SVNet: Where SO(3) Equivariance Meets Binarization on Point Cloud Representation

Zhuo Su[1,*] Max Welling[2] Matti Pietikäinen[1] Li Liu[3,1,†]
[1]Center for Machine Vision and Signal Analysis, University of Oulu, Finland
[2]AMLab, University of Amsterdam, Netherlands
[3]National University of Defense Technology, China
zhuo.su@oulu.fi, m.welling@uva.nl, {matti.pietikainen, li.liu}@oulu.fi

Abstract

Efficiency and robustness are increasingly needed for applications on 3D point clouds, with the ubiquitous use of edge devices in scenarios like autonomous driving and robotics, which often demand real-time and reliable responses. The paper tackles the challenge by designing a general framework to construct 3D learning architectures with SO(3) equivariance and network binarization. However, a naive combination of equivariant networks and binarization either causes sub-optimal computational efficiency or geometric ambiguity. We propose to locate both scalar and vector features in our networks to avoid both cases. Precisely, the presence of scalar features makes the major part of the network binarizable, while vector features serve to retain rich structural information and ensure SO(3) equivariance. The proposed approach can be applied to general backbones like PointNet and DGCNN. Meanwhile, experiments on ModelNet40, ShapeNet, and the real-world dataset ScanObjectNN, demonstrated that the method achieves a great trade-off between efficiency, rotation robustness, and accuracy. The codes are available at https://github.com/zhuoinoulu/svnet.

1. Introduction

3D point cloud processing has become a popular topic in recent years, with its broad applications like autonomous driving, augmented reality, and robotics. Deep neural networks are the first choices for building high accuracy architectures to recognize point cloud data, though often with huge computational cost and memory storage. Nowadays, applications on edge devices need more running efficiency and model compactness, meanwhile, rotation robustness, to deal with unseen environments with arbitrary poses in 3D data. Numerous efforts have been taken in tackling the challenge in either model efficiency by leveraging network binarization [32], which has the charming advantages of up to 32× and 64× reduction in memory storage and inference speed, respectively [33], or rotation robustness by manipulating geometric features [51, 8, 50]. We believe it can be solved within a single framework.

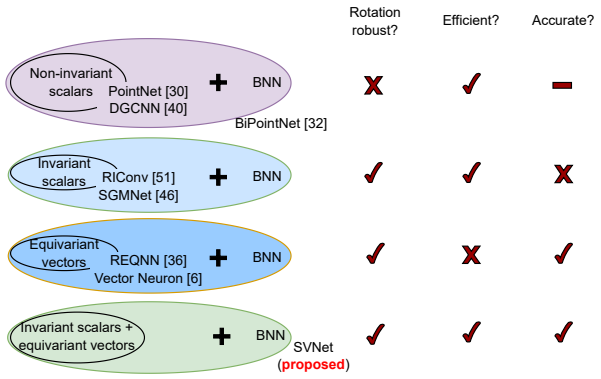


Figure 1. Binarization on different types of architectures.

We start by discussing a naive combination with SO(3) invariant networks and network binarization by taking advantage of the latest approaches on both sides (Fig. 1, Tab. 6). Generally, rotation-invariant networks utilize pose-preserving features during inference, which can be the rotation invariant ones [51, 46, 22, 1] in form of scalar geometric attributes like vector norms and angles, or the rotation equivariant ones [6, 36] with vectors like coordinates or directions. On one hand, directly binarizing invariant scalar features and model weights still preserves rotation invariance and gives great speedup via replacing floating-point multiplications to the cheap XNOR-Count operations (or binary operations), while this combination may cause inevitable loss of geometric information [22, 46], leading to unsatisfactory prediction accuracy. On the other hand, as binarizing equivariant vectors destroys rotation equivari-

*The work was done when visiting AMLab, University of Amsterdam.
† Correspondence to: Li Liu - http://lilyliliu.com.

ance, merely binarizing weights and leaving those vectors unchanged gives another viable but sub-optimal option as the expensive floating-point multiplications are converted to additions, instead of the cheaper binary operations.

Motivated by the recent works on molecular representation learning [17, 16, 35, 34], where both invariant scalar features and equivariant vector features are updated and the two interact with each other during inference (in the rest of the paper, we use "scalar" and "vector" to represent "invariant scalar feature" and "equivariant vector feature" respectively for simplicity), we propose to build the 3D point cloud learning architectures in a similar way, namely, with dual use of scalars and vectors in feature updating. Despite the similar form, our motivation is different. For molecular processing, scalar and vector features naturally exist in data like bond types, electronegativity of the atoms (scalars), and atom orientations, edge directions (vectors), making it more intuitive to update both types of features. While it is not the case in 3D point clouds where usually only the coordinates (vectors) of points are given[1]. Consequently there is a lack of investigation where both are used. We adopt scalars and vectors from the perspective of enhancing model efficiency when it comes to network binarization and rotation robustness: keeping vector features unchanged and binarizing scalars and model weights (Fig. 2). The setting of dual use makes a difference compared with the conventional sole use of scalars or vectors, to particularly construct binary rotation invariant networks for point clouds. Firstly, scalars make a major part of the network fully "binarizable", which is key to gain high efficiency, and conveniently introduce nonlinearity for the model during inference, which provides another benefit as it is nontrivial to introduce nonlinearity for a pure vector-based equivariant model [6, 29]. A certain amount of scalars on their own also enlarge the network capacity. Vector features, in contrast, makes an integral part to preserve the structural information without geometric ambiguities and ensure rotation equivariance. Although introducing a few addition operations from leaving vectors full-precision and binarizing weights, the vast majority of computations in the models are still made up of the cheapest binary operations due to the binarization of scalars. Finally, the invariance of the network is achieved by converting the equivariant vector features in the last layer to invariant ones.

Based on the scalar-vector architectures, we can naively adopt existing binarization algorithms to achieve both efficiency and rotation invariance. The derived networks are dubbed as SVNet. As a "by-product", we also found the full-precision versions of SVNets provided state-of-the-art performances among prior ones which handle complicated geometric attributes [1, 51, 22] or adopt pose disambiguation algorithms [48, 18, 53] for rotation invariance. It again indicates the potential of dual use of scalar and vectors for rotation-robust point cloud representation.

We summarize the contributions as follows:

1. We proposed the dual use of invariant scalar and equivariant vector features for building efficient binary and rotation robust networks for 3D point clouds.

2. We proposed a novel feature updating block for the case of dual use of scalars and vectors. The block is applicable to general backbones such as DGCNN [40] and PointNet [30], to enable high model efficiency and SO(3) equivariance.

3. We conducted extensive experiments on widely used datasets, proving that our method can achieve comparable results with the complexity significantly reduced in terms of both memory storage and computational cost. The full-precision versions of the derived architectures also obtained state-of-the-art performance compared with the latest methods.

2. Related Work

SO(3) symmetry on point clouds. Both rotation invariance and equivariance can achieve SO(3) symmetry for deep neural networks on point clouds. On one hand, works on the former can leverage invariant geometric attributes like vector norms, distances, and relative angles to extract features [1, 51, 22]. This kind of method often suffers from inevitable information loss caused by converting the vectors to scalars, where most of the positional information collapses. Another popular solution on invariance is to identify the canonical directions of the shape, via PCA [48, 18, 21, 50] or SVD [53]. Precisely predicting canonical directions can be well-tackled [9], this paradigm may suffer from the presence of potential geometric ambiguities which hinder their performances [21]. Clarification of possible ambiguities can also lead to linear growth in network complexity [21]. On the other hand, equivariant neural networks have been developed with the theory of group convolution [3, 4] and steerable filters [5, 41]. An arbitrary 3D input can be projected to a unit sphere, or regular 3D voxels, and processed with group convolution either spatially or spectrally. The spatial-based methods [43, 2] cannot easily guarantee equivariance on continuous groups and are often aided by extra data augmentation. In the spectral domain, the calculation of convolution involves the complicated Fourier transformation, and introducing nonlinearity becomes nontrivial [4, 8]. We can also take advantage of the equivariance-preserving steerable filters which can be pre-defined like spherical harmonics [37, 29] or learned [41, 42]. While the filter kernels are not binarizable to preserve equivariance. A more practical and

[1] invariant scalar features need to be manually created via geometric attributes like norms and inner products of vectors.

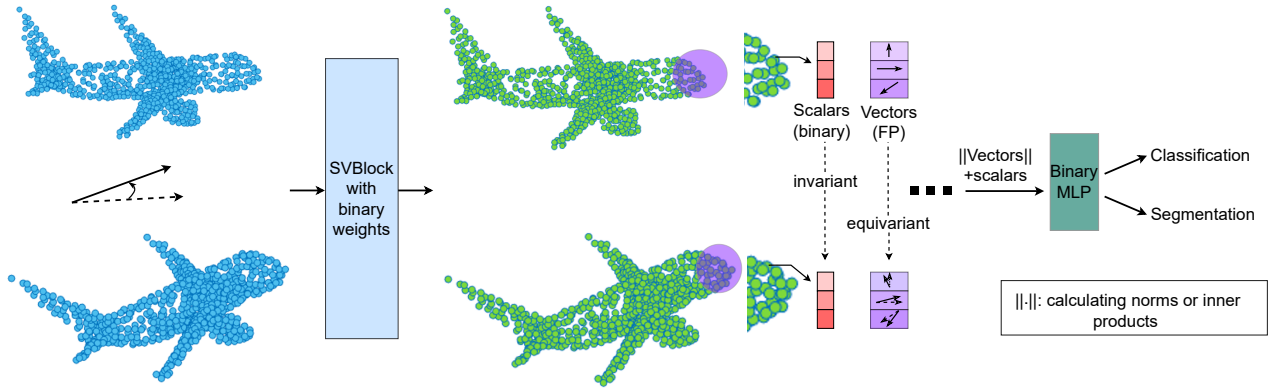


Figure 2. The overall structure of SVNet towards rigorous rotation invariance and model efficiency.

efficient way is vector mapping, where the three "coordinates" of vectors share the same weights/coefficients, as done in [6, 36]. Vector mapping also enjoys the advantage that the weights are equivariantly binarizable.

Binary neural networks (BNNs). BNNs were pioneered by Hubara *et al.* [14] for 2D image recognition using CNNs, where both activations and weights are binarized, thus achieving substantial network compression (up to 32× reduction in memory storage) and acceleration (up to 64× faster in inference speed [33]). The charming advantages of BNNs also introduced considerable attention for later researchers [33, 25, 54]. The most related work of ours in the BNN literature is BiPointNet [32], which was possibly the first binarization approach to deep learning on point clouds. BiPointNet conducted binarization on the rotation sensitive structures like PointNet, without analysis or solutions on SO(3) equivariance. In our method, once the network is made binarizable with SO(3) equivariance, it is compatible with most of the BNN methods in the literature.

Others. Our work also relates to the methods series on point clouds [30, 31, 40, 52, 23, 44]. Please refer to [11] for a more comprehensive review. At the same time, we are inspired by approaches on molecules like *geometric vector perceptrons* on molecular structures where both scalar and vector features were utilized [35, 17, 16].

3. Method

3.1. Problem statement and overall framework

Without loss of generality, we introduce two example tasks on point clouds. Suppose $\mathcal{O} = \{o_1, o_2, ..., o_n\}$ is a given unordered point set, with each point assigned with a 3D coordinate: $o_i \in \mathbb{R}^3$. For the classification task, a network functions as a map: $\mathcal{O} \to \mathbb{Z}$, where $\mathbb{Z}$ is an integer representing the category of $\mathcal{O}$. In a part segmentation task, the network then turns to a dense prediction function: $(\mathcal{O}, C) \to \mathbb{Z}^n$, where C is the category of $\mathcal{O}$, by assigning each point o_i to an integer that indicates which part of the original shape the point belongs to. The purpose of this paper is to make the network own the following properties:

SO(3) Equivariant: Any basic layer f^l in our network: $f^l(\mathcal{X}^l) = \mathcal{X}^{l+1}$, where l is the layer index, $\mathcal{X}^l$ and $\mathcal{X}^{l+1}$ represent the input and output of this layer respectively, should meet:

$$R_g^{l+1} \circ \mathcal{X}^{l+1} = f^l(R_g^l \circ \mathcal{X}^l) \quad \forall g \in G, \tag{1}$$

where R^{l+1} and R^l are the group representations of G (in our paper, $R^{l+1} = R^l$ is a 3×3 rotation matrix), and $R_g \circ \mathcal{X}$ a rotation g on $\mathcal{X}$ as G is the continuous SO(3) group. The above equation means that rotating the input first then passing it through the function f^l gives the same result if we first pass it through f^l and then rotate the output.

Specifically, in the proposed SVNet, $\mathcal{X}$ is formed as $(\mathcal{S}, \mathcal{V})$ with $\mathcal{S}$ and $\mathcal{V}$ representing scalar and vector features respectively. Rotation on $\mathcal{X}$ is defined as:

$$R_g \circ \mathcal{X} = R_g \circ (\mathcal{S}, \mathcal{V}) = (\mathcal{S}, R_g\mathcal{V}). \tag{2}$$

A stacking of equivariant convolutional layers will result in an equivariant network (supposing the index of the last equivariant layer is L), as

$$\begin{aligned} &f^L(...f^2(f^1(R_g^1 \circ \mathcal{X}^1))) = f^L(...f^2(R_g^2 \circ f^1(\mathcal{X}^1))) \\ &= f^L(...R_g^3 \circ f^2(f^1(\mathcal{X}^1))) = ... \\ &= R_g^{L+1} \circ f^L(...f^2(f^1(\mathcal{X}^1))). \end{aligned} \tag{3}$$

In SVNet, $\mathcal{V}^1$ can be created by using the original coordinates and relational positions between points. While $\mathcal{S}^1$ are generated from $\mathcal{O}$ via invariant geometric attributes (*e.g.*, norms). The whole network can be made SO(3) invariant by converting the output $\mathcal{V}^L$ to invariant ones with the same strategy.

Efficient: We take advantage of the BNNs to compose SVNet, where weights and scalar features are binarized,

such that most of the multiplication operations can be circumvented by instead using much more efficient addition and binary operations.

3.2. Design of SVNet

Given an input $\mathcal{X} = (\mathcal{S}, \mathcal{V})$ with $\mathcal{S} \in \mathbb{R}^{p\times\mathcal{N}}$ and $\mathcal{V} \in \mathbb{R}^{3\times q\times\mathcal{N}}$, where $\mathcal{N}$ is the number of nodes or edges and p (q) is the feature dimension (or the number of channels) for scalars (vectors), the core of SVNet is to design a "binarizable" convolutional layer where Eq. 1 holds.

Vector mapping Generally, the invariant $\mathcal{S}$ can be freely manipulated with existing linear and nonlinear functions (including binarization). To equivariantly update vectors features $\mathcal{V}$, we can build linear mapping functions along the feature dimension q and share the mapping along coordinates [36, 6].

As points are orderless, each $v \in \mathbb{R}^{3\times q}$ in $\mathcal{V}$ shares the same mapping functions for permutation equivariance. Supposing the weight matrix is $W \in \mathbb{R}^{q\times q'}$, the linear mapping on v is defined as:

$$v' = f_v(v; W) = vW, \tag{4}$$

transforming v to $v' \in \mathbb{R}^{3\times q'}$ (and sequentially, $\mathcal{V}$ to $\mathcal{V}' \in \mathbb{R}^{3\times q'\times\mathcal{N}}$). It can be easily proved that f_v is SO(3) equivariant: $R_g f_v(v; W) = R_g(vW) = (R_g v)W = f_v(R_g v; W)$. At the same time, binarization on W does not affect this equivariance.

SVBlock: the basic building block The design should also satisfy the following conditions. Firstly, proper nonlinear functions should be incorporated during the updating of $\mathcal{S}$ and $\mathcal{V}$ to make the network more discriminative. Secondly, $\mathcal{S}$ and $\mathcal{V}$ should interact with each other for better information fusion to strengthen network representation. The proposed SVBlock architecture is illustrated in Fig. 3, which we elaborate in detail as follows.

When updating scalar features, similar to [6], we first generate an equivariant coordinate system for each $v \in \mathcal{V}$ using vector mapping, and project v to invariant features:

$$v_c = vW_c; \tag{5}$$

$$v_{in} = v_c^T v, \tag{6}$$

where $W_c \in \mathbb{R}^{q\times 3}$. v_{in} is variant since $v_{in} = W_c^T(v^T v)$, which can also be interpreted as a linear transformation of the covariance matrix of v. The updating of $\mathcal{S}$ can be implemented by concatenation with the obtained $\mathcal{V}_{in}$, followed by a multi-layer perceptron (MLP) module and nonlinear functions. The process is shown in Fig. 3 (a).

When updating vector features, a problem we have to consider is that vector features $\mathcal{V}$ cannot be directly processed with nonlinear functions like ReLU or Sigmoid function. Although Vector Neurons [6] proposed equivariant direction learning and projection, we find this can be addressed more efficiently in our situation, by generating nonlinear re-weighting factors from scalar features. The factors are used to multiply with vector features. Hence this also introduces interactions between scalars and vectors.

Specifically, we again use vector mapping to linearly update $\mathcal{V}$, where the dimension of $\mathcal{V}$ changes from q to the target dimension q'. Next, $\mathcal{S}$ is compressed along $\mathcal{N}$ using average pooling, resulting in $\mathcal{S}_{com} \in \mathbb{R}^p$, which is then used to generate the re-weighting factors for the linearly updated vectors by MLP and Sigmoid, as shown in Fig. 3 (b).

The proposed SVBlock differs from *geometric vector perceptrons* [17, 16] with the presence of data-dependent coordinate systems (Eq. 5) for converting $\mathcal{V}$ to invariant features, and the way to generate re-weighting factors which is made lightweight with average pooling.

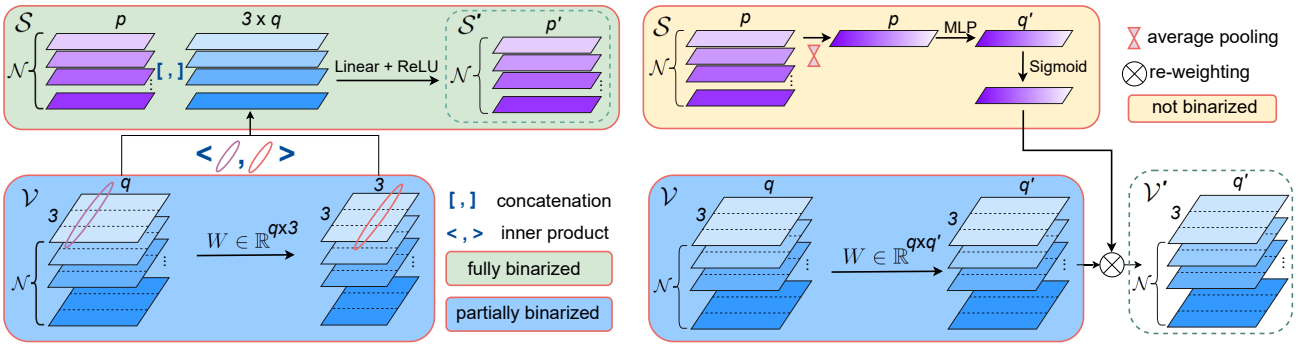
Instantiating SVNet SVNet is not a detailed network architecture but can be instantiated by plugging SVBlock into general backbones, making our method widely applicable. The pipeline of build SVNet is as follows.

1. Extract $(\mathcal{S}, \mathcal{V})$ for the first SVBlock: Given an unordered point set $\mathcal{O} = \{o_1, o_2, ..., o_n\}$ with $o_i \in \mathbb{R}^3$, we firstly find the k nearest neighboring points $\{o_{ij}|j = 1, 2, ..., k\}$ for each point o_i. The vector features of each (o_i, o_{ij}) pair is $v_{ij} = [o_i; o_{ij} - o_i] \in \mathbb{R}^{3\times 2}$. Based on that, we calculate the scalar features for the (o_i, o_{ij}) pair by putting v_{ij} in Eq. 5 and 6, getting $s_{ij} \in \mathbb{R}^6$ (by flattening $\mathbb{R}^{3\times 2}$ to $\mathbb{R}^6$). The vector features of o_i is $v_i \in \mathbb{R}^{3\times 2\times k}$, and of $\mathcal{O}$ is $\mathcal{V} \in \mathbb{R}^{3\times 2\times(k\times n)}$. Similarly, the scalar features of $\mathcal{O}$ is $\mathcal{S} \in \mathbb{R}^{6\times(k\times n)}$. We can regard $(k \times n)$ as a single dimension.

2. Aggregation: Aggregation of $\mathcal{S}$ or $\mathcal{V}$ can be done by pooling along the k dimension, resulting in $\mathcal{S} \in \mathbb{R}^{p\times n}$ and $\mathcal{V} \in \mathbb{R}^{3\times q\times n}$. In DGCNN, we can use the same strategy in 1 to extend the "k" dimension back (please also refer to the original paper [40]).

3. After the last SVBlock: $\mathcal{V}$ is converted to invariant features using Eq. 5 and 6 again, which is then concatenated with $\mathcal{S}$. The network ends with a binary MLP with nonlinear functions according to the backbone architecture. In this step, we can safely binarize both features and weights as there are only scalar features.

Binarization on SVNet It should be noted that the main focus of the paper is to propose an effective SO(3) equivariant framework from the perspective of network binarization. In other words, we enable SVNet to be compatible with previous BNN techniques. For simplicity, we binarize SVNet based on the following equations and use STE [12] for gradient calculation (please see Appendix A for a more detailed description). With a more powerful



(a) Updating Scalars (b) Updating Vectors

Figure 3. The SVBlock architecture. "Fully binarized" means both scalars and weights are binarized, "partially binarized" means keeping the features full-precision and binarizing the weights. The top right block is not binarized since it only introduces negligible computation due to average pooling.

	Eq. 5	Eq. 6	Scalar updating	Re-weighting factors	Vector updating
Vanilla block	-	-	$\mathcal{N}C_1C_2$ (MACs)	-	-
SVBlock	$\frac{3}{2}\mathcal{N}C_1$ (ADDs)	$\frac{3}{2}\mathcal{N}C_1$ (MACs)	$\frac{1}{2}\mathcal{N}C_1C_2$ (BOPs)	$\frac{1}{12}C_1C_2$ (MACs)	$\frac{1}{12}\mathcal{N}C_1C_2$ (ADDs)
Total ($C_1 = C_2 = 256, \mathcal{N} = 1024$)					
Vanilla block	67.1M MACs				
SVBlock (FP)	39.9M MACs				
SVBlcok	0.4M MACs + 6.0M ADDs + 33.6M BOPs				

Table 1. Computational cost of feature updating in a conventional network block (vanilla block) and SVBlock.

binarization method applied, the network can be further enhanced:

$$Y = \text{Sign}(X) \cdot \text{Sign}(W) \quad \text{if both } X \text{ and } W \text{ are binarized,} \tag{7}$$

$$Y = X \cdot \text{Sign}(W) \quad \text{if only } W \text{ is binarized,} \tag{8}$$

where W, X, and Y represents the weights, input, and output respectively, $\text{Sign}(a) = +1$ if $a \geq 0$ otherwise -1, "$\cdot$" means matrix multiplication.

In SVNet, Eq. 7 corresponds to the linear transformation of scalar features while Eq. 8 the vector mapping of vector features. The former can be implemented with BOPs (binary operations) and the latter with ADDs (additions), both are much more efficient than the original MACs (Multiply-Accumulates) in a full-precision counterpart.

Why is SVNet efficient? We give an example on comparing the computational cost between a vanilla network block, where only scalar features are used) in conventional methods [30, 40], and SVBlock, by analyzing the complexity of feature updating. For the former, the scalar features are updated with function: $\mathbb{R}^{C_1 \times \mathcal{N}} \rightarrow \mathbb{R}^{C_2 \times \mathcal{N}}$. To keep the same amount of features, SVBlock functions as: $(\mathbb{R}^{\frac{C_1}{2} \times \mathcal{N}}, \mathbb{R}^{3 \times \frac{C_1}{6} \times \mathcal{N}}) \rightarrow (\mathbb{R}^{\frac{C_2}{2} \times \mathcal{N}}, \mathbb{R}^{3 \times \frac{C_2}{6} \times \mathcal{N}})$. As shown in Tab. 1, the existence of both scalar and vector features makes the full-precision SVBlock (FP) more compact than a vanilla counterpart (39.9M MACs *vs.* 67.1M MACs). With binarization, most of the MACs is replaced with BOPs in SVBlock, which is the key to achieve better efficiency.

4. Experiments

In this section, we evaluate the proposed SVNet on the tasks of 3D object classification and part segmentation, based on the widely used ModelNet40 [45], ShapeNet [47], and ScanObjectNN [38] datasets. To show its rotation robustness and compare it with other methods, we follow the training/testing settings as in prior literature: z/z, z/SO3, and SO3/SO3, where z and SO3 mean "random rotation around z axis" and "random rotation" respectively. All the experiments were conducted with Pytorch library [28]. We adopt batch normalization [15] following [6, 36] and keep the first and last layer of SVNet full-precision to avoid severe information loss. Please also refer to Appendix B for a more challenging scenario where there was no rotation during training but was during testing.

4.1. Experiments on ModelNet40

ModelNet40 [45] has been extensively used for synthetic shape classification, including 12,311 CAD models with 40 man-made object categories (*e.g.*, airplane, bathtub, *etc.*). The dataset was split into 9,843 models for training and 2,468 for testing. To make a fair comparison with prior methods, we randomly extracted 1,024 3D points

	Method	Binarized	z/z	z/SO(3)	SO(3)/SO(3)
Rotation sensitive	PointNet* [30]	✗	85.9	17.0	74.7
	DGCNN [40]	✗	90.3	33.8	88.6
	PointNet++ [31]	✗	91.8	28.4	85.0
	PointCNN [23]	✗	92.5	41.2	84.5
	ShellNet [52]	✗	93.1	19.9	87.8
	BiPointNet* [32]	✓	39.9	13.7	16.6
Rotation invariant	RIConv [51]	✗	86.5	86.4	86.4
	ClusterNet [1]	✗	87.1	87.1	87.1
	Yu *et al.* [48]	✗	89.2	89.2	89.2
	RI-GCN [18]	✗	89.5	89.5	89.5
	GC-Conv [50]	✗	89.0	89.1	89.2
	Li *et al.* [22]	✗	89.4	89.4	89.3
	SGMNet [46]	✗	90.0	90.0	90.0
	Li *et al.* [21]	✗	90.2	90.2	90.2
Rotation equivariant	Spherical CNNs [8]	✗	88.9	78.6	86.9
	α^3SCNN [24]	✗	-	-	88.7
	Poulenard *et al.* [29]	✗	90.5	88.2	89.3
	VN-DGCNN [6]	✗	89.5	89.5	90.2
Rotation equivariant (**Proposed**)	SVNet-PointNet (FP)	✗	86.3	86.3	86.6
	SVNet-DGCNN (FP)	✗	90.3	90.3	90.0
	SVNet-PointNet	✓	76.3	76.3	75.8
	SVNet-DGCNN	✓	83.8	83.8	83.8
	SVNet-DGCNN†	✓	86.8	86.8	86.8

Table 2. Comparison on ModelNet40. † indicates a two-step training scheme. "∗" indicates our implementations using the code provided by the authors. Numbers show overall accuracies (%).

from each model for both training and testing. SVNet was instantiated with DGCNN [40] and PointNet [30] backbones. Following [6], it was trained for 250 epochs for the DGCNN backbone with a cosine annealing learning rate [26], and 200 epochs for the PointNet backbone with a multi-step annealing learning rate by decaying ($\times 0.7$) in every 20 epochs. For both cases, the learning rate was initialized as 0.001 and decayed towards 0, batch size was 32, and the Adam optimizer [19] was adopted. We also trained a full-precision version of our model, namely SVNet (FP) for each backbone, for a better comparison.

In Tab. 2, SVNet was compared with prior state-of-the-art methods including the rotation-sensitive, rotation-invariant, and rotation-equivariant ones. We had two main findings. Firstly, the full-precision version SVNet (FP) achieves 90.3% accuracy, surpassing the latest Vector Neurons [6] with 89.5% accuracy via the sole use of vectors, and SGMNet [46] with 90.0% accuracy via pure scalars, validating the effectiveness of the proposed scalar-vector configuration. Secondly, when binarized, the complexity of the network is significantly reduced in both memory and computational cost (Tab. 4). Nevertheless, SVNet still retains rigorous rotation invariance with competitive prediction performance. For instance, with our vanilla binarization, it achieves 83.8% accuracy regardless of the training/testing settings, *vs.* 39.9%, 13.7%, and 16.6% by BiPointNet [32]. In addition, with a more powerful binarization technique[2], SVNet achieves 86.8% accuracy, being highly comparable with most of the prior works which adopted full-precision weights and activations.

	Method	Binarized	z/SO(3)	SO(3)/SO(3)
Rotation sensitive	PointNet [30]	✗	41.8	62.3
	DGCNN [40]	✗	49.3	78.6
	PointNet++ [31]	✗	48.3	76.7
	PointCNN [23]	✗	34.7	71.4
	ShellNet [52]	✗	47.2	77.1
	BiPointNet [32]	✓	31.4	36.0
Rotation invariant	RIConv [51]	✗	75.3	75.5
	RI-GCN [18]	✗	77.2	77.3
	Li *et al.* [22]	✗	79.2	79.4
	SGMNet [46]	✗	79.3	79.3
	Li *et al.* [21]	✗	81.7	81.7
Rotation equivariant	Poulenard *et al.* [29]	✗	78.1	78.2
	VN-DGCNN [6]	✗	81.4	81.4
Rotation equivariant (**Proposed**)	SVNet-PointNet (FP)	✗	78.2	78.6
	SVNet-DGCNN (FP)	✗	<u>81.4</u>	<u>81.4</u>
	SVNet-PointNet	✓	67.3	67.3
	SVNet-DGCNN	✓	68.4	68.9
	SVNet-DGCNN†	✓	71.5	71.5

Table 3. Comparison on ShapeNet. Numbers show results on mean intersection of union (mIoU, %) over all classes. Underline numbers in our submission version were both 80.9 as we incidentally used label smoothing, which were unnecessary.

4.2. Experiments on ShapeNet

We used the ShapeNet part dataset [47] to evaluate our method on the part segmentation task. The dataset consists of 16,881 shapes with annotations of 50 parts in total, from 16 categories. It was split into 14,007 and 2,874 shapes for training and testing respectively. Different from ModelNet40, we utilized 2,048 points for each shape during training and testing. Again, we built SVNet via DGCNN and PointNet backbones. For both cases, SVNet was trained for 200 epochs with batch size of 32, using Adam optimizer [19] with an initial learning rate of 0.001. The learning rate was decayed to 0 with cosine annealing [26] for DGCNN and multi-step annealing with step size of 20 and decaying rate of 0.5 for PointNet.

The experimental results and comparison with prior state-of-the-art methods are given in Tab. 3. Similar to 4.1, we also give a comparison of network complexity in Tab. 4. We can get the consistent observations as in 4.1: the full-precision SVNet with DGCNN backbone achieves comparable results beating most of the prior rotation invariant/equivariant methods; equipped with SVBlock, the original backbones can be effectively improved in terms of

[2]A two-step binarization, by firstly training the network with full-precision weights and activations, then training with binary weights and activations.

	Method	ModelNet40 Params	MACs	ADDs	BOPs	z/SO(3)	ShapeNet Params	MACs	ADDs	BOPs	z/SO(3)
DGCNN	Original [40]	57.7M	2.4B	0	0	33.8	46.7M	4.4B	0	0	49.3
	Vector Neurons [6]	92.8M	3.2B	0	0	89.5	41.8M	6.6B	0	0	81.4
	SVNet (FP)	49.7M	1.4B	0	0	90.3	43.2M	7.2B	0	0	81.4
	SVNet	**3.4M**	**0.05B**	**0.2B**	**1.2B**	**83.8**	**4.0M**	**0.2B**	**1.0B**	**6.0B**	**68.4**
PointNet	Original [30]*	111.1M	0.4B	0	0	17.0	267.0M	5.8B	0	0	41.8
	Vector Neurons* [6]	63.1M	2.0B	0	0	85.6	162.6M	20.5B	0	0	79.8
	SVNet (FP)	78.8M	1.5B	0	0	86.3	234.8M	14.2B	0	0	78.2
	BiPointNet* [32]	4.2M	0.01B	0	0.4B	13.7	9.0M	0.1B	0	5.7B	31.4
	SVNet	**8.7M**	**0.03B**	**0.2B**	**1.2B**	**76.3**	**14.0M**	**0.2B**	**0.2B**	**13.8B**	**67.3**
	SVNet-small	**4.3M**	**0.02B**	**0.03B**	**0.2B**	**66.4**	**7.2M**	**0.1B**	**0.2B**	**5.2B**	**64.3**

Table 4. Complexity comparison on ModelNet40 and ShapeNet. The numbers show overall accuracies (%) for ModelNet40 and mIoU (%) for ShapeNet. Params (memory storage of the model) were recorded in bits. We also constructed SVNet-small to compare with BiPointNet with similar complexity. "$*$" indicates our implementations using the code provided by the authors.

	Method	Binarized	z/z	z/SO(3)
Rotation-sensitive	PointNet [30]	✗	68.2	17.1
	DGCNN [40]	✗	78.1	16.1
	PointNet++ [31]	✗	77.9	15.8
	PointCNN [23]	✗	78.5	14.9
Rotation-robust	RIConv [51]	✗	67.9	67.9
	LGR-Net [53]	✗	72.7	72.7
Rotation-robust (**Proposed**)	**SVNet (FP)**	✗	**76.2**	**76.2**
	SVNet	✓	**52.9**	**52.9**
	SVNet†	✓	**60.9**	**60.9**

Table 5. Comparison on ScanObjectNN with the setting **PB_T50_RS**. Numbers show overall accuracies (%).

S : V	Property		Params	MACs	ADDs	BOPs	z/SO(3)
$1:0$	invariant	FP	58.0M	2.5B	0	0	84.8
		Binary	2.5M	0.1B	0	2.4B	71.7
$\frac{2}{3}:\frac{1}{9}$	equivariant	FP	52.3M	1.7B	0	0	90.3
		Binary	3.2M	0.05B	0.1B	1.6B	83.4
$\star\ \frac{1}{2}:\frac{1}{6}$	equivariant	FP	49.7M	1.4B	0	0	90.3
		Binary	3.4M	0.05B	0.2B	1.2B	83.8
$0:\frac{1}{3}$	equivariant	FP	60.7M	1.4B	0	0	89.2
		Binary	11.0M	0.08B	1.3B	0.001B	83.4

Table 6. SVNet variants with different scalar-vector proportions on feature channels. "$\star$" indicates the finally adopted one in the paper. See Tab. 4 for other notes.

rotation robustness; and finally, under binarization, SVNet shows a considerable reduction in memory and computation with high predictive results. Similarly, the performance can be further boosted with extra binarization technique as on ModelNet40.

4.3. Experiments on ScanObjectNN

ScanObjectNN [38] is a real-world dataset having 2,902 object with 15 categories. Different from synthetic ones like ModelNet40 [45], it also contains background points, making classification much more challenging. To evaluate our method, we adopted the hardest setting in this dataset: **PB_T50_RS**, which consists of 14,298 objects revised from the base ones, with 11,416 and 2,882 objects for training and testing, respectively. We adopted the same training settings in 4.1 using DGCNN backbone. The results are shown in Tab. 5. The network complexity of SVNet is almost the same as in Tab. 4 with the DGCNN backbone, with the only difference in the last linear layer due to different numbers of categories.

4.4. Ablation and visualization

If not specified, the following studies were conducted using the DGCNN backbone on the ModelNet40 [45] dataset. The training settings followed Section 4.1.

How do we need S and V? In our experiments, we equally divided scalar and vector features (*i.e.*, $\mathcal{S} \in \mathbb{R}^{\frac{C}{2}\times\mathcal{N}}$ and $\mathcal{V} \in \mathbb{R}^{3\times\frac{C}{6}\times\mathcal{N}}$), so that both have the same amount of features. One may consider: *What if we only use scalar features or vector features? What if we change this proportion?* We conducted such exploration as illustrated in Tab. 6, with different scalar-vector proportions. One design principle in this study is to make the complexity of other SVNet variants approximate or higher than the one adopted in our paper, and observe if the adopted one still achieves equal or better performance. We show how this study matches our original motivation.

Specifically, the proposed SVNet can be regarded as a general form of architectures in prior approaches, where only scalar or vector features are used. One case is the pure invariant version of SVNet (corresponding to $S : V = 1 : 0$ in Tab. 6). This can be implemented by

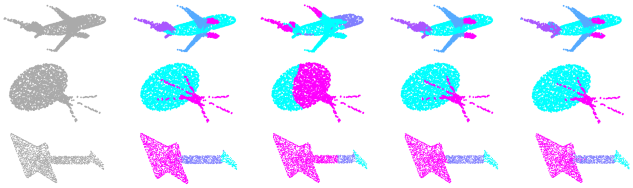


Figure 4. Visualization of part segmentation predictions on ShapeNet testing data using different structures. The input is randomly rotated before being fed to the models. Top to bottom: airplane, table, guitar. Left to right: input, ground truth, results from PointNet [30], results from SVNet-PointNet (FP), results from SVNet-PointNet

	Scalar concatenation	Vector re-weighting	Overall accuracy (z/SO(3), %)
Full-precision	✗	✗	88.8
	✗	✓	89.0
	✓	✗	90.8
	✓	✓	90.3
Binary	✗	✗	79.5
	✗	✓	79.5
	✓	✗	81.5
	✓	✓	83.8

Table 7. Influences of different components in SVBlock.

converting all the vectors (from the original coordinates and relational positions of points) to scalars using Eq. 5 and 6. Though most of the computation can be implemented with BOPs, the invariant SVNet suffers from a considerable accuracy degradation. This simple generation of scalar features causes severe loss of information on geometric structures which leads to potential ambiguities. To solve this, prior methods take nontrivial effort on generating invariant geometric attributes from different perspectives like angles and distances [1, 51], or by considering global information [50]. While we found this problem can be trivially alleviated by introducing more vector features, as shown in the third and fourth rows of Tab. 6.

The other extreme case of SVNet is when we completely discard scalar features in SVBlock (corresponding to the last row of the table). This reduces our method to ones like Vector Neurons [6] or REQNN [36]. The performance in this case is competitive. While compared with the "standard" SVNet we adopted in our experiments, it suffers from a sub-optimal level of efficiency as most of its computation turns to ADDs, rather than BOPs.

Feature interactions in SVBlock Another consideration is the effectiveness of interactions between scalars and vectors during feature updating. In Tab. 7, we validated the two interaction modules in SVBlock: scalar concatenation and vector re-weighting, as introduced in 3.2. Both SVNet (FP) and SVNet were investigated since we wanted to see if there are any differences in full-precision and binary structures. The observations are generally consistent. For both structure types, interactions lead to better performances, especially for binary structures. The only exception is for the full-precision models, vector re-weighting is not that necessary, which implies the nonlinearity is primarily for scalar features. In that case, vector mapping preserves structural information while scalar updating provides feature discrimination. In contrast, for binary structures, both interactions are needed.

Visualization In Fig. 4, we gave qualitative evaluations by visualizing the part segmentation predictions based on PointNet. The original PointNet is rotation sensitive, which produces unstable predictions if the testing input is randomly rotated. As shown in the figure, the issue was desirably solved when configuring it via SVNet.

Future work As we mentioned in the paper, techniques for building strong binary networks like knowledge distillation [13], learnable thresholds [49], attentions [27], and multi-step training [27] can further boost SVNet. Another potential future direction is to extend SVNet for transformers [39, 7, 10] and graph convolutional neural networks (GCNs) [20] on regular/irregular data, which have been proved to be powerful, yet energy-consuming architectures. SVNet has a similar form to transformers or GCNs, *i.e.*, information aggregation from orderless points (tokens in transformers) and point-wise updating (token-wise feed-forward updating in transformers). A rotation-robust property with high running efficiency is fundamentally desirable for such models for wider applications.

5. Conclusion

In this paper, we proposed a general 3D learning structure named SVNet that integrates binarization and SO(3) equivariance for point clouds. Binarization aims to enhance the network efficiency via constraining features and weights to {-1, +1}, so that most of the computation can be implemented with efficient binary operations rather than full-precision multiplications. Meanwhile, SO(3) equivariance enables the network to be robust to 3D rotations, which is the key for 3D point cloud models to adapt to environments with unseen poses. We analyzed the necessity of using both scalar and vector features, which leads us to obtain a better trade-off between efficiency, equivariance, and accuracy.

Acknowledgement. This work was partially supported by National Key Research and Development Program of China No. 2021YFB3100800, the Academy of Finland under grant 331883 and the National Natural Science Foundation of China under Grant 61872379 and 62022091. The CSC IT Center for Science, Finland, is also acknowledged for computational resources.

References

[1] 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 *CVPR*, pages 4994–5002, 2019. 4321, 4322, 4326, 4328

[2] Haiwei Chen, Shichen Liu, Weikai Chen, Hao Li, and Randall Hill. Equivariant point network for 3d point cloud analysis. In *CVPR*, pages 14514–14523, 2021. 4322

[3] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *ICML*, pages 2990–2999. PMLR, 2016. 4322

[4] Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. In *ICLR*, 2018. 4322

[5] Taco S. Cohen and Max Welling. Steerable cnns. In *ICLR*, 2017. 4322

[6] 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 *ICCV*, pages 12200–12209, 2021. 4321, 4322, 4323, 4324, 4325, 4326, 4327, 4328

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 4328

[8] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning so (3) equivariant representations with spherical cnns. In *ECCV*, pages 52–68, 2018. 4321, 4322, 4326

[9] Atzmon *et al*. Frame averaging for equivariant shape space learning. In *CVPR*, pages 631–641, 2022. 4322

[10] Fabian Fuchs, Daniel Worrall, Volker Fischer, and Max Welling. Se (3)-transformers: 3d roto-translation equivariant attention networks. *NeurIPS*, 33:1970–1981, 2020. 4328

[11] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4338–4364, 2020. 4323

[12] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. Neural networks for machine learning. *Coursera, video lectures*, 264(1):2146–2153, 2012. 4324

[13] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 4328

[14] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *NeurIPS*, 29, 2016. 4323

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, volume 37, pages 448–456, 2015. 4325

[16] Bowen Jing, Stephan Eismann, Pratham N Soni, and Ron O Dror. Equivariant graph neural networks for 3d macromolecular structure. *arXiv preprint arXiv:2106.03843*, 2021. 4322, 4323, 4324

[17] Bowen Jing, Stephan Eismann, Patricia Suriana, Raphael John Lamarre Townshend, and Ron O. Dror. Learning from protein structure with geometric vector perceptrons. In *ICLR*, 2021. 4322, 4323, 4324

[18] Seohyun Kim, Jaeyoo Park, and Bohyung Han. Rotation-invariant local-to-global representation learning for 3d point cloud. *NeurIPS*, 33:8174–8185, 2020. 4322, 4326

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 4326

[20] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 4328

[21] Feiran Li, Kent Fujiwara, Fumio Okura, and Yasuyuki Matsushita. A closer look at rotation-invariant deep point cloud analysis. In *ICCV*, pages 16218–16227, 2021. 4322, 4326

[22] 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. 4321, 4322, 4326

[23] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *NeurIPS*, 31, 2018. 4323, 4326, 4327

[24] Min Liu, Fupin Yao, Chiho Choi, Ayan Sinha, and Karthik Ramani. Deep learning 3d shapes using alt-az anisotropic 2-sphere convolution. In *ICLR*, 2019. 4326

[25] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *ECCV*, pages 143–159. Springer, 2020. 4323

[26] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *ICLR*, 2017. 4326

[27] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. In *ICLR*, 2020. 4328

[28] 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 *NeurIPS*, pages 8024–8035, 2019. 4325

[29] Adrien Poulenard and Leonidas J Guibas. A functional approach to rotation equivariant non-linearities for tensor field networks. In *CVPR*, pages 13174–13183, 2021. 4322, 4326

[30] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017. 4322, 4323, 4325, 4326, 4327, 4328

[31] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *NeurIPS*, 30, 2017. 4323, 4326, 4327

[32] Haotong Qin, Zhongang Cai, Mingyuan Zhang, Yifu Ding, Haiyu Zhao, Shuai Yi, Xianglong Liu, and Hao Su. Bipointnet: Binary neural network for point clouds. In *ICLR*, 2021. 4321, 4323, 4326, 4327

[33] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542. Springer, 2016. 4321, 4323

[34] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *ICML*, pages 9323–9332. PMLR, 2021. 4322

[35] Kristof Schütt, Oliver Unke, and Michael Gastegger. Equivariant message passing for the prediction of tensorial properties and molecular spectra. In *ICML*, pages 9377–9388. PMLR, 2021. 4322, 4323

[36] Wen Shen, Binbin Zhang, Shikun Huang, Zhihua Wei, and Quanshi Zhang. 3d-rotation-equivariant quaternion neural networks. In *ECCV*, pages 531–547. Springer, 2020. 4321, 4323, 4324, 4325, 4328

[37] 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. 4322

[38] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *ICCV*, pages 1588–1597, 2019. 4325, 4327

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 30, 2017. 4328

[40] 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. 4322, 4323, 4324, 4325, 4326, 4327

[41] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco S Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In *NeurIPS*, volume 31, 2018. 4322

[42] Maurice Weiler, Fred A Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant cnns. In *CVPR*, pages 849–858, 2018. 4322

[43] Daniel Worrall and Gabriel Brostow. Cubenet: Equivariance to 3d rotation and translation. In *ECCV*, pages 567–584, 2018. 4322

[44] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, pages 9621–9630, 2019. 4323

[45] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920, 2015. 4325, 4327

[46] Jianyun Xu, Xin Tang, Yushi Zhu, Jie Sun, and Shiliang Pu. Sgmnet: Learning rotation-invariant point cloud representations via sorted gram matrix. In *ICCV*, pages 10468–10477, 2021. 4321, 4326

[47] 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 (ToG)*, 35(6):1–12, 2016. 4325, 4326

[48] Ruixuan Yu, Xin Wei, Federico Tombari, and Jian Sun. Deep positional and relational feature learning for rotation-invariant point cloud analysis. In *ECCV*, pages 217–233. Springer, 2020. 4322, 4326

[49] Jiehua Zhang, Zhuo Su, Yanghe Feng, Xin Lu, Matti Pietikäinen, and Li Liu. Dynamic binary neural network by learning channel-wise thresholds. In *ICASSP*, pages 1885–1889. IEEE, 2022. 4328

[50] Zhiyuan Zhang, Binh-Son Hua, Wei Chen, Yibin Tian, and Sai-Kit Yeung. Global context aware convolutions for 3d point cloud understanding. In *3DV*, pages 210–219. IEEE, 2020. 4321, 4322, 4326, 4328

[51] Zhiyuan Zhang, Binh-Son Hua, David W Rosen, and Sai-Kit Yeung. Rotation invariant convolutions for 3d point clouds deep learning. In *3DV*, pages 204–213. IEEE, 2019. 4321, 4322, 4326, 4327, 4328

[52] Zhiyuan Zhang, Binh-Son Hua, and Sai-Kit Yeung. Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. In *ICCV*, pages 1607–1616, 2019. 4323, 4326

[53] 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. 4322, 4327

[54] Wenyu Zhao, Teli Ma, Xuan Gong, Baochang Zhang, and David Doermann. A review of recent advances of binary neural networks for edge computing. *IEEE Journal on Miniaturization for Air and Space Systems*, 2(1):25–35, 2020. 4323