Verifiability and Predictability: Interpreting Utilities of Network Architectures for Point Cloud Processing: Supplementary Materials

Wen Shen[2,*], Zhihua Wei[2,*], Shikun Huang[2], Binbin Zhang[2], Panyue Chen[2], Ping Zhao[2], Quanshi Zhang[1,†]
[1]Shanghai Jiao Tong University, Shanghai, China
[2]Tongji University, Shanghai, China
{wen_shen,zhihua_wei,hsk,0206zbb,2030793,zhaoping}@tongji.edu.cn,zqs1022@sjtu.edu.cn

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 [7] 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. Details about the function $g(\cdot)$

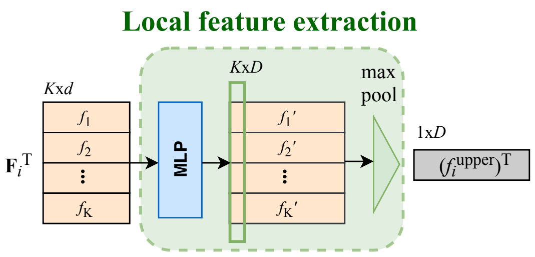


Figure 1. **Illustration of the special max pooling operator $g(\cdot)$.** $\mathbf{F}_i$ denotes features correspond to points in neighborhood $\mathbf{N}(i)$ *w.r.t.* point x_i. Each row of $\mathbf{F}_i^\top$ in the figure represents the feature of a specific point in $\mathbf{N}(i)$.

In point cloud processing, a special element-wise max operator, $g(\cdot)$, is widely used for aggregating a set of neighboring points' features into a local feature. As shown in Figure 1, 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 feature in the upper layer, *i.e.* f_i^{upper}, which is the local feature of $\mathbf{N}(i)$, can be formulated as follows.

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

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

[*]Wen Shen and Zhihua Wei have equal contributions.
[†]Quanshi Zhang is the corresponding author. He is with the John Hopcroft Center and the MoE Key Lab of Artificial Intelligence, AI Institute, at the Shanghai Jiao Tong University, China.

$\mathbf{F}_i' = MLP(\mathbf{F}_i)$.

$$\underset{i=1,\ldots,K}{\mathbf{MAX}}(\mathbf{F}_i') = \underset{i=1,\ldots,K}{\mathbf{MAX}} \begin{bmatrix} f_{11}' & \cdots & f_{1K}' \\ \vdots & \ddots & \vdots \\ f_{D1}' & \cdots & f_{DK}' \end{bmatrix} \overset{\text{define}}{=\!=\!=} < \max_{k=1,\ldots,K} f_{1k}', \ldots, \max_{k=1,\ldots,K} f_{Dk}' >^\top \qquad (2)$$

C. Summaries of relevant technologies in existing DNNs

For the convenience of readers to quickly understand relevant technologies in existing DNNs, we summarize relevant technologies of PointNet++ [9], PointConv [12], Point2Sequence [5], PointSIFT [4], and RSCNN [6] in this section.

C.1. PointNet++

PointNet++ [9] 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, $\{x_i\}, i = 1, \ldots, N'$. 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 dimensional 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 Section 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 1 (left column) for details about the network architecture.

C.2. PointConv

PointConv [12] 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*].

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 dimensional 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 3 (left column) for details about the network architecture.

C.3. Point2Sequence

Point2Sequence [5] 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) make 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 T nearest points of x_j', $[K_1, ..., K_t, ..., K_T]$. In this way, multi-scale areas are established. Then, Point2Sequence extracts a feature $f_{j,\text{scale}=K_t}^{\text{upper}} \in \mathbb{R}^d$ for each scale area $\mathbf{A}(j)^t$ by the *MLP* and the *Maxpooling layer* introduced in Section C.1. In this way, for each local region $\mathbf{N}(j)$, a feature sequence $f_j^{\text{upper}} = \{f_{j,\text{scale}=K_1}^{\text{upper}}, ..., f_{j,\text{scale}=K_t}^{\text{upper}}, ..., f_{j,\text{scale}=K_T}^{\text{upper}}\}$ is obtained. Then, f_j^{upper} 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 [5] 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 4 for details about the network architecture.

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	

Table 1. 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.

C.4. PointSIFT

PointSIFT [4] 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, $\mathbf{F}_i^{\mathrm{oe}} \in \mathbb{R}^{d\times2\times2\times2}$, which reside in a $2\times2\times2$ cube for local pattern description centering at x_i, the three dimensions $2\times2\times2$ correspond to three axes. An orientation-encoding convolution, *i.e.* $Conv^{\mathrm{oe}}$, which is a three-stage operator, is used to convolve the $2\times2\times2$ cube along x, y, and z axis. The three-stage convolution $Conv^{\mathrm{oe}}$ is formulated as:

$$\begin{aligned} f_i^{\mathrm{x-axis}} &= ReLU(Conv(W_{\mathrm{x}}, \mathbf{F}_i^{\mathrm{oe}}))) \in \mathbb{R}^{d\times2\times2\times1} \\ f_i^{\mathrm{(x,y)-axis}} &= ReLU(Conv(W_{\mathrm{y}}, f_i^{\mathrm{x-axis}})) \in \mathbb{R}^{d\times2\times1\times1} \\ f_i^{\mathrm{oe}} = f_i^{\mathrm{(x,y,z)-axis}} &= ReLU(Conv(W_{\mathrm{z}}, f_i^{\mathrm{(x,y)-axis}})) \in \mathbb{R}^{d\times1\times1\times1} \end{aligned} \tag{3}$$

where $W_{\mathrm{x}} \in \mathbb{R}^{d\times1\times1\times2}$, $W_{\mathrm{y}} \in \mathbb{R}^{d\times1\times2\times1}$, and $W_{\mathrm{z}} \in \mathbb{R}^{d\times2\times1\times1}$ 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 Section C.1) to extract contextual information. Please see Table 8 (left column) for details about the network architecture.

C.5. RSCNN

RSCNN [6] adopts the similar hierarchical structure as PointNet++, which is composed of a number of SA modules. The difference is that RSCNN uses a special *relation-shape convolution* (RS-Conv) to learn from the relation, *i.e. the geometric topology constraint among points. Specifically, the convolutional weight for local point set is forced to learn a high-level relation expression from predefined geometric priors, between a sampled point from this point set and the others.*

The goal of the RS-Conv operation is to learn an inductive representation of the neighborhood of each point. Given a point x_i, let $\mathcal{N}(x_i)$ be the neighborhood centered at x_i. Each point $x_j \in \mathcal{N}(x_i)$ is the surrounding point of x_i. The RS-Conv operation consists of two steps: (1) learning from relation, and (2) channel-raising mapping. The first step can be formulated as follows.

$$\mathbf{f}_{P_{\mathrm{sub}}} = \sigma(\mathcal{A}(\{MLP(\mathbf{h}_{ij}) \cdot \mathbf{f}_{x_j}, \forall x_j\})), \quad d_{ij} < r \;\; \forall x_j \in \mathcal{N}(x_i), \tag{4}$$

PointNet++	PointNet++ with Architecture 3		
Sample (512)	Sample (512)		
Group (0.2,32)	**Group (0.1,16)**	Group (0.2,32)	**Group (0.4,128)**
MLP [64,64,128]	**MLP [32,32,64]**	MLP [64,64,128]	**MLP [64,96,128]**
Maxpooling	**Maxpooling**	Maxpooling	**Maxpooling**
Sample (128)	Multi-Scale Feature Aggregation		
Group (0.4,64)	Sample (128)		
MLP [128,128,256]	**Group (0.2,32)**	Group (0.4,64)	**Group (0.8,128)**
Maxpooling	**MLP [64,64,128]**	MLP [128,128,256]	**MLP [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 2. The original PointNet++ and the PointNet++ with Architecture 3.

where d_{ij} is the Euclidean distance between x_i and x_j , and r is the sphere radius.

The step of *learning from relation* can be summarized as follows. First, transform features of all the points in $\mathcal{N}(x_i)$ with function $MLP(\mathbf{h}_{ij}) \cdot \mathbf{f}_{x_j}$, where $MLP(\mathbf{h}_{ij})$ uses a shared MLP to abstract high-level relation expression between points x_i and x_j and $\mathbf{h}_{ij}$ is defined as a compact vector with 10 channels, *i.e.* (3D Euclidean distance, $x_i - x_j$, x_i, x_j). Then, aggregate the transformed features with function $\mathcal{A}$ followed by a nonlinear activator σ. σ is implemented as the special maxpooling operation introduced in Section B.

The step of *channel-raising mapping* is to increase the channel number of $\mathbf{f}_{P_{\text{sub}}}$. Specifically, a shared MLP is added on $\mathbf{f}_{P_{\text{sub}}}$ to achieve the goal. Please see Table 9 (left column) for details about the network architecture.

D. Global architectures of existing DNNs and their revised versions

D.1. PointNet++

In this study, we reconstructed the PointNet++ [9] using four specific modules. Table 1 and Table 2 compare the different versions of PointNet++, including the original one, the one with Architecture 1 [12], the one with Architecture 2 [12], the one with Architecture 4 [4], and the one with Architecture 3 [5].

To obtain the PointNet++ with Architecture 1 (as shown in Table 1), 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 1), 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 1), 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.

To obtain the PointNet++ with Architecture 3 (as shown in Table 2), we used the multi-scale version of PointNet++ designed in [9]. Compared with the single-scale version of PointNet++ (as shown in Table 1 (left)), the multi-scale version added two blocks after the first *Sample layer*, *i.e.* $[\textit{Group}(16) \rightarrow \textit{MLP}[32,32,64] \rightarrow \textit{Maxpooling}]$ and $[\textit{Group}(128) \rightarrow \textit{MLP}[64,96,128] \rightarrow \textit{Maxpooling}]$, The multi-scale version added another two blocks after the second *Sample layer*, *i.e.* $[\textit{Group}(32) \rightarrow \textit{MLP}[64,64,128] \rightarrow \textit{Maxpooling}]$ and $[\textit{Group}(128) \rightarrow \textit{MLP}[128,128,256] \rightarrow \textit{Maxpooling}]$. In this way, the multi-scale version of PointNet++ extracted two more scale features. This network was used to verify the effect of Architecture 3 on the adversarial robustness and the neighborhood inconsistency.

D.2. PointConv

Table 3 compares different versions of PointConv [12], including the original one, the one without Architecture 1 [12], and the one without Architecture 2 [12].

To obtain the PointConv without Architecture 1 (as shown in Table 3 (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

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		

Table 3. 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.

on the adversarial robustness.

To obtain the PointConv without Architecture 2 (as shown in Table 3 (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 4) 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 $[\textit{Group}(16) \rightarrow \textit{MLP}[32, 64, 128] \rightarrow \textit{Maxpooling}]$ to obtain the first version of Point2Sequence. We then removed features extracted by $[\textit{Group}(16) \rightarrow \textit{MLP}[32, 64, 128] \rightarrow \textit{Maxpooling}]$ and $[\textit{Group}(32) \rightarrow \textit{MLP}[64, 64, 128] \rightarrow \textit{Maxpooling}]$ 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.

To obtain the Point2Sequence with Architecture 1 (as shown in Table 5), 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 6), 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 7), 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.

Point2Sequence			
Sample (384)			
Group (16) MLP [32,64,128] Maxpooling	Group (32) MLP [64,64,128] Maxpooling	Group (64) MLP [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 4. Illustration of the original Point2Sequnence 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 with Architecture 1			
Sample (384)			
Group (16) MLP [32,64,128] Maxpooling	Group (32) MLP [64,64,128] Maxpooling	Group (64) MLP [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			

Table 5. Illustration of the Point2Sequnence with Architecture 1.

Point2Sequence with Architecture 2			
Sample (384)			
Group (16) MLP [32,64,128] Maxpooling	Group (32) MLP [64,64,128] Maxpooling	Group (64) MLP [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 6. Illustration of the Point2Sequnence with Architecture 2.

Point2Sequence with Architecture 4			
Sample (384)			
Group (16) MLP [32,64,128] Maxpooling	Group (32) MLP [64,64,128] Maxpooling	Group (64) MLP [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			

Table 7. Illustration of the Point2Sequnence with Architecture 4.

D.4. PointSIFT

To obtain the PointSIFT without Architecture 4 (as shown in Table 8), 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.

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	

Table 8. 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.

D.5. RSCNN

In this study, we reconstructed the RSCNN [6] using four specific modules. Table 9 and Table 10 compare the different versions of RSCNN, including the original one, the one with Architecture 1 [12], the one with Architecture 2 [12], the one with Architecture 4 [4], and the one with Architecture 3 [5].

RSCNN	RSCNN with Architecture 1	RSCNN with Architecture 2	RSCNN with Architecture 4
Sample (512)	Sample (512)	Sample (512)	Sample (512)
Group (0.23,48)	Group (0.23,48)	Group (0.23,48)	Group (0.23,48)
RS-Conv: MLP [64,16]	RS-Conv: MLP [64,16]	RS-Conv: MLP [64,16]	RS-Conv: MLP [64,16]
RS-Conv: Maxpooling	**Architecture 1**	**Architecture 2**	RS-Conv: Maxpooling
RS-Conv: MLP [128]	RS-Conv: Maxpooling	RS-Conv: Maxpooling	RS-Conv: MLP [128]
Sample (128)	RS-Conv: MLP [128]	RS-Conv: MLP [128]	Sample (128)
Group (0.32,64)	Sample (128)	Sample (128)	Group (0.32,64)
RS-Conv: MLP [32,128]	Group (0.32,64)	Group (0.32,64)	RS-Conv: MLP [32,128]
RS-Conv: Maxpooling	RS-Conv: MLP [32,128]	RS-Conv: MLP [32,128]	RS-Conv: Maxpooling
RS-Conv: MLP [512]	**Architecture 1**	**Architecture 2**	RS-Conv: MLP [512]
MLP [1024]	RS-Conv: Maxpooling	RS-Conv: Maxpooling	**Architecture 4 [512]**
Maxpooling	RS-Conv: MLP [512]	RS-Conv: MLP [512]	MLP [1024]
FC [512,256,40]	MLP [1024]	MLP [1024]	Maxpooling
Softmax	Maxpooling	Maxpooling	FC [512,256,40]
	FC [512,256,40]	FC [512,256,40]	Softmax
	Softmax	Softmax	

Table 9. Different versions of RSCNN, including the original one, the one with Architecture 1, the one with Architecture 2, and the one with Architecture 4. The layerwise operation prefixed by "RS-Conv:" indicates that the operation is a component of the RS-Conv operation, which has been well introduced in Section D.5.

To obtain the RSCNN with Architecture 1 (as shown in Table 9), we added modules of Architecture 1 before all *RS-Conv: Maxpooling* modules in the RSCNN. 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.

RSCNN	RSCNN with Architecture 3		
Sample (512)	Sample (512)		
Group (0.23,48)	**Group (0.075,16)**	**Group (0.1,32)**	Group (0.23,48)
RS-Conv: MLP [64,16]	**RS-Conv: MLP [64,16]**	**RS-Conv: MLP [64,16]**	RS-Conv: MLP [64,16]
RS-Conv: Maxpooling	**RS-Conv: Maxpooling**	**RS-Conv: Maxpooling**	RS-Conv: Maxpooling
RS-Conv: MLP [128]	**RS-Conv: MLP [128]**	**RS-Conv: MLP [128]**	RS-Conv: MLP [128]
Sample (128)	Sample (128)		
Group (0.32,64)	**Group (0.1,16)**	**Group (0.15,48)**	Group (0.32,64)
RS-Conv: MLP [32,128]	**RS-Conv: MLP [32,128]**	**RS-Conv: MLP [32,128]**	RS-Conv: MLP [32,128]
RS-Conv: Maxpooling	**RS-Conv: Maxpooling**	**RS-Conv: Maxpooling**	RS-Conv: Maxpooling
RS-Conv: MLP [512]	**RS-Conv: MLP [512]**	**RS-Conv: MLP [512]**	RS-Conv: MLP [512]
MLP [1024]	MLP [1024]		
Maxpooling	Maxpooling		
FC [512,256,40]	FC [512,256,40]		
Softmax	Softmax		

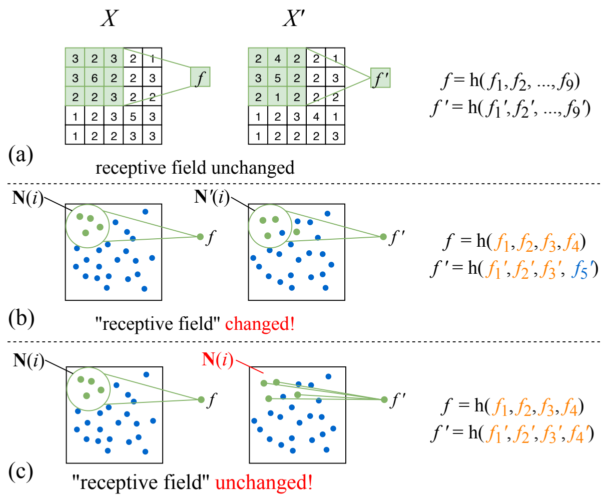
Table 10. The original RSCNN and the RSCNN with Architecture 3. The layerwise operation prefixed by "RS-Conv:" indicates that the operation is a component of the RS-Conv operation, which has been well introduced in Section D.5.

To obtain the RSCNN with Architecture 2 (as shown in Table 9), we added modules of Architecture 2 before all *RS-Conv: Maxpooling* modules in the RSCNN. 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 RSCNN with Architecture 4 (as shown in Table 9), we added the module of Architecture 4 after the *RS-Conv: MLP [512] layer* in the RSCNN. This network was designed to verify the effect of Architecture 4 on the rotation robustness.

To obtain the RSCNN with Architecture 3 (as shown in Table 10), we added two blocks after the first *Sample layer*, *i.e.* [$\textit{Group}(0.075, 16) \rightarrow \textit{RS-Conv: MLP}[64, 16] \rightarrow \textit{RS-Conv: Maxpooling} \rightarrow \textit{RS-Conv: MLP}[128]$ and [$\textit{Group}(0.1, 32) \rightarrow \textit{RS-Conv: MLP}[64, 16] \rightarrow \textit{RS-Conv: Maxpooling} \rightarrow \textit{RS-Conv: MLP}[128]$. The multi-scale version added another two blocks after the second *Sample layer*, *i.e.* [$\textit{Group}(0.1, 16) \rightarrow \textit{RS-Conv: MLP}[32, 128] \rightarrow \textit{RS-Conv: Maxpooling} \rightarrow \textit{RS-Conv: MLP}[512]$ and [$\textit{Group}(0.15, 48) \rightarrow \textit{RS-Conv: MLP}[32, 128] \rightarrow \textit{RS-Conv: Maxpooling} \rightarrow \textit{RS-Conv: MLP}[512]$. In this way, the multi-scale version of RSCNN extracted two more scale features. This network was used to verify the effect of Architecture 3 on the adversarial robustness and the neighborhood inconsistency.

E. Implementation details about how to extend the entropy-based method [7]



X
X'
f
f'
f = h(f1, f2, ..., f9)
f' = h(f1', f2', ..., f9')
(a)
receptive field unchanged
N(i)
N'(i)
f = h(f1, f2, f3, f4)
f' = h(f1', f2', f3', f5')
(b)
"receptive field" changed!
N(i)
N(i)
f = h(f1, f2, f3, f4)
f' = h(f1', f2', f3', f4')
(c)
"receptive field" unchanged!

Figure 2. **Illustration of fixed sampling and grouping.** f_i denotes the pixel value/point-wise feature of pixel/point x_i.

In this study, we used the entropy-based method [7] 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 $f = h(X)$, $f' = h(X')$, $X' = X + \boldsymbol{\delta}$. $\boldsymbol{\delta}$ denoted a random noise. For image processing, changing the pixel values would not change the receptive field of an interneuron, thereby features f and f' were computed using the same set of pixels (as shown in Figure 2 (a)). However, for point cloud processing, changing the coordinates of points would change the "receptive field" of an interneuron, *i.e.*

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	90.84	98.88	98.57	99.58	93.90
	RSCNN	92.02	92.46	98.50	98.40	99.10	99.30
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
	RSCNN	85.72	86.29	95.86	96.88	81.93	82.17
Architecture 3	PointNet++	89.50	89.58	98.43	98.78	95.60	95.00
	RSCNN	92.63	92.46	99.20	98.40	99.23	99.30
	Point2Sequence (4 scales *vs.* 3 scales)	90.84	91.28	98.57	98.71	93.90	94.10
	Point2Sequence (4 scales *vs.* 2 scales)		91.00		98.74		93.00
Architecture 4	PointSIFT	83.27	84.01	96.26	91.68	84.40	90.93
	PointNet++	85.98	85.64	94.40	97.13	88.81	87.10
	Point2Sequence	81.20	88.45	93.51	96.63	78.33	87.09
	RSCNN	85.03	86.29	97.05	96.88	83.01	82.17

Table 11. Accuracy of different versions of DNNs on ModelNet40, ShapeNet, and 3D MNIST. For DNNs with or without Architecture 2 and Architecture 4, during the training time, all point clouds were rotated by random angles. For DNNs with or without Architecture 1 and Architecture 3, during the training time, all point clouds were rotated around z-axis. We compared the top-1 accuracy of the network with and without each specific architecture.

features f and f' were computed using contexts of different set of points (as shown in Figure 2 (b)).

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 2 (c)). To simplify the description, here let f and f' denote local features that were 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 $\mathbf{N}'(i)$ denoted local regions of x_i and x'_i. As shown in Figure 2 (b), changing the coordinates of points would change the "receptive field", *i.e.* $\mathbf{N}'(i) \neq \mathbf{N}(i)$, f' and f were 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.

F. Classification 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 11 lists the top-1 accuracy comparison results of different versions of DNNs on three different datasets, including ModelNet40, ShapeNet, and 3D MNIST.

From Table 11 we can see, adding Architecture 1 had relatively equal positive and negative effects on performance. For PointConv, the network with Architecture 1 performed better than the network without Architecture 1 on the ModelNet40 dataset, while worse on the ShapeNet dataset and the 3D MNIST dataset. For PointNet++, the network with Architecture 1 performed better than the network without Architecture 1 on all three datasets. For Point2Sequence, the network with Architecture 1 performed better than the network without Architecture 1 on the ShapeNet dataset and the 3D MNIST dataset, while worse on the ModelNet40 dataset. For RSCNN, the network with Architecture 1 performed better than the network without Architecture 1 on the ShapeNet dataset, while worse on the ModelNet40 dataset and the 3D MNIST dataset.

Experimental results indicate that adding Architecture 2 to existing DNNs had negative effects on performance. For PointConv, the network with Architecture 2 performed better than the network without Architecture 2 on the ModelNet40 dataset, while worse on the ShapeNet dataset and the 3D MNIST dataset. For PointNet++, Point2Sequence, and RSCNN, networks with Architecture 2 performed worse than networks without Architecture 2 on all three datasets.

Experimental results show that adding Architecture 3 had relatively equal positive and negative effects on performance. For PointNet++, the network with Architecture 3 performed better than the network without Architecture 3 on the 3D MNIST dataset, while worse on the ModelNet40 dataset and the ShapeNet dataset. For RSCNN, the network with Architecture 3 performed better than the network without Architecture 3 on the ModelNet40 dataset and the ShapeNet dataset, while worse on the 3D MNIST dataset. For Point2Sequence, removing features extracted from neighborhoods with different scales decreased the accuracy on the ModelNet40 dataset, while increased the accuracy on the ShapeNet dataset.

Experimental results indicate that, in most cases, adding Architecture 4 to existing DNNs had negative effects on performance. For PointSIFT, networks with Architecture 4 performed better than networks without Architecture 4 on the ShapeNet dataset, while worse than networks without Architecture 4 on the ModelNet40 dataset and the 3D MNIST dataset. For PointNet++, networks with Architecture 4 performed better than networks without Architecture 4 on the ModelNet40 dataset and the 3D MNIST dataset, while worse than networks without Architecture 4 on the ModelNet40 dataset and the ShapeNet dataset. For Point2Sequence, networks with Architecture 4 performed worse than networks without Architecture 4 on all three datasets. For RSCNN, the network with Architecture 4 performed better than the network without Architecture 4 on the ShapeNet dataset and the 3D MNIST dataset, while worse on the ModelNet40 dataset.

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 [10, 14], meaningful capsules, which were composed of a group of neurons, were learned to represent specific entities. [11] learned explainability features with additive nature. The infoGAN [1] learned disentangled representations for generative models. The β-VAE [3] further developed a measure to quantitatively compare the degree of disentanglement learnt by different models. [13] proposed an interpretable CNN, where filters were mainly activated by a certain object part. [2] learned interpretable low-dimensional representations of time series and provided additional explanatory insights. [8] 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.

References

[1] 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*, pages 2172–2180, 2016. 10

[2] 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. 10

[3] 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. 10

[4] 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. 2, 3, 4, 7

[5] 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. 2, 4, 7

[6] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019. 2, 3, 7

[7] Haotian Ma, Yinqing Zhang, Fan Zhou, and Quanshi Zhang. Quantifying layerwise information discarding of neural networks. *arXiv preprint arXiv:1906.04109*, 2019. 1, 8

[8] 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. 10

[9] 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*, pages 5099–5108, 2017. 2, 4

[10] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017. 10

[11] 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. 10

[12] Wenxuan Wu, Zhongang Qi, and Fuxin Li. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conferenc e on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019. 2, 4, 7

[13] 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*, pages 8827–8836, 2018. 10

[14] 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*, pages 1009–1018, 2019. 10