datasets. This might result from the multi-classes of graphs. Instead of two classes in the other four datasets, ENZYMES includes six kinds of class labels, which brings more challenges to the classification problem.

4.4 Ablation study

To study the impact of the Graph-Graph Similarity Network and the function of SuperGraph, we run an ablation study on the five datasets. The results are reported in Table 3. We can clearly see that with Heterogeneous Space Alignment part, the performance improves, which demonstrates the positive effect of Aversarial Autoencoder, because it learns a common graph-level representation space. While by adding a Graph-Graph Similarity Network to the "C+H" model, the performance is improved even without using the learned similarity. It is because the triplet loss optimizes the representations such that graphs belonging to the same class are closer to each other than those of different classes. The "C+H+G" model already outperforms the baseline methods on MUTAG and ENZYMES and has comparable results with them on PROTEINS, D&D, and NCI1. Furthermore, building the SuperGraph with the generated similarity matrix increased the performance again. Overall, The

Table 3: Ablation study results (% accuracy with standard deviation). The best scores are in **bold**. Here C, H, G and S respectively denote Graph Convolutional Network, Heterogeneous Space Alignment, Graph-Graph Similarity Network, and SuperGraph.

Models	MUTAG	ENZYMES	PROTEINS	D&D	NCI1
C	83.0 ± 11.5	55.2 ± 6.6	70.4 ± 4.1	72.4 ± 2.5	72.6 ± 2.0
C + H	86.2 ± 7.1	57.7 ± 7.9	70.7 ± 2.8	72.7 ± 4.4	75.1 ± 2.4
C + H + G	87.1 ± 7.1	65.7 ± 6.6	73.0 ± 3.8	74.4 ± 3.6	77.9 ± 2.1
C + H + G + S	**90.3** ± **6.4**	**68.0** ± **3.6**	**78.3** ± **2.2**	**77.0** ± **4.8**	**80.3** ± **2.4**

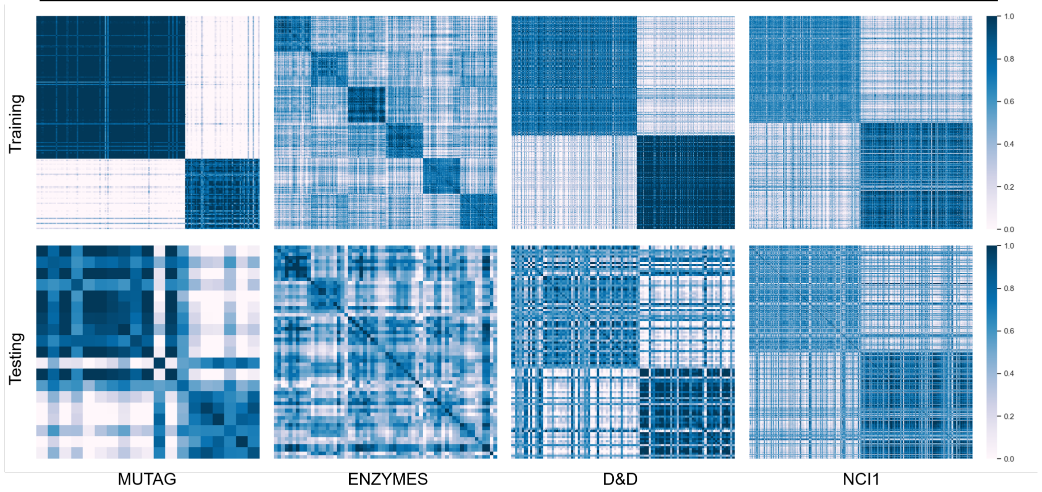

Figure 3: Visualization of final similarity matrices on MUTAG, ENZYMES, D&D, and NCI1.

controlled ablation study in Table 3 shows that each component has a necessary and positive effect on G2G model, which verifies our motivations.

4.5 Comparison with Different Distance Metrics

We run some experiments on the representations extracted by the G2G model with Euclidean distance, cosine distance, and Wasserstein distance as the comparisons. We first calculate the pair-wise distance of the graph-level features and then apply the K-Nearest Neighbour algorithm like G2G to get the predictions for each kind of distance metrics. Figure 4 exhibits the average accuracy of each distance metric on each dataset, indicating that our Graph-Graph Similarity Network is more effective than other distance metrics in the graph classification problem. Our method is a learned non-linear transformation, which is more expressive and flexible.

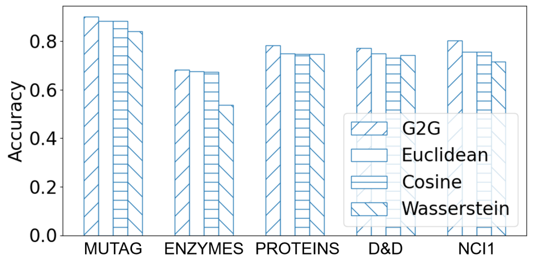

Figure 4: Distance metrics for graph similarity.

5 Conclusion

In this paper, we addressed the graph classification problem and presented a Graph-Graph model, which explored the relationship between graphs. Specifically, we first employed Graph Convolutional Network and Adversarial Autoencoder to learn effective graph-level representations under a common representation space. Next, a Graph-Graph Similarity Network was designed to learn the similarity between graphs by zero-one loss and triplet loss, and a Supergraph was built upon it, which transformed graph classification to label propagation on the Supergraph. Finally, we adopted the K-Nearest Neighbors algorithm to get the predictions. The experiments on public datasets validated that our method performed better than other baseline models, and the Graph-Graph Similarity Network as well as the SuperGraph performed positive impacts on tackling the graph classification tasks.

REFERENCES

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *ACM International Conference on Web Search and Data Mining*, pp. 384–392, 2019.

Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *IEEE International Conference on Data Mining*, pp. 8–pp, 2005.

Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21:47–56, 2005.

Thang D Bui, Sujith Ravi, and Vivek Ramavajjala. Neural graph learning: Training neural networks using graphs. In *ACM International Conference on Web Search and Data Mining*, pp. 64–71, 2018.

Gavin C Cawley and Nicola LC Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(Jul):2079–2107, 2010.

Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

Richard O Duda and Peter E Hart. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya, and Tomaso A Poggio. Learning with a wasserstein loss. In *Advances in Neural Information Processing Systems*, pp. 2053–2061, 2015.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pp. 1735–1742, 2006.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017a.

William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

Da-Cheng Juan, Chun-Ta Lu, Zhen Li, Futang Peng, Aleksei Timofeev, Yi-Ting Chen, Yaxi Gao, Tom Duerig, Andrew Tomkins, and Sujith Ravi. Graph-rise: Graph-regularized image semantic embedding. *arXiv preprint arXiv:1902.10814*, 2019.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

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.

Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483*, 2012.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10):1995, 1995.

Lam Hong Lee, Chin Heng Wan, Rajprasad Rajkumar, and Dino Isa. An enhanced support vector machine classification framework by using euclidean distance function for text document categorization. *Applied Intelligence*, 37(1):80–99, 2012.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. *arXiv preprint arXiv:1904.12787*, 2019.

Daryl Lim and Gert Lanckriet. Efficient learning of mahalanobis metrics for ranking. In *International Conference on Machine Learning*, pp. 1980–1988, 2014.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Brian McFee and Gert R Lanckriet. Metric learning to rank. In *International Conference on Machine Learning*, pp. 775–782, 2010.

Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. *arXiv preprint arXiv:1802.04407*, 2018.

Zhaoqing Pan, Weijie Yu, Xiaokai Yi, Asifullah Khan, Feng Yuan, and Yuhui Zheng. Recent progress on generative adversarial networks (gans): A survey. *IEEE Access*, 7:36322–36333, 2019.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic Acids Research*, 32(suppl_1):D431–D433, 2004.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 815–823, 2015.

Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation. In *AAAI Conference on Artificial Intelligence*, 2018.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77): 2539–2561, 2011.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3693–3702, 2017.

Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

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

S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

Donglin Wang, Ting Wang, Feng Zhao, and Xuetao Zhang. Improved graph-based semi-supervised learning for fingerprint-based indoor localization. In *IEEE Global Communications Conference*, pp. 1–6, 2018.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

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*, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pp. 4800–4810, 2018.

Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE Transactions on Big Data*, 2018a.

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

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.