Interpretable Rotation-Equivariant Quaternion Neural Networks for 3D Point Cloud Processing

Wen Shen, Zhihua Wei, Qihan Ren, Binbin Zhang, Shikun Huang, Jiaqi Fan, and Quanshi Zhang, *Member, IEEE*

***Abstract*—This study proposes a set of generic rules to revise existing neural networks for 3D point cloud processing to rotation-equivariant quaternion neural networks (REQNNs), in order to make feature representations of neural networks to be rotation-equivariant and permutation-invariant. Rotation equivariance of features means that the feature computed on a rotated input point cloud is the same as applying the same rotation transformation to the feature computed on the original input point cloud. We find that the rotation-equivariance of features is naturally satisfied, if a neural network uses quaternion features. Interestingly, we prove that such a network revision also makes gradients of features in the REQNN to be rotation-equivariant *w.r.t.* inputs, and the training of the REQNN to be rotation-invariant *w.r.t.* inputs. Besides, permutation-invariance examines whether the intermediate-layer features are invariant, when we reorder input points. We also evaluate the stability of knowledge representations of REQNNs, and the robustness of REQNNs to adversarial rotation attacks. Experiments have shown that REQNNs outperform traditional neural networks in both terms of classification accuracy and robustness on rotated testing samples.**

***Index Terms*—Rotation equivariance, permutation invariance, 3D point cloud processing, quaternion.**

I. Introduction

3D POINT cloud processing has achieved increasing attention in recent years owing to its vital role in applications, such as robotics and autonomous driving. Neural networks for image processing typically extract features from rich color information. In comparison, neural networks for 3D shape classification are supposed to extract features from 3D structures, without using information of orientations or orders of input points. In other words, people expect neural networks for 3D shape classification to satisfy the rotation-invariance and the permutation-invariance properties.

Beyond the rotation invariance and the permutation invariance, in this study, we simultaneously pursue four properties, 1. the rotation-equivariance of features, 2. the rotation-equivariance of feature gradients, 3. the rotation-invariance of the network training, and 4. the permutation-invariance of features.

• *Rotation-equivariance of features* is a property beyond the rotation-invariance, which disentangles rotational information and rotation-agnostic structural information. Given a 3D point cloud, the rotation-equivariance property refers to the equivariance of the following two ways of computing the feature of an intermediate layer of a neural network [9]. The first way is to first rotate the point cloud by a specific angle and then compute the feature on the rotated sample. The second way is to compute the feature on the point cloud and then apply the same rotation transformation to the feature. If the above two ways generate the same feature, i.e., *encoder(rotater(sample)) = rotater(encoder(sample)),* then we consider that this neural network satisfies the rotation-equivariance of features, as shown in Fig. 1.

Note that both the rotation-invariance of features and the rotation-equivariance of features prevent rotations from affecting the inference. Some neural networks achieve the rotation-invariance of features by simply removing rotational information from features [5], [8]. In comparison, we expect a neural network to pursue the rotation-equivariance of features, which maintains the rotational feature in intermediate layers by disentangling the rotational feature away from the rotation-independent structural feature. The rotation-equivariance of features is of significant value in many applications. For example, we can use the disentangled structural information for inference, and use the rotational information to control the object orientation in object synthesis.

• *Rotation-equivariance of feature gradients*. Similar to the rotation-equivariance of features, we expect that the feature gradients of a neural network are also rotation-equivariant. It means that all dimensions in the feature gradient are always

Manuscript received 18 July 2022; revised 29 November 2023; accepted 7 December 2023. Date of publication 8 January 2024; date of current version 3 April 2024. This work was supported in part by the National Nature Science Foundation of China under Grants 62206170, 62276165, and 62376199, in part by National Key R&D Program of China under Grant 2021ZD0111602, in part by Shanghai Natural Science Foundation under Grants 21JC1403800 and 21ZR1434600, in part by China Postdoctoral Science Foundation under Grant 2023M732224, and in part by Huawei Technologies Inc. Recommended for acceptance by G. Hua. *(Wen Shen and Zhihua Wei contributed equally to this work.) (Corresponding author: Quanshi Zhang.)*

Wen Shen and Qihan Ren are with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: wen_shen@sjtu.edu.cn; renqihan@sjtu.edu.cn).

Zhihua Wei is with the Department of Computer Science and Technology, and the Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai 200070, China (e-mail: zhihua_wei@tongji.edu.cn).

Binbin Zhang, Shikun Huang, and Jiaqi Fan are with the Department of Computer Science and Technology, Tongji University, Shanghai 200070, China (e-mail: 0206zbb@tongji.edu.cn; hsk@tongji.edu.cn; 1930795@tongji.edu.cn).

Quanshi Zhang is with the Department of Computer Science and Engineering, the John Hopcroft Center, and Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: zqs1022@sjtu.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TPAMI.2023.3346383, provided by the authors.

Digital Object Identifier 10.1109/TPAMI.2023.3346383

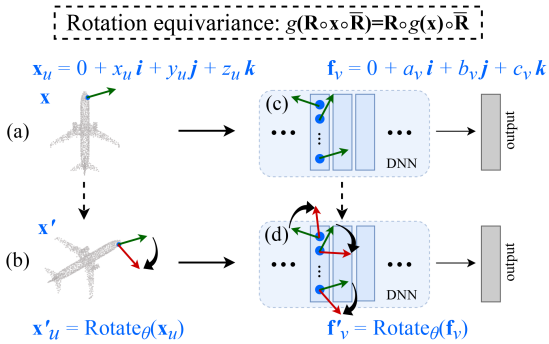

Fig. 1. Illustration of the rotation-equivariance of features. We find that when we use quaternion features in a neural network, such quaternion features are naturally rotation-equivariant under certain conditions. The rotation-equivariance of features is defined as follows. When we rotate the input point cloud $\mathbf{x}$ (a) with a specific angle to obtain the same point cloud $\mathbf{x}'$ (b) with a different orientation, then the intermediate-layer feature (d) generated by the neural network is equivalent to applying the transformation *w.r.t.* the same rotation to the feature (c) of the original point cloud, i.e., $g(\text{Rotate}_\theta(\mathbf{x})) = \text{Rotate}_\theta(g(\mathbf{x}))$, where g denotes the function of signal processing from the input $\mathbf{x}$ to the quaternion feature in the intermediate layer.

aligned with dimensions in the feature, no matter how we rotate the input 3D point cloud. More crucially, rotation-equivariance of feature gradients further leads to rotation-invariance of the network training, which guarantees the stability of training neural networks.

• *Rotation-invariance of the network training* refers to the property that the neural network can learn the same parameters, no matter how we rotate training samples.

• *Permutation-invariance of features* refers to the property that given the point cloud $\mathbf{x}$, the feature $g(\mathbf{x})$ in the intermediate layer of the neural network does not change, no matter how people reorder 3D points in $\mathbf{x}$, i.e., $g(\mathbf{x}^{\text{reorder}}) = g(\mathbf{x})$. Here, g denotes the function of signal processing from the input $\mathbf{x}$ to the quaternion feature in the intermediate layer.

REQNN: In this study, we propose a set of generic rules to revise any arbitrary neural network for 3D point cloud processing, and make the revised neural network satisfy the above four properties. Specifically, we revise the neural network to use quaternion features, instead of using real-valued features. Thus, the revised neural network is termed the rotation-equivariant quaternion neural network (REQNN).

Crucially, such rules can be broadly used to revise most existing neural networks for 3D point cloud processing. The effectiveness and the broad applicability of the proposed rules have been verified on four classic neural networks for 3D shape classification and reconstruction, including PointNet [29], PointNet++ [30], DGCNN [45], and PointConv [49].

Specifically, to satisfy the above four properties, the REQNN uses quaternion features. Both the input 3D point cloud and intermediate-layer features of the REQNN are represented using quaternions. A quaternion $\mathbf{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k} \in \mathbb{H}$ is a hyper-complex number with three imaginary parts ($\boldsymbol{i}, \boldsymbol{j}$, and $\boldsymbol{k}$) [17]. Each 3D point in a poind cloud is represented as a pure quaternion $\mathbf{x} = 0 + x_1\boldsymbol{i} + x_2\boldsymbol{j} + x_3\boldsymbol{k}$. Quaternion features can be vectors/matrices/tensors, each element of which is a pure quaternion. For each quaternion feature, we can rotate this feature around an quaternion axis $\mathbf{o} = 0 + o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k} \in \mathbb{H}$ (o_1, o_2, $o_3 \in \mathbb{R}$) by an angle $\theta \in [0, 2\pi)$. Such a rotation operation is conducted in an element-wise manner. For each quaternion element $\mathbf{f} = 0 + f_1\boldsymbol{i} + f_2\boldsymbol{j} + f_3\boldsymbol{k} \in \mathbb{H}$ of the feature vector/matrix/tensor, the rotated quaternion is given as $\mathbf{R}\mathbf{f}\overline{\mathbf{R}}$, where $\mathbf{R} = e^{\mathbf{o}\frac{\theta}{2}}$ and $\overline{\mathbf{R}} = e^{-\mathbf{o}\frac{\theta}{2}}$ are two quaternions. $\overline{\mathbf{R}}$ is the conjugation of $\mathbf{R}$.

In this way, $\mathbf{f}$ can denote either the feature in an intermediate layer or the gradient *w.r.t.* the feature. Then, the rotation-equivariance of features or gradients is defined as the equivariance of the two features/gradients, $\mathbf{f}_1 = \mathbf{f}_2$, computed in two cases, i.e., (Case 1) rotating the input and generating the quaternion features/gradients $\mathbf{f}_1 = g(\mathbf{R}\mathbf{x}\overline{\mathbf{R}})$, and (Case 2) directly applying the same rotation to the quaternion features/gradients $\mathbf{f}_2 = \mathbf{R}g(\mathbf{x})\overline{\mathbf{R}}$.

Generic Rules for Revising Neural Networks: In order to ensure a neural network to satisfy 1. the rotation-equivariance of features, 2. the rotation-equivariance of feature gradients, and 3. the rotation-invariance of the network training, we revise several layerwise operations in existing neural networks. Specifically, we revise operations of the convolution, ReLU, batch-normalization, max-pooling, and 3D coordinates weighting [49]. We have proven that revisions of these layerwise operations ensure the above three properties of the entire neural network. In addition, we also introduce how to adapt rotation-equivariant features into rotation-invariant features for rotation-independent tasks.

In addition, we discover that when we revise the farthest point sampling [30] and the ball-query-search-based grouping [30] operations, we can change the neural network to be permutation-invariant.

Advantages: 1) REQNNs learn more stable feature representations than traditional neural networks when we rotate input point clouds with different angles. To this end, we measure the stability of input attributions *w.r.t.* different rotations and prove that input attributions encoded by REQNNs are more stable than those encoded by traditional neural networks.

2) REQNNs are more robust to rotation-based attacks than traditional neural networks.

3) REQNNs naturally exhibit high generalization on rotated samples without any rotation augmentations, which enhances the stability of network training.

4) Rules proposed in this study can be used to revise most existing neural networks into REQNNs. In comparison, previous studies developed specific network architectures [59] or designed specific operations [41] to achieve rotation-equivariance of features, which hurt the broad applicability.

Contributions of this study can be summarized as follows:

- In this study, we learn neural networks for 3D point cloud processing with the following four properties, i.e., the rotation-equivariance of features, the rotation-equivariance of feature gradients, the rotation-invariance of the network training, and the permutation-invariance of features.
- We propose a set of generic rules for modifying various neural networks into REQNNs. We prove that these rules ensure the above four properties.

- Experiments on 3D shape classification and reconstruction have demonstrated that REQNNs exhibit higher generalization on rotated samples, encode more stable feature representations, are more robust to rotation-based attacks, and have a more stable training process than traditional neural networks.

A preliminary version of this paper appeared in [35].

II. Related Work

In recent years, we have witnessed the rapid development of deep neural networks (DNNs) for 3D point cloud processing. As a pioneer, PointNet [29] used a point-wise multi-layer perceptron to extract features. However, PointNet ignored the contextual information of 3D point clouds. To this end, many studies proposed different operations to extract contextual information from 3D point clouds to improve the accuracy [19], [24], [25], [30], [36], [58]. Besides, some studies considered the 3D point cloud as a graph and further introduced graph convolutions for 3D point cloud processing [22], [38], [45], or considered the 3D point cloud as a tree and computed features from leaves to the root for inference [21]. In addition, several studies designed special convolution operators for 3D point cloud processing. PointConv [49] used nonlinear functions to estimate optimal convolution weights from the input 3D coordinates. InterpCNN [27] learned a set of convolutional filters, whose weights were interpolated based on features of neighbor points, and applied such convolutional filters to point clouds.

The robustness to rotation is important for DNNs to classify 3D point clouds. Data augmentation is the most direct way to improve the rotation robustness [43], but the computational cost is expensive. Besides, the spatial transformer network (STN) [18] was proposed to perform rotations, scaling, cropping, and etc. on feature maps for data augmentation, so as to improve the robustness of feature maps.

Some studies paid attention to the rotation-invariance property. An intuitive way was to first project 3D points onto a sphere and then extract rotation-invariant features [8], [32], [40], [52], [53], [55]. Some studies directly discarded orientation information of the input, so as to learn rotation-invariant features [5], [12], [56]. Besides, a few studies combined local 3D structures with their contextual information [23] or with global 3D structures [57] to ensure the rotation-invariance property.

However, such rotation-invariant methods typically removed rotational information from features. In comparison, rotation-equivariant methods encoded the rotational information in the feature by disentangling the rotational information away from rotation-independent structural information. To the best of our knowledge, very few studies focused on this topic. Previous studies either developed specific network structures [54], [59] or designed specific operations [13], [31], [41] to achieve the rotation-equivariance property. Wang et al. [44] learned rotation-equivariant orientation information based on a feature map defined on the icosahedral group [6], which is theoretically equivariant to rotations in the icosahedral group. Deng et al. [11] extended each dimension of a feature to a 3D vector, so as to enable the rotation of features. The technique proposed in this paper and its extension study [7] can be considered as a certain REQNN, if we rewrite the feature vector (the vector neuron) as a unit quaternion. More crucially, our study further theoretically proves that our REQNN satisfies the rotation-equivariance of features, the rotation-equivariance of feature gradients, the rotation-invariance of the training, and the permutation-invariance of features. They are all beyond the findings in [7], [11], [31].

Complex and Quaternion Networks: In recent studies, neural networks using complex-valued features or quaternion-valued [17] features have attracted more and more attention [2], [10], [14], [16], [20], [28], [42], [47], [48], [51], [62]. In this study, we use quaternions to represent input 3D point clouds, features, and 3D rotations, so as to make a DNN satisfy properties of the rotation-equivariance of features, the rotation-equivariance of feature gradients, the rotation-invariance of the network training, and the permutation-invariance of features.

III. Approach

A. Quaternion Features and Rotations

Preliminaries: Quaternion Numbers: A quaternion [17] $\mathbf{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k} \in \mathbb{H}$ is a hyper-complex number with a real component (q_0) and three imaginary components ($q_1\boldsymbol{i}, q_2\boldsymbol{j}, q_3\boldsymbol{k}$), where q_0, q_1, q_2 and q_3 are real numbers and $\boldsymbol{i}, \boldsymbol{j}$, and $\boldsymbol{k}$ are imaginary units; $\mathbb{H}$ denotes the algebra of quaternions.

The products of imaginary units $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$ are defined as $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1$ and $\boldsymbol{ij} = \boldsymbol{k}, \boldsymbol{jk} = \boldsymbol{i}, \boldsymbol{ki} = \boldsymbol{j}, \boldsymbol{ji} = -\boldsymbol{k}, \boldsymbol{kj} = -\boldsymbol{i}$, and $\boldsymbol{ik} = -\boldsymbol{j}$. Note that the multiplication of two quaternions is non-commutative, i.e., $\boldsymbol{ij} \neq \boldsymbol{ji}, \boldsymbol{jk} \neq \boldsymbol{kj}$, and $\boldsymbol{ki} \neq \boldsymbol{ik}$. The *conjugation* of $\mathbf{q}$ is defined as $\overline{\mathbf{q}} = q_0 - q_1\boldsymbol{i} - q_2\boldsymbol{j} - q_3\boldsymbol{k}$. $\mathbf{q}$ is a *pure quaternion* when the real part of $\mathbf{q}$ is 0, i.e., $q_0 = 0$. $\mathbf{q}$ is a *unit quaternion* when the norm of $\mathbf{q}$ equals to 1, i.e., $\|\mathbf{q}\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$.

The polar decomposition of a unit quaternion is given as $\mathbf{q} = e^{\mathbf{o}\frac{\theta}{2}} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k})$, where $\mathbf{o} = o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k}$; $\sqrt{o_1^2 + o_2^2 + o_3^2} = 1$. Note that the multiplication of two quaternions is non-commutative, thereby $e^{\mathbf{o}\frac{\theta}{2}}\mathbf{p}e^{-\mathbf{o}\frac{\theta}{2}} \neq \mathbf{p}$.

Quaternion Rotations: A pure quaternion $\mathbf{q} = 0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$ ($q_1, q_2, q_3 \in \mathbb{R}$) can be considered to have an orientation $[q_1, q_2, q_3]^\top$. In this way, we can rotate a pure quaternion around an axis $[o_1, o_2, o_3]^\top$ ($o_1, o_2, o_3 \in \mathbb{R}, \mathbf{o} = 0 + o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k}, \|\mathbf{o}\| = 1$) with an angle $\theta \in [0, 2\pi)$ using the quaternion $\mathbf{R} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k})$ and its conjugation $\overline{\mathbf{R}} = \cos\frac{\theta}{2} - \sin\frac{\theta}{2}(o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k})$.

$$\mathbf{q}' = \mathbf{R}\mathbf{q}\overline{\mathbf{R}}. \tag{1}$$

The advantage of using quaternions to represent rotations is that quaternions do not suffer from the singularity problem, but the Euler angle [46] and the Rodrigues parameters [37] do. Besides, although the redundancy ratio of quaternions is two (i.e., $\mathbf{R}\mathbf{q}\overline{\mathbf{R}} = (-\mathbf{R})\mathbf{q}\overline{(-\mathbf{R})}$), the redundancy does not affect the rotation-equivariance of features.

Neural Networks With Quaternion Features: To make neural networks satisfy the aforementioned four properties, we revise traditional neural networks to use quaternion features. i.e., each element of the feature vector/matrix/tensor is a quaternion.

In particular, inputs are also represented as quaternion features. However, parameters of such neural networks are still vectors/matrices/tensors composed of real-valued scalars. In comparison, in a traditional neural network, each element of inputs/features/parameters is a real-valued scalar.

Specifically, given an input point cloud, we use a pure quaternion to represent each u-th point $[x_u, y_u, z_u]^\top \in \mathbb{R}^3$ as $\mathbf{x}_u = 0 + x_u\boldsymbol{i} + y_u\boldsymbol{j} + z_u\boldsymbol{k} \in \mathbb{H}$. Similarly, given an intermediate-layer feature, each v-th element is represented as $\mathbf{f}_v = 0 + a_v\boldsymbol{i} + b_v\boldsymbol{j} + c_v\boldsymbol{k} \in \mathbb{H}$, $a_v, b_v, c_v \in \mathbb{R}$.

For a quaternion feature vector $\mathbf{f} = [\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_d]^\top \in \mathbb{H}^d$ with d quaternion elements, we can consider $\mathbf{f}$ as three real-valued feature vectors $a = [a_1, a_2, \ldots, a_d]^\top$, $b = [b_1, b_2, \ldots, b_d]^\top$, and $c = [c_1, c_2, \ldots, c_d]^\top$, where each quaternion element in $\mathbf{f}$, i.e., $\mathbf{f}_v$, can be written as $\mathbf{f}_v = 0 + a_v\boldsymbol{i} + b_v\boldsymbol{j} + c_v\boldsymbol{k}, 1 \le v \le d$. In this way, applying a real-valued convolutional filter w to a quaternion feature $\mathbf{f}$ is equivariant to applying w to three real-valued features a, b and c, i.e., $w \otimes \mathbf{f} = w \otimes (0 + a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k}) = 0 + (w \otimes a)\boldsymbol{i} + (w \otimes b)\boldsymbol{j} + (w \otimes c)\boldsymbol{k}$.

Both quaternion elements of the input and quaternion elements of the feature have orientations. Therefore, we can rotate the quaternion feature vector/matrix/tensor $\mathbf{f}$ by applying the same rotation to each quaternion element in this feature, as follows.

$$\mathbf{f}' = \mathbf{R} \circ \mathbf{f} \circ \overline{\mathbf{R}}, \tag{2}$$

where $\circ$ denotes the element-wise multiplication.[1]

B. Rotation Equivariance

In this section, we define 1. the rotation-equivariance of features, 2. the rotation-equivariance of gradients *w.r.t.* features, and 3. the rotation-invariance of the network training. Then, in Section III-C, we revise layerwise operations of traditional neural networks, so as to ensure that revised layerwise operations satisfy the above three properties.

Rotation-Equivariance of Features: Let $\mathbf{x} \in \mathbb{H}^n$ and $\mathbf{y} = \mathbf{\Phi}(\mathbf{x}) \in \mathbb{H}^C$ denote the input point cloud and the output vector of the neural network, respectively. When we rotate the point cloud $\mathbf{x}$ using a rotation quaternion $\mathbf{R} = e^{\mathbf{o}\frac{\theta}{2}}$, $\mathbf{\Phi}(\mathbf{R} \circ \mathbf{x} \circ \overline{\mathbf{R}}) \in \mathbb{H}^C$ represents the output vector of the neural network given the rotated point cloud. Then, the rotation-equivariance of the network output is formulated as follows.

$$\mathbf{\Phi}(\mathbf{x}^{(\theta)}) = \mathbf{R} \circ \mathbf{\Phi}(\mathbf{x}) \circ \overline{\mathbf{R}}, \quad \text{s.t.} \quad \mathbf{x}^{(\theta)} \overset{\text{def}}{=\!=} \mathbf{R} \circ \mathbf{x} \circ \overline{\mathbf{R}}^1, \tag{3}$$

Equation (3) means that when we rotate the network output around the axis $\mathbf{o}$ with the angle θ (i.e., $\mathbf{R} \circ \mathbf{\Phi}(\mathbf{x}) \circ \overline{\mathbf{R}}$), it is equivalent to rotating the input point cloud first and then computing the network output (i.e., $\mathbf{\Phi}(\mathbf{x}^{(\theta)})$).

Beyond the rotation-equivariance of the network output, in this study, we focus on a more ambitious objective, i.e., making features of each intermediate layer rotation-equivariant. Let $\mathbf{\Phi}(\mathbf{x})$ represent the cascaded functions of a neural network with L layers.

$$\mathbf{\Phi}(\mathbf{x}) = \Phi_L(\Phi_{L-1}(\cdots \Phi_1(\mathbf{x}))), \tag{4}$$

where $\Phi_l(\cdot)$ denotes the function of the l-th layer. Let $\mathbf{f}_l = \Phi_l(\mathbf{f}_{l-1}) \in \mathbb{H}^d$ denote the output of the l-th layer. In fact, as long as we ensure the rotation-equivariance property of each layer's operation $(\Phi_1(\cdot), \ldots, \Phi_l(\cdot))$ in (3), we can recursively ensure the rotation-equivariance of the quaternion feature $\mathbf{f}_l$, as follows (see footnote[2]).

$$\Phi_l(\mathbf{R} \circ \mathbf{f}_{l-1} \circ \overline{\mathbf{R}}) = \mathbf{R} \circ \Phi_l(\mathbf{f}_{l-1}) \circ \overline{\mathbf{R}}. \tag{5}$$

• *Using rotation equivariance that disentangles rotation information, or using rotation invariance that ignores rotation information?* In fact, some applications require rotation-equivariant network outputs, such as the reconstruction task, whereas some applications require rotation-invariant network outputs (i.e., $\mathbf{\Phi}(\mathbf{R} \circ \mathbf{x} \circ \overline{\mathbf{R}}) = \mathbf{\Phi}(\mathbf{x})$), such as the classification task. However, for those applications requiring rotation-invariant network outputs, we can still use rotation-equivariant features in most intermediate layers (except for using rotation-invariant features in the last few layers of the neural network), thereby guaranteeing the rotation robustness of intermediate-layer features. Many previous studies discovered that classic neural networks for 3D point cloud processing were not robust to rotations of input point clouds [34], [60]. The disentanglement of rotation-equivariant representations in intermediate-layer features (i.e., disentangling the information of point clouds' orientations into orientations of quaternion features) guarantees the rotation robustness of neural networks.

Rotation-equivariance of gradients w.r.t. *features* means that in the backpropagation process of a neural network, the feature gradient computed by each layerwise operation $\Phi_l(\cdot)$ is rotation-equivariant *w.r.t.* the orientation of the input point cloud. In other words, when we rotate the input point cloud around the axis $\mathbf{o}$ with the angle θ (i.e., $\mathbf{x}^{(\theta)} \overset{\text{def}}{=\!=} \mathbf{R} \circ \mathbf{x} \circ \overline{\mathbf{R}}$), s.t. $\mathbf{R} = e^{\mathbf{o}\frac{\theta}{2}}$, the feature gradient will rotate with the same angle, as follows.

$$\mathbf{R} \circ \frac{\partial Loss(\mathbf{x})}{\partial \mathbf{f}_l} \circ \overline{\mathbf{R}} = \frac{\partial Loss(\mathbf{x}^{(\theta)})}{\partial \mathbf{f}_l} \tag{6}$$

Here, the loss function $Loss(\mathbf{x})$ is required to be rotation invariant, i.e., no matter how we rotate the input point cloud, the loss function will not change. The loss functions of many tasks are rotation invariant. For example, both the shape classification task and the shape reconstruction task have rotation invariant losses. No matter how we rotate the input object, the loss is not supposed to be affected, since the reconstruction target is also rotated.

In fact, the above rotation-equivariance of feature gradients can be mathematically ensured by the rotation-equivariance of feature gradients of each layerwise operation. Specifically, for all layers, $\forall l$, given the forward function, $\mathbf{f}_l = \Phi_l(\mathbf{f}_{l-1})$, the layerwise rotation-equivariance of feature gradients is given as

[1] $\mathbf{R} \circ \mathbf{f} \circ \overline{\mathbf{R}}$ can also be formulated as $\mathbf{R} \circ [\mathbf{f}_1, \mathbf{f}_2, \ldots]^\top \circ \overline{\mathbf{R}} = [\mathbf{R}\mathbf{f}_1\overline{\mathbf{R}}, \mathbf{R}\mathbf{f}_2\overline{\mathbf{R}}, \ldots]^\top$. Similarly, $\mathbf{x}^{(\theta)} \overset{\text{def}}{=\!=} \mathbf{R} \circ \mathbf{x} \circ \overline{\mathbf{R}}$ can also be formulated as $\mathbf{x}_u^{(\theta)} \overset{\text{def}}{=\!=} \mathbf{R}\mathbf{x}_u\overline{\mathbf{R}}, u = 1, 2, \ldots, n$.

[2] Please see supplementary materials, available online, for the proof that layerwise operations' rotation-equivariance of features can recursively ensure the rotation-equivariance of features of the entire network.

TABLE I
COMPARISON OF FOUR PROPERTIES OF LAYERWISE OPERATIONS IN THE ORIGINAL NEURAL NETWORK

Operation	Rotation-equivariance of features	Rotation-equivariance of feature gradients	Rotation-invariance of the training	Permutation-invariance of features
Convolution	×	×	×	–
ReLU	×	×	×	–
Batch-normalization	×	×	×	–
Max-pooling	×	×	×	–
Dropout	×	×	×	–
Farthest point sampling [30]	invariance	invariance	✓	×
k-NN-search-based grouping [49]	invariance	invariance	✓	✓
Ball-query-search-based grouping [30]	invariance	invariance	✓	×
Density estimation [49]	invariance	invariance	✓	✓
3D coordinates weighting [49]	×	×	×	✓
Graph construction [45]	invariance	invariance	✓	✓

"×" denotes that the operation does not have the property. "✓" denotes that the operation naturally has the property. "–" denotes that the layerwise operation is naturally unrelated to the property (which will be discussed in the last paragraph of Section 3.4). "invariance" means that the operation is naturally rotation-invariant, which is a special case of rotation equivariance that ignores orientation information. This has been discussed in the last paragraph of Section 3.3. Please see Section 3.3 for rules of revising layerwise operations to be rotation-equivariant towards features, feature gradients, and to be rotation-invariant towards the training. Please see Section 3.4 for rules of revising layerwise operations to be permutation-invariant towards features.

follows.

$$\mathbf{R} \circ \left(\frac{\partial \mathbf{f}_l(\mathbf{x})}{\partial \mathbf{f}_{l-1}}^{\top} \nabla_{\mathbf{f}_l} Loss(\mathbf{x}) \right) \circ \overline{\mathbf{R}} = \frac{\partial \mathbf{f}_l(\mathbf{x}^{(\theta)})}{\partial \mathbf{f}_{l-1}}^{\top} \left(\mathbf{R} \circ \nabla_{\mathbf{f}_l} Loss(\mathbf{x}) \circ \overline{\mathbf{R}} \right) \tag{7}$$

where $\nabla_{\mathbf{f}_l} Loss(\mathbf{x}) \overset{\mathbf{def}}{=\!=} \frac{\partial Loss(\mathbf{x})}{\partial \mathbf{f}_l}$ denotes the gradient *w.r.t.* $\mathbf{f}_l$; $\frac{\partial \mathbf{f}_l(\mathbf{x})}{\partial \mathbf{f}_{l-1}}^{\top}$ denotes the gradient of the feature $\mathbf{f}_l$ *w.r.t.* the feature $\mathbf{f}_{l-1}$ given the input sample $\mathbf{x}$; $\frac{\partial \mathbf{f}_l(\mathbf{x}^{(\theta)})}{\partial \mathbf{f}_{l-1}}^{\top}$ denotes the gradient of the feature $\mathbf{f}_l$ *w.r.t.* the feature $\mathbf{f}_{l-1}$ given the rotated sample $\mathbf{x}^{(\theta)}$.

Discussions on Rotation-Equivariance of Feature Gradients: The rotation-equivariance of feature gradients guarantees that all dimensions in the feature gradient are always aligned with dimensions in the feature, no matter how we rotate the input 3D point cloud. This property ensures the stability of training neural networks (i.e., the rotation-invariance of the network training), when training samples have random orientations. Please see supplementary materials, available online, for the proof that the rotation-equivariance of feature gradients ensures the rotation-invariance of the network training.

Rotation-invariance of the training means that when we rotate training samples with arbitrary angles, the neural network trained on the rotated samples will have the same parameters as the neural network trained on the unrotated samples. Thus, we can formulate the rotation-invariance of the training from the following two perspectives. First, we can consider optimal parameters *w.r.t.* the unrotated training samples are the same as optimal parameters *w.r.t.* the rotated training samples, as follows.

$$\arg\min_{w} Loss(\mathbf{x}^{(\theta)}, w) = \arg\min_{w} Loss(\mathbf{x}, w). \tag{8}$$

where $w = \{w_l | l = 1, 2, \ldots, L\}$ denotes network parameters of all layers. In fact, the above rotation-invariance of training can be also defined and mathematically ensured by the rotation-invariance of the gradients *w.r.t.* network parameters, as follows.

$$\frac{\partial Loss(\mathbf{x})}{\partial w_l} = \frac{\partial Loss(\mathbf{x}^{(\theta)})}{\partial w_l} \tag{9}$$

It means that no matter how we rotate input samples, the parameter gradients remain invariant.

C. Rules for Rotation-Equivariance of Features, Rotation-Equivariance of Feature Gradients, and Rotation-Invariance of the Training

To ensure the above three types of layerwise properties, i.e., the rotation-equivariance of features (5), the rotation-equivariance of feature gradients (7), and the rotation-invariance of the training (9), we propose to revise layerwise operations in existing DNNs. It is because most existing layerwise operations do not satisfy the above three layerwise properties, as shown in Table I.

Therefore, in this study, we propose a set of rules to revise typical layerwise operations, which have been widely used in classic neural networks for 3D point cloud processing. In the supplementary material, available online, we prove that the following revisions of layerwise operations make each revised layer satisfy the rotation-equivariance of features. Besides, we also prove that the rotation-equivariance of features of each layerwise operation mathematically ensures the entire neural network's rotation-equivariance of features, the rotation-equivariance of feature gradients, and the rotation-invariance of the training.

Convolution: The convolution operation, $Conv(\mathbf{f}) = w \otimes \mathbf{f} + b$, is revised as $Conv(\mathbf{f}) = w \otimes \mathbf{f}$, where the bias term b is removed. Here w is the real-valued convolutional filter and $\mathbf{f}$ is the quaternion feature.

ReLU: We revise the ReLU operation as follows.

$$ReLU(\mathbf{f}_v) = \frac{\|\mathbf{f}_v\|}{\max\{\|\mathbf{f}_v\|, c\}} \mathbf{f}_v, \tag{10}$$

where $\mathbf{f}_v \in \mathbb{H}$ denotes the v-th element in the feature $\mathbf{f} \in \mathbb{H}^d$; c is a positive constant.

Batch-Normalization: We revise the batch-normalization operation as follows.

$$norm(\mathbf{f}_v^{(i)}) = \frac{\mathbf{f}_v^{(i)}}{\sqrt{\mathbb{E}_j[\|\mathbf{f}_v^{(j)}\|^2] + \epsilon}}, \tag{11}$$

where $\mathbf{f}^{(i)} \in \mathbb{H}^d$ denotes the feature of the i-th sample in the batch; ϵ is a tiny positive constant, which is used to avoid dividing by 0.

Max-Pooling: We revise the traditional max-pooling operation $maxPool(\mathbf{f}) = maxPool\{\mathbf{f}_1, \ldots, \mathbf{f}_d\}$ as follows.

$$maxPool(\mathbf{f}) = \mathbf{f}_{\hat{v}} \quad \text{s.t.} \quad \hat{v} = \arg\max_{v=1,\ldots,d.} [\|\mathbf{f}_v\|]. \tag{12}$$

Note that neural networks for 3D point cloud processing usually use a special max-pooling operation [29]. Given features of n points, $F = [[\mathbf{f}_{1,1}, \ldots, \mathbf{f}_{1,d}]^\top, \ldots, [\mathbf{f}_{n,1}, \ldots, \mathbf{f}_{n,d}]^\top] \in \mathbb{H}^{d\times n}$, each point has a d-dimensional quaternion feature vector, which corresponds to each column of F. Then, the special max-pooling operation in [29] can be actually represented as follows.

$$maxPool_{pc}(F) = \begin{bmatrix} maxPool\{\mathbf{f}_{1,1}, \mathbf{f}_{2,1}, \ldots, \mathbf{f}_{n,1}\} \\ maxPool\{\mathbf{f}_{1,2}, \mathbf{f}_{2,2}, \ldots, \mathbf{f}_{n,2}\} \\ \vdots \\ maxPool\{\mathbf{f}_{1,d}, \mathbf{f}_{2,d}, \ldots, \mathbf{f}_{n,d}\} \end{bmatrix} \tag{13}$$

Equation (13) shows that the special max-pooling operation in [29] can be decomposed into d classic max-pooling operations. Therefore, the revison of (13) is essentially the same as (12).

Dropout: For the dropout operation, we randomly drop out a certain percentage of quaternion elements from the quaternion feature, just in the same manner as the dropout operation in traditional neural networks. Specifically, if the v-th quaternion element ($\mathbf{f}_v = 0 + a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k}$) is dropped, then we set $\mathbf{f}_v = 0 + 0\boldsymbol{i} + 0\boldsymbol{j} + 0\boldsymbol{k}$.

3D Coordinates Weighting: The 3D coordinates weighting [49] is designed to use 3D coordinates to compute weights of intermediate-layer features and reweight features, i.e., $F' = FW^\top$, $F \in \mathbb{H}^{d\times K}$, $W \in \mathbb{R}^{M\times K}$. Specifically, given a 3D point $x_0 \in \mathbb{R}^3$ and its K neighbors $\{x_1, \ldots, x_K\} \in \mathbb{R}^{3\times K}$ (see footnote[3]), weights of intermediate-layer features corresponding to these K points are computed as the result of a single-layer perceptron network, $W = perceptron([x_1 - x_0, \ldots, x_K - x_0]^\top)$.

The basic idea to make the 3D coordinates weighting operation satisfy the rotation-equivariance of features is to make relative coordinates $[x_1 - x_0, \ldots, x_K - x_0]^\top$ be rotation-invariant. To implement this, we use the principal component analysis (PCA) to find out eigenvectors corresponding to the first three principal components of 3D points of the point cloud, i.e., $e_1, e_2, e_3 \in \mathbb{R}^3$. We think that the three principal components of the point cloud are rotation-equivariant. Therefore, we only need to project each point x_k onto these three principal directions, i.e., $x'_k = [x_k^\top e_1, x_k^\top e_2, x_k^\top e_3]^\top$. Here, x'_k is rotation-invariant, thereby $[x'_1 - x'_0, \ldots x'_K - x'_0]^\top$ being also rotation-invariant. In this way, $W = perceptron([x_1 - x_0, \ldots, x_K - x_0]^\top)$ is rotation-invariant, thereby the output feature $F' = FW^\top$ being rotation-equivariant *w.r.t.* the input feature F. It is because the form $F' = FW^\top$ can be considered as a fully-connected layer without the bias term. Because a fully-connected layer can be considered as a special case of a convolutional layer with 1×1 convolutional kernels, according to the revision rule of the convolution operation, removing the bias term ensures that the operation $F' = FW^\top$ satisfies the proposed three types of properties in Section III-B.

Note that when a network architecture contains multiple 3D coordinates weighting layers, we only need to do the PCA operation once to obtain PCA-aligned 3D coordinates of the input sample. All 3D coordinates weighting layers use PCA-aligned coordinates to compute weights. However, except for the 3D coordinates weighting layer, all other network layers, which also depend on 3D coordinates, use the original 3D coordinates for inference.

Operations that are rotation-invariant, not rotation-equivariant. The following operations are naturally rotation-invariant, including the farthest point sampling [30], the k-NN-search-based grouping [45], [49], the ball-query-search-based grouping [30], the density estimation [49], and the graph construction [45] operations. It is because these operations only rely on the euclidean distance between 3D points to learn relationships between 3D points, and the rotation has no effects on the euclidean distance between 3D points. More crucially, the rotation-invariance of these operations' outputs ensures the rotation-equivariance of features of layers after these operations.

D. Rules for Permutation-Invariance

Besides the above three properties, we also expect neural networks to satisfy the permutation-invariance of features. Therefore, we revise layerwise operations to make them permutation-invariant. The farthest point sampling [30] and the ball-query-search-based grouping [30] are two classic widely-used operations of 3D point cloud processing, which are not permutation-invariant. Therefore, we revise these two operations to be permutation-invariant as follows.

Farthest Point Sampling: The farthest point sampling (FPS) [30] is a sampling operation that selects a subset of points from the input point cloud to extract local features. The goal of sampling is to select i points from the input point cloud, which has n points $\Omega = [1, 2, \ldots, n]$. If a total of $i - 1$ points have already been selected, indexed by $S_{i-1} = \{s_1, s_2, \ldots, s_{i-1}\}$, then the FPS operation selects the next point $\mathbf{x}_{\hat{j}}$ as the point farthest from S_{i-1}, i.e., $\hat{j} = \arg\max_{j\in\Omega\setminus S_{i-1}} \min_{k\in S_{i-1}} \|\mathbf{x}_j - \mathbf{x}_k\|$. The first point selected by the FPS affects the following $n - 1$ points, therefore the FPS opereation is not permutation-invariant. Thus, to revise the FPS to be permutation-invariant, we just need to fix the first selected point as the centroid of the input point cloud, which is a virtual point.[4] Therefore, no matter how we reorder input points, the FPS operation would always select the virtual point as the first point, which makes the FPS operation permutation-invariant.

Ball-Query-Search-Based Grouping: To extract contextual information of a given center point, the ball-query-search-based

[3]Note that in the REQNN, each input 3D point is represented as a pure quaternion. Here we use a vector to represent a 3D point, in order to help understand how to map a point to a new coordinate system.

[4]In real implementations, this virtual point does not participate in the network training.

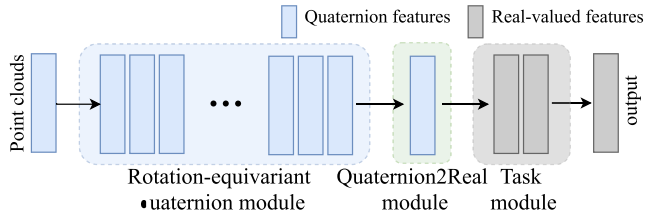

Fig. 2. Illustration of the overview architecture of the REQNN.

grouping [30] operation is typically used to find K neighboring points within a radius for each given center point. This operation is not permutation-invariant, because when there are more than K points within the radius, the first K points will be selected according to the order of points. We revise this operation as follows. When the number of points within the radius exceeds the required number, we select the K nearest neighbor points.

Besides, the following four operations naturally satisfy the permutation-invariant of features, because these operations only rely on euclidean distances between points, including the k-NN-search-based grouping [45], [49], the density estimation [49], the 3D coordinates weighting [49], and the graph construction [45] operations.

Operations That are Orthogonal to the Permutation Operation: There is no need to discuss the permutation-invariance of features of the convolution, the ReLU, the batch-normalization, the max-pooling, and the dropout operations. It is because whether output features of these operations are permutation-invariant depends on whether input features of such operations are permutation-invariant or not. These operations do not directly determine the permutation-invariance of features of the neural network.

E. Overview of the REQNN

Although quaternion features help achieve the rotation-equivariance of features, the rotation-equivariance of feature gradients, and the rotation-invariance of the training, most of the downstream tasks of neural networks (e.g., the 3D shape classification) require outputs of real numbers, as we discussed before. Therefore, after using quaternion features in most layers, we need to transform quaternion features into ordinary real-valued features in the last few layers. Note that for those tasks that outputs can be represented using quaternions, features of the entire neural network are all quaternions. For example, in the point cloud reconstruction task, the reconstructed output 3D coordinates can be represented by quaternions.

Therefore, as Fig. 2 shows, the REQNN consists of three modules, i.e., (a) the rotation-equivariant quaternion module, (b) the Quaternion2Real module, and (c) the task module.

Rotation-Equivariant Quaternion Module: Except for very few layers at the top of the REQNN, other layers in the REQNN comprise the rotation-equivariant quaternion module. This module is used to extract rotation-equivariant quaternion features. We use rules proposed in Sections III-C and III-D to revise layerwise operations in the original neural network to be rotation-equivariant and permutation-invariant, respectively.

Quaternion2Real Module: The Quaternion2Real module is located after the rotation-equivariant quaternion module. The Quaternion2Real module is used to transform quaternion features into vectors/matrices/tensors composed of real numbers. Specifically, given each v-th element of a quaternion feature, $\mathbf{f}_v = 0 + a_v\boldsymbol{i} + b_v\boldsymbol{j} + c_v\boldsymbol{k}$, we compute the square of the norm of each quaternion element as the real-valued feature element, i.e., $\|\mathbf{f}_v\|^2 = a_v^2 + b_v^2 + c_v^2$. In this way, we transform quaternion features into real-valued features, i.e., $[\|\mathbf{f}_1\|^2, \|\mathbf{f}_2\|^2, \ldots, \|\mathbf{f}_d\|^2]^\top \in \mathbb{R}^d$, which are rotation-invariant.

Task Module: The task module is composed of the last few layers of the REQNN. The task module takes real-valued features as the input and generates the final output, so as to implement various tasks similar to traditional neural networks.

• *The proof of the rotation-equivariance of feature gradients and the rotation-invariant of the training.* In supplementary materials, available online, we have proven that revisions in Section III-C can guarantee the layerwise rotation-equivariance of features. We further prove that the rotational-equivariance of features can guarantee the rotation-equivariance of the feature gradients (see Theorems 1 and 2). Besides, the rotational-equivariance of features can also guarantee the rotation-invariance of the training (see Theorems 3). *Please see Section III-F for the experimental verification of Theorems 2 and 3.*

Theorem 1 (Layerwise rotation-equivariance of feature gradients in the rotation-equivariant quaternion module, proven in the supplementary material, available online): In the rotation-equivariant quaternion module of the REQNN, each layerwise operation mentioned above can be written as $\mathbf{f}_l = \Phi_l(\mathbf{f}_{l-1})$. If the input point cloud is rotated with the rotation quaternion $\mathbf{R}$, i.e., $\mathbf{x}^{(\theta)} = \mathbf{R} \circ \mathbf{x} \circ \overline{\mathbf{R}}$, then the gradient w.r.t. $\mathbf{f}_{l-1}$ will also be rotated with $\mathbf{R}$. Thus, the layerwise rotation-equivariance of feature gradients can be formulated as $\mathbf{R} \circ (\frac{\partial \mathbf{f}_l(\mathbf{x})}{\partial \mathbf{f}_{l-1}}^\top \nabla_{\mathbf{f}_l} Loss(\mathbf{x})) \circ \overline{\mathbf{R}} = \frac{\partial \mathbf{f}_l(\mathbf{x}^{(\theta)})}{\partial \mathbf{f}_{l-1}}^\top (\mathbf{R} \circ \nabla_{\mathbf{f}_l} Loss(\mathbf{x}) \circ \overline{\mathbf{R}})$, where $\nabla_{\mathbf{f}_l} Loss(\mathbf{x}) \overset{\mathbf{def}}{=\!=} \frac{\partial Loss}{\partial \mathbf{f}_l}$ denotes the gradient w.r.t. the feature $\mathbf{f}_l$.

Lemma 1 (Rotation-equivariance of the gradient w.r.t. the output of the rotation-equivariant quaternion module*, proven in the supplementary material, available online):* In the REQNN, the gradient of the loss w.r.t. the output feature $\mathbf{f}^{\text{out}}$ of the rotation-equivariant quaternion module is rotation-equivariant, i.e., $\mathbf{R} \circ \frac{\partial Loss(\mathbf{x})}{\partial \mathbf{f}^{\text{out}}} \circ \overline{\mathbf{R}} = \frac{\partial Loss(\mathbf{x}^{(\theta)})}{\partial \mathbf{f}^{\text{out}}}$.

Theorem 2 (Rotation-equivariance and rotation-invariance of feature gradients, proven in the supplementary material, available online, and experimentally verified in Section III-F): According to Theorem 1 and Lemma 1, in the REQNN, the gradient w.r.t. the feature $\mathbf{f}_l$ of any intermediate layer in the rotation-equivariant quaternion module is rotation-equivariant, i.e., $\mathbf{R} \circ \frac{\partial Loss(\mathbf{x})}{\partial \mathbf{f}_l} \circ \overline{\mathbf{R}} = \frac{\partial Loss(\mathbf{x}^{(\theta)})}{\partial \mathbf{f}_l}$. In each layer of the the Quaternion2Real module and the task module, the gradient w.r.t. the feature is rotation-invariant,

Theorem 3 (Rotation-invariance of parameter gradients, proven in the supplementary material, available online, and experimentally verified in Section III-F): In the REQNN (including layers in the rotation-equivariant quaternion module, the

TABLE II
COMPARISONS OF THE PARAMETER NUMBER (#PARAM.), THE FEATURE DIMENSION NUMBER (#FEAT. DIM.), AND THE OPERATION NUMBER (#FLOPS) BETWEEN ORIGINAL DNNS AND REQNNS

	PointNet++[5] [30]			DGCNN[6] [45]			PointConv [49]		
	#Param.(M)	#Feat. dim.(M)	#FLOPs(G)	#Param.(M)	#Feat. dim.(M)	#FLOPs(G)	#Param.(M)	#Feat. dim.(M)	#FLOPs(G)
Ori.	1.48	9.21	0.86	2.86	16.78	3.76	19.57	13.04	1.22
REQNN	1.47	26.44	2.51	2.86	39.85	9.04	20.61	33.66	3.58

All neural networks were tested on the ModelNet40 dataset. "FLOPs" denotes the floating-point operations per second.

Quaternion2Real module, and the task module), the gradient w.r.t. the parameter w_l of any intermediate layer is rotation-invariant, i.e., $\frac{\partial Loss(\mathbf{x})}{\partial w_l} = \frac{\partial Loss(\mathbf{x}^{(\theta)})}{\partial w_l}$.

The rotation-invariance of parameter gradients mathematically ensures the rotation-invariance of the training of the REQNN (or the stability of learning the REQNN). This has been proved in the supplementary material, available online, and has been experimentally verified in Section III-F (please see Table VI for results). The rotation-invariance/stability of training means that no matter how we rotate input samples, the REQNN will be trained to have the same parameters. The most direct benefit of this property is that we do not need to conduct rotation data augmentation on input samples, and the REQNN can achieve the same performance as that of a REQNN with rotation data augmentation.

• *Numbers of parameters, feature dimensions, and floating-point operations, and time for training.* We conducted experiments to compare the number of parameters, feature dimensions, floating-point operations, and the training time between traditional DNNs and REQNNs. We revised the PointNet++,[5] the DGCNN,[6] and the PointConv into their corresponding REQNNs by following the setting in Section III-G. All DNNs were tested on the ModelNet40 [50] dataset. As Table II shows, the parameter number of each REQNN was almost the same as that of the corresponding original DNN. The feature dimension number of the REQNN was approximately three times of that of the corresponding original DNN. Besides, the floating-point operation number of each REQNN was approximately three times of that of the corresponding original DNN.

Besides, we also measured the time for network training. All models were evaluated on the same environment, including the Python of the version 3.9.12, the CUDA of the version 11.6, the numpy of the version 1.22.3, the torchvision of the version 0.14.1, the scikit-learn of the version 1.2.2, and the type of GPU is NVIDIA TITAN RTX. We trained both the traditional DNN and the REQNN revised from the traditional DNN on the ModelNet40 dataset in the above environment, in order to measure the training time. Results in Table III show that the training time for the REQNN was about 2-4 times more than the average training time for the traditional DNN.

[5]The PointNet++ for shape classification used in this study is slightly revised by concatenating 3D coordinates to input features of the 1st and 4th convolution layers, in order to enrich the input information. For fair comparisons, both the REQNN and the original PointNet++ are revised in this way.

[6]We add one more convolution layer in the Quaternion2Real module in the REQNN revised from DGCNN, in order to obtain reliable real-valued features considering that the DGCNN has no downsampling operations. For fair comparisons, we add the same convolution layer to the same location of the original DGCNN.

TABLE III
AVERAGE TIME (SECOND) FOR TRAINING DIFFERENT DNNS ON THE MODELNET40 DATASET FOR ONE EPOCH

	PointNet++[5]	DGCNN[6]	PointConv
Ori.	53.04	66.32	62.35
REQNN	132.53	269.51	140.24

F. Verification of Theorems 2 and 3

We conducted experiments on the 3D shape classification task to verify the rotation-equivariance of feature gradients in Theorem 2, the rotation-invariance of parameter gradients in Theorem 3, and the rotation-invariance of the training.

Verifying the rotation-equivariance of feature gradients in Theorem 2: We constructed an REQNN based on the DGCNN by following settings in Section III-G, and we then trained the REQNN on the ModelNet10 dataset [50]. This experiment was designed to prove the rotation-equivalence of feature gradients computed in the following two ways. *The first way:* given the original point cloud $\mathbf{x}$, we computed the gradient of the loss *w.r.t.* the l-th layer's feature, i.e., $\frac{\partial Loss(\mathbf{x})}{\partial \mathbf{f}_l}$. Then, we rotated the gradient as $\mathbf{R} \circ \frac{\partial Loss(\mathbf{x})}{\partial \mathbf{f}_l} \circ \overline{\mathbf{R}}$. *The second way:* given the point cloud that was rotated by the same angle, $\mathbf{x}^{(\theta)} = \mathbf{R} \circ \mathbf{x} \circ \overline{\mathbf{R}}$, we computed the gradient of the loss *w.r.t.* the l-th layer's feature, i.e., $\frac{\partial Loss(\mathbf{x}^{(\theta)})}{\partial \mathbf{f}_l}$. Therefore, given two types of feature gradients computed by two ways, we used the metric $\Delta grad_l^{\text{feature}} = \left\| \frac{\partial Loss(\mathbf{x}^{(\theta)})}{\partial \mathbf{f}_l} - \mathbf{R} \circ \frac{\partial Loss(\mathbf{x})}{\partial \mathbf{f}_l} \circ \overline{\mathbf{R}} \right\|_F \Big/ \left\| \frac{\partial Loss(\mathbf{x}^{(\theta)})}{\partial \mathbf{f}_l} \right\|_F$ to examine whether we could synthesize the feature gradient of a rotated sample by rotating the feature gradient of the unrotated sample.

According to Table IV, $\forall l, \Delta grad_l^{\text{feature}} \approx 0$ indicated that the feature gradient of a rotated sample could be also obtained by directly rotating the feature gradient of the unrotated sample. It proved the rotation-equivariance of feature gradients.

Note that the accumulation of tiny systematic computation errors in the k-NN-search-based grouping operation in the DGCNN might destroy the rotation-equivariance of feature gradients. Please see supplementary materials, available online, for the detailed analysis. Therefore, for the first layer of the k-NN-search-based grouping operation in the REQNN, we kept the neighboring points of each given point unchanged, when we rotated the input point cloud, in order to avoid the problem caused by the tiny systematic computational error. We applied such settings in two experimental verifications in Section III-F. In addition, we did not fix neighboring points of each given 3D point in all experiments in Section IV, which better reflected the true performance of the REQNN.

Verifying the rotation-invariance of parameter gradients in Theorem 3 and the rotation-invariance of the network training:

TABLE IV
VERIFY THE ROTATION-EQUIVARIANCE OF GRADIENTS *W.R.T.* FEATURES OF THE REQNN

$\Delta grad_l^{\text{feature}}$	the 1st linear layer	the 2nd linear layer	the 3rd linear layer	the 4th linear layer	the 5th linear layer
REQNN (revised from DGCNN)	2.16e-15	2.18e-15	2.18e-15	2.16e-15	1.52e-15

$\Delta grad_l^{\text{feature}}$ measured the relative difference between the feature gradients of the rotated sample, $\frac{\partial Loss(\mathbf{x}^{(\theta)})}{\partial \mathbf{f}_l}$, and the rotated feature gradients of the unrotated sample, $\mathbf{R} \circ \frac{\partial Loss(\mathbf{x})}{\partial \mathbf{f}_l} \circ \overline{\mathbf{R}}$. $\forall l, \Delta grad_l^{\text{feature}} \approx 0$ proved that feature gradients of the REQNN were rotation-equivariant. The small deviation was caused by the accumulation of tiny systematic computational errors in a DNN.

TABLE V
VERIFY THE ROTATION-INVARIANCE OF GRADIENTS *W.R.T.* PARAMETERS OF THE REQNN

$\Delta grad_l^{\text{parameter}}$	the 1st linear layer	the 2nd linear layer	the 3rd linear layer	the 4th linear layer	the 5th linear layer
REQNN (revised from DGCNN)	1.83e-15	1.90e-15	2.04e-15	2.07e-15	1.62e-15

$\Delta grad_l^{\text{parameter}}$ measured the relative difference between parameter gradients of the REQNN trained on rotated samples and those of the REQNN trained on unrotated samples. $\forall l, \Delta grad_l^{\text{parameter}} \approx 0$ proved that parameter gradients of the REQNN were rotation-invariant. The small deviation was caused by the accumulation of tiny systematic computational errors in a DNN.

TABLE VI
VERIFY THE ROTATION-INVARIANCE OF NETWORK PARAMETERS OF THE REQNN

$\Delta\text{parameter}_l$	the 1st linear layer	the 2nd linear layer	the 3rd linear layer	the 4th linear layer	the 5th linear layer
REQNN (revised from DGCNN)	1.85e-16	1.19e-15	1.05e-15	1.14e-15	1.36e-15

$\Delta\text{parameter}_l$ measured the relative difference between parameters of the REQNN trained on rotated samples and those of the REQNN trained on unrotated samples. $\forall l, \Delta\text{parameter}_l \approx 0$ proved that parameters of the REQNN were rotation-invariant. The small deviation was caused by the accumulation of tiny systematic computational errors in a DNN.

We constructed an REQNN based on the DGCNN by following settings in Section III-G, and we then trained the REQNN using the following two training sets. The first training set contained the original point clouds $\mathbf{x}$ in the ModelNet10 dataset. The second training set contained the rotated point clouds $\mathbf{x}^{(\theta)} = \mathbf{R} \circ \mathbf{x} \circ \overline{\mathbf{R}}$. Thus, these two REQNNs were termed REQNN^{ori} and REQNN^{rotated}. In this way, we examined whether parameter gradients of these two REQNNs were the same, as well as whether parameters of these two REQNNs were the same.

We used the metric $\Delta grad_l^{\text{parameter}} = \left\| \frac{\partial Loss(\mathbf{x})}{w_l^{\text{ori}}} - \frac{\partial Loss(\mathbf{x}^{(\theta)})}{w_l^{\text{rotated}}} \right\|_F \Big/ \left\| \frac{\partial Loss(\mathbf{x})}{w_l^{\text{ori}}} \right\|_F$ to measure the difference between the parameters in the l-th layer trained on original samples and those trained on rotated samples. According to Table V, $\forall l, \Delta grad_l^{parameter} \approx 0$ indicated that parameter gradients of the REQNN^{ori} and that of the REQNN^{rotated} were almost the same. Thus, we proved the rotation-invariance of parameter gradients.

Besides, we also used the metric $\Delta parameter_l = \frac{\|w_l^{\text{ori}} - w_l^{\text{rotated}}\|_F}{\|w_l^{\text{ori}}\|_F}$ to measure the relative difference between network parameters trained on original samples and network parameters trained on rotated samples. According to Table VI, $\forall l, \Delta parameter_l \approx 0$ indicated that parameters of the REQNN^{ori} and those of the REQNN^{rotated} were almost the same, although the error would be accumulated during the training process. Thus, we proved the rotation-invariance of the network training.

G. Revisions of Traditional DNNs into REQNNs

In this study, we revise the following four neural networks into REQNNs, including PointNet++ [30], DGCNN [45], PointConv [45], and PointNet [29].

TABLE VII
LAYERWISE OPERATIONS USED BY DIFFERENT DNNS

Layerwise operation	[30]	[45]	[49]	[29]
Convolution	✓	✓	✓	✓
ReLU	✓	✓	✓	✓
Batch-normalization	✓	✓	✓	✓
Max-pooling	✓	✓		✓
Dropout	✓	✓	✓	✓
Farthest point sampling	✓		✓	
k-NN-search-based grouping		✓	✓	
Ball-query-search-based grouping	✓			
Density estimation			✓	
3D coordinates weighting			✓	
Graph construction		✓		

[30] refers to the PointNet++, [45] refers to the DGCNN, [49] refers to the PointConv, and [29] refers to the PointNet.

Model 1, PointNet++: To revise the PointNet++ [30] for shape classification[5] into an REQNN, we take the last three fully-connected (FC) layers as the task module and take other layers as the rotation-equivariant quaternion module. Besides, there is a Quaternion2Real module between these two modules for converting quaternions to real numbers. As Table VII shows, we revise four types of layerwise operations to be rotation-equivariant according to the rules in Section III-C, including the convolution, ReLU, batch-normalization, and max-pooling operations. We also revise the farthest point sampling and ball-query-search-based grouping operations to make them permutation-invariant according to the rules in Section III-D.

Model 2, DGCNN: To revise the DGCNN [45] for shape classification into an REQNN, we take the last three FC layers as the task module and take other layers as the rotation-equivariant quaternion module. Besides, we add a Quaternion2Real module[6] between these two modules. As Table VII shows, we revise four types of layerwise operations to be rotation-equivariant

according to the rules in Section III-C, including the convolution, ReLU, batch-normalization, and max-pooling operations. Note that all layerwise operations in the original DGCNN are naturally permutation-invariant.

Model 3, PointConv: To revise the PointConv [49] for shape classification into an REQNN, we use the last three FC layers as the task module and take other layers as the rotation-equivariant quaternion module. The Quaternion2Real module is added between these two modules. As Table VII shows, we revise four types of layerwise operations to be rotation-equivariant according to the rules in Section III-C, including the convolution, ReLU, batch-normalization, and 3D coordinates weighting operations. We also revise the farthest point sampling operation to make it permutation-invariant according to the rule in Section III-D.

Model 4, PointNet: In order to construct an REQNN for shape reconstruction, we slightly revise the architecture of the PointNet [29] for shape classification. We take all remaing layers in the PointNet as the rotation-equivariant quaternion module, except for the max-pooling and Spatial Transformer Network (STN) [18]. The STN discards all the spatial information (including the rotation information) of the input point cloud. Therefore, in order to encode the rotation information, we remove the STN from the original PointNet. Note that there is no the Quaternion2Real module or the task module in this REQNN, so that all features in the REQNN for reconstruction are quaternion features. As Table VII shows, we revise four types of layerwise operations to be rotation-equivariant according to the rules in Section III-C, including the convolution, ReLU, batch-normalization, and dropout operations.

IV. Experiments

Since we have introduced the rotation-equivariance and the permutation-invariance properties in the REQNN, in this section, we mainly conducted experiments to demonstrate the superior performance of the REQNN. We used our method to revise different classic DNNs for 3D point cloud processing into different REQNNs, and tested their performance. We also evaluated the representation capacity of the REQNN from two other perspectives, i.e., the stability of input attributions *w.r.t.* different rotations and adversarial robustness to the rotation attack. Furthermore, in all experiments in this section, we set $c = 1$ (see (10)) and $\epsilon = 10^{-5}$ (see (11)). We detached the derivatives of $\|\mathbf{f}_v\|/\max\{\|\mathbf{f}_v\|, c\}$ (see (10)) and $1/\sqrt{\mathbb{E}_j[\|\mathbf{f}_v^{(j)}\|^2] + \epsilon}$ (see (11)) in the back-propagation.

A. REQNNs for Different Tasks

1) 3D Shape Classification: In this subsection, we compared the classification accuracy on 3D point clouds under arbitrary rotations between the original DNN and the REQNN revised from the original DNN. To this end, we revised three widely used DNNs for the shape classification task into different REQNNs, including PointNet++ [30], DGCNN [45], and PointConv [49]. All DNNs were learned based on the ModelNet40 [50] dataset provided by [29], the 3D MNIST [1] dataset, and the ShapeNet dataset [4]. The above three datasets consisted of 40 categories, 10 categories, and 16 categories, respectively. For each shape, we selected the first 1024 points in the original point set by following common settings in [29], [30], [45], [49]. Given each benchmark dataset, we generated a testing set by arbitrarily rotating each sample in the original testing set ten times, in order to test the classification accuracy under different rotations.

We compared the classification accuracy of the following three models. The first baseline model was the original DNN trained on point clouds without rotations. The second baseline model was the original DNN was trained on point clouds arbitrarily rotated along the y-axis. The y-axis rotation augmentation has been widely used in [30], [45]. The proposed REQNN was the third competing method, which was revised from the traditional DNN and trained on point clouds without rotations. In fact, we have proven the rotation-invariance of the training of the REQNN. i.e., no matter whether we rotated the training samples, the REQNN converged to the same parameters. We have verified the rotation-invariance of the training of the REQNN in experiments in Section III-F. Therefore, there was no need to compare REQNNs trained on randomly rotated samples with REQNNs trained on unrotated samples.

Comparing classification accuracy on arbitrarily rotated objects: Table VIII shows the classification accuracy of DNNs tested with point clouds under arbitrary rotations.[7] Results indicate that the classification accuracy of the REQNN was always higher than that of all traditional DNNs, no matter whether the traditional DNNs were trained with or without rotation augmentations. The REQNN revised from DGCNN[6] achieved the highest accuracy of 84.57%. In comparison, traditional DNNs trained without rotation augmentations exhibited very low accuracy, 25.01%–32.08% on the ModelNet40 dataset, 44.19%–45.90% on the 3D MNIST dataset, and 37.03%–44.06% on the ShapeNet dataset. Results also indicate that traditional DNNs trained with y-axis rotation augmentations had higher classification accuracy than traditional DNNs trained without rotation augmentations. However, the improvement in accuracy brought by y-axis rotation data augmentations was limited. It was because the y-axis rotation could not guarantee that the DNN learned the information of arbitrary rotations.

Comparing the drop of the generalization power on arbitrarily rotated objects: We found that the above traditional DNNs usually had a performance drop on arbitrarily rotated testing objects, compared to the performance on original testing objects. The performance drop could be considered as an over-fitting problems due to the lack of rotation invariance/equvariace. Therefore, for each trained DNN, we hoped to compare the generalization power on objects in original datasets and the generalization power on arbitrarily rotated objects. We compared DNNs in the following two scenarios, i.e., DNNs learned with **N**o **R**otations and tested with **N**o **R**otations (NR/NR), and DNNs learned with **N**o **R**otations and tested with **A**rbitrary **R**otations

[7]The classification accuracy in the scenario of NR/AR in Tables VIII and IX was slightly different for PointNet++ [30] (25.87% *versus* 21.35%) and DGCNN [45] (32.08% *versus* 29.74%). It was because architectures of PointNet++[5] and DGCNN[6] examined in Table VIII and Table IX were slightly different. Nevertheless, this did not essentially change our conclusions.

TABLE VIII
ACCURACY OF 3D SHAPE CLASSIFICATION OF THE TRADITIONAL/ORIGINAL DNN AND THE REQNN REVISED FROM THE TRADITIONAL DNN

Architecture	ModelNet40 dataset			3D MNIST dataset			ShapeNet dataset		
	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations
PointNet++[5]	25.87[7]	29.25	**62.03**	44.19	51.48	**72.01**	41.60	43.53	**94.42**
DGCNN[6]	32.08[7]	33.78	**84.57**	45.90	50.00	**85.07**	44.06	50.39	**96.90**
PointConv	25.01	26.46	**81.93**	45.51	48.08	**85.71**	37.03	39.60	**97.59**

"Ori. DNN trained w/o rotations" indicates the original neural network learned without rotations. "Ori. DNN trained w/ rotations" indicates the original neural network learned with the y-axis rotation augmentation (the y-axis rotation augmentation has been widely applied in [30], [45]). "REQNN trained w/o rotations" indicates the REQNN learned without rotations. Note that the accuracy of shape classification reported in [30], [45], [49] was obtained under the test without rotations. The accuracy reported here was obtained under the test with rotations. Therefore, it is normal that the accuracy in this paper is lower than the accuracy in those papers.

TABLE IX
ACCURACY OF 3D SHAPE CLASSIFICATION OF DIFFERENT DNNS ON THE MODELNET40 DATASET

Method	**NR/NR** (do **not** consider rotation) in testing)	**NR/AR** (consider rotation in testing)
PointNet [29]	88.45	12.47
PointNet++ [30]	89.82	21.35[7]
Point2Sequence [24]	92.60	10.53
KD-Network [21]	86.20	8.49
RS-CNN [25]	92.38	22.49
DGCNN [45]	**92.90**	29.74[7]
PRIN [53]	80.13	68.85
QE-CapsuleNet [59]	74.73	74.07
REQNN (revised from DGCNN[6])	84.64 $\approx$	**84.57**

NR/NR denotes that DNNs were learned and tested with **N**o **R**otations. **NR/AR** denotes that DNNs were learned with **N**o **R**otations and tested with **A**rbitrary **R**otations. Experimental results show that the REQNN exhibited the highest rotation robustness.

(NR/AR). We measured the generalization power using the following two methods, (1) comparing the classification accuracy on objects in the original datasets and arbitrarily rotated objects; and (2) comparing the gap between the training loss and the testing loss.

First, Comparing Classification Accuracy: We compared the REQNN with eight state-of-the-art DNNs for 3D point cloud classification. Table IX shows that REQNNs had approximately the same classification accuracy in scenarios of NR/NR and NR/AR, because rotations had no influence on the training of REQNNs. A tiny difference in the classification accuracy was caused by computational errors.

As Table IX shows, the REQNN revised from the DGCNN[6] achieved the highest accuracy of 84.57% in the scenario of NR/AR, which indicated that the REQNN was significantly robust to rotations. In contrast, traditional DNNs had much lower classification accuracy in the scenario of NR/AR than in the scenario of NR/NR, including PointNet [29], PointNet++ [30], Point2Sequence [24], KD-Network [21], RS-CNN [25], and DGCNN [45]. It was because traditional DNNs could not handle point clouds with unseen orientations, although they achieved high accuracy in the scenario of NR/NR. Compared with traditional DNNs, PRIN [53] and QE-CapsuleNet [59], which also made efforts to improve the rotation robustness, had obtained certain robustness to rotations. Nevertheless, our REQNN outperformed them by 15.72% and 10.5%, respectively, in the scenario of NR/AR.

Second, Comparing the Loss Gap: We used the gap between the training loss and the testing loss to measure REQNNs' generalization of classification. If a DNN suffered from overfitting, then the DNN would exhibit a significant drop in classification generalization on rotated samples. To this end, we analyzed the classification generalization on different testing samples between REQNNs and traditional DNNs. Specifically, we measured the gap between the training loss and the testing loss in scenarios of NR/NR, i.e., $gap^{\text{NR/NR}} = |Loss^{\text{NR}}_{\text{train}} - Loss^{\text{NR}}_{\text{test}}| \in \mathbb{R}^+$, which represented the generalization error when both training samples and testing samples were unrotated. Similarly, $gap^{\text{NR/AR}} = |Loss^{\text{NR}}_{\text{train}} - Loss^{\text{AR}}_{\text{test}}| \in \mathbb{R}^+$ represented the generalization error when training samples were unrotated and testing samples were rotated. Therefore, the relative difference between the two gaps $diff = \frac{|gap^{\text{NR/AR}} - gap^{\text{NR/NR}}|}{gap^{\text{NR/AR}}}$ measured the additional generalization error of the DNN caused by the rotation sensitivity.

We also conducted experiments on REQNNs and two types of baseline DNNs as mentioned above. Experimental results in Table X show that all REQNNs had very small values of $diff$ (not greater than 0.002), which indicated the excellent classification generalization of REQNNs. In contrast, traditional networks (both learned with and without rotations) had vary high values of $diff$ (0.821–0.968 on the ModelNet40 dataset, 0.879–0.964 on the 3D MNIST dataset, and 0.987–0.999 on the ShapeNet dataset), which indicated the poor classification generalization of traditional networks.

Discussion: When we tested DNNs on unrotated samples, each REQNN exhibited lower test accuracy than the corresponding traditional DNN. For example, the test accuracy of the DGCNN on unrotated samples was 92.90%, while that of the REQNN revised from the DGCNN was 84.64%. It was because when we revised the DGCNN into an REQNN for rotation equivariance, the cost was that it increased the model complexity. Training a complex DGCNN was more difficult to optimize than training a relatively simple DGCNN, which was responsible for the performance drop. However, if we removed the bias in testing samples by rotating testing samples to arbitrary orientations, then the REQNN outperformed traditional DNNs, as shown in the column of "NR/AR (consider rotation in testing)" in Table IX. This demonstrated the true representation power of REQNNs in a more convincing manner.

Training on unrotated objects versus*training on arbitrarily rotated objects.* We conducted a new experiment to compare the DNNs trained on the original dataset with the DNNs trained on arbitrarily rotated objects. It was because in the previous experiment corresponding to Table IX, we found that traditional

TABLE X
COMPARING THE INCREASE OF THE OVER-FITTING LEVEL ON ROTATED SAMPLES BETWEEN REQNNS AND TRADITIONAL/ORIGINAL DNNS

Architecture	ModelNet40 dataset			3D MNIST dataset			ShapeNet dataset		
	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations
PointNet++[5]	0.968	0.821	**0**	0.949	0.879	**0.002**	0.999	0.996	**0**
DGCNN[6]	0.892	0.854	**0**	0.938	0.887	**0.001**	0.991	0.987	**0.001**
PointConv	0.891	0.889	**0**	0.964	0.912	**0.001**	0.991	0.995	**0**

Specifically, we computed the relative difference $gap^{\mathrm{NR/NR}} = |Loss^{\mathrm{NR}}_{\mathrm{train}} - Loss^{\mathrm{NR}}_{\mathrm{test}}|$ to measure the over-fitting level of the DNN tested on unrotated samples. Similarly, we used $gap^{\mathrm{NR/AR}} = |Loss^{\mathrm{NR}}_{\mathrm{train}} - Loss^{\mathrm{AR}}_{\mathrm{test}}|$ to measure the over-fitting level of the DNN tested on arbitrarily rotated samples. Then, this table reports $diff = \frac{|gap^{\mathrm{NR/AR}} - gap^{\mathrm{NR/NR}}|}{gap^{\mathrm{NR/AR}}}$ as the additional over-fitting level when the DNN was tested on more generic samples with random rotations. Thus, this metric reflected the generalization power affected by rotation variances. Results show that REQNNs exhibited stronger generalization power than traditional DNNs.

TABLE XI
ACCURACY OF 3D SHAPE CLASSIFICATION OF THE TRADITIONAL/ORIGINAL DNN AND THE REQNN REVISED FROM THE TRADITIONAL DNN. FOR EACH NETWORK ARCHITECTURE, THREE TRADITIONAL DNNS WERE TRAINED ON RAW OBJECTS IN THE MODELNET 40 DATASET, ON THE Y-AXIS ROTATED OBJECTS, AND ON THE RANDOMLY ROTATED OBJECTS, RESPECTIVELY. REQNNS WERE TRAINED ON RAW UNROTATED OBJECTS

	PointNet++	DGCNN	PointConv
Ori. DNN on unrotated objects	25.87	32.08	25.01
Ori. DNN on y-axis rotated objects	25.04	31.34	23.93
Ori. DNN on randomly rotated objects	27.45	33.43	28.50
REQNN on unrotated objects	**62.03**	**84.57**	**81.93**

DNNs had a performance drop on arbitrarily rotated samples, which was caused by the bias[8] of sample collection in the dataset. Thus, the new experiment was designed to illustrate the significance of the dataset bias. To this end, we constructed the following new datasets. The first new dataset was constructed by rotating each training sample in the ModelNet40 dataset around an arbitrary axis with an arbitrary angle, termed the *random rotation dataset*. The second new dataset was constructed by rotating each training sample in the ModelNet40 dataset around the y-axis with an arbitrary angle, termed the *y-axis rotation dataset*. Then, for each type of network architecture, we trained the following four types of models, including the traditional DNN trained on the original/unrotated ModelNet40 dataset, the traditional DNN trained on the *random rotation dataset*, the traditional DNN trained on the *y-axis rotation dataset*, and the REQNN trained on the original/unrotated ModelNet40 dataset. Each DNN was tested on randomly rotated objects, which followed settings in experiments corresponding to Table VIII.

Results in Table XI show that REQNNs trained on unrotated objects exhibited higher classification accuracy than all three traditional DNNs (including the DNN trained on unrotated objects, the DNN trained on y-axis rotated objects, and the DNN trained on randomly rotated objects). This meant that traditional DNNs were more sensitive to the dataset bias. Besides, even when we used an unbiased dataset (including randomly rotated objects) to train different DNNs, REQNNs still exhibited higher classification accuracy than traditional DNNs on randomly rotated testing objects. Note that traditional DNNs trained on the *y-axis rotation dataset* exhibited a bit lower classification accuracy than the traditional DNN trained on the original dataset, because unlike rotation augmentation, the newly constructed *y-axis rotation dataset* did not contain more objects than the original dataset, but it had more diverse 3D point clouds. The diversity of object orientations prevented the DNN from being over-fitted to certain short-cut orientation features, thereby boosting the training difficulty. In addition, there was still a gap between the y-axis rotated training samples and the randomly rotated testing samples. This explained the tiny performance drop of the DNN trained on y-axis rotated objects.

2) 3D Point Cloud Reconstruction: In this experiment, we rotated intermediate-layer quaternion features of the original point cloud, so as to synthesize new point clouds with target orientations. To this end, we learned an REQNN revised from the PointNet [29] for point cloud reconstruction on the ShapeNet [4] dataset. In our implementation, each point cloud consisted of 1024 points. We used the output quaternion feature of the top fourth linear transformation layer of the REQNN to synthesize quaternion features with different orientations. Such synthesized quaternion features were used to reconstruct point clouds with target orientations.

As shown in Fig. 3, for each point cloud (Fig. 3 "original" (a)), we directly rotated it with different angles (Fig. 3 "original" (b)–(e)). For comparison, we rotated the quaternion feature corresponding to the original point cloud with the same angles to synthesize quaternion features. These generated quaternion features were used to reconstruct point clouds (Fig. 3 "reconstructed" (b)–(e)). We observed that these reconstructed point clouds had the same orientations as those of point clouds generated by directly rotating the original point cloud.

B. *Representation Stability* w.r.t. *Rotations*

In this subsection, we proposed a new metric to evaluate the stability of input attributions encoded by REQNNs when we rotated input point clouds with different angles. Ideally, given two point clouds with the same 3D structure but different orientations, the REQNN was supposed to encode similar feature representations. Thus, attributions of different point cloud regions were supposed to keep unchanged, when the point cloud was rotated.

Specifically, we uniformly divided the entire point cloud into n cloud regions by following [34], which were denoted by $N = \{1, 2, \ldots, n\}$. Then, we computed the Shapley value for each cloud region. The Shapley value had been widely used to measure the numerical attribution of each input variable (here, each point cloud region) to the output score of the neural network [15], [26], [33], [34], [39].

To compute the Shapley value for each region in the point cloud, we defined the neural network's output score $v(S)$ given a subset of point cloud regions $S \subseteq N$ as follows. Let x_S denote

[8]The dataset bias meant that point clouds in the same category usually had similar orientations. For example, the directions of heads of most airplanes in the ModelNet40 dataset were approximately parallel to the axis of [0,0,1] or the axis of [1,0,0]. Thus, if training samples were all biased to specific orientations, then the DNN would probably be overfitted to the orientation feature.

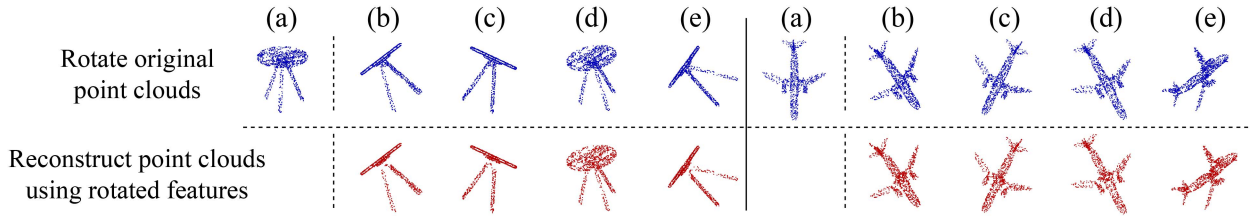

Fig. 3. Manual manipulation of intermediate-layer features to control the object rotation in 3D point cloud reconstruction. The experiment was conducted to prove that point clouds reconstructed using the synthesized quaternion features had the same orientations as point clouds generated by directly rotating the original point cloud. Here we displayed results of four random orientations for each point cloud. Point clouds ("original" (b)–(e)) were generated by directly rotating the original point cloud ("original" (a)) around axis $[0.46, 0.68, 0.56]^\top$ with angle $\frac{\pi}{3}$, around axis $[-0.44, -0.61, 0.66]^\top$ with angle $\frac{\pi}{4}$, around axis $[0.34, 0.94, 0.00]^\top$ with angle $\frac{\pi}{6}$, and around axis $[0.16, 0.83, 0.53]^\top$ with angle $\frac{2\pi}{3}$, respectively. Given a specific intermediate-layer quaternion feature of the original point cloud ("original" (a)), we rotated the quaternion feature with the same angles to obtain quaternion features with different orientations, which were used to reconstruct point clouds ("reconstructed" (b)–(e)).

TABLE XII
COMPARING THE STABILITY OF INPUT ATTRIBUTIONS ENCODED BY DIFFERENT DNNS

Architecture	ModelNet40 dataset			3D MNIST dataset			ShapeNet dataset		
	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations
PointNet++[5]	0.383	0.242	**1.0**	0.249	0.392	**0.986**	0.199	0.462	**1.0**
DGCNN[6]	0.546	0.371	**1.0**	0.323	0.318	**0.994**	0.406	0.463	**0.999**
PointConv	0.204	0.603	**1.0**	0.268	0.645	**1.0**	0.272	0.447	**1.0**

The stability value of each REQNNs was 1.0 or very close to 1.0, which indicated that input attributions of REQNNs kept almost unchanged *w.r.t.* the rotation of input samples. The slight change was caused by the systematic errors of the computation. This exhibited that REQNNs were robust to rotations. In contrast, the low stability values of traditional DNNs indicated their high sensitivity to rotations.

the point cloud that only contained regions in S, and regions in $N \setminus S$ had been removed from the point cloud. Considering that existing neural networks for 3D point cloud processing could usually handle point clouds with a fixed number of points, we reset coordinates of points in $N \setminus S$ to the center of the entire point cloud by following [61], instead of physically removing these points. Given the input x_S, the output score of the pre-trained neural network for shape classification was computed as $v(S) = \log \frac{p}{1-p}$, where $p = p(y = y^{\text{truth}} \mid x_S)$ denoted the probability of the ground-truth category. In this way, the numerical attribution of region i to the overall network output was estimated as the Shapley value $\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-|S|-1)!}{n!}(v(S \cup \{i\}) - v(S))$. ϕ_i was computed via the sampling-based approximation method in [3].

Given an input point cloud x, let $x^{(1)} = \theta_1(x)$ and $x^{(2)} = \theta_2(x)$ denote two point clouds obtained by rotating x with two different rotation operations θ_1 and θ_2, respectively. We measured the distribution of attributions over different point cloud regions, i.e., Shapley values $\Phi = [\phi_1, \ldots, \phi_n]^\top \in \mathbb{R}^n$. The stability of regional attributions encoded by a neural network under different rotations is quantified as follows.

$$stability = \mathbb{E}_x \mathbb{E}_{\theta_1,\theta_2}[\text{cosine}(\Phi_{x^{(1)}=\theta_1(x)}, \Phi_{x^{(2)}=\theta_2(x)})]. \tag{14}$$

where $\text{cosine}(\Phi_{x^{(1)}=\theta_1(x)}, \Phi_{x^{(2)}=\theta_2(x)}) = \frac{\Phi_{x^{(1)}}^\top \Phi_{x^{(2)}}}{\|\Phi_{x^{(1)}}\| \|\Phi_{x^{(2)}}\|} \in \mathbb{R}$ measured the similarity between two point clouds' regional attributions.

We compared the stability between three types of baseline neural networks mentioned in Section IV-A1, i.e., the traditional neural network trained on samples without rotations, the traditional neural network trained on samples with y-axis rotation augmentation, and the corresponding REQNN trained on samples without rotations. We computed the stability on the testing sets with arbitrarily rotated samples. i.e., given each test point cloud x, we measured the stability under ten different rotation operations $\{\theta(x)\}$ as introduced in Section IV-A1. Table XII shows that input attributions of each REQNN kept almost unchanged, no matter how the point cloud was rotated. The slight change was caused by the systematic errors of the computation. In comparison, input attributions of traditional neural networks were sensitive to rotations.

C. Adversarial Robustness to Rotation Attacks

The previous study [60] proved that classic neural networks for 3D point cloud processing were vulnerable to rotation attacks. I.e., even without perturbing the input point cloud, people could successfully attack a neural network by simply rotating the input. However, REQNNs were supposed to be robust to rotation attacks. Therefore, we used the black-box rotation attack method in [60] to evaluate the adversarial robustness to rotation attacks of REQNNs. The objective of rotation attacks was given as follows.

$$\min_{\theta} p(y = y^{\text{truth}} | \theta(x)), \tag{15}$$

where $p(y = y^{\text{truth}}|\theta(x))$ denoted the probability of the ground-truth category given a rotated sample $\theta(x)$. If $p(y = y^{\text{truth}}|\theta(x))$ was still the largest probability among all categories when reaching the stop condition, then the attack on x was considered failed. Then, the adversarial robustness to rotation attacks was defined as the failure rate of the attacking.

We compared failure rates of attacking between REQNNs and two types of baseline neural networks, i.e., the baseline trained on samples without rotations and the baseline trained on samples with y-axis rotation augmentation. Table XIII shows that it was difficult to attack REQNNs using rotations, while it was easy to attack traditional neural networks.

TABLE XIII
ADVERSARIAL ROBUSTNESS TO ROTATION ATTACKS, I.E., THE FAILURE RATE OF THE ROTATION ATTACK, OF DIFFERENT DNNS

Architecture	ModelNet40 dataset			3D MNIST dataset			ShapeNet dataset		
	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations	Ori. DNN trained w/o rotations	Ori. DNN trained w/ rotations	REQNN trained w/o rotations
PointNet++[5]	0.034	0.050	**1.0**	0.044	0.051	**0.987**	0.031	0.038	**0.999**
DGCNN[6]	0.027	0.031	**0.999**	0.056	0.019	**0.972**	0.012	0.028	**1.0**
PointConv	0.028	0.034	**1.0**	0	0.029	**0.971**	0.007	0.007	**1.0**

A high failure rate indicated that the DNN was robust to rotation attacks. Failure rates of REQNNs were 1.0 or very close to 1.0 (more than 0.97). Sometimes, the failure rate was not exactly 1.0, which was caused by the accumulation of tiny systematic errors of computation in the forward propagation process. In comparison, traditional DNNs could be easily attacked by rotations.

V. CONCLUSION

In this paper, we have proposed a set of generic rules to revise layerwise operations in various neural networks for 3D point cloud processing to construct REQNNs. The revised layerwise operations have been proved to make the REQNN satisfy four properties, including the rotation-equivariance of features, the rotation-equivariance of feature gradients, the rotation-invariance of the training, and the permutation-invariance of features. Experiments on various tasks have demonstrated that REQNNs exhibit superior rotation robustness than traditional neural networks.

REFERENCES

[1] 2016. [Online]. Available: https://www.kaggle.com/daavoo/3d-mnist/version/13

[2] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1120–1128.

[3] J. Castro, D. Gómez, and J. Tejada, "Polynomial calculation of the shapley value based on sampling," *Comput. Operations Res.*, vol. 36, no. 5, pp. 1726–1730, 2009.

[4] A. X. Chang et al., "ShapeNet: An information-rich 3D model repository," 2015, *arXiv:1512.03012*.

[5] 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 *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4994–5002.

[6] H. Chen, S. Liu, W. Chen, H. Li, and R. Hill, "Equivariant point network for 3D point cloud analysis," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 14514–14523.

[7] Y. Chen, B. Fernando, H. Bilen, M. Nießner, and E. Gavves, "3D equivariant graph implicit functions," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2022, pp. 485–502.

[8] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, "Spherical CNNs," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–15.

[9] T. S. Cohen and M. Welling, "Steerable CNNs," 2016, *arXiv:1612.08498*.

[10] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves, "Associative long short-term memory," 2016, *arXiv:1602.03032*.

[11] C. Deng, O. Litany, Y. Duan, A. Poulenard, A. Tagliasacchi, and L. J. Guibas, "Vector neurons: A general framework for SO(3)-equivariant networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 12200–12209.

[12] H. Deng, T. Birdal, and S. Ilic, "PPF-FoldNet: Unsupervised learning of rotation invariant 3D local descriptors," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 602–618.

[13] C. Esteves, C. Allen-Blanchette, A. Makadia, and K. Daniilidis, "Learning SO(3) equivariant representations with spherical CNNs," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 52–68.

[14] C. J. Gaudet and A. S. Maida, "Deep quaternion networks," in *Proc. Int. Joint Conf. Neural Netw.*, 2018, pp. 1–8.

[15] M. Grabisch and M. Roubens, "An axiomatic approach to the concept of interaction among players in cooperative games," *Int. J. Game Theory*, vol. 28, no. 4, pp. 547–565, 1999.

[16] N. Guberman, "On complex valued convolutional neural networks," 2016, *arXiv:1602.09046*.

[17] W. R. Hamilton, "XI. On quaternions; or on a new system of imaginaries in algebra," *London, Edinburgh, Dublin Philos. Mag. J. Sci.*, vol. 33, no. 219, pp. 58–60, 1848.

[18] M. Jaderberg et al., "Spatial transformer networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2017–2025.

[19] M. Jiang, Y. Wu, T. Zhao, Z. Zhao, and C. Lu, "PointSIFT: A SIFT-like network module for 3D point cloud semantic segmentation," 2018, *arXiv:1807.00652*.

[20] A. Kendall, M. Grimes, and R. Cipolla, "PoseNet: A convolutional network for real-time 6-DOF camera relocalization," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 2938–2946.

[21] R. Klokov and V. Lempitsky, "Escape from cells: Deep KD-networks for the recognition of 3D point cloud models," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 863–872.

[22] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4558–4567.

[23] X. Li, R. Li, G. Chen, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "A rotation-invariant framework for deep point cloud analysis," *IEEE Trans. Vis. Comput. Graph.*, vol. 28, no. 12, pp. 4503–4514, Dec. 2022.

[24] X. Liu, Z. Han, Y.-S. Liu, and M. Zwicker, "Point2Sequence: Learning the shape representation of 3D point clouds with an attention-based sequence to sequence network," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 8778–8785.

[25] Y. Liu, B. Fan, S. Xiang, and C. Pan, "Relation-shape convolutional neural network for point cloud analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 8895–8904.

[26] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4765–4774.

[27] J. Mao, X. Wang, and H. Li, "Interpolated convolutional networks for 3D point cloud understanding," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1578–1587.

[28] T. Parcollet et al., "Quaternion convolutional neural networks for end-to-end automatic speech recognition," in *Proc. 19th Annu. Conf. Int. Speech Commun. Assoc.*, 2018, pp. 22–26.

[29] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 652–660.

[30] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5099–5108.

[31] S. Qin, X. Zhang, H. Xu, and Y. Xu, "Fast quaternion product units for learning disentangled representations in SO(3)," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 4, pp. 4504–4520, Apr. 2023.

[32] Y. Rao, J. Lu, and J. Zhou, "Spherical fractal convolutional neural networks for point cloud recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 452–460.

[33] L. S. Shapley, "A value for n-person games," *Contributions Theory Games*, vol. 2, no. 28, pp. 307–317, 1953.

[34] W. Shen, Q. Ren, D. Liu, and Q. Zhang, "Interpreting representation quality of DNNs for 3D point cloud processing," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, 2021, pp. 8857–8870.

[35] W. Shen, B. Zhang, S. Huang, Z. Wei, and Q. Zhang, "3D-rotation-equivariant quaternion neural networks," in *Proc. 16th Eur. Conf. Comput. Vis.*, Springer, 2020, pp. 531–547.

[36] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining point cloud local structures by kernel correlation and graph pooling," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4548–4557.

[37] M. D. Shuster et al., "A survey of attitude representations," *Navigation*, vol. 8, no. 9, pp. 439–517, 1993.

[38] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3693–3702.

[39] M. Sundararajan and A. Najmi, "The many shapley values for model explanation," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9269–9278.

[40] H. Thomas, "Rotation-invariant point convolution with multiple equivariant alignments," in *Proc. Int. Conf. 3D Vis.*, 2020, pp. 504–513.

[41] N. Thomas et al., "Tensor field networks: Rotation-and translation-equivariant neural networks for 3D point clouds," 2018, *arXiv: 1802.08219*.
[42] C. Trabelsi et al., "Deep complex networks," 2017, *arXiv: 1705.09792*.
[43] D. A. Van Dyk and X.-L. Meng, "The art of data augmentation," *J. Comput. Graphical Statist.*, vol. 10, no. 1, pp. 1–50, 2001.
[44] H. Wang et al., "RoReg: Pairwise point cloud registration with oriented descriptors and local rotations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 8, pp. 10376–10393, Aug. 2023.
[45] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," 2018, *arXiv: 1801.07829*.
[46] E. W. Weisstein, "Euler Angles," 2005. [Online]. Available: http://mathworld.wolfram.com/EulerAngles.html
[47] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas, "Full-capacity unitary recurrent neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4880–4888.
[48] M. Wolter and A. Yao, "Complex gated recurrent neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 10536–10546.
[49] W. Wu, Z. Qi, and L. Fuxin, "PointConv: Deep convolutional networks on 3D point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9621–9630.
[50] Z. Wu et al., "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1912–1920.
[51] L. Xiang, H. Ma, H. Zhang, Y. Zhang, and Q. Zhang, "Complex-valued neural networks for privacy protection," in *Proc. Int. Conf. Learn. Representations*, 2020.
[52] J. Xu, X. Tang, Y. Zhu, J. Sun, and S. Pu, "SGMNet: Learning rotation-invariant point cloud representations via sorted Gram matrix," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 10468–10477.
[53] Y. You et al., "PRIN: Pointwise rotation-invariant network," 2018, *arXiv: 1811.09361*.
[54] X. Zhang, S. Qin, Y. Xu, and H. Xu, "Quaternion product units for deep learning on 3D rotation groups," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 7304–7313.
[55] Y. Zhang, Z. Lu, J.-H. Xue, and Q. Liao, "A new rotation-invariant deep network for 3D object recognition," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2019, pp. 1606–1611.
[56] Z. Zhang, B.-S. Hua, D. W. Rosen, and S.-K. Yeung, "Rotation invariant convolutions for 3D point clouds deep learning," in *Proc. Int. Conf. 3D Vis.*, 2019, pp. 204–213.
[57] C. Zhao, J. Yang, X. Xiong, A. Zhu, Z. Cao, and X. Li, "Rotation invariant point cloud classification: Where local geometry meets global topology," 2019, *arXiv: 1911.00195*.
[58] H. Zhao, L. Jiang, C.-W. Fu, and J. Jia, "PointWeb: Enhancing local neighborhood features for point cloud processing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 5565–5573.
[59] Y. Zhao, T. Birdal, J. E. Lenssen, E. Menegatti, L. Guibas, and F. Tombari, "Quaternion equivariant capsule networks for 3D point clouds," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2020, pp. 1–19.
[60] Y. Zhao, Y. Wu, C. Chen, and A. Lim, "On isometry robustness of deep 3D point cloud models under adversarial attacks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1201–1210.
[61] T. Zheng, C. Chen, J. Yuan, B. Li, and K. Ren, "Pointcloud saliency maps," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1598–1606.
[62] X. Zhu, Y. Xu, H. Xu, and C. Chen, "Quaternion convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 631–647.

Wen Shen received the PhD degree from Tongji University, in 2022. She is a post-doctoral researcher with Shanghai Jiao Tong University, China. Her research interests are mainly machine learning and computer vision. In recent years, she has made many influential research in explainable AI (XAI). She has published many papers in international conferences, such as ICML2023, NeurIPS 2021, CVPR 2021, IJCAI 2021, etc. She was the speaker of the tutorial on XAI at IJCAI 2021.

Zhihua Wei is a professor with Tongji University. Her research interests include machine learning and data mining. She has published more than 30 papers in international journals and conferences. She has finished three programs of Chinese National Science Foundation and more than twenty cooperation programs from industry field.

Qihan Ren is currently working toward the undergraduate degree with Shanghai Jiao Tong University, China. His research interests include explainable AI (XAI), computer vision, and machine learning. He has published papers in international conferences, such as NeurIPS 2021 and ICLR 2022.

Binbin Zhang received the graduate degree from Tongji University. His research interests include computer vision and machine learning.

Shikun Huang received the graduate degree from Tongji University. His research interests include computer vision and machine learning.

Jiaqi Fan is currently working toward the master degree with Tongji University. Her research interests include computer vision and machine learning.

Quanshi Zhang (Member, IEEE) received the PhD degree from the University of Tokyo, in 2014. He is an associate professor with Shanghai Jiao Tong University, China. From 2014 to 2018, he was a post-doctoral researcher with the University of California, Los Angeles. His research interests are mainly machine learning and computer vision. In particular, he has made influential research in explainable AI (XAI). He was the co-chairs of the workshops towards XAI in ICML 2021, AAAI 2019, and CVPR 2019. He is the speaker of the tutorials on XAI at IJCAI 2020 and IJCAI 2021. He won the ACM China Rising Star Award at ACM TURC 2021.