All in a Row: Compressed Convolution Networks for Graphs

Junshu Sun [1 2] Shuhui Wang [1 3] Xinzhe Han [1 2] Zhe Xue [4] Qingming Huang [1 2 3]

Abstract

Compared to Euclidean convolution, existing graph convolution methods generally fail to learn diverse convolution operators under limited parameter scales and depend on additional treatments of multi-scale feature extraction. The challenges of generalizing Euclidean convolution to graphs arise from the irregular structure of graphs. To bridge the gap between Euclidean space and graph space, we propose a differentiable method for regularization on graphs that applies permutations to the input graphs. The permutations constrain all nodes in a row regardless of their input order and therefore enable the flexible generalization of Euclidean convolution. Based on the regularization of graphs, we propose Compressed Convolution Network (CoCN) for hierarchical graph representation learning. CoCN follows the local feature learning and global parameter sharing mechanisms of Convolution Neural Networks. The whole model can be trained end-to-end and is able to learn both individual node features and the corresponding structure features. We validate CoCN on several node classification and graph classification benchmarks. CoCN achieves superior performance over competitive convolutional GNNs and graph pooling models. Codes are available at https://github.com/sunjss/CoCN.

1. Introduction

Graph Neural Networks (GNNs) based on convolution have become ubiquitous in various graph representation learning tasks, such as node classification (Hamilton et al., 2017) and graph classification (Xu et al., 2019). Despite their effectiveness, convolutional GNNs (Defferrard et al., 2016; Kipf & Welling, 2017; He et al., 2021; Chien et al., 2022) suffer from inherent problems. First, the customized graph convolution based on polynomials is less expressive under limited parameter scales (Li et al., 2018; Huang et al., 2022; Eliasof et al., 2022). Second, convolutional GNNs fail to directly perform hierarchical representation learning (Ying et al., 2018). Existing graph convolution models require additional node clustering (Ying et al., 2018) or node drop methods (Gao & Ji, 2019) to extract multi-scale features. In contrast, Convolution Neural Networks (CNNs) based on Euclidean convolution are free of the above problems. Owning to the local feature learning ability and global parameter sharing mechanism, CNNs can learn diverse convolution operators and capture multi-scale local patterns on regular grid data. The expressiveness of CNN has led to its great success in various domains such as image understanding (Krizhevsky et al., 2017; He et al., 2016) and video understanding (Ji et al., 2013; Tran et al., 2015). In light of this, a straightforward solution is to adopt Euclidean convolution for GNNs on non-grid graph data.

Nevertheless, directly generalizing Euclidean convolution to graphs faces with challenges from various aspects. From the graph perspective, they generally have irregular local structures that are different from the grid data in Euclidean space. Therefore, a graph regularization module is required to extend the expressiveness of Euclidean convolution from regular grids to irregular topology. Methods have been proposed for the regularization of graphs based on node sequence selection (Niepert et al., 2016; Eliasof et al., 2022). However, these methods may generate less informative regularized graphs for the subsequent convolution operations, since their regularization methods on graphs are independent of the convolution process and hence cannot be optimized for specific tasks. From the Euclidean convolution perspective, it is sensitive to the local spatial order, while GNNs should preserve permutation invariance, *i.e.*, produce the same node representations regardless of the order of the nodes. For permutation invariant transformation, recent studies (Murphy et al., 2019a;b; Huang et al., 2022) propose to enumerate or sample permutations to adopt permutation-sensitive operators for GNNs, but they are computationally

[1]Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, China [2]School of Computer Science and Technology, University of Chinese Academy of Sciences, China [3]Peng Cheng Laboratory, China [4]Beijing Key Laboratory of Intelligent Telecommunication Software and Multimedia, School of Computer Science, Beijing University of Posts and Telecommunications, China. Correspondence to: Shuhui Wang <wangshuhui@ict.ac.cn>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

intractable for learning on large graphs.

In this paper, we propose Compressed Convolution Network (CoCN), a hierarchical GNN model for end-to-end graph representation learning. Compared to existing models, we highlight our CoCN as follows: **(1)** it optimizes regular receptive field for convolution while preserving permutation invariance; **(2)** it can learn diverse local operators; and **(3)** it directly applies hierarchical feature learning on graphs. Technically, CoCN generalizes Euclidean convolution to graphs with two main components, *i.e.*, Permutation Generation and Diagonal Convolution. For Permutation Generation, we consider regularization on graphs as a permutation problem that arranges input nodes under a proper order. It constrains all nodes in a row with permutation invariance, enabling flexible generalization of Euclidean convolution to graphs. This is achieved by approximating the permutation matrix through a differentiable transformation with node position regression and cyclic shift.

We theoretically demonstrate the convergence of our proposed permutation generation method and the permutation invariance of the permuted features. Based on the permuted graph representation, Diagonal Convolution is proposed to aggregate both individual node features and the corresponding structure features. It inherits the local feature learning and global parameter sharing mechanisms from Euclidean convolution and follows the diagonal sliding fashion for edge feature learning. Moreover, to directly achieve hierarchical feature learning with Diagonal Convolution, anti-diagonal compression (Fig. 1) is proposed for edge features update. Therefore, CoCN can learn diverse local operators and extract both node features and edge features explicitly with hierarchical convolution. Our contribution can be summarized as follows:

- We propose a novel method for regularization on graphs that enables the generalization of the permutation-sensitive Euclidean convolution to graphs.
- We propose a hierarchical GNN model, CoCN, which can learn both individual node features and the corresponding structure features from coarse to fine.
- We demonstrate the advantages of CoCN on six node classification and six graph classification benchmarks.

2. Related Work

Convolution Operator on Graphs. To generalize convolution operators to graph-structured data, Bruna et al. first propose graph convolution based on graph signal processing theory. To reduce filter parameter scale, ChebyNet (Defferrard et al., 2016) uses parameterized graph Laplacian polynomial to learn convolution filters. Following up this work, models with simpler filter structures have been proposed, *e.g.*, first-order low-pass filter model (Kipf & Welling, 2017; Wu et al., 2019). Except for graph Laplacian polynomials, methods have been proposed for more expressive convolution operators. Klicpera et al. extend graph Laplacian to a more general transition matrix. Inspired by Feynman path integral theory, PAN (Ma et al., 2020) formulates graph convolution as the polynomial of adjacency matrix with coefficients depending on the corresponding path. Though polynomials can approximate any sophisticated filter theoretically (Wang & Zhang, 2022), its expressive power is bounded by the order, and high-order filters are known to be computationally expensive.

The other vital type of graph convolution is combining regularized graphs with shared filters. These methods are able to learn complex filters without the constraint of polynomial order. Niepert et al. propose to extract normalized neighborhoods to serve as input data to CNNs. However, the model is less expressive since its normalization cannot be integrated into the learning process. PathConv (Eliasof et al., 2022) regularizes graphs with random walk and applies convolution to the generated node sequence. PG-GNN (Huang et al., 2022) also utilizes sampling methods to convert the neighborhood of nodes into sequences and models pairwise correlations by RNN. Despite using expressive spatial filters, the above-mentioned sampling strategies involve increasing computational costs and may inject noise signals. Compared to existing regularization-based models, our CoCN regularizes the input graphs with differentiable permutations that can be optimized for specific tasks.

Hierarchical Pooling GNNs. Except for node-level and graph-level features, the intermediate-scale features are also crucial for graph representation learning (Ying et al., 2018; Boguñá et al., 2021). GNNs with hierarchical representation learning have been studied, which are generally called graph pooling or graph coarsening. Defferrard et al. use Graclus greedy algorithm to select and combine node pairs at every coarsening level. To further preserve structural information, more topology-based clustering methods are adopted for iterative graph pooling, such as edge collapsing (Hu, 2006; Chen et al., 2018), structural similarity (Hu et al., 2019) and spectral similarity (Deng et al., 2020). Other node clustering methods (Ying et al., 2018; Baek et al., 2022) learn soft assignment matrix rather than deterministic clustering to perform graph pooling. Node drop methods (Cangea et al., 2018; Gao & Ji, 2019; Lee et al., 2019) use top-K selection method to drop irrelevant nodes. Despite their effectiveness, these methods suffer from information loss (Wu et al., 2022). To address this problem, Wu et al. use structural entropy to get a learning-free hierarchical structure. Different from graph pooling methods, our CoCN avoids node clustering or node drop layer-by-layer and performs hierarchical representation learning directly through diagonal convolution with anti-diagonal compression.

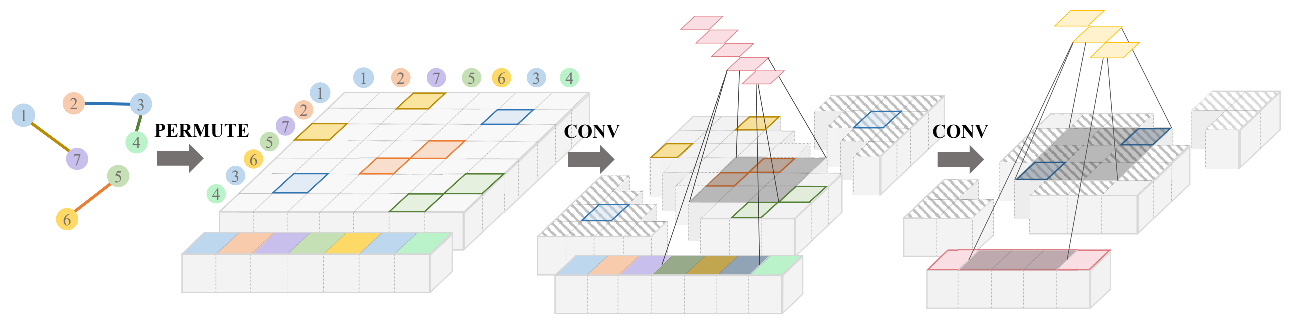

Figure 1. **Compressed Convolution Network Pipeline.** CoCN first permutes the node feature matrix and the edge feature matrix based on the learnable permutation matrix. Then diagonal convolution is applied to both feature matrices following the diagonal sliding fashion on the edge feature matrix. The off-diagonal features, displayed with twill pattern, are not involved in the convolution, so they are learned in the subsequent layers.

3. Regularization on Graphs

We describe how to regularize graph-structured data through learnable permutations, which facilitates the whole model to be trained end-to-end. This enables task-specific optimization on the regularized graph.

3.1. Notations

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with node set $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ of n nodes and edge set $\mathcal{E}$. Each node $v \in \mathcal{V}$ has a feature vector $\mathbf{x}_v \in \mathbb{R}^d$ where d denotes the number of features. We use $\mathbf{X} \in \mathbb{R}^{n\times d}$ to denote the graph node feature matrix. Node feature vector $\mathbf{x}_{v_i}^\top$ corresponds to the i-th row of $\mathbf{X}$. Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be the adjacency matrix of $\mathcal{G}$ and $\mathbf{D} \in \mathbb{R}^{n\times n}$ be the diagonal degree matrix. $\mathbf{A}_{i,j} = 1$ if there exists an edge $e_{i,j} = (i, j) \in \mathcal{E}$ and $\mathbf{D}_{i,i} = \sum_j \mathbf{A}_{i,j}$. We use $\mathbf{1} \in \mathbb{R}^n$ to denote the all-ones vector. Let $\mathcal{P}$ be the permutation set over n indices, $|\mathcal{P}| = n!$. For any $\mathbf{P} \in \mathcal{P}$, $\mathbf{P} \in \mathbb{R}^{n\times n}$, $\mathbf{P1} = \mathbf{1}^\top\mathbf{P} = \mathbf{1}$, $\mathbf{P}_{i,j} \in \{0, 1\}$.

3.2. Position Regression

We formulate permutation generation as a position regression problem. The intuition behind is that nodes with similar features or short paths in between will get closer position prediction. The approximate position $\mathbf{r}_A \in \mathbb{R}^n$ is given by:

$$\mathbf{r}_A = f\left(\mathbf{X}, \mathbf{A}\right), \tag{1}$$

where f can be any permutation equivariant function. Since the approximate position for each node is a scalar, we can apply global pair-wise value comparison to rank the nodes. The resulted absolute position $\mathbf{r} \in \mathbb{R}^n$ can be formulated as:

$$\mathbf{r} = \mathtt{sgn}\left(\mathbf{r}_A\mathbf{1}^\top - \mathbf{1}\mathbf{r}_A^\top\right)\mathbf{1}, \tag{2}$$

where $\mathtt{sgn}(\cdot)$ denotes the sign function. If $x > 0$, $\mathtt{sgn}(x) = 1$ otherwise $\mathtt{sgn}(x) = 0$. Note that the sign function is not differentiable. To address this problem, we use the sigmoid function to approximate the sign function in backpropagation.

3.3. Permutation Matrix Generation

With Eq. 1 and Eq. 2, the generated node positions are assigned to all input nodes. A natural follow-up question is how to convert the absolute position $\mathbf{r}$ into a permutation matrix. We resort to the cyclic shift (Carter, 2009) of matrix entries.

Take row permutation as an example, $\mathbf{P}_{i,j} = 1$ indicates moving the j-th row to the i-th row. Therefore, given absolute position $\mathbf{r}_i$ for the i-th node, the corresponding permutation matrix $\mathbf{P}$ should have an entry equal to 1 at $(\mathbf{r}_i, i)$. To convert absolute positions into a permutation matrix, the model needs to assign 1 to the given positions.

We first consider the column vector $\mathbf{P}_{\cdot,j}$ which corresponds to the j-th node assignment. Let $\mathbf{m} \in \mathbb{R}^n$ denote the indicator of $\mathbf{P}_{\cdot,j}$, initialized as $\mathbf{m}_i = i$. The zero entry of $\mathbf{m}$ indicates that the corresponding entry of $\mathbf{P}_{\cdot,j}$ equals 1, while non-zero entries indicate the number of cyclic shift steps to 1. $\mathbf{P}_{i,j}$ is $\mathbf{m}_i$ steps away from value 1. The cyclic shift on $\mathbf{m}$ is equivalent to value assignment on $\mathbf{P}_{\cdot,j}$. We can use element-wise addition and modulus to cyclically shift the indicator entries. Given absolute position k, the initial indicator $\mathbf{m}$ takes k steps cyclic shift by $(\mathbf{m} - k + n) \pmod n$.

We now extend to determine the whole permutation matrix for the given absolute position $\mathbf{r}$. Each column of the permutation matrix corresponds to a single node assignment. Let $\mathbf{m1}^\top$ denote the initial indicator of the permutation matrix. The absolute position $\mathbf{r}$ can be converted into permutation matrix as:

$$\hat{\mathbf{P}} = \exp\left\{-\tau\left[\left(\mathbf{m1}^\top - \mathbf{1r}^\top + n\right) \pmod n\right]\right\}, \tag{3}$$

where τ denotes the relaxation factor and $\exp(\cdot)$ denotes the exponential function for mapping indicator entries to permutation entries. Note that the Eq. 3 gives the relaxed permutation matrix to overcome the gradient vanishing problem, where $\hat{\mathbf{P}}_{i,j} \in (0, 1]$. One can instead use Proposition 3.1 or replace $\exp(\cdot)$ with other functions to get the standard permutation matrix. Proofs for propositions in this section are provided in Appendix A.

Proposition 3.1. ***(Permutation Convergence)*** *$\hat{\mathbf{P}}$ converges to standard permutation matrix as relaxation factor τ approaching positive infinity: $\underset{\tau\to+\infty}{lim} \hat{\mathbf{P}} = \mathbf{P}, \mathbf{P} \in \mathcal{P}$.*

For simplicity, we compile Eq. 1-3 as operation $\texttt{PERM}(\cdot)$. The output of $\texttt{PERM}(\cdot)$ can be used to generate permutation invariant input for the subsequent convolution.

Proposition 3.2. ***(Permutation Invariant Input)*** *Let f be a permutation equivariant function such that for any $\mathbf{P} \in \mathcal{P}$, $f(\mathbf{PX}, \mathbf{PAP}^\top) = \mathbf{P}f(\mathbf{X}, \mathbf{A})$. Then given $\hat{\mathbf{P}} = \texttt{PERM}(\mathbf{X}, \mathbf{A})$, $\hat{\mathbf{X}} = \hat{\mathbf{P}}\mathbf{X}$, and $\hat{\mathbf{A}} = \hat{\mathbf{P}}\mathbf{A}\hat{\mathbf{P}}^\top$, for any $\mathbf{P} \in \mathcal{P}$ on $\mathbf{X}$ and $\mathbf{A}$, $\hat{\mathbf{X}}$ and $\hat{\mathbf{A}}$ are invariant.*

Regularization on graphs with position regression and order permutation transfers the irregular graph to a regular vector with proper order. Proposition 3.1 and 3.2 demonstrate that such permutation is differentiable, convergent, and input-order invariant, which facilitates flexible calculation of spatial convolutions.

4. Compressed Convolution Network

In this section, we present Compressed Convolution Network (CoCN), a GNN model based on diagonal convolution for hierarchical graph representation learning. We first outline the implementation of the building blocks of CoCN and then describe the network architecture for node classification and graph classification.

4.1. Permutation Generation

The permutation module is described in Section 3. There are many possible implementations of the permutation equivariant function in Eq. 1. In this paper, we use MLP to learn node features and the Laplacian operator to smooth the approximate position of connected nodes. Nodes with similar features or short paths in between will have similar positions. Eq. 1 can be written as:

$$\mathbf{r}_A = \tilde{\mathbf{A}}^t \texttt{MLP}(\mathbf{X}), \tag{4}$$

where $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ denotes the Laplacian operator, and t denotes the power of $\tilde{\mathbf{A}}$ to adjust the smoothness. MLP uses ReLU as the activation function at each layer. In practice, we produce multiple approximate positions $\mathbf{r}_A$ to allow node arrangements under different similarities. The permutation generation module then follows Eq. 2 and Eq. 3 to generate multiple permutations to the subsequent convolution layers.

4.2. Diagonal Convolution

The permutations can be seen as a spatial position mapping to Euclidean space where all nodes are in a row. The adjacent nodes can be defined as a sequence of neighbors regardless of the edge connection. All nodes share the constant local geometry structure, allowing us to generalize convolution in Euclidean space to graphs. We now present our diagonal convolution operator.

Let $\hat{\mathbf{X}} = \hat{\mathbf{P}}\mathbf{X}$ be the sequential node feature matrix and $\hat{\mathbf{A}} = \hat{\mathbf{P}}\mathbf{A}\hat{\mathbf{P}}^\top$ be the sequential topological feature matrix. Note that the adjacency matrix can be seen as the edge feature matrix with a single channel. We can easily generalize our model to multi-channel edge features. For simplicity, we only consider the adjacency matrix here. Similar to 2D convolution, the diagonal structural convolution with a single kernel can be formulated as:

$$\mathcal{K}_i^{str}\left(\hat{\mathbf{A}}, k\right) = \sum_{p=0}^{k-1}\sum_{q=0}^{k-1} \mathbf{w}_{p,q}\hat{\mathbf{A}}_{i+p,i+q}, \tag{5}$$

where $\mathbf{w} \in \mathbb{R}^{k\times k}$ denotes the convolution kernel parameters for $\hat{\mathbf{A}}$, k denotes the kernel size and i denotes the index of $\hat{\mathbf{A}}$ that corresponds to the top-left entry of convolution kernel. Eq. 5 gives the single-step diagonal structural convolution. Typical 2D convolution follows the progressive sliding fashion on images where the kernels first slide in a column-wise manner and then row-wise. Instead, we perform diagonal sliding on $\hat{\mathbf{A}}$ to adapt to the scale of graphs. Specifically, for $k \times k$ convolution with step length s, the top-left entry index i starts at $(0, 0)$, moves to (s, s) and stops at $(n-k+1, n-k+1)$. Further including sequential node features, the diagonal structural convolution can be generalized to a unified diagonal convolution as:

$$\mathcal{K}_i\left(\hat{\mathbf{A}}, \hat{\mathbf{X}}, k\right) = \sum_{p=0}^{k-1}\left(\sum_{q=0}^{k-1} \mathbf{w}_{p,q}\hat{\mathbf{A}}_{i+p,i+q} + \sum_{t=0}^{d-1} \mathbf{v}_{p,t}\hat{\mathbf{X}}_{i+p,t}\right), \tag{6}$$

where $\mathbf{v} \in \mathbb{R}^{k\times d}$ denotes the convolution kernel parameters for $\hat{\mathbf{X}}$. For the input graph of n nodes, applying $k \times k$ diagonal convolution with diagonal sliding step length s gives rise to the node feature vector of length $\lfloor \frac{n-k}{s} \rfloor + 1$. Let $\mathbf{S} = (\mathbf{S}_j) \in \mathbb{R}^{\lfloor \frac{n-k}{s} \rfloor + 1}$ denote the output single channel node feature vector. We can formulate the multi-step diagonal convolution with diagonal sliding as:

$$\begin{aligned} \mathbf{S} &= \mathcal{K}\left(\hat{\mathbf{A}}, \hat{\mathbf{X}}, k, s\right), \\ \mathbf{S}_j &= \mathcal{K}_i\left(\hat{\mathbf{A}}, \hat{\mathbf{X}}, k\right),\ i = sj. \end{aligned} \tag{7}$$

4.3. Compressed Convolution Layer

To construct a multi-layer network with diagonal convolution, we need to update the input features at each layer. As described in Subsection 4.2, the diagonal convolution follows the diagonal sliding fashion on the input edge feature matrix ($\hat{\mathbf{A}}$ for example). At each step, the diagonal convolution extracts node set features including individual node features from $\hat{\mathbf{X}}$ and the structure features among nodes from the main diagonal blocks of $\hat{\mathbf{A}}$.

For individual node feature update, we take input nodes as n node sets where each node set only contains a single node. Let $\mathbf{H}$ be the node set feature matrix and $\mathbf{H}^{(0)} = \hat{\mathbf{X}}$ be the initial node set feature matrix. $\mathbf{H}$ can be updated with the diagonal convolution output.

For structure feature update, let $\mathbf{E}$ be the structure feature matrix initialized with $\mathbf{E}^{(0)} = \hat{\mathbf{A}}$[1]. The off-diagonal features in $\mathbf{E}$ that are not involved in the convolution represent the topological structure features among node sets. If the diagonal convolution takes unit sliding steps, we can update $\mathbf{E}$ by removing the main diagonal blocks. We use PyTorch-style pseudo-code to formulate the update equation as $\mathbf{E}^{(l)} = \mathtt{Tri}(\mathbf{E}^{(l-1)}, k) = \mathtt{triu}(\mathbf{E}^{(l-1)}, k) + \mathtt{tril}(\mathbf{E}^{(l-1)}, -k)$, where k denotes the kernel size, $\mathtt{triu}(\cdot, i)$ denotes the upper triangular matrix with i diagonals above the main diagonal and $\mathtt{tril}(\cdot, -i)$ denotes the lower triangular matrix with i diagonals below the main diagonal. If the diagonal convolution takes non-unit steps, we use standard 2D max pooling with the same kernel size and step size as diagonal convolution to update the structure features.

With the proposed update method, the structure feature matrix $\mathbf{E}$ is compressed from the anti-diagonal direction. We name this network layer with diagonal convolution and anti-diagonal compression as *compressed convolution layer*. The l-th compressed convolution layer with $k^{(l)} \times k^{(l)}$ diagonal convolution and sliding step $s^{(l)}$ is formulated as:

$$\mathbf{H}^{(l)} = \sigma\left(\mathcal{K}\left(\mathbf{E}^{(l-1)}, \mathbf{H}^{(l-1)}, k^{(l)}, s^{(l)}\right)\right), \tag{8}$$

$$\mathbf{E}^{(l)} = \begin{cases} \mathtt{Tri}\left(\mathbf{E}^{(l-1)}, k^{(l)}\right), & \text{if } s^{(l)} = 1; \\ \mathtt{MaxPool}\left(\mathbf{E}^{(l-1)}, k^{(l)}, s^{(l)}\right), & \text{otherwise.} \end{cases} \tag{9}$$

where σ denotes the ReLU function, $\mathbf{E}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l)}}$ and $n^{(l)}$ denotes the number of the output node sets. Following the common practice in CNN models, compressed convolution layers will contain multiple convolution kernels which give rise to $\mathbf{H}^{(l)} \in \mathbb{R}^{n^{(l)} \times c^{(l)}}$, where $c^{(l)}$ denotes the number of kernels. We refer to the compressed convolution layer with non-unit step as *compressed pooling layer*.

[1] $\hat{\mathbf{A}}$ can be replaced with any multi-channel edge feature matrix.

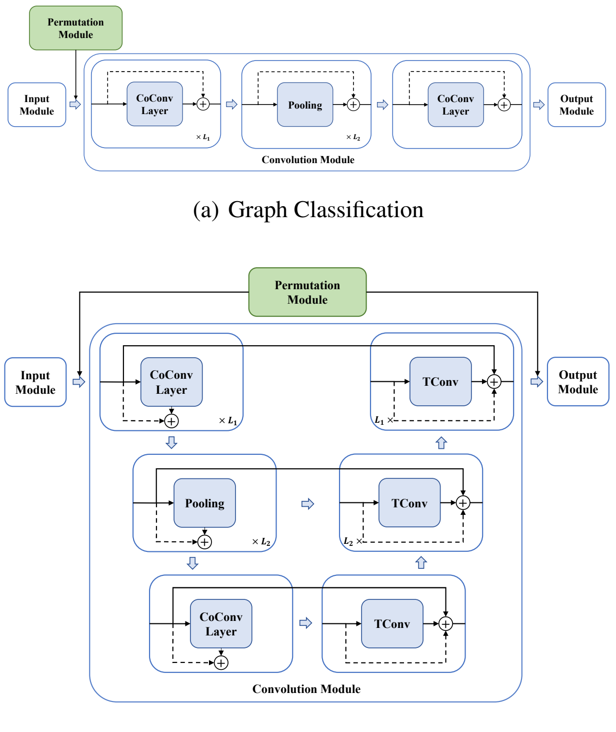

Figure 2. **Compressed Convolution Network Structure.** The CoConv layer represents compressed convolution layers. Pooling represents compressed pooling layers. TConv represents transposed convolution with ReLU. The dashed lines represent shortcut with average pooling

4.4. Network Architecture

We now present our hierarchical graph representation learning model CoCN. CoCN mainly contains four modules, *i.e.*, input module, permutation generation module, convolution module and output module. The network structures of CoCN for graph classification and node classification are presented in Fig. 2(a) and Fig. 2(b). The input module is implemented with a single-layer linear transformation followed by layer normalization and ReLU. The permutation generation module has been described in Subsection 4.1. In the convolution module, features permuted under different permutation matrices are learned in parallel, where different permutation heads share the same module parameters.

For both tasks, the down-sampling convolution block consists of L_1 compressed convolution layers with unit step, L_2 compressed pooling layers with the same step length as filter size, and a single compressed convolution layer with unit step. To recover node representations from multi-scale features, we stack $L_1 + L_2 + 1$ transposed convolution layers (Noh et al., 2015) as an up-sampling convolution block for node classification. Direct shortcut connections are established between the compressed convolution/pooling layers

Table 1. **Graph Classification Results (measured by accuracy: %).**

	MUTAG	PROTEINS	NCI1	IMDB-B	IMDB-M	COLLAB
GRAPHS	188	1,113	4,110	1,000	1,500	5,000
AVG NODES	17.93	39.06	29.87	19.77	13	74.5
AVG EDGES	39.6	145.6	64.6	193.1	131.87	4,914.4
NODE FEATURES	7	3	37	0	0	0
PATCHY-SAN	**88.95±4.21**	75.00±2.51	78.60±1.90	71.00±2.29	45.23±2.84	72.60±2.15
GCN	69.50±1.78	73.24±0.73	76.29±1.79	73.26±0.46	50.39±0.41	80.59±0.27
GIN	81.39±1.53	71.46±1.66	80.00±1.40	72.78±0.86	48.13±1.36	78.19±0.63
GRAPHSAGE	83.60±9.60	73.00±4.50	76.00±1.80	68.80±4.50	47.60±3.50	73.90±1.70
PG-GNN	-	76.80±3.80	82.80±1.30	76.80±2.60	53.20±3.60	80.90±0.80
TOPKPOOL	67.61±3.36	70.48±1.01	67.02±2.25	71.58±0.95	48.59±0.72	77.58±0.85
SAGPOOL	73.67±4.28	71.56±1.49	67.45±1.11	72.55±1.28	50.23±0.44	78.03±0.31
ASAP	77.83±1.49	73.92±0.63	71.48±0.42	72.81±0.50	50.78±0.75	78.64±0.50
DIFFPOOL	79.22±1.02	76.25±1.00	62.32±1.90	73.14±0.70	51.31±0.72	82.13±0.43
MINCUTPOOL	79.17±1.64	74.72±0.48	74.25±0.86	72.65±0.75	51.04±0.70	80.87±0.34
STRUCTPOOL	79.50±0.75	75.16±0.86	78.64±1.53	72.06±0.64	50.23±0.53	77.27±0.51
EDGEPOOL	74.17±1.82	75.12±0.76	-	72.46±0.74	50.79±0.59	-
SEP	85.56±1.09	76.42±0.39	79.35±0.33	74.12±0.56	51.53±0.65	81.28±0.15
GMT	83.44±1.33	75.09±0.59	76.35±2.62	73.48±0.76	50.66±0.82	80.74±0.54
COCN (OURS)	87.08±0.17	**76.86±0.13**	**82.89±0.19**	**77.26±0.27**	**56.32±0.18**	**86.15±0.10**

and the transposed convolution layers for better feature recovery. Within both the compressed convolution/pooling layers and the transposed convolution layers, we also add average pooling as shortcuts. Eq. 8 can be written as:

$$\begin{aligned}\mathbf{H}^{(l)} =&\sigma\left(\mathcal{K}\left(\mathbf{E}^{(l-1)}, \mathbf{H}^{(l-1)}, k^{(l)}, s^{(l)}\right)\right)\\ &+\texttt{AvgPool}\left(\mathbf{H}^{(l-1)}, k^{(l)}, s^{(l)}\right).\end{aligned} \tag{10}$$

In the output module of graph classification, CoCN uses max pooling to extract graph-level representation. While for node classification, CoCN re-permutes the extracted features from the L-th layer with $\hat{\mathbf{P}}^\top\mathbf{H}^{(L)}$ and concatenates the re-permuted features under different permutations to perform the final predictions.

5. Experiments

We empirically evaluate CoCN on real-world benchmarks. The code of CoCN is implemented with PyTorch (Paszke et al., 2019) and PyTorch-Geometric (Fey & Lenssen, 2019). CoCN is trained on a single Nvidia Geforce RTX 3090. The detailed model structure and hyper-parameter setting are presented in Appendix B.

5.1. Graph Classification

Datasets. For graph classification, we use six datasets (Morris et al., 2020) including three biochemical datasets MUTAG, PROTEINS, NCI1 and three social network datasets COLLAB, IMDB-BINARY and IMDB-MULTI. We perform 10-fold cross-validation with LIB-SVM following Xu et al. and report average performance. Since COLLAB, IMDB-BINARY and IMDB-MULTI have no input node features, we use the one-hot encoding of node degrees as node features following Xu et al.. The statistics of these datasets are summarized in Tab. 1.

Baselines. For baseline models, we consider **(1)** GNNs with different convolution designs including PATCHY-SAN (Niepert et al., 2016), GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GIN (Xu et al., 2019) and PG-GNN (Huang et al., 2022); **(2)** Node drop based hierarchical graph pooling models including TopKPool (Gao & Ji, 2019), SAGPool (Lee et al., 2019) and ASAP (Ranjan et al., 2020); and **(3)** Node clustering based hierarchical graph pooling models including DiffPool (Ying et al., 2018), EdgePool (Diehl, 2019), MinCutPool (Bianchi et al., 2020), StructPool (Yuan & Ji, 2020), SEP (Wu et al., 2022) and GMT (Baek et al., 2022).

Performance. The results of CoCN and baselines for graph classification are presented in Tab. 1. CoCN surpasses convolutional GNNs and graph pooling methods on all datasets except MUTAG, especially on social network datasets like COLLAB that do not provide any node features. For MUTAG, only PATCHY-SAN outperforms our method but it is very unstable across repeat tests and degrades a lot on COLLAB and IMDB-M due to the lack of edge feature learning. In Section 5.3, we further demonstrate that CoCN can achieve comparable performance to PATCHY-SAN with only structure features on MUTAG.

5.2. Node Classification

Datasets. For node classification, we conduct experiments on six datasets including Chameleon, Squirrel (Rozemberczki et al., 2021), Cornell, Texas, Wisconsin (Pei et al.,

Table 2. **Node Classification Results (measured by accuracy: %).**

	CHAMELEON	SQUIRREL	CORNELL	TEXAS	WISCONSIN	ACTOR	AVG RANK
NODES	2,277	5,201	198	183	251	7,600	
EDGES	36,101	198,493	295	309	499	33544	
FEATURES	2,325	2,089	1,703	1,703	1,703	932	
GCN	59.82 ± 2.58	36.89 ± 1.34	57.03 ± 4.67	59.46 ± 5.25	59.80 ± 6.99	30.26 ± 0.79	11.17
CHEBYNET	55.24 ± 2.76	43.86 ± 1.64	74.30 ± 7.46	77.30 ± 4.07	79.41 ± 4.46	34.11 ± 1.09	8.17
GEOM-GCN	60.00 ± 2.81	38.15 ± 0.92	60.54 ± 3.67	66.76 ± 2.72	64.51 ± 3.66	31.59 ± 1.15	10.17
GRAPHSAGE	49.06 ± 1.88	36.73 ± 1.21	80.08 ± 2.96	82.03 ± 2.77	81.36 ± 3.91	35.07 ± 0.15	7.00
FAGCN	46.07 ± 2.11	30.83 ± 0.69	76.76 ± 5.87	76.49 ± 2.87	79.61 ± 1.58	34.82 ± 1.35	9.17
APPNP	40.44 ± 2.02	29.20 ± 1.45	56.76 ± 4.58	55.10 ± 6.23	54.59 ± 6.13	30.02 ± 0.89	13.00
MIXHOP	60.50 ± 2.53	43.80 ± 1.48	73.51 ± 6.34	77.84 ± 7.73	75.88 ± 4.90	32.22 ± 2.34	8.33
LINKX	68.42 ± 1.38	61.81 ± 1.80	77.84 ± 5.81	74.60 ± 8.37	75.49 ± 5.72	36.10 ± 1.55	6.33
GGCN	71.14 ± 1.84	55.17 ± 1.58	85.68 ± 6.63	84.86 ± 4.55	86.86 ± 3.29	$\mathbf{37.54\pm1.56}$	2.50
ACMII-GCN	68.46 ± 1.70	51.80 ± 1.50	85.95 ± 5.64	$\mathbf{86.76\pm4.75}$	$\mathbf{87.45\pm3.74}$	36.43 ± 1.20	2.33
GPRGCN	62.59 ± 2.04	46.31 ± 2.46	78.11 ± 6.55	81.35 ± 5.32	82.55 ± 6.23	35.16 ± 0.90	5.17
GCNII	63.86 ± 3.04	38.47 ± 1.58	77.86 ± 3.79	80.39 ± 3.83	77.57 ± 3.40	37.44 ± 1.30	5.83
COCN (OURS)	$\mathbf{79.17\pm0.17}$	$\mathbf{72.95\pm0.23}$	$\mathbf{86.22\pm0.49}$	85.21 ± 0.49	86.88 ± 0.45	36.35 ± 0.12	**1.83**

2020) and Actor (Tang et al., 2009). For Chameleon, Squirrel, Cornell, Texas and Wisconsin, we use the same random train/validation/test splits of 48%/32%/20% as Pei et al.[2] and report average performance over ten splits. The statistics of node classification datasets are summarized in Tab. 2

Baselines. We consider GNNs with different convolution designs which can make node-level predictions, including ChebyNet (Defferrard et al., 2016), GCN (Kipf & Welling, 2017), GraphSAGE(Hamilton et al., 2017), APPNP (Gasteiger et al., 2018), MixHop (Abu-El-Haija et al., 2019), Geom-GCN (Pei et al., 2020), GCNII (Chen et al., 2020), FAGCN (Bo et al., 2021), GGCN (Yan et al., 2022), LINKX (Lim et al., 2022), GPRGCN (Chien et al., 2022) and ACMII-GCN (Luan et al., 2022).

Performance. The results of CoCN and baselines are presented in Tab. 2. CoCN gets competitive results on all six datasets and outperforms other baseline models on Chameleon, Squirrel and Cornell. For Texas, Wisconsin and Actor, CoCN slightly underperforms GGCN and ACMII-GCN. This is because the node connectivity is weak on these datasets, making the structure features less informative for CoCN. However, CoCN still constantly achieves superior performance compared to other models, which proves the efficacy of our model for node classification.

Filtered Datasets. Platonov et al. find that there are many duplicate nodes among the train, validation and test splits in Chameleon and Squirrel. This results in the train-test data leakage that induces GNNs to fit the test splits during the training stage, making the performances on Chameleon and Squirrel become less convincing. To further validate CoCN on Chameleon and Squirrel, we adopt filtered Chameleon and Squirrel datasets that do not contain duplicate nodes following Platonov et al.. The performance comparison between original and filtered datasets is presented in Tab. 3. The results of baselines are from Platonov et al.. The ranks denote the accuracy ranking out of 18 models on original and filtered datasets respectively. We only list four top-ranking baselines here: FSGNN (Maurya et al., 2022) and ResNet+adj (Platonov et al., 2023) for original datasets, GCN and FAGCN for filtered datasets. For full comparison results, please refer to Appendix C.1. As presented in Tab. 3, CoCN consistently outperforms baseline models on both original and filtered datasets. Compared to other models, CoCN is less sensitive to node duplicates and has stronger generalizing ability. The results further validate the reliability of CoCN on Chameleon and Squirrel.

Table 3. **Node Classification Results (measured by accuracy: %) on Original / Filtered Chameleon and Squirrel.**

	Chameleon			Squirrel		
	Ori.	Filtered	Ranks	Ori.	Filtered	Ranks
FAGCN	64.23	41.90	6 / 2	47.63	**41.08**	5 / 1
GCN	50.18	40.89	11 / 4	39.06	39.47	7 / 3
ResNet+adj	71.07	38.67	3 / 13	65.46	38.37	3 / 5
FSGNN	77.85	40.61	2 / 5	68.93	35.92	2 / 11
CoCN (Ours)	**79.61**	**41.95**	1 / 1	**73.27**	39.69	1 / 2

Mini-Batch Classification. CoCN achieves permutation invariance through learnable permutations. Compared to permutation enumeration, the permutation module of CoCN depends on specific tasks to generate the required permutations. As a result, CoCN reduces the time complexity of permutation modeling on n nodes from $O(n!)$ to $O(n^2)$ which enables node permutation on large graphs. However, as the scale of real-world graphs continuously grows, mini-batching becomes vital for scalable graph learning

[2] The original random splits in Pei et al. is 60%/20%/20%, which is different from their open source data splits.

Table 4. **Batching Node Classification Results (amazon-ratings measured by accuracy: %, questions and genius measured by ROC AUC: %).**

	questions	amazon-ratings	genius
Nodes	48,921	24,492	421,961
Edges	153,540	93,050	984,979
Features	301	300	12
GCN	76.09	48.70	87.42
GAT	77.43	49.09	55.80
APPNP	64.49	46.62	85.36
GPRGCN	55.48	44.88	90.05
GCNII	75.17	46.13	90.24
GloGNN	65.74	36.89	90.66
MixHop	58.51	42.06	90.58
LINKX	69.43	44.76	90.77
CoCN (Ours)	77.86	49.05	90.58

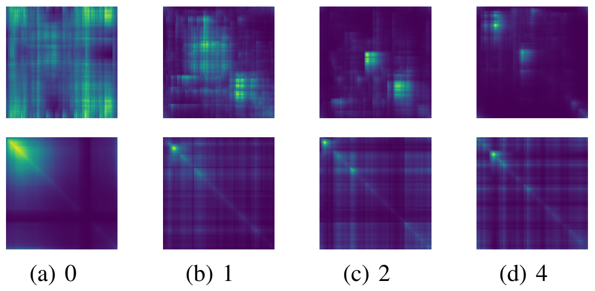

Figure 4. **Permutation Results.** The results are compared under different values of the smoothness parameter t. The first row displays the permuted adjacency matrix and the brighter pixels indicate that the nodes are connected in the adjacency matrix. The second row displays the permuted node feature similarity and the brightness of the pixels indicate the similarity.

(Zeng et al., 2020). To evaluate the potential of CoCN on large-scale graphs, we conduct batching node classification experiments on three datasets including genius (Lim et al., 2022), questions and amazon-ratings (Platonov et al., 2023). We use the mini-batching method Cluster-GCN (Chiang et al., 2019) for CoCN and full-batch training for baselines. The batch size is selected from $\{10, 50, 100\}$ that achieves the best performance on the validation set. CoCN performs comparably to full-batch classification results in Tab. 4. The results show that CoCN still demonstrates promising node classification efficacy under mini-batch training settings. This facilitates its application to large-scale graphs.

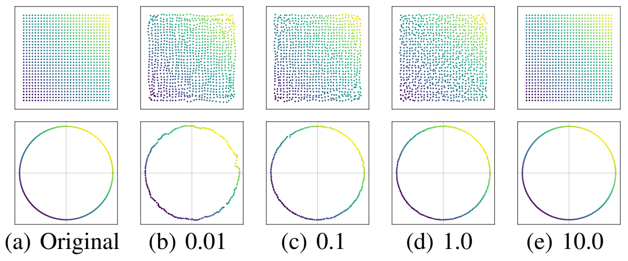

Figure 3. **Permutation Information Loss.** The results under different relaxation factors τ are compared.

5.3. Model Analysis

5.3.1. Permutation Analysis

Information Loss. We conduct graph reconstruction experiments following Bianchi et al. to study whether the approximate permutations cause information loss. Given input features, an autoencoder is trained to recover the original features from the permuted features. The learning objective is to minimize the mean squared error (MSE) between the input features and the recovered features. The input graphs include synthetic graphs (Bianchi et al., 2020) of ring and grid. The input features are the coordinates of nodes in 2D Euclidean space. The original synthetic graphs and the corresponding recovered graphs under different relaxation factors τ are presented in Fig. 3. The autoencoder can recover the original node position with small distortion. As τ increases, the distortion gets reduced. When τ equals 10, the autoencoder can reconstruct the graph with almost no visible distortion. This suggests that our permutation generation module can well approximate the permutation matrix with little information loss.

Smoothness. We compare the permutation matrix $\hat{\mathbf{P}}_t$ with different values of the smoothness parameter t on Chameleon. The relaxation factor τ is set to 0.01. The permuted adjacency matrix $\hat{\mathbf{P}}_t\mathbf{A}\hat{\mathbf{P}}_t^\top$ and the permuted node feature similarity $\hat{\mathbf{P}}_t\mathbf{X}\mathbf{X}^\top\hat{\mathbf{P}}_t^\top$ are visualized in the first row and second row of Fig. 4. When t equals 0, the absolute positions only depend on node features. Nodes with similar features are assigned together. As t increases, the adjacency of graph nodes has more influence on the permutation. Edges in the adjacency matrix, *i.e.*, the brighter pixels in Fig. 4, are permuted close to the main diagonal. Moreover, in Subsection 5.1 and 5.2 we find that the improvement of CoCN is limited for weakly connected datasets, such as MUTAG for graph classification and TEXAS for node classification. We speculate that it is due to the implicit encoding of structure information in Eq. 4. More detailed analysis is provided in Appendix C.2.

Multiple Permutations. We conduct ablation studies on the number of permutations and analyze the convolution module output of different permutations. Fig. 5 presents the classification accuracy under different numbers of permutations on MUTAG, Chameleon, Cornell and Wisconsin. As the number of permutations increases, CoCN gets improvement in accuracy for all the datasets. Following this

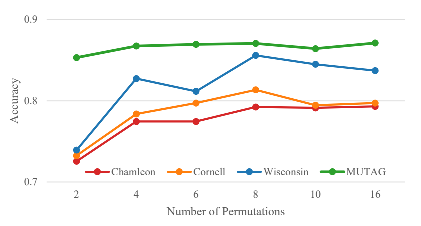

Figure 5. **Ablation Study on the Number of Permutations.**

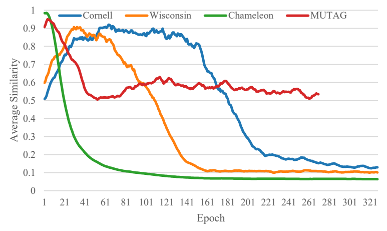

Figure 6. **Feature Similarity between Different Permutations.**

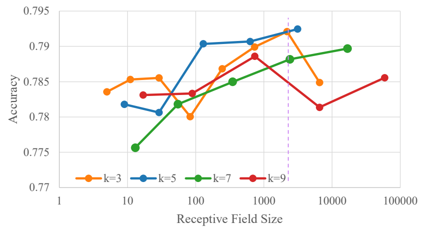

Figure 7. **Ablation Study on Kernel Size and Receptive Field Size.** The dashed line represents the number of input nodes.

result, we further analyze the relationship between different permutations. Fig. 6 presents the average feature similarity between permutations during the training process. Given two feature vectors $\mathbf{x}$ and $\mathbf{y}$, the feature similarity is $\mathbf{x}^\top \mathbf{y}/d$ where d denotes the number of feature channels. The results are normalized into $[0, 1]$ for a unified presentation. From Fig. 6, the model gradually learns to reduce the feature similarity between different permutations. This indicates that CoCN learns complementary feature representations under different permutations and requires multiple permutations for sufficient feature learning.

5.3.2. Network Structure Analysis

We conduct ablation studies on kernel size k and the number of compressed pooling layers L_2 for Chameleon (Fig. 7) and MUTAG (presented in Appendix C.3). As shown in Fig. 7, CoCN achieves better performance as the receptive field size increases. However, if the receptive field size gets larger than the number of input nodes, CoCN cannot extract informative features anymore and therefore shows performance degradation. For different kernel sizes k, CoCN with $k = 5$ has better performance. This is because that larger kernels ($k = 7, 9$) can hardly capture fine-grain features well and smaller kernels ($k = 3$) requires more layers to get the larger receptive field. CoCN mainly relies on compressed pooling layers to get larger receptive fields. For the compressed convolution layer with unit-step, stacking multiple layers generally degrades the model performance. We provide the quantitative results in Appendix C.3.

5.3.3. Feature Analysis

We perform ablation studies on node features and structure features for the datasets described above. Results on Chameleon, Cornell, MUTAG and IMDB-BINARY are presented in Tab. 5. Full results are provided in Appendix C.4. From Tab. 5 we observe that CoCN with both types of features has better performance on most datasets. Therefore, both node features and structure features generally contribute to the model prediction. CoCN can still obtain good performance with only structure features, which indicates its ability in capturing the structure information.

Table 5. **Ablation Study (measured by accuracy: %) on Structure Features and Node Features.**

	Str. Feat.	Nod. Feat.	Both
MUTAG	**88.35**	85.12	87.08
IMDB-B	70.47	75.09	**77.26**
Chameleon	69.14	77.98	**79.17**
Cornell	66.22	79.19	**81.08**

6. Conclusion

We presented a differentiable permutation method for generalizing Euclidean convolution to graphs. We formulated the permutation generation as a node position assignment problem and transformed the generated node positions into approximated permutation matrix with cyclic shift. Based on Euclidean convolution, we designed diagonal convolution for graphs to learn both node features and the corresponding structure features. We further presented Compressed Convolution Network (CoCN), a hierarchical GNN model for graph representation learning. Experiment results show that CoCN achieves superior performance on both node classification and graph classification.

Limitations For homophilous graph datasets, the nodes with short paths in-between tends to be assigned closer to each other after permutation. However, due to the implicit encoding of structure information for the permutation process, this tendency cannot be fully fulfilled. As a result, CoCN performs less promising on graphs with strong homophily and gains limited improvement on weakly connected datasets. We will explore more efficient position regression techniques in future work.

Acknowledgements

This work was supported in part by the National Key R&D Program of China under Grant 2018AAA0102000, in part by National Natural Science Foundation of China: 62022083, 62236008, 61931008 and U21B2038, and in part by ICT Innovative Project: E161060. Z. Xue is sponsored by CCF-Tencent Open Fund: RAGR20220125. We would like to thank anonymous reviewers for helpful suggestions that improved this work.

References

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Steeg, G. V., and Galstyan, A. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *International Conference on Machine Learning*, pp. 21–29, Long Beach, USA, 2019. PMLR.

Baek, J., Kang, M., and Hwang, S. J. Accurate Learning of Graph Representations with Graph Multiset Pooling. In *International Conference on Learning Representations*, virtual, 2022.

Bianchi, F. M., Grattarola, D., and Alippi, C. Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning*, pp. 874–883, virtual, 2020. PMLR.

Bo, D., Wang, X., Shi, C., and Shen, H. Beyond Low-frequency Information in Graph Convolutional Networks. In *AAAI Conference on Artificial Intelligence*, volume 35, pp. 3950–3957, virtual, 2021.

Boguñá, M., Bonamassa, I., De Domenico, M., Havlin, S., Krioukov, D., and Serrano, M. Á. Network geometry. *Nature Reviews Physics*, 3(2):114–135, 2021.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral Networks and Locally Connected Networks on Graphs. In *International Conference for Learning Representations*, Banff, Canada, 2013.

Cangea, C., Veličković, P., Jovanović, N., Kipf, T., and Liò, P. Towards Sparse Hierarchical Graph Classifiers. In *Neural Information Processing Systems Workshop on Relational Representation Learning*, Montréal, Canada, 2018.

Carter, N. C. Visual group theory. pp. 65–66. Mathematical Association of America, 2009.

Chen, H., Perozzi, B., Hu, Y., and Skiena, S. HARP: Hierarchical Representation Learning for Networks. In *AAAI Conference on Artificial Intelligence*, volume 32, New Orleans, Louisiana, USA, 2018. AAAI Press.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and Deep Graph Convolutional Networks. In *International Conference on Machine Learning*, pp. 1725–1735, virtual, 2020. PMLR.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, New York, NY, USA, 2019. Association for Computing Machinery.

Chien, E., Peng, J., Li, P., and Milenkovic, O. Adaptive Universal Generalized PageRank Graph Neural Network. In *International Conference on Learning Representations*, virtual, 2022.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *International Conference on Neural Information Processing Systems*, pp. 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc.

Deng, C., Zhao, Z., Wang, Y., Zhang, Z., and Feng, Z. GraphZoom: A Multi-level Spectral Approach for Accurate and Scalable Graph Embedding. In *International Conference on Learning Representations*, Addis Ababa, Ethiopia, 2020.

Diehl, F. Edge Contraction Pooling for Graph Neural Networks. *arXiv preprint*, arXiv.1905.10990, 2019.

Du, L., Shi, X., Fu, Q., Ma, X., Liu, H., Han, S., and Zhang, D. GBK-GNN: Gated Bi-Kernel Graph Neural Networks for Modeling Both Homophily and Heterophily. In *ACM Web Conference*, pp. 1550–1558, New York, NY, USA, 2022. Association for Computing Machinery.

Eliasof, M., Haber, E., and Treister, E. pathGCN: Learning General Graph Spatial Operators from Paths. In *International Conference on Machine Learning*, pp. 5878–5891, Baltimore, Maryland, USA, 2022. PMLR.

Fey, M. and Lenssen, J. E. Fast Graph Representation Learning with PyTorch Geometric. In *International Conference on Learning Representations Workshop on Graphs and Manifolds*, 2019.

Gao, H. and Ji, S. Graph U-Nets. In *International Conference on Machine Learning*, pp. 2083–2092, Long Beach, California, USA, 2019. PMLR.

Gasteiger, J., Bojchevski, A., and Günnemann, S. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations*, Addis Ababa, Ethiopia, 2018.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *International Conference on Neural Information Processing Systems*, pp. 1025–1035, Red Hook, USA, 2017. Curran Associates Inc.

He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, Las Vegas, NV, USA, 2016.

He, M., Wei, Z., Huang, z., and Xu, H. BernNet: Learning Arbitrary Graph Spectral Filters via Bernstein Approximation. In *Advances in Neural Information Processing Systems*, volume 34, pp. 14239–14251, virtual, 2021. Curran Associates, Inc.

Hu, F., Zhu, Y., Wu, S., Wang, L., and Tan, T. Hierarchical graph convolutional networks for semi-supervised node classification. In *International Joint Conference on Artificial Intelligence*, pp. 4532–4539, Macao, China, 2019. AAAI Press.

Hu, Y. Efficient, high-quality force-directed graph drawing. *Mathematica journal*, 10:37–71, 2006.

Huang, Z., Wang, Y., Li, C., and He, H. Going Deeper into Permutation-Sensitive Graph Neural Networks. In *International Conference on Machine Learning*, pp. 9377–9409, Baltimore, Maryland, USA, 2022. PMLR.

Ji, S., Xu, W., Yang, M., and Yu, K. 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *International Conference for Learning Representations*, San Diego, USA, 2015. Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, Toulon, France, 2017.

Klicpera, J., Weiß enberger, S., and Günnemann, S. Diffusion Improves Graph Learning. In *International Conference on Advances in Neural Information Processing Systems*, volume 32, pp. 13333–13345, Vancouver, Canada, 2019. Curran Associates, Inc.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Lee, J., Lee, I., and Kang, J. Self-Attention Graph Pooling. In *International Conference on Machine Learning*, pp. 3734–3743, Long Beach, California, USA, 2019. PMLR.

Li, Q., Han, Z., and Wu, X.-M. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI Conference on Artificial Intelligence*, New Orleans, USA, 2018.

Li, X., Zhu, R., Cheng, Y., Shan, C., Luo, S., Li, D., and Qian, W. Finding Global Homophily in Graph Neural Networks When Meeting Heterophily. In *International Conference on Machine Learning*, volume 162, pp. 13242–13256, Baltimore, Maryland, USA, 2022. PMLR.

Lim, D., Hohne, F. M., Li, X., Huang, S. L., Gupta, V., Bhalerao, O. P., and Lim, S.-N. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods. In *Advances in Neural Information Processing Systems*, pp. 20887–20902, virtual, 2022.

Luan, S., Hua, C., Lu, Q., Zhu, J., Zhao, M., Zhang, S., Chang, X.-W., and Precup, D. Revisiting Heterophily For Graph Neural Networks. In *Advances in Neural Information Processing Systems*, virtual, 2022.

Ma, Z., Xuan, J., Wang, Y. G., Li, M., and Liò, P. Path Integral Based Convolution and Pooling for Graph Neural Networks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 16421–16433, virtual, 2020. Curran Associates, Inc.

Maurya, S. K., Liu, X., and Murata, T. Simplifying approach to node classification in Graph Neural Networks. *Journal of Computational Science*, 62:101695, 2022.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. TUDataset: A collection of benchmark datasets for learning with graphs. In *International Conference on Machine Learning Workshop on Graph Representation Learning and Beyond*, 2020.

Murphy, R. L., Srinivasan, B., Rao, V., and Ribeiro, B. Janossy Pooling: Learning Deep Permutation-Invariant Functions for Variable-Size Inputs. In *International Conference on Learning Representations*, New Orleans, LA, USA, 2019a.

Murphy, R. L., Srinivasan, B., Rao, V., and Ribeiro, B. Relational Pooling for Graph Representations. In *Internet Conference on Machine Learning*, pp. 4663–4673, Long Beach, California, USA, 2019b. PMLR.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, volume 48, pp. 2014–2023, New York, NY, USA, 2016. PMLR.

Noh, H., Hong, S., and Han, B. Learning Deconvolution Network for Semantic Segmentation. In *IEEE International Conference on Computer Vision*, pp. 1520–1528, Santiago, Chile, 2015. IEEE Computer Society.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *International Conference on Neural Information Processing Systems*, pp. 8026–8037, Red Hook, NY, USA, 2019. Curran Associates Inc.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*, Addis Ababa, Ethiopia, 2020.

Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *International Conference on Learning Representations*, Kigali, Rwanda, 2023.

Ranjan, E., Sanyal, S., and Talukdar, P. ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations. In *AAAI Conference on Artificial Intelligence*, volume 34, pp. 5470–5477, New York, NY, USA, 2020.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-Scale attributed node embedding. *Journal of Complex Networks*, 9(2), 2021.

Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., and Sun, Y. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. In *International Joint Conference on Artificial Intelligence*, volume 2, pp. 1548–1554. ijcai.org, 2021.

Tang, J., Sun, J., Wang, C., and Yang, Z. Social influence analysis in large-scale networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 807–816, New York, NY, USA, 2009. Association for Computing Machinery.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. Learning Spatiotemporal Features with 3D Convolutional Networks. In *IEEE International Conference on Computer Vision*, pp. 4489–4497, Santiago, Chile, 2015. IEEE Computer Society.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. In *International Conference on Learning Representations*, Vancouver, Canada, 2018.

Wang, X. and Zhang, M. How Powerful are Spectral Graph Neural Networks. In *International Conference on Machine Learning*, pp. 23341–23362, Baltimore, Maryland, USA, 2022. PMLR.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning*, pp. 6861–6871, Long Beach, USA, 2019. PMLR.

Wu, J., Chen, X., Li, S., and Xu, K. Structural entropy guided graph hierarchical pooling. In *International Conference on Machine Learning*, volume 162, pp. 24017–24030, Baltimore, Maryland, USA, 2022. PMLR.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, New Orleans, LA, USA, 2019.

Yan, Y., Hashemi, M., Swersky, K., Yang, Y., and Koutra, D. Two Sides of the Same Coin: Heterophily and Oversmoothing in Graph Convolutional Neural Networks. In *IEEE International Conference on Data Mining*, pp. 1287–1292, Orlando, FL, USA, 2022. IEEE.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*, volume 31, pp. 4805–4815, Montréal, Canada, 2018. Curran Associates, Inc.

Yuan, H. and Ji, S. StructPool: Structured Graph Pooling via Conditional Random Fields. In *International Conference on Learning Representations*, Addis Ababa, Ethiopia, 2020.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *International Conference on Learning Representations*, Addis Ababa, Ethiopia, 2020.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. In *International Conference on Neural Information Processing Systems*, pp. 7793–7804, Red Hook, NY, USA, 2020. Curran Associates Inc.

Zhu, J., Rossi, R. A., Rao, A., Mai, T., Lipka, N., Ahmed, N. K., and Koutra, D. Graph neural networks with heterophily. In *AAAI Conference on Artificial Intelligence*, pp. 11168–11176, virtual, 2021. AAAI Press.

A. Proofs

A.1. Proof of Proposition 3.1

Proposition 3.1. (Permutation Convergence) $\hat{\mathbf{P}}$ *converges to standard permutation matrix as relaxation factor* τ *approaching positive infinity:* $\lim_{\tau \to +\infty} \hat{\mathbf{P}} = \mathbf{P}, \mathbf{P} \in \mathcal{P}$.

Proof. Assuming that we have absolute position $\mathbf{r} \in \mathbb{R}^n$ given by Eq. 2 where $\texttt{sgn}(\cdot) \in \{0, 1\}$, $\mathbf{r}$ is a integer vector and $\mathbf{r}_i \in [0, n-1]$. Let $\mathbf{m}_P = (\mathbf{m}\mathbf{1}^\top - \mathbf{1}\mathbf{r}^\top + n) \pmod n$ and $\hat{\mathbf{P}} = \texttt{exp}(-\tau \mathbf{m}_P)$. For every row of $\mathbf{m}_P$, there always exists an entry equal to 0 where the corresponding entry in $\hat{\mathbf{P}}$ constantly equals 1. The rest of entries are integers belonging to $[1, n-1]$ and the corresponding entries in $\hat{\mathbf{P}}$ converge to 0 as τ approaches positive infinity. □

A.2. Proof of Proposition 3.2

Proposition 3.2. (Permutation Invariant Input) *Let* f *be a permutation equivariant function such that for any* $\mathbf{P} \in \mathcal{P}$, $f(\mathbf{PX}, \mathbf{PAP}^\top) = \mathbf{P}f(\mathbf{X}, \mathbf{A})$. *Then given* $\hat{\mathbf{P}} = \texttt{PERM}(\mathbf{X}, \mathbf{A})$, $\hat{\mathbf{X}} = \hat{\mathbf{P}}\mathbf{X}$, *and* $\hat{\mathbf{A}} = \hat{\mathbf{P}}\mathbf{A}\hat{\mathbf{P}}^\top$, *for any* $\mathbf{P} \in \mathcal{P}$ *on* $\mathbf{X}$ *and* $\mathbf{A}$, $\hat{\mathbf{X}}$ *and* $\hat{\mathbf{A}}$ *are invariant.*

Proof. We first prove that $\texttt{PERM}(\cdot)$ is permutation equivariant. Given $\mathbf{PX}$ and $\mathbf{PAP}^\top$ for any $\mathbf{P} \in \mathcal{P}$, the output of f can be written as $\mathbf{Pr}_A = f(\mathbf{PX}, \mathbf{PAP}^\top)$. Substituting $\mathbf{Pr}_A$ into Eq. 2 gives:

$$\begin{aligned}
&\texttt{sgn}\left(\mathbf{Pr}_A\mathbf{1}^\top - \mathbf{1}\mathbf{r}_A^\top\mathbf{P}^\top\right)\mathbf{1} \\
=&\texttt{sgn}\left(\mathbf{Pr}_A\mathbf{1}^\top\mathbf{P}^\top - \mathbf{P}\mathbf{1}\mathbf{r}_A^\top\mathbf{P}^\top\right)\mathbf{1} \\
=&\mathbf{P}\texttt{sgn}\left(\mathbf{r}_A\mathbf{1}^\top - \mathbf{1}\mathbf{r}_A^\top\right)\mathbf{P}^\top\mathbf{1} \\
=&\mathbf{P}\texttt{sgn}\left(\mathbf{r}_A\mathbf{1}^\top - \mathbf{1}\mathbf{r}_A^\top\right)\mathbf{1} \\
=&\mathbf{Pr}
\end{aligned} \tag{A.11}$$

Therefore, Eq. 2 is permutation equivariant. Further substituting $\mathbf{Pr}$ into Eq. 3 gives:

$$\begin{aligned}
&\texttt{exp}\left\{-\tau\left[\left(\mathbf{m}\mathbf{1}^\top - \mathbf{1}\mathbf{r}^\top\mathbf{P}^\top + n\right) \pmod n\right]\right\} \\
=&\texttt{exp}\left\{-\tau\left[\left(\mathbf{m}\mathbf{1}^\top\mathbf{P}^\top - \mathbf{1}\mathbf{r}^\top\mathbf{P}^\top + n\right) \pmod n\right]\right\} \\
=&\texttt{exp}\left\{-\tau\left[\left(\mathbf{m}\mathbf{1}^\top - \mathbf{1}\mathbf{r}^\top + n\right) \pmod n\right]\right\}\mathbf{P}^\top \\
=&\hat{\mathbf{P}}\mathbf{P}^\top
\end{aligned} \tag{A.12}$$

Hence $\texttt{PERM}(\cdot)$ is permutation equivariant. For any $\mathbf{P} \in \mathcal{P}$, we can use the generated permutation matrix $\hat{\mathbf{P}}\mathbf{P}^\top$ to permute the input features where $\hat{\mathbf{P}}\mathbf{P}^\top(\mathbf{PX}) = \hat{\mathbf{P}}\mathbf{X}$ and $\hat{\mathbf{P}}\mathbf{P}^\top(\mathbf{PAP}^\top)\mathbf{P}\hat{\mathbf{P}}^\top = \hat{\mathbf{P}}\mathbf{A}\hat{\mathbf{P}}^\top$. Therefore, given $\hat{\mathbf{P}} = \texttt{PERM}(\mathbf{X}, \mathbf{A})$, $\hat{\mathbf{X}} = \hat{\mathbf{P}}\mathbf{X}$ and $\hat{\mathbf{A}} = \hat{\mathbf{P}}\mathbf{A}\hat{\mathbf{P}}^\top$, $\hat{\mathbf{X}}$ and $\hat{\mathbf{A}}$ are invariant for any $\mathbf{P} \in \mathcal{P}$ on $\mathbf{X}$ and $\mathbf{A}$. □

B. Details for Experiments

B.1. Graph Classification

Datasets. We use six datasets (Morris et al., 2020) for graph classification experiments including three biochemical datasets MUTAG, PROTEINS and NCI1, and three social network datasets COLLAB, IMDB-BINARY and IMDB-MULTI. MUTAG is a collection of nitroaromatic compounds. It includes 188 graphs of chemical compounds labeled with their mutagenicity on Salmonella typhimurium. PROTEINS is a collection of 1,113 proteins with amino acids as nodes. The goal is to predict whether the protein graphs are enzymes or not. NCI1 is collected by National Cancer Institute (NCI) with 4,110 graphs of chemical compounds. The graphs are labeled by whether the corresponding compounds are positive or negative for cell lung cancer. All three biochemical datasets use the one-hot node labels as node features. IMDB-BINARY and IMDB-MULTI are two movie actor/actress collaboration network datasets labeled by the genre of the movies. COLLAB is a scientific collaboration network dataset. Each graph represents a researcher's ego network with collaborators and is labeled by the scientific study field of the researcher.

Table A.6. **Detailed Experimental Setup For Node Classification.**

	CHAMELEON	SQUIRREL	CORNELL	TEXAS	WISCONSIN	ACTOR
t	1	1	0	0	0	0
L_1	1	2	1	1	1	2
L_2	4	4	3	3	3	4
DROPOUT	0.1	0.1	0.5	0.5	0.5	0.3

Experimental Setup. For experiments on all datasets, the learning rate is set to 1×10^{-4}, the hidden size is set to 64, and the dropout rate is set to 0.5. For Adam (Kingma & Ba, 2015) optimizer, weight decay is set $\in \{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}\}$. We also use early stopping regularization, where we stop the training if there is no further reduction in the validation loss during 100 epochs. The maximum epoch number is set to 200. The batch size is set to 4 on MUTAG and PROTEINS, 8 on NCI1, IMDB-BINARY and IMDB-MULTI, and 32 on COLLAB. We follow the network structure in Fig. 2(a) to construct CoCN. For all datasets, the smoothness parameter t in Eq. 4 is set $\in \{6, 8\}$, the number of permutations is set to 8, the relaxation factor τ is set to 0.1, L_1 and L_2 are set to 1, the kernel size is set $\in \{5, 7, 9\}$.

B.2. Node Classification

Datasets. For node classification, we use six datasets including Chameleon, Squirrel (Rozemberczki et al., 2021), Cornell, Texas, Wisconsin (Pei et al., 2020) and Actor (Tang et al., 2009). Chameleon and Squirrel are collected from Wikipedia. Each graph corresponds to a topic in Wikipedia. The nodes represent Web pages with keywords of the pages as node features. The edges are links between Web pages. Based on the average monthly traffic, the nodes are classified into five classes. Cornell, Texas and Wisconsin are Web page datasets collected from computer departments of different universities. Actor is an actor-actor relation dataset with nodes representing actors. For each node, the features are the keywords of the corresponding actor on Wikipedia. Two nodes are connected if they appear on the same page.

Experimental Setup. For experiments on all datasets, the learning rate is set to 1×10^{-4}, the hidden size is set $\in \{64, 128\}$, and the weight decay is set $\in \{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}\}$. The maximum epoch number is set to 500. We stop the training if there is no further reduction in the validation loss during 150 epochs. We follow the network structure in Fig. 2(b) to construct CoCN. The number of permutations is set $\in \{8, 10\}$, the relaxation factor τ is set to 0.1 and the kernel size is set to 5. The rest of the hyper-parameter setting and detailed network structure for all datasets are summarized in Tab. A.6.

B.3. Model Analysis

For all the experiments in Subsection 5.3, the number of permutations is set to 8, the learning rate is set to 1×10^{-4}, and the hidden size is set to 64. For the permutation information loss experiment, we construct the autoencoder with the permutation generation module. The position smoothness parameter t is set to 6 and the dropout rate is set to 0, the maximum epoch is set to $10,000$ and the early stopping patience is set to $2,000$ epochs. For the rest of the experiments, the experimental setup is the same as the classification experiments except for the ablation items.

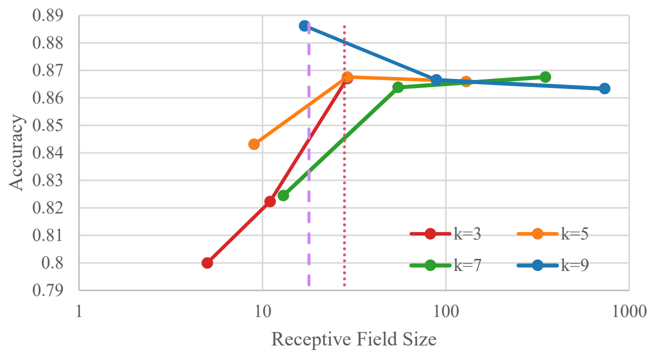

Figure A.8. **Ablation Study on the Kernel Size and Receptive Field Size for MUTAG.** The dashed line represents the average number of nodes for all graphs. The dotted line represents the maximum number of nodes in one graph.

Table A.7. **Full Comparison Results (measured by accuracy: %) on Original / Filtered Chameleon and Squirrel.**

	CHAMELEON			SQUIRREL		
	ORIGINAL	FILTERED	RANKS	ORIGINAL	FILTERED	RANKS
RESNET	49.52	36.73	13 / 15	33.88	36.55	13 / 8
RESNET+SGC	49.93	41.01	12 / 3	34.36	38.36	12 / 6
RESNET+ADJ	71.07	38.67	3 / 13	65.46	38.37	3 / 5
GCN	50.18	40.89	10 / 4	39.06	39.47	7 / 3
GRAPHSAGE	50.18	37.77	10 / 14	35.83	36.09	10 / 10
GAT	45.02	39.21	17 / 10	32.21	35.62	15 / 12
GAT-SEP	50.24	39.26	9 / 9	35.72	35.46	11 / 14
GT	44.93	38.87	18 / 12	31.61	36.30	16 / 9
GT-SEP	50.33	40.31	8 / 6	36.08	36.66	9 / 7
H_2GCN	46.27	26.75	16 / 17	29.45	35.10	18 / 16
CPGNN	48.77	33.00	14 / 16	30.91	30.04	17 / 17
GPRGNN	47.26	39.93	15 / 7	33.39	38.95	14 / 4
FSGNN	77.85	40.61	2 / 5	68.93	35.92	2 / 11
GLOGNN	70.04	25.90	4 / 18	61.21	35.11	4 / 15
FAGCN	64.23	41.90	6 / 2	47.63	**41.08**	5 / 1
GBK-GNN	51.36	39.61	7 / 8	37.06	35.51	8 / 13
JACOBICONV	68.33	39.00	5 / 11	46.17	29.71	6 / 18
COCN (OURS)	**79.61**	**41.95**	1 / 1	**73.27**	39.69	1 / 2

Table A.8. **Ablation Study on Smoothness Parameter t (measured by accuracy: %).**

	0	1	2	4	6	8
CHAMELEON	49.45	**78.39**	77.26	76.63	75.77	74,04
CORNELL	**82.16**	75.95	76.49	75.95	75.14	72.43
MUTAG	85.09	85.94	85.84	86.62	86.51	**87.19**

C. Additional Results

C.1. Node Classification on Filtered Datasets

Following Platonov et al., we further conduct node classification experiments on filtered Chameleon and Squirrel. Baselines include: GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), SGC (Wu et al., 2019), H_2GCN (Zhu et al., 2020), GT (Shi et al., 2021), CPGNN (Zhu et al., 2021), FAGCN (Bo et al., 2021), GPRGNN (Chien et al., 2022), FSGNN (Maurya et al., 2022), GloGNN (Li et al., 2022), GBK-GNN (Du et al., 2022), JacobiConv (Wang & Zhang, 2022) and ResNet with GNN methods (Platonov et al., 2023). Platonov et al. also separate ego- and neighbor-embeddings in the GNN aggregation step of GAT and GT, which is denoted as *sep*. All the results of baselines are from Platonov et al.. Full comparison results are presented in Tab. A.7. It shows that CoCN consistently outperforms other models on both the original and filtered datasets, which further validates the reliability of CoCN.

C.2. Permutation Analysis

We conduct ablation studies on the smoothness parameter t in the permutation generation module. The results for Chameleon, Cornel and MUTAG are presented in Tab. A.8. For node classification datasets, the optimal smoothness parameter t is small, *i.e.*, 1 for Chameleon and 0 for Cornell. The optimal value of t for graph classification dataset MUTAG is 8, which is larger than that on node classification datasets. This indicates that graph classification requires representation learning from node clusters with strong connections while node classification concentrates on nodes clustered by features.

We also find that the classification performance of CoCN is limited on the weakly connected datasets. The average degrees for all datasets are summarized in Tab. A.9. CoCN slightly underperforms one or two baseline models on MUTAG, Texas, Wisconsin and Actor which has a lower average degree than other datasets. To explain this, we speculate that it is due to the implicit encoding of structure information in permutation generation in Eq. 4. We leave the further improvement of this module to future work.

Table A.9. **Statistics on Average Degree.**

	MUTAG	PROTEINS	NCI1	IMDB-B	IMDB-M	COLLAB
AVG DEGREE	4	7	4	39	40	263
	CHAMELEON	SQUIRREL	CORNELL	TEXAS	WISCONSIN	ACTOR
AVG DEGREE	15.85	41.74	1.63	1.78	2.05	3.95

Table A.10. **Ablation Study on L_1 (measured by accuracy: %).**

	1	2	3	4	6	8	10
CHAMELEON	**79.25**	78.86	78.81	79.08	78.64	77.94	77.74
MUTAG	**87.08**	86.06	86.60	85.27	84.58	84.79	84.62

C.3. Network Structure Analysis

From Fig. A.8, CoCN also gains improvement on MUTAG as the size of the receptive field increases, and the improvement is bounded by the maximum number of nodes in a single graph. Since the maximum number of nodes is small in MUTAG, the number of compressed pooling layers can only be set to 1 with kernel size $\{5, 7, 9\}$. Among these configurations, CoCN with kernel size 9 has the largest receptive field and shows the best performance.

For ablation studies on the number of compressed convolution layers with unit-step which is denoted as L_1, the full results are presented in Tab. A.10. We can see that on both Chameleon and MUTAG, CoCN achieves the best performance on $L_1 = 1$. This is because the compressed convolution layer cannot expand the size of the receptive field efficiently while stacking multiple compressed convolution layers increases the difficulty of training.

C.4. Feature Analysis

The full ablation study results on structure features and node features are presented in Tab. A.11 and Tab. A.12. Compared to baseline models in Tab. 1 and Tab. 2, CoCN can achieve comparable or even better performance with only structure features which indicates that CoCN can capture structure information. CoCN with both node features and structure features has better performance. Therefore, both kinds of features contribute to the classification.

Table A.11. **Ablation Study on Structure Features and Node Features for Graph Classification Datasets (measured by accuracy: %).**

	MUTAG	PROTEINS	NCI1	IMDB-B	IMDB-M	COLLAB
STR. FEAT.	**88.35**	72.03	64.08	70.47	48.16	78.14
NOD. FEAT.	85.12	76.22	78.65	76.17	54.03	84.46
BOTH	87.08	**76.86**	**82.28**	**77.26**	**56.32**	**86.15**

Table A.12. **Ablation Study on Structure Features and Node Features for Node Classification Datasets (measured by accuracy: %).**

	CHAMELEON	SQUIRREL	CORNELL	TEXAS	WISCONSIN	ACTOR
STR. FEAT.	67.82	72.79	66.22	72.16	79.23	33.73
NOD. FEAT.	77.98	71.91	79.19	79.41	83.53	34.36
BOTH	**79.17**	**72.95**	**81.08**	**80.54**	**84.90**	**35.55**