Revisiting Transformation Invariant Geometric Deep Learning: An Initial Representation Perspective

Ziwei Zhang, *Member, IEEE,* Xin Wang, *Member, IEEE,* Zeyang Zhang, Peng Cui, *Senior Member, IEEE,* and Wenwu Zhu, *Fellow, IEEE*

Abstract—Deep neural networks have achieved great success in the last decade. When designing neural networks to handle the ubiquitous geometric data such as point clouds and graphs, it is critical that the model can maintain invariance towards various transformations such as translation, rotation, and scaling. Most existing graph neural network (GNN) approaches can only maintain permutation-invariance, failing to guarantee invariance with respect to other transformations. Besides GNNs, other works design sophisticated transformation-invariant layers, which are computationally expensive and difficult to be extended. In this paper, we revisit why general neural networks cannot maintain transformation invariance. Our findings show that transformation-invariant and distance-preserving initial point representations are sufficient to achieve transformation invariance rather than needing sophisticated neural layer designs. Motivated by these findings, we propose Transformation Invariant Neural Networks (TinvNet), a straightforward and general plug-in for geometric data. Specifically, we realize transformation invariant and distance-preserving initial point representations by modifying multi-dimensional scaling and feed the representations into existing neural networks. We prove that TinvNet can strictly guarantee transformation invariance, being general and flexible enough to be combined with the existing neural networks. Extensive experimental results on point cloud analysis and combinatorial optimization demonstrate the effectiveness and general applicability of our method. We also extend our method into equivariance cases. Based on the results, we advocate that TinvNet should be considered as an essential baseline for further studies of transformation-invariant geometric deep learning.

Index Terms—Transformation Invariance, Geometric Deep Learning, Combinatorial Optimization, Point Cloud, Graph Neural Network

1 Introduction

Deep neural networks [1] have achieved enormous successes in many fields such as computer vision [2], natural language processing [3], and game playing [4]. On the other hand, geometric data, such as graphs or point clouds, is ubiquitous in practice, ranging from molecules in proteins to 3D objects. Compared with grid-structured data such as acoustics, images, or videos, geometric data poses more challenges for designing suitable neural networks due to the irregular structure [5].

To handle geometric data effectively, one critical *inductive bias* is to design invariant and equivariant models with respect to various transformations such as permutation, translation, rotation, reflection, and scaling. Take 3D object recognition in point cloud analysis as an example. The shape of an object is invariant to isometric transformations such as translation, rotation, and reflection. For the classic NP-hard travelling salesman problem (TSP), the solution of a TSP is invariant to isometric and the scaling transformations of the coordinates. Maintaining invariance and equivariance with respect to these transformations can enhance the generalization ability, robustness, and interpretability of neural networks in handling geometric data [5], [6]. However, designing transformation invariant neural networks for geometric data poses great challenges. For instance, though convolutional neural networks (CNNs) are known to enjoy and benefit from translation equivariance in handling images and videos [7], it is non-trivial to extend such merit to geometric data since there is no grid structure.

As an emerging type of neural networks to process geometric data, graph neural networks (GNNs) have been shown effective in a wide range of geometric applications such as protein interaction prediction [8], point cloud analysis [9], combinatorial optimization [10], etc. Maintaining permutation-equivariance is a crucial reason behind the success of GNNs, i.e., if we randomly permute the IDs of nodes, the representations produced by GNNs are permuted accordingly. By adopting the message-passing framework [11], most GNNs can easily satisfy permutation-equivariance [12], [13]. However, most GNNs largely ignore other transformations mentioned above, e.g., rotation and scaling. Ideally, GNNs should be able to produce equivariant or invariant representations when geometric data is transformed. But the existing GNNs consider transformed data as independent samples, failing to produce desired representations.

Other attempts to alleviate the problems caused by transformations include data augmentation and manually extracting transformation invariant features such as the distance and angle between geometric objects [14], [15]. Due to the massive number of possible transformations, data augmentation cannot guarantee effectiveness and works poorly in practice. Meanwhile, manually designed features can only preserve a limited amount of information. Some works

- *All authors are with the Department of Computer Science and Technology at Tsinghua University, Beijing 100084, China. Email: {zwzhang,xin_wang,cuip,wwzhu}@tsinghua.edu.cn, zy-zhang20@mails.tsinghua.edu.cn*

Manuscript received April 19, 2005; revised August 26, 2015.

have designed specific neural network layers targeting certain transformations such as rotations [16], [17]. Though having made some progresses, these methods usually resort to complicated group theories and geometric analysis, suffering from being computationally expensive and difficult to be extended [17]. For example, when incorporating the attention mechanism [18] into an existing rotation-equivariant network named tensor field network [16], great efforts are needed to redesign all operators [17].

To address this problem, we first revisit why the existing neural networks cannot maintain transformation invariance when handling geometric data. We find that transformation-invariant and distance-preserving initial point representations are sufficient to achieve transformation invariance rather than needing to design sophisticated neural network layers as proposed in the existing methods. Motivated by these findings, we propose Transformation Invariant Neural Network (TinvNet), a straightforward and general plug-in for geometric data. Specifically, we realize transformation invariant and distance-preserving initial point representations by modifying multi-dimensional scaling (MDS), a classical dimensionality reduction technique. We then feed the initial representation into neural networks. We prove that such a simple mechanism can strictly guarantee transformation invariance. Besides, since TinvNet is a general framework compatible with existing neural networks, it is flexible to be combined with various architectures such as different GNN variants or other neural networks for geometric data. We further provide extending our method to equivariance cases in Appendix C.

We conduct extensive experiments on tasks including point cloud analysis and combinatorial optimization. The results show that TinvNet is indeed strictly invariant to various transformations such as translation, rotation, reflection, and scaling. In the rotation transformation setting of point cloud analysis, TinvNet combined with DGCNN [9], a well-known non-rotation-invariant GNN model, outperforms or matches the performance of various recently proposed models specifically designed to be rotation-invariant. Our proposed model also significantly outperforms a recent learning-based model for combinatorial problems when transformations are involved. Besides, thanks to the simple and general mechanism of TinvNet, it is easily compatible with multiple architectures. Based on the experimental results, we advocate that TinvNet should be considered a new starting point and an essential baseline for further studies of transformation-invariance on geometric data. Our contributions are summarized as follows:

- We revisit transformation invariant geometric deep learning and show that transformation-invariant and distance-preserving initial point representation is sufficient to solve the problem.
- Motivated by the findings, we propose Transformation Invariant Neural Network, a straightforward and general plug-in that is proved to be strictly transformation-invariant as well as general and flexible to combine with various neural networks.
- Extensive experimental results on tasks including point cloud analysis and combinatorial optimization demonstrate the efficacy and general applicability of our model.

The rest of the paper is organized as follows. In Section 2, we review related work. The problem formulation is introduced in Section 3. We revisit the transformation invariance problem and propose the TinvNet model in Section 4. We report experimental results in Section 5, and conclude our paper in Section 6.

2 Related Work

In this section, we first review GNNs and their permutation equivariance and invariance properties. Then, we review other invariance for geometric deep learning.

2.1 Graph Neural Networks and Permutation Equivariance/Invariance

GNNs are one emerging type of neural networks to process geometric data. Early GNNs such as recursive architectures [19], [20] and contextual realizations [21] predate the rise of deep neural networks. Nevertheless, it is not until the deep learning era that GNNs gain popularity. Recent advances in GNNs can be broadly categorized into spectral approaches [22], [23], [24] and spatial approaches [25], [26], [27]. For spectral approaches, graph signal processing techniques [28], [29] are adopted to process graph data in the spectral domain. For spatial approaches, the neural networks directly work on the connectivity patterns of graphs. Due to the efficiency and effectiveness, the message-passing framework [11] is a de facto standard in recent GNN designs, including GCN [30], GraphSAGE [31], GAT [32], JK-Nets [33], GIN [34], and Graph Nets [35] as particular instantiations.

One fundamental property of the message-passing GNNs is permutation-equivariance, i.e., the node representations are not dependent on node IDs. For example, many studies [36], [37], [38] analyze the connection between GNNs and the Weisfeiler-Lehman (WL) algorithm [39] of graph isomorphism tests. Since the WL algorithm is strictly permutation-equivariant, GNNs also need to be strictly permutation-equivariant to mimic WL algorithms. By applying permutation-invariant pooling layers [40], [41], [42] on permutation-equivariant node representations, permutation-invariant graph representations can be obtained [12], [13]. Permutation is orthogonal to the similarity transformations studied in this paper. By adopting permutation-equivariant GNNs as backbones, our model can also satisfy permutation equivariance and invariance.

2.2 Other Invariance for Geometric Deep Learning

Rotation-invariant and equivariant neural networks for geometric data have been studied previously [6], [43], [44], particularly in point clouds [15], [16], [45], [46]. Most of these methods do not consider other transformations such as scaling, translation, or reflection. Besides, these methods design sophisticated neural network layers inspired by group theories and geometric analysis to guarantee rotation-invariance and equivariance. In general, these methods are complicated, computationally expensive, and difficult to be extended. In comparison, our proposed method is simple and straightforward. We also empirically compare our method with these methods in Section 5.1.

TABLE 1: A Summary of Notations

Symbol	Meaning
$\mathcal{V} = \{v_1, ..., v_N\}$	The set of points
$\mathbf{F} \in \mathbb{R}^{N \times d}$	The coordinate matrix
$\mathcal{D}(\cdot, \cdot), \mathbf{D}$	The distance metric and distance matrix
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A graph
$\mathbf{W}$	Learnable parameters in the neural network
$\mathcal{T}(\cdot)$	A similarity transformation
c, c'	Scaling constants
$\mathbf{H}^{(l)}$	The point representation in the l^{th} layer
$\mathbf{H}^{(0)} = \mathcal{P}(\mathbf{F})$	The initial representation and mapping function
$\mathbf{H} = \mathbf{H}^{(L)}$	The final point representation
$\mathbf{\Lambda}, \mathbf{X}$	Eigenvalues and the corresponding eigenvectors
$\mathbf{S}$	The similarity matrix
$\mathbf{I}_N, \mathbf{1}_N$	$N \times N$ identity matrix/matrix of ones

A scale-invariant GNN is proposed in [47] to handle different scales of node features. However, it cannot handle other transformations such as translation and rotation. Very recently, geometrically invariant and equivariant GNNs began to receive attentions [48]. For example, IsoGCN [49] is proposed to handle isometric transformations and EGNN [50] is proposed to handle various transformations by designing sophisticated message-passing functions. In general, these methods also design sophisticated neural network layers, e.g., certain types of message-passing, to realize invariance and equivariance. In comparison, our method studies the problem from another perspective and is more straightforward and compatible with existing neural networks.

There are also recent works combining eigen-analysis and neural networks for geometric learning called intrinsic coordinations. For example, IEConv [51] adopts one extrinsic and two intrinsic distances and designs a new convolution operator for protein modeling. Koestler et al. [52] proposes an intrinsic neural field method for shapes. TinvNet can also be regarded as a type of intrinsic coordinates. In comparison, our method is more straight-forward and compatible (only operating in the initial representation), empirically more effective, and fully distance-preserving.

3 Problem Formulation

In this section, we introduce notations and preliminaries of transformation invariance and similarity transformation. We summarize notations in Table 1.

We consider each geometric data instance as a collection of points $\mathcal{V} = \{v_1, v_2, ..., v_N\}$ with N denoting the number of points. The points have a coordinate matrix $\mathbf{F} \in \mathbb{R}^{N \times d}$ with d denoting the dimensionality. Denote by $\mathbf{F}_{i,:}$, $\mathbf{F}_{:,j}$, and $\mathbf{F}_{i,j}$, the i^{th} row, j^{th} column, and an element of the matrix, respectively. $\mathbf{F}_{i,:}$ is the coordinate of point v_i. We denote a symmetric distance metric associated with the coordinates as $\mathcal{D}(\cdot, \cdot)$. In this paper, we assume the metric is the Euclidean distance by default. There is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to describe the relationships between points, where $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. The graph can be provided in the data or constructed from the coordinates, e.g., the k-nearest neighbors graph. We denote the adjacency matrix of the graph as $\mathbf{A}$. $\mathcal{N}(i) = \{v_j : (v_i, v_j) \in \mathcal{E}\}$ is the neighborhood of v_i.

Neural networks for geometric data usually aim to learn representation $\mathbf{H}$ for the points using the coordinate matrix $\mathbf{F}$ and the graph $\mathbf{A}$. We generally denote such neural networks as[1]

$$\mathbf{H} = \mathrm{NN}\left(\mathbf{F}, \mathbf{A}; \mathbf{W}\right), \tag{1}$$

where $\mathbf{W}$ are learnable parameters. We mainly study how to maintain invariance with respect to different transformations applied to the geometric data.

Definition 1 (Transformation Invariance)**.** *For a given transformation $\mathcal{T}(\cdot) : \mathbb{R}^d \to \mathbb{R}^d$, a neural network following Eq.* (1) *is transformation invariant if $\forall$ $\mathbf{F}$, $\mathbf{A}$, $\mathbf{W}$, the following equation holds:*

$$\mathrm{NN}\left(\mathcal{T}(\mathbf{F}), \mathbf{A}; \mathbf{W}\right) = \mathrm{NN}\left(\mathbf{F}, \mathbf{A}; \mathbf{W}\right), \tag{2}$$

i.e., the model outputs identical point representations after the transformation.

Transformation invariance in Definition 1 is compositional, i.e., if a model is invariant with respect to both $\mathcal{T}_1(\cdot)$ and $\mathcal{T}_2(\cdot)$, the model is also invariant with respect to $\mathcal{T}_2\left(\mathcal{T}_1(\cdot)\right)$. Thus, we can study invariance for basic transformations, and the results hold for a combination of these transformations. In this paper, we mainly consider similarity transformations.

Definition 2 (Similarity Transformation)**.** *A similarity transformation is an arbitrary combination of the following transformations: (1) (Uniform) Scaling: the coordinate matrix is scaled by a constant, i.e., $\mathcal{T}(\mathbf{F}) = c\mathbf{F}$, where $c \neq 0$ is a constant. (2) Isometric transformation: any transformation that is isometric with respect to the metric $\mathcal{D}(\cdot, \cdot)$, i.e.,*

$$\left\{\mathcal{T}(\cdot) : \mathcal{D}\left(\mathbf{F}_{i,:}, \mathbf{F}_{j,:}\right) = \mathcal{D}\left(\mathcal{T}(\mathbf{F})_{i,:}, \mathcal{T}(\mathbf{F})_{j,:}\right), \forall \mathbf{F}, i, j\right\}. \tag{3}$$

For the Euclidean distance, isometric transformations include rotation, translation, and reflection.

Without proper designs, most message-passing GNNs and other neural networks for point clouds cannot guarantee transformation invariance for geometric data.

4 Methodology

In this section, we first revisit transformation invariance, and then introduce our proposed method and provide some discussions. We further provide extending our method to equivariance cases in Appendix C.

4.1 Revisiting Transformation Invariance

To investigate why typical neural network are not transformation invariant, we revisit transformation invariance of geometric data. We use GNNs as examples, but the analyses generalize to other neural networks following Eq. (1).

We denote $\mathrm{AGG}^{(l)}(\cdot)$ as an order-invariant aggregation function and $\mathrm{COMBINE}^{(l)}(\cdot)$ as a combining function. The message-passing framework of GNNs [11] is formulated as:

$$\begin{aligned} \mathbf{m}_i^{(l)} &= \mathrm{AGG}^{(l)}\left(\left\{\mathbf{h}_j^{(l)}, \forall j \in \mathcal{N}(i)\right\}\right) \\ \mathbf{h}_i^{(l+1)} &= \sigma\left(\mathrm{COMBINE}^{(l)}\left[\mathbf{m}_i^{(l)}, \mathbf{h}_i^{(l)}\right]\right), \end{aligned} \tag{4}$$

1. For non-graph-based neural networks for geometric data, $\mathbf{A}$ can be removed from Eq. (1). For notation convenience, we keep $\mathbf{A}$ in Eq. (1).

where $\mathbf{h}_i^{(l)}$ denotes the representation of point v_i at the l^{th} layer, $\mathbf{m}_i^{(l)}$ is the message vector for point v_i, and $\sigma(\cdot)$ is an activation function. We denote $\mathbf{H}^{(l)} = \left[\mathbf{H}_1^{(l)}, ..., \mathbf{H}_N^{(l)}\right]$ as the representation of all the points. The initial representation is

$$\mathbf{H}^{(0)} = \mathcal{P}\left(\mathbf{F}\right), \tag{5}$$

where $\mathcal{P}(\cdot)$ is the mapping function. The final representation is $\mathbf{H} = \mathbf{H}^{(L)}$, where L is the number of layers. We easily have the following remark.

Remark 1. *A GNN following Eq.* (4) *is transformation-invariant if $\mathcal{P}(\cdot)$ is transformation-invariant.*

The remark can be proven by mathematical induction, i.e., if $\mathbf{H}^{(l)}$ is transformation-invariant, $\mathbf{H}^{(l+1)}$ is also transformation-invariant. Remark 1 shows that to empower GNNs to be transformation invariant, we simply need to ensure that the initial mapping function is transformation-invariant. However, the existing GNNs directly adopt the coordinates as the initial representations, i.e., $\mathcal{P}(\mathbf{F}) = \mathbf{F} = \mathbf{H}^{(0)}$, and thus cannot satisfy Remark 1. It is natural to ask: can we have a principled method to obtain transformation-invariant initial representation from the coordinates? If so, we can realize transformation invariant neural networks without modifying the message-passing mechanism.

Manually designing heuristics is obviously one choice. For example, we can calculate the distances and angles of points with their nearest neighbors, i.e., the kNN method. However, vital information may be lost in the heuristics, leading to sub-optimal results. Ideally, we expect the mapping function $\mathcal{P}(\cdot)$ to be "information lossless' so that $\mathbf{H}^{(0)}$ contains the same amount of information as $\mathbf{F}$.

For transformation-invariant geometric problems, useful information is encoded in the relative distance between points instead of the coordinates per se. Thus, if $\mathbf{H}^{(0)}$ can be distance-preserving, it is safe to say it is information lossless. We formulate the distance-preserving requirement as:

$$\mathcal{D}(\mathbf{H}_{i,:}^{(0)}, \mathbf{H}_{j,:}^{(0)}) = \mathcal{D}\left(\mathbf{F}_{i,:}, \mathbf{F}_{j,:}\right), \forall i, j. \tag{6}$$

Besides, the transformation-invariant requirement is formulated as

$$\mathcal{P}\left(\mathbf{F}\right) = \mathcal{P}\left(\mathcal{T}\left(\mathbf{F}\right)\right), \forall\, \mathbf{F}, \tag{7}$$

where $\mathcal{T}(\cdot)$ is any transformation in Definition 2.

However, there exists a conflict between Eq. (6) and Eq. (7) for the scaling transformation. Specifically, when $\mathcal{T}(\mathbf{F}) = c\mathbf{F}$, the distance between points scales accordingly, i.e., $\mathcal{D}\left(c\mathbf{F}_{i,:}, c\mathbf{F}_{j,:}\right) = c\mathcal{D}\left(\mathbf{F}_{i,:}, \mathbf{F}_{j,:}\right)$, since the Euclidean distance has homogeneity of degree 1. However, Eq. (7) requires the new features to be invariant, i.e., $\mathbf{H}^{(0)} = \mathcal{P}\left(c\mathbf{F}\right) = \mathcal{P}\left(\mathbf{F}\right)$, and thus $\mathcal{D}(\mathbf{H}_{i,:}^{(0)}, \mathbf{H}_{j,:}^{(0)})$ is also invariant. Therefore, Eq. (6) cannot hold when $c \neq 1$.

To solve that conflict, we relax Eq. (6) by adding an additional scaling term, i.e., assuming there exists a constant c' so that

$$\mathcal{D}\left(\mathbf{H}_{i,:}^{(0)}, \mathbf{H}_{j,:}^{(0)}\right) = c'\mathcal{D}\left(\mathbf{F}_{i,:}, \mathbf{F}_{j,:}\right), \forall i, j. \tag{8}$$

In other words, the distance-preserving requirement is relaxed to not care for the absolute scale of the distance, but only preserve the relative ratios between different distances.

In summary, we require $\mathcal{P}(\cdot)$ to simultaneously satisfy Eqs. (7) (8). Then, using Remark 1, we can adopt the initial point representation obtained by $\mathcal{P}(\cdot)$ to realize transformation-invariant neural networks. Notice that the solution to these two constraints is not unique. Therefore, we also require $\mathcal{P}(\cdot)$ to work in a deterministic way to ensure invariance. Next, we introduce our proposed plug-in to instantiate $\mathcal{P}(\cdot)$ that satisfies these requirements.

4.2 The TinvNet Method

In this section, we present our proposed method based on the findings in Section 4.1. Specifically, we find that we can easily achieve the goal by slightly modifying multi-dimensional scaling (MDS), a classical dimensionality reduction technique [53], as a plug-in for the existing neural networks.

The core idea of MDS is to obtain distance-preserving features by an eigen-decomposition problem. Specifically, we denote the distance matrix as $\mathbf{D}_{i,j} = \mathcal{D}\left(\mathbf{F}_{i,:}, \mathbf{F}_{j,:}\right)$ and further construct a similarity matrix $\mathbf{S} \in \mathbb{R}^{N\times N}$ as follows:

$$\mathbf{S}_{i,j} = -\frac{1}{2}\mathbf{D}_{i,j}^2 = -\frac{1}{2}\left(\mathcal{D}\left(\mathbf{F}_{i,:}, \mathbf{F}_{j,:}\right)\right)^2. \tag{9}$$

Then, we center the similarity matrix by

$$\tilde{\mathbf{S}}_{i,j} = \mathbf{S}_{i,j} - \bar{\mathbf{S}}_{i,\cdot} - \bar{\mathbf{S}}_{\cdot,j} + \bar{\mathbf{S}}_{\cdot,\cdot}, \tag{10}$$

where

$$\begin{aligned} \bar{\mathbf{S}}_{i,\cdot} = \frac{1}{N}\sum\nolimits_{k=1}^{N}\mathbf{S}_{i,k},\ \bar{\mathbf{S}}_{\cdot,j} = \frac{1}{N}\sum\nolimits_{l=1}^{N}\mathbf{S}_{l,j}, \\ \bar{\mathbf{S}}_{\cdot,\cdot} = \frac{1}{N^2}\sum\nolimits_{k=1}^{N}\sum\nolimits_{l=1}^{N}\mathbf{S}_{k,l}, \end{aligned} \tag{11}$$

i.e., the average of the i^{th} row, the average of the j^{th} column, and the average of the matrix, respectively. We can combine Eqs. (9) (10) in an equivalent matrix form:

$$\tilde{\mathbf{S}} = -\frac{1}{2}\left(\mathbf{I}_N - \frac{1}{N}\mathbf{1}_N\right)\left(\mathbf{D}\odot\mathbf{D}\right)\left(\mathbf{I}_N - \frac{1}{N}\mathbf{1}_N\right), \tag{12}$$

where $\mathbf{I}_N$ is a $N\times N$ identity matrix, $\mathbf{1}_N$ is a $N\times N$ matrix of ones, and $\odot$ is the Hadamard product.

$\tilde{\mathbf{S}}$ is a $N\times N$ matrix containing the information of distances. To reduce the dimensionality, we calculate the eigen-decomposition of $\tilde{\mathbf{S}}$. We denote the eigenvalues of $\tilde{\mathbf{S}}$ sorted in descending order as a diagonal matrix $\mathbf{\Lambda}$, i.e., $\mathbf{\Lambda}_{1,1} \geq \mathbf{\Lambda}_{2,2} \geq ... \geq \mathbf{\Lambda}_{N,N}$ are eigenvalues, and $\mathbf{X}$ is a matrix of eigenvectors with $\mathbf{X}_{:,i}$ being the eigenvector associated with $\mathbf{\Lambda}_{i,i}$. The point representation is:

$$\tilde{\mathbf{H}}^{(0)} = \mathbf{X}\sqrt{\mathbf{\Lambda}}. \tag{13}$$

However, the original MDS in Eq. (13) can only satisfy Eq. (6) but not Eq. (8). Thus, we slightly modify Eq. (13) to a normalized form

$$\mathbf{H}^{(0)} = \mathbf{X}\sqrt{\frac{\mathbf{\Lambda}}{\mathbf{\Lambda}_{1,1}}}. \tag{14}$$

From the properties of MDS, Eq. (14) exactly produces our desired $\mathbf{H}^{(0)} = \mathcal{P}(\mathbf{F})$. We formalize the results as follows.

Algorithm 1 TinvNet: A Transformation-Invariant Neural Network Plug-in

Require: The coordinate matrix $\mathbf{F}$, the distance metric $\mathcal{D}(\cdot,\cdot)$, the adjacency matrix $\mathbf{A}$
1: Calculate $\mathbf{D}_{i,j} = \mathcal{D}(\mathbf{F}_{i,:}, \mathbf{F}_{j,:}), \forall i,j$
2: Calculate $\tilde{\mathbf{S}}$ using Eq. (12)
3: Calculate the eigenvalues $\mathbf{\Lambda}$ and eigenvectors $\mathbf{X}$ of $\tilde{\mathbf{S}}$
4: Calculate $\mathbf{H}^{(0)}$ using Eq. (14)
5: Input $\mathbf{H}^{(0)}$ into neural networks, e.g., GNN message-passings in Eq. (4) or general neural network in Eq. (1)

Theorem 1. *If $\mathcal{D}(\cdot,\cdot)$ is the Euclidean distance, the point representation obtained in Eq.* (14) *satisfies Eq.* (8)*, i.e., there exists a constant c' so that*

$$\mathcal{D}(\mathbf{H}^{(0)}_{i,:}, \mathbf{H}^{(0)}_{j,:}) = c'\mathcal{D}(\mathbf{F}_{i,:}, \mathbf{F}_{j,:}), \forall i,j. \tag{15}$$

Theorem 2. *The representation obtained in Eq.* (14) *satisfies Eq.* (7)*, i.e., given the coordinate matrix $\mathbf{F}$ and any $\mathcal{T}(\cdot)$ in Definition 2, the point representation $\mathbf{H}^{(0)}$ is invariant.*

The proofs are provided in Section A in the appendix.

We show our overall framework in Algorithm 1. We name our proposed method TinvNet to highlight that it is **T**ransformation **inv**ariant. Since our method is a general plug-in with alterable neural network components, it is extremely simple to combine with the existing GNN models (see line 6 of Algorithm 1). In fact, we can use any neural network in Eq. (1) as the backbone of TinvNet, including non-graph-based neural networks for geometric data.

4.3 Discussions

4.3.1 Uniqueness of eigenvectors

One caveat to notice is that the eigenvectors can have arbitrary signs, i.e., u and $-u$ are eigenvectors with the same eigenvalue. To tackle this ambiguity, canonical approaches can be adopted to determine the sign [54], e.g., by letting the sum of all values be positive. However, there are potential failure cases (e.g., the sum of values is zero) and issues regarding directions for different points. Therefore, we take another approach to enumerate all 2^d possible eigenvectors as a new data augmentation method. Notice that d is typically small for real-world cases, e.g.,less than 3. Therefore, the enumeration will result in slightly but not too heavy of computational burdens. We provide more justification and empirical evidence for such an approach in Section 5.3.3.

Another potential issue is eigenvalue multiplicity, i.e., multiple eigenvectors have the same eigenvalue. In that case, obtaining unique eigenvectors is more challenging. Luckily, we do not find eigenvalue multiplicity for our tested real-world datasets, and leave handling the issue as future works.

4.3.2 Time complexity

The extra computational cost of TinvNet compared to base models mainly comes from the eigen-decomposition of $\tilde{\mathbf{S}}$. Since it is easy to see that the rank of $\tilde{\mathbf{S}}$ is the same as raw feature $\mathbf{F}$, $\tilde{\mathbf{S}}$ has at most d non-zero eigenvalues, where d denotes the dimensionality of $\mathbf{F}$. Therefore, we only need to calculate the top-d eigen-decomposition of $\tilde{\mathbf{S}}$, which has a time complexity $O(N^2 d)$, where N is the number of points. Experimental results to empirically support the time complexity analysis are provided in Section 5.3.1.

4.3.3 Extension

One may also wonder whether TinvNet can be generalized to non-Euclidean problems, e.g., the distance metric $\mathcal{D}(\cdot,\cdot)$ is not Euclidean. In those cases, TinvNet can be directly adopted while guaranteeing transformation invariance, but the distance-preserving guarantee may not hold. To preserve non-Euclidean distances, we may need generalized MDS [55], [56], non-linear dimensionality reduction methods [57], [58], or other more advanced methods. We leave such explorations as future works.

5 Experiments

In this section, we conduct experiments to verify our proposed method. Specifically, we aim to answer the following questions:

- **Q1**: Can TinvNet guarantee invariance with respect to various kinds of transformations in Definition 2, such as translation, rotation, and scaling?
- **Q2**: Can TinvNet easily combine with different neural network architectures for geometric data?
- **Q3**: How does TinvNet perform compared with other invariant and non-invariant models?

Notice that we do not aim to create new records in the leaderboard. Instead, we aim to provide a fresh perspective for the geometric deep learning problem and empirically verify its usefulness.

5.1 Point Cloud Analysis

The point cloud is one important type of geometric data. Since the shape of objects is invariant to transformations such as rotations, transformation-invariant models are vital to point cloud analysis. We adopt two tasks: object classification and object part segmentation.

5.1.1 Object Classification

For object classification, we adopt ModelNet40 [61], a point cloud dataset containing 40 categories of CAD objects such as airplanes, cars, tables, etc. Each object is represented by 1,024 points with the 3-D coordinate of points. The task is to predict the categories of point clouds, i.e., a 40-class graph classification problem, since each point cloud can be considered a graph. We use a pre-processed dataset suggested in [60], containing 9,843 objects for training and 2,468 objects for testing. Following [15], [45], we adopt the following settings with different rotations. Notice that we only adopt rotation transformations here following the literature, while more transformations are adopted in combinatorial optimization problems in Section 5.2.1.

- z/z: both training and testing data are augmented with rotations about the z-axis (the gravity axis).
- SO3/SO3: both training and testing data are augmented with arbitrary SO3 (3-D rotations).
- z/SO3: training data is augmented with z-axis rotations, and testing data is augmented with SO3.

TABLE 2: The results of point cloud analysis on the test set. The object classification results are accuracy (%) on the ModelNet40 dataset. The object part segmentation results are the mean per-class IoU (%) on the ShapeNet dataset. Larger values indicate better results for both tasks. Best results are in bold and "—" means the result is not reported in the paper.

Task		Object Classification			Object Part Segmentation		
Setting		z/z	SO3/SO3	z/SO3	z/z	SO3/SO3	z/SO3
Invariant baselines	RIConv [45]	86.5	86.4	86.4	—	75.5	75.3
	ClusterNet [15]	87.1	87.1	87.1	—	—	—
	PR-invNet [59]	89.2	89.2	89.2	79.4	79.4	79.4
	RI-GCN [46]	89.5	**89.5**	**89.5**	—	77.3	77.2
Base models	PointNet [60]	87.0	63.6	13.4	81.0	71.4	29.0
	DGCNN [9]	**92.2**	73.3	22.3	**82.0**	75.9	29.6
Our method	TinvNet(PointNet)	86.5	86.5	86.2	80.9	80.0	80.0
	TinvNet(DGCNN)	89.5	**89.5**	**89.5**	**82.0**	**82.1**	**82.0**

TABLE 3: Showcases of point cloud classification for rotated inputs from ModelNet40. All methods adopt the z/SO3 setting.

Method				
Ground Truth	**bookshelf**	**airplane**	**chair**	**night-stand**
PointNet [60]	car	desk	stairs	dresser
DGCNN [9]	sofa	**airplane**	stairs	glass-box
TinvNet(PointNet)	**bookshelf**	**airplane**	**chair**	**night-stand**
TinvNet(DGCNN)	**bookshelf**	**airplane**	**chair**	**night-stand**

The first setting is standard but less challenging since the data is aligned to known axes. The other two cases focus more on models' ability to handle rotations. The z/SO3 setting is most challenging since it requires models to generalize well to different rotations without seeing these rotations in the training data. All results are measured by accuracy, i.e., how many percentages of point clouds are correctly classified.

For baselines, we adopt both non-rotation-invariant models and rotation-invariant models. For the former, we adopt PointNet [60] and DGCNN [9], which are widely adopted neural networks for geometric data. For rotation-invariant models, we adopt RIConv [45], ClusterNet [15], PR-invNet [59], and RI-GCN [46], which are recently proposed rotation-invariant models. Notice that permutation-equivariant models cannot be simply applied here and thus are not compared, as in all previous works.

For our proposed method, we adopt PointNet and DGCNN, which are representative and effective neural networks but cannot maintain transformation invariance, as the backbone, i.e., using PointNet or DGCNN as line 5 of Algorithm 1, denoted as TinvNet(PointNet) and TinvNet(DGCNN), respectively. Notice that except for the initial representations, we keep other settings such as architectures and hyper-parameters unchanged (please refer to Appendix B.1 for the exact settings). Thus, all changes in the performance can be attributed to the differences in initial representations. Besides, we would also emphasize that all the baselines are specially designed for point clouds, while our proposed TinvNet is general and does not utilize any special property of point clouds.

The results are shown in Table 2 and showcases are provided in Table 3. We make the following observations.

Base models such as PointNet and DGCNN show promising results in the standard z/z setting, demonstrating their effectiveness in extracting useful information from point clouds. However, when generalizing to rotations, the performance drops significantly. In the most challenging z/SO3 setting, both PointNet and DGCNN perform miserably. Such results are consistent with the literature since PointNet and DGCNN do not consider rotations.

Both TinvNet(PointNet) and TinvNet(DGCNN) perform equally well on all three settings, indicating TinvNet can effectively handle rotation transformations of point clouds. The results provide empirical evidence that TinvNet is strictly transformation-invariant.

The results in the SO3/SO3 setting show that adopting data augmentation in the training phase alleviates fairly the problem caused by rotations. For example, the performance of PointNet and DGCNN indeed improves when the training data is augmented with rotations. However, the performance gap between z/z and z/SO3 is still considerable. Since the number of possible 3D rotations is infinite, it is infeasible to enumerate all rotated objects.

In the standard z/z setting, the results of TinvNet closely match the corresponding backbone, i.e., TinvNet(PointNet) as of PointNet and TinvNet(DGCNN) as of DGCNN. The results demonstrate that TinvNet does not lose useful information compared to using the coordinates, verifying our findings of distance-preserving. Notice that we use the identical hyper-parameters as in the original backbones.

Our proposed method outperforms RIConv, ClusterNet and PR-invNet, and closely matches RI-GCN. Notice that these methods are specifically designed to be rotation-

invariant, while TinvNet is a simple and general method that does not depend on specific neural architectures. In other words, TinvNet could be extended more easily to new advancements for geometric data, such as novel neural network architectures.

5.1.2 Object Part Segmentation

For object part segmentation, we adopt ShapeNet [62] that contains 16,880 CAD objects of 16 categories. Each object is represented by 2,048 points and has an annotation of 2 to 6 parts, adding up to 50 different parts in total. The task is to predict which part each point belongs to, i.e., a 50-class node classification problem since each point corresponds to one node. We follow the standard dataset splits with 14,006 objects for training and 2,874 objects for testing. Other settings such as transformations and baselines are the same as object classification in Section 5.1.1.

We report the results in Table 2 and provide some showcases in Figure 1. Our proposed method TinvNet manages to beat all comparing methods for object part segmentation in the SO3/SO3 and z/SO3 setting. The results reconfirm that TinvNet is highly capable of handling rotation transformations of point clouds. Though other invariant baselines are not affected by rotations, they fail to be as expressive as our proposed method. Besides, owing to the simplicity and general applicability of TinvNet, we expect the performance to improve further when adopting more powerful neural networks.

In summary, the results of object classification and object part segmentation clearly demonstrate the effectiveness of TinvNet in handling rotation transformations of geometric data, outperforming or matching baselines.

5.2 Combinatorial Optimization

Geometric data combined with simple objectives and constraints form challenging combinatorial optimization problems. Using neural networks to solve combinarotial optimization has a long history [63] with many important applications [10]. Therefore, we test the effectiveness of TinvNet in two such tasks: the travelling salesman problem and the capacitated vehicle routing problem.

5.2.1 Travelling Salesman Problem

Travelling salesman problem (TSP) is a well-known NP-hard problem [64] with many practical applications such as logistics and scheduling [65]. Given a set of points and the distances between points, TSP aims to find the shortest possible route that visits each point exactly once and returns to the origin point. Due to the difficulty in finding optimal solutions, learning-based methods have been adopted to facilitate TSP solvers [10].

We follow [66] for the experimental setting. We consider three cases: TSP-20, TSP-50, and TSP-100, containing $N = 20$, $N = 50$, and $N = 100$ points per data instance, respectively. For each case, we randomly generate 100,000 instances for training and 10,000 instances for testing. All the points have a random 2-D coordinate in the range $[0, 1]$. The distance between points is the Euclidean distance. We adopt the following settings with different transformations of the coordinates:

- **None**: The coordinate of the points is kept the same as the raw coordinates.
- **Translation**: We add a random constant to the coordinate of the points, i.e., $\mathcal{T}(\mathbf{F}) = \mathbf{F} + c$, where the constant c is drawn uniformly from $[-100, 100]$.
- **Rotation**: The coordinate of the points is randomly rotated with respect to the centroid, i.e., $\mathcal{T}(\mathbf{F})_{i,:} = \bar{\mathbf{F}} + \left(\mathbf{F}_{i,:} - \bar{\mathbf{F}}\right)\mathbf{R}$, where $\bar{\mathbf{F}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{F}_{i,:}$ is the centroid and $\mathbf{R} \in \mathrm{SO2}$ is a random 2-D rotation matrix.
- **Reflection**: The coordinate is reflected with respect to the x-axis, i.e., $\mathcal{T}(\mathbf{F})_{i,j} = \mathbf{F}_{i,j}(-1)^j$.
- **Scaling**: The coordinate of the points is uniformly scaled, i.e., $\mathcal{T}(\mathbf{F}) = c\mathbf{F}$, where c is a random number drawn uniformly from $(0, 100]$.

It is easy to that the optimal routes of TSP should be invariant to the above transformations.

For the baseline and our proposed method, we adopt GAT [32] as the GNN backbone, which has been shown to outperform various other GNNs [66]. The detailed settings, e.g., architectures, hyper-parameters, and decoder structures, are kept the same as in the original paper (please refer to Appendix B.1 for details). Besides, we adopt LKH3 [67], a specialized solver for TSP, as a reference line. Notice that, similar to Section 5.1, we do not alter our model to consider the special characteristics of TSP. Instead, we want to demonstrate the effectiveness of TinvNet in handling various transformations.

We show the results in Table 4. TinvNet achieves identical results across different transformations, including translation, rotation, reflection, and scaling, verifying Theorem 2 that our proposed method is strictly transformation invariant to all the transformations in Definition 2. Notice that though we study these transformations independently, the compositionality of transformation invariance guarantees that TinvNet can also handle an arbitrary combination of these transformations.

Though GAT achieves similar results as TinvNet in the **None** setting, it performs poorly when transformations are applied. The results show that GAT is most sensitive to translation and scaling since the range of input features differs significantly compared to the coordinates seen during training. The results are consistent with the literature [47]. In contrast, our TinvNet model does not suffer this issue and greatly outperforms GAT.

There are still gaps between learning-based methods and the specialized solver LHK3, and the gap grows larger as the problem size grows. These results indicate that novel neural networks and training strategies are still needed to advance further the study of learning to solve combinatorial optimizations. Note that though the existing dedicated solvers may lead to better results for certain problems at the current stage, we believe it is important to continue the study of machine learning based combinatorial optimization methods with potential benefits including scalability to extremely large-scale instances, generalization to new problems, inspiring new solvers, etc [10], [63]. Since TinvNet is simple and general, we expect TinvNet to be easily extended to these yet-to-come methods. Future studies may only concern the **None** case to design more powerful architectures and let TinvNet handle the transformation invariance problem.

TABLE 4: The results of the travelling salesman problem (TSP) measured by the length of the output route (lower is better). The numbers in parentheses indicate the performance gap with respect to the specialized solver LHK3.

Size	Method	None	(gap)	Translation	(gap)	Rotation	(gap)	Reflection	(gap)	Scaling[2]	(gap)
TSP-20	LKH3	3.83±0.01		3.83±0.01		3.83±0.01		3.83±0.01		3.83±0.01	
	GAT	3.88±0.01	(1.3%)	10.45±0.02	(172.8%)	3.89±0.01	(1.6%)	4.60±0.02	(20.1%)	10.16±0.02	(165.4%)
	TinvNet	3.87±0.01	(1.1%)	3.87±0.01	(1.1%)	3.87±0.01	(1.1%)	3.87±0.01	(1.1%)	3.87±0.01	(1.1%)
TSP-50	LKH3	5.69±0.01		5.69±0.01		5.69±0.01		5.69±0.01		5.69±0.01	
	GAT	5.96±0.01	(4.7%)	26.07±0.04	(358.0%)	6.02±0.01	(5.8%)	7.93±0.04	(39.3%)	24.96±0.03	(338.6%)
	TinvNet	5.98±0.01	(5.1%)	5.98±0.01	(5.1%)	5.98±0.01	(5.1%)	5.98±0.01	(5.12%)	5.98±0.01	(5.1%)
TSP-100	LKH3	7.76±0.01		7.76±0.01		7.76±0.01		7.76±0.01		7.76±0.01	
	GAT	8.49±0.01	(9.4%)	52.15±0.06	(571.7%)	8.66±0.01	(11.5%)	21.60±0.09	(178.2%)	52.83±0.04	(580.7%)
	TinvNet	8.52±0.01	(9.8%)	8.52±0.01	(9.8%)	8.52±0.01	(9.8%)	8.52±0.01	(9.8%)	8.52±0.01	(9.8%)

TABLE 5: The results of the capacitated vehicle routing problem (CVRP) measured by the length of the output route (lower is better). The numbers in parentheses indicate the performance gap with respect to the specialized solver LHK3.

Size	Method	None	(gap)	Translation	(gap)	Rotation	(gap)	Reflection	(gap)	Scaling[2]	(gap)
CVRP-20	LKH3	6.13±0.02		6.13±0.02		6.13±0.02		6.13±0.02		6.13±0.02	
	GAT	6.55±0.02	(6.8%)	20.86±0.08	(240.5%)	6.57±0.02	(7.2%)	7.84±0.03	(28.0%)	20.23±0.06	(230.3%)
	TinvNet	6.56±0.02	(7.1%)	6.56±0.02	(7.1%)	6.56±0.02	(7.1%)	6.56±0.02	(7.1%)	6.56±0.02	(7.1%)
CVRP-50	LKH3	10.36±0.02		10.36±0.02		10.36±0.02		10.36±0.02		10.36±0.02	
	GAT	11.31±0.03	(9.1%)	52.09±0.18	(402.8%)	11.37±0.03	(9.8%)	15.56±0.07	(50.2%)	48.14±0.12	(364.7%)
	TinvNet	11.38±0.03	(9.8%)	11.38±0.03	(9.8%)	11.38±0.03	(9.8%)	11.38±0.03	(9.8%)	11.38±0.03	(9.8%)
CVRP-100	LKH3	15.61±0.04		15.61±0.04		15.61±0.04		15.61±0.04		15.61±0.04	
	GAT	17.73±0.04	(13.6%)	104.23±0.35	(567.8%)	17.87±0.04	(14.5%)	62.95±0.50	(303.3%)	100.95±0.24	(546.6%)
	TinvNet	17.31±0.04	(10.9%)	17.31±0.04	(10.9%)	17.31±0.04	(10.9%)	17.31±0.04	(10.9%)	17.31±0.04	(10.9%)

We also plot the output route of GAT and TinvNet for one example instance of TSP-20 in Figure 2. The figure clearly demonstrates the importance of maintaining transformation invariance and the efficacy of TinvNet.

5.2.2 Capacitated Vehicle Routing Problem

Capacitated vehicle routing problem (CVRP) [68] is a generalization to TSP with more practical usages. Given a set of points and the distances between points, instead of finding one shortest route as in TSP, we need to construct multiple routes starting and ending from a central depot. The goal is to minimize the length of all the routes while meeting the demand of each point. Besides, the total demand of points in each route is capacitated, i.e., corresponding to constraints in real delivery problems that our "vehicles" have a limited capacity. CVRP is also known to be an NP-hard problem [68].

Similar to Section 5.2.1, we follow [66] to set up the experimental setting for CVRP. Specifically, we adopt three cases: CVRP-20, CVRP-50, and CVRP-100 containing 20, 50, and 100 points per instance, respectively. The coordinates of points are in the range $[0, 1]$, and the metric is the Euclidean distance. For each case, we generate 100,000 instances for training and 10,000 instances for testing. We adopt the same five transformations as in TSP, i.e., **None**, **Translation**, **Rotation**, **Reflection**, and **Scaling**. The baselines and experimental settings are also the same as in Section 5.2.1. For more details, please refer to Appendix B.1.

We report the results in Table 5 and provide a showcase in Figure 3. The results show similar trends as in Table 4 for TSP. Concretely, GAT performs reasonably well in the **None** setting but fails to generalize to different transformations. For some transformations like translation and scaling, the results of GAT even become intolerable. On the other hand, thanks to the transformation invariance property of TinvNet, it is able to handle different transformations with zero performance drop. The results demonstrate again that TinvNet is an effective and general solution towards transformation invariant combinatorial problems.

5.3 Analysis

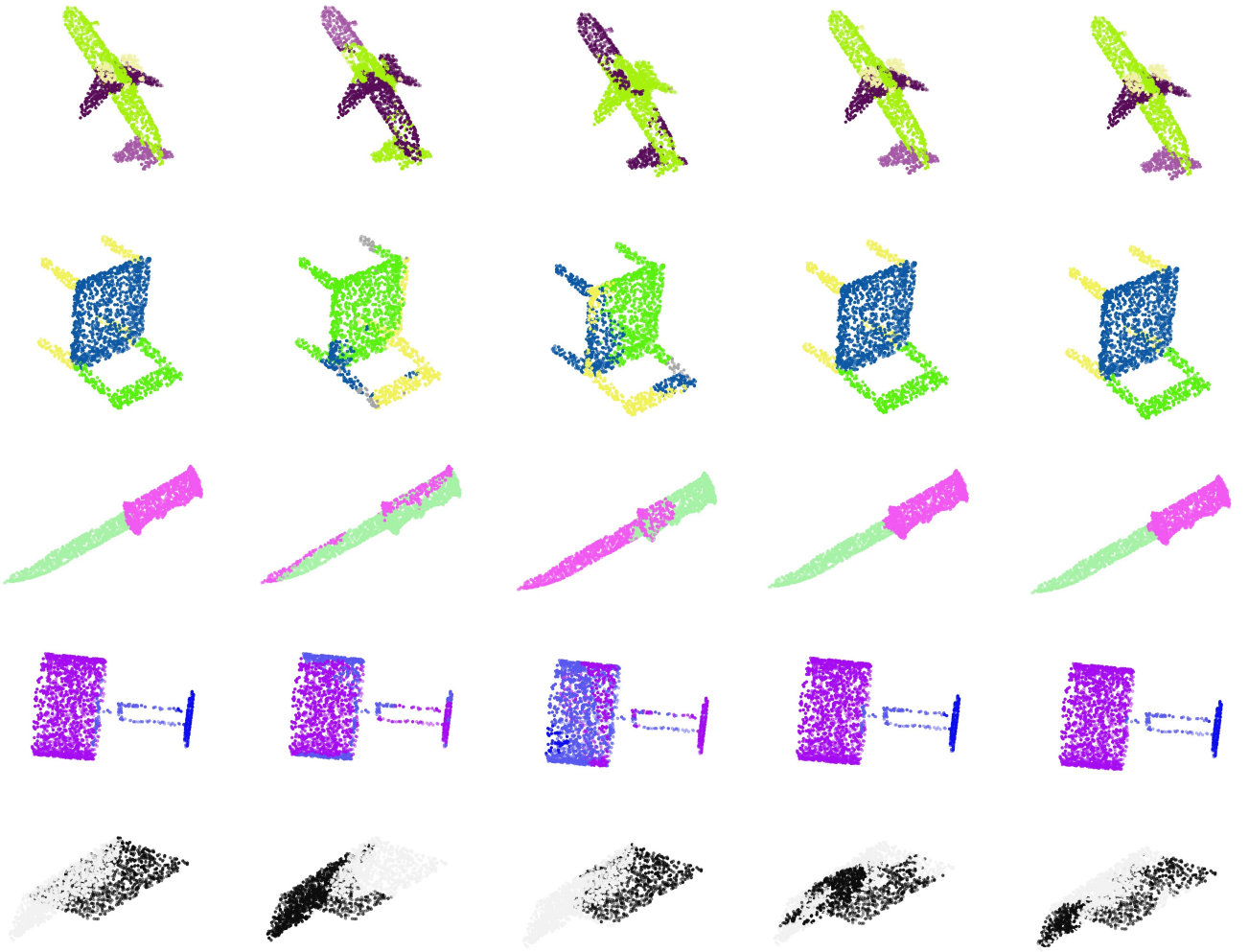
5.3.1 Scalability

To empirically analyze the scalability of our proposed method, we record the running time of calculating the initial point representations while varying the number of points. The average results of 10 runs are reported in Figure 4. We also fit a linear regression curve after applying log transformation on both axes and report the fitting statistics. The results show that the running time grows quadratically with respect to the number of points, which is consistent with our analysis in Section 4.2. Besides, for a data instance with 2,048 points, the running time is less than 1 second. Notice that the initial point representations only need to be calculated once and can be pre-computed, while the optimization of backbone neural networks usually needs dozens or hundreds of epochs. Therefore, TinvNet does not incur high extra computational costs.

5.3.2 Comparison with kNN and Data Whitening

A straightforward heuristic to obtain transformation-invariant features is k-NN, i.e., calculating the distance of each point with its k nearest neighbors. Besides,

2. Since the length of routes is proportional to scaling, we normalize the results in the scaling setting to be consistent with other settings, i.e., when $\mathcal{T}(\mathbf{F}) = c\mathbf{F}$, the length of output routes is scaled by $\frac{1}{c}$.



(a) Ground truth (b) PointNet (c) DGCNN (d) TinvNet(PointNet) (e) TinvNet(DGCNN)

Fig. 1: Showcases of point cloud part segmentation for rotated inputs from ShapeNet. All methods adopt the z/SO3 setting.

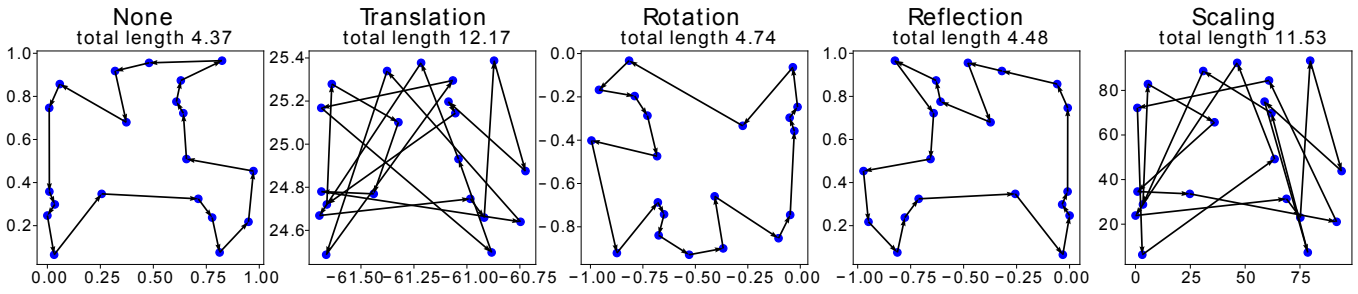
data whitening is a classical data pre-processing method by decorrelating different dimensions of the input data. From its properties, data whitening can also ensure transformation-invariance (but not distance-preserving). Next, we empirically compare our proposed method with k-NN and data whitening. Specifically, we choose two k values for k-NN: $k = 3$, which has the same dimensionality as raw features and our proposed method, and $k = 10$, which contains more flexibility. For data whitening, we adopt the PCA-whitening. We report the results for the point cloud classification task in Table 6, while other tasks indicate similar results. The results show that, though k-NN and data whitening can maintain rotation-invariance, their performance is much lower than our proposed method.

TABLE 6: The results of comparing with kNN and data whitening for point cloud classification on the ModelNet40 dataset. The best results for each backbone are in bold.

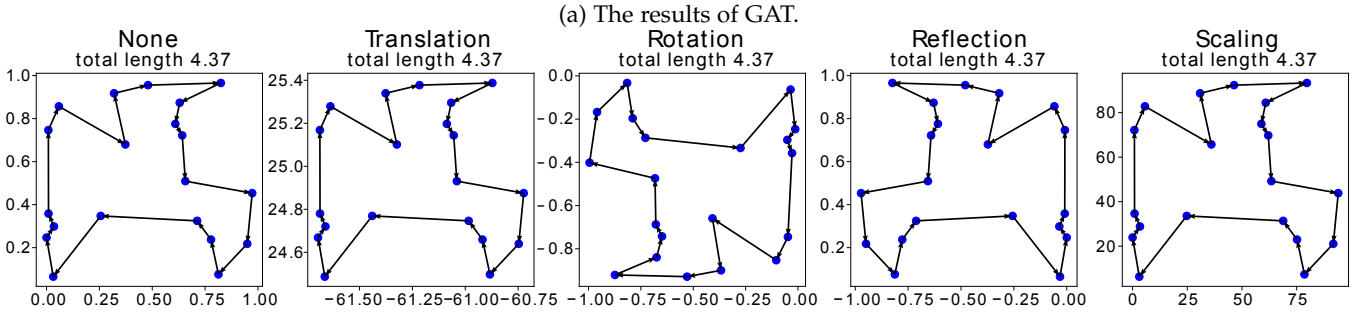
Method	z/z	SO3/SO3	z/SO3
PointNet(k-NN,k=3)	24.9	24.9	24.9
PointNet(k-NN,k=10)	29.9	29.9	29.9
PointNet(Whitening)	81.9	81.9	81.9
TinvNet(PointNet)	**86.5**	**86.5**	**86.2**
DGCNN(k-NN,k=3)	29.9	29.9	29.9
DGCNN(k-NN,k=3)	36.1	36.1	36.1
DGCNN(Whitening)	85.8	85.8	85.8
TinvNet(DGCNN)	**89.5**	**89.5**	**89.5**

5.3.3 Comparison with Canonical Eigenvector Sign

As discussed in Section 4.3, the signs may bring ambiguity to TinvNet. We propose to enumerate all 2^d eigenvectors with different signs as a data augmentation technique to solve this issue. An alternative is to use a canonical approach to determine a unique, e.g., by letting the sum of all values be positive [54]. We compare these two approaches empirically for the point cloud classification task and report the results in Table 7, where fixing the sign is denoted as TinvNet-F. The results indicate that enumerating the 2^d possible signs can consistently improve the performance, with the cost of increasing computations. A plausible explanation is that there are potential direction issues for the canonical approach. For example, consider point clouds all representing airplanes. Though the canonical approach ensures that each airplane has a unique direction, there is no guarantee that different airplanes are aligned to the same

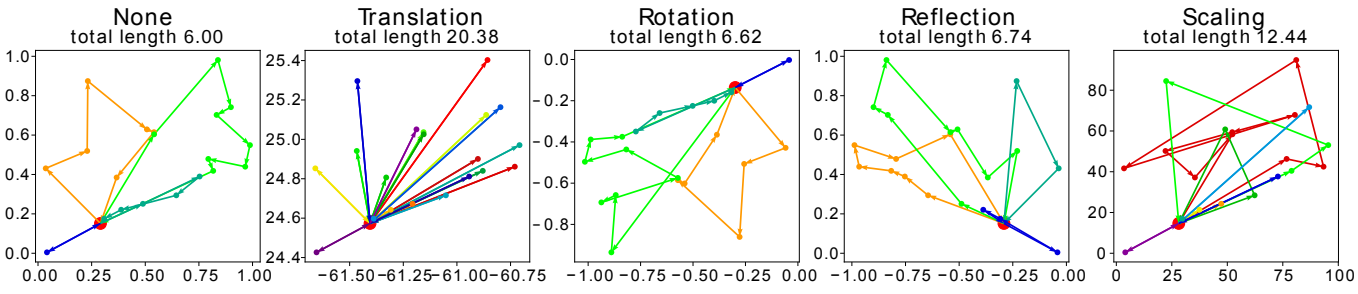


(a) The results of GAT.

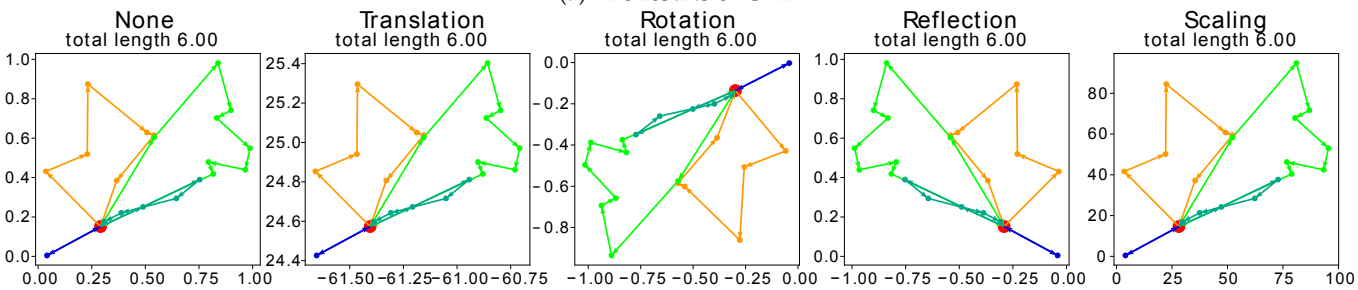


(b) The results of TinvNet.

Fig. 2: The output route of (a) GAT, and (b) TinvNet for one example instance of TSP-20 after different transformations.



(a) The results of GAT.



(b) The results of TinvNet.

Fig. 3: The output route of (a) GAT, and (b) TinvNet for one example instance of CVRP-20 after different transformations.

TABLE 7: The results of enumerating signs of eigenvectors as data augmentation and using a unique sign by the canonical approach for point cloud classification task.

Method	z/z	SO3/SO3	z/SO3
TinvNet(PointNet)-F	85.4	85.2	85.4
TinvNet(PointNet)	86.5	86.5	86.2
TinvNet(DGCNN)-F	88.3	89.2	88.5
TinvNet(DGCNN)	89.5	89.5	89.5

direction, e.g., some airplanes may point towards the left, others may point towards the right, upside down, etc. The direction issue may harm the model learning. Besides, the canonical approaches can have failure cases (e.g., the sum of all values is zero), though we do not observe such failure cases in our experiments.

5.3.4 *Synthetic Dataset with Eigenvalue Multiplicity*

As discussed in Section 4.3.1, eigenvalue multiplicity may bring challenges for TinvNet. To gain more empirical insights, we conduct experiments on synthetic datasets by generating and classifying point cloud objects with symmetry, which naturally results in eigenvalue multiplicity.

TABLE 8: The results of object classification on synthetic dataset.

Noise	0	0.25	0.50	0.75	1.00
TinvNet(PointNet)	93.33±2.14	94.56±1.08	94.33±1.13	91.89±1.47	89.67±1.43

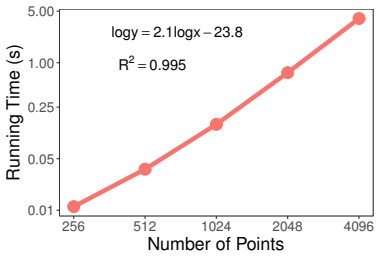


Fig. 4: The running time of calculating the initial point representations with respect to the number of points.

Specifically, we randomly generate three types of objects: cylinder, regular quadrangular prism, and regular hexagonal prism. For each category, we generate 200 objects, where 70% is used for training and the rest for testing, and each object is represented by 512 3D points. We also randomly add Gaussian noises into the input coordinate matrix. Specifically, we adopt the poisoning attack setting, i.e., the random noises are added into both the training and testing stage. We report the average results in Table 8 with 5 random seeds. The results show that our proposed method works reasonably well and stable on the synthetic dataset, even with random noises, indicating that eigenvalue multiplicity does not greatly affect our model empirically.

6 Conclusion

In this paper, we first revisit transformation invariance of geometric data using neural networks and find that transformation invariant and distance-preserving initial representations are sufficient to solve the problem. Motivated by these findings, we propose TinvNet, a straightforward and general transformation invariant neural network plug-in for geometric data. We prove that TinvNet can strictly guarantee transformation invariance and is general and compatible with various architectures. Experimental results on point cloud analysis and combinatorial optimization demonstrate the effectiveness and general applicability of TinvNet.

One limitation of TinvNet is that it can only handle similarity transformation, and we plan to study other transformations (e.g., affine transformations) in the future. It would also be interesting to test TinvNet for more applications.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China No. 2020AAA0106300, National Natural Science Foundation of China (No. 62250008, 62222209, 62102222, 62206149), China National Postdoctoral Program for Innovative Talents No. BX20220185 and China Postdoctoral Science Foundation No. 2022M711813. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, 2015.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, 2012.
[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.
[4] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Human-level control through deep reinforcement learning," *nature*, 2015.
[5] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, 2017.
[6] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *International conference on machine learning*, 2016.
[7] S. Ravanbakhsh, J. Schneider, and B. Poczos, "Equivariance through parameter-sharing," in *International Conference on Machine Learning*, 2017.
[8] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, 2017.
[9] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *Acm Transactions On Graphics (tog)*, 2019.
[10] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, 2020.
[11] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, 2017.
[12] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and equivariant graph networks," in *International Conference on Learning Representations*, 2019.
[13] N. Keriven and G. Peyré, "Universal invariant and equivariant graph neural networks," in *Advances in Neural Information Processing Systems*, 2019.
[14] J. Klicpera, J. Groß, and S. Günnemann, "Directional message passing for molecular graphs," in *International Conference on Learning Representations*, 2019.
[15] C. Chen, G. Li, R. Xu, T. Chen, M. Wang, and L. Lin, "Clusternet: Deep hierarchical cluster network with rigorously rotation-invariant representation for point cloud analysis," in *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2019.
[16] N. Thomas, T. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley, "Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds," *arXiv:1802.08219*, 2018.
[17] F. Fuchs, D. Worrall, V. Fischer, and M. Welling, "Se (3)-transformers: 3d roto-translation equivariant attention networks," *Advances in Neural Information Processing Systems*, 2020.
[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017.
[19] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2005.
[20] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, 2008.
[21] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, 2009.
[22] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations*, 2014.
[23] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng, "Graph wavelet neural network," in *International Conference on Learning Representations*, 2018.

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

[25] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016.

[26] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, 2015.

[27] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *International Conference on Learning Representations*, 2016.

[28] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE signal processing magazine*, 2013.

[29] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, 2018.

[30] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[31] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017.

[32] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[33] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International Conference on Machine Learning*, 2018.

[34] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019.

[35] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv:1806.01261*, 2018.

[36] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[37] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[38] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," in *Advances in Neural Information Processing Systems*, 2019.

[39] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, 2011.

[40] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[41] R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Relational pooling for graph representations," in *International Conference on Machine Learning*, 2019.

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

[43] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, "Spherical cnns," in *International Conference on Learning Representations*, 2018.

[44] M. Weiler, M. Geiger, M. Welling, W. Boomsma, and T. S. Cohen, "3d steerable cnns: Learning rotationally equivariant features in volumetric data," in *Advances in Neural Information Processing Systems*, 2018.

[45] Z. Zhang, B.-S. Hua, D. W. Rosen, and S.-K. Yeung, "Rotation invariant convolutions for 3d point clouds deep learning," in *International Conference on 3D Vision (3DV)*, 2019.

[46] S. KIM, J. Park, and B. Han, "Rotation-invariant local-to-global representation learning for 3d point cloud," *Advances in Neural Information Processing Systems*, 2020.

[47] H. Tang, Z. Huang, J. Gu, B.-L. Lu, and H. Su, "Towards scale-invariant graph-related problem solving by iterative homogeneous graph neural networks," in *Advances in Neural Information Processing Systems*, 2020.

[48] J. Han, Y. Rong, T. Xu, and W. Huang, "Geometrically equivariant graph neural networks: A survey," *arXiv preprint arXiv:2202.07230*, 2022.

[49] M. Horie, N. Morita, Y. Ihara, and N. Mitsume, "Isometric transformation invariant and equivariant graph convolutional networks," in *International Conference on Learning Representations*, 2021.

[50] V. G. Satorras, E. Hoogeboom, and M. Welling, "E (n) equivariant graph neural networks," in *Proceedings of the 38th International Conference on Machine Learning, ICML*, 2021.

[51] P. Hermosilla, M. Schäfer, M. Lang, G. Fackelmann, P.-P. Vázquez, B. Kozlikova, M. Krone, T. Ritschel, and T. Ropinski, "Intrinsic-extrinsic convolution and pooling for learning on 3d protein structures," in *International Conference on Learning Representations*, 2020.

[52] L. Koestler, D. Grittner, M. Moeller, D. Cremers, and Z. Lähner, "Intrinsic neural fields: Learning functions on manifolds," in *European Conference on Computer Vision*, 2022, pp. 622–639.

[53] M. A. Cox and T. F. Cox, "Multidimensional scaling," in *Handbook of data visualization*, 2008.

[54] D. A. Spielman, "Testing isomorphism of graphs with distinct eigenvalues," http://www.cs.yale.edu/homes/spielman/561/lect08-18.pdf, 2018, [Online; accessed 6-July-2021].

[55] S. Agarwal, J. Wills, L. Cayton, G. Lanckriet, D. Kriegman, and S. Belongie, "Generalized non-metric multidimensional scaling," in *Artificial Intelligence and Statistics*, 2007.

[56] A. M. Bronstein, M. M. Bronstein, and R. Kimmel, "Generalized multidimensional scaling: a framework for isometry-invariant partial surface matching," *Proceedings of the National Academy of Sciences*, 2006.

[57] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, 2000.

[58] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science*, 2000.

[59] R. Yu, X. Wei, F. Tombari, and J. Sun, "Deep positional and relational feature learning for rotation-invariant point cloud analysis," in *European Conference on Computer Vision*, 2020.

[60] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[61] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[62] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.

[63] K. A. Smith, "Neural networks for combinatorial optimization: a review of more than a decade of research," *Informs journal on Computing*, vol. 11, no. 1, pp. 15–34, 1999.

[64] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," *Theoretical computer science*, 1977.

[65] J. K. Lenstra and A. R. Kan, "Some simple applications of the travelling salesman problem," *Journal of the Operational Research Society*, 1975.

[66] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations*, 2018.

[67] K. Helsgaun, "An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems," *Roskilde: Roskilde University*, vol. 12, 2017.

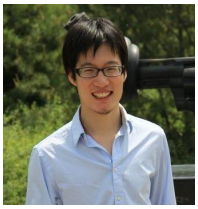
[68] P. Toth and D. Vigo, *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[69] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 13, no. 04, pp. 376–380, 1991.

Ziwei Zhang received his Ph.D. from the Department of Computer Science and Technology, Tsinghua University, in 2021. He is currently an associate professor in the School of Computer Science and Engineering at Beihang University. His research interests focus on machine learning on graphs, including graph neural network (GNN), network embedding, and automated graph machine learning. He has published 50 papers in prestigious conferences and journals, including ACM SIGKDD, ICML, NeurIPS, AAAI, IJCAI, IEEE TPAMI, and IEEE TKDE.

Xin Wang is currently an Assistant Professor at the Department of Computer Science and Technology, Tsinghua University. He got both of his Ph.D. and B.E degrees in Computer Science and Technology from Zhejiang University, China. He also holds a Ph.D. degree in Computing Science from Simon Fraser University, Canada. His research interests include relational media big data analysis, multimedia intelligence and recommendation in 64878853 media. He has published several high-quality research papers in top conferences including ICML, KDD, WWW, ACM Multimedia, etc. He is the recipient of 2017 China Postdoctoral innovative talents supporting program. He receives the ACM China Rising Star Award in 2020.

Zeyang Zhang received the B.E. from the Department of Computer Science and Technology, Tsinghua University in 2020. He is currently a Ph.D. candidate in the Department of Computer Science and Technology of Tsinghua University. His main research interests focus on graph representation learning and automated machine learning. He has published several papers in prestigious conferences, e.g., AAAI, NeurIPS, etc.



Peng Cui is an Associate Professor with tenure in Tsinghua University. He got his PhD degree from Tsinghua University in 2010. His research interests include causally-regularized machine learning, network representation learning, and social dynamics modeling. He has published more than 100 papers in prestigious conferences and journals in data mining and multimedia. His recent research won the IEEE Multimedia Best Department Paper Award, SIGKDD 2016 Best Paper Finalist, ICDM 2015 Best Student Paper Award, SIGKDD 2014 Best Paper Finalist, IEEE ICME 2014 Best Paper Award, ACM MM12 Grand Challenge Multimodal Award, and MMM13 Best Paper Award. He is PC co-chair of CIKM2019 and MMM2020, SPC or area chair of WWW, ACM Multimedia, IJCAI, AAAI, etc., and Associate Editors of IEEE TKDE, IEEE TBD, ACM TIST, and ACM TOMM etc. He received ACM China Rising Star Award in 2015, and CCF-IEEE CS Young Scientist Award in 2018. He is now a Distinguished Member of ACM and CCF, and a Senior Member of IEEE.

Wenwu Zhu is currently a Professor of the Computer Science Department of Tsinghua University. Prior to his current post, he was a Senior Researcher and Research Manager at Microsoft Research Asia. He was the Chief Scientist and Director at Intel Research China from 2004 to 2008. He worked at Bell Labs New Jersey as a Member of Technical Staff during 1996-1999. He received his Ph.D. degree from New York University in 1996.

He served as the Editor-in-Chief for the IEEE Transactions on Multimedia (T-MM) from January 1, 2017, to December 31, 2019. He has been serving as Vice EiC for IEEE Transactions on Circuits and Systems for Video Technology (TCSVT) and the chair of the steering committee for IEEE T-MM since January 1, 2020. His current research interests are in the areas of multimedia computing and networking, and big data. He has published over 350 papers in the referred journals and received nine Best Paper Awards including IEEE TCSVT in 2001 and 2019, and ACM Multimedia 2012. He is an IEEE Fellow, AAAS Fellow, SPIE Fellow and a member of the European Academy of Sciences (Academia Europaea).