Tensor-view Topological Graph Neural Network

Tao Wen
Center for Data Science
New York University
tw2672@nyu.edu

Elynn Chen
Stern School of Business
New York University
elynn.chen@stern.nyu.edu

Yuzhou Chen
Computer and Information Sciences
Temple University
yuzhou.chen@temple.edu

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 multi-modal 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.

Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

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], Hamilton et al. [2017], 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-

ied Wasserman [2018], Carlsson and Gabrielsson [2020], O'Bray et al. [2021], Edelsbrunner and Harer [2022]. PH is a methodology under the topological data analysis (TDA) framework that captures the topological features (like connected components, holes, voids, etc.) of a shape at various scales and provides a multi-scale description of the shape. In this case, we can say that PH studies the observed object at multiple resolutions or evaluates topological patterns and structures through multiple lenses. By incorporating this multi-scale topological information, topological-based models can capture both the geometric and topological structures, and gain a more comprehensive representation of the data. Previous research has generated a topological signature and computed a parameterized vectorization that can be integrated into kernel functions Bubenik [2015], Kusano et al. [2016], Reininghaus et al. [2015]. With growing interest in DL, several topological DL (TDL) methods have been proposed Hofer et al. [2017], Carrière et al. [2020]. Specifically, they extract topological features from underlying data (e.g., topological features encoded in persistence diagrams) and integrate them into any type of DL. Recently, Zhao et al. [2020], Chen et al. [2021], Yan et al. [2021], Horn et al. [2021] prove that it is important to learn node representations based on both the topological structure and node attributes for the graph learning problems. However, these above topological-based models cannot (1) fully capture the rich multi-dimensional/multi-filtrations topological features in objects and (2) exploit the low-rank structure from intermediate layers of TDL models. For instance, topological GNN Zhao et al. [2020], Chen et al. [2021] only calculates the topological features of nodes via a single filtration. In Horn et al. [2021], Chen et al. [2022b], Horn et al. and Chen et al. have addressed the filtration issues, but their methods fail to model topological feature tensors while preserving their low-rank structures.

In this paper, we develop a novel framework, namely Tensor-view Topological Graph Neural Network (TTG-NN) to address the above problems for real-world graph data. More specifically, we propose two novel and effective tensor-based graph representation learning schemes, i.e., Tensor-view Topological Convolutional Layers (TT-CL) and Tensor-view Graph Convolutional Layers (TG-CL). Technically, we first produce topological and structural feature tensors of graphs as 3D or 4D tensors by using *multi-filtrations* and graph convolutions respectively. Then, we utilize TT-CL and TG-CL to learn hidden local and global topological representations of graphs.

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], Zhou et al. [2022], 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) Shervashidze 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. [2016] 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 $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X})$ be an attributed graph, where $\mathcal{V}$ is a set of nodes ($|\mathcal{V}| = N$), $\mathcal{E}$ is a set of edges, and $\boldsymbol{X} \in \mathbb{R}^{N\times F}$ is a feature matrix of nodes (here F is the dimension of the node features). Let $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ be a symmetric adjacency matrix whose entries are $a_{ij} = \omega_{ij}$ if nodes i and j are connected and 0 otherwise (here ω_{ij} is an edge weight and $\omega_{ij} \equiv 1$ for unweighted graphs). Furthermore, $\boldsymbol{D}$ represents the degree matrix of $\boldsymbol{A}$, that is $d_{ii} = \sum_j a_{ij}$. In the graph classification setting, we have a set of graphs $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_\aleph\}$, where each graph $\mathcal{G}_i$ 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 $\mathcal{G}$ that remain preserved at different resolutions under continuous transformations (where $q = \{0, 1, \ldots, \mathcal{Q}\}$ and $\mathcal{Q}$ 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 $\mathcal{G}$ which may play an essential role in a given learning task. The key is to select a suitable scale parameter ϵ and then to study changes in the shape of $\mathcal{G}$ that occur as $\mathcal{G}$ evolves to ϵ. Thus, given an increasing sequence $\epsilon_1 < \cdots < \epsilon_n$, we no longer study $\mathcal{G}$ as a single object but as a *filtration* $\mathcal{G}_{\epsilon_1} \subseteq \ldots \subseteq \mathcal{G}_{\epsilon_n} = \mathcal{G}$. To ensure that the process of pattern selection and count are objective and efficient, we build an abstract simplicial complex $\mathscr{C}(\mathcal{G}_{\epsilon_j})$ on each $\mathcal{G}_{\epsilon_j}$, which results in filtration of complexes $\mathscr{C}(\mathcal{G}_{\epsilon_1}) \subseteq \ldots \subseteq \mathscr{C}(\mathcal{G}_{\epsilon_n})$. For instance, we consider a function on a node set $\mathcal{V}$. 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 $u \in \mathcal{V}$, and get a descriptor function (i.e., filtration function) $f(u) = \deg(u)$. When scanning $\mathcal{G}$ via the degree-based filtration function f, it results in a sequence of induced subgraphs of $\mathcal{G}$ with a maximal degree of ϵ_j for each $j \in \{1, \ldots, n\}$. A standard descriptor of the above topological evolution is *Persistence Diagram* (PD) Barannikov [1994] $Dg = \{(b_\rho, d_\rho) \in \mathbb{R}^2 | b_\rho < d_\rho\}$, which is a multi-set of points in $\mathbb{R}^2$. Each persistence point (b_ρ, d_ρ) corresponds to the lifespan (i.e., $d_\rho - b_\rho$) of one topological feature, where b_ρ and d_ρ represent the birth and death time of the topological feature ρ.

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.

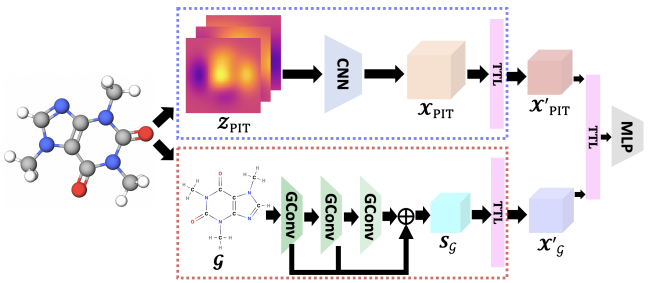

Figure 1: The architecture of TTG-NN.

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* $\boldsymbol{\mathcal{Z}}_{\text{PIT}}$ 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 $\boldsymbol{\mathcal{Z}}_{\text{PIT}}$, (ii) leverage and preserve the multi-modal structure in the $\boldsymbol{\mathcal{Z}}_{\text{PIT}}$, and (iii) capture the structure in trainable weights (with fewer parameters). Firstly, hidden representations of the PI tensor $\boldsymbol{\mathcal{Z}}_{\text{PIT}}$ 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

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

where f_{CNN} is a CNN-based neural network, ξ_{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 $\boldsymbol{\mathcal{Z}}_{\text{PIT}}$, we can apply any CNN-based model to learn the latent feature of the $\boldsymbol{\mathcal{Z}}_{\text{PIT}}$; (ii) if 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, $\boldsymbol{\mathcal{X}}_{\text{PIT}}$ is fed into our *Tensor Transformation Layer (TTL)* as $\boldsymbol{\mathcal{H}}^{(0)} = \boldsymbol{\mathcal{X}}_{\text{PIT}}$, whose ℓ-th layer is defined in (4). The output of TTL in TT-CL is denoted as $\boldsymbol{\mathcal{X}}'_{\text{PIT}}$, 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 $\boldsymbol{X}$ 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 τ-th power of the normalized adjacency matrix. The τ-th power operator contains statistics from the τ-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 τ-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 ℓ-th layer is given by

$$\boldsymbol{S}_{\mathcal{G}}^{(\ell+1)} = f_{\text{MLP}}\left(\varphi\left(\widehat{\boldsymbol{A}}^{\intercal}\boldsymbol{S}_{\mathcal{G}}^{(\ell)}\boldsymbol{\Theta}^{(\ell)}\right)\right), \quad (2)$$

where $\widehat{\boldsymbol{A}} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{\frac{1}{2}}$, $\widetilde{\boldsymbol{A}} = \boldsymbol{A}+\boldsymbol{I}$, and $\widetilde{\boldsymbol{D}}$ is the degree matrix of $\widetilde{\boldsymbol{A}}$. $\boldsymbol{S}_{\mathcal{G}}^{(0)} = \boldsymbol{X}$, f_{MLP} is an MLP with batch normalization, $\varphi(\cdot)$ is a non-linear activation function, $\boldsymbol{\Theta}^{(\ell)}$ is a trainable weight of ℓ-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 $\left[\boldsymbol{S}_{\mathcal{G}}^{(1)}, \boldsymbol{S}_{\mathcal{G}}^{(2)}, \ldots, \boldsymbol{S}_{\mathcal{G}}^{(L)}\right]$ to form a node embedding tensor denoted by $\boldsymbol{\mathcal{X}}_{\mathcal{G}}$ of dimension $N \times L \times D \times D$; then $\boldsymbol{\mathcal{X}}_{\mathcal{G}}$ is fed into TTL as $\boldsymbol{\mathcal{H}}^{(0)} = \boldsymbol{\mathcal{X}}_{\mathcal{G}}$, whose ℓ-th layer is defined in (4). Note that $\boldsymbol{\mathcal{X}}_{\mathcal{G}}$ is of dimension $N \times L \times D \times D$ so 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 $\boldsymbol{\mathcal{X}}'_{\mathcal{G}}$, 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

$$\boldsymbol{s}_o = f_{\text{MLP}}(\text{TTL}([\boldsymbol{\mathcal{X}}'_{\text{PIT}}, \boldsymbol{\mathcal{X}}'_{\mathcal{G}}])),$$

where $\boldsymbol{s}_o$ 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 $\boldsymbol{\mathcal{X}}$ of dimension $D = \prod_{m=1}^{M} D_m$ and hidden throughput. Let L be any positive integer and $\boldsymbol{d} = \left[d^{(1)}, \cdots, d^{(L+1)}\right]$ collects the width of all layers. A *deep ReLU Tensor Neural Network* is a function mapping taking the form of

$$f(\boldsymbol{\mathcal{X}}) = \mathcal{L}^{(L+1)} \circ \sigma \circ \mathcal{L}^{(L)} \circ \sigma \cdots \circ \mathcal{L}^{(2)} \circ \sigma \circ \mathcal{L}^{(1)}(\boldsymbol{\mathcal{X}}), \quad (3)$$

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

$$\begin{aligned} \mathcal{L}^{(\ell)}\left(\boldsymbol{\mathcal{H}}^{(\ell)}\right) &:= \left\langle \boldsymbol{\mathcal{W}}^{(\ell)}, \boldsymbol{\mathcal{H}}^{(\ell)} \right\rangle + \boldsymbol{\mathcal{B}}^{(\ell)}, \\ \text{and} \quad \boldsymbol{\mathcal{H}}^{(\ell+1)} &:= \sigma\left(\mathcal{L}^{(\ell)}\left(\boldsymbol{\mathcal{H}}^{(\ell)}\right)\right) \end{aligned} \quad (4)$$

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

The Tucker low-rank structure is defined by

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{C}} \times_1 \boldsymbol{U}_1 \times_2 \cdots \times_M \boldsymbol{U}_M + \boldsymbol{\mathcal{E}}, \quad (5)$$

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

The complete definitions of three low-rank structures are given in Appendix A. CP low-rank (13) is a special case where the core tensor $\boldsymbol{\mathcal{C}}$ 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 $\boldsymbol{\mathcal{X}}$ or low-rank weight $\boldsymbol{\mathcal{W}}$. Without loss of generality, we consider the low-rankness of feature tensor $\boldsymbol{\mathcal{X}}$ for theoretical development. The feature tensor $\boldsymbol{\mathcal{X}}$ in this section corresponds to the aggregated feature $[\boldsymbol{\mathcal{X}}'_{\text{PIT}}, \boldsymbol{\mathcal{X}}'_{\mathcal{G}}]$ constructed from TT-CL and TG-CL. The theoretical result applies generally to any M-th order tensor feature $\boldsymbol{\mathcal{X}}$ of dimension $D_1 \times \cdots \times D_M$.

The feature tensors and their corresponding labels $\{\boldsymbol{\mathcal{X}}_i, y_i\}_{i=1}^n$ are observable and their corresponding latent cores $\{\boldsymbol{\mathcal{C}}_i\}_{i=1}^n$ are i.i.d. copies of latent $\boldsymbol{\mathcal{C}}$. The underlying true regression model is given by

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

TTL uses deep ReLU Neural Networks to approximate $m^*(\boldsymbol{\mathcal{C}})$. When $\{\boldsymbol{\mathcal{C}}_i\}_{i=1}^n$ is not directly observable, TTL estimates it from $\{\boldsymbol{\mathcal{X}}_i\}_{i=1}^n$ 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 $\boldsymbol{\mathcal{X}}$)**.** *The tensor feature $\boldsymbol{\mathcal{X}}$ assumes an intrinsic low-rank structure specified by Tucker* (5) *low-rank structure.*

Condition 3.2 (Sub-Gaussian noise)**.** *Consider the regression model $y = m^*(\boldsymbol{\mathcal{X}}) + \varepsilon$, there exists a universal constant C such that $\mathbb{P}(|\varepsilon| \geq t \mid \boldsymbol{\mathcal{X}}) \leq 2e^{-ct^2}$ 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 $1 \leq M^* \leq M \leq c'M^*$ for some universal constant $c' > 1$.*

Condition 3.4 (Boundedness)**.** *For Tucker low-rank model* (5)*, there exists universal constants C_1 such that (i) The factor loading matrices satisfies $\|\boldsymbol{U}_m\|_{\max} \leq C_1$ for all $m \in [M]$; (ii) Elements in the core tensor $\mathcal{C}$ 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* $\boldsymbol{d} \in \mathbb{N}^{L+1}$*,* $B, M \in \mathbb{R}^+ \cup \{\infty\}$*, the family of deep ReLU Tensor Network truncated by* M *with depth* L*, width parameter* $\boldsymbol{d}$*, and weights bounded by* B *is defined as*

$$\mathcal{C}(L, \boldsymbol{d}, M, B) = \left\{ \bar{f}(\boldsymbol{\mathcal{X}}) = \mathcal{T}_M(f(\boldsymbol{\mathcal{X}})) \right\}$$

where

$$f(\boldsymbol{\mathcal{X}}) \text{ defined in (3) and (4) with } \left\| \boldsymbol{\mathcal{W}}^{(l)} \right\|_{\max} \le B, \ \left\| \boldsymbol{b}^{(l)} \right\|_{\max} \le B,$$

and $\mathcal{T}_M(\cdot)$ *applies truncation operator at level* C *to each entry of a* d_{L+1} *dimensional vector, that is,* $[\mathcal{T}_M(\mathcal{Z})]_{i_1 \cdots i_{M_L}} = \operatorname{sgn}(z_{i_1 \cdots i_{M_L}})(|z_{i_1 \cdots i_{M_L}}| \wedge M)$*. We denote it as* $\mathcal{C}(L, D_{in}, D_{out}, W, M, B)$ *if the width parameter* $\boldsymbol{d} = (D_{in}, W, W, \cdots, W, D_{out})$*, 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 ℓ_2 loss is defined as

$$R_n(f) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\boldsymbol{\mathcal{X}}_i))^2 \equiv \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \widetilde{f}(\boldsymbol{\mathcal{C}}_i) \right)^2, \tag{7}$$

where the last equality holds since $\boldsymbol{\mathcal{X}}$ 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,

$$\begin{aligned} \widehat{m}(\boldsymbol{\mathcal{X}}) = \widehat{f}(\boldsymbol{\mathcal{C}}) \quad \text{with} \\ R_n(\widehat{f}) \le \inf_{f \in \mathcal{C}(L, \bar{r}, 1, W, M, \infty)} R_n(f) + \delta_{opt} \end{aligned} \tag{8}$$

with some optimization error δ_{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* $\max \mathbb{E} |e_i|^2 \le c$ *for a universal constant* c*. Tensor feature* $\boldsymbol{\mathcal{X}}$ *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*

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^{n} |\widehat{m}(\boldsymbol{\mathcal{X}}_i) - m^*(\boldsymbol{\mathcal{C}}_i)|^2 + \mathbb{E}_{\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{C}}} \left[\widehat{m}(\boldsymbol{\mathcal{X}}_i) - m^*(\boldsymbol{\mathcal{C}}_i) \right]^2 \\ \le C \left(\delta_{opt} + \delta_{apr} + \delta_{sto} + \delta_{cor} + t n^{-1} \right) \end{aligned} \tag{9}$$

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

$$\delta_{apr} = \inf_{f \in \mathcal{C}(L, \bar{r}, 1, W, M, \infty)} \|f - m^*\|_\infty^2 \tag{10}$$

Stochastic error

$$\delta_{sto} = (W^2 L^2 + WLR) \log(WLR) \log n / n \tag{11}$$

Tensor core error

$$\delta_{cor} = \sigma^2 \left(\prod_{m=1}^{M} D_m \right)^{-1} \sum_{m=1}^{M} D_m R_m \tag{12}$$

Theorem 3.6 establishes a high probability bound on both the out-of-sample mean squared error $\mathbb{E}_{\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{C}}} [\widehat{m}(\boldsymbol{\mathcal{X}}_i) - m^*(\boldsymbol{\mathcal{C}}_i)]$ and in-sample mean squared error $\frac{1}{n} \sum_{i=1}^{n} |\widehat{m}(\boldsymbol{\mathcal{X}}_i) - m^*(\boldsymbol{\mathcal{C}}_i)|$. The error bound is composed of four terms: the optimization error δ_{opt}, the neural network approximation error δ_{apr} to the underlying true regression function m^*, the stochastic error δ_{sto} scales linearly with $\log n / n$ and $(W^2 L^2 + WLR) \log(WLR)$ which is proportional to the Pseudo-dimension of the neural network class we used, and the error δ_{cor} related to inferring the latent core tensor $\boldsymbol{\mathcal{C}}$ from the observation $\boldsymbol{\mathcal{X}}$. 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 δ_{apr} and the stochastic error δ_{sto}.

The NN approximation error δ_{apr} can be controlled by the architecture of NN and the optimization error δ_{opt} can be controlled by an optimization algorithm. Given the depth L and hidden width W of a NN, the stochastic error δ_{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 δ_{cor} remarkably well.

The benefit of Tucker low-rankness shows up in the stochastic error δ_{sto} where $R = \prod_{m=1}^{M} R_m$ is the total number of elements in the core tensor $\boldsymbol{\mathcal{C}}$ and is also equivalent to the total number of unknown coefficients in *low-rank weight* tensor $\boldsymbol{\mathcal{W}}^{(\ell)}$ 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) Comprised of the Subgraph Matching kernel (CSM) Kriege and Mutzel [2012], (2) Shortest Path Hash Graph Kernel (HGK-SP) Morris et al. [2016], (3) Weisfeiler-Lehman Subtree Kernel (HGK-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 Neural 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 (PersLay) 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 $\mathrm{PI}_{\mathrm{Dg}}$ from 20×20 to 50×50. 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-

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

Model	BZR	COX2	DHFR	D&D	MUTAG	PROTEINS	PTC_MR	PTC_MM	PTC_FM	PTC_FR	IMDB_B
CSM Kriege and Mutzel [2012]	84.54±0.65	79.78±1.04	77.99±0.96	OOT	87.29±1.25	OOT	58.24±2.44	63.30±1.70	63.80±1.00	65.51±9.82	OOT
HGK-SP Morris et al. [2016]	81.99±0.30	78.16±0.00	72.48±0.65	78.26±0.76	80.90±0.48	74.53±0.35	57.26±1.41	57.52±9.98	52.41±1.79	66.91±1.46	73.34±0.47
HGK-WL Morris et al. [2016]	81.42±0.60	78.16±0.00	75.35±0.66	79.01±0.43	75.51±1.34	74.53±0.35	59.90±4.30	67.22±5.98	64.72±1.66	67.90±1.81	72.75±1.02
WL Shervashidze et al. [2011]	86.16±0.97	79.67±1.32	81.72±0.80	79.78±0.36	85.75±1.96	73.06±0.47	57.97±0.49	67.28±0.97	64.80±0.85	67.64±0.74	71.15±0.47
DGCNN Zhang et al. [2018]	79.40±1.71	79.85±2.64	70.70±5.00	79.37±0.94	85.83±1.66	75.54±0.94	58.59±2.47	62.10±14.09	60.28±6.67	65.43±11.30	70.00±0.90
GCN Kipf and Welling [2017]	79.34±2.43	76.53±1.82	74.56±1.44	79.12±3.07	80.42±2.07	70.31±1.93	62.26±4.80	67.80±4.00	62.39±0.85	69.80±4.40	66.53±2.33
ChebNet Defferrard et al. [2016]	N/A	N/A	N/A	N/A	84.40±1.60	75.50±0.40	N/A	N/A	N/A	N/A	N/A
GIN Xu et al. [2018]	85.60±2.00	80.30±5.17	**82.20±4.00**	75.40±2.60	89.39±5.60	76.16±2.76	64.60±7.00	67.18±7.35	64.19±2.43	66.97±6.17	75.10±5.10
g-U-Nets Gao and Ji [2019]	79.40±1.20	80.30±4.21	69.10±4.80	75.10±2.20	67.61±3.36	69.60±3.50	64.70±6.80	67.51±5.96	65.88±4.26	66.28±3.71	N/A
MinCutPool Bianchi et al. [2020]	82.64±5.05	80.07±3.85	72.78±6.25	77.60±3.10	79.17±1.64	76.52±2.58	64.16±3.47	N/A	N/A	N/A	70.77±4.89
DiffPool Ying et al. [2018]	83.93±4.41	79.66±2.64	70.50±7.80	77.90±2.40	79.22±1.02	73.63±3.60	64.85±4.30	66.00±5.36	63.00±3.40	69.80±4.40	68.60±3.10
EigenGCN Ma et al. [2019]	83.05±6.00	80.16±5.80	N/A	75.90±3.90	79.50±0.66	74.10±3.10	N/A	N/A	N/A	N/A	70.40±3.30
CapsGNN Xinyi and Chen [2019]	N/A	N/A	N/A	75.38±4.17	86.67±6.88	76.28±3.63	66.00±5.90	N/A	N/A	N/A	N/A
SAGPool Lee et al. [2019]	82.95±4.91	79.45±2.98	74.67±4.64	76.45±0.97	76.78±2.12	71.86±0.97	56.41±1.63	66.67±8.57	67.65±3.72	65.71±10.69	74.87±4.09
HaarPool Wang et al. [2020]	83.95±5.68	82.61±2.69	73.33±3.72	77.40±3.40	90.00±3.60 dd	73.23±2.51	66.68±3.22	69.69±5.10	65.59±5.00	69.40±5.21	73.29±3.40
PersLay Carrière et al. [2020]	82.16±3.18	80.90±1.00	N/A	N/A	89.80±0.90	74.80±0.30	N/A	N/A	N/A	N/A	71.20±0.70
FC-V O'Bray et al. [2021]	85.61±0.59	81.01±0.88	81.43±0.48	N/A	87.31±0.66	74.54±0.48	N/A	N/A	N/A	N/A	73.84±0.36
MPR Bodnar et al. [2021a]	N/A	N/A	N/A	N/A	84.00±8.60	75.20±2.20	66.36±6.55	68.60±6.30	63.94±5.19	64.27±3.78	73.80±4.50
SIN Bodnar et al. [2021b]	N/A	N/A	N/A	N/A	N/A	76.50±3.40	66.80±4.56	70.55±4.79	68.68±6.80	69.80±4.36	75.60±3.20
TOGL Horn et al. [2021]	N/A	N/A	N/A	75.70±2.10	N/A	76.00±3.90	N/A	N/A	N/A	N/A	N/A
TTG-NN (ours)	**87.40 ± 2.62**	**86.73±3.41**	78.72±5.33	**80.90±2.57**	**93.65±4.18**	**77.62±3.92**	**68.91±4.02**	**74.11±4.57**	**69.33±2.09**	**73.23±3.91**	**76.40±2.50**

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

Dataset	GCN	GIN	GSN	PNA	TTG-NN (ours)
ogbg-molhiv	75.99 ± 1.19	77.07 ± 1.49	77.90 ± 0.10	79.05 ± 1.32	**81.50 ± 0.86**

performs TTG-NN without the tensor transformation layer (TTL) with an average relative gain of 3.41%. Furthermore, TTG-NN significantly outperforms TTG-NN without TG-CL on all datasets, which illustrates the importance of learning the tensor-view global structural information. In summary, these results show the effectiveness of both convolutional and topological tensor representation learning for the graph classification problem. In summary, ablating each of the above components leads to performance drops on all datasets compared with the full TTG-NN model, which suggests that the designed components are critical and need to be sufficiently learned.

Table 3: TTG-NN ablation study.

Architecture	MUTAG	BZR	COX2	PTC_MM	PTC_FM
TTG-NN W/o TT-CL	85.67±7.80	82.73±3.05	78.13±1.96	68.67±7.16	60.74±3.37
TTG-NN W/o TG-CL	90.60±3.15	86.14±6.31	79.58±1.84	68.20±7.50	62.42±1.96
TTG-NN W/o TTL	91.22±5.26	85.36±5.58	83.08±2.49	73.80±5.05	64.10±1.83
TTG-NN	**93.65±4.18**	**87.40±2.62**	**86.73±3.41**	**74.11±4.57**	**69.33±2.09**

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_MM and PTC_FM, we can see that TTG-NN equipped with TT outperforms TTG-NN with Tucker and CP decompositions respectively.

Table 4: Sensitivity analysis of tensor decomposition.

Decomposition	MUTAG	BZR	COX2	PTC_MM	PTC_FM
TTL With Tucker	88.89±9.94	86.91±4.41	79.23±6.89	66.12±5.52	61.03±3.89
TTL With TT	88.36±6.37	86.91±3.98	82.22±3.49	**74.11±4.57**	**69.33±2.09**
TTL With CP	**93.65±4.18**	**87.40±2.62**	**86.73±3.41**	67.00±5.23	62.45±5.31

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 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.

4.5 Computational Complexity

The topological complexity of the standard PH matrix reduction algorithm Edelsbrunner et al. [2000] runs in time at most $\mathcal{O}(\Xi^3)$, where Ξ is the number of simplices in a filtration. For 0-dimensional PH, it can be computed efficiently using disjoint sets with complexity $\mathcal{O}(\Xi\alpha^{-1}\Xi)$, where $\alpha^{-1}(\cdot)$ is the inverse Ackermann function Cormen et al. [2022]. Furthermore, in Table 5 we show the running time (i.e., training time per epoch) of the proposed TTL 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

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).

Decomposition	MUTAG	BZR	COX2	PTC_MM	PTC_FM
TTG-NN With Tucker	13.41 s	44.71 s	7.52 s	3.73 s	4.47 s
TTG-NN With TT	17.58 s	36.28 s	7.20 s	3.97 s	3.60 s
TTG-NN With CP	18.58 s	45.02 s	17.55 s	12.72 s	12.85 s

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

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):249–270, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *Proceedings of the International Conference on Learning Representations*, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *Proceedings of the International Conference on Learning Representations*, 2018.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

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

Y. Chen, Y. R. Gel, and H. V. Poor. BScNets: Block simplicial complex neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6333–6341, 2022a.

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

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 922–929, 2019.

Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems*, 21(9):3848–3858, 2019.

Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. *Advances in neural information processing systems*, 33:17804–17815, 2020.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *Advances in neural information processing systems*, 31, 2018.

Kexin Huang, Cao Xiao, Lucas M Glass, Marinka Zitnik, and Jimeng Sun. Skipgnn: predicting molecular interactions with skip-graph networks. *Scientific reports*, 10(1):1–16, 2020.

Ruoxi Sun, Hanjun Dai, and Adams Wei Yu. Does gnn pretraining help molecular representation? *Advances in Neural Information Processing Systems*, 35:12096–12109, 2022.

Larry Wasserman. Topological data analysis. *Annual Review of Statistics and Its Application*, 5:501–532, 2018.

Gunnar Carlsson and Rickard Brüel Gabrielsson. Topological approaches to deep learning. In *Topological Data Analysis*, pages 119–146. Springer, 2020.

L. O'Bray, B. Rieck, and K. Borgwardt. Filtration curves for graph representation. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1267–1275, 2021.

Herbert Edelsbrunner and John L Harer. *Computational topology: an introduction*. American Mathematical Society, 2022.

Peter Bubenik. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16:77–102, 2015.

Genki Kusano, Yasuaki Hiraoka, and Kenji Fukumizu. Persistence weighted gaussian kernel for topological data analysis. In *International conference on machine learning*, pages 2004–2013. PMLR, 2016.

Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4741–4748, 2015.

C. Hofer, R. Kwitt, M. Niethammer, and A. Uhl. Deep learning with topological signatures. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

M. Carrière, F. Chazal, Y. Ike, T. Lacombe, M. Royer, and Y. Umeda. Perslay: A neural network layer for persistence diagrams and new graph topological signatures. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 2786–2796, 2020.

Q. Zhao, Z. Ye, C. Chen, and Y. Wang. Persistence enhanced graph neural network. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 2896–2906, 2020.

Yuzhou Chen, Baris Coskunuzer, and Yulia Gel. Topological relational learning on graphs. *Advances in neural information processing systems*, 34:27029–27042, 2021.

Z. Yan, T. Ma, L. Gao, Z. Tang, and C. Chen. Link prediction with persistent homology: An interactive view. In *Proceedings of International Conference on Machine Learning*, pages 11659–11669, 2021.

M. Horn, E. De Brouwer, M. Moor, Y. Moreau, B. Rieck, and K. Borgwardt. Topological graph neural networks. In *Proceedings of International Conference on Learning Representations*, 2021.

Yuzhou Chen, Ignacio Segovia-Dominguez, Baris Coskunuzer, and Yulia Gel. Tamp-s2gcnets: coupling time-aware multipersistence knowledge representation with spatio-supra graph convolutional networks for time-series forecasting. In *International Conference on Learning Representations*, 2022b.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1: 57–81, 2020.

Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. Graph learning: A survey. *IEEE Trans AI*, 2(2):109–127, 2021.

Yu Zhou, Haixia Zheng, Xin Huang, Shufeng Hao, Dengao Li, and Jumin Zhao. Graph neural networks: Taxonomy, advances, and trends. *ACM Trans. Intell. Syst. Technol.*, 13(1), jan 2022. ISSN 2157-6904. doi: 10.1145/3495161. URL `https://doi.org/10.1145/3495161`.

Z. Zhang, P. Cui, and W. Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge & amp; Data Engineering*, 34(01):249–270, jan 2022. ISSN 1558-2191. doi: 10.1109/TKDE.2020.2981333.

Ijeoma Amuche Chikwendu, Xiaoling Zhang, Isaac Osei Agyemang, Isaac Adjei-Mensah, Ukwuoma Chiagoziem Chima, and Chukwuebuka Joseph Ejiyi. A comprehensive survey on deep graph representation learning methods. *J. Artif. Int. Res.*, 78, dec 2023. ISSN 1076-9757. doi: 10.1613/jair.1.14768. URL `https://doi.org/10.1613/jair.1.14768`.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Christopher Morris, Nils M Kriege, Kristian Kersting, and Petra Mutzel. Faster kernels for graphs with continuous attributes via hashing. In *IEEE International Conference on Data Mining*, pages 1095–1100, 2016.

Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *Workshop on Relational Representation Learning, NeurIPS*, 2018.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the International Conference on Machine Learning*, pages 2083–2092, 2019.

Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016.

Jean Kossaifi, Aran Khanna, Zachary C. Lipton, Tommaso Furlanello, and Anima Anandkumar. Tensor contraction layers for parsimonious deep nets. *CVPR*, pages 1940–1946, 2017.

Jean Kossaifi, Zachary C. Lipton, Arinbjorn Kolbeinsson, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *JMLR*, 21:1–21, 2020.

Yao Lei Xu, Kriton Konstantinidis, and Danilo P Mandic. Graph tensor networks: An intuitive framework for designing large-scale neural learning systems on multiple domains. *arXiv preprint arXiv:2303.13565*, 2023.

Serguei Barannikov. The framed morse complex and its invariants. 02 1994. doi: 10.1090/advsov/021/03.

Krzysztof Choromanski, Han Lin, Haoxian Chen, Tianyi Zhang, Arijit Sehanobish, Valerii Likhosherstov, Jack Parker-Holder, Tamas Sarlos, Adrian Weller, and Thomas Weingarten. From block-toeplitz matrices to differential equations on graphs: towards a general theory for scalable masked transformers. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 3962–3983. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/choromanski22a.html`.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. In *Proceedings of the International Conference on Machine Learning*, pages 291–298, 2012.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, volume 32, 2018.

Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *Proceedings of the International Conference on Machine Learning*, pages 874–883, 2020.

Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In *SIGKDD*, pages 723–731, 2019.

Zhang Xinyi and Lihui Chen. Capsule graph neural network. In *International conference on learning representations*, 2019.

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

Yu Guang Wang, Ming Li, Zheng Ma, Guido Montufar, Xiaosheng Zhuang, and Yanan Fan. Haar graph pooling. In *Proceedings of the International Conference on Machine Learning*, pages 9952–9962, 2020.

Cristian Bodnar, Cătălina Cangea, and Pietro Liò. Deep graph mapper: Seeing graphs through the neural lens. *Frontiers in Big Data*, page 38, 2021a.

C. Bodnar, F. Frasca, Y. G. Wang, N. Otter, G. Montúfar, P. Lio, and M. Bronstein. Weisfeiler and Lehman go topological: Message passing simplicial networks. In *Proceedings of the International Conference on Machine Learning*, 2021b.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In *FOCS*, pages 454–463, 2000.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18, 2017.

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 $\boldsymbol{\mathcal{X}}$ of dimension $D_1 \times \cdots \times D_M$. If $\boldsymbol{\mathcal{X}}$ assumes a (canonical) rank-R *CP low-rank* structure, then it can be expressed as

$$\boldsymbol{\mathcal{X}} = \sum_{r=1}^{R} c_r \, \boldsymbol{u}_{1r} \circ \boldsymbol{u}_{2r} \circ \cdots \boldsymbol{u}_{Mr}, \tag{13}$$

where $\circ$ denotes the outer product, $\boldsymbol{u}_{mr} \in \mathbb{R}^{D_m}$ and $\|\boldsymbol{u}_{mr}\|_2 = 1$ for all mode $m \in [M]$ and latent dimension $r \in [R]$. Concatenating all R vectors corresponding to a mode m, we have $\boldsymbol{U}_m = [\boldsymbol{u}_{m1}, \cdots, \boldsymbol{u}_{mR}] \in \mathbb{R}^{D_m \times R}$ which is referred to as the loading matrix for mode $m \in [M]$.

If $\boldsymbol{\mathcal{X}}$ assumes a rank-$(R_1, \cdots, R_M)$ *Tucker low-rank* structure (5), then it writes

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

where $\boldsymbol{u}_{mr_m}$ 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 $\boldsymbol{\mathcal{C}}$.

Tensor Train (TT) low-rank Oseledets [2011] approximates a $D_1 \times \cdots \times D_M$ tensor $\boldsymbol{\mathcal{X}}$ with a chain of products of third order *core tensors* $\boldsymbol{\mathcal{C}}_i$, $i \in [M]$, of dimension $R_{i-1} \times D_i \times R_i$. Specifically, each element of tensor $\boldsymbol{\mathcal{X}}$ can be written as

$$x_{i_1, \cdots, i_M} = \boldsymbol{c}_{1,1,i_1,:}^{\top} \times \boldsymbol{c}_{2,:,i_2,:} \times \cdots \times \boldsymbol{c}_{M,:,i_M,:} \times \boldsymbol{c}_{M+1,:,1,1}, \tag{14}$$

where $\boldsymbol{c}_{m,:,i_m,:}$ 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 $\boldsymbol{\mathcal{C}}_1$ 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. $\boldsymbol{c}_{1,1,i_1,:}$) is actually a R_1 vector. To deal with the "boundary condition" at the end, we augmented the chain with an additional tensor $\boldsymbol{\mathcal{C}}_{M+1}$ 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 (13) is a special case where the core tensor $\boldsymbol{\mathcal{C}}$ 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., $\mathrm{PH}(\mathcal{G}, f_i) = \overrightarrow{Dg}_i = \{Dg_i^{(1)}, \ldots, Dg_i^{(\mathcal{Q})}\}$, where $\mathcal{Q} \in \mathbb{Z}_0^+$ 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) Adams et al. [2017]. 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 $\varrho_{Dg} : \mathbb{R}^2 \mapsto \mathbb{R}^2$, which is referred to as a persistence surface. The persistence surface ϱ_{Dg} is constructed by summing weighted Gaussian kernels centered at each point in Dg. In the second step, we integrate the persistence surface ϱ_{Dg} over each grid box to obtain the value of the PI_{Dg}.

Table 6: The main symbols and definitions in this paper.

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

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 $PI_{Dg}(z) = \iint_z \sum_{\mu \in T(Dg)} \frac{g(\mu)}{2\pi\delta_x\delta_y} e^{-\left(\frac{(x-\mu_x)^2}{2\delta_x^2} + \frac{(y-\mu_y)^2}{2\delta_y^2}\right)} dydx$, 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 δ_x and δ_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.

Table 7: Summary statistics of the benchmark datasets.

Dataset	# Graphs	Avg. $\lvert\mathcal{V}\rvert$	Avg. $\lvert\mathcal{E}\rvert$	# 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

Dataset/filtrations	deg.	betw.	eigen.	close.	[deg., betw.]	[deg., eigen.]	[deg., close.]	[deg., betw., eigen.]	[deg., betw., close.]	[deg., eigen., close.]	[deg., betw., eigen., close.]
MUTAG	86.17 ± 6.23	80.29 ± 10.96	83.42 ± 11.64	82.43 ± 7.87	88.89 ± 5.79	85.12 ± 6.65	86.23 ± 6.68	88.36 ± 8.08	88.30 ± 6.12	87.31 ± 1.00	93.65 ± 4.18
PROTEINs	72.32 ± 6.43	74.48 ± 5.01	72.75 ± 6.15	73.50 ± 4.88	73.31 ± 6.17	74.39 ± 4.56	73.68 ± 5.71	74.75 ± 5.47	73.22 ± 7.04	73.76 ± 6.61	77.62 ± 3.92