TetraSphere: A Neural Descriptor for O(3)-Invariant Point Cloud Analysis

Pavlo Melnyk, Andreas Robinson, Michael Felsberg, Mårten Wadenbäck
Computer Vision Laboratory, Department of Electrical Engineering, Linköping University, Sweden
{pavlo.melnyk, andreas.robinson, michael.felsberg, marten.wadenback}@liu.se

Abstract

In many practical applications, 3D point cloud analysis requires rotation invariance. In this paper, we present a learnable descriptor invariant under 3D rotations and reflections, i.e.*, the* $\mathrm{O}(3)$ *actions, utilizing the recently introduced steerable 3D spherical neurons and vector neurons. Specifically, we propose an embedding of the 3D spherical neurons into 4D vector neurons, which leverages end-to-end training of the model. In our approach, we perform TetraTransform—an equivariant embedding of the 3D input into 4D, constructed from the steerable neurons—and extract deeper* $\mathrm{O}(3)$*-equivariant features using vector neurons. This integration of the TetraTransform into the VN-DGCNN framework, termed* ***TetraSphere****, negligibly increases the number of parameters by less than 0.0002%. TetraSphere sets a new state-of-the-art performance classifying randomly rotated real-world object scans of the challenging subsets of ScanObjectNN. Additionally, TetraSphere outperforms all equivariant methods on randomly rotated synthetic data: classifying objects from ModelNet40 and segmenting parts of the ShapeNet shapes. Thus, our results reveal the practical value of steerable 3D spherical neurons for learning in 3D Euclidean space. The code is available at* https://github.com/pavlo-melnyk/tetrasphere.

1. Introduction

Automatic processing of 3D data obtained with sensors such as LIDARs, sparse stereo, and sparse time-of-flight is a central problem for many autonomous systems [13, 19, 34]. Point clouds—in the form of an array of a fixed number of 3D coordinates and corresponding optional features (*e.g.*, color or intensity)—are a common representation of such data in various 3D vision tasks.

Consider, for example, the task of 3D object classification, where the goal is to predict the correct class given a point cloud. Importantly, the order of the points and different orientations of the shape do not alter its class membership. This imposes the requirements of permutation and rotation invariance on the classifier. Furthermore, in certain real-world scenarios (such as left- and right-hand traffic), global reflection invariance is desired. For instance, a vehicle designed for either type of traffic may be considered the same.

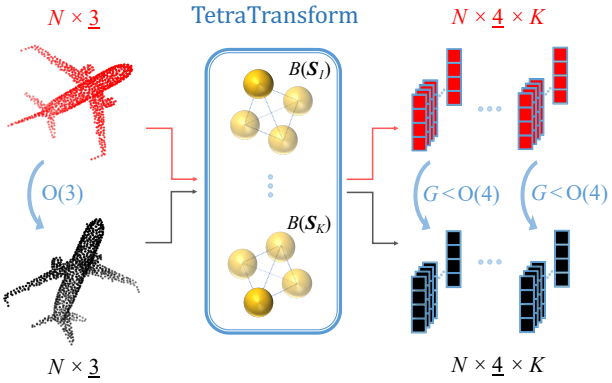

Figure 1. Key component in our method (best viewed in color): a learnable O(3)-equivariant TetraTransform layer consisting of K steerable 3D spherical neurons [28] that lifts the input 3D points to equivariant 4D representations (see Section 4.1 for details).

Fulfilling the first requirement is commonly done by constructing a model using shared multilayer perceptrons (MLPs) and a global aggregation function, producing permutation-invariant features, as in, *e.g.*, PointNet [32].

To attain rotation invariance [40], a common approach is to augment available data by performing random rotations and train the model in the hope that it can generalize to other, possibly unknown, orientations during inference. However, such an approach relies heavily on augmentation and requires an increased model capacity. Such methods are commonly referred to as rotation-sensitive, *e.g.*, [33, 42]. Using the augmentation approach with rotation-sensitive methods only *approximates* rotation invariance. There are also rotation-equivariant methods [10, 38, 43], in which the learned features rotate correspondingly with the input, and rotation-invariant (RI) techniques [6, 8, 18, 24, 47], in which the central trend is to construct RI low-level geometric features and use them instead of point coordinates. An alternative approach is to compute a canonical pose and then de-rotate the input point cloud and perform processing on it [12, 23, 37].

Our method is a combination of SO(3)-equivariant steer-

able 3D spherical neurons [28] and vector neurons[10], where deep rotation-equivariant features are learned using vector neurons and invariant predictions are obtained by taking the inner products of these features *point-wise*. However, unlike the original SO(3)-equivariant framework [10], we propagate equivariant features through the network by constructing a specific 4D space spanned by what we call a *tetra-basis*, as shown in Figure 1. Our main hypothesis is that features from learned rotation-equivariant *TetraTransform* projections are more expressive than the points themselves.

We summarize our contributions as follows:

(1) We propose an embedding of 3D spherical neurons [28] into 4D vector neurons [10], which we show they are both O(3)-equivariant, and propose TetraSphere—a learnable O(3)-invariant descriptor for 3D point cloud classification, built upon VN-DGCNN [10].

(2) We unveil the practical utility of the steerable neurons, which, to the best of our knowledge, have never been used in an end-to-end framework previously.

(3) We demonstrate the effectiveness of TetraSphere by evaluating it on standard benchmarks, consistently outperforming the baseline VN-DGCNN, and setting new state-of-the-art performance classifying arbitrarily rotated real-world scans from ScanObjectNN [39], even when they are significantly perturbed and occluded, and the best performance among equivariant methods benchmarked with the randomly rotated synthetic data from ModelNet40 [44] and ShapeNet [3].

2. Related Work

2.1. Rotation-sensitive 3D point cloud learning

PointNet [32] is the pioneering work for learning on raw point sets as input data for the tasks of classification, part segmentation, and semantic segmentation. Its limited ability for recognizing fine-grained patterns was addressed in the PointNet++ method [33] that recursively applies PointNet on a nested partitioning of the input point cloud. Other noteworthy methods include PointCNN [25] with a special type of convolution operator applied to the input points and features before they are processed by an ordinary convolution, and dynamic graph CNN (DGCNN) [42], where a graph convolution is applied to edges of the k-nearest neighbor graph of the point clouds. Xiang *et al.* [45] introduced CurveNet based on a sequence-of-points (curve) grouping operator and a curve aggregation operator. A more geometrically inspired approach was presented by Melnyk *et al.* [27], who revisited modeling spherical decision surfaces with conformal embedding [30] in the context of learning 3D point cloud representations.

Somewhat surprisingly, similar to the projective method for 3D semantic segmentation by Järemo Lawin *et al.* [22], it was shown by Goyal *et al.* [15] that on a point cloud classification task, a simple projection-based baseline called SimpleView performs on par with 3D approaches. Moreover, the authors designed a protocol for a fair comparison between point cloud learning methods revealing the importance of many factors independent of the proposed architectures, such as evaluation procedure and hyperparameter tuning. Recently, a transformer-based approach combining local and global attention mechanisms was presented by Berg *et al.* [1].

Notably, the aforementioned approaches are rotation-variant, *i.e.*, they require data augmentation if rotation invariance is desired. This also entails the model having an increased number of parameters for memorizing the data in various orientations.

2.2. Rotation-aware models

As an alternative, approaches have been proposed for learning rotation equivariant features, in which learned representations rotate in accordance with the input [2, 14, 26, 31, 38, 52]. Among these are quaternion-based models [35, 52] and methods that perform a projection of the 3D input to a unit sphere [9, 11] and realize convolutions in the spherical harmonic domain.

The work of Deng *et al.* [10] introduced *vector neurons* by extending neurons from 1D scalars to 3D vectors, and thereby enabling a simple mapping of SO(3)-actions to latent spaces in the general rotation-equivariant framework. In the context of equivariant methods, Melnyk *et al.* proposed steerable 3D spherical neurons [28], which are SO(3)-equivariant filter banks obtained by virtue of conformal modeling [27, 30] and the symmetries of spheres as geometric entities [28].

Other methods make use of group representation theory and transform the input points into a space in which it is easier to express rotation-equivariant maps [14, 31, 38], and after that obtain rotation-invariant prediction, *e.g.*, when performing classification. This is achieved using filters constrained to be combinations of spherical harmonics, which limits their expressiveness. Therefore, such methods have naturally limited learning capability, and their performance falls short compared to rotation-sensitive methods for tasks that do not require rotation invariance.

There is a plethora of conceptually different works on hand-crafting low-level rotation invariant (RI) geometric features for arbitrary pairs of points (PPF) based on angles and distances [8, 17, 18, 47, 49, 51], proposed to be used instead of the input point coordinates. For instance, similar to the triplets used by Granlund *et al.* [16], Zhang *et al.* [50] introduced a convolution operator that uses a point neighborhood constructed with triple-point (reference-neighbor-centroid) local triangles. In contrast, vector norm and relative angles between points were used by Chen *et al.* [4]. A robust RI representation, capturing both local and global shape structures, and region relation convolution, alleviating global information loss, were presented by Li *et al.* [24].

The pose information loss problem was revealed and addressed by introducing a pose-aware RI convolution (PaRI-Conv) with compact and efficient kernels by Chen and Cong [5]. Therein, a lightweight augmented PPF (APPF) is proposed, encoding the local pose of each point in a local neighborhood in an ambiguity-free manner. Notably, their approach is also invariant under reflections, *i.e.*, O(3)-invariant, and they use local reference frames (LRFs) as input. However, utilizing principal component analysis (PCA) to construct the LRF for RI point cloud learning, as done by Kim *et al.* [21] and Xiao *et al.* [46], is sensitive to perturbations. This is why Chen and Cong [5] proposed to build the LRFs upon local geometry only.

Input canonicalization is another category of methods that includes both rotation-variant (*e.g.*, variants of [32, 41] that use spatial transformers), -equivariant (*e.g.*, [12, 36, 37]), and -invariant [23] methods. The key idea in these approaches is to bring the input to a computed or predicted canonical reference frame and process it there.

Recently, Yu *et al*. [48] utilized the point-cloud registration approach to achieve rotation invariance. They proposed registering the deep features to rotation-invariant features at intermediate levels in their Aligned Integration Transformer (AIT), thereby increasing feature similarities in the embedding space and attaining rotation invariance.

Our approach builds upon the equivariant framework [10]: we apply steerable 3D spherical neurons [28] to learn in K different spaces O(3)-equivariant 4D features from the 3D input point coordinates, and then aggregate the result by means of an equivariant pooling over K; finally, we propagate through the VN-backbone, wherein the inner product of these features in the equivariant feature space is computed (see Figure 2). This way, we create a learnable O(3)-invariant descriptor, encoding both unambiguous pose information and local and global context.

3. Preliminaries

In this section, we introduce the necessary notation and recap the notion of equivariance and invariance and theoretical results from prior work, which will enable us to realize an embedding of steerable 3D spherical neurons into 4D vector neurons.

We define a 3D point cloud $\mathcal{X} \in \mathbb{R}^{N\times(3+C)}$ as a collection of N points, represented by their coordinates $\mathbf{x} \in \mathbb{R}^3$ concatenated with the corresponding optional features $\mathbf{q} \in \mathbb{R}^C$: $\mathcal{X} = \{\mathbf{x}_n \oplus \mathbf{q}_n\}_{n=1}^N$. In the scope of this paper, we focus only on the point coordinates and assume that the optional features are rotation- and reflection-invariant.

3.1. Equivariance and invariance

Given a group G and a set of transformations $T_g : \mathcal{X} \to \mathcal{X}$ for $g \in G$, a function $f : \mathcal{X} \to \mathcal{Y}$ is said to be *G-equivariant* if for every g, there exists a transformation $V_g : \mathcal{Y} \to \mathcal{Y}$ such that

$$V_g[f(\mathbf{x})] = f(T_g[\mathbf{x}]) \quad \text{for all } g \in G,\ \mathbf{x} \in \mathcal{X}, \tag{1}$$

where T_g represents transformation parameters.

Invariance is a particular type of equivariance. A function $f : \mathcal{X} \to \mathcal{Y}$ is said to be *G-invariant* if for every $g \in G$, the transformation $V_g : \mathcal{Y} \to \mathcal{Y}$ is the identity, *i.e.*,

$$f(\mathbf{x}) = f(T_g[\mathbf{x}]) \quad \text{for all } g \in G,\ \mathbf{x} \in \mathcal{X}. \tag{2}$$

In particular, we consider invariance under 3D orthogonal transformations (rotations and reflections), *i.e.*, the group O(3), and, as an intermediate step, equivariance under 3D rotations—the group SO(3). In order to act as a transformation T_g on a 3D vector $\mathbf{x} \in \mathbb{R}^3$, the elements $g \in$ SO(3) are often represented by 3×3 rotation matrices $\boldsymbol{R}$ [7]. However, this representation is not unique [53].

Our proposed descriptor, which we present in Section 4, is O(3)-invariant and equivariant under permutations of the input points. That is, permuting point indices $1, \ldots, N$ results in the corresponding permutation of the descriptor outputs.

In the remainder of the manuscript, we use the same notation to represent a 3D rotation matrix $\boldsymbol{R}$ in the Euclidean space $\mathbb{R}^3$, the projective (homogeneous) space $P(\mathbb{R}^3) \subset \mathbb{R}^4$, and $\mathbb{R}^5$, by appending the required number of ones to the diagonal of the original rotation matrix without changing the transformation itself [27].

3.2. Spherical neurons

Spherical neurons are defined as neurons with (hyper)spherical decision surfaces [27, 30]. Following Perwass *et al.* [30], one embeds both a data vector $\mathbf{x} \in \mathbb{R}^n$ and a hypersphere $(\mathbf{c}, r)$ in $\mathbb{R}^{n+2}$ as

$$\begin{aligned} \boldsymbol{X} &= \left(x_1, \ldots, x_n, -1, -\frac{1}{2}\|\mathbf{x}\|^2\right) \in \mathbb{R}^{n+2}, \\ \boldsymbol{S} &= \left(c_1, \ldots, c_n, \frac{1}{2}(\|\mathbf{c}\|^2 - r^2), 1\right) \in \mathbb{R}^{n+2}, \end{aligned} \tag{3}$$

where $\mathbf{c} = (c_1, \ldots, c_n) \in \mathbb{R}^n$ is the hypersphere center and $r \in \mathbb{R}$ is its radius. Their scalar product in $\mathbb{R}^{n+2}$ is given by

$$\boldsymbol{X}^\top \boldsymbol{S} = -\frac{1}{2}\|\mathbf{x} - \mathbf{c}\|^2 + \frac{1}{2}r^2 \,. \tag{4}$$

The sign of this scalar product depends on the relative position of the point to the sphere in the Euclidean space $\mathbb{R}^n$: inside the sphere if positive, outside of the sphere if negative, and on the sphere if zero [30]. Perwass *et al.* [30] suggested to use the scalar product (4) as a classifier, *i.e.*, a *spherical neuron* $f_S(\boldsymbol{X}; \boldsymbol{S}) = \boldsymbol{X}^\top \boldsymbol{S}$ with learnable parameters $\boldsymbol{S} \in \mathbb{R}^{n+2}$. Importantly, as noted by Melnyk *et al.* [27], spherical neurons do not necessarily require an activation function, due to the non-linearity of the embedding (3).

During training, the components of $\boldsymbol{S}$ in (3) are treated as independent learnable parameters. Therefore, a spherical neuron effectively learns *non-normalized* hyperspheres of the form $\widetilde{\boldsymbol{S}} = (s_1, \ldots, s_{n+2}) \in \mathbb{R}^{n+2}$. Due to the chosen representation [30], both normalized and non-normalized hyperspheres represent the same decision surface, and the spherical neuron can thus be written as

$$f_S\left(\boldsymbol{X}; \widetilde{\boldsymbol{S}}\right) = \boldsymbol{X}^\top \widetilde{\boldsymbol{S}} = \gamma \boldsymbol{X}^\top \boldsymbol{S}, \tag{5}$$

where $\gamma := s_{n+2}$ is the (learned) normalization parameter and $\boldsymbol{S} \in \mathbb{R}^{n+2}$ is the normalized sphere defined in (3). From this point, we will write $\boldsymbol{S}$ when referring to a spherical decision surface, specifying its normalization if needed.

Further details are found in the work of Melnyk *et al.* [27], where, inter alia, it is demonstrated that the spherical neuron activations are isometries in 3D. That is, rigid transformations commute with the application of the spherical neuron. This result is a necessary condition to design rotation equivariant feature extractors based on spherical neurons [28], that we review in Section 3.3.

3.3. Steerable 3D spherical neurons

A steerable 3D spherical neuron, recently introduced by Melnyk *et al.* [28], is a filter bank consisting of one learnable spherical decision surface $\boldsymbol{S} \in \mathbb{R}^5$ (3) and three copies: The original (learned) sphere center $\mathbf{c}_0$ is first rotated to $\frac{\|\mathbf{c}_0\|}{\sqrt{3}}(1,1,1)$ with the corresponding (geodesic) rotation denoted as $\boldsymbol{R}_O$. The resulting sphere is then rotated into the other three vertices of the regular tetrahedron. This is followed by rotating all four spheres back to the original coordinate system. One steerable 3D spherical neuron is thus composed as the 4×5 matrix

$$B(\boldsymbol{S}) = \left[\left(\boldsymbol{R}_O^\top \boldsymbol{R}_{T_i} \boldsymbol{R}_O \, \boldsymbol{S} \right)^\top \right]_{i=0\ldots3}, \tag{6}$$

where each of $\{\boldsymbol{R}_{T_i}\}_{i=0}^3$ is the isomorphism in $\mathbb{R}^5$ corresponding to a 3D rotation from $(1,1,1)$ to the vertex $i+1$ of the regular tetrahedron. Hence, $\boldsymbol{R}_{T_0} = \mathbf{I}_5$, *i.e.*, $\boldsymbol{S}$ remains at $\mathbf{c}_0$.

We can view the steerable spherical neuron (6) as a function $f_{4S}(\,\cdot\,; \boldsymbol{S}) : \mathbb{R}^5 \to \mathbb{R}^4$ with five learnable parameters as a vector $\boldsymbol{S}$. Crucially for our work, Melnyk *et al.* [28] proved that it is equivariant under 3D rotations:

$$V_{\boldsymbol{R}}\, B(\boldsymbol{S})\, \boldsymbol{X} = B(\boldsymbol{S})\, \boldsymbol{R}\boldsymbol{X}, \tag{7}$$

where $\boldsymbol{X} \in \mathbb{R}^5$ is a properly embedded 3D input point, $\boldsymbol{R}$ is a representation of the 3D rotation in the space $\mathbb{R}^5$, and $V_{\boldsymbol{R}} \in G < \mathrm{SO}(4)$ is the 3D rotation representation in the filter bank output space:

$$V_{\boldsymbol{R}} = \mathbf{M}^\top \boldsymbol{R}_O \, \boldsymbol{R} \, \boldsymbol{R}_O^\top \mathbf{M}\,, \tag{8}$$

where $\mathbf{M} \in \mathrm{SO}(4)$ is a change-of-basis matrix that holds the homogeneous coordinates of the tetrahedron vertices (scaled by $1/2$) in its columns as

$$\mathbf{M} = \frac{1}{2} \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \tag{9}$$

We will use the equivariant filter bank output as a replacement for 3D points, *e.g.*, in vector neural networks (VNNs) by Deng *et al.* [10].

3.4. Vector neurons

Vector neurons (VNs) [10] are designed for processing data embedded in $\mathbb{R}^3$ and produce an ordered set of 3D vectors $\mathbf{y} \in \mathbb{R}^3$ as output. Taking a point cloud $\mathcal{X} \in \mathbb{R}^{N\times 3}$ as input, a VN extracts vector-list features $\mathcal{Y} = \{\boldsymbol{Y}_n\}_{n=1}^N \in \mathbb{R}^{N\times C\times 3}$, where $\boldsymbol{Y} \in \mathbb{R}^{C\times 3}$ is a vector-feature and C is the number of latent channels.

Specifically, a linear layer $f_{\text{lin}}(\,\cdot\,; \mathbf{W})$ comprised of VNs is defined by means of a weight matrix $\mathbf{W} \in \mathbb{R}^{C'\times C}$ acting on a vector-feature $\boldsymbol{Y} \in \mathcal{Y}$ as $f_{\text{lin}}(\boldsymbol{Y}; \mathbf{W}) = \mathbf{W}\boldsymbol{Y}$, and is SO(3)-equivariant since

$$f_{\text{lin}}(\boldsymbol{Y}\boldsymbol{R}; \mathbf{W}) = \mathbf{W}\boldsymbol{Y}\boldsymbol{R} = f_{\text{lin}}(\boldsymbol{Y}; \mathbf{W})\boldsymbol{R} = \boldsymbol{Y}'\boldsymbol{R}, \tag{10}$$

where $\boldsymbol{R} \in \mathrm{SO}(3)$ and $\boldsymbol{Y}' \in \mathbb{R}^{C'\times 3}$.

Deng *et al.* [10] also presented how common neural network operations, such as batch norm [20], pooling, and non-linearities, can be adopted for VNs, and how VNs can be used in other point cloud processing networks. In particular, their VN-DGCNN modifies the permutation-equivariant edge convolution of the predecessor DGCNN [42] by computing adjacent edge features $\boldsymbol{E}'_{nm} \in \mathcal{E}$ of vector-list representations $\boldsymbol{Y}_n \in \mathbb{R}^{C\times 3}$, followed by a local SO(3)-equivariant pooling as

$$\boldsymbol{E}'_{nm} = l_{\text{VN-nonlin}}(\Theta(\boldsymbol{Y}_m - \boldsymbol{Y}_n) + \Phi\boldsymbol{Y}_n), \tag{11}$$

$$\boldsymbol{Y}'_n = l_{\text{VN-pool}\, m:(n,m)\in\mathcal{E}}(\boldsymbol{E}'_{nm}), \tag{12}$$

where Θ and Φ are learnable weight matrices, VN-nonlin and VN-pool are the respective equivariant non-linear and pooling layers (see Section 3 in Deng *et al.* [10] for details). Notably, average pooling, being a linear operation, maintains rotation-equivariance and helps to achieve higher performance [10].

To summarize, the important properties of a VNN [10] are that 1) it is SO(3)-equivariant and produces RI features at the later layers, and 2) the local interaction between the points is modeled by exploiting edges by means of edge convolutions introduced in DGCNN [42].

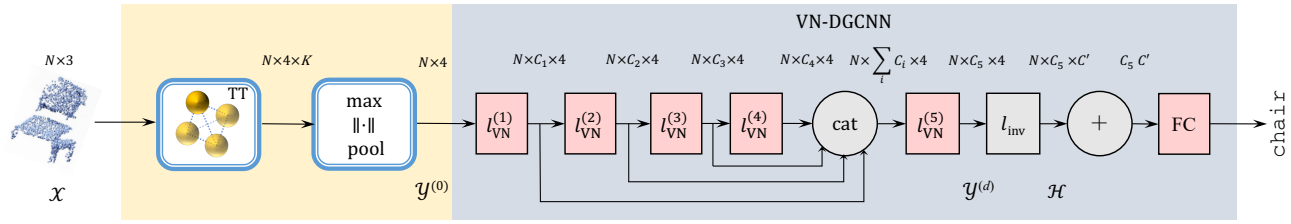

Figure 2. High-level architecture of TetraSphere (for classification): the equivariant TT layer (13) is followed by pooling over K steerable spherical neurons and the application of the equivariant VN-DGCNN [10], consisting of d VN-layers l_{VN} (15), and the block $l_{\text{inv}}(\,\cdot\,;\Theta,\Phi)$ (17), producing invariant features. The first (yellow) block contains the contributions of our work.

4. TetraSphere

In this section, we present TetraSphere—a learnable descriptor for O(3)-invariant point cloud processing—based on steerable 3D spherical neurons and the VN-framework (see Figure 2). Firstly, we note that $\boldsymbol{R}$ in (8) can be a reflection, *i.e.*, have a determinant of -1, which will change the sign of $\det V_{\boldsymbol{R}}$ accordingly, and (7) will still hold in this case. Therefore, $V_{\boldsymbol{R}} \in G < \mathrm{O}(4)$ and steerable neurons (6) are O(3)-equivariant. The same applies to vector neurons: (10) holds even if $\det \boldsymbol{R} = -1$, which means that vector neurons are also O(3)-equivariant.

Our overall approach consists of two steps: 1) we extract O(3)-equivariant features, and 2) we obtain O(3)-invariant representations from them. As the first step, we perform TetraTransform (TT), *i.e.*, lift the 3D input to a specific 4D space spanned by what we call a *tetra-basis* (see Figure 1). Transforming points in the tetra-basis implies embedding a 3D rotation/reflection into a proper subgroup of O(4), as $V_{\boldsymbol{R}} \in G < \mathrm{O}(4)$. Since the entire theory of VNs [10] applies to $\mathbb{R}^4$ and O(4) exactly the same way it does to $\mathbb{R}^3$ and O(3), we plug our TetraTransform into VNs of dimension 4 and achieve O(3) invariance. Note, however, that the VN layers operating on 4D vectors in our model maintain the equivariance under the subgroup $G < \mathrm{O}(4)$.

4.1. Learning O(3)-equivariant features

TetraTransform The first layer $l^{(0)}$ is formed by the TT layer $l_{\text{TT}}(\,\cdot\,;\mathbf{S}) : \mathbb{R}^{N\times 3} \to \mathbb{R}^{N\times 4\times K}$, consists of K steerable spherical neurons $B(\boldsymbol{S}_k)$ (6), representing a $K \times 5$ learnable weight matrix $\mathbf{S}$.

TT first takes in a point cloud of 3D points $\mathcal{X} \in \mathbb{R}^{N\times 3}$ and embeds them in the conformal space $\mathbb{R}^5$ according to (3), resulting in $\{\boldsymbol{X}_n\}_{n=1}^N \in \mathbb{R}^{N\times 5}$.

Following the structure of point cloud processing networks [32, 42], the subsequent application of the steerable spherical neurons (6) as $B(\boldsymbol{S}_k)\boldsymbol{X}$ is shared across points, thus making the output

$$\mathcal{Y}^{(0)} = l_{\text{TT}}(\mathcal{X};\mathbf{S}) \in \mathbb{R}^{N\times 4\times K} \tag{13}$$

both rotation- and permutation-equivariant. Importantly, thanks to the embedding of vectors (3), $l_{\text{TT}}(\,\cdot\,;\mathbf{S})$ is a non-linear layer, which is essential for neural networks.

Tetra-basis projections Note that each of the K steerable spherical neurons (6) in $l_{\text{TT}}(\,\cdot\,;\mathbf{S})$ has its own representation of a 3D rotation $\boldsymbol{R}$, given as $V_{\boldsymbol{R}}^k \in G < \mathrm{O}(4)$, $k \in \{1,\dots K\}$, due to the rotation $\boldsymbol{R}_O$ in (8) (and (6)) being computed from a learnable $\boldsymbol{S}_k$. In fact, we see $\mathcal{Y}^{(0)}$ as a collection of N rotation-equivariant 4D vectors in K different tetra-bases. This must be taken into consideration when transforming $\mathcal{Y}^{(0)}$ so as to preserve equivariance.

Aggregating over tetra-bases The case $K = 1$ corresponds to a non-linear change of the coordinate system from 3D to a 4D space spanned by the tetra-basis. However, to accumulate the features captured in the $K > 1$ tetra-bases, we need to consider aggregation operators that respect equivariance. In our work, we propose to use *maximum* pooling over K steerable neurons/tetra-bases for each of the input N points, thus selecting one of the K 4D outputs of the TT layer, indexed with k^*, and define this operation as follows:

$$l_{\text{pool}}(\mathcal{Y}^{(0)}) = \mathcal{Y}^{(0)}_{:,:,k^*}, \quad k^* = \operatorname*{mode}_n \arg\max_k \|\mathcal{Y}^{(0)}_{n,:,k}\|. \tag{14}$$

For an input point cloud, this operation corresponds to the selection of the k^*-th steerable neuron, the output of which has the maximum l^2-norm for the majority of points. Since each of the K steerable neurons (6) is O(3)-equivariant and hence, preserves the l^2-norm of the output, the proposed selection of one of them is O(3)-invariant, thereby respecting the equivariance of the model.

Deeper equivariant propagation We proceed by adding the O(3)-equivariant VN-framework [10], reviewed in Section 3.4, on top of l_{TT}: we apply VNs to the (pooled over K) first layer output $\mathcal{Y}^{(0)}$, which we, therefore, need to view as a list of vector-features $\mathcal{Y}^{(0)} = \{\boldsymbol{Y}_n\}_{n=1}^N \in \mathbb{R}^{N\times 4}$.

We can thus extend VNs [10] to operate on our specific 4D vectors, contained in $\mathcal{Y}^{(0)}$. Obviously, a linear layer comprised of VNs $f_{\text{lin}}(\,\cdot\,;\mathbf{W})$ is also equivariant under $V_{\boldsymbol{R}} \in G < \mathrm{O}(4)$. By replacing $\boldsymbol{R}$ in (10) with $V_{\boldsymbol{R}}$ in (8), and keeping in mind that vector-features $\boldsymbol{Y}$ contain now 4D vectors, we see that (10) holds. The same applies to other equivariant VN-layers (*e.g.*, non-linearities, batch norm); see Deng *et al.* [10].

We denote a consequent application of O(3)-equivariant (and non-linear) edge convolution (EC) (11) and pooling

(12) layers as $l_{\text{VN}}(\,\cdot\,;\Theta,\Phi) : \mathbb{R}^{N\times C\times 4} \to \mathbb{R}^{N\times C'\times 4}$. In general, the d-th VN-layer taking $\mathcal{Y}^{(d)} \in \mathbb{R}^{N\times C\times 4}$ as input produces an O(3)-equivariant and permutation-equivariant feature map

$$\mathcal{Y}^{(d+1)} = l_{\text{VN}}(\mathcal{Y}^{(d)};\Theta,\Phi) \in \mathbb{R}^{N\times C'\times 4}, \tag{15}$$

where C' are the latent channels. Given the (pooled over K) TT output (13) $\mathcal{Y}^{(0)} \in \mathbb{R}^{N\times 4}$, a VN-layer outputs a feature map $\mathcal{Y}^{(d)} \in \mathbb{R}^{N\times C\times 4}$.

4.2. O(3)-invariant representations

The TetraSphere architecture (see Figure 2), presented in this section, performs TT (13) as the first step.

To obtain RI features, we follow related work (*e.g.*, [10] and [47]) and exploit the fact that the inner product of two roto-equivariant vectors, rotated in $\mathbb{R}^n$ with the same $\boldsymbol{R}$, is invariant:

$$\boldsymbol{U}\boldsymbol{R}\,(\boldsymbol{T}\boldsymbol{R})^\top = \boldsymbol{U}\boldsymbol{R}\boldsymbol{R}^\top\boldsymbol{T}^\top = \boldsymbol{U}\boldsymbol{T}^\top = \mathbf{H}, \tag{16}$$

where $\boldsymbol{U} \in \mathbb{R}^{C\times n}$, $\boldsymbol{T} \in \mathbb{R}^{C'\times n}$, and $\mathbf{H} \in \mathbb{R}^{C\times C'}$. Note that $\mathbf{H}$ is O(n)-invariant since the sign of $\det(\boldsymbol{R})$ does not change the equality (16). To the best of our knowledge, this has not been observed in prior work.

If we take (16) and consider $\boldsymbol{U} \in \mathbb{R}^{C\times 3}$ and $\boldsymbol{T} \in \mathbb{R}^{C'\times 3}$ to be 3D vector-features of the same 3D point, but at two different layers with C and C' channels, respectively, we will get the VN-framework approach (see Section 3.5 in Deng *et al.* [10]). In this case, we refer to (16) as a *point-wise* inner product of features. We adopt this procedure to our 4D vectors: In the first step, TT (13) produces $\mathcal{Y}^{(0)} \in \mathbb{R}^{N\times 4\times K}$. We then apply pooling over K spheres and a desired number of VN-layers (15) to it, obtaining $\mathcal{Y}^{(d)} \in \mathbb{R}^{N\times C\times 4}$. To produce RI features, we follow Deng *et al.* [10] and concatenate $\mathcal{Y}^{(d)}$ with its global mean (over N), $\overline{\mathcal{Y}}^{(d)} = \frac{1}{N}\sum_n \mathcal{Y}^{(d)}_n \in \mathbb{R}^{C\times 4}$, and propagate the result through m additional VN-layers to obtain $\mathcal{Y}^{(d+m)} \in \mathbb{R}^{N\times C'\times 4}$. We then extract matrices $\boldsymbol{U} \in \mathbb{R}^{C\times 4}$ from $\mathcal{Y}^{(d)}$ and $\boldsymbol{T} \in \mathbb{R}^{C'\times 4}$ from $\mathcal{Y}^{(d+m)}$ and perform (16) for all N. Note that the complexity of this product is linear, *i.e.*, $\mathcal{O}(N)$, as opposed to, *e.g.*, the quadratic complexity of the product in the SGM approach [47].

We denote the propagation from VN-layer d to layer $d+m$ with the subsequent point-wise product as a block $l_{\text{inv}}(\,\cdot\,;\Theta,\Phi) : \mathbb{R}^{N\times C\times 4} \to \mathbb{R}^{N\times C\times C'}$, where Θ and Φ denote the learnable parameters of the VN-layers. In practice, we select $C' = 3$ following the original VN-approach [10].

In the case of a single VN-layer (15) following after the TT-layer (13), we describe TetraSphere operating on $\mathcal{X} \in \mathbb{R}^{N\times 3}$ as

$$\mathcal{H} = l_{\text{inv}}(\,l_{\text{VN}}(\,l_{\text{pool}}(\,l_{\text{TT}}(\mathcal{X};\mathbf{S})\,)\,)\,), \tag{17}$$

where $\mathcal{H} \in \mathbb{R}^{N\times C\times C'}$ is an O(3)-invariant and permutation-equivariant descriptor of $\mathcal{X}$, that can be used for various point-cloud analysis tasks.

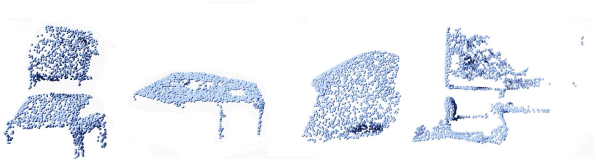

Figure 3. Examples of the objects from the hardest subset of ScanObjectNN [39]: *chair*, *table*, *pillow*, and *display*.

5. Experiments

In this section, we conduct experiments with TetraSphere based on the rotation-equivariant VN-DGCNN architecture [10]. We evaluate our model using both synthetic and real-world 3D data and compare it with other methods.

5.1. Datasets and tasks

Real data classification We first consider the task of classification and evaluate our method on real-world indoor scenes. For this, we use ScanObjectNN [39], which consists of 2902 unique object instances belonging to 15 classes. We employ two subsets: the easiest, called *OBJ_BG* and containing objects with background, and the most challenging subset called *PB_T50_RS*, consisting of approximately 15,000 point clouds that undergo 50% bounding box translation, random rotation around the gravity axis, and random scaling, in which perturbations introduce various levels of partiality to the objects. We follow the train/test split provided by the original repository[1]. Some examples are presented in Figure 3. We preprocess both datasets the same way, sampling 1024 points per object instance, centering them at the origin, and normalizing them to be within a unit sphere.

Synthetic data classification and part segmentation In addition, we evaluate our model on the tasks of classifying ModelNet40 data [44] provided by [32] that consist of 12,311 CAD models randomly sampled as point clouds comprised of 1024 points, and the task of part segmentation using ShapeNet-part [3], consisting 16,881 point clouds of 16 categories partitioned with 50 part labels in total. We follow [10] for the train/test split and sample 2048 points for the model input.

Rotation setup In general, we employ the following train/test rotation settings, following the general convention [10, 48, 51]: z/z, $z/\{\text{SO}(3),\ \text{O}(3)\}$ and $\text{SO}(3)/\,\text{SO}(3)$, with the second one being the most challenging and the most practical (in which we also include O(3) to test the invariance under the transformations of the full orthogonal group). Here, z denotes vertical-axis rotation augmentation, SO(3) stands for arbitrary 3D rotations, and O(3) for arbitrary rotations+reflections, all generated and applied to the input shapes during training/testing. Note that the *PB_T50_RS* subset of ScanObjectNN already includes z-axis rotation augmentations, and therefore, we do not need to use additional augmentation during training for the $z/\cdot$ scenarios.

[1] https://github.com/hkust-vgd/scanobjectnn

5.2. Architecture and implementation details

We use VN-DGCNN [10] as the backbone, with the standard choice of $k = 20$ (nearest-neighbor graph computation parameter) for classification and $k = 40$ for part-segmentation for all layers and the dropout in the last two fully-connected layers of 0.5. We apply VN-LeakyReLU as the learnable equivariant non-linearity in (11), and use average pooling in VN-layers (12), given its reported higher performance. The architecture for part-segmentation experiments is the same, except the VN-DGCNN backbone is adjusted accordingly (as per [10]). We experiment with different numbers K of steerable spherical neurons in the TT layer and refer to the resulting model simply as **TetraSphere**.

We adopt the official implementation of Deng *et al.* [10] to implement our model in PyTorch [29]. Following Melnyk *et al.* [28], we initialize the parameters in the TT layer (*i.e.*, the spheres) using the standard initialization for the linear layers in PyTorch. We use the same hyperparameters for training TetraSphere as the baseline [10]: We employ SGD with an initial learning rate of 0.1 and momentum equal to 0.9, and a cosine annealing strategy for gradually reducing the learning rate to 0.001, and minimize cross-entropy with smoothed labels. Like the baseline, we augment the data with random translation in the range $[-0.2, 0.2]$ and scaling in the range $[2/3, 3/2]$ during training. We train TetraSphere for 1000 epochs for all ScanObjectNN experiments. Following the baseline, we set the number of epochs to 250 for ModelNet40 classification, and 200 for ShapeNet part segmentation. The batch size is set to 32.

5.3. Results and discussion

The main results of our experiments are presented in Tables 1, 2, and 3, where for a fair comparison, we list methods that only use point clouds as input, and no additional information, such as normals or features, or test-time augmentation. From Table 1, in the task of classifying the easier subset of real-world object scans with background (and no perturbations), our method outperforms the baseline VN-DGCNN, especially in the more practical $z/\,\mathrm{SO}(3)$ scenario, and the recent method by Yu *et al.* [48], thus setting a new state-of-the-art performance. Here, we observe that increasing the number of steerable neurons beyond $K = 2$ does not systematically improve the performance. In general, the results under the $\mathrm{SO}(3)/\,\mathrm{SO}(3)$ protocol indicate that additional rotation augmentation when classifying non-perturbed shapes is not required for our method.

The previous best result published in the literature in our comparison for the classification of the perturbed real object scans from the most challenging subset of ScanObjectNN (see Table 2) is 3D-GFE [8]. We used the open-source implementations and evaluated the recent related methods, showing that VN-DGCNN exhibits better robustness to perturbations (comparing with the results of the perturbation-free *OBJ_BG* test in Table 1) — a property we attribute to the equivariant feature extraction scheme of the VN framework. With our TetraSphere (built upon VN-DGCNN), the performance is further boosted thanks to the 4D representation learning enabled by the TetraTransform layer with $K \geq 1$: TetraSphere achieves state-of-the-art classification performance.

As shown in Table 3, TetraSphere outperforms equivariant baselines at the tasks of classifying and segmenting parts of the synthetic shapes. Our model is only surpassed

Methods	z/z	$z/\,\mathrm{SO}(3)$	$\mathrm{SO}(3)/\,\mathrm{SO}(3)$
Rotation-sensitive			
PointCNN [25]	86.1	14.6	63.7
DGCNN [41]	82.8	17.7	71.8
Rotation-robust			
Li *et al.* [23]	84.3	84.3	84.3
PaRINet [6]	77.8	77.8	78.1
PaRINet + PCA [6]	83.3	83.3	83.3
Yu *et al.* [48]	-	<u>86.6</u>	<u>86.3</u>
VN-DGCNN [10] *	83.5	83.5	84.2
TetraSphere$_{K=1}$	84.7	84.7	86.2
TetraSphere$_{K=2}$	**87.3**	**87.3**	84.9
TetraSphere$_{K=4}$	84.5	84.5	**87.1**
TetraSphere$_{K=8}$	86.2	86.2	84.9
TetraSphere$_{K=16}$	85.4	85.4	85.9

Table 1. Classification acc. (%) on the real-world objects from the *OBJ_BG* (easiest) subset of ScanObjectNN under different train/test settings of rotation augmentation. The overall best results are presented in **bold**, and the second-best are <u>underlined</u>. We evaluated methods marked with * using their open-source implementation.

Methods	z/z	$z/\,\mathrm{SO}(3)$	$\mathrm{SO}(3)/\,\mathrm{SO}(3)$
Rotation-sensitive			
PointCNN [25]	78.5	14.9	51.8
DGCNN [41]	78.1	16.1	63.4
Rotation-robust			
3D-GFE [8]	73.5	72.7	73.5
Li *et al.* [23] *	74.6	74.6	74.9
PaRINet [6] *	71.6	71.6	72.2
Yu *et al.* [48] *	77.2	77.2	77.4
VN-DGCNN [10] *	77.9	77.9	78.5
TetraSphere$_{K=1}$	78.5	78.5	<u>78.7</u>
TetraSphere$_{K=2}$	<u>78.9</u>	<u>78.9</u>	**79.0**
TetraSphere$_{K=4}$	**79.2**	**79.2**	**79.0**
TetraSphere$_{K=8}$	78.7	78.7	**79.0**
TetraSphere$_{K=16}$	78.8	78.8	**79.0**

Table 2. Classification acc. (%) on the real-world objects from the *PB_T50_RS* (hardest) subset of ScanObjectNN under different train/test settings of rotation augmentation. The overall best results are presented in **bold**, and the second-best are <u>underlined</u>. We evaluated methods marked with * using their open-source code.

Methods	ModelNet40		ShapeNet	
	z/z	$z/$ SO(3)	z/z	$z/$ SO(3)
TFN [31]	89.7	89.7	-	78.1
VN-DGCNN [10]	89.5	89.5	81.4	81.4
TetraSphere$_{K=1}$	89.5	89.5	82.1	82.1
TetraSphere$_{K=2}$	89.7	89.7	**82.3**	**82.3**
TetraSphere$_{K=4}$	<u>90.0</u>	<u>90.0</u>	<u>82.2</u>	<u>82.2</u>
TetraSphere$_{K=8}$	**90.5**	**90.5**	**82.3**	**82.3**
TetraSphere$_{K=16}$	89.8	89.8	**82.3**	**82.3**

Table 3. Comparison of rotation-equivariant methods using synthetic noiseless data. Left: Classification accuracy (%) on the ModelNet40 shapes. Right: Part segmentation of the ShapeNet shapes, mIoU (%). The best results are presented in **bold**, and the second-best are <u>underlined</u>.

by PaRINet [6] (the complete tables are presented in the Supplementary Material) and by Yu *et al*. [48] (only on ModelNet40), both of which TetraSphere exceeds the performance of on the other two real-data benchmarks (see Tables 1 and 2), even when PaRINet is aided by PCA. Compared to the proposed equivariant method, the previous state-of-the-art methods degrade when the effects of real data occur: noise, occlusion, and outliers.

We also experimentally verify that TetraSphere is O(3)-invariant by applying random reflections in addition to rotations during inference, as shown in Table 4: the accuracies of the TetraSphere evaluated on the data augmented with z-axis rotations and O(3)-transformations are identical.

Furthermore, we perform an ablation study testing the importance of the 4D representation learned by TetraSpheres as opposed to the baseline VN-DGCNN operating on the original 3D point coordinates appended with a fourth, invariant, component. For this, we append each input point, $\mathbf{x} \in \mathbb{R}^3$ with its norm, thus making the input 4D, *i.e*., $[\mathbf{x}, \|\mathbf{x}\|]$. We also compare our model to VN-DGCNN$_{+l_0}$ and VN-DGCNN$_{+l_0}([\mathbf{x}, \|\mathbf{x}\|])$ models, in which the baseline VN-DGCNN has an additional 0-th (equivariant) VN-layer inserted at the beginning, which makes the model have the same depth as TetraSphere and adds 366 parameters to the baseline (0.01265%). As presented in Table 4, TetraSphere outperforms the baseline.

Learned Tetra-selection Even though our proposed equivariant pooling (14) allows for selecting different steerable neurons (from the K available ones) for different inputs, we found that TetraSphere learns to select the same neuron (*i.e*., tetra-basis) for *all* inputs. As we present in the Supplementary Material, our model does so by learning all but one γ parameter of the spherical decision surfaces (see (5)) defining the steerable neuron (6) in the TT layer (13), to be close to 0. This renders the l^2-norms of the 4D activations of the corresponding steerable neurons negligible, thus making TetraSphere always select the steerable neuron with a non-zero γ. This means that one can prune the network after training, based on the learned parameters of the TT layer, effectively obtaining $K = 1$ at inference, thus reducing the computational time.

Methods	z/z	$z/$O(3)
VN-DGCNN	82.8 ± 0.6 (83.5)	82.8 ± 0.6 (83.5)
VN-DGCNN($[\mathbf{x}, \|\mathbf{x}\|]$)	82.7 ± 1.6 (84.5)	82.7 ± 1.6 (84.5)
VN-DGCNN$_{+l_0}$	83.3 ± 0.8 (83.8)	83.3 ± 0.8 (83.8)
VN-DGCNN$_{+l_0}([\mathbf{x}, \|\mathbf{x}\|])$	82.7 ± 0.1 (82.8)	82.7 ± 0.1 (82.8)
TetraSphere$_{K=2}$	**85.5** $\pm$ **2.2** (**87.3**)	**85.5** $\pm$ **2.2** (**87.3**)

Table 4. O(3)-test and ablation: Classification acc. (mean and std over 3 runs with the best result in parentheses, %) on ScanObjectNN *OBJ_BG* objects under different train/test transformation settings.

Complexity analysis Since TetraSphere is predominantly based on VN-DGCNN, which is in turn based on DGCNN, its computational complexity is not fundamentally different from other methods [10]. The parameter difference between TetraSphere and the baseline VN-DGCNN is negligible: the former has only one additional TetraTransform layer (see Figure 2), containing K learnable spheres with 5 parameters each (less than 0.0002% of the baseline size). The time complexity difference between the two comes from the usage of 4D vectors by TetraSphere and partially from the TetraTransform operations—the application of the steerable neurons (6). Benchmarked on NVIDIA A100, a forward pass through VN-DGCNN takes 5.1ms vs. 6.6ms ($\Delta = 1.5$ms) through VN-DGCNN operating on 4D vectors ($[\mathbf{x}, \|\mathbf{x}\|]$) vs. 7.9ms ($\Delta = 1.3$ms) through our implementation of TetraSphere.

6. Conclusion

In this paper, we proposed the O(3)-invariant TetraSphere descriptor as an embedding of steerable 3D spherical neurons into 4D vector neurons. To the best of our knowledge, we use the steerable neurons in an *end-to-end* approach for the first time, thereby unveiling their practical utility. TetraSphere sets a new state-of-the-art performance on the task of classifying randomly rotated 3D objects from the challenging real-world ScanObjectNN dataset, and the best results among equivariant methods for classifying and segmenting parts of randomly rotated synthetic shapes from ModelNet40 and ShapeNet, respectively. We look forward to our work paving the path to geometrically justified and more robust handling of real-world 3D data.

Acknowledgments This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), by the Swedish Research Council through a grant for the project Uncertainty-Aware Transformers for Regression Tasks in Computer Vision (2022-04266), and the strategic research environment ELLIIT. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) partially funded by the Swedish Research Council through grant agreement no. 2022-06725.3, and by the Berzelius resource provided by the Knut and Alice Wallenberg Foundation at the National Supercomputer Centre.

References

[1] Axel Berg, Magnus Oskarsson, and Mark O'Connor. Points to patches: Enabling the use of self-attention for 3d shape recognition. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 528–534. IEEE, 2022. 2

[2] Georg Bökman, Fredrik Kahl, and Axel Flinth. Zz-net: A universal rotation equivariant architecture for 2d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10976–10985, 2022. 2

[3] 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. 2, 6

[4] 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/CVF conference on computer vision and pattern recognition*, pages 4994–5002, 2019. 2

[5] Jiayi Chen, Yingda Yin, Tolga Birdal, Baoquan Chen, Leonidas J. Guibas, and He Wang. Projective manifold gradient layer for deep rotation regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6646–6655, 2022. 3

[6] Ronghan Chen and Yang Cong. The Devil is in the Pose: Ambiguity-free 3D Rotation-invariant Learning via Pose-aware Convolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7472–7481, 2022. 1, 7, 8, 2

[7] Gregory S Chirikjian. *Engineering applications of noncommutative harmonic analysis: with emphasis on rotation and motion groups*. CRC press, 2000. 3

[8] Yu-Chen Chou, Yen-Po Lin, Yang-Ming Yeh, and Yi-Chang Lu. 3d-gfe: a three-dimensional geometric-feature extractor for point cloud data. In *2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 2013–2017, 2021. 1, 2, 7

[9] Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. *arXiv preprint arXiv:1801.10130*, 2018. 2

[10] Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas J Guibas. Vector neurons: A general framework for SO(3)-equivariant networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12200–12209, 2021. 1, 2, 3, 4, 5, 6, 7, 8

[11] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) Equivariant Representations with Spherical CNNs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 2

[12] Jin Fang, Dingfu Zhou, Xibin Song, Shengze Jin, Ruigang Yang, and Liangjun Zhang. Rotpredictor: Unsupervised canonical viewpoint learning for point cloud classification. In *2020 International Conference on 3D Vision (3DV)*, pages 987–996, 2020. 1, 3

[13] Hamidreza Fazlali, Yixuan Xu, Yuan Ren, and Bingbing Liu. A versatile multi-view framework for lidar-based 3d object detection with guidance from panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17192–17201, 2022. 1

[14] Fabian Fuchs, Daniel Worrall, Volker Fischer, and Max Welling. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. In *Advances in Neural Information Processing Systems*, pages 1970–1981. Curran Associates, Inc., 2020. 2

[15] Ankit Goyal, Hei Law, Bowei Liu, Alejandro Newell, and Jia Deng. Revisiting point cloud shape classification with a simple and effective baseline. *International Conference on Machine Learning*, 2021. 2

[16] Gosta H Granlund and Anders Moe. Unrestricted recognition of 3d objects for robotics using multilevel triplet invariants. *AI Magazine*, 25(2):51–51, 2004. 2

[17] Ruibin Gu, Qiuxia Wu, Hongbin Xu, Wing W.Y. Ng, and Zhiyong Wang. Learning efficient rotation representation for point cloud via local-global aggregation. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2021. 2

[18] Ruibin Gu, Qiuxia Wu, Yuqiong Li, Wenxiong Kang, Wing W. Y. Ng, and Zhiyong Wang. Enhanced local and global learning for rotation-invariant point cloud representation. *IEEE MultiMedia*, 29(4):24–37, 2022. 1, 2

[19] Hanjiang Hu, Zuxin Liu, Sharad Chitlangia, Akhil Agnihotri, and Ding Zhao. Investigating the impact of multi-lidar placement on object detection for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2550–2559, 2022. 1

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 4

[21] Seohyun Kim, Jaeyoo Park, and Bohyung Han. Rotation-invariant local-to-global representation learning for 3d point cloud. *Advances in Neural Information Processing Systems*, 33:8174–8185, 2020. 3

[22] Felix Järemo Lawin, Martin Danelljan, Patrik Tosteberg, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Deep projective 3d semantic segmentation. In *International Conference on Computer Analysis of Images and Patterns*, pages 95–107. Springer, 2017. 2

[23] Feiran Li, Kent Fujiwara, Fumio Okura, and Yasuyuki Matsushita. A closer look at rotation-invariant deep point cloud analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16218–16227, 2021. 1, 3, 7, 2

[24] Xianzhi Li, Ruihui Li, Guangyong Chen, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. A rotation-invariant framework for deep point cloud analysis. *IEEE Transactions on Visualization and Computer Graphics*, 2021. 1, 2

[25] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution on x-transformed points. *Advances in neural information processing systems*, 31:820–830, 2018. 2, 7

[26] Shitong Luo, Jiahan Li, Jiaqi Guan, Yufeng Su, Chaoran Cheng, Jian Peng, and Jianzhu Ma. Equivariant point cloud

analysis via learning orientations for message passing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18932–18941, 2022. 2

[27] Pavlo Melnyk, Michael Felsberg, and Mårten Wadenbäck. Embed Me if You Can: A Geometric Perceptron. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1276–1284, 2021. 2, 3, 4, 1

[28] Pavlo Melnyk, Michael Felsberg, and Mårten Wadenbäck. Steerable 3D Spherical Neurons. In *Proceedings of the 39th International Conference on Machine Learning*, pages 15330–15339. PMLR, 2022. 1, 2, 3, 4, 7

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019. 7

[30] Christian Perwass, Vladimir Banarer, and Gerald Sommer. Spherical decision surfaces using conformal modelling. In *Joint Pattern Recognition Symposium*, pages 9–16. Springer, 2003. 2, 3, 4

[31] Adrien Poulenard and Leonidas J. Guibas. A functional approach to rotation equivariant non-linearities for tensor field networks. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13169–13178, 2021. 2, 8

[32] 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. 1, 2, 3, 5, 6

[33] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 1, 2

[34] Corentin Sautier, Gilles Puy, Spyros Gidaris, Alexandre Boulch, Andrei Bursuc, and Renaud Marlet. Image-to-lidar self-supervised distillation for autonomous driving data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9891–9901, 2022. 1

[35] Wen Shen, Binbin Zhang, Shikun Huang, Zhihua Wei, and Quanshi Zhang. 3d-rotation-equivariant quaternion neural networks. In *European Conference on Computer Vision*, pages 531–547. Springer, 2020. 2

[36] Riccardo Spezialetti, Federico Stella, Marlon Marcon, Luciano Silva, Samuele Salti, and Luigi Di Stefano. Learning to orient surfaces by self-supervised spherical cnns. *Advances in Neural information processing systems*, 33:5381–5392, 2020. 3

[37] Weiwei Sun, Andrea Tagliasacchi, Boyang Deng, Sara Sabour, Soroosh Yazdani, Geoffrey E Hinton, and Kwang Moo Yi. Canonical capsules: Self-supervised capsules in canonical pose. In *Advances in Neural Information Processing Systems*, pages 24993–25005. Curran Associates, Inc., 2021. 1, 3

[38] 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. 1, 2

[39] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *International Conference on Computer Vision (ICCV)*, 2019. 2, 6

[40] Luc Van Gool, Theo Moons, Eric Pauwels, and André Oosterlinck. Vision and Lie's approach to invariance. *Image and vision computing*, 13(4):259–277, 1995. 1

[41] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. Graph.*, 38(5), 2019. 3, 7, 2

[42] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019. 1, 2, 4, 5

[43] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco S Cohen. 3D steerable CNNs: Learning rotationally equivariant features in volumetric data. In *Advances in Neural Information Processing Systems*, pages 10381–10392, 2018. 1

[44] 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. 2, 6

[45] Tiange Xiang, Chaoyi Zhang, Yang Song, Jianhui Yu, and Weidong Cai. Walk in the Cloud: Learning Curves for Point Clouds Shape Analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 915–924, 2021. 2

[46] Zelin Xiao, Hongxin Lin, Renjie Li, Lishuai Geng, Hongyang Chao, and Shengyong Ding. Endowing deep 3d models with rotation invariance based on principal component analysis. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2020. 3

[47] Jianyun Xu, Xin Tang, Yushi Zhu, Jie Sun, and Shiliang Pu. SGMNet: Learning rotation-invariant point cloud representations via sorted Gram matrix. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10468–10477, 2021. 1, 2, 6

[48] Jianhui Yu, Chaoyi Zhang, and Weidong Cai. Rethinking rotation invariance with point cloud registration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3313–3321, 2023. 3, 6, 7, 8, 1, 2

[49] Junming Zhang, Ming-Yuan Yu, Ram Vasudevan, and Matthew Johnson-Roberson. Learning rotation-invariant representations of point clouds using aligned edge convolutional neural networks. In *2020 International Conference on 3D Vision (3DV)*, pages 200–209. IEEE, 2020. 2

[50] Zhiyuan Zhang, Binh-Son Hua, David W Rosen, and Sai-Kit Yeung. Rotation invariant convolutions for 3d point clouds deep learning. In *2019 International Conference on 3D Vision (3DV)*, pages 204–213. IEEE, 2019. 2

[51] Chen Zhao, Jiaqi Yang, Xin Xiong, Angfan Zhu, Zhiguo Cao, and Xin Li. Rotation invariant point cloud classification:

Where local geometry meets global topology. *arXiv preprint arXiv:1911.00195*, 2019. 2, 6

[52] Yongheng Zhao, Tolga Birdal, Jan Eric Lenssen, Emanuele Menegatti, Leonidas Guibas, and Federico Tombari. Quaternion equivariant capsule networks for 3d point clouds. In *European Conference on Computer Vision*, pages 1–19. Springer, 2020. 2

[53] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5745–5753, 2019. 3