Customized Graph Nerual Networks

Yiqi Wang[1*], Yao Ma[2*], Wei Jin[1], Chaozhuo Li[3], Charu Aggarwal[4], Jiliang Tang[1]
[1] Michigan State University
[2] New Jersey Institute of Technology
[3] Microsoft Research Asia
[4] IBM T. J. Watson Research Center
{wangy206,mayao4,jinwei2,tangjili}@msu.edu, {cli}@microsoft.com, {charu}@us.ibm.com

Abstract

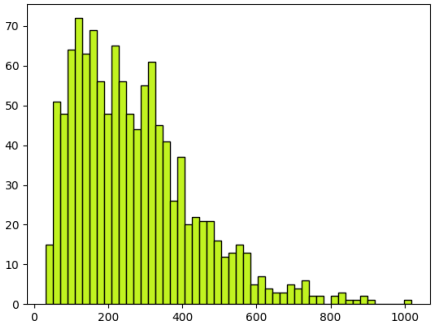
Recently, Graph Neural Networks (GNNs) have greatly advanced the task of graph classification. Typically, we first build a unified GNN model with graphs in a given training set and then use this unified model to predict labels of all the unseen graphs in the test set. However, graphs in the same dataset often have dramatically distinct structures, which indicates that a unified model may be sub-optimal given an individual graph. Therefore, in this paper, we aim to develop customized graph neural networks for graph classification. Specifically, we propose a novel customized graph neural network framework, i.e., Customized-GNN. Given a graph sample, Customized-GNN can generate a sample-specific model for this graph based on its structure. Meanwhile, the proposed framework is very general that can be applied to numerous existing graph neural network models. Comprehensive experiments on various graph classification benchmarks demonstrate the effectiveness of the proposed framework.

1 Introduction

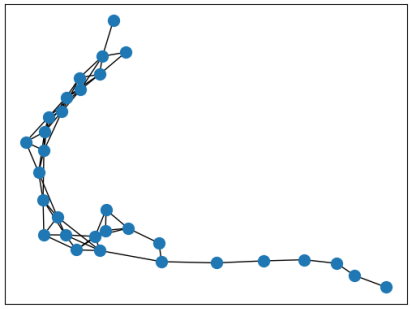
Graphs are natural representations for many real-world data such as social networks [1, 2, 3, 4], biological networks [5, 6] and chemical molecules [7, 8]. A crucial step to perform downstream tasks on graph data is to learn better representations. Deep neural networks have demonstrated great capabilities in representation learning for Euclidean data and thus have advanced numerous fields including speech recognition [9], computer vision [10] and natural language processing [11]. However, they cannot be directly applied to graph-structured data since graphs have complex topological structures. Recently, graph neural networks (GNNs) have generalized deep neural networks to graph data. GNNs typically update node representations by transforming, propagating and aggregating node features across the graph. They have boosted the performance of many graph related tasks such as node classification [3, 2], link prediction [12, 13, 14], and graph classification [15, 16, 17, 18].

Graph classification is one of the most important and prevalent graph related tasks [19], and in this work, we aim to advance graph neural networks for the graph classification task. There are numerous real-world applications for graph classification. For example, proteins can be denoted as graphs [20] and the task to infer whether a protein functions as an enzyme or not can be regarded as a graph classification task; and it can also be applied to forecast Alzheimer's disease progression in which individual brains are represented as graphs [21]. Unlike data samples in classification tasks in other domains such as computer vision [22] and natural language processing [23], graph samples in the graph classification task are described not only by the input (node) features but their graph structures. Both the input node features and the graph structures play crucial roles in the graph classification tasks [15, 16, 18].

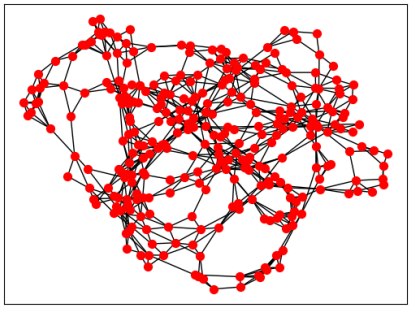
In reality, graphs in the same data set can present significantly different structural properties. Figure 1a demonstrates the distribution of graph size (i.e., the number of nodes) for protein graphs in the D&D dataset [20], where the graph size varies dramatically from 30 to $5,748$. We further illustrate two graphs sampled from the



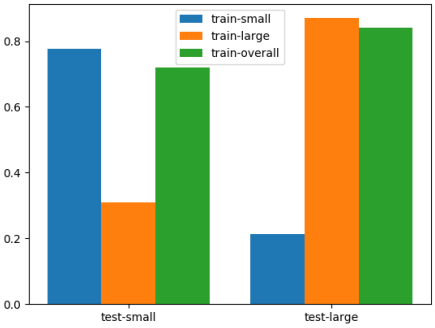
(a) Node size distribution



(b) A graph with 31 nodes



(c) A graph with 302 nodes



(d) Classification accuracy

Figure 1: An illustrative example of varied structural information and its impact on the performance of graph neural network based graph classification. (a) demonstrates the distribution of graph size (i.e., the number of nodes) for protein graphs in the D&D dataset, where the graph size varies dramatically from 30 to 5, 748. ; (b) and (c) show two graphs sampled from the D&D dataset, which present very different structural properties; (d) shows classification performance of three models trained on training sets with various graph sizes.

D&D dataset in Figures 1b and 1c, respectively. These two graphs present very different structural properties such as the number of nodes, graph shapes, and diameters.

The above investigations indicate that graphs in the same data set could have dramatically distinct structural properties. It naturally raises the question – whether we should treat these graphs differently? To investigate this question, we take the graph size as the representative structure-property and demonstrate how it affects the graph classification performance. Specifically, we divide graphs from D&D into two groups based on their graph size – one for graphs with a small number of nodes and the other for graphs with a large number of nodes. Then, we split each group into a training set and a test set. Next, we train three GNN models [1] based on three training sets – the small one, the large one and the overall (a combination of the small and large one), separately. Then, we test their performance on the two test sets. The results are shown in the Figure 1d. In the test set with small/large graph sizes, the model trained on the training set with the same graph size can significantly outperform the other two models.

Investigations and consistent observations on more settings can be found in the Section "Preliminary Data Analysis". These investigations suggest that a unified model is not optimal for graphs with diverse structure properties and efforts are desired to consider the structure-property difference among graphs. Hence, in this paper, we aim to learn "customized models" for graphs with different structural properties. A natural way is to divide the dataset into different splits according to structure properties and train a model for each split. However, we face enormous challenges to achieve this goal in practice. First, there are potentially several structure properties (graph size, density, and etc.) affecting the performance, and we have no explicit knowledge about how the graphs should be split according to these properties. Second, dividing the dataset leads to small training sets for the splits, which may not be sufficient to train satisfactory models. To address these challenges, we propose a novel graph neural network framework, Customized-GNN, for graph classification. The Customized-GNN framework is trained on all graphs in the given training set (without splitting) and able to produce customized GNN models for each individual graph. Specifically, we design an adaptor, which is able to smoothly adjust a general GNN model to a specific one according to the structural properties of a given graph. The general GNN model and the adaptor are learned during the training stage simultaneously utilizing all graphs.

Our major contributions are listed as follows: 1) We empirically observed that graphs in a given dataset could have dramatically distinct structural properties. Furthermore, it is not optimal to train a unified model for graphs with various structure properties for a graph classification task; 2) We propose a framework, Customized-GNN, which is able to generate a customized GNN model for each graph sample based on its structural properties. The proposed framework is general and can be directly applied to many existing graph neural network models; 3) We designed and conducted comprehensive experiments on numerous graph datasets from various domains to verify the effectiveness of the proposed framework.

2 Related Work

Graph Neural Networks have recently drawn great interest due to its strong representation capacity in graph-structured data in many real-world applications. Generally, graph neural networks can be divided into two categories: the spectral approaches and the non-spectral approaches. The spectral methods aim at defining the parameterized filters based on graph spectral theory by using graph Fourier transform and graph Laplacian [31, 32, 33, 34], and the non-spectral methods aim at defining parameterized filters based on nodes' spatial relations by aggregating information from neighboring nodes directly [2, 35].

Graph neural networks have advanced a wide variety of tasks including node classification [3, 2], link prediction [36, 12, 13] and graph classification [15, 16]. In the task of graph classification, one of the most important step is to get a good graph-level representation. A straight-forward way is to directly summarize the graph representation by globally combining the node representations [37]. Recently, there are some works investigating

[1]The GNN model for graph classification uses GCN [3] as the filtering operation and maxpooling as the pooling operation.

learning hierarchical graph representations by leveraging deterministic graph clustering algorithms [32, 38]. There also exist end-to-end models aiming at learning hierarchical graph representations, such as DiffPool [15]. MuchGNN [39] proposed to learn a set of graph channels at each layer to shrink the graph hierarchically. Furthermore, some methods [13, 40, 41] propose principles to select the most important k nodes to form a coarsened graph in each network layer. EigenPooling [16] is based on graph Fourier transform and is able to capture the local structural information. In [26], conditional random fields (CRF) are used to design the pooling operation.

3 Preliminary Data Analysis

Table 1: Graph classification accuracy on different node-size sets

	D&D		**ENZ**		**PROT**		**RE-BI**	
Accuracy (%)	S-test	L-test	S-test	L-test	S-test	L-test	S-test	L-test
S-training	66.2	55.7	45.0	20.0	76.5	51.4	88.6	46.6
L-training	47.2	77.8	27.0	39.0	45.6	78.6	29.3	83.1

Table 2: The classification accuracy of the models trained from four training sets in D&D dataset and Statistics for four training sets.

Training set	Node size range	#Graphs	Accuracy			
			test 1	test 2	test 3	test
training 1	[0,200]	369	76.1	41.2	26.7	50.9
training 2	[200,400]	392	43.5	75.3	82.2	62.4
training 3	[400,2000]	180	23.9	75.3	88.9	56.8
training	[0,2000]	941	64.1	61.9	75.6	66.2

In Figure 1, we have demonstrated that graphs in D&D have varied properties, which affected the performance of GNNs for graph classification. In this section, we aim to further investigate this phenomenon by answering the following two questions – (1) can the observations on D&D be extended to other datasets? and (2) whether incorporating these properties into the models can facilitate the performance?

We choose four representative graph datasets from different domains for this study including **D&D** [20], **ENZ** [6], **PROT** [5] and **RE-BI** [1]. We checked the properties such as node size and edge size. Similar to D&D, graphs in all datasets present very diverse properties. More details about these datasets can be found in Section 5. Following the same setting as D&D, we divide each data into two groups according to the node size, i.e., large training and test (denoted as “L-training" and “L-test") and small training and test (indicated as “S-training" and “S-test"). The results are demonstrated in Table 1. From the table, we make consistent observations with these in D&D – models trained on one property group (e.g., L-training) cannot perform well on the other property group (e.g., S-test).

To answer the second question, we divide D&D into several subsets based on the node size, and then divide each subset into a sub-training set (80%) and a sub-test set (20%). We train models on different sub-training sets separately, and then test their performance on all the sub-test sets. Specifically, we have trained four models on

four different training sets from D&D, which are *training 1*, *training 2*, *training 3* containing graph samples with node sizes from different ranges, and *training* which is the combination of *training 1*, *2* and *3*. Then, we test four models on four test sets, i.e., *test 1*, *test 2*, *test 3* and *test* which is the combination of *test 1*, *2* and *3*. Statistics about these training sets are summarized in Table 2.

The performance of four models on the test sets are illustrated in Table 2. We note that the model trained on a specific training set performs much better on the corresponding test set that shares the same node size range than the other test sets. This suggests the potential to incorporate the structure properties into the model training. In addition, though *training 1*, *training 2* and *training 3* have much fewer training samples, the models trained on specific training sets can achieve better performance on the corresponding test sets compared to these trained from the entire training set (or *training*). This indicates that a unified graph neural network that is trained from the entire training set is not optimal for graphs with various structure properties in the test set.

Discussion. Via the preliminary data analysis, we have established: (1) graphs in real-world data present distinct structure properties that tend to impact the graph classification performance of GNNs; and (2) incorporating the difference has the potential to boost the graph classification performance. These observations lay the foundations of the model design in the next section.

4 The Proposed Framework

In this section, we introduce the proposed framework Customized-GNN that has been designed for graphs with inherently distinct structure properties.

4.1 The Overall Design

As mentioned in earlier sections, graphs in real-world data inherently present distinct structural properties. Thus, we are desired to build distinct GNN models for them. To achieve this goal, we face tremendous challenges. First, we have no explicit knowledge about how the graph structure properties will influence graph neural network models. Second, if we separately train different models for graphs with different structure properties, we have to split the training sets for each model; as a consequence, the training data for each model could be very limited. For example, in the extreme case where each graph has unique graph structural properties, we only have one training sample for the corresponding model. Third, even if we can well train distinct GNN models for different graphs, during the test stage, for an unlabelled graph with unseen structural property, it is hard to decide which trained model we should adopt to make the prediction. In this work, we propose a customized graph neural network framework, i.e., Customized-GNN, which can tackle the aforementioned challenges simultaneously.

An overview of the architecture of Customized-GNN is demonstrated in Figure 2. The basic idea of Customized-GNN is – it generates customized adaptor parameters for each graph sample g_i via an adaptor network with the graph structure properties as input. These generated adaptor parameters are used to adapt a shared GNN model denoted as GNN (this could be any GNN model that works for the graph classification task) to a model specific for the graph sample g_i. The adapted model GNN_i incorporates the structure information of graph g_i, and thus, is customized for the graph sample g_i.

With the proposed Customized-GNN framework, the first challenge is handled, since the influence is implicitly modeled by the adaptor networks, which can customize the shared GNN model to a graph sample specific one. Furthermore, Customized-GNN can be trained on the entire training set without splitting it according to graphs' structure properties. This not only solves the second challenge but also ensures that the trained model can preserve common knowledge from the entire training set. The third challenge is also automatically addressed by the Customized-GNN framework. Given an unseen graph g_j, the Customized-GNN framework first takes its graph structure information as input and generates adaptor parameters. Then, these generated adaptor parameters can be used to customize the general GNN model to a customized one GNN_j to predict the label of g_j.

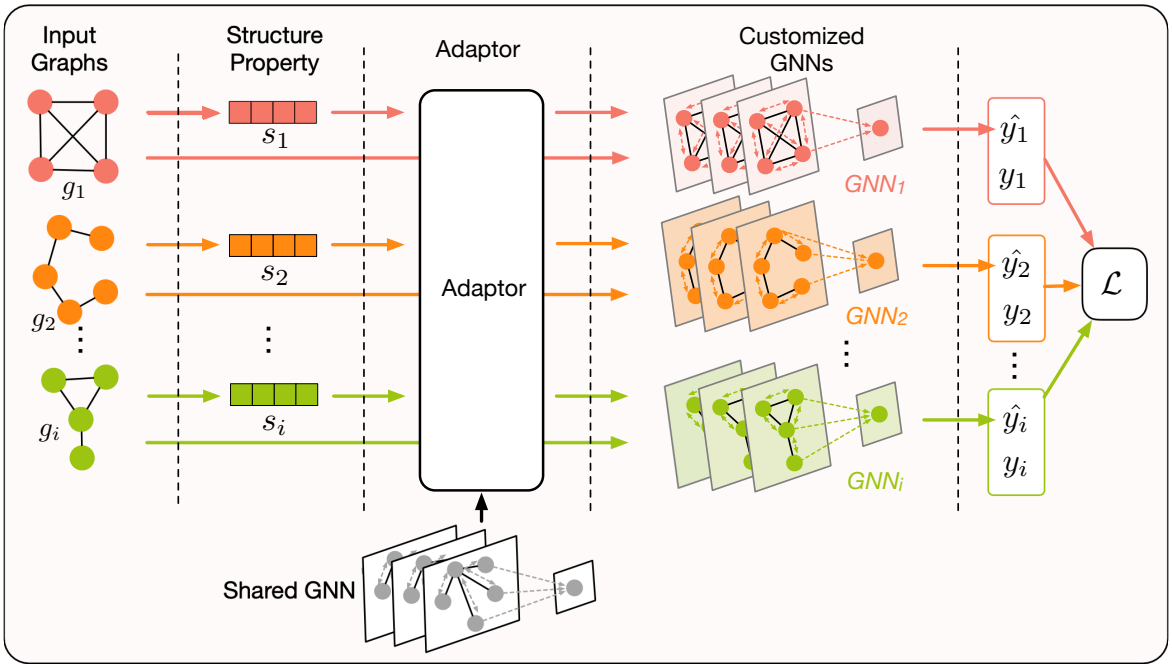


Figure 2: An overview of the proposed customized graph neural networks. Given a set of graph samples, the adaptor networks take their structure properties as input, and generate corresponding adaptor parameters, which are used to adapt a shared GNN model (this could be any GNN model that works for the graph classification task) to a model specific for each graph sample. Each adapted model incorporates the structure information a graph, and thus, is customized for this graph sample to make label prediction. With the predicted label and the ground truth label, we can calculate the overall loss, which is used to guide the optimization of the adaptor networks and the shared GNN model.

4.2 The Adapted Graph Neural Network

Next, we introduce details about the adaptor network, the process of adapting a shared model to a specific one for a given graph, and the time complexity analysis of the proposed framework.

4.3 The Adaptor Network

The goal of the adaptor network is to generate the adaptor parameters for a given graph. From the preliminary data analysis, we have the intuition that the customized GNN model for a specific graph sample should be correlated to its structural properties. However, there is no explicit knowledge about how these structural properties influence graph neural network models. To model this implicit mapping function, we propose to utilize a powerful neural network to generate the model adaptor parameters from the observed structure information of a given sample.

In addition, graph neural networks often consist of several subsequent filtering and pooling layers, which can be viewed as different GNN blocks. For example, K GNN blocks are shown in Figure 3. The graph structure properties of a given sample may have different influences on different GNN blocks. Hence, for each GNN block, we introduce one adaptor network to generate adaptor parameters for each block.

Specifically, we first extract a vector $\mathbf{s}_i$ to denote the structure information of a given graph g_i. We will discuss more details about $\mathbf{s}_i$ in the experiment section. As shown in the Figure 3, the adaptor networks take the structure information $\mathbf{s}_i$ as input and generate the adaptation parameters for each block. In the case where there are K blocks in the graph neural network, we have K independent adaptor networks corresponding to the K

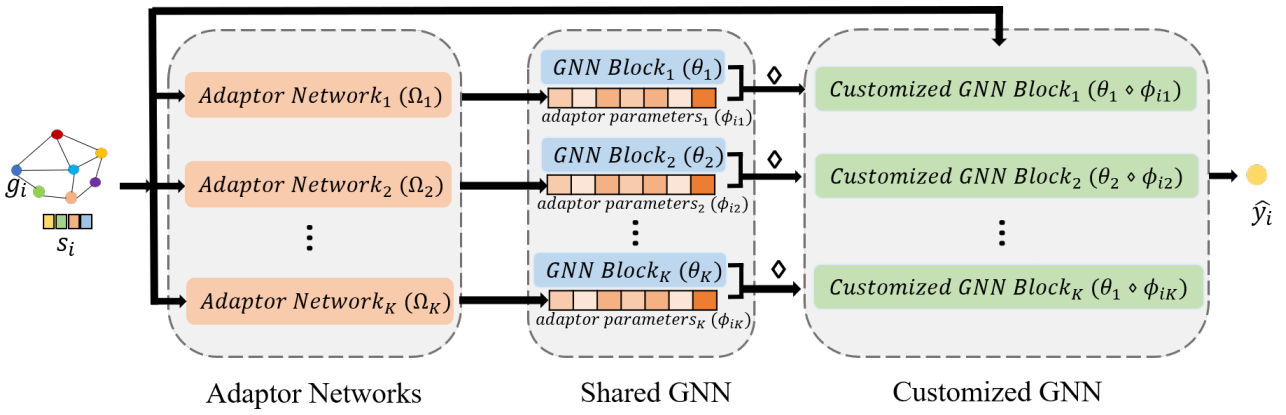


Figure 3: An overview of the GNN adaptation process. Given a specific graph sample g_i, the adaptor networks consisting of K blocks take its graph structure properties s_i as input, and generate corresponding adaptor parameters to adapt each shared GNN block to get a customized GNN for g_i. This customized GNN is then used to make prediction for g_i.

blocks. Note that these adaptor networks share the same input $\mathbf{s}_i$ while their outputs are different. Specifically, the adaptor network for the j-th block can be expressed as follows:

$$\boldsymbol{\phi}_{ij} = h_j(\boldsymbol{s}_i; \Omega_j), j = 1, \ldots, K, \tag{29}$$

where Ω_j denotes the parameters of the j-th adaptor network and $\boldsymbol{\phi}_{ij}$ denotes its output, which will be used to adapt the j-th learning block. The adaptor network h_j can be modeled using any functions. In this work, we utilize feed-forward neural networks due to their strong capability. According to the universal approximation theorem [24], a feed-forward neural network can approximate any nonlinear functions. For convenience, we summarize the process of the K adaptor networks with s_i as input below:

$$\Phi_i = H(\mathbf{s}_i; \boldsymbol{\Omega}_H), \tag{30}$$

where Φ_i contains the generated adaptation parameters of all the GNN blocks for graph g_i and $\boldsymbol{\Omega}_H$ denotes the parameters of the K adaptor networks.

Any existing graph neural network model can be adapted by the Customized-GNN framework to generate sample-specific models based on the structure information. Therefore, we first generally introduce the GNN model for graph classification and describe how to adapt it given a specific sample. Then, we illustrate how to adapt specific GNN models.

4.3.1 A General Adapted Framework

A typical GNN framework for graph classification contains two types of layers, i.e., the filtering layer and the pooling layer. The filtering layer takes the graph structure and node representations as input and generates refined node representations as output. The pooling layer takes graph structure and node representations as input to produce a coarsened graph with a new graph and new node representations. A general GNN framework for graph classification contains K_p pooling layers, each of which follows K_f stacking filtering layers. Hence, there are $K = K_p * K_f$ learning blocks in the GNN framework. A graph-level representation can be obtained from these layers that can be further utilized to perform the prediction. Given a graph sample g_j, we need to adapt each of the K layers according to its adaptor parameters generated from the adaptor network. Via this process, we can generate a GNN model GNN_j specific to g_j.

Without loss of generality, when introducing a filtering layer or a pooling layer, we use an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and node representations $\mathbf{X} \in \mathbb{R}^{n \times d}$ to denote the input of these layers where n is the number of nodes and d is the dimension of node features. Then, the operation of a filtering layer can be described as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f) \tag{31}$$

where θ_f denotes the parameters in the filtering layer and $\mathbf{X}_{new} \in \mathbb{R}^{n \times d_{new}}$ denotes the refined node representations with dimension d_{new} generated by the filtering layer. Assuming ϕ_f is the corresponding adaptor parameters for this filtering layer, we adapt the model parameter θ_f of this filtering layer as follows:

$$\theta_f^m = \theta_f \diamond \phi_f, \tag{32}$$

where θ_f^m is the adapted model parameter that has the same dimension as the original model parameter θ_f; and $\diamond$ is the adaptation operator. The adaption operator can have various designs, which can be determined according to the specific GNN model. We will provide the details of the adaptation operator when we introduce concrete examples in the following subsections. Then, with the adapted model parameters, we can define the adapted filtering layer as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f \diamond \phi_f). \tag{33}$$

On the other hand, the process of a pooling layer can be described as follows:

$$\mathbf{A}_{new}, \mathbf{X}_{new} = p(\mathbf{A}, \mathbf{X}; \theta_p), \tag{34}$$

where θ_p denotes the parameters of the pooling layer, $\mathbf{A}_{new} \in \mathbb{R}^{n_{new} \times n_{new}}$ with $n_{new} < n$ is the adjacency matrix for the newly generated coarsened graph and $\mathbf{X}_{new} \in \mathbb{R}^{n_{new} \times d_{new}}$ is the learned node representations for the coarsened graph. Similarly, we adapt the model parameters of the pooling layer as follows:

$$\theta_p^m = \theta_p \diamond \phi_p, \tag{35}$$

which leads to the following adapted pooling layer:

$$\mathbf{A}_{new}, \mathbf{X}_{new} = p(\mathbf{A}, \mathbf{X}; \theta_p \diamond \phi_p), \tag{36}$$

where ϕ_p is the adaptation parameters generated by the adaptor network for this pooling layer.

For convenience, we summarize a general GNN model as $GNN(\cdot \mid \Theta_{GNN})$, where Θ_{GNN} is the parameters in all GNN blocks(i.e., θ_f, θ_p in all filtering and pooling layers). Then, for a graph sample g_i, we can adapt the GNN model $GNN(\cdot \mid \Theta_{GNN})$ to a customized model for g_i denoted as $GNN(\cdot \mid \Theta_{GNN} \lozenge \Phi_i)$. Note that, as shown in Eq. equation 30, Φ_i contains adaptation parameters of all GNN blocks for a graph sample g_i. The adaptation operations in all GNN blocks (including filtering and pooling layers) are summarized in $\Theta_{GNN} \lozenge \Phi_i$. There are numerous GNN models designed for graph classification [13, 25, 16, 26]. The proposed framework can be applied to the majority of these models, i.e., these models all can serve as the $GNN(\cdot \mid \Theta_{GNN})$ model mentioned above. In this work, we focus on three representative GNN models including GCN [3], DiffPool [15] and gPool [13]. We would like to leave the investigations of other GNN models as one future work. Next, we will give details on how to adapt GCN and DiffPool since gPool follows a similar adaptation process.

4.3.2 Adapted GCN: Customized-GCN

Graph Convolutional Network (GCN) [3] is originally proposed for semi-supervised node classification task. The filtering layer in GCN is defined as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f) = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X} \mathbf{W}), \tag{37}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ represents the adjacency matrix with self-loops, $\tilde{\mathbf{D}} = \sum_j \tilde{\mathbf{A}}_{ij}$ is the diagonal degree matrix of $\tilde{\mathbf{A}}$ and $\mathbf{W} \in \mathbb{R}^{d \times d_{new}}$ denotes the trainable weight matrix in filtering layer and $\sigma(\cdot)$ is a nonlinear activation function. With the adaptation parameter ϕ_f for this corresponding filtering layer, the adapted filtering layer can be represented as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f \diamond \phi_f) = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X} (\mathbf{W} \diamond \phi_f)). \tag{38}$$

Specifically, we adopt FiLM [27] as the adaption operator. In this case, the dimension of the adaptor parameter is $2d$, i.e., $\phi_f \in \mathbb{R}^{2d}$. We split ϕ_f into two parts $\gamma_f \in \mathbb{R}^d$ and $\beta_f \in \mathbb{R}^d$ and then the adaptation operation can be expressed as follows:

$$\mathbf{W} \diamond \phi_f = (\mathbf{W} \odot br(\gamma_f, d_{new})) + br(\beta_f, d_{new}), \tag{39}$$

where $br(\mathbf{a}, k)$ is a broadcasting function that repeats k times for the vector $\mathbf{a}$; hence, $br(\gamma_f, d_{new}) \in \mathbb{R}^{d \times d_{new}}$ and $br(\beta_f, d_{new}) \in \mathbb{R}^{d \times d_{new}}$ have the same shape as $\mathbf{W}$ and $\odot$ denotes the element-wise multiplication between two matrices.

To utilize GCN for graph classification, we introduce a node-wise max pooling layer to generate graph representation from the node representations as follows:

$$\mathbf{x}_G = p(\mathbf{A}, \mathbf{X}; \theta_p) = max(\mathbf{X}), \tag{40}$$

where $\mathbf{x}_G \in \mathbb{R}^{1 \times d_{new}}$ denotes the graph-level representation and $max()$ takes the maximum over all the nodes. Note that the max-pooling operation does not involve learnable parameters and thus no adaptation is needed for it. We refer to an adapted GCN framework as Customized-GCN.

4.3.3 Adapted diffpool: Customized-DiffPool

DiffPool is a hierarchical graph level representation learning method for graph classification [15]. The filtering layer in DiffPool is the same as Eq. equation 37 and its corresponding adapted version is shown in Eq. equation 38. Its pooling layer is defined as follows:

$$\mathbf{S} = softmax(f_a(\mathbf{A}, \mathbf{X}; \theta_{f_a})), \tag{41}$$

$$\mathbf{X}_{new} = \mathbf{S}^T \mathbf{Z}, \tag{42}$$

$$\mathbf{A}_{new} = \mathbf{S}^T \mathbf{A} \mathbf{S}, \tag{43}$$

where f_a is a filtering layer embedded in the pooling layer, $\mathbf{S} \in \mathbb{R}^{n \times n_{new}}$ is a soft-assignment matrix, which softly assigns each node into a supernode to generate a coarsened graph. Specifically, the structure and the node representations for the coarsened graph are generated by Eq. equation 43 and Eq. equation 42 respectively, where $\mathbf{Z} \in \mathbb{R}^{n \times d_{new}}$ is the output of the filtering layers. To adapt the pooling layer, we only need to adapt Eq. equation 41, which follows the same way as introduced in Eq. equation 38 as it is also a filtering layer. We refer to the adapted diffpool model as Customized-DiffPool.

4.4 Training and Test via the Customized Framework

Given a graph sample g_i with the adjacency matrix $\mathbf{A}_i$, and the feature matrix $\mathbf{X}_i$, the Customized-GNN framework performs the classification task as follows:

$$\tilde{y_i} = GNN(\boldsymbol{A}_i, \mathbf{X}_i; \boldsymbol{\Theta}_{\mathcal{GNN}} \lozenge H(\boldsymbol{s}_i; \boldsymbol{\Omega}_H)). \tag{44}$$

During the training, we are given a set $\mathcal{G} = \{g_i, y_i\}$ of N graphs as training samples, where each graph g_i is associated with a ground truth label y_i. Then, the objective function of Customized-GNN can be represented as follows:

$$\min_{\boldsymbol{\Omega}_H, \boldsymbol{\theta}_{\mathcal{GNN}}} \sum_{i=1}^{N} \mathcal{L}(y_i, GNN(\boldsymbol{A}_i, \mathbf{X}_i; \boldsymbol{\Theta}_{\mathcal{GNN}} \diamondsuit H(\boldsymbol{s}_i; \boldsymbol{\Omega}_H))), \tag{45}$$

where N is the number of training samples and $\mathcal{L}$ is a loss function. In this work, we use Cross-Entropy as the loss function and adopt ADAM [28] to optimize the objective.

During the test phase, the label of a given sample g_ℓ can be inferred using equation 44. Specifically, the graph structure information $\mathbf{s}_\ell$ of the sample is first utilized as the input of the adaptor network $H(\cdot; \boldsymbol{\Omega})$ to identify its distribution information, which is then utilized to adapt the shared model parameter Θ_{GNN} to generate a sample-specific model GNN_ℓ. This sample-specific model finally performs the classification for this sample.

Table 3: The statistics of seven datasets. #Graphs denotes the number of graphs. #Class is the number of graph classes. Node size indicates range, average and standard deviation of the number of nodes among the graphs. Edge size represents range, average and standard deviation of the number of edges among the graphs.

Dataset	#Graphs	#Class	Node size			Edge size		
			range	mean	std	range	mean	std
COLLAB	5000	3	[32, 492]	74.5	62.3	[60, 40120]	2457.8	6439.0
ENZ	600	6	[2, 125]	32.6	14.9	[1, 149]	62.14	25.5
PROT	1113	2	[4, 620]	39.1	45.7	[5, 1049]	72.82	84.6
D&D	1178	2	[30, 5748]	284.3	272.0	[63, 14267]	715.65	693.9
RE-BI	2000	2	[63, 782]	429.6	554.0	[4, 4071]	497.8	623.0
RE-5K	4999	5	[22, 3648]	508.5	452.6	[21, 4783]	594.9	566.8
NCI109	4127	2	[4, 111]	29.6	13.6	[3, 119]	32.1	15.0

4.5 Time Complexity Analysis

In this subsection, we analyze the additional time required to calculate the adaptation parameters and perform the adaptation. Specifically, we use the FiLM adaptation operator, as an example for the adaptor network. For convenience, the dimension of the output node features in all layers is assumed to be the same d. The dimension of the output of the adaptation network ϕ_f is $2d$. Furthermore, we assume that the input of the adaptation network, i.e., the graph property information $\mathbf{s}_i$ is with dimension s. Then, the time complexity to generate the adaptation parameters for a single block using Eq. equation 29 is $O(2d \cdot s) = O(d \cdot s)$. Furthermore, the time required to adapt the parameters for a single block with Eq. equation 39 is $O(d^2)$. Hence, for graph neural networks with K learning blocks, the time complexity to calculate the adaptation parameters and perform the adaptation for all learning blocks is $O(K \cdot d \cdot s + K \cdot d^2)$. Note that, the time complexity of a single filtering operation in Eq. equation 37 is $O(m \cdot d + n \cdot d^2)$ where m denotes the number of edges while n is the number of nodes. Therefore, the total time complexity for K learning blocks without adaptation is $O(K \cdot m \cdot d + K \cdot n \cdot d^2)$. Furthermore, s is typically small (much smaller than m); hence, the additional time complexity introduced by the adaptation operation is rather small.

5 Experiments

In this section, we conducted comprehensive experiments to verify the effectiveness of the proposed Customized-GNN framework. We first describe the experimental settings. Then, we evaluate the performance of the

framework by comparing original GCN, DiffPool and gPool with the adapted GCN, DiffPool, gPool models by the Customized-GNN framework. Next, we analyze the importance of different components in the adaptor operator. Finally, we conduct case studies to further facilitate our understanding of the proposed method.

5.1 Experimental Settings

We carried out graph classification tasks on seven datasets. Some key statistics of these datasets used in our experiments are shown in Table 3, and more details of them are introduced as follows:

- **COLLAB** [1] is a dataset of scientific collaboration networks, which describes collaboration pattern of researchers from three different research fields.
- **ENZ** [6] is a dataset of protein tertiary structures of six classes of enzymes.
- **PROT** [5] is a dataset of protein structures, where each graph represents a protein and each node represents a secondary structure element (SSE) in the protein.
- **D&D** [20] is a dataset of protein structures. Each protein is represented as a graph, where each node in a graph represents an amino aid and each edge between two nodes denotes that they are less than 6 Ångstroms apart.
- **RE-BI** and **RE-5K** [1] are two datasets of online discussion threads crawled from different subreddits in Reddit, where each node represents an user and each edge between two users represents their interaction.
- **NCI109** [6] is a dataset of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, which are provided by Natinal Cancer Institue (NCI).

Next, we describe the baselines. In the Section "The Proposed Framework" , we apply the proposed framework to adapt three graph neural networks models: a basic graph convolutional network (GCN) [3], and two SOTA graph classification models DiffPool [15] and gPool [13], respectively. The corresponding adapted versions are Customized-GCN, Customized-DiffPool and Customized-gPool, respectively. *Our evaluation purpose is if the proposed framework can boost the performance of existing models by adapting them to their corresponding customized versions.* Thus, (1) to validate the effectiveness of the proposed model, we compare Customized-GCN, Customized-DiffPool, Customized-gPool with GCN, DiffPool and gPool; and (2) we have not chosen models in [25, 16, 26] as baselines here but the proposed framework can be directly applied to adapt them as well. Note that in this work, we construct a set of simple structural features $\mathbf{s}_i$ of g_i such as the number of nodes, the number of edges and the graph density; however, it is flexible to include other complex features by the proposed framework. Furthermore, we create baselines to directly concatenate the graph structure properties $\mathbf{s}_i$ to the output graph embedding of the GCN, DiffPool and gPool model. Correspondingly, we call these three methods as Concat-GCN, Concat-Diff and Concat-gPool. In addition, we develop baseline methods, Multi-GCN, Multi-Diff and Multi-gPool. They learn multiple graph convolutional networks for graph samples with different structural information. More details of these baselines are as follows:

- **GCN** [3] is originally proposed for semi-supervised node classification. It consists of a stack of GCN layers, where a new representation of each node is computed via transforming and aggregating node representations of its neighbouring nodes. Finally, a graph representation is generated from node representations in the last GCN layer via a global max-pooling layer, and then used for graph classification.
- **Diffpool** [15] is a recently proposed method which has achieved state-of-the-art performance on the graph classification task. It proposes a differentiable graph pooling approach to hierarchically generate a graph-level representation by coarsening the input graph level by level.

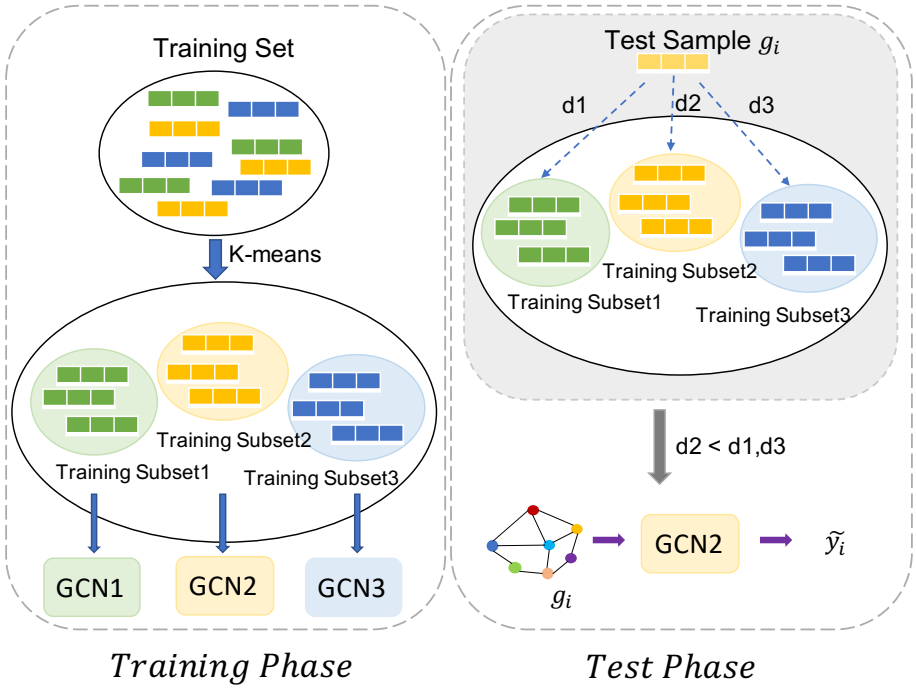


Figure 4: The framework of Multi-GCN with 3 clusters. we group training samples into 3 clusters, and train a GCN model for each cluster. Given a test sample g_i, we measure the distance between this sample and the centroids of the three clusters and utilize the corresponding model of the closest cluster to perform the prediction for this sample.

- **gpool** [13] is a newly proposed pooling method that has achieved state-of-the-art performance on the graph classification task. It develops a U-Net-like architecture for graph dat, consisting of graph pooling operation and unpooling operation based on node importance value.
- **Concat-GCN (or Concat-Diff, Concat-gpool)** is baseline method that directly concatenates the graph structure properties $\mathbf{s}_i$ to the output graph embedding of the GCN, DiffPool and gPool model.
- **Multi-GCN (or Multi-Diff, Multi-gpool)** consists of several GCN (or Diffpool, gpool) models trained from different subsets of the training datset. As shown in Figure 4, we first cluster data samples from training set into different training subsets via K-means method based on the graph structural information. Note that in this work, the structural information $\mathbf{s}_i$ of g_i includes the number of nodes, the number of edges and the graph density. Then train different models are trained from different training subsets. During the test phase, given a test graph sample, we first compute the euclidean distance between its graph structural properties and the centroids of different training subsets. Then, the model trained on the closest training subset is selected to do label prediction for this graph sample. In this experiment, we set the number of clusters to 2 and 3, and denote the corresponding frameworks as Multi-GCN-2 (or Multi-Diff-2, Multi-gpool-2) and Multi-GCN-3 (or Multi-Diff-3, Multi-gpool-3).

5.2 Graph Classification Performance Comparison

In this subsection, we first perform the comparison following the traditional setting. To further demonstrate the advantage of the proposed frameworks, we show their adaptability when the properties of test graphs are

Table 4: Comparisons of graph classification performance in terms of accuracy.

Accuracy (%)	Datasets						
	COLLAB	**ENZ**	**PROT**	**DD**	**RE-BI**	**RE-5K**	**NCI109**
GCN	67.9±1.4	50.4±3.0	77.0±2.3	79.3±5.3	82.6±4.9	50.7±1.3	74.9±2.7
Concat-GCN	68.4±1.4	52.5±5.1	78.4±1.9	77.6±3.2	80.7±3.5	50.7±1.0	75.6±1.2
Multi-GCN-2	68.3±1.4	47.0±1.8	79.5±1.3	77.1±2.4	80.6±3.5	50.3±1.9	74.2±1.9
Multi-GCN-3	67.0±1.4	44.6±5.4	79.9±2.2	76.7±3.5	77.5±7.3	48.5±2.1	75.8 ±1.5
Customized-GCN	71.3±1.0	55.4±4.6	78.8±3.2	79.6±3.9	91.5±1.6	53.3±1.3	76.7±1.1
DiffPool	70.6±1.2	57.9±2.5	78.6±2.5	81.5±4.1	89.6±1.1	56.2±1.1	77.5±0.7
Concat-Diff	70.7±0.7	60.0±1.7	77.7±2.5	81.3±2.9	91.1±1.7	54.9±1.4	78.0±0.5
Multi-Diff-2	70.7±0.6	56.3±1.3	80.0±1.2	79.3±2.9	89.9±2.5	53.7±0.6	76.8±0.7
Multi-Diff-3	70.8±1.1	52.5±0.8	80.9±1.7	80.6±2.3	88.8±0.7	53.4±2.4	78.5±1.2
Customized-DiffPool	73.6±0.5	57.9±7.2	78.6±2.9	80.6±2.6	95.1±1.6	55.8±1.1	78.2±0.9
gPool	69.4±2.2	53.8±3.2	77.3±3.0	78.9±5.5	88.9±1.6	51.3±0.6	77.1±1.2
Concat-gPool	69.7±0.5	57.1±1.8	79.1±2.2	78.0±2.6	88.5±1.3	50.9±2.2	76.3±0.7
Multi-gPool-2	69.0±1.9	50.8±5.1	79.7±1.0	79.5±2.7	84.0±3.2	49.3±2.3	73.5±2.1
Multi-gPool-3	68.9±1.6	46.2±3.2	80.6±0.8	80.0±3.6	83.1±4.5	48.9±1.8	75.2±1.9
Customized-gPool	72.3±1.0	62.9±3.6	80.6±1.6	80.0±3.1	91.1±0.7	53.3±1.4	76.5±1.9

different from these of training graphs. Following the setting in [15], for each graph dataset, we randomly shuffle the dataset and then split 90% of the data as the training set and the remaining 10% as test set. We train all the models on the training set and evaluate their performance on the test set with accuracy as the measure. We repeat this process with different data shuffling and initialization seeds for 4 times and report the average performance and standard variance. In terms of the implementation details, the GCN/Customized-GCN model consists of 3 filtering layers and a single max-poling layer; the hidden dimension of each filtering layer is 20; and ReLU [29] activation is applied after each filtering layer. For DiffPool/Customized-DiffPool and gPool/Customized-gPool, we set $K_p = 2$, $K_f = 3$ and the dimension of hidden filtering layer 20. We adopt fully-connected networks to implement the adaptor networks in the Customized-GNN frameworks. Its input dimension is the same as the dimension of the graph structural information.

The results are shown in Table 4. We notice that the Concat- and the Multi- version of the GNN models can, in some cases, achieve comparable or even better performance than their corresponding original versions. This indicates that utilizing the graph structure properties has the potential to help improve the model performance. However, the performance of these variants is not so stable across different datasets, which means that these simple methods are not suitable for all datasets. For example, the Concat- versions may work well on datasets where the label is directly related with the graph structure properties but fail on those datasets where graph structure properties have more implicit impact on the labels. On the other hand, the performance of the Multi-version of the GNN models is heavily dependent on how the data is split into different groups. It is not practical to find good splits manually. Furthermore, simply training different models for different graphs can lead to unsatisfactory performance because less training data is available for each model. In contrast, our proposed Customized models learn sample-wise adaptation, which automatically finds suitable models for different data samples according to their graph structure properties. Compared with the original GCN, DiffPool and gPool, the corresponding Customized models achieve better performance in most of the datasets. This demonstrates that the sample-wise adaptation performed by the Customized-GNN framework can boost the performance of GNN frameworks.

Adaptability Study. To further show the adaptability of the proposed framework to new graphs with different structures, we order graphs according to their node sizes in non-decreasing order. Then, we use the first 80% of the data as training set and the remaining 20% as test set. The purpose of this setting is to simulate the case where structures of graphs in the test set are different from those in the training set. We only show the results on

Table 5: Adaptability study. (Note here Cust-X denotes Customized-X)

Accuracy(%)	Methods					
	Cust-GCN	GCN	Cust-DiffPool	DiffPool	Cust-gPool	gPool
ENZ	22.2	20.5	25.6	22.2	35.0	24.8
RE-BI	70.2	50.4	78.6	52.7	80.0	59.9

the **ENZ** and **RE-BI** datasets in Table 5, since observation from other datasets are consistent. We note that (1) GCN, DiffPool and gPool cannot work properly in this setting; and (2) the customized frameworks perform much better under this setting. These results demonstrate the ability of the learned Customized-GNNs to adapt GNNs to graphs with new properties.

5.3 Ablation Study

In this subsection, we investigate the effectiveness of different components in the adaptor operator in Eq. equation 39 used in our model. Specifically, we want to investigate whether γ_f and β_f play important roles in the adaptor operator by defining the variants of Customized-GCN – **Customized-GCN**$_\gamma$: It is a variant of the adaptor operator with only element-wise multiplication operation where instead of Eq. equation 39, the adaptation process is now expressed as: $\mathbf{W} \diamond \phi_f = (\mathbf{W} \odot br(\gamma_f, d_{new}))$; and **Customized-GCN**$_\beta$: It is a variant of the adaptor operator with only element-wise addition operation where instead of Eq. equation 39, the adaptation process is now: $\mathbf{W} \diamond \phi_f = \mathbf{W} + br(\beta_f, d_{new})$.

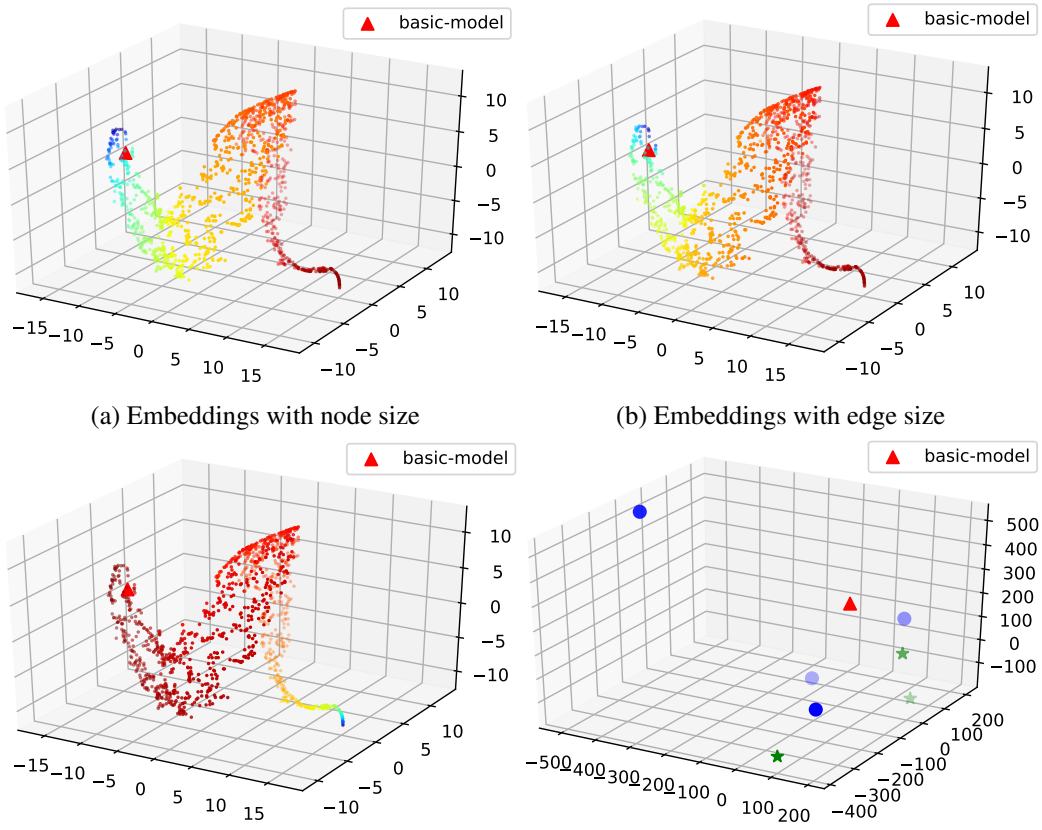
Table 6: Ablation study.

Accuracy (%)	Datasets						
	COLLAB	**ENZ**	**PROT**	**DD**	**RE-BI**	**RE-5K**	**NCI109**
GCN	69.9	51.8	76.6	77.2	81.9	50.4	75.7
Customized-GCN$_\gamma$	70.8	52.3	77.6	78.1	85.2	51.7	76.0
Customized-GCN$_\beta$	71.2	54.0	77.9	78.0	88.8	51.9	77.1
Customized-GCN	73.2	55.9	77.9	79.3	90.4	52.9	77.1

Following the previous experimental setting, we compared Customized-GCN with its variants. The results are presented in Table 6. We observe that both **Customized-GCN**$_\gamma$ and **Customized-GCN**$_\beta$ can outperform the original GCN model. It indicates that both terms with γ and β are effective for the adaptation and utilizing either one of them can already adapt the original model in a reasonable manner. We also note that the Customized-GCN model outperforms both **Customized-GCN**$_\gamma$ and **Customized-GCN**$_\beta$ on most of the datasets. It demonstrates that the adaption effects of the term with γ and β are complementary to each other and combining them together can further enhance the performance.

5.4 Case Study

To further illustrate the effectiveness of the proposed framework, we conducted case studies on D&D. First, we visualize the distribution of embeddings of sample-specific model parameters for different graph samples with various node sizes, edge sized and densities. Specifically, we take the parameters of the first filtering layer of each sample-specific Customized-GCN framework and then utilize t-sne [30] to project these parameters to 3-dimensional embeddings. We visualize these 3-d embeddings in the form of scatter plot as shown in Figure 5a, 5b and 5c. Note that in these figures, the red triangle denotes the embedding of the parameters (i.e. $\mathbf{W}$) of the



(a) Embeddings with node size

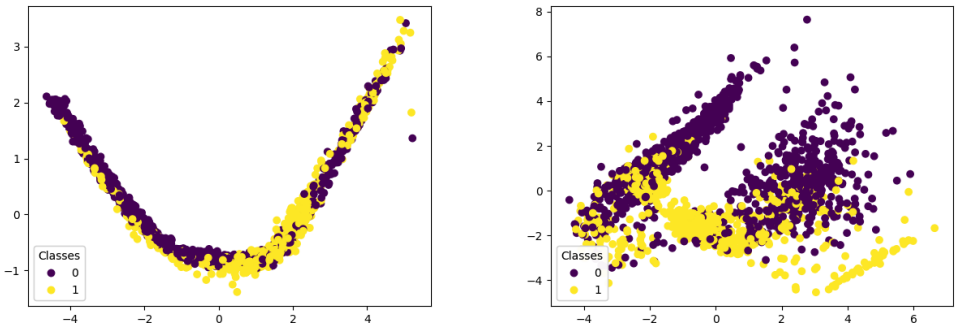
(b) Embeddings with edge size

(c) Embeddings with density

(d) Sample demonstration

Figure 5: Case Study. (a) depicts the model embeddings and (b) demonstrates model embeddings for graphs that are mistakenly classified by GCN, but correctly classified by Customized GCN; and (c) and (d) illustrate the embeddings for graphs extracted by GCN and Customized-GCN, respectively;

original GCN model (the one before adaptation). For each point in these figures, we use color to represent the scale of values in terms of node size (or edge size, density). Specifically, a deeper red color indicates a larger value, while a deeper blue color indicates a smaller value. We make some observations from Figure 5a, 5b and 5c. First, the proposed Customized-GCN framework indeed generates distinct models for different graph samples that are different from the original model. Second, the points with similar colors stay closely with each other, which means that graphs with similar structural information share similar models. In addition, in Figure 5d, we illustrate the sample-specific model parameters for seven samples with different number of nodes. They are mis-classified by the original GCN model but correctly classified by the proposed Customized-GCN framework. It is obvious that Customized-GCN has generated seven different GCN models for these graph samples, each of which can successfully predict the label for the corresponding sample. We further visualize the graph embeddings before the classification layer, extracted by the model GCN and Customized-GCN. These embeddings from the two models are then projected to a 2-dimensional space via PCA and shown in Figure 6a and 6b, respectively. We observe that the embeddings from different categories are better separated by Customized-GCN. This demonstrates that, compared to the original GCN, the proposed framework can get more distinct embeddings, and thus can achieve better classification performance.



(a) GCN (b) Customized-GCN

Figure 6: Embedding Visualization.

6 Conclusion

In this paper, we propose a general graph neural network framework, Customized-GNN, to deal with graphs that have various graph structure properties. Comprehensive experiments demonstrated that the Customized-GNN framework can effectively adapt both flat and hierarchical GNNs to enhance their performance. Future research directions include better modeling the adaptor networks, considering more complex properties, and adapting more existing graph neural networks models.

References

[1] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in Advances in neural information processing systems, 2017, pp. 1024–1034.

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

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," arXiv preprint arXiv:1710.10903, 2017.

[5] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," Bioinformatics, vol. 21, no. suppl_1, 2005.

[6] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," Journal of Machine Learning Research, vol. 12, no. Sep, pp. 2539–2561, 2011.

[7] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017, pp. 1263–1272.

[8] Y. Rong, Y. Bian, T. Xu, W. Xie, Y. Wei, W. Huang, and J. Huang, "Self-supervised graph transformer on large-scale molecular data," in NeurIPS, 2020.

[9] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," IEEE Access, vol. 7, pp. 19 143–19 165, 2019.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.

[12] Z. Zhang, H. Yang, J. Bu, S. Zhou, P. Yu, J. Zhang, M. Ester, and C. Wang, "Anrl: Attributed network representation learning via deep neural networks." in IJCAI, vol. 18, 2018, pp. 3155–3161.

[13] H. Gao and S. Ji, "Graph u-nets," arXiv preprint arXiv:1905.05178, 2019.

[14] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar, "Composition-based multi-relational graph convolutional networks," in International Conference on Learning Representations, 2020. [Online]. Available: https://openreview.net/forum?id=BylA_C4tPr

[15] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in Advances in neural information processing systems, 2018, pp. 4800–4810.

[16] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 723–731.

[17] J. Li, Y. Ma, Y. Wang, C. Aggarwal, C.-D. Wang, and J. Tang, "Graph pooling with representativeness," in 2020 IEEE International Conference on Data Mining (ICDM). IEEE, 2020, pp. 302–311.

[18] H. Gao, Y. Liu, and S. Ji, "Topology-aware graph pooling networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.

[19] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," arXiv preprint arXiv:1912.09893, 2019.

[20] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," Journal of molecular biology, vol. 330, no. 4, pp. 771–783, 2003.

[21] T.-A. Song, S. R. Chowdhury, F. Yang, H. Jacobs, G. El Fakhri, Q. Li, K. Johnson, and J. Dutta, "Graph convolutional neural networks for alzheimer's disease classification," in 2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019). IEEE, 2019, pp. 414–417.

[22] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," Neural computation, vol. 29, no. 9, pp. 2352–2449, 2017.

[23] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," Information, vol. 10, no. 4, p. 150, 2019.

[24] K. Hornik, "Approximation capabilities of multilayer feedforward networks," Neural networks, vol. 4, no. 2, pp. 251–257, 1991.

[25] E. Ranjan, S. Sanyal, and P. P. Talukdar, "Asap: Adaptive structure aware pooling for learning hierarchical graph representations," arXiv preprint arXiv:1911.07979, 2019.

[26] H. Yuan and S. Ji, "Structpool: Structured graph pooling via conditional random fields," in International Conference on Learning Representations, 2020.

[27] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[29] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814.

[30] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," Journal of machine learning research, vol. 9, pp. 2579–2605, 2008.

[31] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014.

[32] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in Advances in neural information processing systems, 2016, pp. 3844–3852.

[33] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in Thirty-second AAAI conference on artificial intelligence, 2018.

[34] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[35] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[36] K. Schütt, P.-J. Kindermans, H. E. S. Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "Schnet: A continuous-filter

convolutional neural network for modeling quantum interactions," in Advances in neural information processing systems, 2017, pp. 991–1001.

[37] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in Advances in neural information processing systems, 2015, pp. 2224–2232.

[38] M. Fey, J. Eric Lenssen, F. Weichert, and H. Müller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 869–877.

[39] K. Zhou, Q. Song, X. Huang, D. Zha, N. Zou, and X. Hu, "Multi-channel graph neural networks," arXiv preprint arXiv:1912.08306, 2019.

[40] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," arXiv preprint arXiv:1904.08082, 2019.

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