Vector Neurons: A General Framework for SO(3)-Equivariant Networks

Congyue Deng[1]   Or Litany[2]   Yueqi Duan[1]   Adrien Poulenard[1]
Andrea Tagliasacchi[3,4]   Leonidas Guibas[1]
[1]Stanford University   [2]NVIDIA   [3]Google Research   [4]University of Toronto

Abstract

Invariance and equivariance to the rotation group have been widely discussed in the 3D deep learning community for pointclouds. Yet most proposed methods either use complex mathematical tools that may limit their accessibility, or are tied to specific input data types and network architectures. In this paper, we introduce a general framework built on top of what we call Vector Neuron *representations for creating SO(3)-equivariant neural networks for pointcloud processing. Extending neurons from 1D scalars to 3D vectors, our vector neurons enable a simple mapping of SO(3) actions to latent spaces thereby providing a framework for building equivariance in common neural operations – including linear layers, non-linearities, pooling, and normalizations. Due to their simplicity, vector neurons are versatile and, as we demonstrate, can be incorporated into diverse network architecture backbones, allowing them to process geometry inputs in arbitrary poses. Despite its simplicity, our method performs comparably well in accuracy and generalization with other more complex and specialized state-of-the-art methods on classification and segmentation tasks. We also show for the first time a rotation equivariant reconstruction network. Source code is available at* https://github.com/FlyingGiraffe/vnn.

1. Introduction

With the proliferation of lower-cost depth sensors, learning on 3D data has seen rapid progress in recent years. Of particular interest are pointcloud networks, such as Point-Net [27] or ACNe [33] that fully respect the inherent set symmetry – that point sets are not ordered – by incorporating order-invariant and/or order-equivariant layers. Yet, there are other important symmetries that have been less perfectly addressed in the context of pointcloud processing, with 3D rotations being a prime example. Consider a scenario where one scans an object using their LIDAR-equipped phone to retrieve similar objects. Clearly, the global object pose should not affect the query result. Point-Net uses spatial transformer layers [16], which only attain
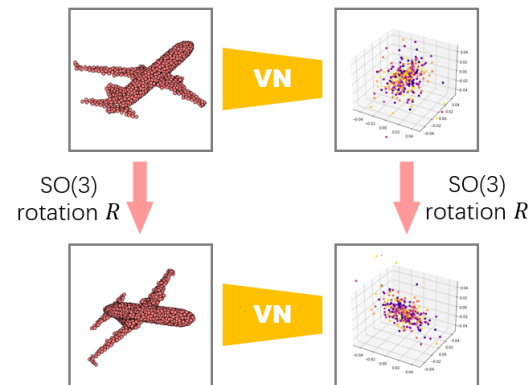

Figure 1: By lifting latent representations from vectors of scalar entries to vectors of 3D points (i.e., matrices) we facilitate the creation of a simple rotation equivariant toolbox allowing the implementation of fully equivariant pointcloud networks.

approximate pose invariance while also requiring extensive augmentation at train time.

To avoid an exhaustive data augmentation with all possible rotations, there is a need for network layers that are equivariant to both order and SO(3) symmetries. Recently, two approaches have been introduced to tackle this setting: Tensor Field Networks [35] and SE(3)-Transformers [14]. While guaranteeing equivariance by construction, both frameworks involve an intricate formulation and are hard to incorporate into existing pipelines as they are restricted to convolutions and rely on relative positions of adjacent points.

In this work, we address these issues by proposing a simple, lightweight framework to build SO(3) equivariant and invariant pointcloud networks. A core ingredient in our framework is a *Vector Neuron* (VN) representation, extending classical scalar neurons to 3D vectors. Consequently, instead of latent vector representations which can be views as ordered sequences of scalars, we deploy latent matrix representations which can be viewed as (ordered) sequences of 3-vectors. Such a representation supports a direct map-
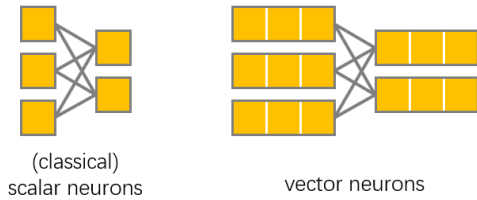
(classical)
scalar neurons

vector neurons

Figure 2: **Linear layer** – Typical neural networks today are built with "scalar" neurons – where the output of the non-linearities in a given layer is an ordered list of *scalars*. We extend deep networks to allow for "vector" neurons – where the output of the non-linearity is an ordered list of *vectors*.

ping of rotations applied to the input pointcloud to intermediate layers. This is in contrast to more complex solutions based on Wigner D-matrices [8]. Another appealing property of VN representations is that they remain equivariant to linear layers by construction. The challenge in building a fully-equivariant network lies in the non-linear activations. In particular, standard neuron-wise activation functions such as ReLU will not commute with a rotation operation. A key contribution in this work is a 3D generalization of classical activation functions by implementing them through a learned direction. For example, when applied to a vector neuron, a standard fixed direction ReLU activation would simply truncate the half-plane that points in its opposite direction. Instead, dynamically predicting an activation direction in a linear data-dependent fashion allows us to guarantee equivariance. We further provide an invariant pooling operation as well as normalization layers, which altogether render our framework compatible with various pointcloud network backbones. To demonstrate its versatility and efficiency, we implemented vector neuron versions of two popular architectures: PointNet and DGCNN, and tested them on three different downstream tasks: classification (permutation invariant and rotation invariant), segmentation (permutation equivariant and rotation invariant), and reconstruction (rotation equivariant on the encoder side, and rotation invariant on the decoder side). Despite its simplicity and lightweight architecture, in all tasks, our VN achieved top performance when tested on randomly rotated shapes compared with other equivariant architectures, and markedly improved performance compared to augmentation-induced equivariance approaches.

To summarize, our key contributions are:

- We propose a new versatile framework for constructing SO(3)-equivariant pointcloud networks.
- Our building blocks are lightweight in terms of the number of learnable parameters and can be easily incorporated into existing network architectures.
- We support a variety of learning tasks, in particular, we

are the first to demonstrate a 3D equivariant network for 3D reconstruction.

- When evaluated on classification and segmentation, our VN version of popular non-equivariant architectures achieve state-of-the-art performance.

2. Related Work

The lack of robustness to rotation in classical pointcloud deep learning architectures like PointNet [27], PointNet++ [28], DGCNN [38], PCNN [2], PointCNN [21] (and many others) has driven interest for rotation invariant and equivariant designs. In recent years, the field of rotation invariant and equivariant deep learning for geometry processing has been rapidly developing. In what follows, we briefly review methods that achieve invariance and equivariance, including those that achieve equivariance via pose estimation.

Rotation invariant methods. Rotation invariance is a desirable property for tasks like shape classification or segmentation. Many rotation invariant architectures [22, 26, 6, 43, 45, 19, 46, 29] have been proposed to address these issues. For example, [6, 43, 45, 19] introduce cleverly designed rotation invariant operations. GC-Conv [45] relies on multi-scale reference frames based on PCA. RI-Framework [19] and LGR-Net [46] pairs local invariant information with global context, some of which like LGR-Net [46] use surface normals in addition to the points coordinates. SFCNN [29] proposes an approach similar to multiview by mapping input pointclouds to a sphere and performing operations on the sphere. Other works like [22, 26] rely on more principled approaches borrowing tools from equivariant deep learning.

Rotation equivariant methods. Recently, rotation equivariant architectures have emerged. A whole body of work is built on the theory of SO(3) representations [35, 17, 12, 39, 1, 32, 14, 25] – most of these works rely on the concept of convolution with steerable kernel bases. A steerable kernel basis is a family of function undergoing a rotation in function space given a rotation of their input parameter. Features computed through these convolution inherit this equivariant behavior. A rotation of the object in Euclidean space induces a rotation of the features in feature space. We refer to [18] for a general theory of steerable kernels. Other works like EMVnet [13] considers a multi-view image based representation of the shapes based on renderings of meshes. In the context of pointcloud network, the universality of rotation equivariance has been studied in [11].

Equivariance via pose estimation. Qi et al. [27] achieved approximate pose equivariance by factoring out SO(3) transformations through object pose estimation. Most works in the literature study *instance-level* pose estimation, where the ground-truth canonical pose of the 3D CAD mod-

els corresponding to the input pointcloud is available [4]. More recently Wang et al. [37] introduced *category-level* pose estimation, and its extension to articulated objects has also been proposed [20]. While both methods [37, 20] need explicit 2D-to-3D supervision, relaxing supervision is possible by borrowing ideas from Transforming Auto-Encoders [15, 30, 40, 10]. However, while Sun et al. [34] learns *category-level* as well as *multi-category* pose estimation in a fully unsupervised fashion, the underlying equivariant backbone [33] is only equivariant by *augmentation*.

3. Method

We introduce *Vector Neuron Networks* (VNNs), a straightforward extension to classical ReLU networks that provides SO(3) equivariance by construction. Neurons in standard artificial neural networks are built from scalars $z \in \mathbb{R}$. When stacked into an ordered list, these neurons form a $C^{(d)}$ dimensional latent feature $\boldsymbol{z} = [z_1, z_2, \cdots, z_{C^{(d)}}]^\top \in \mathbb{R}^{C^{(d)}}$, where (d) indexes the layer depth[1].

However, when processing data embedded in $\mathbb{R}^3$ like 3D pointclouds, realizing the effect of SO(3) transformations applied to the input shape on these vector hidden layers is not obvious. In particular, here we are interested in constructing rotation-equivariant learnable layers, namely layers that commute with the action of the rotation group.

To this end, we propose to "lift" the neuron representation from a scalar $z \in \mathbb{R}$ to a vector $\boldsymbol{v} \in \mathbb{R}^3$, leading to what we call a Vector Neuron (VN). This results in list of Vector Neurons (matrix) $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_C]^\top \in \mathbb{R}^{C \times 3}$. Similar to standard latent representations, this vector-list feature can be used to encode an entire 3D shape, part of it, or a single point. In particular, when representing an (order-less) set of N points $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\} \in \mathbb{R}^{N \times 3}$ in a pointcloud we can consider a collection of N such vector-list features $\mathcal{V} = \{\boldsymbol{V}_1, \boldsymbol{V}_2, \cdots, \boldsymbol{V}_N\} \in \mathbb{R}^{N \times C \times 3}$. Similar to standard neural networks, the number of latent channels $C^{(d)}$ can change between layers via a mapping:

$$\mathcal{V}^{(d+1)} = f(\mathcal{V}^{(d)}; \theta) : \mathbb{R}^{N \times C^{(d)} \times 3} \to \mathbb{R}^{N \times C^{(d+1)} \times 3}, \quad (1)$$

where θ represents learnable parameters.

Using this representation, the action of a rotation is made simple to implement, as it involves the application of a standard rotation matrix. Critically, we require that the mapping satisfies rotation equivariance, namely, for any rotation matrix $\boldsymbol{R} \in \mathrm{SO}(3)$:

$$f(\mathcal{V}R; \theta) = f(\mathcal{V}; \theta)R, \quad (2)$$

where we interpret the application of the rotation matrix to the set as $\mathcal{V}R = \{\boldsymbol{V}_i R\}_{i=1}^N$. To facilitate equivariance

---

[1]For ease of notation, in what follows we will remove the layer index d whenever it is clear from context. Keep in mind that the operations we introduce are per-layer.
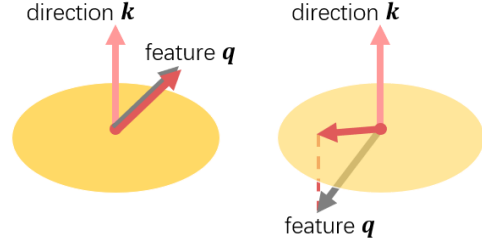

Figure 3: **Non-linearity** – Our non-linearity generalizes ReLU by acting on vectors rather than scalar inputs, and is parametric with respect to a learned direction $\boldsymbol{k}$: (left) when the input feature $\boldsymbol{q}$ lies in the half-space defined by $\boldsymbol{k}$, the feature stays unchanged; (right) when the input feature $\boldsymbol{q}$ lies in the half-space defined by $-\boldsymbol{k}$, the feature component in that half-space is clipped. See equation 6.

in standard pointcloud network architectures, we construct VN layers following traditional designs via a combination of a linear map (Section 3.1) followed by a per-neuron non-linearity (Section 3.2). We additionally introduce equivariant pooling (Section 3.3) and normalization layers (Section 3.4). With these building blocks we are able to assemble a rich variety of complex neural networks in equivariance, including the most basic VN Multi-Layer Perceptron (VN-MLP) as a sequence of alternating linear and non-linear layers.

3.1. Linear layers – Figure 2

We begin by realizing the mapping f introduced in equation 1 as a linear operator – a fundamental module of neural networks. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{C' \times C}$, we define a linear operation $f_{\text{lin}}(\cdot; \mathbf{W})$ acting on a vector-list feature $\boldsymbol{V} \in \mathcal{V} \in \mathbb{R}^{N \times C \times 3}$ as follows:

$$\boldsymbol{V}' = f_{\text{lin}}(\boldsymbol{V}; \mathbf{W}) = \mathbf{W}\boldsymbol{V} \in \mathbb{R}^{C' \times 3}. \quad (3)$$

We verify that a rotation matrix $R \in \mathrm{SO}(3)$ commutes with this linear layer:

$$f_{\text{lin}}(\boldsymbol{V}R; \mathbf{W}) = \mathbf{W}\boldsymbol{V}R = f_{\text{lin}}(\boldsymbol{V}; \mathbf{W})R = \boldsymbol{V}'R, \quad (4)$$

yielding the desired equivariance property. Note that we omit a bias term as an addition of a constant vector that would interfere with equivariance. Further, note that while this layer is SO(3) equivariant, we can achieve SE(3) equivariance by centering $\boldsymbol{V}$ at the origin. Finally, depending on the setting, $\mathbf{W}$ may or may not be shared across the elements $\boldsymbol{V}$ of $\mathcal{V}$.

3.2. Non-linear layers – Figure 3 and Figure 4

Per-neuron non-linearity is key to the representation power of neural networks. As evident from recent literature, especially useful are functions that split the input domain
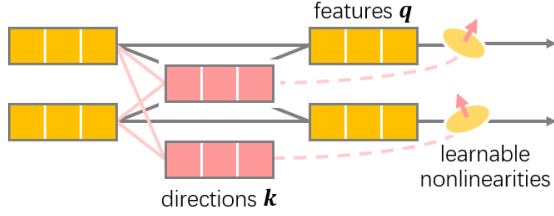
Figure 4: **Non-linear layer** – Our VN non-linearity is parametric in a learned direction $\boldsymbol{k}$, derived from the input features via a learnable linear layer.
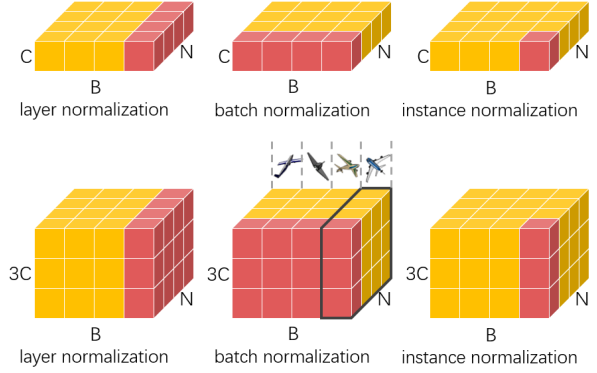

Figure 5: **Normalizations** – B, C, N stand for the batch, channel, and point dimensions respectively. (Top) classical scalar neurons; (bottom) vector neurons, batch normalization can only be done on vector norms as features within a batch are from different poses.

into two half spaces and map them differently (e.g. ReLU, leaky-ReLU, ELU, etc.). In the case of VN, a 3D version of these non-linearities, $\boldsymbol{V}' = f_{\mathrm{ReLU}}(\boldsymbol{V})$, is needed. Yet, committing to a fixed frame (i.e., one that does not depend on the input pose) like the standard coordinate system would violate equivariance. Instead, we propose to dynamically predict a direction from the input vector-list feature. We then generalize the classical ReLU by truncating the portion of a vector that points into the negative half-space of the learned direction.

More formally, given an input vector-list feature $\boldsymbol{V} \in \mathbb{R}^{C \times 3}$, for each output vector-neuron $\boldsymbol{v}' \in \boldsymbol{V}'$ we learn *two* weight matrices $\mathbf{W} \in \mathbb{R}^{1 \times C}$ and $\mathbf{U} \in \mathbb{R}^{1 \times C}$, linearly mapping the input feature $\boldsymbol{V}$ to a feature $\boldsymbol{q} \in \mathbb{R}^{1 \times 3}$ and a direction $\boldsymbol{k} \in \mathbb{R}^{1 \times 3}$:

$$\boldsymbol{q} = \mathbf{W}\boldsymbol{V}, \quad \boldsymbol{k} = \mathbf{U}\boldsymbol{V}. \tag{5}$$

We then define the output VN as:

$$\boldsymbol{v}' = \begin{cases} \boldsymbol{q} & \text{if } \langle \boldsymbol{q}, \boldsymbol{k} \rangle \geqslant 0 \\ \boldsymbol{q} - \left\langle \boldsymbol{q}, \frac{\boldsymbol{k}}{\|\boldsymbol{k}\|} \right\rangle \frac{\boldsymbol{k}}{\|\boldsymbol{k}\|} & \text{otherwise,} \end{cases} \tag{6}$$

resulting in an output vector-list: $f_{\mathrm{ReLU}}(\boldsymbol{V}) = [\boldsymbol{v}']_{c=1}^{C}$ [2].

As illustrated in Figure 3, $\boldsymbol{q}$ can be decomposed into two components: $\boldsymbol{q}_{\|}$ and $\boldsymbol{q}_{\perp}$ that are parallel and orthogonal to $\boldsymbol{k}$, respectively. Analogous to the standard scalar ReLU, we apply the nonlinear function to $\boldsymbol{q}_{\|}$ along the direction $\boldsymbol{k}$ by clipping $\boldsymbol{q}_{\|}$ to zero, while keeping $\boldsymbol{q}_{\perp}$ unchanged. Other types of split-case functions (e.g. leaky-ReLU) follow immediately from this definition. We discuss these and other types of non-linearities in the supplementary material.

It is easy to verify that f_{ReLU} is rotation equivariant. In particular, both $\boldsymbol{q}$ and $\boldsymbol{k}$ are linear maps of $\boldsymbol{V}$ and thus commute with a rotation matrix as discussed in (4). Moreover, the inner-product term in the second case would cancel out an orthogonal matrix $\langle \boldsymbol{q}\boldsymbol{R}, \boldsymbol{k}\boldsymbol{R} \rangle = \langle \boldsymbol{q}, \boldsymbol{k} \rangle$ resulting in a scalar multiplication of a $\boldsymbol{k}$, which is again equivariant.

---

[2] In practice, when computing for the unit direction vector $\boldsymbol{k}/\|\boldsymbol{k}\|$ we implement $\boldsymbol{k}/(\|\boldsymbol{k}\| + \varepsilon)$ with a small margin ε in the denominator to avoid division by zero at the origin

3.3. Pooling layers

Pooling is widely used when aggregating local/global neighbourhood information, either spatially (e.g. Point-Net++) or by feature similarity (e.g. DGCNN). While mean pooling is a linear operation that respects rotation equivariance, we also define a VN max pooling layer as a counterpart to the classical max pooling on scalars.

For global pooling, we are given a set of vector-lists $\mathcal{V} \in \mathbb{R}^{N \times C \times 3}$. We learn an element-wise signal of data dependent directions $\mathcal{K} \in \mathbb{R}^{N \times C \times 3}$. Similarly to Section 3.2, these directions are obtained via applying a weight matrix $\mathbf{W} \in \mathbb{R}^{C \times C}$ to each $\boldsymbol{V}_n \in \mathcal{V}$:

$$\mathcal{K} = \{\mathbf{W}\boldsymbol{V}_n\}_{n=1}^{N}, \tag{7}$$

and then computing the element of $\mathcal{V}$ that best aligns with $\mathcal{K}$ and selecting it as our global feature: for each channel $c \in [C]$,

$$f_{\mathrm{MAX}}(\mathcal{V})[c] = \boldsymbol{V}_{n^*}[c] \tag{8}$$
$$\text{where} \quad n^*(c) = \arg\max_n \langle \mathbf{W}\boldsymbol{V}_n[c], \boldsymbol{V}_n[c] \rangle. \tag{9}$$

where $\boldsymbol{V}_n[c]$ stands for the vector channel $\boldsymbol{v}_c \in \boldsymbol{V}_n$.

Similarly, we can aggregate information locally (local pooling) by grouping k nearest neighbours in $\mathcal{V}$ and perform the aforementioned pooling seperately for each group.

3.4. Normalization layers – Figure 5

Normalization often give rise to significant performance improvements. Layer [3] and instance normalizations [36] are done pre-sample (and the latter also per channel) and thus can be trivially generalized to VN networks, where the distributions are normalized with respect to vector distributions in $\mathbb{R}^3$.

Batch normalization. In contrast to other forms of normalizations, batch normalization aggregates statistics across all batch samples. While technically possible, in the context of rotation equivariant networks, averaging across arbitrarily rotated inputs would not necessarily be meaningful. For example, averaging two input features rotated in opposite directions would zero them out instead of producing that feature in a canonical pose.

We instead apply batch normalization to the *invariant* component of the vector-list features, by normalizing the 2-norms of the vector-list features.

Given a batch of B vector-list features $\{V_b\}_{b=1}^B$ with each $V_b \in \mathbb{R}^{C \times 3}$, our batch normalization is defined as:

$$N_b = \text{ElementwiseNorm}(V_b) \in \mathbb{R}^{N \times 1} \quad (10)$$

$$\{N_b'\}_{b=1}^B = \text{BatchNorm}\left(\{N_b\}_{b=1}^B\right) \quad (11)$$

$$V_b'[c] = V_b[c]\,\frac{N_b'[c]}{N_b[c]}, \quad \forall\, c \in [C], \quad (12)$$

where $V_b'[c], V_b[c]$ are the vector channels, $N_b'[c], N_b[c]$ are their scalar 2-norms, and $\text{ElementWiseNorm}(V_b)$ computes the 2-norm of every vector channel $v_c = V_b[c] \in V_b$.

3.5. Invariant layers

General invariant architectures are comprised of equivariant layers followed by invariant ones. We now introduce our invariant layer, that can be appended as needed to the output of the equivariant VN layers. Rotation-invariant networks are essential for both classification and segmentation tasks, where the identity of an object or its parts should be invariant to pose.

Key to our approach is the idea that the product of an equivariant signal $V \in \mathbb{R}^{C \times 3}$ by the transpose of an equivariant signal $T \in \mathbb{R}^{C' \times 3}$ is rotation invariant:

$$(VR)(TR)^\top = VRR^\top T^\top = VT^\top. \quad (13)$$

Note that a specific case of (13) is the inner product of two vectors, in particular the norm of equivariant vector features is rotation invariant.

We could compute an invariant feature from a vector-list $V \in \mathbb{R}^{C \times 3}$ as the Gram matrix $VV^\top$. However, this would result in a large $O(C^2)$ storage complexity. We could also consider taking the norm of each row of V but this would result in the loss of the relative directional information between the rows. Instead we propose a scalable solution with more manageable $O(C)$ complexity that can preserve directional information.

Our idea is to produce a coordinate system $T \in \mathbb{R}^{3 \times 3}$ from V and read V in this coordinate system thus producing rotation invariant features. In practice we consider our usual set of equivariant vector-list feature $\mathcal{V} \in \mathbb{R}^{N \times C \times 3}$. Inspired by Maron et al. [23], we produce a matrix T_n for

each element by concatenating its feature $V_n \in \mathbb{R}^{C \times 3}$ with the global mean $\overline{V} := \frac{1}{N}\sum_n V_n \in \mathbb{R}^{C \times 3}$ and running through a vector neuron MLP with a target number of channel $C' = 3$:

$$T_n := \text{VN-MLP}([V_n, \overline{V}]) \quad (14)$$

Finally we define our invariant layer by:

$$\text{VN-In}(V_n) := V_n T_n^\top. \quad (15)$$

4. Network Architectures

We now show how we can plug vector neurons into two widely used 3D learning architectures, PointNet [27] and DGCNN [38]. These two backbones are representative of the richness of pointcloud networks, as PointNet is free from convolutions, and DGCNN comprises convolutions but the message passing is on dynamic graphs whose edges are not directly embedded in $\mathbb{R}^3$. As we show next, VN networks fit well into these backbones, while previous convolution-based methods such as TFN [35] and EGCL [31] do not. To make clear the ease of generalization, in the following we will adopt a naming convention to the functions defined in section 3 by using a "VN" prefix.

VN-DGCNN. DGCNN performs a permutation equivariant edge convolution by computing adjacent edge features e'_{nm} followed by a local max pooling:

$$e'_{nm} = \text{ReLU}(\Theta(x_m - x_n) + \Phi x_n) \quad (16)$$

$$x'_n = \text{Pool}_{m:(n,m)\in\mathcal{E}}(e'_{nm}), \quad (17)$$

where $x_n \in \mathbb{R}^3$ are per-point features and Θ, Φ are learnable weight matrices. Our VN-DGCNN requires a straightforward modification:

$$E'_{nm} = \text{VN-ReLU}(\Theta(V_m - V_n) + \Phi V_n) \quad (18)$$

$$V'_n = \text{VN-Pool}_{m:(n,m)\in\mathcal{E}}(E'_{nm}) \quad (19)$$

using our vector-list representation $V_n \in \mathbb{R}^{C \times 3}$.

VN-PointNet. PointNet approximates a permutation symmetric function using

$$x' = \text{Pool}_{x_n \in \mathcal{X}}(h(x_1), \cdots, h(x_N)), \quad (20)$$

where h is the same for all x_n. Its VN version is written as

$$V' = \text{VN-Pool}_{V_n \in \mathcal{V}}(f(V_1), \cdots, f(V_N)), \quad (21)$$

where f is a shared VN-MLP. One issue here exists in the first input layer where the input pointcloud coordinates V_i are $\mathbb{R}^{1 \times 3}$ vectors and thus applying f to them would degenerate to a set of $\mathbb{R}^{C \times 3}$ vector-lists whose vector components are all linearly dependent (pointing to one direction). This is analogous to applying a per-pixel 1×1 convolution to a gray-scale image (single input channel). Therefore, in VN-PointNet we add an edge convolution at the input layer, mapping $\mathbb{R}^{1 \times 3}$ features into $\mathbb{R}^{C \times 3}$ with $C > 1$ and then continue with per-point VN-MLP operations.

5. Experiments

We evaluate our method on three core tasks in pointcloud processing: classification (Section 5.1), segmentation (Section 5.2), and reconstruction (Section 5.3). In addition to their diversity in the required output, these tasks span different use cases of our proposed equivariant framework: classification and segmentation are rotation-invariant tasks, while reconstruction is rotation-equivariant.

Datasets. We employed the ModelNet40 [5] and the ShapeNet [5] datasets for evaluation. The ModelNet40 dataset consists of 40 classes with 12,311 CAD models in total. We used 9,843 models for training and the others for testing in the classification task. For the ShapeNet dataset, we followed [42] by using ShapeNet-part for part segmentation, which has 16 shape categories with more than 30,000 models. We also applied the subset of ShapeNet in [7] for shape reconstruction, containing 13 major categories with 50,000 models.

Train/test rotation setup. In classification and segmentation, following the conventions from Esteves et al. [12], we adopt three train/test settings: z/z, $z/\mathrm{SO}(3)$ and $\mathrm{SO}(3)/\mathrm{SO}(3)$, where z stands for data augmentation with rotations only around the z axis, and $\mathrm{SO}(3)$ for arbitrary rotations. All rotations are generated on the fly at the training time, thereby comparing the equivariance-by-construction of VN architectures with a *learned-by-augmentation* equivariance. At test time, each shape is presented at a single rotation. For reconstruction, we show results on extreme settings: no-rotation (I) – the standard evaluation setup for prior methods, and arbitrary rotations $\mathrm{SO}(3)$. Since outputs in this task are static and optimization for each shape takes multiple iterations at both train and test times, here the $\mathrm{SO}(3)$ random rotations are generated for each shape in a pre-processing stage and all shapes stay at fixed poses during training.

Network implementations. In classification and segmentation, we implement our VN networks in the identical architectures to their classical counterparts, but with each layer in the shape of $\lfloor \frac{N}{3} \rfloor \times 3$ while the corresponding layer in the scalar network has size N. This in fact greatly reduces the number of learnable parameters in VN networks, resulting in roughly $\leqslant 2/3^2 = 2/9$ times of parameters compared to the counterpart scalar networks – here the factor 2 in the numerator is because in nonlinearities two components q, k are both learned (Equation 5). In reconstruction we slightly extend the layer size for the VN encoder. Moreover, in VN-PointNet, we discard the input spatial transformation MLP which learns 3×3 transformation matrices as our VN network already takes rigid transformations into consideration by construction. In the following experiments, we use mean pooling as aggregation in all networks, which performed better in practice. We will discuss

more about the max pooling as well as ablation study on other structures in the supplementary material.

Methods	z/z	$z/\mathrm{SO}(3)$	$\mathrm{SO}(3)/\mathrm{SO}(3)$
Point / mesh inputs			
PointNet [27]	85.9	19.6	74.7
DGCNN [38]	90.3	33.8	88.6
VN-PointNet	77.5	77.5	77.2
VN-DGCNN	89.5	**89.5**	**90.2**
PCNN [2]	92.3	11.9	85.1
ShellNet [44]	**93.1**	19.9	87.8
PointNet++ [28]	91.8	28.4	85.0
PointCNN [21]	92.5	41.2	84.5
Spherical-CNN [12]	88.9	76.7	86.9
a^3S-CNN [22]	89.6	87.9	88.7
SFCNN [29]	91.4	84.8	90.1
TFN [35]	88.5	85.3	87.6
RI-Conv [43]	86.5	86.4	86.4
SPHNet [26]	87.7	86.6	87.6
ClusterNet [6]	87.1	87.1	87.1
GC-Conv [45]	89.0	89.1	89.2
RI-Framework [19]	89.4	89.4	89.3
Point + normal inputs			
SFCNN [29]	92.3	85.3	91.0
LGR-Net [46]	90.9	90.9	91.1

Table 1: Test classification accuracy on the ModelNet40 dataset [41] in three train/test scenarios. z stands for aligned data augmented by random rotations around the vertical axis and $\mathrm{SO}(3)$ indicates data augmented by random rotations.

5.1. Classification – Table 1

We evaluate classification results on ModelNet40 compared with vanilla PointNet, DGCNN, and other rotation invariant or equivariant methods which takes point coordinates (meshes or pointclouds) as inputs. Compared with their non-equivariant counterparts, the VN networks attain consistently good results on all the three settings, which indicates their robustness over rotations, especially in the $z/\mathrm{SO}(3)$ case where the test set contains unseen rotations in the train set. Even in the $\mathrm{SO}(3)/\mathrm{SO}(3)$ case with abundant train-time data augmentation, the rotation sensitive networks cannot perform as well as the equivariance by construction in VN networks. One the other hand, our VN network with DGCNN backbone (VN-DGCNN) outperforms all other equivariant or invariant methods with only point coordinate inputs in the $z/\mathrm{SO}(3)$ and $\mathrm{SO}(3)/\mathrm{SO}(3)$ cases. Note that methods that use surface normals [29, 46] still achieve better slightly better results.

Methods	z/SO(3)	SO(3)/SO(3)
Point / mesh inputs		
PointNet [27]	38.0	62.3
DGCNN [38]	49.3	78.6
VN-PointNet	72.4	72.8
VN-DGCNN	**81.4**	**81.4**
PointCNN [21]	34.7	71.4
PointNet++ [28]	48.3	76.7
ShellNet [44]	47.2	77.1
RI-Conv [43]	75.3	75.3
TFN [35]	76.8	76.2
GC-Conv [45]	77.2	77.3
RI-Framework [19]	79.2	79.4
Point + normal inputs		
LGR-Net [46]	80.0	80.1

Table 2: ShapeNet part segmentation. The results are reported in overall average category mean IoU over 16 categories in two train/test scenarios. With z, we refer to data augmented only by random rotations around the vertical axis, and SO(3) indicates random rotations.

5.2. Part segmentation – Table 2

Table 2 shows our results in ShapeNet part segmentation. Again our method shows consistent results across different rotations and achieves best performance with VN-DGCNN compared with other works, including [46] that uses surface normals in addition to the point coordinates.

5.3. Neural implicit reconstruction – Table 3

We follow the pointcloud *completion* experiment from OccNet [24], where we reconstruct neural implicit functions from sparse and noisy input pointclouds: we subsample 300 points from the surface of each (water-tight) ShapeNet model, and perturb them with normal noise with zero mean and 0.005 standard deviation. The outputs are occupancy probability functions $\mathcal{O} : \mathbb{R}^3 \to [0, 1]$ which can be parameterized by a shared neural implicit function $h_\theta(\cdot \mid z) : \mathbb{R}^3 \to [0, 1]$ conditioned by a latent code z derived from the input point set. For fair comparisons, we retrain the original OccNet [24] together with our methods for 300k iterations, and select the models with the best performance on the validation set.

Encoder network. We build an encoder-decoder framework with the architecture similar to [24] but in the language of VN. The encoder is rotation equivariant, encoding a pointcloud $\{x_1, x_2, \cdots, x_N\}$ into a global vector-list feature $Z \in \mathbb{R}^{C \times 3}$. While in Mescheder et al. [24] the encoder

is a PointNet, here we use a VN-PointNet:

$$Z = \text{VN-PointNet}(\{x_1, x_2, \cdots, x_N\}). \quad (22)$$

Decoder network. The decoder is rotation invariant between vector-list latent code $Z \in \mathbb{R}^{C \times 3}$ and query point coordinate $x \in \mathbb{R}^3$ – if the shape and the query point are simultaneously rotated, the occupancy value stays unchanged. We define the decoder as a function on the three invariant features $\|x\|^2, \langle x, Z \rangle, \text{VN-In}(Z)$:

$$\mathcal{O}(x \mid Z) = \text{ResNet}([\langle x, Z \rangle, \|x\|^2, \text{VN-In}(Z)]), \quad (23)$$

where $\text{VN-In}(\cdot)$ is the VN invariant layer defined in Section 3.5. As an ablation study, we also replace VN-PointNet with a standard PointNet encoder (with the same invariant decoder), where the encoder generates latent codes $z \in \mathbb{R}^C$ and we reshape them into $Z \in \mathbb{R}^{(C/3) \times 3}$.

On the contrary, the decoder in [24] is a simple nonlinear function $\mathcal{O}(x \mid z) = h(\varphi(x), \psi(z))$, which given latent code $z \in \mathbb{R}^C$ and query point $x \in \mathbb{R}^3$ outputs an occupancy probability $\mathcal{O} \in [0, 1]$ [3].

Quantitative results – Table 3. We evaluate the volumetric mean IoU of the reconstructions in the three train/test settings. Compared with the original OccNet, our equivariant-encoder/invariant-decoder exhibits excellent coherence in reconstructing shapes in any poses, with a minor loss in accuracy in the I/I case. Even simply adopting the invariant decoder without an equivariant encoder slightly improves the performances in all the three settings.

Qualitative results – Figure 6. We show some reconstructions from the test set using the original OccNet (light pink) and our VN-OccNet (yellow). Meshes are extracted from the neural implicits using the Multi-resolution IsoSurface Extraction (MISE) method from Mescheder et al. [24]. Although OccNet better recognizes the fine details in the I/I case when the dataset is pre-aligned, it is extremely sensitive to rotations. In the I/SO(3) case when unseen rotations are applied at test time, OccNet totally fails by hardly learning anything meaningful; these findings are also consistent with those in Deng et al. [9]. Even in the SO(3)/SO(3) case when data augmentation is adopted at train time, it still shows its limitation by generating blurry shapes (top left), averaged shapes (top right, the box-like output consists of sofa features averaged from different poses), or shapes with incorrect priors (bottom right, a shape in the car class is falsely identified as a chair).

6. Conclusions

We have introduced Vector Neurons – a novel framework that facilitates rotation equivariant neural networks by lifting standard neural network representations to 3D space. To

---

[3][24] provides multiple versions of decoders. We select this simplest one in our experiments for easier comparisons.

Methods	Encoder	Latent code	Decoder	I/I	I/SO(3)	SO(3)/SO(3)
OccNet [24]	PointNet	$z \in \mathbb{R}^C$	$h(\varphi(x), \psi(z))$	71.4	30.9	58.2
-	PointNet	$Z \in \mathbb{R}^{\lfloor C/3 \rfloor \times 3}$	$h(\langle x, Z \rangle, \|x\|^2, \text{VN-In}(Z))$	**72.0**	31.0	59.4
VN-OccNet	VN-PointNet	$Z \in \mathbb{R}^{\lfloor C/3 \rfloor \times 3}$	$h(\langle x, Z \rangle, \|x\|^2, \text{VN-In}(Z))$	69.3	**69.3**	**68.8**

Table 3: Volumetric mIoU on ShapeNet reconstruction with neural implicits. We show results on extreme settings: no-rotation (I) – the standard evaluation setup for prior methods, and arbitrary rotations SO(3). Here the SO(3) random rotations are generated for each shape in a pre-processing stage and all shapes stay at fixed poses during training.
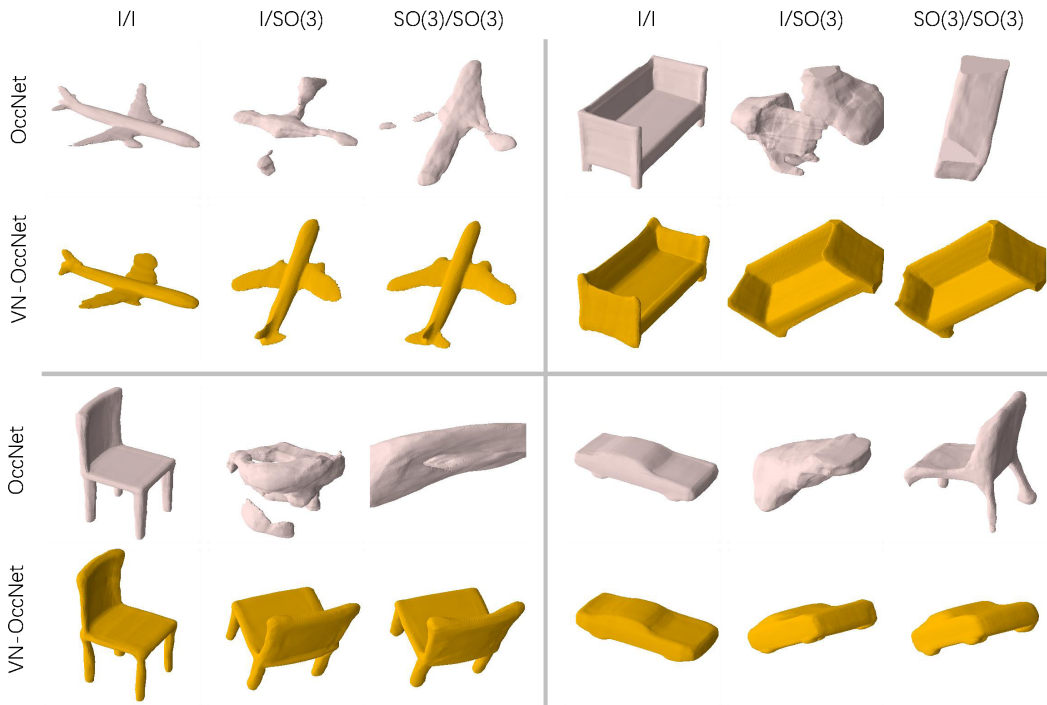

Figure 6: Reconstruction results on ShapeNet with OccNet (light pink) and VN-OccNet (yellow). Meshes are extracted from the neural implicits using the Multi-resolution IsoSurface Extraction (MISE) method.

that end, we have introduced the vector-neuron counterpart of standard network modules: linear layers, non-linearities, pooling and normalization. Using our framework, we have built a rotation-equivariant version of two leading point-cloud network backbones: PointNet and DGCNN, and evaluated them on 3 tasks: classification, segmentation and reconstruction. Our results demonstrate a consistent advantage to our modified architecture when the input shapes pose is arbitrary, compared to an augmentation based approach.

Limitation and future work. While our method shines under arbitrary rotation settings, on aligned input shapes and specifically in the task of reconstruction, our VN-OccNet was not able to match the reconstruction quality of vanilla OccNet by a small margin. In future works we plan to investigate this matter.

In this work, we have focused on 3D pointcloud net-works, yielding permutation and rotation equivariant architectures. However, it should be clear that our framework has obvious generalizations to higher-dimensional pointclouds in a completely analogous way. Generalization of vector neurons to other transformation groups of interest, such as the full affine group, can also be investigated (the addition of uniform scalings in our framework is quite straightforward).

In summary, by making rotation equivariant modules simple and accessible we hope to alleviate the need to curate and pre-align shapes for supervision and inspire future research on this fascinating topic.

References

[1] Brandon Anderson, Truong-Son Hy, and Risi Kondor. Cormorant: Covariant molecular neural networks. *arXiv preprint arXiv:1906.04015*, 2019. 2

[2] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*, 2018. 2, 6

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. 4

[4] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6D object pose estimation using 3D object coordinates. In *European Conference on Computer Vision*, pages 536–551, 2014. 3

[5] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], 2015. 6

[6] 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 *Conference on Computer Vision and Pattern Recognition*, pages 4994–5002, 2019. 2, 6

[7] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conference on Computer Vision*, pages 628–644, 2016. 6

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

[9] Boyang Deng, JP Lewis, Timothy Jeruzalski, Gerard Pons-Moll, Geoffrey Hinton, Mohammad Norouzi, and Andrea Tagliasacchi. Nasa: Neural articulated shape approximation, 2020. 7

[10] Emilien Dupont, Miguel Bautista Martin, Alex Colburn, Aditya Sankar, Josh Susskind, and Qi Shan. Equivariant neural rendering. In *International Conference on Machine Learning*, pages 2761–2770. PMLR, 2020. 3

[11] Nadav Dym and Haggai Maron. On the universality of rotation equivariant point cloud networks, 2020. 2

[12] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) equivariant representations with spherical cnns. In *European Conference on Computer Vision*, pages 52–68, 2018. 2, 6

[13] Carlos Esteves, Yinshuang Xu, Christine Allen-Blanchette, and Kostas Daniilidis. Equivariant multi-view networks. In *International Conference on Computer Vision*, pages 1568–1577, 2019. 2

[14] Fabian B Fuchs, Daniel E Worrall, Volker Fischer, and Max Welling. Se(3)-transformers: 3d roto-translation equivariant attention networks. *arXiv preprint arXiv:2006.10503*, 2020. 1, 2

[15] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International conference on artificial neural networks*. Springer, 2011. 3

[16] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015. 1

[17] Risi Kondor, Zhen Lin, and Shubhendu Trivedi. Clebsch-gordan nets: a fully fourier space spherical convolutional neural network. *arXiv preprint arXiv:1806.09231*, 2018. 2

[18] Leon Lang and Maurice Weiler. A wigner-eckart theorem for group equivariant convolution kernels. *arXiv preprint arXiv:2010.10952*, 2020. 2

[19] 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. *arXiv preprint arXiv:2003.07238*, 2020. 2, 6, 7

[20] Xiaolong Li, He Wang, Li Yi, Leonidas J Guibas, A Lynn Abbott, and Shuran Song. Category-level articulated object pose estimation. In *Conference on Computer Vision and Pattern Recognition*, pages 3706–3715, 2020. 3

[21] 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. 2, 6, 7

[22] Min Liu, Fupin Yao, Chiho Choi, Ayan Sinha, and Karthik Ramani. Deep learning 3d shapes using alt-az anisotropic 2-sphere convolution. In *International Conference on Learning Representations*, 2018. 2, 6

[23] Haggai Maron, Or Litany, Gal Chechik, and Ethan Fetaya. On learning sets of symmetric elements. In *International Conference on Machine Learning*, pages 6734–6744, 2020. 5

[24] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 7, 8

[25] Adrien Poulenard and Leonidas J Guibas. A functional approach to rotation equivariant non-linearities for tensor field networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13174–13183, 2021. 2

[26] Adrien Poulenard, Marie-Julie Rakotosaona, Yann Ponty, and Maks Ovsjanikov. Effective rotation-invariant point cnn with spherical harmonics kernels. In *IEEE International Conference on 3D Vision*, pages 47–56, 2019. 2, 6

[27] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 1, 2, 5, 6, 7

[28] 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. 2, 6, 7

[29] Yongming Rao, Jiwen Lu, and Jie Zhou. Spherical fractal convolutional neural networks for point cloud recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 452–460, 2019. 2, 6

[30] Helge Rhodin, Mathieu Salzmann, and Pascal Fua. Unsupervised geometry-aware representation for 3d human pose estimation. In *European Conference on Computer Vision*, pages 750–767, 2018. 3

[31] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. *arXiv preprint arXiv:2102.09844*, 2021. 5

[32] Wen Shen, Binbin Zhang, Shikun Huang, Zhihua Wei, and Quanshi Zhang. 3d-rotation-equivariant quaternion neural networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*, pages 531–547. Springer, 2020. 2

[33] Weiwei Sun, Wei Jiang, Andrea Tagliasacchi, Eduard Trulls, and Kwang Moo Yi. Attentive context normalization for robust permutation-equivariant learning. In *Conference on Computer Vision and Pattern Recognition*, 2020. 1, 3

[34] Weiwei Sun, Andrea Tagliasacchi, Boyang Deng, Sara Sabour, Soroosh Yazdani, Geoffrey Hinton, and Kwang Moo Yi. Canonical capsules: Unsupervised capsules in canonical pose, 2020. 3

[35] 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, 5, 6, 7

[36] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017. 4

[37] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J Guibas. Normalized object coordinate space for category-level 6D object pose and size estimation. In *Conference on Computer Vision and Pattern Recognition*, pages 2642–2651, 2019. 3

[38] 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*, 38(5):1–12, 2019. 2, 5, 6, 7

[39] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. *arXiv preprint arXiv:1807.02547*, 2018. 2

[40] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Interpretable transformations with encoder-decoder networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5726–5735, 2017. 3

[41] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015. 6

[42] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics*, 35(6):1–12, 2016. 6

[43] Zhiyuan Zhang, Binh-Son Hua, David W Rosen, and Sai-Kit Yeung. Rotation invariant convolutions for 3d point clouds deep learning. In *IEEE International Conference on 3D Vision*, pages 204–213, 2019. 2, 6, 7

[44] Zhiyuan Zhang, Binh-Son Hua, and Sai-Kit Yeung. Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. In *International Conference on Computer Vision*, pages 1607–1616, 2019. 6, 7

[45] Zhiyuan Zhang, Binh-Son Hua, Wei Chen, Yibin Tian, and Sai-Kit Yeung. Global context aware convolutions for 3d point cloud understanding. *arXiv preprint arXiv:2008.02986*, 2020. 2, 6, 7

[46] 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, 7