UTILITY ANALYSIS OF NETWORK ARCHITECTURES FOR 3D POINT CLOUD PROCESSING

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we diagnose deep neural networks for 3D point cloud processing to explore the utility of different network architectures. We propose a number of hypotheses on the effects of specific network architectures on the representation capacity of DNNs. In order to prove the hypotheses, we design five metrics to diagnose various types of DNNs from the following perspectives, information discarding, information concentration, rotation robustness, adversarial robustness, and neighborhood inconsistency. We conduct comparative studies based on such metrics to verify the hypotheses, which may shed new lights on the architectural design of neural networks. Experiments demonstrate the effectiveness of our method. *The code will be released when this paper is accepted.*

1 INTRODUCTION

Recently, a series of works use the deep neural network (DNN) for 3D point cloud processing and have achieved superior performance in various 3D tasks. However, traditional studies usually design network architectures based on empiricism. There does not exist a rigorous and quantitative analysis about the utility of specific network architectures for 3D point cloud processing. Exploring and verifying the utility of each specific intermediate-layer architecture from the perspective of a DNN's representation capacity still present significant challenges for state-of-the-art algorithms.

In this study, we aim to bridge the gap between the intermediate-layer network architecture and its utility. Therefore, we propose a few hypotheses of the utility of specific network architectures. Table 1 lists the hypotheses to be verified in this study, towards three kinds of utilities, *i.e.* rotation robustness, adversarial robustness, and neighborhood inconsistency. We design and conduct comparative studies to verify the hypotheses. Finally, we obtain some new insights into the utility of specific network architectures as follows.

- The specific architecture in (Wu et al., 2019), which uses the local density information to reweight features (Figure 1 (a)), improves adversarial robustness (Table 1 (a)).
- Another specific architecture in (Wu et al., 2019), which uses local 3D coordinates' information to reweight features (Figure 1 (b)), improves rotation robustness (Table 1 (b)).
- The specific architecture in (Qi et al., 2017b; Liu et al., 2018), which extracts multi-scale features (Figure 1 (c)), improves adversarial robustness and neighborhood consistency (Table 1 (c)). Neighborhood consistency measures whether a DNN assigns similar attention to neighboring points.
- The specific architecture in (Jiang et al., 2018), which encodes the information of different orientations (Figure 1 (d)), improves rotation robustness (Table 1 (d)).

More specifically, in order to verify the above hypotheses, we design the following five evaluation metrics and conduct a number of comparative experiments to quantify the utility of different network architectures.

1. Information discarding and 2. information concentration: Information discarding measures how much information of an input point cloud is forgotten during the computation of a specific intermediate-layer feature. From the perspective of information propagation, the forward propagation through layers can be regarded as a hierarchical process of discarding input information (Shwartz-Ziv & Tishby, 2017). Ideally, the DNN is supposed to discard information that is not related to the task. Let us take the task of object classification for example. The information of

Architectures	Rotation robustness	Adversarial robustness	Neighborhood inconsistency
(a) Modules of using information of local density to reweight features.	–	✓	–
(b) Modules of using information of local coordinates to reweight features.	✓	–	–
(c) Modules of concatenating multi-scale features.	–	✓	✓
(d) Modules of computing orientation-aware features.	✓	–	–

Table 1: Illustration of the verified utilities of specific architectures. Blank regions correspond to utilities that have not been examined, instead of indicating non-existence of the utilities. Please see Figure 1 for architectural details.

foreground points is usually supposed to be related to the task, while that of background points is not related to the task and is discarded.

To this end, we further propose information concentration to measure the gap between the information related to the task and the information not related to the task. Information concentration can be used to evaluate a DNN's ability to focus on points related to the task.

3. Rotation robustness: Rotation robustness measures whether a DNN will use the same logic to recognize the same object when a point cloud has been rotated by a random angle. In other words, if two point clouds have the same global shape but different orientations, the DNN is supposed to select the same regions/points to compute the intermediate-layer feature. Unlike images with rich color information, point clouds usually only use spatial contexts for classification. Therefore, a well-trained DNN is supposed to have rotation robustness.

4. Adversarial robustness: A reliable DNN should be robust to adversarial attacks.

5. Neighborhood inconsistency: Neighborhood inconsistency measures whether adjacent points have similar importance in the computation of an intermediate-layer feature. Adjacent points usually have similar shape contexts, so they are supposed to have similar importance. Therefore, ideally, a well-trained DNN should have a low value of neighborhood inconsistency.

Contributions of our study are summarized as follows. We propose a few hypotheses on the utility of specific network architectures. Then, we design five metrics to conduct comparative studies for verifying these hypotheses, which provide a new insightful understanding of architectural utility.

2 Related work

3D Point Cloud Processing: Recently, a number of approaches use DNNs for 3D point cloud processing and have exhibited superior performance in various 3D tasks (Qi et al., 2017a; Su et al., 2018; Valsesia et al., 2018; Yu et al., 2018; Yang et al., 2018; Gadelha et al., 2018; Wang et al., 2018a; Komarichev et al., 2019; Shi et al., 2019). PointNet (Qi et al., 2017a) was a pioneer in this direction, which used a max pooling layer to aggregate all individual point features into a global feature. However, such architecture fell short of capturing local features. PointNet++ (Qi et al., 2017b) hierarchically used PointNet as a local descriptor to extract contextual information. Some studies (Wu et al., 2019; Jiang et al., 2018; Wang et al., 2018b; Komarichev et al., 2019) further improved the networks' ability to capture local geometric features. Other researches focused on the correlations between different regions of the 3D point cloud (Liu et al., 2018) or interaction between points (Zhao et al., 2019a). In comparison, our study focuses on the utility analysis of intermediate-layer network architectures for point cloud processing.

Visualization or diagnosis of representations: The visualization of visual patterns corresponding to a feature map or the network output is the most intuitive way of interpreting DNNs (Zeiler & Fergus, 2014; Mahendran & Vedaldi, 2015; Dosovitskiy & Brox, 2016; Zhou et al., 2014), such as gradient-based methods (Fong & Vedaldi, 2018; Selvaraju et al., 2017), and the estimation of the saliency map (Ribeiro et al., 2016; Lundberg & Lee, 2017; Kindermans et al., 2017; Qi et al., 2017a; Zheng et al., 2019). In comparison, our study aims to explore the utility of intermediate-layer network architectures by diagnosing the information-processing logic of DNNs.

Quantitative evaluation of representations: Recently, some studies quantified the representation similarity to help understand the neural networks (Gotmare et al., 2018; Kornblith et al., 2019; Morcos et al., 2018; Raghu et al., 2017). The method (Li et al., 2019) quantitated the importance of different feature dimensions to guide model compression. Other studies evaluated the representations via quantifying the information they contain. The information-bottleneck theory (Tishby et al., 2000; Shwartz-Ziv & Tishby, 2017; Cheng et al., 2018) explained the trade-off between the information compression and the discrimination power of features in a neural network. Achille & Soatto

(2018) designed an information Dropout layer and quantified the information transmitted through it. Ma et al. (2019) presented a method to quantify the layer-wise information discarding of DNNs and defined two model-agnostic and task-agnostic metrics of the input information. Inspired by (Ma et al., 2019), we propose five metrics to diagnose feature representations of different DNNs and finally explore the utility of different network architectures.

3 METRICS TO DIAGNOSE NETWORKS

3.1 PRELIMINARY: ENTROPY-BASED INFORMATION DISCARDING QUANTIFICATION

We extend the method of (Ma et al., 2019) as the technical foundation. Based on which, a number of metrics are designed to diagnose the DNN. The method quantifies the discarding of the input information during the layerwise forward propagation by computing the entropy of the input information given the specific feature of an intermediate layer. Given a point cloud X, let $f = h(X)$ denote the feature of a specific intermediate layer. It is assumed that f and f' represent the same object concept[1] when f' satisfies $\|f' - f\|^2 < \epsilon$, where feature $f' = h(X')$, $X' = X + \boldsymbol{\delta}$. $\boldsymbol{\delta}$ denotes a random noise. The conditional entropy of the input information given a specific feature, which represents a specific object concept, is computed, *i.e.* calculating entropy $H(X')$, *s.t.* $\|f' - f\|^2 < \epsilon$. It is assumed that X' follows a Gaussian distribution $X' \sim \mathcal{N}(X, \Sigma = diag[\sigma_1^2, \sigma_2^2, \dots])$, where Σ can be regarded as the maximum perturbation added to X following the maximum-entropy principle, which subjects to a specific concept. Considering the assumption of independent and identically distributed variables of each dimension of X', the overall entropy $H(X')$ can be decomposed into point-wise entropies.

$$\max_{\boldsymbol{\sigma}=[\sigma_1,\sigma_2,\dots]^\top} H(X'), \qquad \text{s.t. } \left\|h(X') - f\right\|^2 < \epsilon, \qquad \text{where } H(X') = \sum_i H_i, \tag{1}$$

where $H_i = \log \sigma_i + \frac{1}{2}\log(2\pi e)$ denotes the entropy of the i-th point, which quantifies how much information of the i-th point can be discarded, when the feature $h(X')$ is required to represent the concept of the target object.

3.2 FIVE METRICS

Metric 1, information discarding: The information discarding is defined as $H(X')$ in Equation (1). The information discarding is measured at the point level, *i.e.* H_i, which quantifies how much information of the i-th point is discarded during the computation of an intermediate-layer feature. The point with a lower value of H_i is regarded more important in the computation of the feature.

Metric 2, information concentration: The information concentration is based on the metric of information discarding. The information concentration is used to analyze a DNN's ability to maintain the input information related to the task, and discard redundant information unrelated to the task. For example, in the task of object classification, the background points are usually supposed not to be related to the task and are therefore more likely to be discarded by the DNN. Let $\Lambda^{\text{foreground}}$ denote the set of points in the foreground object in the point cloud X, and let $\Lambda^{\text{background}}$ denote the set of points in the background. Information concentration can be implemented as the relative background information discarding *w.r.t.* foreground information discarding.

$$\mathbb{E}_{i\in\Lambda^{\text{background}}}[H_i] - \mathbb{E}_{i\in\Lambda^{\text{foreground}}}[H_i]. \tag{2}$$

A higher value of information concentration indicates that the DNN concentrates more on the foreground information during the computation of the feature.

Metric 3, rotation robustness: The rotation robustness is proposed to measure whether a DNN uses similar subsets of two point clouds to compute the intermediate-layer feature, if the two point clouds have the same shape but different orientations. Let X^{θ_1} and X^{θ_2} denote the point clouds that have the same global shape but different orientations θ_1 and θ_2. To quantify the similarity of the attention on the two point clouds, we compute the Jensen-Shannon divergence between the distributions of the perturbed inputs $\hat{X}^{\theta_1} = X^{\theta_1} + \boldsymbol{\delta}_1$ and $\hat{X}^{\theta_2} = X^{\theta_2} + \boldsymbol{\delta}_2$. $\hat{X}^{\theta_1}$ and $\hat{X}^{\theta_2}$ denote the perturbed inputs, which are computed to measure information discarding in Equation (1). *I.e.* we measure whether the DNN ignores similar sets of points to compute features of the two point clouds.

$$JSD(\hat{X}^{\theta_1}||\hat{X}^{\theta_2}), \qquad \text{s.t. } ||h(\hat{X}^{\theta_1}) - h(X^{\theta_1})|| < \epsilon, \;\; ||h(\hat{X}^{\theta_2}) - h(X^{\theta_2})|| < \epsilon, \tag{3}$$

[1] In this study, the concept of an object is referred to as a small range of features that represent the same object instance.

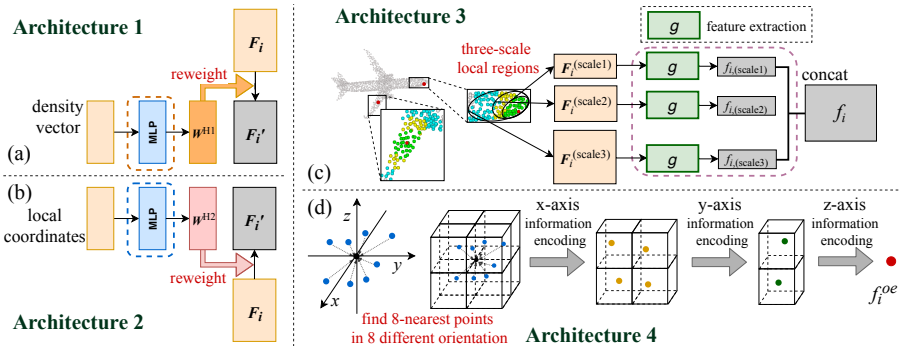

Figure 1: **Illustration of the specific intermediate-layer architectures.** Please see texts in Section 4.1, as well as Appendix C for architectural details.

where $JSD(\hat{X}^{\theta_1}||\hat{X}^{\theta_2})$ measures the dissimilarity between information distributions over the two point clouds.

The rotation non-robustness is defined as the average of the dissimilarity of attention on any two point clouds with different orientations, *i.e.* $\mathbb{E}_{\forall\theta_1,\theta_2}[JSD(\hat{X}^{\theta_1}||\hat{X}^{\theta_2})]$. Note that we do not directly compare the attention on features, because there is no mechanism to ensure that dimensions of X^{θ_1} and X^{θ_2} are semantically aligned. In this study, we use the method in Hershey & Olsen (2007) to approximate the Jensen-Shannon divergence.

Metric 4, adversarial robustness: We use the method in (Szegedy et al., 2013) to perform adversarial attacks. The objective is

$$\min\|\boldsymbol{\epsilon}\|_2^2, \qquad \text{s.t.} \quad C(X+\boldsymbol{\epsilon}) = \hat{l} \neq l^*, \tag{4}$$

where $C(\cdot)$ is the predicted label; l^* is the correct label of X; $\hat{l}$ is a target incorrect label. In this study, we perform adversarial attacks against all incorrect classes. We use the average of $\|\boldsymbol{\epsilon}\|_2$ over all incorrect classes to measure the adversarial robustness.

Metric 5, neighborhood inconsistency: The neighborhood inconsistency is proposed to evaluate a DNN's ability to assign similar attention to neighboring points during the computation of an intermediate-layer feature, *i.e.* a well-trained DNN should have a low value of neighborhood inconsistency. Ideally, for a DNN, except for special points (*e.g.* those on the edge), most neighboring points in a small region of a point cloud usually have similar shape contexts, so they are supposed to make similar contributions to the classification and receive similar attention. Let $\mathbf{N}(i)$ denote a set of K nearest neighboring points of the i-th point. We define the neighborhood inconsistency as the difference between the maximum and minimum point-wise information discarding within $\mathbf{N}(i)$.

$$\mathbb{E}_i[\max_{j\in\mathbf{N}(i)} H_j - \min_{j\in\mathbf{N}(i)} H_j]. \tag{5}$$

4 Hypotheses and Comparative Study

4.1 Introduction of specific architectures

- **Notation:** Let $x_i \in \mathbb{R}^3$ denote the i-th point, $i = 1, 2, \ldots, n$; let $\mathbf{N}(i)$ denote a set of K nearest points of x_i; let $\mathbf{F}_i \in \mathbb{R}^{d\times K}$ denote intermediate-layer features that correspond to neighboring points in $\mathbf{N}(i)$, where each column of $\mathbf{F}_i$ represents the feature of a specific point in $\mathbf{N}(i)$.

- **Architecture 1, features reweighted by the information of the local density:** Architecture 1 focuses on the use of the local density information to reweight features (Wu et al., 2019). As shown in Figure 1 (a), for each point x_i, Architecture 1 uses the local density *w.r.t.* neighboring points of x_i to compute $\boldsymbol{W}^{\mathrm{H1}} \in \mathbb{R}^K$, which reweights intermediate-layer features $\mathbf{F}_i$.

$$\mathbf{F}'_i = \mathbf{F}_i\, diag[\boldsymbol{W}^{\mathrm{H1}}], \qquad \text{where} \quad \boldsymbol{W}^{\mathrm{H1}} = MLP(density(\mathbf{N}(i))), \tag{6}$$

where $diag[\boldsymbol{W}^{\mathrm{H1}}]$ transforms the vector $\boldsymbol{W}^{\mathrm{H1}}$ into a diagonal matrix; $density(\mathbf{N}(i))$ is a density vector *w.r.t.* points in $\mathbf{N}(i)$; the MLP is a two-layer perceptron network.

- **Architecture 2, features reweighted by the information of local coordinates:** As shown in Figure 1 (b), for each point x_i, Architecture 2 uses the information of local 3D coordinates to compute $\boldsymbol{W}^{\mathrm{H2}} \in \mathbb{R}^{M\times K}$ to reweight intermediate-layer features $\mathbf{F}_i$.

$$\mathbf{F}'_i = \mathbf{F}_i(\boldsymbol{W}^{\mathrm{H2}})^\top, \qquad \text{where} \quad \boldsymbol{W}^{\mathrm{H2}} = MLP(\{x_j|j \in \mathbf{N}(i)\}), \tag{7}$$

where the MLP is a single-layer perceptron network.

• **Architecture 3, multi-scale features:** Architecture 3 focuses on the use of multi-scale contextual information (Qi et al., 2017b; Liu et al., 2018). As illustrated in Figure 1 (c), $\{\mathbf{F}_i^{\text{scale}=K_1}, ..., \mathbf{F}_i^{\text{scale}=K_T}\}$ denote features that are extracted using contexts of x_i at different scales, $\mathbf{F}_i^{\text{scale}=K_t} \in \mathbb{R}^{d \times K_t}$. Each specific context *w.r.t.* x_i is composed of K_t nearest neighboring points around x_i. Then, $f_{i,(\text{scale}=K_t)}^{\text{upper}} \in \mathbb{R}^D$ in the upper layer is computed using $\mathbf{F}_i^{\text{scale}=K_t}$. Architecture 3 concatenates these multi-scale features to obtain f_i^{upper}.

$$f_i^{\text{upper}} = concat \left\{ \begin{array}{c} f_{i,(\text{scale}=K_1)}^{\text{upper}} \\ f_{i,(\text{scale}=K_2)}^{\text{upper}} \\ \vdots \\ f_{i,(\text{scale}=K_T)}^{\text{upper}} \end{array} \right\}, \qquad \text{where } f_{i,(\text{scale}=K_t)}^{\text{upper}} = g(\mathbf{F}_i^{\text{scale}=K_t}), \tag{8}$$

where $concat$ indicates the concatenation operator; $g(\cdot)$ is a function for feature extraction (Qi et al., 2017a). Please see Appendix B for details about this function.

• **Architecture 4, orientation-aware features:** Architecture 4 focuses on the use of orientation information (Jiang et al., 2018). As illustrated in Figure 1 (d), for each point x_i, $\mathbf{F}_i^{\text{oe}} \in \mathbb{R}^{d \times O}$ denotes the feature of x_i, which encodes the information of various orientations, where O is the number of orientations. Architecture 4 uses $\mathbf{F}_i^{\text{oe}}$ to compute the orientation-aware feature $f_i^{\text{oe}} \in \mathbb{R}^d$.

$$f_i^{\text{oe}} = Conv^{\text{oe}}(\mathbf{F}_i^{\text{oe}}), \tag{9}$$

where $Conv^{\text{oe}}$ is a special convolution operator. Please see (Jiang et al., 2018) or Appendix C.4 for details about this operator and the computation of f_i^{oe}.

4.2 Four hypotheses and comparative study design

Hypothesis 1: Architecture 1 designed by (Wu et al., 2019), as shown in Figure 1 (a), increases the adversarial robustness.

This hypothesis is proposed based on the observation that PointConv (Wu et al., 2019) has good performance in adversarial robustness, which may stem from Architecture 1. To verify this hypothesis, we design comparative studies on PointConv, PointNet++ (Qi et al., 2017b), and Point2Sequence (Liu et al., 2018). For each network, we construct two versions for comparison, *i.e.* one with Architecture 1 and the other without Architecture 1.

PointConv w/ and w/o Architecture 1: To obtain the PointConv without Architecture 1, we remove all the modules of Architecture 1 from the original network (see the footnote[2]), which are located behind the 2-nd, 5-th, 8-th, 11-th, and 14-th nonlinear transformation layers. Please see Appendix D.2 for the global architectures of different versions of PointConv.

$$f_i^{\text{upper}} = MLP(\mathbf{F}_i)\ diag[\boldsymbol{W}^{\text{H1}}] \quad \Longrightarrow \quad f_i^{\text{upper}} = MLP(\mathbf{F}_i), \tag{10}$$

where f_i^{upper} is the feature in the upper layer; $diag[\boldsymbol{W}^{\text{H1}}]$ transforms the vector $\boldsymbol{W}^{\text{H1}}$ into a diagonal matrix.

PointNet++ w/ and w/o Architecture 1: To obtain the PointNet++ with Architecture 1, we add three modules of Architecture 1, which are located behind the 3-rd, 6-th, and 9-th nonlinear transformation layers. Please see Appendix D.1 for the global architectures of different versions of PointNet++.

$$f_i^{\text{upper}} = MLP(\mathbf{F}_i) \qquad \Longrightarrow \qquad f_i^{\text{upper}} = MLP(\mathbf{F}_i)\ diag[\boldsymbol{W}^{\text{H1}}]. \tag{11}$$

Point2Sequence w/ and w/o Architecture 1: To obtain the Point2Sequence with Architecture 1, we add the module of Architecture 1 behind the last nonlinear transformation layer, as shown in Equation (11). Please see Appendix D.3 for the global architectures of different versions of Point2Sequence.

Hypothesis 2: Architecture 2 designed by (Wu et al., 2019), as shown in Figure 1 (b), increases the rotation robustness.

[2]The PointConv for classification is revised from the code for segmentation released by (Wu et al., 2019).

This hypothesis is proposed based on the observation that PointConv (Wu et al., 2019) has good performance in rotation robustness, which may stem from Architecture 2. To verify this hypothesis, we design comparative studies on PointConv, PointNet++, and Point2Sequence.

PointConv w/ and w/o Architecture 2: To obtain the PointConv without Architecture 2, we remove all the modules of Architecture 2, which are located before the 3-rd, 6-th, 9-th, 12-th, and 15-th nonlinear transformation layers. Please see Appendix D.2 for the global architectures of different versions of PointConv.

$$f_i^{\text{upper}} = MLP(\mathbf{F}_i)(\boldsymbol{W}^{\text{H2}})^\top \quad \Longrightarrow \quad f_i^{\text{upper}} = MLP(\mathbf{F}_i). \tag{12}$$

PointNet++ w/ and w/o Architecture 2: Just like Hypothesis 1, to obtain the PointNet++ with Architecture 2, we add three modules of Architecture 2, which are located behind the 3-rd, 6-th, and 9-th nonlinear transformation layers. Please see Appendix D.1 for the global architectures of different versions of PointNet++.

$$f_i^{\text{upper}} = MLP(\mathbf{F}_i) \qquad \Longrightarrow \qquad f_i^{\text{upper}} = MLP(\mathbf{F}_i)(\boldsymbol{W}^{\text{H2}})^\top. \tag{13}$$

Point2Sequence w/ and w/o Architecture 2: Just like Hypothesis 1, to obtain the Point2Sequence with Architecture 2, we add the module of Architecture 2 behind the last nonlinear transformation layer, as shown in Equation (13). Please see Appendix D.3 for the global architectures of different versions of Point2Sequence.

Hypothesis 3: Architecture 3 designed by (Qi et al., 2017b), as shown in Figure 1 (c), increases the adversarial robustness and the neighborhood consistency.

This hypothesis is proposed inspired by (Qi et al., 2017b; Liu et al., 2018), which encode multi-scale contextual information. To verify this hypothesis, we design the following comparative studies on Point2Sequence and PointNet++.

PointNet++ w/ and w/o Architecture 3: To obtain the PointNet++ with Architecture 3, we use the multi-scale version of PointNet++ (Qi et al., 2017b), which extracts features at three scales.

Point2Sequence w/ and w/o Architecture 3: To obtain different versions of Point2Sequence, we use three networks as follows. The baseline network of Point2Sequence concatenates features of 4 different scales to compute the feature in the upper layer, $\{f_{i,(\text{scale}=K_1)}^{\text{upper}}, f_{i,(\text{scale}=K_2)}^{\text{upper}}, f_{i,(\text{scale}=K_3)}^{\text{upper}}, f_{i,(\text{scale}=K_4)}^{\text{upper}}\}$. In this study, we set K_1 = 128, K_2 = 64, K_3 = 32, and K_4 = 16. The first network for comparison extracts three different scale features, $\{f_{i,(\text{scale}=K_1)}^{\text{upper}}, f_{i,(\text{scale}=K_2)}^{\text{upper}}, f_{i,(\text{scale}=K_3)}^{\text{upper}}\}$, and the second one extracts two different scale features, $\{f_{i,(\text{scale}=K_1)}^{\text{upper}}, f_{i,(\text{scale}=K_2)}^{\text{upper}}\}$.

Hypothesis 4: Architecture 4 designed by (Jiang et al., 2018), as shown in Figure 1 (d), increases the rotation robustness.

This hypothesis is proposed based on the observation that PointSIFT (Jiang et al., 2018) performs well in rotation robustness, which may stem from Architecture 4. Because Architecture 4 ensures that features contain information from various orientations. To verify this hypothesis, we design comparative studies on PointSIFT, PointNet++, and Point2Sequence as follows.

PointSIFT w/ and w/o Architecture 4: To get the PointSIFT without Architecture 4, we remove all the modules of Architecture 4 from the original network (see the footnote[3]), which are located before the 1-st, 3-rd, 5-th, and 7-th nonlinear transformation layers. Please see Appendix D.4 for the global architectures of different versions of PointSIFT.

PointNet++ w/ and w/o Architecture 4: To obtain the PointNet++ with Architecture 4, we add the module of Architecture 4 before the 7-th nonlinear transformation layer.

Point2Sequence w/ and w/o Architecture 4: To obtain the Point2Sequence with Architecture 4, we add the module of Architecture 4 before the 14-th nonlinear transformation layer.

[3]The PointSIFT for classification is revised from the code for segmentation released by (Jiang et al., 2018).

Table 2: Quantification of the representation capacity of different DNNs on the ModelNet40 dataset.

Models	Information discarding	Information concentration	Rotation non-robustness	Adversarial robustness	Neighborhood inconsistency
PointNet	-8128.10	1.043	8.409	1.994	2.946
PointNet++	-8578.97	1.116	5.000	2.504	3.451
PointConv	-8720.98	0.380	3.918	2.878	3.735
DGCNN	-9165.82	1.042	4.383	2.421	1.445
PointSIFT	-8391.08	0.032	4.747	2.839	2.387
Point2Sequence	-8145.27	1.141	5.786	2.526	3.655

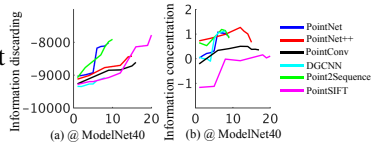

Figure 2: Comparisons of layer-wise information discarding and layerwise information concentration between DNNs.

Table 3: Comparisons of the rotation non-robustness between DNNs trained using the ModelNet40, ShapeNet, and 3D MNIST datasets. The column of Δ denotes the increase of the rotation robustness of the network with the specific architecture *w.r.t.* the network without the specific architecture. $\Delta > 0$ indicates that the corresponding hypothesis has been verified. Experimental results show that Architecture 2 and Architecture 4 improved the rotation robustness, *i.e.* reporting relatively lower values of rotation non-robustness. Please see Appendix F (Table 14) for the classification accuracy of these DNNs.

Architecture	Model	ModelNet40			ShapeNet			3D MNIST		
		w/	w/o	Δ	w/	w/o	Δ	w/	w/o	Δ
Architecture 2	PointConv	3.918	3.954	**0.036**	4.250	2.703	-1.547	5.140	6.221	**1.081**
	PointNet++	2.658	5.000	**2.342**	4.186	6.709	**2.523**	5.256	6.754	**1.498**
	Point2Sequence	4.020	5.786	**1.766**	2.821	5.222	**2.401**	4.590	7.410	**2.820**
Architecture 4	PointSIFT	4.747	7.090	**2.343**	4.598	5.118	**0.520**	7.851	6.154	-1.697
	PointNet++	5.505	5.000	-0.505	6.152	6.709	**0.557**	5.298	6.754	**1.456**
	Point2Sequence	2.917	5.786	**2.869**	2.909	5.222	**2.313**	5.942	7.410	**1.468**

5 EXPERIMENTS

To demonstrate the broad applicability of our method, we applied our method to diagnose six widely used DNNs, including PointNet, PointNet++, PointConv, DGCNN, PointSIFT, and Point2Sequence. These DNNs were trained using three benchmark datasets, including the ModelNet40 dataset (Wu et al., 2015), the ShapeNet[4] dataset (Chang et al., 2015), the 3D MNIST[5] dataset. In sum, we conducted four comparative experiments. The first experiment was to analyze DNNs using the proposed five metrics. The remaining three experiments were conducted to verify the effects of different network architectures on DNNs' rotation robustness, adversarial robustness, and neighborhood inconsistency, respectively.

Implementation details: To analyze the information concentration of DNNs, we generated a new dataset that contained both the foreground objects and the background, since most widely used benchmark datasets for point cloud classification only contain foreground objects. Specifically, for each sample (*i.e.* the foreground object) in the ModelNet40, we used the following three steps to generate the background. First, we randomly sampled a set of 500 points from point clouds, which had different labels from the foreground object. Second, we resized this set of points to the density of the foreground object. Finally, we randomly located it around the foreground object. *The dataset will be released when this paper is accepted.*

The entropy-based method (Ma et al., 2019) quantified the layerwise information discarding. This method assumed the feature space of the concept of a specific object satisfied $\|f' - f\|^2 < \epsilon$, where $f = h(X)$, $f' = h(X')$, $X' = X + \boldsymbol{\delta}$. $\boldsymbol{\delta}$ denotes a random noise. For point cloud processing, each dimension of the intermediate-layer feature is computed using the context of a specific point x_i. However, adding noise to a point cloud will change the context of each point. In order to extend the entropy-based method to point cloud processing, we selected the same set of points as the contexts *w.r.t.* x_i and x_i', so as to generate a convincing evaluation. Please see Appendix E for details.

Experiment 1, quantifying the representation capacity of DNNs: As shown in Table 2, we measured information discarding, information concentration, rotation robustness, and neighborhood inconsistency of the representation of the fully connected layer close to the network output, which had 512 hidden units. We measured adversarial robustness by performing adversarial attacks over all

[4]The ShapeNet dataset for classification is converted from the ShapeNet part segmentation dataset.

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

Table 4: Comparisons of the adversarial robustness between DNNs trained using the ModelNet40, ShapeNet, and 3D MNIST datasets. The column of Δ denotes the increase of the adversarial robustness of the network with the specific architecture *w.r.t.* the network without the specific architecture. $\Delta > 0$ indicates that the corresponding hypothesis has been verified. Experimental results show that Architecture 1 and Architecture 3 improved the adversarial robustness. Please see Appendix F (Table 15) for the classification accuracy of these DNNs.

Architecture	Model	ModelNet40			ShapeNet			3D MNIST		
		w/	w/o	Δ	w/	w/o	Δ	w/	w/o	Δ
Architecture 1	PointConv	2.878	2.629	**0.249**	2.407	2.271	**0.136**	2.737	2.530	**0.207**
	PointNet++	2.519	2.504	**0.015**	2.496	2.437	**0.059**	2.427	2.352	**0.075**
	Point2Sequence	2.544	2.526	**0.018**	2.500	2.520	-0.020	2.475	2.468	**0.007**
Architecture 3	PointNet++	3.010	2.504	**0.506**	2.987	2.437	**0.550**	2.604	2.352	**0.252**
	Point2Sequence (4 scales *vs.* 3 scales)	2.526	2.521	**0.005**	2.520	2.514	**0.006**	2.468	2.479	-0.011
	Point2Sequence (4 scales *vs.* 2 scales)		2.513	**0.013**		2.488	**0.032**		2.460	**0.008**

Table 5: Comparisons of the neighborhood inconsistency between DNNs trained using the ModelNet40, ShapeNet, and 3D MNIST datasets. The column of Δ denotes the increase of the neighborhood consistency of the network with the specific architecture *w.r.t.* the network without the specific architecture. $\Delta > 0$ indicates that the corresponding hypothesis has been verified. Experimental results show that Architecture 3 increased the neighborhood consistency, *i.e.* reporting relatively lower values of neighborhood inconsistency. Please see Appendix F (Table 15) for the classification accuracy of these DNNs.

Model	ModelNet40			ShapeNet			3D MNIST		
	w/	w/o	Δ	w/	w/o	Δ	w/	w/o	Δ
PointNet++	3.149	3.451	**0.302**	3.321	3.346	**0.025**	3.496	3.519	**0.023**
Point2Sequence (4 scales *vs.* 3 scales)	3.655	4.182	**0.527**	3.091	3.179	**0.088**	3.226	3.411	**0.185**
Point2Sequence (4 scales *vs.* 2 scales)		4.253	**0.598**		3.199	**0.108**		3.537	**0.311**

incorrect classes. Figure 2 compares layerwise information discarding and layerwise information concentration of different layers in different DNNs. We found that PointNet and Point2Sequence had relatively higher values of information discarding. PointConv and PointSIFT discarded more information of points in the background. PointConv, DGCNN, and PointSIFT performed well in rotation robustness. PointConv, PointSIFT, and Point2Sequence exhibited higher adversarial robustness. DGCNN and PointSIFT exhibited lower neighborhood inconsistency.

Experiment 2, verifying the effects on rotation robustness: For the computation of rotation robustness, during the training and testing phases, each point cloud was rotated by random angles. Table 3 shows that Architecture 2 and Architecture 4 improved the rotation robustness.

Experiment 3, verifying the effects on adversarial robustness: Table 4 shows that both Architecture 1 and Architecture 3 improved the adversarial robustness. For Architecture 3, we found that the adversarial robustness increased with the scale number of features.

Experiment 4, verifying the effects on neighborhood inconsistency: For the computation of neighborhood inconsistency, we used k-NN search to select 16 neighbors for each point. Table 5 shows that networks with Architecture 3 usually had lower neighborhood inconsistency than those without Architecture 3. Besides, DNNs, which extracted features from contexts of more scales, usually exhibited lower neighborhood inconsistency.

6 Conclusion

In this paper, we have verified a few hypotheses of the utility of four specific network architectures for 3D point cloud processing. Comparative studies are conducted to prove the utility of the specific architectures, including rotation robustness, adversarial robustness, and neighborhood inconsistency. In preliminary experiments, we have verified that Architecture 2 and Architecture 4 mainly improve the rotation robustness; Architecture 1 and Architecture 3 have positive effects on adversarial robustness; Architecture 3 usually alleviates the neighborhood inconsistency.

REFERENCES

Alessandro Achille and Stefano Soatto. Information dropout: Learning optimal representations through noisy computation. *IEEE transactions on pattern analysis and machine intelligence*, 40 (12):2897–2905, 2018.

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

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pp. 2172–2180, 2016.

Hao Cheng, Dongze Lian, Shenghua Gao, and Yanlin Geng. Evaluating capability of deep neural networks for image classification via information plane. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 168–182, 2018.

Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4829–4837, 2016.

Ruth Fong and Andrea Vedaldi. Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8730–8738, 2018.

Vincent Fortuin, Matthias Hüser, Francesco Locatello, Heiko Strathmann, and Gunnar Rätsch. Som-vae: Interpretable discrete representation learning on time series. *arXiv preprint arXiv:1806.02199*, 2018.

Matheus Gadelha, Rui Wang, and Subhransu Maji. Multiresolution tree networks for 3d point cloud processing. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 103–118, 2018.

Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243*, 2018.

John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pp. IV–317. IEEE, 2007.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2(5):6, 2017.

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

Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. *arXiv preprint arXiv:1705.05598*, 2017.

Artem Komarichev, Zichun Zhong, and Jing Hua. A-cnn: Annularly convolutional neural networks on point clouds. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

Yuchao Li, Shaohui Lin, Baochang Zhang, Jianzhuang Liu, David Doermann, Yongjian Wu, Feiyue Huang, and Rongrong Ji. Exploiting kernel sparsity and entropy for interpretable cnn compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2800–2809, 2019.

Xinhai Liu, Zhizhong Han, Yu-Shen Liu, and Matthias Zwicker. Point2sequence: Learning the shape representation of 3d point clouds with an attention-based sequence to sequence network. *arXiv preprint arXiv:1811.02565*, 2018.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pp. 4765–4774, 2017.

Haotian Ma, Yinqing Zhang, Fan Zhou, and Quanshi Zhang. Quantifying layerwise information discarding of neural networks. *arXiv preprint arXiv:1906.04109*, 2019.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5188–5196, 2015.

Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems*, pp. 5727–5736, 2018.

Alex Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo J Rezende. Towards interpretable reinforcement learning using attention augmented agents. *arXiv preprint arXiv:1906.02500*, 2019.

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

Charles R 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*, pp. 5099–5108, 2017b.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, pp. 6076–6085, 2017.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144. ACM, 2016.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pp. 3856–3866, 2017.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 618–626, 2017.

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

Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

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

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Learning localized generative models for 3d point clouds via graph convolution. 2018.

Joel Vaughan, Agus Sudjianto, Erind Brahimi, Jie Chen, and Vijayan N Nair. Explainable neural networks based on additive index models. *arXiv preprint arXiv:1806.01933*, 2018.

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

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

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

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

Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 206–215, 2018.

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

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8827–8836, 2018.

Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019a.

Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3d point capsule networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1009–1018, 2019b.

Tianhang Zheng, Changyou Chen, Junsong Yuan, Bo Li, and Kui Ren. Pointcloud saliency maps. *arXiv preprint arXiv:1812.01687*, 2019.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.

A OVERVIEW

This Appendix provides more details about comparative studies in the main paper and includes more implementation details about experiments. In Section B, we introduce a special element-wise max operator widely used in point cloud processing. In Section C, we briefly introduce DNNs used in comparative studies. In Section D, we show details about different versions of DNNs for comparison. In Section E, we show implementation details about extending the entropy-based method (Ma et al., 2019) to point cloud processing. In Section F, we compare the accuracy of different versions of DNNs. In Section G, we supplement related work about learning interpretable representations.

B A SPECIAL ELEMENT-WISE MAX OPERATOR IN POINT CLOUD PROCESSING

In point cloud processing, a special element-wise max operator is widely used for aggregating a set of neighboring points' features into a local feature. As shown in Figure 3, given a set of K nearest neighboring points of x_i, $\mathbf{N}(i)$, let $\mathbf{F}_i \in \mathbb{R}^{d\times K}$ denote intermediate-layer features that correspond to the set of neighboring points in $\mathbf{N}(i)$ *w.r.t.* the point x_i. Each specific column of $\mathbf{F}_i$ represents the feature of a specific point in $\mathbf{N}(i)$. The process of extracting the feature in the upper layer, *i.e.* f_i^{upper}, can be formulated as follows, which is the local feature of $\mathbf{N}(i)$.

$$f_i^{\text{upper}} = g(\mathbf{F}_i) = \underset{i=1,\dots,K}{\mathbf{MAX}}(MLP(\mathbf{F}_i)), \tag{14}$$

where MLP is an MLP network with a few layers; $MLP(\mathbf{F}_i) \in \mathbb{R}^{D\times K}$; **MAX** is an element-wise max operator as follows. Let $\mathbf{F}'_i = MLP(\mathbf{F}_i)$.

$$\underset{i=1,\dots,K}{\mathbf{MAX}}(\mathbf{F}'_i) = \underset{i=1,\dots,K}{\mathbf{MAX}} \begin{bmatrix} f'_{11} & \cdots & f'_{1K} \\ \vdots & \ddots & \vdots \\ f'_{D1} & \cdots & f'_{DK} \end{bmatrix} \overset{\textbf{define}}{=\!=\!=} < \max_{k=1,\dots,K} f'_{1k}, \dots, \max_{k=1,\dots,K} f'_{Dk} >^{\top} \tag{15}$$

C INTRODUCTION OF DNNS USED IN COMPARATIVE STUDIES

For a better understanding of different versions of DNNs in the next section, we briefly introduce DNNs used in comparative studies, including PointNet++, PointConv, Point2Sequence, and PointSIFT.

C.1 POINTNET++

PointNet++ (Qi et al., 2017b) is a hierarchical structure composed of a number of *set abstraction* modules (SA module). For each SA module, a set of points is processed and abstracted to produce a new set with fewer elements. An SA module includes four parts: the *Sampling layer*, the *Grouping layer*, the *MLP*, and the *Maxpooling layer*. Given a set of N input points, the *Sampling layer* uses the farthest point sampling algorithm to select a subset of points from the input points, which defines the centroids of local regions, $\{x_i\}, i = 1, \dots, N'$. Then, for each selected point, the *Grouping layer* constructs a local region by using ball query search to find K neighboring points within a radius r. For each local region $\mathbf{N}(i)$ centered at x_i, $\mathbf{F}_i \in \mathbb{R}^{d\times K}$ denotes the intermediate-layer features that correspond to points in $\mathbf{N}(i)$. The *MLP* transforms $\mathbf{F}_i$ into higher dimension features $\mathbf{F}'_i \in \mathbb{R}^{D\times K}$, where $D > d$. Finally, the *Maxpooling layer* encodes $\mathbf{F}'_i$ into a local feature f_i^{upper}, which will be fed to the upper SA module. Please see Appendix B for details about the *Maxpooling layer*.

In this study, the baseline network of PointNet++ is composed of three SA modules and a few fully connected layers. Please see Table 6 (left column) for details about the network architecture.

C.2 POINTCONV

PointConv (Wu et al., 2019) has a similar architecture with PointNet++, *i.e.* hierarchically using a few blocks to extract contextual information. In this study, the baseline network of PointConv is composed of five blocks. Each block is constructed as [*Sample layer*→*Group layer*→*MLP*→Architecture 1→Architecture 2→*Conv layer*].

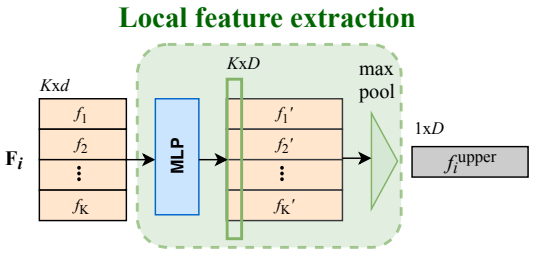

Figure 3: **Illustration of the special max pooling operator.** $\mathbf{F}_i$ denotes features correspond to points in neighborhood $\mathbf{N}(i)$ *w.r.t.* point x_i. Each column of $\mathbf{F}_i$ represents the feature of a specific point in $\mathbf{N}(i)$.

The *Sampling layer* uses the farthest point sampling algorithm to select a subset of points from the input points, which defines the centroids of local regions. Then, for each selected point, the *Grouping layer* constructs a local region by using k-NN search to find K neighboring points. For each local region, the *MLP* transforms features of points in the local region into higher dimension features. Different from PointNet++, PointConv uses the information of density (*i.e.* Architecture 1) and local 3D coordinates (*i.e.* Architecture 2) to reweight the features learned by the *MLP*. Finally, a 1×1 convolution is used to compute the output feature of each local region. Please see Table 8 (left column) for details about the network architecture.

C.3 POINT2SEQUENCE

Point2Sequence (Liu et al., 2018) is composed of five parts: (a) multi-scale area establishment, (b) area feature extraction, (c) encoder-decoder feature aggregation, (d) local region feature aggregation, and (e) shape classification, where parts (a) and (b) makes up Architecture 3 in our study.

Specifically, given a point cloud $\mathbb{X} = \{x_i\}, i = 1, 2, ..., N$, Point2Sequence first uses the farthest point sampling algorithm to select N' points from the input point cloud, $\mathbb{X}' = \{x'_j\}, j = 1, 2, ..., N'$, to define the centroids of local regions $\{\mathbf{N}(j)\}, j = 1, 2, ...N'$. For each local region $\mathbf{N}(j)$, T different scale areas $\{\mathbf{A}(j)^1, ..., \mathbf{A}(j)^t, ..., \mathbf{A}(j)^T\}$ are established by using k-NN search to select $[K_1, ..., K_t, ..., K_T]$ nearest points of x'_j. In this way, multi-scale areas are established. Then, Point2Sequence extracts a feature $f^{\text{upper}}_{j,\text{scale}=K_t} \in \mathbb{R}^d$ for each scale area $\mathbf{A}(j)^t$ by the *MLP* and the *Maxpooling layer* introduced in Appendix C.1. Therefore, for each local region $\mathbf{N}(j)$, a feature sequence $f^{\text{upper}}_j = \{f^{\text{upper}}_{j,\text{scale}=K_1}, ..., f^{\text{upper}}_{j,\text{scale}=K_t}, ..., f^{\text{upper}}_{j,\text{scale}=K_T}\}$ is acquired. Then, f^{upper}_j is aggregated into a d-dimensional feature $\mathbf{r}_j$ by the encoder-decoder feature aggregation part. The sequence encoder-decoder structure used here is an LSTM network, where an attention mechanism is proposed to highlight the importance of different area scales (please see (Liu et al., 2018) for details). Then, a 1024-dimensional global feature is aggregated from the features $\mathbf{r}_j$ of all local regions by the local region feature aggregation part. Finally, the global feature is used for shape classification. Please see Table 9 for details about the network architecture.

C.4 POINTSIFT

PointSIFT (Jiang et al., 2018) adopts the similar hierarchical structure as PointNet++, which is composed of a number of SA modules. The difference is that PointSIFT uses a special orientation encoding unit, *i.e.*, Architecture 4, to learn an orientation-aware feature for each point.

Architecture 4 is a point-wise local feature descriptor that encodes information of eight orientations. Unlike the unordered operator, *e.g. max pooling*, which discards all inputs except for the maximum, Architecture 4 is an ordered operator, which could be more informative.

Architecture 4 first selects 8-nearest points of x_i from eight octants partitioned by the ordering of three coordinates. Since distant points provide little information for the description of local patterns, when no point exists within searching radius r in some octant, x_i will be duplicated as the nearest neighbor of itself. Then, Architecture 4 processes features of 8-nearest neighboring points, $\mathbf{F}^{\text{oe}}_i \in \mathbb{R}^{d \times 2 \times 2 \times 2}$, which reside in a $2 \times 2 \times 2$ cube for local pattern description centering at x_i (as shown in Figure 1 (d)), the three dimensions $2 \times 2 \times 2$ correspond to three axes. An orientation-

Table 6: Different versions of PointNet++, including the original one, the one with Architecture 1, the one with Architecture 2, and the one with Architecture 4. Sample (N) indicates the *Sample layer*, which selects a subset of N points from the input point cloud. Group (r, K) indicates the *Group layer*, which uses the ball query search to find K neighboring points around each sampled point within a radius r. Group (all) means constructing a region with all the input points. MLP $[u_1, \ldots, u_l]$ indicates the MLP with l layers, where u_i is the number of hidden units of the i-th layer. Architecture 4 [d] indicates Architecture 4, which outputs d-dimensional features.

Pointnet++	Pointnet++ with Architecture 1	Pointnet++ with Architecture 2	Pointnet++ with Architecture 4
Sample (512)	Sample (512)	Sample (512)	Sample (512)
Group (0.2,32)	Group (0.2,32)	Group (0.2,32)	Group (0.2,32)
MLP [64,64,128]	MLP [64,64,128]	MLP [64,64,128]	MLP [64,64,128]
Maxpooling	Architecture 1	Architecture 2	Maxpooling
Sample (128)	Maxpooling	Maxpooling	Sample (128)
Group (0.4,64)	Sample (128)	Sample (128)	Group (0.4,64)
MLP [128,128,256]	Group (0.4,64)	Group (0.4,64)	MLP [128,128,256]
Maxpooling	MLP [128,128,256]	MLP [128,128,256]	Maxpooling
Sample (1)	Architecture 1	Architecture 2	Architecture 4 [256]
Group (all)	Maxpooling	Maxpooling	Sample (1)
MLP [256,512,1024]	Sample (1)	Sample (1)	Group (all)
Maxpooling	Group (all)	Group (all)	MLP [256,512,1024]
FC [512,256,40]	MLP [256,512,1024]	MLP [256,512,1024]	Maxpooling
Softmax	Architecture 1	Architecture 2	FC [512,256,40]
	Maxpooling	Maxpooling	Softmax
	FC [512,256,40]	FC [512,256,40]	
	Softmax	Softmax	

encoding convolution, *i.e.* $Conv^{\text{oe}}$, which is a three-stage operator, is used to convolve the $2 \times 2 \times 2$ cube along x, y, and z axis. The three-stage convolution $Conv^{\text{oe}}$ is formulated as:

$$\begin{aligned} f_i^{\text{x}-\text{axis}} &= ReLU(Conv(W_{\text{x}}, \mathbf{F}_i^{\text{oe}})) \in \mathbb{R}^{d \times 2 \times 2 \times 1} \\ f_i^{(\text{x,y})-\text{axis}} &= ReLU(Conv(W_{\text{y}}, f_i^{\text{x}-\text{axis}})) \in \mathbb{R}^{d \times 2 \times 1 \times 1} \\ f_i^{\text{oe}} = f_i^{(\text{x,y,z})-\text{axis}} &= ReLU(Conv(W_{\text{z}}, f_i^{(\text{x,y})-\text{axis}})) \in \mathbb{R}^{d \times 1 \times 1 \times 1} \end{aligned} \tag{16}$$

where $W_{\text{x}} \in \mathbb{R}^{d \times 1 \times 1 \times 2}$, $W_{\text{y}} \in \mathbb{R}^{d \times 1 \times 2 \times 1}$, and $W_{\text{z}} \in \mathbb{R}^{d \times 2 \times 1 \times 1}$ are weights of the convolution operator.

In this way, Architecture 4 learns the orientation-aware feature f_i^{oe} for each point x_i. Such orientation-aware features will be fed to SA modules introduced in Appendix C.1 to extract contextual information.

D Comparative versions of DNNs

D.1 PointNet++:

In this study, we reconstructed the PointNet++ (Qi et al., 2017b) using four specific modules. Table 6 and Table 7 compare the different versions of PointNet++, including the original one, the one with Architecture 1 (Wu et al., 2019), the one with Architecture 2 (Wu et al., 2019), the one with Architecture 4 (Jiang et al., 2018), and the one with Architecture 3 (Liu et al., 2018).

To obtain the PointNet++ with Architecture 1 (as shown in Table 6), we added modules of Architecture 1 after all the *MLPs* in PointNet++, *i.e.* the output of the *MLP* was reweighted by the weights learned by Architecture 1. Architecture 1 used in this study was an MLP with two layers, the first layer contained 16 hidden units, and the second layer contained 1 hidden unit. This network was designed to verify the effect of Architecture 1 on the adversarial robustness.

To obtain the PointNet++ with Architecture 2 (as shown in Table 6), we added modules of Architecture 2 after all the *MLPs* in PointNet++, *i.e.* the output of the *MLP* was reweighted by the weights learned by Architecture 2. Architecture 2 used in this study was an MLP with a single-layer, which contained 32 hidden units. This network was designed to verify the effect of Architecture 2 on the rotation robustness.

To obtain the PointNet++ with Architecture 4 (as shown in Table 6), we added the module of Architecture 4 before the last *Sample layer* in PointNet++. This network was designed to verify the effect of Architecture 4 on the rotation robustness.

Table 7: The original PointNet++ and the PointNet++ with Architecture 3.

PointNet++	PointNet++ with Architecture 3		
Sample (512)	Sample (512)		
Group (0.2,32)	Group1 (0.1,16)	Group2 (0.2,32)	Group3 (0.4,128)
MLP [64,64,128]	MLP1 [32,32,64]	MLP2 [64,64,128]	MLP3 [64,96,128]
Maxpooling	Maxpooling	Maxpooling	Maxpooling
Sample (128)	Multi-Scale Feature Aggregation		
Group (0.4,64)	Sample (128)		
MLP [128,128,256]	Group1 (0.2,32)	Group2 (0.4,64)	Group3 (0.8,128)
Maxpooling	MLP1 [64,64,128]	MLP2 [128,128,256]	MLP3 [128,128,256]
Sample (1)	Maxpooling	Maxpooling	Maxpooling
Group (all)	Multi-Scale Feature Aggregation		
MLP [256,512,1024]	Sample (1)		
Maxpooling	Group (all)		
FC [512,256,40]	MLP [256,512,1024]		
Softmax	FC [512,256,40]		
	Softmax		

Table 8: Different versions of PointConv, including the original one, the one without Architecture 1, the one without Architecture 2. Here Group (K) indicates the *Group layer*, which uses the k-NN search to find K neighboring points around each sampled point.

PointConv	PointConv without Architecture 1	PointConv without Architecture 2
Sample (1024)	Sample (1024)	Sample (1024)
Group (32)	Group (32)	Group (32)
MLP [32,32]	MLP [32,32]	MLP [32,32]
Architecture 1	Architecture 2	Architecture 1
Architecture 2	Conv [64]	Conv [64]
Conv [64]	Sample (256)	Sample (256)
Sample (256)	Group (32)	Group (32)
Group (32)	MLP [64,64]	MLP [64,64]
MLP [64,64]	Architecture 2	Architecture 1
Architecture 1	Conv [128]	Conv [128]
Architecture 2	Sample (64)	Sample (64)
Conv [128]	Group (32)	Group (32)
Sample (64)	MLP [128,128]	MLP [128,128]
Group (32)	Architecture 2	Architecture 1
MLP [128,128]	Conv [256]	Conv [256]
Architecture 1	Sample (36)	Sample (36)
Architecture 2	Group (32)	Group (32)
Conv [256]	MLP [256,256]	MLP [256,256]
Sample (36)	Architecture 2	Architecture 1
Group (32)	Conv [512]	Conv [512]
MLP [256,256]	Sample (1)	Sample (1)
Architecture 1	Group (all)	Group (all)
Architecture 2	MLP [512,512]	MLP [512,512]
Conv [512]	Architecture 2	Architecture 1
Sample (1)	Conv [1024]	Conv [1024]
Group (all)	FC [512,128,40]	FC [512,128,40]
MLP [512,512]	Softmax	Softmax
Architecture 1		
Architecture 2		
Conv [1024]		
FC [512,128,40]		
Softmax		

To obtain the PointNet++ with Architecture 3 (as shown in Table 7), we used the multi-scale version of PointNet++ designed in (Qi et al., 2017b). Compared with the single-scale version of PointNet++ (as shown in Table 6 (left)), the multi-scale version added two blocks after the first *Sample layer*, *i.e.* $[\textit{Group1}(16) \rightarrow \textit{MLP1}[32, 32, 64] \rightarrow \textit{Maxpooling}]$ and $[\textit{Group1}(128) \rightarrow \textit{MLP1}[64, 96, 128] \rightarrow \textit{Maxpooling}]$, The multi-scale version added another two blocks after the second *Sample layer*, *i.e.* $[\textit{Group1}(32) \rightarrow \textit{MLP1}[64, 64, 128] \rightarrow \textit{Maxpooling}]$ and $[\textit{Group1}(128) \rightarrow \textit{MLP1}[128, 128, 256] \rightarrow \textit{Maxpooling}]$. In this way, the multi-scale version of PointNet++ extracted two more scale features.

Table 9: Illustration of the original Point2Sequence network architecture. Here Group (K) indicates the *Group layer*, which uses the k-NN search to find K neighboring points around each sampled point.

Point2Sequence			
Sample (384)			
Group1 (16) MLP1 [32,64,128] Maxpooling	Group2 (32) MLP2 [64,64,128] Maxpooling	Group3 (64) MLP3 [64,64,128] Maxpooling	Group4 (128) MLP4 [128,128,128] Maxpooling
Multi-Scale Feature Aggregation			
LSTM [128]			
Sample (1)			
Group (all)			
MLP [256,512,1024]			
Maxpooling			
FC [512,256,40]			
Softmax			

Table 10: Illustration of the Point2Sequence with Architecture 1.

Point2Sequence with Architecture 1			
Sample (384)			
Group1 (16) MLP1 [32,64,128] Maxpooling	Group2 (32) MLP2 [64,64,128] Maxpooling	Group3 (64) MLP3 [64,64,128] Maxpooling	Group4 (128) MLP4 [128,128,128] Maxpooling
Multi-Scale Feature Aggregation			
LSTM [128]			
Sample (1)			
Group (all)			
MLP [256,512,1024]			
Architecture 1			
Maxpooling			
FC [512,256,40]			
Softmax			

This network was used to verify the effect of Architecture 3 on the adversarial robustness and the neighborhood inconsistency.

D.2 PointConv:

Table 8 compares different versions of PointConv (Wu et al., 2019), including the original one, the one without Architecture 1 (Wu et al., 2019), and the one without Architecture 2 (Wu et al., 2019).

To obtain the PointConv without Architecture 1 (as shown in Table 8 (middle column)), we removed all the five modules of Architecture 1 from the original PointConv architecture. This network was designed to verify the effect of Architecture 1 on the adversarial robustness.

To obtain the PointConv without Architecture 2 (as shown in Table 8 (right column)), we removed all the five modules of Architecture 2 from the original PointConv architecture. This network was designed to verify the effect of Architecture 2 on the rotation robustness.

D.3 Point2Sequence:

The baseline network of Point2Sequence (as shown in Table 9) extracted features of four different scales, *i.e.*, for each local region centered at point x_i, features were computed using the contextual information of 16, 32, 64, and 128 nearest neighbors of x_i, respectively. To obtain different versions of Point2Sequence for comparison, we removed features of specific scales. We first removed the feature extracted by $[\textit{Group1}(16) \rightarrow \textit{MLP1}[32, 64, 128] \rightarrow \textit{Maxpooling}]$ to obtain the first version of Point2Sequence. We then removed features extracted by $[\textit{Group1}(16) \rightarrow \textit{MLP1}[32, 64, 128] \rightarrow \textit{Maxpooling}]$ and $[\textit{Group1}(32) \rightarrow \textit{MLP1}[64, 64, 128] \rightarrow \textit{Maxpooling}]$ to obtain the second version for comparison. These two versions for comparison were designed to verify the effect of Architecture 3 on the adversarial robustness and the neighborhood inconsistency.

Table 11: Illustration of the Point2Sequnence with Architecture 2.

Point2Sequence with Architecture 2			
Sample (384)			
Group1 (16) MLP1 [32,64,128] Maxpooling	Group2 (32) MLP2 [64,64,128] Maxpooling	Group3 (64) MLP3 [64,64,128] Maxpooling	Group4 (128) MLP4 [128,128,128] Maxpooling
Multi-Scale Feature Aggregation LSTM [128] Sample (1) Group (all) MLP [256,512,1024] Architecture 2 Maxpooling FC [512,256,40] Softmax			

Table 12: Illustration of the Point2Sequnence with Architecture 4.

Point2Sequence with Architecture 4			
Sample (384)			
Group1 (16) MLP1 [32,64,128] Maxpooling	Group2 (32) MLP2 [64,64,128] Maxpooling	Group3 (64) MLP3 [64,64,128] Maxpooling	Group4 (128) MLP4 [128,128,128] Maxpooling
Multi-Scale Feature Aggregation LSTM [128] Architecture 4 [128] Sample (1) Group (all) MLP [256,512,1024] Maxpooling FC [512,256,40] Softmax			

To obtain the Point2Sequence with Architecture 1 (as shown in Table 10), we added the module of Architecture 1 after the last *MLP*, *i.e. MLP* [256,512,1024], in Point2Sequence. This network was designed to verify the effect of Architecture 1 on the adversarial robustness.

To obtain the Point2Sequence with Architecture 2 (as shown in Table 11), we added the module of Architecture 2 after the last *MLP*, *i.e. MLP* [256,512,1024], in Point2Sequence. This network was designed to verify the effect of Architecture 2 on the rotation robustness.

To obtain the Point2Sequence with Architecture 4 (as shown in Table 12), we added the module of Architecture 4 after the *LSTM*. This network was designed to verify the effect of Architecture 4 on the rotation robustness.

D.4 PointSIFT:

To obtain the PointSIFT without Architecture 4 (as shown in Table 13), we removed all the four modules of Architecture 4 from the original PointSIFT. This network was designed to verify whether Architecture 4 can improve the rotation robustness.

E Fixing the contexts of points

In this study, we used the entropy-based method (Ma et al., 2019) to quantify the layerwise information discarding of DNNs. This method assumed the feature space of the concept of a specific object satisfied $\|f' - f\|^2 < \epsilon$, where $f = h(X)$, $f' = h(X')$, $X' = X + \boldsymbol{\delta}$. $\boldsymbol{\delta}$ denotes a random noise. For image processing, changing the pixel values will not change the receptive field of an interneuron, thereby features f and f' are computed using the same set of pixels (as shown in Figure 4 (a)). However, for point cloud processing, changing the coordinates of points will change the "receptive field" of an interneuron, *i.e.* features f and f' are computed using contexts of different set of points (as shown in Figure 4 (b)).

Table 13: Illustration of the PointSIFT without Architecture 4. Here Group (r, K) indicates the *Group layer*, which uses the ball query search to find K neighboring points around each sampled point within a radius r.

PointSIFT	PointSIFT without Architecture 4
Architecture 4 [64]	Sample (1024)
Sample (1024)	Group (0.1,32)
Group (0.1,32)	MLP [64,128]
MLP [64,128]	Maxpooling
Maxpooling	Sample (256)
Architecture 4 [128]	Group (0.2,32)
Sample (256)	MLP [128,256]
Group (0.2,32)	Maxpooling
MLP [128,256]	Sample (64)
Maxpooling	Group (0.4,32)
Architecture 4 [256]	MLP [256,512]
Sample (64)	Maxpooling
Group (0.4,32)	Sample (1)
MLP [256,512]	Group (all)
Maxpooling	MLP [512,1024]
Architecture 4 [512]	Maxpooling
Sample (1)	FC [512,256,40]
Group (all)	Softmax
MLP [512,1024]	
Maxpooling	
FC [512,256,40]	
Softmax	

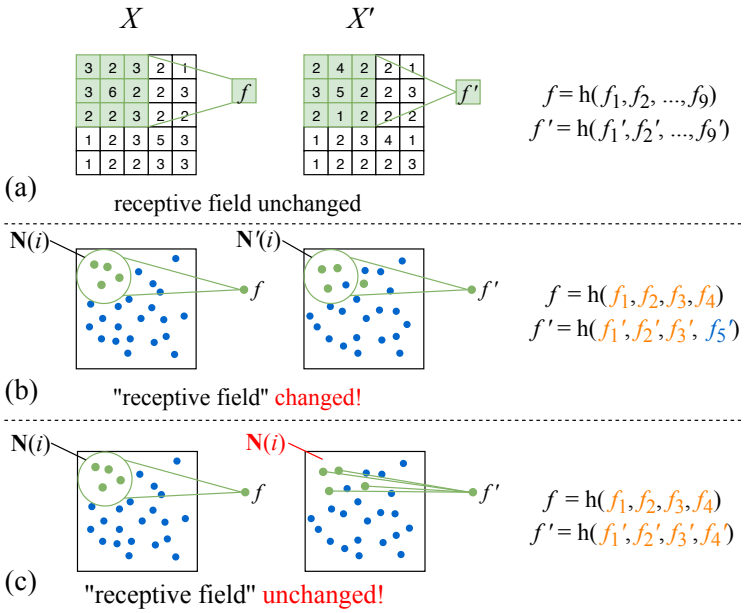

Figure 4: **Illustration of fixed sampling and grouping.** f_i denotes the pixel value/point-wise feature of pixel/point x_i.

To extend the entropy-based method to point cloud processing, we selected the same set of points as the contexts *w.r.t.* x_i and x_i'. In this way, each dimension of f and f' were computed based on the same context (as shown in Figure 4 (c)). To simplify the description, here let f and f' denote local features that are computed using contextual information of x_i and x_i', *i.e.* $f = h(\{f_j | j \in \mathbf{N}(i)\})$, and $f' = h(\{f_j' | j \in \mathbf{N}'(i)\})$, where $\mathbf{N}(i)$ and $\mathbf{N}'(i)$ denote local regions of x_i and x_i'. As shown in Figure 4 (b), changing the coordinates of points will change the "receptive field", *i.e.* $\mathbf{N}'(i) \neq \mathbf{N}(i)$, f' and f are computed using different set of points. In order to keep the "receptive field" unchanged, f' was computed as $f' = h(\{f_j' | j \in \mathbf{N}(i)\})$. In this way, features f and f' were computed using information of the same set of points.

Table 14: Accuracy of different versions of DNNs on ModelNet40, ShapeNet, and 3D MNIST. During the training and testing phases, all the point clouds were rotated by random angles. For each specific architecture, we compared the top-1 accuracy of the network with and without the specific architecture. Experimental results show that, in most cases, removing or adding Architecture 2 and Architecture 4 does not have significant effects on accuracy.

Architecture	Model	ModelNet40		ShapeNet		3D MNIST	
		w/	w/o	w/	w/o	w/	w/o
Architecture 2	PointConv	85.94	85.33	96.07	96.59	85.20	89.10
	PointNet++	82.21	85.65	95.82	97.13	82.50	87.10
	Point2Sequence	85.49	88.45	93.95	96.63	77.30	87.09
Architecture 4	PointSIFT	83.27	82.01	90.22	91.68	84.40	85.70
	PointNet++	85.98	85.64	94.40	97.13	81.30	87.10
	Point2Sequence	81.20	88.45	86.46	96.63	80.60	87.09

Table 15: Accuracy of different versions of DNNs on ModelNet40, ShapeNet, and 3D MNIST. For each specific architecture, we compared the top-1 accuracy of the network with and without the specific architecture. Experimental results show that, in most cases, removing or adding Architecture 1 and Architecture 3 does not have significant effects on accuracy.

Architecture	Model	ModelNet40		ShapeNet		3D MNIST	
		w/	w/o	w/	w/o	w/	w/o
Architecture 1	PointConv	89.02	88.33	98.50	98.53	95.00	95.40
	PointNet++	90.07	89.58	98.82	98.78	96.10	95.00
	Point2Sequence	90.35	92.13	98.88	98.57	99.58	93.90
Architecture 3	PointNet++	89.50	89.58	98.43	98.78	95.60	95.00
	Point2Sequence (4 scales *vs.* 3 scales)	92.13	91.28	98.57	98.71	93.90	94.10
	Point2Sequence (4 scales *vs.* 2 scales)		91.00		98.74		93.00

F ACCURACY COMPARISON OF DIFFERENT VERSIONS OF DNNS

Note that this study does not aim to improve the accuracy of DNNs. This study focuses on the utility analysis of different network architectures. Table 14 and Table 15 list the top-1 accuracy comparison results of different versions of DNNs on three different datasets, including ModelNet40, ShapeNet, and 3D MNIST. Experimental results show that removing or adding a specific architecture has little effects on accuracy.

G RELATIONSHIP WITH LEARNING INTERPRETABLE REPRESENTATIONS:

Compared to the visualization or diagnosis of representations, directly learning interpretable representations is more meaningful to improving the transparency of DNNs. In the capsule nets (Sabour et al., 2017; Zhao et al., 2019b), meaningful capsules, which were composed of a group of neurons, were learned to represent specific entities. Vaughan et al. (2018) learned explainability features with additive nature. The infoGAN (Chen et al., 2016) learned disentangled representations for generative models. The β-VAE Higgins et al. (2017) further developed a measure to quantitatively compare the degree of disentanglement learnt by different models. Zhang et al. (2018) proposed an interpretable CNN, where filters were mainly activated by a certain object part. Fortuin et al. (2018) learned interpretable low-dimensional representations of time series and provided additional explanatory insights. Mott et al. (2019) presented a soft attention mechanism for the reinforcement learning domain, the interpretable output of which can be used by the agent to decide its action.