3D-Rotation-Equivariant Quaternion Neural Networks

Binbin Zhang[2*] Wen Shen[2*] Shikun Huang[2*] Zhihua Wei[2] Quanshi Zhang[1]
[1]Shanghai Jiao Tong University [2]Tongji University

Abstract

This paper proposes a set of rules to revise various neural networks for 3D point cloud processing to rotation-equivariant quaternion neural networks (REQNNs). We find that when a neural network uses quaternion features under certain conditions, the network feature naturally has the rotation-equivariance property. Rotation equivariance means that applying a specific rotation transformation to the input point cloud is equivalent to applying the same rotation transformation to all intermediate-layer quaternion features. Besides, the REQNN also ensures that the intermediate-layer features are invariant to the permutation of input points. Compared with the original neural network, the REQNN exhibits higher rotation robustness.

1. Introduction

3D point cloud processing receives increasing attention in recent years. Unlike images with rich color information, 3D point clouds usually mainly use spatial contexts for feature extraction. Therefore, the rotation is not supposed to have essential impacts on various tasks, such as 3D shape classification and reconstruction. Besides, reordering input points should not have crucial effects on these tasks as well, which is termed permutation-invariance property.

In this study, we focus on the problem of whether we can learn neural networks for 3D point cloud processing with rotation equivariance and permutation invariance.

• **Rotation equivariance:** Rotation equivariance has been discussed in recent research [7]. In this study, we define rotation equivariance for neural networks as follows. If an input point cloud is rotated by a specific angle, then the feature generated by the network is equivalent to applying the transformation *w.r.t.* the same rotation to the feature of the original point cloud (Figure 1). In this way, we can use the feature of a specific point cloud to synthesize features of the same point cloud with different orientations. Specifically, we can apply the transformation (oriented to the target rotation) to the current feature to synthesize the target feature.

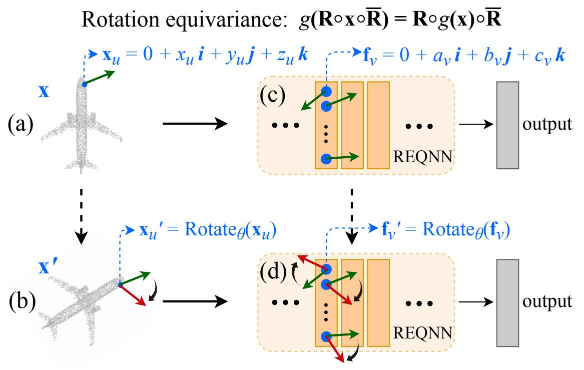
Rotation equivariance: $g(\mathbf{R}\circ\mathbf{x}\circ\overline{\mathbf{R}}) = \mathbf{R}\circ g(\mathbf{x})\circ\overline{\mathbf{R}}$
$\mathbf{x}_u = 0 + x_u\boldsymbol{i} + y_u\boldsymbol{j} + z_u\boldsymbol{k}$
$\mathbf{f}_v = 0 + a_v\boldsymbol{i} + b_v\boldsymbol{j} + c_v\boldsymbol{k}$
$\mathbf{x}$
(a)
(c)
REQNN
output
$\mathbf{x}_u' = \text{Rotate}_\theta(\mathbf{x}_u)$
$\mathbf{f}_v' = \text{Rotate}_\theta(\mathbf{f}_v)$
$\mathbf{x}'$
(b)
(d)
REQNN
output

Figure 1. Quaternion features in a neural network naturally satisfy rotation equivariance under certain conditions in the scenario of 3D point cloud processing. Therefore, in this study, we propose a set of generic rules to revise most existing neural networks to rotation-equivariant quaternion neural networks (REQNNs), where each layerwise operation is adapted for quaternion features. The proposed rules can be broadly applied to various neural networks learned for different tasks, such as 3D shape classification and reconstruction. In the REQNN, both input point clouds and intermediate-layer features are represented by quaternion features (in blue). In this paper, the rotation equivariance is defined as follows. When we rotate the input point cloud $\mathbf{x}$ (a) with a specific angle (*e.g.* 60°) to obtain the same point cloud $\mathbf{x}'$ (b) with a different orientation, then the intermediate-layer feature (d) generated by the REQNN is equivalent to applying the transformation *w.r.t.* the same rotation to the feature (c) of the original point cloud. *I.e.* the rotation equivariance is defined as $g(\text{Rotate}_\theta(\mathbf{x})) = \text{Rotate}_\theta(g(\mathbf{x}))$. The REQNN exhibits significantly higher rotation robustness than traditional neural networks.

• **Permutation invariance:** Permutation invariance measures whether intermediate-layer features essentially keep unchanged when we reorder input 3D points.

Fortunately, we find that quaternion features in a neural network naturally satisfy the rotation-equivariance property under certain conditions (details will be introduced later). Therefore, we propose a set of rules to revise most existing neural networks to rotation-equivariant quaternion neural networks (REQNNs). Given a specific neural network for 3D point cloud processing (*e.g.* PointNet [21], Point-

*Equal contribution. Quanshi Zhang zqs1022@sjtu.edu.cn and Zhihua Wei zhihua_wei@tongji.edu.cn are corresponding authors. This work was done when Shikun Huang, Wen Shen, and Binbin Zhang were research interns at Quanshi Zhang's lab.

Net++ [22], DGCNN [32], PointConv [35], etc., which are learned for various tasks, such as 3D shape classification and reconstruction), our rules can help revise the network to a REQNN with properties of rotation equivariance and permutation invariance.

To revise a neural network to a REQNN with rotation equivariance, we transform both the input and intermediate-layer features of the original neural network into quaternion features (*i.e.* vectors/matrices/tensors, in which each element is a quaternion). A quaternion is a hyper-complex number with three imaginary parts ($\boldsymbol{i}, \boldsymbol{j}$, and $\boldsymbol{k}$) [11]. 3D rotations can be represented using quaternions. *I.e.* rotating a quaternion $\mathbf{q} \in \mathbb{H}$ with an angle $\theta \in [0, 2\pi)$ around an axis $\mathbf{o} = 0 + o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k} \in \mathbb{H}$ ($o_1, o_2, o_3 \in \mathbb{R}$) can be represented as $\mathbf{R}\mathbf{q}\overline{\mathbf{R}}$, where $\mathbf{R} = e^{\mathbf{o}\frac{\theta}{2}} \in \mathbb{H}$; $\overline{\mathbf{R}} \in \mathbb{H}$ is the conjugation of $\mathbf{R}$.

In this way, in a REQNN, the rotation equivariance is defined as follows. When we apply a specific rotation to the input $\mathbf{x} \in \mathbb{H}^n$, *i.e.* $\mathbf{x}'{=}\mathbf{R}{\circ}\mathbf{x}{\circ}\overline{\mathbf{R}}$ ($\circ$ denotes the element-wise multiplication), the network will generate an intermediate-layer quaternion feature $g(\mathbf{x}') \in \mathbb{H}^d$. The rotation equivariance ensures that $g(\mathbf{x}') = \mathbf{R} \circ g(\mathbf{x}) \circ \overline{\mathbf{R}}$. Note that the input and the feature here can be vectors/matrices/tensors, in which each element is a quaternion.

Therefore, we revise a number of layerwise operations in the original neural network to make them rotation-equivariant, such as the convolution operation, the ReLU operation, the batch-normalization operation, the max-pooling operation, the 3D coordinates weighting [35] operation, etc., in order to construct a REQNN with the rotation-equivariance property.

Note that most tasks, such as the shape classification, require outputs composed of real numbers. However, the REQNN's features consist of quaternions. Therefore, for real applications, we revise quaternion features of a specific high layer of the REQNN into ordinary features, in which each element is a real number. We transform quaternion features into real numbers by using the square of the norm of each quaternion element in the feature to replace the corresponding feature element. We put the revised real-valued features into the last few layers to generate real-valued outputs. Such revision ensures that the last few layerwise operations are rotation invariant. We will introduce this revision in Section 3.4.

Besides the rotation-equivariance property, the REQNN is also supposed to have the permutation-invariance property. In a neural network, the permutation invariance is usually regarded as follows. When we reorder 3D points in the input point cloud $\mathbf{x}$ to obtain the same point cloud $\mathbf{x}'$ with a different order, the network will generate the same feature, *i.e.* $g(\mathbf{x}') = g(\mathbf{x})$.

Therefore, we revise a few operations in the original neural network to be permutation invariant, such as the farthest point sampling [22], the ball-query-search-based grouping [22], etc., in order to ensure the permutation-invariance property of the REQNN.

Previous methods [28, 2] developed specific architectures to achieve rotation equivariance and permutation invariance. Our previous work [37] used complex-valued rotation-equivariance neural networks for privacy protection. In comparison, in this study, we do not limit our attention to a specific architecture. Experiments on various neural networks for different tasks in 3D point cloud processing have proved that REQNNs exhibit superior performance than the original networks in terms of rotation robustness.

Contributions of our study are summarized as follows. We propose a set of generic rules to revise various neural networks to REQNNs with both rotation equivariance and permutation invariance. The proposed rules can be broadly applied to different neural networks learned for different tasks, such as 3D shape classification and point cloud reconstruction. Experiments demonstrate the effectiveness of our method that REQNNs exhibit higher rotation robustness than traditional neural networks.

2. Related work

Deep learning for 3D point cloud processing: Recently, a series of studies have focused on deep neural networks (DNNs) for 3D point cloud processing and have achieved superior performance in various 3D tasks [21, 27, 26, 39, 31, 25]. As a pioneer of using DNNs for 3D point cloud processing, PointNet [21] aggregated all individual point features into a global feature using a max-pooling operation. In order to further extract contextual information of 3D point clouds, existing studies have made lots of efforts. PointNet++ [22] hierarchically used PointNet as a local descriptor. KC-Net [24] proposed kernel correlation to measure the similarity between two point sets, so as to represent local geometric structures around each point. PointSIFT [14] proposed a SIFT-like operation to encode contextual information of different orientations for each point. Point2Sequence [19] employed an RNN-based encoder-decoder structure to capture correlations between different areas in a local region by aggregating multi-scale areas of each local region with attention.

Unlike images, 3D point clouds cannot be processed by traditional convolution operators. To address this problem, Kd-network [16] built a kd-tree on subdivisions of the point cloud, and used such kd-tree structure to mimics the convolution operator to extract and aggregate features according to the subdivisions. PointCNN [18] proposed an $\mathcal{X}$-Conv operator to aggregate features from neighborhoods into fewer representative points. Pointwise CNN [12] binned nearest neighbors into kernel cells of each point and convolved them with kernel weights. PointConv [35] treated convolution kernels as nonlinear functions that were learned from local coordinates of 3D points and their densities, respectively. Besides, some studies introduced graph convolutional neural networks for the extraction of geodesic information [26, 32]. Some studies focused on the use of spatial relations between neighboring points [20, 41]. In this study, we aim to learn DNNs with properties of rotation equivariance and permutation invariance.

3D rotation robustness: The most widely used method to improve the rotation robustness was data augmentation [30]. However, data augmentation significantly boosted the computational cost. Spatial Transformer Networks (STNs) [13] allowed spatial manipulations of data and features within the network, which improved the rotation robustness.

Some studies went beyond rotation robustness and focused on rotation invariance. The rotation-invariance property means that the output always keeps unchanged when we rotate the input. One intuitive way to achieve rotation invariance was to project 3D points onto a sphere [38, 23, 40] and constructed spherical CNNs [6] to extract rotation-invariant features. ClusterNet [5] proposed a rotation-invariant representation that discarded orientation information of input point clouds.

However, such rotation-invariant methods directly discarded rotation information, so the rotation equivariance is proposed as a more promising property of feature representations. Rotation-equivariant methods both encode rotation information and disentangle rotation-independent information from the point cloud. To the best of our knowledge, there were very few studies in this direction. Previous studies developed specific network architectures [2] or used specialized group laws [28] to achieve rotation equivariance. In comparison, we aim to propose a set of generic rules to revise most existing neural networks to achieve the rotation-equivariance property.

Complex and quaternion networks: Recently, besides neural networks using real-valued features, neural networks using complex-valued features or quaternion-valued [11] features have been developed [3, 33, 8, 34, 10, 29, 37, 9, 15, 42]. In this study, we use quaternions to represent intermediate-layer features and 3D rotations to achieve 3D rotation-equivariance property.

3. Approach

3.1. Quaternion network

Quaternion: A quaternion [11] $\mathbf{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k} \in \mathbb{H}$ is a hyper-complex number which consists of a real part (q_0) and three imaginary parts ($q_1\boldsymbol{i}, q_2\boldsymbol{j}, q_3\boldsymbol{k}$), where $q_0, q_1, q_2, q_3 \in \mathbb{R}$; $\mathbb{H}$ denotes the algebra of quaternions. When the real part of $\mathbf{q}$ is 0, $\mathbf{q}$ is a **pure quaternion**. When the norm of a quaternion $\|\mathbf{q}\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$, $\mathbf{q}$ is a **unit quaternion**. The **conjugation** of $\mathbf{q}$ is $\overline{\mathbf{q}} = q_0 - q_1\boldsymbol{i} - q_2\boldsymbol{j} - q_3\boldsymbol{k}$.

The products of basis elements $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$ are defined by $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1$ 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 multiplication of two quaternions is non-commutative, *i.e.* $\boldsymbol{ij} \neq \boldsymbol{ji}, \boldsymbol{jk} \neq \boldsymbol{kj}$, and $\boldsymbol{ki} \neq \boldsymbol{ik}$.

Each quaternion has a **polar decomposition**. In this study, we only focus on the polar decomposition of a unit quaternion in the form of $\mathbf{q} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k})$, $\sqrt{o_1^2 + o_2^2 + o_3^2} = 1$. The polar decomposition of such a unit quaternion is $\mathbf{q} = e^{\mathbf{o}\frac{\theta}{2}}$, where $\mathbf{o} = o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k}$. As aforementioned, multiplication of two quaternions is non-commutative, therefore, $e^{\mathbf{o}\frac{\theta}{2}}\mathbf{p}e^{-\mathbf{o}\frac{\theta}{2}} \neq \mathbf{p}$. Please see the supplementary material for more details about this property.

For a traditional neural network, inputs, features, and parameters are vectors/matrices/tensors, in which each element is a real number. However, in a REQNN, inputs and features are vectors/matrices/tensors composed of quaternions; parameters are still vectors/matrices/tensors composed of real numbers.

Specifically, in a REQNN, each u-th point ($[x_u, y_u, z_u]^\top \in \mathbb{R}^3$) in a 3D point cloud is represented by a pure quaternion $\mathbf{x}_u = 0 + x_u\boldsymbol{i} + y_u\boldsymbol{j} + z_u\boldsymbol{k}$. Each v-th element of the intermediate-layer feature is also represented by a pure quaternion $\mathbf{f}_v = 0 + a_v\boldsymbol{i} + b_v\boldsymbol{j} + c_v\boldsymbol{k}$, where $a_v, b_v, c_v \in \mathbb{R}$.

Each element of a feature, $\mathbf{f}_v = 0 + a_v\boldsymbol{i} + b_v\boldsymbol{j} + c_v\boldsymbol{k}$, can be considered to have an orientation, *i.e.* $[a_v, b_v, c_v]^\top$. In this way, 3D rotations can be represented using quaternions. Suppose we rotate $\mathbf{f}_v$ around an axis $\mathbf{o} = 0 + o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k}$ (where $o_1, o_2, o_3 \in \mathbb{R}$, $\|\mathbf{o}\| = 1$) with an angle $\theta \in [0, 2\pi)$ to get $\mathbf{f}'_v$. Such a rotation can be represented using a unit quaternion $\mathbf{R} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k}) = e^{\mathbf{o}\frac{\theta}{2}}$ and it conjugation $\overline{\mathbf{R}}$ as follows.

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

Note that $\mathbf{f}'_v = \mathbf{R}\mathbf{f}_v\overline{\mathbf{R}} \neq \mathbf{f}_v$.

To ensure that all quaternion features are rotation equivariant, all imaginary parts (*i.e.* $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$) of a quaternion element share the same **real-valued** parameters w. Take the convolution operation $\otimes$ as an example, $w \otimes \mathbf{f} = w \otimes (0 + a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k}) = 0 + (w \otimes a)\boldsymbol{i} + (w \otimes b)\boldsymbol{j} + (w \otimes c)\boldsymbol{k}$, where a, b, and c are real-valued tensors of the same size for the convolution operation; but a, b, and c can be real-valued vectors/matrices in other operations.

3.2. Rules to achieve rotation equivariance

In order to achieve the rotation-equivariance property for a REQNN, we should ensure that each layerwise operation

Operation	Rotation equivariance	Permutation invariance
Convolution	×	–
ReLU	×	–
Batch-normalization	×	–
Max-pooling	×	–
Dropout	×	–
Farthest point sampling	✓	×
Grouping (k-NN)	✓	✓
Grouping (ball query) [22]	✓	×
Density estimation [35]	✓	✓
3D coordinates weighting [35]	×	✓
Graph construction [32]	✓	✓

Table 1. Rotation-equivariance and permutation-invariance properties of layerwise operations in the original neural network. "×" denotes that the operation does not have the property, "✓" denotes that the operation naturally has the property, and "–" denotes that the layerwise operation is naturally unrelated to the property (which will be discussed in the last paragraph of Section 3.3). Please see Section 3.2 and Section 3.3 for rules of revising layerwise operations to be rotation-equivariant and permutation-invariant, respectively.

of the REQNN has the rotation-equivariance property. In this way, due to the transitivity of the rotation equivariance, the layerwise rotation equivariance ensures that the output of a REQNN is rotation equivariant. Please see [17] or the supplementary material for the proof of transitivity of the rotation equivariance.In a REQNN, the rotation equivariance is defined as follows. Let $\mathbf{x} \in \mathbb{H}^n$ and $\mathbf{y} = \mathbf{\Phi}(\mathbf{x}) \in \mathbb{H}^C$ denote the input and the output of the REQNN, respectively. Note that outputs for most tasks are traditional vectors/matrices/tensors, in which each element is a real number. In this way, we learn rotation-equivariant quaternion features in most layers, and then transform these features into ordinary real-valued features in the last few layers (these features are rotation invariant). We will introduce details for such revision in Section 3.4.

For each rotation represented using $\mathbf{R} = e^{\mathbf{o}\frac{\theta}{2}}$ (where $\mathbf{o} = 0 + o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k}$ denotes the rotation axis, $o_1, o_2, o_3 \in \mathbb{R}$, $\|\mathbf{o}\| = 1$; and $\theta \in [0, 2\pi)$ denotes the rotation angle) and its conjugation $\overline{\mathbf{R}}$, the rotation equivariance of a REQNN is defined as follows.

$$\mathbf{\Phi}(\mathbf{x}^{(\theta)}) = \mathbf{R} \circ \mathbf{\Phi}(\mathbf{x}) \circ \overline{\mathbf{R}}, \quad \text{s.t.} \quad \mathbf{x}^{(\theta)} \triangleq \mathbf{R} \circ \mathbf{x} \circ \overline{\mathbf{R}}, \tag{2}$$

where $\circ$ denotes the element-wise multiplication.

To achieve the above rotation equivariance, we must ensure the layerwise rotation equivariance. Let $\mathbf{\Phi}(\mathbf{x}) = \Phi_L(\Phi_{L-1}(\cdots \Phi_1(\mathbf{x})))$ represent the cascaded functions of multiple layers of a neural network, 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. The layerwise rotation equivariance is defined as follows.

$$\Phi_l(\mathbf{f}_{l-1}^{(\theta)}) = \mathbf{R} \circ \Phi_l(\mathbf{f}_{l-1}) \circ \overline{\mathbf{R}}, \ \text{s.t.} \ \mathbf{f}_{l-1}^{(\theta)} \triangleq \mathbf{R} \circ \mathbf{f}_{l-1} \circ \overline{\mathbf{R}}. \tag{3}$$

Equation (3) can ensure the rotation-equivariance property of the REQNN. Please see our supplementary material for the proof.

We propose a set of rules to revise layerwise operations in the original neural network, so as to make them rotation-equivariant, *i.e.* satisfying Equation (3). Table 1 shows the list of layerwise operations in the original neural network with the rotation-equivariance property, and those without the rotation-equivariance property.

Convolution: We revise the operation of the convolution layer, $Conv(\mathbf{f}) = w \otimes \mathbf{f} + b$, to be rotation-equivariant by removing the bias term b. The revised convolution operation, *i.e.* $Conv(\mathbf{f}) = w \otimes \mathbf{f}$, is rotation-equivariant, because $Conv(\mathbf{f}^{(\theta)}) = \mathbf{R} \circ Conv(\mathbf{f}) \circ \overline{\mathbf{R}}$. Please see the supplementary material for the proof.

Although our revision is oriented to all convolution operations, 3D point cloud processing usually uses the specific convolution with 1×1 kernels. Our revision is also compatible with such convolution operation.

ReLU: We revise the ReLU operation as follows to make it rotation-equivariant.

$$ReLU(\mathbf{f}_v) = \frac{\|\mathbf{f}_v\|}{\max\{\|\mathbf{f}_v\|, c\}} \mathbf{f}_v, \tag{4}$$

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, which can be implemented as $c = \frac{1}{d}\sum_{v=1}^{d} \|\mathbf{f}_v\|$. It can be easily proved that such ReLU operation is rotation-equivariant, because $ReLU(\mathbf{f}_v^{(\theta)}) = \mathbf{R} \circ ReLU(\mathbf{f}_v) \circ \overline{\mathbf{R}}$. Please see our supplementary material for the details about the proof.

Batch-normalization: We revise the batch-normalization operation to be rotation-equivariant as follows.

$$norm(\mathbf{f}_v^{(i)}) = \frac{\mathbf{f}_v^{(i)}}{\sqrt{\mathbb{E}_j[\|\mathbf{f}_v^{(j)}\|^2] + \epsilon}}, \tag{5}$$

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 used to avoid dividing by 0. We set $\epsilon = 10^{-5}$ in this study. The revised batch-normalization is rotation-equivariant, because $norm(\mathbf{f}_v^{n(\theta)}) = \mathbf{R} \circ norm(\mathbf{f}_v^n) \circ \overline{\mathbf{R}}$. The detailed proof can be found in our supplementary material.

Max-pooling: We revise the max-pooling operation as follows.

$$maxpool(\mathbf{f}) = \arg\max_{\mathbf{f}_v}\{\|\mathbf{f}_v\|\}, v = 1, \ldots, d. \tag{6}$$

It can be easily proved that the revised max-pooling operation is rotation-equivariant, because $maxpool(\mathbf{f}^{(\theta)}) = \mathbf{R} \circ maxpool(\mathbf{f}) \circ \overline{\mathbf{R}}$. Please see the supplementary material for the detailed proof. Note that for 3D point cloud

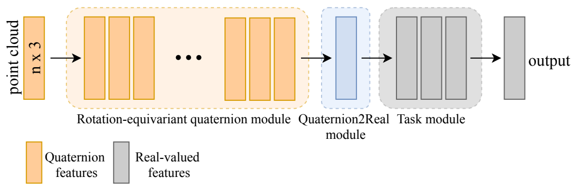

Figure 2. Illustration of the overview architecture of the REQNN.

processing, a special element-wise max-pooling operation designed in [21] is widely used. The revision for this special max-pooling operation can be decomposed to a group of operations as Equation (6) defined. Please see the supplementary material for revision details of this operation.

Dropout: For the dropout operation, we randomly drop out a number of quaternion elements from the feature. For each dropped element, both the real and imaginary parts are set to zero. Such revision naturally satisfies the rotation-equivariance property defined in Equation (3).

3D coordinates weighting: The 3D coordinates weighting operation designed in [35] focuses on the use of 3D coordinates' information to reweight intermediate-layer features. This operation is not rotation-equivariant, because the rotation changes coordinates of points. To make this operation rotation-equivariant, we use the PrincipalComponents Analysis (PCA) to transform 3D points to a new local coordinate system. Specifically, we choose eigenvectors corresponding to the first three principal components as new axes x, y, and z of the new local coordinate system. In this way, the coordinate system rotates together with input points, so the transformed new coordinates are not changed. Note that such local coordinate system is only used for the 3D coordinates weighting operation.

The following five layerwise operations in the original neural network, which are implemented based on distances between points, are naturally rotation-equivariant, including the farthest point sampling [22], the k-NN-search-based grouping [32, 35], the ball-query-search-based grouping [22], the density estimation [35], and the graph construction [32] operations.

3.3. Rules to achieve permutation invariance

As shown in Table 1, the farthest point sampling [22], and the ball-query-search-based grouping [22] are not permutation-invariant. Therefore, we revise these two operations to be permutation-invariant as follows.

Farthest point sampling: The farthest point sampling (FPS) is an operation for selecting a subset of points from the input point cloud, in order to extract local features [22]. Suppose that we aim to select n points from the input point cloud, if $i-1$ points have already been selected, *i.e.* $S_{i-1} = \{x_1, x_2, \ldots, x_{i-1}\}$, then the next selected point x_i is the farthest point from S_{i-1}. The FPS is not permutation-invariant, because the subset selected by this operation depends on which point is selected first. To revise the FPS to be permutation-invariant, we always use the centroid of a point cloud, which is a virtual point, as the first selected point. In this way, the FPS would be permutation-invariant.

Grouping (ball query): The ball-query-search-based grouping is used to find K neighboring points within a radius for each given center point, in order to extract contextual information [22]. This operation is not permutation-invariant, because when there are more than K points within the radius, the top K points will be selected according to the order of points. To revise this operation to be permutation-invariant, we replace the ball query search by k-NN search when there are more points than the required number.

Other operations that implemented based on distances between points are permutation-invariant, because reordering input points has no effects on distances between points, including the k-NN-search-based grouping [32, 35], the density estimation [35], the 3D coordinates weighting [35], and the graph construction [32] operations.

Note that there is no need to discuss the permutation invariance of the convolution, the ReLU, the batch-normalization, the max-pooling, and the dropout operations. It is because the permutation invariance of these operations depends on receptive fields. *I.e.* if the receptive field of each neural unit keeps the same when we reorder input points, then the operation is permutation-invariant. Whereas receptive fields are determined by other operations (*e.g.* the FPS and the grouping).

3.4. Overview architecture of the REQNN

Although using quaternions to represent intermediate-layer features helps achieve the rotation-equivariance property, most existing tasks (*e.g.* the shape classification) require outputs of real numbers. Thus, we need to transform quaternion features into oridinary real-valued features, in which each element is a real number. Note that for the point cloud reconstruction task, features of the entire neural network are quaternions. It is because outputs required by the point cloud reconstruction task are 3D coordinates, which can be represented by quaternions.

Therefore, as shown in Figure 2, the REQNN consists of three modules: (a) rotation-equivariant quaternion module, (b) Quaternion2Real module, and (c) task module.

Rotation-equivariant quaternion module: Except for very few layers on the top of the REQNN, other layers in the REQNN make up the rotation-equivariant quaternion module. This module is used to extract rotation-equivariant quaternion features. We use rules proposed in Section 3.2 to revise layerwise operations in the original neural network to be rotation-equivariant, so as to obtain the rotation-equivariant quaternion module. We also use rules proposed in Section 3.3 to revise these layerwise operations to be permutation invariant.

Layerwise operation	PointNet++ [22]	DGCNN [32]	PointConv [35]	PointNet [21]
Convolution	✓	✓	✓	✓
ReLU	✓	✓	✓	✓
Batch-normalization	✓	✓	✓	✓
Max-pooling	✓	✓		✓
Dropout	✓	✓	✓	✓
Farthest point sampling	✓		✓	
Grouping (k-NN)		✓	✓	
Grouping (ball query) [22]	✓			
Density estimation [35]			✓	
3D coordinates weighting [35]			✓	
Graph construction [32]		✓		

Table 2. Layerwise operations of different neural networks. "✓" denotes that the network contains the layerwise operation.

Quaternion2Real module: The Quaternion2Real module is located after the rotation-equivariant quaternion module. The Quaternion2Real module is used to transform quaternion features into real-valued vectors/matrices/tensors as features. Specifically, we use an element-wise operation to compute the square of the norm of each quaternion element as the real-valued feature element. *I.e.* for 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 $\|\mathbf{f}_v\|^2 = a_v^2 + b_v^2 + c_v^2$ as the corresponding element of the real-valued feature. Note that the transformed features are rotation-invariant.

Task module: The task module is composed of the last few layers of the REQNN. The task module is used to obtain ordinary real-valued outputs, which are required by tasks of shape classification and point cloud reconstruction. As aforementationed, the Quaternion2Real module transforms quaternion features into real-valued vectors/matrices/tensors as features. In this way, the task module (*i.e.* the last few layers) in the REQNN implements various tasks just like traditional neural networks.

The proposed rules can be broadly used to revise various existing neural networks to REQNNs, which are learned for different tasks, including the 3D shape classification and reconstruction. Note that in comparative studies, unnecessarily complex network architectures usually bring in additional uncertainty, which will hamper our experiments from obtaining reliable and rigorous results. Thus, we conduct experiments on simple and classic network architectures.

In this study, we revise the following four neural networks to REQNNs, including the PointNet++ [22], the DGCNN [32], the PointConv [32], and the PointNet [21].

Model 1, PointNet++: As shown in Table 2, the PointNet++ [22] for shape classification includes the following seven types of layerwise operations, *i.e.* the convolution, the ReLU, the batch-normalization, the max-pooling, the dropout, the farthest point sampling, and the ball-query-search-based grouping.

To revise the PointNet++[1] for shape classification to a REQNN, we take the last three fully-connected (FC) layers as the task module and take other layers as the rotation-equivariant quaternion module. We add a Quaternion2Real module between these two modules. We use rules proposed in Section 3.2 to revise four types of layerwise operations to be rotation-equivariant, including the convolution, the ReLU, the batch-normalization, and the max-pooling operations. We also use rules proposed in Section 3.3 to revise farthest point sampling and ball-query-search-based grouping operations in the original PointNet++ to be permutation-invariant.

Model 2, DGCNN: As shown in Table 2, the DGCNN [32] for shape classification contains the following seven types of layerwise operations, *i.e.* the convolution, the ReLU, the batch-normalization, the max-pooling, the dropout, the k-NN-search-based grouping, and the graph construction operations.

To revise the DGCNN for shape classification to a REQNN, we take the last three FC layers as the task module and take other layers as the rotation-equivariant quaternion module. The Quaternion2Real module[2] is added between these two modules. We revise four types of layerwise operations to be rotation-equivariant, including the convolution, the ReLU, the batch-normalization, and the max-pooling operations. All layerwise operations in the original DGCNN are naturally permutation-invariant. Therefore, there is no revision for permutation invariance here.

Model 3, PointConv: As shown in Table 2, the PointConv [35] for shape classification includes eight types of layerwise operations, including the convolution, the ReLU, the batch-normalization, the dropout, the farthest point sampling, the k-NN-search-based grouping, the density estimation, and the 3D coordinates weighting operations.

To revise the PointConv for shape classification to a REQNN, we take 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. We revise the following four types of layerwise operations to be rotation-equivariant, *i.e.* the con-

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

[2]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.

Method	ModelNet40 dataset			3D MNIST dataset		
	Baseline w/o rotations	Baseline w/ rotations	REQNN	Baseline w/o rotations	Baseline w/ rotations	REQNN
PointNet++[1] [22]	22.77[3]	26.37	**63.78**	40.80	46.30	**70.50**
DGCNN[2] [32]	30.47[3]	31.72	**83.02**	42.10	46.70	**83.40**
PointConv [35]	23.70	44.98	**78.73**	41.80	48.10	**79.10**

Table 3. Accuracy of 3D shape classification on the ModelNet40 and the 3D MNIST datasets. "Baseline w/o rotations" indicates the original neural network learned without rotations. "Baseline w/ rotations" indicates the original neural network learned with the z-axis rotations (data augmentation with the z-axis rotations has been widely applied in [22, 32, 35]). "REQNN" indicates the REQNN learned without rotations. Note that the accuracy of shape classification reported in [22, 32, 35] 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.

Method	**NR/NR** (do **not** consider rotation in testing)	**NR/AR** (consider rotation in testing)
PointNet [21]	88.45	12.47
PointNet++ [22]	89.82	21.35[3]
Point2Sequence [19]	92.60	10.53
KD-Network [16]	86.20	8.49
RS-CNN [20]	92.38	22.49
DGCNN [32]	**92.90**	29.74[3]
PRIN [38]	80.13	68.85
QE-Capsule network [2]	74.73	74.07
REQNN (revised from DGCNN[2])	83.02 =	**83.02**

Table 4. Comparisons of 3D shape classification accuracy between different methods on the ModelNet40 dataset. **NR/NR** denotes that neural networks were learned and tested with **N**o **R**otations. **NR/AR** denotes that neural networks 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. Note that the classification accuracy of the REQNN in scenarios of NR/NR and NR/AR was the same due to the rotation-equivariance property of the REQNN.

volution, the ReLU, the batch-normalization, and the 3D coordinates weighting operations. We also revise all farthest point sampling operations in the original PointConv to be permutation-invariant.

Model 4, PointNet: In order to construct a REQNN for shape reconstruction, we slightly revise the architecture of the PointNet [21] for shape classification. The PointNet for shape classification contains the following five types of layerwise operations, *i.e.* the convolution, the ReLU, the batch-normalization, the max-pooling, and the dropout operations. We take all remaing layers in the PointNet as the rotation-equivariant quaternion module except for the max-pooling operation and the Spatial Transformer Network (STN) [13]. The STN discards all spatial information (including the rotation information) of the input point cloud. Therefore, in order to encode 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. We revise the following four types of layerwise operations to be rotation-equivariant, *i.e.* the convolution, the ReLU, the batch-normalization, and the dropout operations.

4. Experiments

Theoretically, our REQNNs naturally satisfy the rotation-equivariance property. Unlike previous neural networks for 3D point cloud processing (*e.g.* PointNet [21], PointNet++ [22], DGCNN [32], PointConv [35], etc.), quaternion features in the REQNN naturally satisfy the rotation-equivariance property, *i.e.* disentangling rotation information away from the point cloud.

Properties of the rotation equivariance and the permutation invariance of REQNNs could be proved theoretically, please see our supplementary material for details. In order to show other advantages of REQNNs, we conducted the following experiments. We revised three widely used neural networks to REQNNs for the shape classification task, including PointNet++ [22], DGCNN [32], and PointConv [35]. We revised the PointNet [21] to a REQNN for the point cloud reconstruction task. For all experiments, we set $c = 1$ in Equation (4) and set $\epsilon = 10^{-5}$ in Equation (5).

3D shape classification: We used the ModelNet40 [36] dataset (in this study, we used corresponding point clouds provided by PointNet [21]) and the 3D MNIST [1] dataset for shape classification. The ModelNet40 dataset consisted of 40 categories; and the 3D MNIST dataset consisted of 10 categories. Each shape consisted of 1024 points. In this experiment, we conducted experiments on three types of baseline neural networks, including (1) the original neural network learned without rotations, (2) the original neural network learned with the z-axis rotations (the z-axis rotations were widely used in [22, 32, 35] for data augmentation), and (3) the REQNN learned without rotations (the REQNN naturally had the rotation-equivariance property, so it did not require any rotation augmentation). All neural networks were tested using a testing set which was generated by arbitrarily rotating each example. We will release this testing set when this paper is accepted.

As shown in Table 3[3], the REQNN always outperformed

[3]The classification accuracy of PointNet++ [22] in the scenario of NR/AR in Table 3 was 22.77% and that in Table 4 was 21.35%. The classification accuracy of DGCNN [32] in the scenario of NR/AR in Table 3 was 30.47% and that in Table 4 was 29.74%. The reason for such slight gaps was that the architecture of PointNet++[1] or DGCNN[2] examined in Table 3 were slightly different from the architecture examined in Table 4. Nevertheless, this did not essentially change our conclusions.

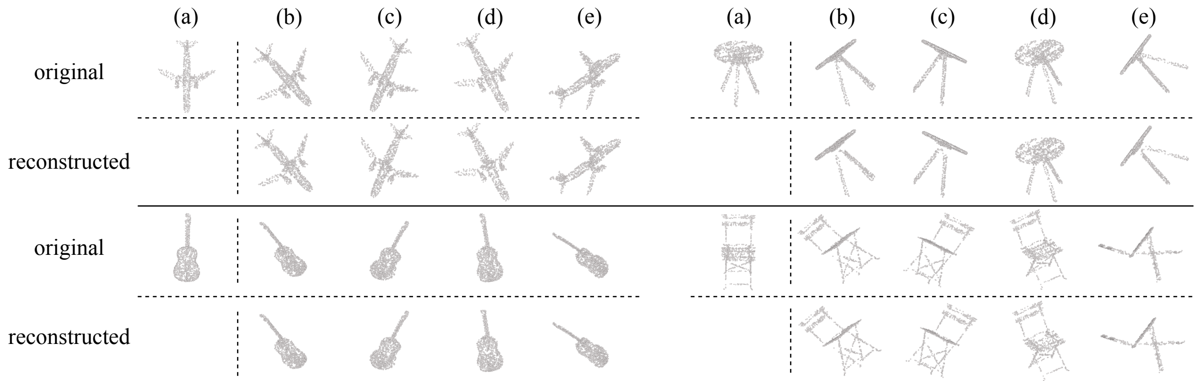

Figure 3. Manual manipulation of intermediate-layer features to control the object rotation in 3D point cloud reconstruction. The experiment was conducted to prove that point clouds reconstructed using the synthesized quaternion features had the same orientations as point clouds generated by directly rotating the original point cloud. Here we displayed results of four random orientations for each point cloud. Point clouds of "original" (b-e) were generated by directly rotating original point clouds ("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. The rotated quaternion features were used to reconstruct point clouds ("reconstructed" (b-e)).

all baseline neural networks learned with or without rotations. We achieved the highest accuracy of 83.02% using the REQNN revised from DGCNN[2]. Baseline neural networks that were learned without rotations exhibited very low accuracy (22.77%-30.47% on the ModelNet40 dataset and 40.80%-42.10% on the 3D MNIST dataset). In comparison, baseline neural networks that were learned with z-axis rotations had little improvement in rotation robustness.

Besides, we compared the REQNN with several state-of-the-art methods for 3D point cloud processing in two scenarios, including (**NR/NR**) neural networks learned with **N**o **R**otations and tested with **N**o **R**otations, and (**NR/AR**) neural networks learned with **N**o **R**otations and tested with **A**rbitrary **R**otations. Note that the classification accuracy of the REQNN in the scenario of NR/NR was the same as that of NR/AR, because the REQNN was rigorously rotation equivariant. The best REQNN in this paper (*i.e.* the REQNN revised from the DGCNN[2]) achieved the highest accuracy of 83.02% in the scenario of NR/AR, which indicated the significantly high rotation robustness of the REQNN. Traditional methods, including PointNet [21], PointNet++ [22], Point2Sequence [19], KD-Network [16], RS-CNN [20], and DGCNN [32], achieved high accuracy in the scenario of NR/NR. However, these methods performed poor in the scenario of NR/AR, because they could not deal with point clouds with unseen orientations. Compared with these methods, PRIN [38] and QE-Capsule network [2] made some progress in handling point clouds with unseen orientations. Our REQNN outperformed them by 14.17% and 8.95%, respectively, in the scenario of NR/AR.

3D point cloud reconstruction: In this experiment, we aimed to prove that we could rotate intermediate-layer quaternion features of the original point cloud to synthesize new point clouds with target orientations. Therefore, we learned a REQNN revised from the PointNet [21] for point cloud reconstruction on the ShapeNet [4] dataset. Each point cloud consisted of 1024 points in our implementation. We took 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 Figure 3, for each given point cloud (Figure 3 ("original" (a))), we directly rotated it with different angles (Figure 3 ("original" (b-e))). For comparison, we rotated the corresponding quaternion feature of the original point cloud with the same angles to synthesize quaternion features. These generated quaternion features were used to reconstruct point clouds (Figure 3 ("reconstructed" (b-e))). We observed that these reconstructed point clouds had the same orientations with those of point clouds generated by directly rotating the original point cloud.

5. Conclusion

In this paper, we have proposed a set of generic rules to revise various neural networks for 3D point cloud processing to REQNNs. We have theoretically proved that the proposed rules can ensure that each layerwise operation in the neural network is rotation equivariant and permutation invariant. Experiments on various tasks have shown the rotation robustness of REQNNs.

We admit that revising a neural network to a REQNN have some negative effects on its representation capacity. Besides, it is challenging to revise all layerwise operations in all neural networks for 3D point cloud processing.

References

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

[2] Anonymous. Quaternion equivariant capsule networks for 3d point clouds. In *Submitted to International Conference on Learning Representations*, 2020. under review.

[3] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

[4] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[5] Chao Chen, Guanbin Li, Ruijia Xu, Tianshui Chen, Meng Wang, and Liang Lin. Clusternet: Deep hierarchical cluster network with rigorously rotation-invariant representation for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4994–5002, 2019.

[6] Taco S. Cohen, Mario Geiger, Jonas Khler, and Max Welling. Spherical CNNs. In *International Conference on Learning Representations*, 2018.

[7] Taco S Cohen and Max Welling. Steerable cnns. *arXiv preprint arXiv:1612.08498*, 2016.

[8] Ivo Danihelka, Greg Wayne, Benigno Uria, Nal Kalchbrenner, and Alex Graves. Associative long short-term memory. *arXiv preprint arXiv:1602.03032*, 2016.

[9] Chase J Gaudet and Anthony S Maida. Deep quaternion networks. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[10] Nitzan Guberman. On complex valued convolutional neural networks. *arXiv preprint arXiv:1602.09046*, 2016.

[11] William Rowan Hamilton. Xi. on quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 33(219):58–60, 1848.

[12] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 984–993, 2018.

[13] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[14] Mingyang Jiang, Yiran Wu, Tianqi Zhao, Zelin Zhao, and Cewu Lu. Pointsift: A sift-like network module for 3d point cloud semantic segmentation. *arXiv preprint arXiv:1807.00652*, 2018.

[15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[16] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 863–872, 2017.

[17] Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. *arXiv preprint arXiv:1802.03690*, 2018.

[18] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pages 820–830, 2018.

[19] Xinhai Liu, Zhizhong Han, Yu-Shen Liu, and Matthias Zwicker. Point2sequence: Learning the shape representation of 3d point clouds with an attention-based sequence to sequence network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8778–8785, 2019.

[20] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019.

[21] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

[22] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[23] Yongming Rao, Jiwen Lu, and Jie Zhou. Spherical fractal convolutional neural networks for point cloud recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 452–460, 2019.

[24] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4548–4557, 2018.

[25] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–779, 2019.

[26] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.

[27] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2530–2539, 2018.

[28] Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

[29] Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and

Christopher J Pal. Deep complex networks. *arXiv preprint arXiv:1705.09792*, 2017.

[30] David A Van Dyk and Xiao-Li Meng. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50, 2001.

[31] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2569–2578, 2018.

[32] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[33] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4880–4888, 2016.

[34] Moritz Wolter and Angela Yao. Complex gated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 10536–10546, 2018.

[35] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

[36] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[37] Liyao Xiang, Haotian Ma, Hao Zhang, Yifan Zhang, and Quanshi Zhang. Complex-valued neural networks for privacy protection. *arXiv preprint arXiv:1901.09546*, 2019.

[38] Yang You, Yujing Lou, Qi Liu, Lizhuang Ma, Weiming Wang, Yuwing Tai, and Cewu Lu. Prin: Pointwise rotation-invariant network. *arXiv preprint arXiv:1811.09361*, 2018.

[39] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2018.

[40] Yachi Zhang, Zongqing Lu, Jing-Hao Xue, and Qingmin Liao. A new rotation-invariant deep network for 3d object recognition. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1606–1611. IEEE, 2019.

[41] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5565–5573, 2019.

[42] Xuanyu Zhu, Yi Xu, Hongteng Xu, and Changjian Chen. Quaternion convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 631–647, 2018.

A. Quaternion operations

Just like complex numbers, a series of quaternion operations can be defined as follows.

Addition: $\mathbf{p}+\mathbf{q}=(p_0+q_0)+(p_1+q_1)\boldsymbol{i}+(p_2+q_2)\boldsymbol{j}+(p_3+q_3)\boldsymbol{k}$.

Scalar multiplication: $\lambda\mathbf{q}=\lambda q_0+\lambda q_1\boldsymbol{i}+\lambda q_2\boldsymbol{j}+\lambda q_3\boldsymbol{k}$.

Element multiplication: $\mathbf{pq}=(p_0q_0-p_1q_1-p_2q_2-p_3q_3)+(p_0q_1+p_1q_0+p_2q_3-p_3q_2)\boldsymbol{i}+(p_0q_2-p_1q_3+p_2q_0+p_3q_1)\boldsymbol{j}+(p_0q_3+p_1q_2-p_2q_1+p_3q_0)\boldsymbol{k}$.

Note that multiplication of two quaternions is non-commutative, because $\mathbf{qp}=(q_0p_0-q_1p_1-q_2p_2-q_3p_3)+(q_0p_1+q_1p_0+q_2p_3-q_3p_2)\boldsymbol{i}+(q_0p_2-q_1p_3+q_2p_0+q_3p_1)\boldsymbol{j}+(q_0p_3+q_1p_2-q_2p_1+q_3p_0)\boldsymbol{k}\neq\mathbf{pq}$.

Norm: $\|\mathbf{q}\|=\sqrt{\mathbf{q}\overline{\mathbf{q}}}=\sqrt{q_0^2+q_1^2+q_2^2+q_3^2}$, where $\overline{\mathbf{q}}=q_0-q_1\boldsymbol{i}-q_2\boldsymbol{j}-q_3\boldsymbol{k}$ is the conjugation of $\mathbf{q}$.

B. Proofs of layerwise rotation equivariance

B.1. Convolution operation

Let $\mathbf{f}=[\mathbf{f}_1,\ldots,\mathbf{f}_d]^\top\in\mathbb{H}^d$ denote the quaternion feature of a point in the point cloud. Note that 3D point cloud processing usually uses the specific convolution with 1×1 kernels. Let us take this specific convolution as the example to prove that the revised convolution operation is rotation equivariant. The revised convolution operation,

$$\begin{aligned}
Conv(\mathbf{f}) &= w\otimes\mathbf{f}\\
&=\begin{bmatrix} w_{11} & \cdots & w_{1d}\\ \vdots & \ddots & \vdots\\ w_{D1} & \cdots & w_{Dd}\end{bmatrix}\otimes\begin{bmatrix}\mathbf{f}_1\\ \vdots\\ \mathbf{f}_d\end{bmatrix}\\
&=\begin{bmatrix}(w_{11}\mathbf{f}_1+\cdots+w_{1d}\mathbf{f}_d)\\ \vdots\\ (w_{D1}\mathbf{f}_1+\cdots+w_{Dd}\mathbf{f}_d)\end{bmatrix},
\end{aligned}\tag{7}$$

is rotation-equivariant, because $Conv(\mathbf{f}^{(\theta)})=\mathbf{R}\circ Conv(\mathbf{f})\circ\overline{\mathbf{R}}$. The proof is given as follows.

$$\begin{aligned}
Conv(\mathbf{f}^{(\theta)}) &= w\otimes(\mathbf{R}\circ\mathbf{f}\circ\overline{\mathbf{R}})\\
&=\begin{bmatrix} w_{11} & \cdots & w_{1d}\\ \vdots & \ddots & \vdots\\ w_{D1} & \cdots & w_{Dd}\end{bmatrix}\otimes(\mathbf{R}\circ\begin{bmatrix}\mathbf{f}_1\\ \vdots\\ \mathbf{f}_d\end{bmatrix}\circ\overline{\mathbf{R}})\\
&=\begin{bmatrix}(w_{11}(\mathbf{R}\mathbf{f}_1\overline{\mathbf{R}})+\cdots+w_{1d}(\mathbf{R}\mathbf{f}_d\overline{\mathbf{R}}))\\ \vdots\\ (w_{D1}(\mathbf{R}\mathbf{f}_1\overline{\mathbf{R}})+\cdots+w_{Dd}(\mathbf{R}\mathbf{f}_d\overline{\mathbf{R}}))\end{bmatrix}\\
&=\begin{bmatrix}(\mathbf{R}(w_{11}\mathbf{f}_1)\overline{\mathbf{R}}+\cdots+\mathbf{R}(w_{1d}\mathbf{f}_d)\overline{\mathbf{R}})\\ \vdots\\ (\mathbf{R}(w_{D1}\mathbf{f}_1)\overline{\mathbf{R}}+\cdots+\mathbf{R}(w_{Dd}\mathbf{f}_d)\overline{\mathbf{R}})\end{bmatrix}\\
&=\mathbf{R}\circ(w\otimes\mathbf{f})\circ\overline{\mathbf{R}}\\
&=\mathbf{R}\circ Conv(\mathbf{f})\circ\overline{\mathbf{R}}.
\end{aligned}\tag{8}$$

B.2. ReLU operation

The revised ReLU operation, $ReLU(\mathbf{f}_v) = \frac{\|\mathbf{f}_v\|}{\max\{\|\mathbf{f}_v\|, c\}}\mathbf{f}_v$, is rotation-equivariant, because $ReLU(\mathbf{f}_v^{(\theta)}) = \mathbf{R} \circ ReLU(\mathbf{f}_v) \circ \overline{\mathbf{R}}$. The proof is given as follows.

$$\begin{aligned} ReLU(\mathbf{f}_v^{(\theta)}) &= \frac{\|\mathbf{R} \circ \mathbf{f}_v \circ \overline{\mathbf{R}}\|}{\max\{\|\mathbf{R} \circ \mathbf{f}_v \circ \overline{\mathbf{R}}\|, c\}}(\mathbf{R} \circ \mathbf{f}_v \circ \overline{\mathbf{R}}) \\ &= \frac{\|\mathbf{f}_v\|}{\max\{\|\mathbf{f}_v\|, c\}}(\mathbf{R} \circ \mathbf{f}_v \circ \overline{\mathbf{R}}) \\ &= \mathbf{R} \circ (\frac{\|\mathbf{f}_v\|}{\max\{\|\mathbf{f}_v\|, c\}}\mathbf{f}_v) \circ \overline{\mathbf{R}} \\ &= \mathbf{R} \circ ReLU(\mathbf{f}_v) \circ \overline{\mathbf{R}}. \end{aligned} \tag{9}$$

Note that $\|\mathbf{R} \circ \mathbf{f}_v \circ \overline{\mathbf{R}}\| = \|\mathbf{f}_v\|$, because rotating a quaternion will not change its norm.

B.3. Batch-normalization

The revised batch-normalization, $norm(\mathbf{f}_v^n) = \frac{\mathbf{f}_v^n}{\sqrt{\mathbb{E}_n[\|\mathbf{f}_v^n\|^2]+\epsilon}}$, is rotation-equivariant, because $norm(\mathbf{f}_v^{n(\theta)}) = \mathbf{R} \circ norm(\mathbf{f}_v^n) \circ \overline{\mathbf{R}}$. The proof is given as follows.

$$\begin{aligned} norm(\mathbf{f}_v^{n(\theta)}) &= \frac{\mathbf{R} \circ \mathbf{f}_v^n \circ \overline{\mathbf{R}}}{\sqrt{\mathbb{E}_n[\|\mathbf{R} \circ \mathbf{f}_v^n \circ \overline{\mathbf{R}}\|^2] + \epsilon}} \\ &= \frac{\mathbf{R} \circ \mathbf{f}_v^n \circ \overline{\mathbf{R}}}{\sqrt{\mathbb{E}_n[\|\mathbf{f}_v^n\|^2] + \epsilon}} \\ &= \mathbf{R} \circ norm(\mathbf{f}_v^n) \circ \overline{\mathbf{R}}. \end{aligned} \tag{10}$$

B.4. Max-pooling operation

The revised max-pooling operation, $maxpool(\mathbf{f}) = \arg\max_{\mathbf{f}_v}\{\|\mathbf{f}_v\|\}, i = 1, \ldots, d$, is rotation-equivariant, because $maxpool(\mathbf{f}^{(\theta)}) = \mathbf{R} \circ maxpool(\mathbf{f}) \circ \overline{\mathbf{R}}$. The proof is given as follows.

$$\begin{aligned} maxpool(\mathbf{f}^{(\theta)}) &= \arg\max_{\mathbf{R} \circ \mathbf{f}_v \circ \overline{\mathbf{R}}}\{\|\mathbf{R} \circ \mathbf{f}_v \circ \overline{\mathbf{R}}\|\}, i = 1, \ldots, d \\ &= \arg\max_{\mathbf{R} \circ \mathbf{f}_v \circ \overline{\mathbf{R}}}\{\|\mathbf{f}_v\|\}, i = 1, \ldots, d \\ &= \mathbf{R} \circ (\arg\max_{\mathbf{f}_v}\{\|\mathbf{f}_v\|\}) \circ \overline{\mathbf{R}}, i = 1, \ldots, d \\ &= \mathbf{R} \circ maxpool(\mathbf{f}) \circ \overline{\mathbf{R}}. \end{aligned} \tag{11}$$

C. Element-wise max-pooling operation

In point cloud processing, a special element-wise max-pooling operation is widely used for aggregating a set of neighboring points' features into a local feature. Let $f \in \mathbb{R}^{D \times K}$ denote the features of K neighboring points. Each element of f, *i.e.* $f_k \in \mathbb{R}^D$, denotes the feature of a specific neighboring point. Let $f^{\text{upper}} \in \mathbb{R}^D$ denote the output local feature. The element-wise max-pooling operation is formulated as follows.

$$\begin{aligned} \mathbf{MAX}(f) = \mathbf{MAX} &\begin{bmatrix} f_{11} & \cdots & f_{1K} \\ \vdots & \ddots & \vdots \\ f_{D1} & \cdots & f_{DK} \end{bmatrix} \\ &\overset{\textbf{define}}{=\!=\!=} < \max_{k=1,\ldots,K} f_{1k}, \ldots, \max_{k=1,\ldots,K} f_{Dk} >^{\top} \end{aligned} \tag{12}$$

To extend this special max-pooling operation to be suitable for quaternion operations, we replace each max operation $\max_{k=1,\ldots,K} f_{dk}$ in Equation (12) by $\arg\max_{\mathbf{f}_{dk}}\{\|\mathbf{f}_{dk}\|\}, k = 1, \ldots, K$, where $\mathbf{f}_{dk} \in \mathbb{H}$ is a quaternion.

D. Transitivity of the rotation equivariance

Let $\mathbf{x} \in \mathbb{H}^n$ and $\mathbf{y} = \mathbf{\Phi}(\mathbf{x}) = \Phi_L(\Phi_{L-1}(\cdots \Phi_1(\mathbf{x}))) \in \mathbb{H}^C$ denote the input and the output of the REQNN, respectively. Let $\mathbf{f}_l = \Phi_l(\mathbf{f}_{l-1}) \in \mathbb{H}^d$ denote the output of the l-th layer. We prove that the layerwise rotation equivariance can ensure the rotation-equivariance property of the REQNN. *I.e.* if $\Phi_l(\mathbf{f}_{l-1}^{(\theta)}) = \mathbf{R} \circ \Phi_l(\mathbf{f}_{l-1}) \circ \overline{\mathbf{R}}$, *s.t.* $\mathbf{f}_{l-1}^{(\theta)} \triangleq \mathbf{R} \circ \mathbf{f}_{l-1} \circ \overline{\mathbf{R}}$, $\forall l \in [1, 2, ..., L]$, then $\mathbf{\Phi}(\mathbf{x}^\theta) = \mathbf{R} \circ \mathbf{y} \circ \overline{\mathbf{R}}$.

$$\begin{aligned}
\mathbf{\Phi}(\mathbf{x}^\theta) =& \Phi_L(\Phi_{L-1}(\cdots \Phi_1(\mathbf{R} \circ \mathbf{x} \circ \overline{\mathbf{R}}))) \\
=& \Phi_L(\Phi_{L-1}(\cdots \mathbf{R} \circ \Phi_1(\mathbf{x}) \circ \overline{\mathbf{R}})) \\
& \cdots \\
=& \Phi_L(\mathbf{R} \circ \Phi_{L-1}(\cdots \Phi_1(\mathbf{x})) \circ \overline{\mathbf{R}}) \\
=& \mathbf{R} \circ \Phi_L(\Phi_{L-1}(\cdots \Phi_1(\mathbf{x}))) \circ \overline{\mathbf{R}} \\
=& \mathbf{R} \circ \mathbf{y} \circ \overline{\mathbf{R}}
\end{aligned} \tag{13}$$