Abstract

Graph classification is an important learning task for graph-structured data. Graph neural networks (GNNs) have recently gained growing attention in graph learning and shown significant improvements on many important graph problems. Despite their state-of-the-art performances, existing GNNs only use local information from a very limited neighborhood around each node, suffering from loss of multimodal information and overheads of excessive computation. To address these issues, we propose a novel Tensor-view Topological Graph Neural Network (TTG-NN), a class of simple yet effective topological deep learning built upon persistent homology, graph convolution, and tensor operations. This new method incorporates tensor learning to simultaneously capture Tensor-view Topological (TT), as well as Tensor-view Graph (TG) structural information on both local and global levels. Computationally, to fully exploit graph topology and structure, we propose two flexible TT and TG representation learning modules that disentangle feature tensor aggregation and transformation, and learn to preserve multi-modal structure with less computation. Theoretically, we derive high probability bounds on both the out-of-sample and in-sample mean squared approximation errors for our proposed Tensor Transformation Layer (TTL). Real data experiments show that the proposed TTG-NN outperforms 20 state-of-the-art methods on various graph benchmarks.

1 Introduction

Graph data are ubiquitous: many real-world objects can be represented by graphs, such as images, text, molecules, social networks, and power grids. Tremendous advances on graph analysis have been achieved in recent years, especially in the field of machine learning (ML) and deep learning (DL) Defferrard et al. [2016], Bronstein et al. [2017], Zhang et al. [2020]. In particular, graph neural networks (GNNs) have emerged as effective architectures for various prediction problems, e.g., node classification Kipf and Welling [2017], Veličković et al. [2018], link prediction Zhang and Chen [2018], Chen et al. [2022a], graph classification Xu et al. [2018], Ying et al. [2018], and spatio-temporal forecast Guo et al. [2019], Zhao et al. [2019], Bai et al. [2020]. GNNs are neural network architectures specifically designed to handle graph-structured data. The fundamental idea behind GNNs involves treating the underlying graph as a computation graph and leveraging neural network primitives to generate node embeddings. This process involves transforming, propagating, and aggregating node features and graph structural information throughout the graph. However, most GNNs follow a neighborhood aggregation process where the feature vector of each node is computed by recursively aggregating and transforming the representation vectors of its neighbors. Consequently, they are unable to capture higher-order relational structures and local topological information concealed in the graph, which are highly relevant for applications that rely on connectivity information You et al. [2018], Huang et al. [2020], Sun et al. [2022]. For instance, understanding the behavior and properties of molecules and protein data within the drug discovery and development process requires capturing crucial information, such as higher-order interactions between atoms, the triangular mesh of a protein surface, and ring-ring interactions within a molecule, etc.

To address these challenges, the integration of ML/DL methods with persistent homology (PH) representations of learned objects have been intensively stud-
A naive aggregation of multiple feature tensors will increase the complexity of NN and incur excessive computational costs. We carefully design a module of Tensor Transformation Layers (TTL) which employs tensor low-rank decomposition to address the model complexity and computation issues. By smartly combining the three modules of TT-CL, TC-GL, and TTL, we can safely incorporate multiple topological and graph features without losing any potential discriminant features, and, at the same time, enjoy a parsimonious NN architecture from the low-rankness of the input feature tensors. The advantages of our TTG-NN are validated both theoretically by Theorem 3.6 and empirically through our extensive experiments.

In short, our main contributions are as follows: (1) This is the first approach bridging tensor methods with an aggregation of multiple features constructed by persistent homology and graph convolution. (2) We provide the first non-asymptotic error bounds of both in-sample and out-of-sample mean squared errors of TTL with Tucker-low-rank feature tensors. (3) Our extensive experiments of TTG-NN on graph classification tasks show that TTG-NN delivers state-of-the-art classification performance with a notable margin, and demonstrates high computational efficiency.

1.1 Related Work

Graph Neural Networks. Recently, Graph Neural Network (GNN) has emerged as a primary tool for graph classification. Zhou et al. [2020], Xia et al. [2021], Zhang et al. [2022], Chikwendu et al. [2023]. Different methods have been proposed to capture the structural and semantic properties of graphs. For instance, Weisfeiler–Lehman (WL) Shepherd et al. [2011] proposes an efficient family of kernels for large graphs with discrete node labels, and Shortest Path Hash Graph Kernel (HGK-SP) Morris et al. [2016] derives kernels for graphs with continuous attributes from discrete ones. Graph Convolutional Network (GCN) Kipf and Welling [2017] extends the convolution operation from regular grids to graphs. To handle large-scale graphs, Top-K pooling operations Cangea et al. [2018], Gao and Ji [2019] design a pooling method by using node features and local structural information to propagate only the top-K nodes with the highest scores at each pooling step. To leverage topological information, Topological Graph Neural Networks (TOGL) Horn et al. [2021] proposes a layer that incorporates global topological information of a graph using persistent homology and can be integrated into any type of GNN. A common limitation is that they fail to accurately capture higher-order and local topological properties of graphs or incorporate rich structure information both in local and global domains.

Tensor-input Neural Networks. Neural Networks that take tensors as inputs are designed to process and analyze data in a tensor format, allowing for the efficient...
processing of high-dimensional data. A tensor analysis on the expressive power of deep neural networks [Cohen et al. 2010] derives a deep network architecture based on arithmetic circuits that inherently employs locality, sharing and pooling, and establishes an equivalence between neural networks and hierarchical tensor factorizations. Tensor Contraction Layer (TCL) [Kossaifi et al. 2017] incorporates tensor contractions as end-to-end trainable neural network layers, regularizes networks by imposing low-rank constraints on the activations, and demonstrates significant model compression without significant impact on accuracy. Tensor Regression Layer (TRL) [Kossaifi et al. 2020] further regularizes networks by regression weights, and reduces the number of parameters while maintaining or increasing accuracy. Graph Tensor Network (GTN) [Xu et al. 2023] introduces a Tensor Network-based framework for describing neural networks through tensor mathematics and graphs for large and multi-dimensional data. In conclusion, current Neural Networks with tensor inputs lack both theoretical and empirical in-depth study.

2 Preliminaries

Problem Setting Let $G = (V, E, X)$ be an attributed graph, where $V$ is a set of nodes ($|V| = N$), $E$ is a set of edges, and $X \in \mathbb{R}^{N \times F}$ is a feature matrix of nodes (here $F$ is the dimension of the node features). Let $A \in \mathbb{R}^{N \times N}$ be a symmetric adjacency matrix whose entries are $a_{ij} = \omega_{ij}$ if nodes $i$ and $j$ are connected and 0 otherwise (here $\omega_{ij}$ is an edge weight and $\omega_{ij} \equiv 1$ for unweighted graphs). Furthermore, $D$ represents the degree matrix of $A$, that is $d_{ii} = \sum_j a_{ij}$. In the graph classification setting, we have a set of graphs $\{G_1, G_2, \ldots, G_k\}$, where each graph $G_i$ is associated with a label $y_i$. The goal of the graph classification task is to take the graph as the input and predict its corresponding label.

Persistent Homology Persistent Homology (PH) is a subfield of algebraic topology which provides a way for measuring topological features of shapes and functions. These shape patterns represent topological properties such as 0-dimensional topological features (connected components), 1-dimensional topological features (cycles), 2-dimensional topological features (voids), and, in general, $q$-dimensional “holes” represent the characteristics of the graph $G$ that remain preserved at different resolutions under continuous transformations (where $q = \{0, 1, \ldots, Q\}$ and $Q$ denotes the maximum dimension of the simplicial complex). Through the use of this multi-resolution scheme, PH tackles the inherent restrictions of traditional homology, enabling the extraction of latent shape characteristics of $G$ which may play an essential role in a given learning task. The key is to select a suitable scale parameter $\epsilon$ and then to study changes in the shape of $G$ that occur as $G$ evolves to $\epsilon$. Thus, given an increasing sequence $\epsilon_1 < \cdots < \epsilon_n$, we no longer study $G$ as a single object but as a filtration $G_{\epsilon_1} \subseteq \cdots \subseteq G_{\epsilon_n} = G$. To ensure that the process of pattern selection and count are objective and efficient, we build an abstract simplicial complex $\mathcal{C}(G_{\epsilon_i})$ on each $G_{\epsilon_i}$, which results in filtration of complexes $\mathcal{C}(G_{\epsilon_1}) \subseteq \cdots \subseteq \mathcal{C}(G_{\epsilon_n})$. For instance, we consider a function on a node set $V$. That is, we choose a very simple filtration based on the node degree, i.e., the number of edges that are incident to a node $u \in V$, and get a descriptor function (i.e., filtration function) $f(u) = \deg(u)$. When scanning $G$ via the degree-based filtration function $f$, it results in a sequence of induced subgraphs of $G$ with a maximal degree of $\epsilon_j$ for each $j \in \{1, \ldots, n\}$. A standard descriptor of the above topological evolution is Persistence Diagram (PD) [Barronikov 1994] $D_g = \{(b_\rho, d_\rho) \in \mathbb{R}^2 | b_\rho < d_\rho\}$, which is a multi-set of points in $\mathbb{R}^2$. Each persistence point $(b_\rho, d_\rho)$ corresponds to the lifespan (i.e., $d_\rho - b_\rho$) of one topological feature, where $b_\rho$ and $d_\rho$ represent the birth and death time of the topological feature $\rho$.

3 Methodology: Tensor-view Topological Graph Neural Network

In this section, we introduce our Tensor-view Topological Graph Neural Network, dubbed as TTG-NN. Our proposed TTG-NN framework is summarized in Figure 1. As illustrated in Figure 1, our method consists of two components. First, tensor-view topological features are extracted by multi-filtrations from multiple views of a graph, and then we design a tensor-view topological representation learning module (Top) for embedding tensor-view local topological features into a high-dimensional space. Second, we develop a tensor-view graph convolutional module (Bottom) on a graph to generate a global shape descriptor.
3.1 Tensor-view Topological Convolutional Layers (TT-CL)

Our first representation learning module utilizes multiple topological features simultaneously by combining the persistent homology and the proposed tensor learning method. To capture the underlying topological features of a graph $\mathcal{G}$, we employ $K$ vertex filtration functions: $f_i : \mathcal{V} \mapsto \mathbb{R}$ for $i \in \{1, \ldots, K\}$. Each filtration function $f_i$ gradually reveals one specific topological structure at different levels of connectivity, such as the number of relations of a node (i.e., degree centrality score), node flow information (i.e., betweenness centrality score), information spread capability (i.e., closeness centrality score), and other node centrality measurements. With each filtration function $f_i$, we construct a set of $Q$ persistence images of resolution $P \times P$ using tools in persistent homology analysis.

Combining $Q$ persistence images of resolution $P \times P$ from $K$ different filtration functions, we construct a tensor-view topological representation, namely Persistent Image (PI) Tensor $\mathbf{Z}_{\text{PIT}}$ of dimension $K \times Q \times P \times P$. We design the Tensor-view Topological Convolutional Layer (TT-CL) to (i) jointly extract and learn the latent topological features contained in the $\mathbf{Z}_{\text{PIT}}$, (ii) leverage and preserve the multi-modal structure in the $\mathbf{Z}_{\text{PIT}}$, and (iii) capture the structure in trainable weights (with fewer parameters). Firstly, hidden representations of the PI tensor $\mathbf{Z}_{\text{PIT}}$ are encoded by a combination of a CNN-based neural network and global pooling layers. Mathematically, we obtain a learnable topological tensor representation defined by

$$
\mathbf{X}^{\text{PIT}} = \begin{cases} f_{\text{CNN}}(\mathbf{Z}_{\text{PIT}}) & \text{if } |Q| = 1 \\ \xi_{\text{POOL}}(f_{\text{CNN}}(\mathbf{Z}_{\text{PIT}})) & \text{if } |Q| > 1 \end{cases},
$$

where $f_{\text{CNN}}$ is a CNN-based neural network, $\xi_{\text{POOL}}$ is a pooling layer that preserves the information of the input in a fixed-size representation (in general, we consider either global average pooling or global max pooling). Equation (1) provides two simple yet effective methods to extract learnable topological features: (i) if only considering $q$-dimensional topological features in $\mathbf{Z}_{\text{PIT}}$, we can apply any CNN-based model to learn the latent feature of the $\mathbf{Z}_{\text{PIT}}$; (ii) considering topological features with $Q$ dimensions, we can additionally employ a global pooling layer over the latent feature and obtain an image-level feature. Secondly, $\mathbf{X}^{\text{PIT}}$ is fed into our Tensor Transformation Layer (TTL) as $\mathbf{H}^{(0)} = \mathbf{X}^{\text{PIT}}$, whose $\ell$-th layer is defined in (4). The output of TTL in TT-CL is denoted as $\mathbf{X}'^{\text{PIT}}$, which captures the local topological information of a graph.

3.2 Tensor-view Graph Convolutional Layers (TG-CL)

Parallel to the TT-CL is our second representation learning module, Tensor-view Graph Convolutional Layer (TG-CL). It utilizes the graph structure of $\mathcal{G}$ with its node feature matrix $\mathbf{X}$ through the graph convolution operation and a multi-layer perceptron (MLP). Specifically, the graph convolution operation proceeds by multiplying the input of each layer with the $\tau$-th power of the normalized adjacency matrix. The $\tau$-th power operator contains statistics from the $\tau$-th step of a random walk on the graph, thus nodes can indirectly receive more information from farther nodes in the graph. Unlike Choromanski et al. [2022], different $\tau$-th steps of random walk on the graph are allowed to combine thanks to our tensor architecture, which can enhance the representation power of GCN.

Combined with an MLP, the representation learned at the $\ell$-th layer is given by

$$
S_{\mathcal{G}}^{(\ell+1)} = f_{\text{MLP}} \left( \varphi \left( \mathbf{A}^\tau S_{\mathcal{G}}^{(\ell)} \Theta^{(\ell)} \right) \right),
$$

where $\mathbf{A} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$, $\mathbf{A} = \mathbf{A} + \mathbf{I}$, and $\mathbf{D}$ is the degree matrix of $\mathbf{A}$. $S_{\mathcal{G}}^{(0)} = \mathbf{X}$, $f_{\text{MLP}}$ is an MLP with batch normalization, $\varphi(\cdot)$ is a non-linear activation function, $\Theta^{(\ell)}$ is a trainable weight of $\ell$-th layer. To exploit multi-hop propagation information and increase efficiency, we apply our proposed Tensor Transformation Layer (TTL) defined in (4) over an aggregation of the outputs of all layers in Equation (2) to provide structure-aware representations of the input graph. Specifically, we first concatenate all layers of a $L$-layer graph convolutions $\left[ S_{\mathcal{G}}^{(1)}, S_{\mathcal{G}}^{(2)}, \ldots, S_{\mathcal{G}}^{(L)} \right]$ to form a node embedding tensor denoted by $\mathbf{X}_{\mathcal{G}}$ of dimension $N \times L \times D \times D$; then $\mathbf{X}_{\mathcal{G}}$ is fed into TTL as $\mathbf{H}^{(0)} = \mathbf{X}_{\mathcal{G}}$, whose $\ell$-th layer is defined in (4). Note that $\mathbf{X}_{\mathcal{G}}$ is of dimension $N \times L \times D \times D$ as to facilitate TTL. That is, we prefer both the output tensors of TT-CL and TG-CL to have the same number of dimensions. Suppose the dimension of each node’s representation is $D$ originally, we further conduct $D$ convolutional transformations, which increases the dimension from $D$ to $D^2$; then it is reshaped to $D \times D$. The output of TTL in TG-CL is denoted as $\mathbf{X}_{\mathcal{G}}''$, which captures the global topological information of a graph.

3.3 Global and Local Aggregation

Finally, to aggregate both local and global topological information, we combine representations learned from TT-CL and TG-CL together to obtain the final embedding and feed the concatenated tensor into a TTL then
a single-layer MLP for classification. The aggregation
operation is defined as

$$s_o = f_{\text{MLP}}(\text{TTL}([\mathbf{X}'_{\text{PIT}}, \mathbf{X}'_{\text{G}}])), \tag{2}$$

where \(s_o\) is the final score matrix for graph classification.

## 3.4 Tensor Transformation Layer (TTL)

The Tensor Transformation Layer (TTL) preserves the tensor structures of feature \(\mathbf{X}\) of dimension \(D = \prod_{m=1}^{M} D_m\) and hidden throughput. Let \(L\) be any positive integer and \(d = [d^{(1)}, \ldots, d^{(L+1)}]\) collects the width of all layers. A deep ReLU Tensor Neural Network is a function mapping taking the form of

$$f(\mathbf{X}) = L^{(L+1)} \circ \sigma \circ L^{(L)} \circ \ldots \circ \sigma \circ L^{(2)} \circ \sigma \circ L^{(1)}(\mathbf{X}), \tag{3}$$

where \(\sigma(\cdot)\) is an element-wise activation function. Affine transformation \(L^{(\ell)}(\cdot)\) and hidden input and output tensor of the \(\ell\)-th layer, i.e. \(\mathcal{H}^{(\ell+1)}\) and \(\mathcal{H}^{(\ell)}\) are defined by

$$L^{(\ell)} \left( \mathcal{H}^{(\ell)} \right) := \left\langle \mathbf{W}^{(\ell)}, \mathcal{H}^{(\ell)} \right\rangle + \mathbf{B}^{(\ell)}, \tag{4}$$

and \(\mathcal{H}^{(\ell+1)} := \sigma \left( L^{(\ell)} \left( \mathcal{H}^{(\ell)} \right) \right)\)

where \(\mathcal{H}^{(0)} = \mathbf{X}\) takes the tensor feature, \(\langle \cdot, \cdot \rangle\) is the tensor inner product, and low-rank weight tensor \(\mathbf{W}^{(\ell)}\) and a bias tensor \(\mathbf{B}^{(\ell)}\). The tensor structure kicks in when we incorporate tensor low-rank structures such as \(\text{CP low-rank, Tucker low-rank, and Tensor Train low-rank}\).

The Tucker low-rank structure is defined by

$$\mathbf{X} = \mathbf{C} \times_1 \mathbf{U}_1 \times_2 \ldots \times_M \mathbf{U}_M + \mathcal{E}, \tag{5}$$

where \(\mathcal{E} \in \mathbb{R}^{D_1 \times \ldots \times D_M}\) is the tensor of the idiosyncratic component (or noise) and \(\mathbf{C}\) is the latent core tensor representing the true low-rank feature tensors and \(\mathbf{U}_m, m \in [M]\), are the loading matrices.

The complete definitions of three low-rank structures are given in Appendix A. CP low-rank is a special case where the core tensor \(\mathbf{C}\) has the same dimensions over all modes, that is \(R_m = R\) for all \(m \in [M]\), and is super-diagonal. TT low-rank is a different kind of low-rank structure, which inherits advantages from both CP and Tucker decomposition. Specifically, TT decomposition can compress tensors as significantly as CP decomposition, while its calculation is as stable as Tucker decomposition.

Theoretically, the provable gain of preserving tensor structures and incorporating low-rankness is established in the next section. Empirically, we perform an ablation study for the effect of different tensor decomposition methods on molecular and chemical graphs. Results in Table 3 align with our theoretical discovery in Theorem 3.6.

## 3.5 Provable Benefits of TTL with Tucker Low-Rankness

In this section, we show provable benefits from Tensor Transformation Layer (TTL) with Tucker low-rankness. Note that under Tucker low-rankness, it is equivalent to consider either low-rank feature \(\mathbf{X}\) or low-rank weight \(\mathbf{W}\). Without loss of generality, we consider the low-rankness of feature tensor \(\mathbf{X}\) for theoretical development. The feature tensor \(\mathbf{X}\) in this section corresponds to the aggregated feature \([\mathbf{X}'_{\text{PIT}}, \mathbf{X}'_{\text{G}}]\) constructed from TT-CL and TG-CL. The theoretical result applies generally to any \(M\)-th order tensor feature \(\mathbf{X}\) of dimension \(D_1 \times \ldots \times D_M\).

The feature tensors and their corresponding labels \{\(\mathbf{X}_{i}, y_{i}\)\}_{i=1}^{n} are observable and their corresponding latent cores \{\(\mathbf{C}_{i}\)\}_{i=1}^{n} are i.i.d. copies of latent \(\mathbf{C}\). The underlying true regression model is given by

$$\mathbb{E}[y \mid \mathbf{X}] = \mathbb{E}[y \mid \mathbf{C}] = m^*(\mathbf{C}). \tag{6}$$

TTL uses deep ReLU Neural Networks to approximate \(m^*(\mathbf{C})\). When \{\(\mathbf{C}_{i}\)\}_{i=1}^{n} is not directly observable, TTL estimates it from \{\(\mathbf{X}_{i}\)\}_{i=1}^{n} using Tucker decomposition [Kolda and Bader 2009]. In this section, we provide the first theoretical guarantee for TTL. The following definition and regularity Conditions are necessary for the theoretical development.

The following regularity conditions are standard in the literature of Tucker decomposition and non-parametric regression.

**Condition 3.1** (Structured features \(\mathbf{X}\)). The tensor feature \(\mathbf{X}\) assumes an intrinsic low-rank structure specified by Tucker low-rank structure.

**Condition 3.2** (Sub-Gaussian noise). Consider the regression model \(y = m^*(\mathbf{X}) + \varepsilon\), there exists a universal constant \(C\) such that \(P(|\varepsilon| \geq t \mid \mathbf{X}) \leq 2e^{-ct^2}\) for all the \(t > 0\) almost surely.

**Condition 3.3** (Regression function). The true regression function \(m^*\) satisfies \(||m^*||_{\infty} \leq C^*\) and \(m^*\) is \(c\)-Lipschitz for some universal constants \(M^*\) and \(c\). We further assume that \(1 \leq M^* \leq M \leq e^cM^*\) for some universal constant \(c' > 1\).

**Condition 3.4** (Boundedness). For Tucker low-rank model \(\mathbf{X}\), there exists universal constants \(C_{1}\) such that (i) The factor loading matrices satisfies \(||\mathbf{U}_m||_{\text{max}} \leq C_1\) for all \(m \in [M]\); (ii) Elements in the core tensor \(\mathbf{C}\) is
zero-mean and bounded in \([-B, B]\).

**Definition 3.5** (Deep ReLU Tensor Network Class). For any depth \(L \in \mathbb{N}\), width vector \(d \in \mathbb{N}^{L+1}\), \(B, M \in \mathbb{R}^+ \cup \{\infty\}\), the family of deep ReLU Tensor Network truncated by \(M\) with depth \(L\), width parameter \(d\), and weights bounded by \(B\) is defined as

\[
C(L, d, M, B) = \{ \tilde{f}(\mathcal{X}) = \mathcal{T}_M(f(\mathcal{X})) \}
\]

where

\[
f(\mathcal{X}) \text{ defined in (3) and (4) with } \left\| \mathcal{W}^{(i)} \right\|_{\max} \leq B, \left\| \mathcal{b}^{(i)} \right\|_{\max} \leq B,
\]

and \(\mathcal{T}_M(\cdot)\) applies truncation operator at level \(C\) to each entry of a \(d_{L+1}\) dimensional vector, that is, \([\mathcal{T}_M(\mathcal{Z})]_{i_1, \ldots, i_M} = \text{sgn}(z_{i_1, \ldots, i_M})(|z_{i_1, \ldots, i_M}| \wedge M)\). We denote it as \(\mathcal{C}(L, D, m, \bar{W}, M, B)\) if the width parameter \(d = (D_m, W, \cdots, W, D_0, \bar{W})\), which we referred as deep ReLU network with depth \(L\) and width \(W\) for brevity.

We now define quantities we aim to bound theoretically. The empirical \(\ell_2\) loss is defined as

\[
R_n(f) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathcal{X}_i))^2 = \frac{1}{n} \sum_{i=1}^{n} \left( y_i - \tilde{f}(\mathcal{C}_i) \right)^2,
\]

where the last equality holds since \(\mathcal{X}\) assumes a Tucker low-rank structure. For arbitrary given neural network hyper-parameters \(L\) and \(W\), we suppose that our TTL estimator is an approximate empirical loss minimizer, that is,

\[
\hat{m}(\mathcal{X}) = \tilde{f}(\mathcal{C}) \quad \text{with} \quad R_n(\hat{f}) \leq \inf_{f \in \mathcal{C}(L, F, 1, W, M, \infty)} R_n(f) + \delta_{\text{opt}}
\]

with some optimization error \(\delta_{\text{opt}}\). We first present the error bound on the excess risk of the TTL estimator under the Tucker low-rankness.

**Theorem 3.6.** Assume Conditions 3.1-3.4 hold with \(\max \mathbb{E}[\epsilon_i^2] \leq c\) for a universal constant \(c\). Tensor feature \(\mathcal{X}\) is a \(D_1 \times \cdots \times D_M\) tensor with low Tucker rank \((R_1, \cdots, R_M)\). Let \(R = \prod_{m=1}^{M} R_m\). Then with probability at least \(1 - 3 \exp(-t)\), for large enough \(n\), we have

\[
\frac{1}{n} \sum_{i=1}^{n} \left| \hat{m}(\mathcal{X}_i) - m^*(\mathcal{C}_i) \right|^2 + \mathbb{E}_{\mathcal{X}, \mathcal{C}} \left[ \hat{m}(\mathcal{X}_i) - m^*(\mathcal{C}_i) \right]^2 \leq C \left( \delta_{\text{opt}} + \delta_{\text{apr}} + \delta_{\text{sto}} + \delta_{\text{cor}} + tn^{-1} \right)
\]

for a universal constant \(C\) that only depends on \(c\) and constants in Conditions B.1-B.4. The components in the error bounds are, respectively:

**NN approximation error**

\[
\delta_{\text{apr}} = \inf_{f \in \mathcal{C}(L, F, 1, W, M, \infty)} \| f - m^* \|_\infty^2
\]

**Stochastic error**

\[
\delta_{\text{sto}} = (W^2 L^2 + \text{WLR}) \log(WLR) \log n/n
\]

**Tensor core error**

\[
\delta_{\text{cor}} = \sigma^2 \left( \prod_{m=1}^{M} D_m \right)^{-1} \sum_{m=1}^{M} D_m R_m
\]

Theorem 3.6 establishes a high probability bound on both the out-of-sample mean squared error \(\mathbb{E}_{\mathcal{X}, \mathcal{C}} \left[ \hat{m}(\mathcal{X}_i) - m^*(\mathcal{C}_i) \right] \) and in-sample mean squared error \(\frac{1}{n} \sum_{i=1}^{n} \left| \hat{m}(\mathcal{X}_i) - m^*(\mathcal{C}_i) \right|\). The error bound is composed of four terms: the optimization error \(\delta_{\text{opt}}\), the neural network approximation error \(\delta_{\text{apr}}\) to the underlying true regression function \(m^\ast\), the stochastic error \(\delta_{\text{sto}}\) scales linearly with \(\log n/n\) and \((W^2 L^2 + \text{WLR}) \log(WLR)\) which is proportional to the Pseudo-dimension of the neural network class we used, and the error \(\delta_{\text{cor}}\) related to inferring the latent core tensor \(\mathcal{C}\) from the observation \(\mathcal{X}\). Such an error bound is not applicable without specifying the network hyper-parameters \(W\) and \(L\). An optimal rate can be further obtained by choosing \(W\) and \(L\) to trade off the approximation error \(\delta_{\text{apr}}\) and the stochastic error \(\delta_{\text{sto}}\).

The NN approximation error \(\delta_{\text{apr}}\) can be controlled by the architecture of NN and the optimization error \(\delta_{\text{opt}}\) can be controlled by an optimization algorithm. Given the depth \(L\) and hidden width \(W\) of a NN, the stochastic error \(\delta_{\text{sto}}\) is increasing in \(R = \prod_{m=1}^{M} R_m\), which is the intrinsic dimension of the low-rank weight tensor in the affine transformation (4) of a single neuron. The ambient dimension of the weight tensor is \(D = \prod_{m=1}^{M} D_m\). Under a low-rank structure, the intrinsic dimension \(R\) is much smaller than the ambient dimension \(D\). As a result, our proposed Tensor Transformation Layer (TTL) greatly reduces the stochastic error. At the same time, the low-rankness also controls the core tensor estimation error \(\delta_{\text{cor}}\) remarkably well.

The benefit of Tucker low-rankness shows up in the stochastic error \(\delta_{\text{sto}}\) where \(R = \prod_{m=1}^{M} R_m\) is the total number of elements in the core tensor \(\mathcal{C}\) and is also equivalent to the total number of unknown coefficients in low-rank weight tensor \(\mathcal{W}^{(l)}\) in the affine transformation (4). The latent Pseudo-dimension of the neural network class we used is reduced thanks to the incorporation of Tucker low-rankness.
4 Experiments

4.1 Experiment Settings

Datasets We validate our TTG-NN on graph classification tasks using the following real-world chemical compounds, protein molecules, and social network datasets: (i) 4 chemical compound datasets: MUTAG, DHFR, BZR, and COX2, where the graphs are chemical compounds, the nodes are different atoms, and the edges are chemical bonds; (ii) 6 molecular compound datasets: D&D, PROTEINS, PTC_MR, PTC_MM, PTC_FM, and PTC_FR, where the nodes represent amino acids and edges represent relationships or interactions between the amino acids, e.g., physical bonds, spatial proximity, or functional interactions; (iii) 1 social network dataset: IMDB-B, where the nodes represent actors/actresses, and edges exist between them if they appear in the same movie. For all graphs, we follow the training principle [Xu et al. 2018] and results of the 10-fold cross-validation are reported using standard deviations. OOT indicates out of time (we allow 24 hours for each run).

Baselines We compare our TTG-NN with 20 state-of-the-art (SOTA) baselines including (1) Compressed Subgraph Matching kernel (CSM) [Kriege and Mutzel 2012], (2) Shortest Path Hash Graph Kernel (SPK) [Morris et al. 2016], (3) Weisfeiler-Lehman Subtree Kernel (WL) [Morris et al. 2016], and (4) Weisfeiler-Lehman (WL) [Shervashidze et al. 2011], (5) Graph Convolutional Network (GCN) [Kipf and Welling 2017], (6) Chebyshev GCN (ChebNet) [Defferrard et al. 2016], (7) Graph Isomorphism Network (GIN) [Xu et al. 2018], (8) Deep Graph Convolutional Network (DGCNN) [Zhang et al. 2018], and (9) Capsule Graph Neural Network (CapsGNN) [Xinyi and Chen 2019], (10) GNNs with Differentiable Pooling (DiffPool) [Ying et al. 2018], (11) Graph U-Nets (g-U-Nets) [Gao and Ji 2019], (12) GCNs with Eigen Pooling (EigenGCN) [Ma et al. 2019], (13) Self-attention Graph Pooling (SAGPool) [Lee et al. 2019], (14) Spectral Clustering for Graph Pooling (MinCutPool) [Bianchi et al. 2020], and (15) Haar Graph Pooling (HaarPool) [Wang et al. 2020], (16) Topological Graph Neural Networks (TOGL) [Horn et al. 2021], (17) PD-based Neural Networks (PerS-Lay) [Carrière et al. 2020], (18) Filtration Curve-based Random Forest (FC-V) [O’Bray et al. 2021], (19) Deep Graph Mapper (MPR) [Bodnar et al. 2021a], and (20) Simplicial Isomorphism Networ (SIN) [Bodnar et al. 2021b].

TTG-NN Setup We conduct our experiments on one NVIDIA Quadro RTX 8000 GPU card with up to 48GB memory. The TTG-NN is trained end-to-end by using Adam optimizer with the learning rate of {0.001, 0.01, 0.05, 0.1}. We use ReLU as the activation function $\sigma(\cdot)$ across our model. The tuning of TTG-NN on each dataset is done via the grid hyperparameter configuration search over a fixed set of choices and the same cross-validation setup is used to tune the baselines. In our experiments, we set the grid size of $|\Pi|$ from $20 \times 20$ to $50 \times 50$. The batch size is different for every dataset and ranges from 8 to 128. Each graph convolutional layer and MLP has between 16 and 128 hidden units depending on the dataset regarded. The number of hidden units of TTL are set as {4, 16, 32}. We set the layer number of graph convolution blocks as 3, the layer number of MLPs as 2, and choose the dropout ratio as 0.5 for all datasets. We train the models with up to 500 epochs to ensure full convergence and randomly use 50 batches for each epoch. Our code is available on [GitHub].

4.2 Classification Performance

As shown in Table 1, except for DHFR, the performances of our TTG-NN model are significantly better than the runner-ups. More specifically, we found that (i) compared with spectral-based ConvGNNs (i.e., GCN and ChebNet), TTG-NN yields more than 2.20% relative improvements to the existing best results for all datasets, (ii) compared with 3 spatial-based ConvGNNs, i.e., DGCNN, GIN, and CapsGNN, TTG-NN achieves a relative gain of up to 16.20% on all datasets, (iii) TTG-NN outperforms all 6 graph pooling methods (i.e., g-U-Nets, MinCutPool, DiffPool, EigenGCN, SAGPool, and HaarPool) with a significant margin, and (iv) TTG-NN further improves PH-based models and simplicial neural networks by a significant margin on all 11 datasets. To test our TTG-NN’s performance on large-scale dataset, we have conducted experiment on the ogbg-molhiv dataset [Hu et al. 2020] with 41,127 graphs. The results in Table 2 show our proposed model is able to achieve promising results on large-scale networks. To sum up, the results show that our TTG-NN accurately captures the key structural and topological information of the graph, and achieves a highly promising performance in graph classification.

4.3 Ablation Study

To better understand the importance of different components in TTG-NN, we design the ablation study experiments on MUTAG, BZR, COX2, PTC.MM, and PTC_FM. As shown in Table 3, if we remove the tensor-view topological convolutional layer (TT-CL), the performance will drop over 8.70% on average. Specifically, we observe that when removing TT-CL, the performance on graph classification is affected significantly, i.e., TTG-NN outperforms TTG-NN without TT-CL with a relative gain of 12.39% for PTC_FM. Moreover, we find that on all 5 datasets, TTG-NN out-
4.4 Sensitivity Analysis

To evaluate the model performance with different tensor decomposition methods, we test the performance of our proposed TTG-NN model with 3 different tensor decompositions, i.e., Tucker, TT, and CP. As Table 4 shows that (i) on MUTAG, BZR, and COX2, CP method always show better performance than Tucker and TT, and the average relative gain is 5.02%; (ii) on PTC_M and PTC_FM, we can see that TTG-NN equipped with TT outperforms TTG-NN with Tucker and CP decompositions respectively.

Furthermore, we use five filtration functions, i.e., Degree(deg.), Betweenness centrality(betw.), Closeness centrality(close.), Eigenvector centrality(eign.), and Communicability centrality(comm.) on most benchmark datasets, as they cover most essential topological properties. However, comm. would incur computation errors on a few datasets, e.g., PROTEINS, D&D. Also, considering that deg. is arguably the most fundamental property, we test deg., betw., eign., close. individually and in combination on MUTAG and PROTEINS datasets. From Table 4, we can see that the effect of different choices of filtration functions varies from data to data. However, we observe that the model performance generally increases with the number of filtration functions used.

4.5 Computational Complexity

The topological complexity of the standard PH matrix reduction algorithm [Edelsbrunner et al. 2000] runs in time at most $O(\varepsilon^3)$, where $\varepsilon$ is the number of simplices in a filtration. For 0-dimensional PH, it can be computed efficiently using disjoint sets with complexity $O(\varepsilon^2 \log \varepsilon)$, where $\alpha(-1)$ is the inverse Ackermann function $\alpha(x)$ (Cormen et al. 2022).

Furthermore, in Table 5, we show the running time (i.e., training time per epoch) of the proposed TUT with 3 different tensor decomposition methods on MUTAG, BZR, COX2, PTC-MM, and PTC_FM datasets. We also compare the running time (training time per epoch; along with the accuracy (%) between our TTG-NN model and

Table 1: Performance on molecular and chemical graphs. The best results are given in bold.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MUTAG</th>
<th>BZR</th>
<th>COX2</th>
<th>DHFR</th>
<th>D&amp;D</th>
<th>PROTEINS</th>
<th>PTC_M</th>
<th>PTC_MM</th>
<th>PTC_FM</th>
<th>PTC_FR</th>
<th>IMDB_B</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSM</td>
<td>84.54</td>
<td>80.79</td>
<td>79.11</td>
<td>0.04</td>
<td>OOT</td>
<td>87.20</td>
<td>1.25</td>
<td>52.42</td>
<td>44.22</td>
<td>63.01</td>
<td>1.70</td>
</tr>
<tr>
<td>KriGe</td>
<td>84.09</td>
<td>80.76</td>
<td>79.16</td>
<td>0.05</td>
<td>OOT</td>
<td>74.56</td>
<td>2.45</td>
<td>52.42</td>
<td>44.22</td>
<td>63.01</td>
<td>1.70</td>
</tr>
<tr>
<td>CSM</td>
<td>84.54</td>
<td>80.79</td>
<td>79.11</td>
<td>0.04</td>
<td>OOT</td>
<td>87.20</td>
<td>1.25</td>
<td>52.42</td>
<td>44.22</td>
<td>63.01</td>
<td>1.70</td>
</tr>
<tr>
<td>KriGe</td>
<td>84.09</td>
<td>80.76</td>
<td>79.16</td>
<td>0.05</td>
<td>OOT</td>
<td>74.56</td>
<td>2.45</td>
<td>52.42</td>
<td>44.22</td>
<td>63.01</td>
<td>1.70</td>
</tr>
</tbody>
</table>

Table 2: Graph classification results (%) on ogbg-molhiv.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>GIN</th>
<th>GNN</th>
<th>GIN</th>
<th>GNN</th>
<th>PNA</th>
<th>TTG-NN (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ogbg-molhiv</td>
<td>75.99</td>
<td>1.19</td>
<td>77.07</td>
<td>1.49</td>
<td>79.90</td>
<td>1.32</td>
</tr>
</tbody>
</table>

Table 3: TTG-NN ablation study.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>MUTAG</th>
<th>BZR</th>
<th>COX2</th>
<th>PTC_M</th>
<th>PTC_MM</th>
<th>PTC_FM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTG-NN W+TT-CL</td>
<td>85.67</td>
<td>1.70</td>
<td>82.74</td>
<td>0.03</td>
<td>74.15</td>
<td>1.36</td>
</tr>
<tr>
<td>TTG-NN W+TT-CL</td>
<td>83.64</td>
<td>1.56</td>
<td>80.91</td>
<td>0.09</td>
<td>75.05</td>
<td>1.46</td>
</tr>
<tr>
<td>TTG-NN W+TT-CL</td>
<td>91.22</td>
<td>1.56</td>
<td>85.38</td>
<td>0.51</td>
<td>70.02</td>
<td>0.51</td>
</tr>
<tr>
<td>TTG-NN</td>
<td>93.65</td>
<td>1.48</td>
<td>88.40</td>
<td>0.22</td>
<td>86.73</td>
<td>1.41</td>
</tr>
</tbody>
</table>

Table 4: Sensitivity analysis of tensor decomposition.
three runner-ups. Specifically, for MUTAG, TTG-NN: 18.58 seconds (93.65%) vs. HaarPool: 19.31 seconds (90.00%) vs. PersLay: 13.00 seconds (89.80%) vs. GIN 0.38 seconds (89.39%); for PTC_MM: TTG-NN 3.97 seconds (74.11%) vs. SIN: 5.62 seconds (70.55%) vs. HaarPool: 7.72 seconds (69.69%) vs. MPR: 9.61 seconds (68.60%). Compared with runner-ups, TTG-NN always achieves competitive classification performance and computation cost.

Table 5: Run time analysis of tensor decomposition(seconds per epoch).

<table>
<thead>
<tr>
<th>Decomposition</th>
<th>MUTAG</th>
<th>BZR</th>
<th>COX2</th>
<th>PTC_MM</th>
<th>PTC_FM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTG-NN With Tucker</td>
<td>13.41 s</td>
<td>44.71 s</td>
<td>7.52 s</td>
<td>3.73 s</td>
<td>4.47 s</td>
</tr>
<tr>
<td>TTG-NN With TT</td>
<td>17.58 s</td>
<td>36.28 s</td>
<td>7.20 s</td>
<td>3.97 s</td>
<td>3.60 s</td>
</tr>
<tr>
<td>TTG-NN With CP</td>
<td>18.58 s</td>
<td>45.02 s</td>
<td>17.55 s</td>
<td>12.72 s</td>
<td>12.85 s</td>
</tr>
</tbody>
</table>

5 Conclusion

In this paper, we have proposed a novel Tensor-view Topological Graph Neural Network (TTG-NN) with graph topological and structural feature tensors. In TTG-NN, TT-CL and TG-CL harness tensor structures to consolidate features from diverse sources, while TTL exploits tensor low-rank decomposition to proficiently manage both model complexity and computational efficiency. TTG-NN architecture can be flexibly extended by incorporating additional graph representation learning modules through the integration of parallel structures with TT-CL and TG-CL. Moreover, we theoretically show that the proposed Tensor Transformation Layer (TTL) reduces the stochastic noise and error. Extensive experiments on graph classification tasks demonstrate the effectiveness of both TTG-NN and the proposed components. Future research directions include further extending the tensor-view topological deep learning idea to unsupervised/supervised spatiotemporal prediction and community detection.

Acknowledgement. Y.C. has been supported in part by the NASA AIST grant 21-AIST21_2-0059, and NSF Grant DMS-2335846/2335847.
References


**Checklist**

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
(a) The full text of instructions given to participants and screenshots. [Not Applicable]

(b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

(c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]
A Tensor Algebra

A.1 Tensor Low-Rank Structures

Consider an $M$-th order tensor $\mathbf{X}$ of dimension $D_1 \times \cdots \times D_M$. If $\mathbf{X}$ assumes a (canonical) rank-$R$ CP low-rank structure, then it can be expressed as

$$\mathbf{X} = \sum_{r=1}^{R} c_r \mathbf{u}_{1r} \circ \mathbf{u}_{2r} \circ \cdots \circ \mathbf{u}_{Mr},$$

where $\circ$ denotes the outer product, $\mathbf{u}_{mr} \in \mathbb{R}^{D_m}$ and $\|\mathbf{u}_{mr}\|_2 = 1$ for all mode $m \in [M]$ and latent dimension $r \in [R]$. Concatenating all $R$ vectors corresponding to a mode $m$, we have $\mathbf{U}_m = [\mathbf{u}_{m1}, \ldots, \mathbf{u}_{mR}] \in \mathbb{R}^{D_m \times R}$ which is referred to as the loading matrix for mode $m \in [M]$. If $\mathbf{X}$ assumes a rank-$(R_1, \ldots, R_M)$ Tucker low-rank structure \cite{Oseledets2011}, then it writes

$$\mathbf{X} = \mathbf{C} \times_1 \mathbf{U}_1 \times_2 \cdots \times_M \mathbf{U}_M = \sum_{r_1=1}^{R_1} \cdots \sum_{r_M=1}^{R_M} c_{r_1 \cdots r_M} \mathbf{u}_{1r_1} \circ \cdots \circ \mathbf{u}_{Mr_M},$$

where $\mathbf{u}_{mr}$ are all $D_m$-dimensional vectors, and $c_{r_1 \cdots r_M}$ are elements in the $R_1 \times \cdots \times R_D$-dimensional core tensor $\mathbf{C}$. Tensor Train (TT) low-rank \cite{Oseledets2011} approximates a $D_1 \times \cdots \times D_M$ tensor $\mathbf{X}$ with a chain of products of third order core tensors $\mathbf{C}_i$, $i \in [M]$, of dimension $R_{i-1} \times D_i \times R_i$. Specifically, each element of tensor $\mathbf{X}$ can be written as

$$x_{i_1, \ldots, i_M} = c_{1,1,i_{1,:}}^T \times c_{2,:i_{2,:}} \times \cdots \times c_{M,:i_{M,:}} \times c_{M+1,:1,1},$$

where $c_{m,:,:} \in \mathbb{R}^{R_{m-1} \times R_m}$ is an $R_{m-1} \times R_m$ matrix for $m \in [M] \cup \{M + 1\}$. The product of those matrices is a matrix of size $R_0 \times R_{M+1}$. Letting $R_0 = 1$, the first core tensor $\mathbf{C}_1$ is of dimension $1 \times D_1 \times R_1$, which is actually a matrix and whose $i_1$-th slice of the middle dimension (i.e. $c_{1,1,i_{1,:}}$) is actually a $R_1$ vector. To deal with the “boundary condition” at the end, we augmented the chain with an additional tensor $\mathbf{C}_{M+1}$ with $D_{M+1} = 1$ and $R_{M+1} = 1$ of dimension $R_M \times 1 \times 1$. So the last tensor can be treated as a vector of dimension $R_M$.

CP low-rank \cite{Kolda2009} is a special case where the core tensor $\mathbf{C}$ has the same dimensions over all modes, that is $R_m = R$ for all $m \in [M]$, and is super-diagonal. TT low-rank is a different kind of low-rank structure and it inherits advantages from both CP and Tucker decomposition. Specifically, TT decomposition can compress tensors as significantly as CP decomposition, while its calculation is as stable as Tucker decomposition. We further perform the ablation study to the effect of different tensor decomposition methods on molecular and chemical graphs (see Table 3 in the main body).

B Notation and Additional Details of Datasets

B.1 Notation

Frequently used notation is summarized in Table 6.

B.2 Topological Tensor Feature

To incorporate topological features from more dimensions, we calculate a set of PDs for each filtration function $f_i$, i.e., $\text{PH}(G, f_i) = \overrightarrow{Dg_i} = \{Dg_i^{(1)}, \ldots, Dg_i^{(Q)}\}$, where $Q \in \mathbb{Z}_0^+$ is the number of graph topological features. Moreover, to encode above topological information presented in a PD $Dg$ into the embedding function, we use its vectorized representation, i.e., persistence image (PI) \cite{Adams2017}. The PI is a finite-dimensional vector representation obtained through a weighted kernel density function and can be computed in following two steps (see more details in Definition B.1). First, we map the PD $Dg$ to an integrable function $\varrho_{Dg} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, which is referred to as a persistence surface. The persistence surface $\varrho_{Dg}$ is constructed by summing weighted Gaussian kernels centered at each point in $Dg$. In the second step, we integrate the persistence surface $\varrho_{Dg}$ over each grid box to obtain the value of the $PI_{Dg}$.
Table 6: The main symbols and definitions in this paper.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{G}$</td>
<td>an attributed graph</td>
</tr>
<tr>
<td>$\mathcal{V}$</td>
<td>the set of nodes of $\mathcal{G}$</td>
</tr>
<tr>
<td>$\mathcal{E}$</td>
<td>the set of edges of $\mathcal{G}$</td>
</tr>
<tr>
<td>$N$</td>
<td>the number of nodes</td>
</tr>
<tr>
<td>$A$</td>
<td>the adjacency matrix of $\mathcal{G}$</td>
</tr>
<tr>
<td>$D$</td>
<td>the degree matrix of $\mathcal{G}$</td>
</tr>
<tr>
<td>$\mathcal{X}$</td>
<td>the feature matrix of $\mathcal{G}$</td>
</tr>
<tr>
<td>$y$</td>
<td>the label of $\mathcal{G}$</td>
</tr>
<tr>
<td>$\aleph$</td>
<td>the number of graphs</td>
</tr>
<tr>
<td>$\mathcal{C}$</td>
<td>a simplicial complex</td>
</tr>
<tr>
<td>$Q$</td>
<td>the maximum dimension of a simplicial complex</td>
</tr>
<tr>
<td>$K$</td>
<td>the number of filtration functions</td>
</tr>
<tr>
<td>$P$</td>
<td>the number of persistent images for each filtration function</td>
</tr>
<tr>
<td>$L$</td>
<td>the number of graph convolutional layers</td>
</tr>
<tr>
<td>$\mathcal{L}$</td>
<td>an affine transformation</td>
</tr>
<tr>
<td>$\mathcal{X}$</td>
<td>a tensor</td>
</tr>
<tr>
<td>$C$</td>
<td>a core tensor</td>
</tr>
<tr>
<td>$R$</td>
<td>the total number of elements in $\mathcal{C}$</td>
</tr>
</tbody>
</table>

**Definition B.1** (Persistence Image). Let $g: \mathbb{R}^2 \mapsto \mathbb{R}$ be a non-negative weight function for the persistence plane $\mathbb{R}$. The value of each pixel $z \in \mathbb{R}^2$ is defined as $PI_{Dg}(z) = \int \int \sum_{\mu \in T(Dg)} g(\mu) e^{-\left(\frac{(x-\mu_x)^2}{2\delta_x^2} + \frac{(y-\mu_y)^2}{2\delta_y^2}\right)} dy dx$, where $T(Dg)$ is the transformation of the PD $Dg$ (i.e., for each $(x, y)$, $T(x, y) = (x, y - x)$), $\mu = (\mu_x, \mu_y) \in \mathbb{R}^2$, and $\delta_x$ and $\delta_y$ are the standard deviations of a differentiable probability distribution in the $x$ and $y$ directions respectively.

**B.3 Datasets**

Table 7 summarizes the characteristics of all ten datasets used in our experiments.

| Dataset | # Graphs | Avg. $|\mathcal{V}|$ | Avg. $|\mathcal{E}|$ | # Class |
|---------|----------|----------------|----------------|---------|
| BZR     | 405      | 35.75          | 38.35          | 2       |
| COX2    | 467      | 41.22          | 43.45          | 2       |
| DHFR    | 467      | 42.43          | 44.54          | 2       |
| D&D     | 1178     | 284.32         | 715.66         | 2       |
| MUTAG   | 188      | 17.93          | 19.79          | 2       |
| PROTEINS| 1113     | 39.06          | 72.82          | 2       |
| PTC_MR  | 344      | 14.29          | 14.69          | 2       |
| PTC_MM  | 336      | 13.97          | 14.32          | 2       |
| PTC_FM  | 349      | 14.11          | 14.48          | 2       |
| PTC_FR  | 351      | 14.56          | 15.00          | 2       |

**C Additional Experimental Results**

In our study, we use five filtration functions, i.e., Degree(deg.), Betweenness Centrality(betw.), Closeness Centrality(close.), Eigenvector Centrality(eign.) and Communicability Centrality(comm.) on most benchmarks, as they cover most essential topological properties. However, comm. would incur computation errors on a few
datasets, e.g., PROTEINS, D&D. Also, considering that deg. is arguably the most fundamental and essential property, we test deg., betw., eigen., close. individually and in combination on MUTAG and PROTEINS datasets. From Table 8, we can see that the effect of certain choices of filtration functions varies from data to data. However, we observe that the model performance generally increases with the number of filtration functions used.

Table 8: Sensitivity analysis of filtration functions

<table>
<thead>
<tr>
<th>Dataset</th>
<th>filtration</th>
<th>deg</th>
<th>betw</th>
<th>eigen</th>
<th>close</th>
<th>[deg, betw]</th>
<th>[deg, eigen]</th>
<th>[deg, close]</th>
<th>[deg, betw, eigen]</th>
<th>[deg, betw, close]</th>
<th>[deg, eigen, close]</th>
<th>[deg, betw, eigen, close]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUTAG</td>
<td>deg</td>
<td>86.1 ± 6.24</td>
<td>79.42 ± 4.50</td>
<td>88.32 ± 3.48</td>
<td>74.22 ± 7.34</td>
<td>80.38 ± 6.36</td>
<td>76.42 ± 6.64</td>
<td>80.38 ± 6.02</td>
<td>74.22 ± 6.44</td>
<td>77.32 ± 6.00</td>
<td>77.32 ± 6.00</td>
<td></td>
</tr>
<tr>
<td>PROTEINS</td>
<td>deg</td>
<td>73.32 ± 8.64</td>
<td>74.90 ± 5.01</td>
<td>72.75 ± 6.11</td>
<td>71.50 ± 4.66</td>
<td>71.31 ± 4.59</td>
<td>71.69 ± 5.76</td>
<td>74.75 ± 5.47</td>
<td>73.32 ± 7.64</td>
<td>73.32 ± 6.64</td>
<td>77.62 ± 5.92</td>
<td></td>
</tr>
</tbody>
</table>