PointCNN

Yangyan Li* Rui Bu Mingchao Sun Baoquan Chen
Shandong University, China

ABSTRACT

We present a simple and general framework for feature learning from point cloud. The key to the success of CNNs is the convolution operator that is capable of leveraging spatially-local correlation in data represented densely in grids (e.g. images). However, point cloud are irregular and unordered, thus a direct convolving of kernels against the features associated with the points will result in deserting the shape information while being variant to the orders. To address these problems, we propose to learn a $\mathcal{X}$-transformation from the input points, and then use it to simultaneously weight the input features associated with the points and permute them into latent potentially canonical order, before the element-wise product and sum operations are applied. The proposed method is a generalization of typical CNNs into learning features from point cloud, thus we call it *PointCNN*. Experiments show that PointCNN achieves on par or better performance than state-of-the-art methods on multiple challenging benchmark datasets and tasks.

CCS CONCEPTS

• **Computing methodologies** → **Shape representations**; **Neural networks**; **Shape analysis**;

KEYWORDS

Convolutional Neural Network, Sparse Data, Irregular Domains

1 INTRODUCTION

Spatially-local correlation is an ubiquitous property of various types of data that is independent of the data representation. The convolution operator has shown to be quite effective in exploiting such correlation if the data is represented in regular domains, such as images, and has been the key to the success of CNNs for a variety of tasks [LeCun et al. 2015].

For data that is inherently of less dimension than the ambient space, such as surfaces in 3D space, or line sketches in 2D, it can be more effective if the data is represented as point cloud in the ambient space, rather than a dense grid of the entire space. Not only that, 3D point cloud probably is the most common raw output from 3D sensors, and is becoming more accessible. However, point cloud is irregular and unordered, rendering convolution operator ill-suited for leveraging spatially-local correlation in the data.

We illustrate the problems and challenges of applying convolution on point cloud with Figure 1. Suppose the unordered set of the C dimensional input features are the same $\mathbb{F} = \{f_a, f_b, f_c, f_d\}$ in all the cases in Figure 1, and we have one convolution kernel $\mathbf{K} = [k_\alpha, k_\beta, k_\gamma, k_\delta]^T$ in shape $4 \times C$. In (i), by following canonical order given by the regular grid structure, the features in the local 2×2 patch can be casted into $[f_a, f_b, f_c, f_d]^T$ of shape $4 \times C$, for convolving with $\mathbf{K}$, yielding $f_i = Conv(\mathbf{K}, [f_a, f_b, f_c, f_d]^T)$, where

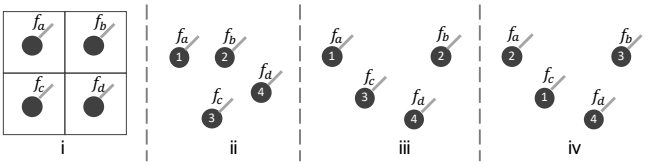

Figure 1: Convolution input from regular grids (i) and point cloud (ii, iii, and iv). In regular grids, each grid cell is associated with a feature. In point cloud, the points are sampled from local neighborhoods, in analogy to local patches in regular grids, and each point is associated with a feature, an order index, as well as its coordinates. However, the lack of regular grids poses the challenge of sorting the points into canonical orders.

$Conv(\cdot, \cdot)$ is simply an element-wise product followed by a sum[1]. In (ii), (iii), and (iv), the points are sampled from local neighborhoods, thus can be in arbitrary orders. By following orders as illustrated in the figure, the input feature set $\mathbb{F}$ can be casted into $[f_a, f_b, f_c, f_d]^T$ in (ii) and (iii), and $[f_c, f_a, f_b, f_d]^T$ in (iv). Based on this, if the convolution operator is directly applied, the output features for the three cases could be computed as:

$$\begin{aligned} f_{ii} &= Conv(\mathbf{K}, [f_a, f_b, f_c, f_d]^T), \\ f_{iii} &= Conv(\mathbf{K}, [f_a, f_b, f_c, f_d]^T), \\ f_{iv} &= Conv(\mathbf{K}, [f_c, f_a, f_b, f_d]^T). \end{aligned} \tag{1}$$

Note that $f_{ii} \equiv f_{iii}$ holds for all cases, while $f_{iii} \neq f_{iv}$ holds for most cases. Now, it is clear that a direct convolving results in deserting the shape information (i.e., $f_{ii} \equiv f_{iii}$) while being variant to the orders (i.e., $f_{iii} \neq f_{iv}$).

In this paper, we propose to learn a $K \times K$ $\mathcal{X}$-transformation from the coordinates of K input points $(p_1, p_2, ..., p_K)$ with multilayer perceptron [Rumelhart et al. 1986], i.e., $\mathcal{X} = MLP(p_1, p_2, ..., p_K)$. then use it to simultaneously weight and permute the input features, and finally apply the typical convolution on the transformed features. We call the process $\mathcal{X}$-Conv, and it is the basic building block for our PointCNN. The $\mathcal{X}$-Conv for (ii), (iii), and (iv) in Figure 1 can be depicted as:

$$\begin{aligned} f_{ii} &= Conv(\mathbf{K}, \mathcal{X}_{ii} \times [f_a, f_b, f_c, f_d]^T), \\ f_{iii} &= Conv(\mathbf{K}, \mathcal{X}_{iii} \times [f_a, f_b, f_c, f_d]^T), \\ f_{iv} &= Conv(\mathbf{K}, \mathcal{X}_{iv} \times [f_c, f_a, f_b, f_d]^T), \end{aligned} \tag{2}$$

where the $\mathcal{X}$s are 4×4 matrices, as $K = 4$ in Figure 1. Note that since $\mathcal{X}_{ii}$ and $\mathcal{X}_{iii}$ are learnt from points in different shapes, they can be different to weight the input features accordingly, thus achieve $f_{ii} \neq f_{iii}$. For $\mathcal{X}_{iii}$ and $\mathcal{X}_{iv}$, if they are learnt to satisfy $\mathcal{X}_{iii} = \mathcal{X}_{iv} \times \Pi$,

*Part of the work was done during Yangyan's Autodesk Research 2017 summer visit.

[1] Actually, this is a special instance of convolution — a convolution that is applied at **one** spatial location. For simplicity, we call it convolution as well.

where Π is the permutation matrix for permuting (c, a, b, d) into (a, b, c, d), then $f_{iii} \equiv f_{iv}$ can be achieved.

From the analysis of the example in Figure 1, it is clear that, with ideal $\mathcal{X}$-transformations, $\mathcal{X}$-Conv is capable of taking the point shapes into consideration, while being independent of point orders. In practice, we found that the learnt $\mathcal{X}$-transformations are far from ideal, especially in terms of the permutation equivalence aspect. Nevertheless, PointCNN built with $\mathcal{X}$-Conv is still significantly better than a direct application of typical convolution on point cloud, and on par or better than state-of-the-art non-convolutional neural networks designed for consuming point cloud data, such as PointNet++ [Qi et al. 2017b].

We explain the detail of $\mathcal{X}$-Conv, as well as PointCNN architectures in Section 3. We show our results on multiple challenging benchmark datasets and tasks in Section 4, together with ablation experiments and visualizations for better understanding of PointCNN.

2 RELATED WORK

Feature Learning from Regular Domains. CNNs were originally designed for taking advantage of spatially-local correlation in images [LeCun et al. 1998], which are represented as pixels in 2D regular grids. Since spatially-local correlation is not only a property of 2D images, but also that of many general data, CNNs have brought about breakthroughs in processing images, video, speech and audio [LeCun et al. 2015]. There has been work in extending CNNs to higher dimensional regular domains, such as 3D voxels [Wu et al. 2015b]. However, as both the input and convolution kernels are of higher dimensions, the amount of both computation and memory inflates dramatically. Octree based approaches have been proposed to save computation by skipping convolution in empty space [Riegler et al. 2017; Wang et al. 2017]. The activations are kept sparse in [Graham and van der Maaten 2017] to retain sparsity in convolved layers. Nevertheless, the kernels themselves are still dense and of high dimension in these approaches. Sparse kernels are proposed in [Li et al. 2016], but this approach cannot be applied recursively for learning hierarchical features. Compared with these methods, PointCNN is sparse in both input representation and convolution kernels.

Feature Learning from Irregular Domains. Stimulated by the rapid advances and demands in 3D sensing, there has been quite a few recent developments in feature learning from 3D point cloud. PointNet [Qi et al. 2017a] and Deep Sets [Zaheer et al. 2017] proposed to achieve input order invariance by the use of symmetric function over inputs. PointNet++ [Qi et al. 2017b] applies PointNet hierarchically for better capturing of local structures. In contrast, PointCNN uses $\mathcal{X}$-Conv in the local feature extraction stage, while PointNet is max-pooling based, and is shown to perform better on various tasks. Besides point cloud, sparse data in irregular domains can be represented as graphs, or meshes, and a line of work have been proposed for feature learning from such representations [Maron et al. 2017; Monti et al. 2017; Yi et al. 2017a]. We refer the interested reader to [Bronstein et al. 2017] for a comprehensive survey of work along these directions.

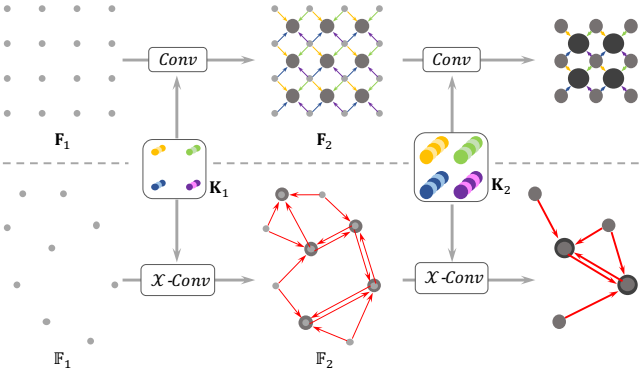

Figure 2: Hierarchical convolution from regular grids (upper) and point cloud (lower). In regular grids, convolution operator is recursively applied on local grid patches, which often reduces the spatial resolution of the grids ($4 \times 4 \rightarrow 3 \times 3 \rightarrow 2 \times 2$), while increases their channel number (visualized by the dots' thickness). Similarly, in point cloud, $\mathcal{X}$-Conv is recursively applied to "project", or "aggregate", information from neighborhoods into less and less representative points ($9 \rightarrow 5 \rightarrow 2$), but each with richer information (again, visualized by the dots' thickness).

Invariance vs. Equivariance. While symmetric function based approaches ([Dieleman et al. 2016; Qi et al. 2017a; Ravanbakhsh et al. 2016; Zaheer et al. 2017]) have theoretical guarantee in terms of achieving order invariance, they come with a price of throwing away information. Hinton et al. proposed a line of pioneering work for addressing this problem through equivariance, rather than invariance [Hinton et al. 2011; Sabour et al. 2017]. The $\mathcal{X}$-transformations in our formulation, ideally, are capable of realizing equivariance, and are demonstrated to be effective in practice. We also found similarity between PointCNN and Spatial Transformer Networks (STNs) [Jaderberg et al. 2015], in the sense that both of them provided a mechanism to "transform" input into latent canonical forms for being further processed. And similar to STNs, though there is no explicit loss or constraint in PointCNN for enforcing the canonicalization, in practice, it turns out that the networks find their ways to leverage the mechanism for learning better. Note that, in PointCNN, the $\mathcal{X}$-transformation is supposed to serve for both weighting and permutation, thus is modelled as a general matrix. This is different than that in [Cruz et al. 2017], where a permutation matrix is the desired output, and is approximated by a doubly stochastic matrix.

3 PointCNN

The hierarchical application of convolution operator is essential to CNNs for learning hierarchical representation. PointCNN shares the same design, and generalizes it to point cloud. In this section, we firstly introduce hierarchical convolution in PointCNN, in analogy to that image CNNs, then explain the core $\mathcal{X}$-Conv operator in detail, and finally present PointCNN architectures for classification and segmentation tasks.

3.1 Hierarchical Convolution

Before we introduce the hierarchical convolution in PointCNN, we briefly go through that in image CNNs, with the illustration of Figure 2 upper. The input to image CNNs is a feature map $\mathbf{F}_1$ in shape $R_1 \times R_1 \times C_1$, where R_1 is the spatial resolution, and C_1 is the feature channel depth. The convolution of kernels $\mathbf{K}$ in shape $K \times K \times C_1 \times C_2$ against local patches in shape $K \times K \times C_1$ from $\mathbf{F}_1$ yields another feature map $\mathbf{F}_2$ in shape $R_2 \times R_2 \times C_2$. Note that in Figure 2 upper, $R_1 = 4$, $K = 2$, and $R_2 = 3$. Compared with $\mathbf{F}_1$, $\mathbf{F}_2$ often is of lower resolution ($R_2 < R_1$) and deeper channels ($C_2 > C_1$), and encodes higher level information. This process is recursively applied, producing feature maps in less and less spatial resolution ($4\times4 \rightarrow 3\times3 \rightarrow 2\times2$ in Figure 2 upper), but deeper and deeper channels (visualized by thicker and thicker dots in Figure 2 upper).

The input to PointCNN is $\mathbb{F}_1 = \{(p_{1,i}, f_{1,i}) : i = 1, 2, ..., N_1\}$, i.e., a set of points $\{p_{1,i} : p_{1,i} \in \mathbb{R}^D\}$, each associated with a feature $\{f_{1,i} : f_{1,i} \in \mathbb{R}^{C_1}\}$. Following the hierarchical construction of image CNNs, we would like to apply $\mathcal{X}$-Conv on $\mathbb{F}_1$ and get a higher level representation $\mathbb{F}_2 = \{(p_{2,i}, f_{2,i}) : f_{2,i} \in \mathbb{R}^{C_2}, i = 1, 2, ..., N_2\}$, where $\{p_{2,i}\}$ is a set representative points of $\{p_{1,i}\}$, i.e., $N_2 < N_1$, and $C_2 > C_1$, so $\mathbb{F}_2$ is of less resolution and deeper feature channels than $\mathbb{F}_1$. When the $\mathcal{X}$-Conv process of turning $\mathbb{F}_1$ into $\mathbb{F}_2$ is recursively applied, the input points with features are "projected", or "aggregated", into less and less points ($9 \rightarrow 5 \rightarrow 2$ in Figure 2 lower), but each with richer and richer features (visualized by thicker and thicker dots in Figure 2 lower).

Note that $\{p_{2,i}\}$ is not necessarily a subset of $\{p_{1,i}\}$. The representative points can be at arbitrary locations in the space whichever are beneficial for the information "projection" or "aggregation". In our implementation, $\{p_{2,i}\}$ is simply a random down-sampling of $\{p_{1,i}\}$ for classification tasks, and farthest point sampling for segmentation tasks, as segmentation tasks are more demanding on a uniform point distribution. We suspect some special points which have shown promising performance in geometric processing, such as Deep Ponits [Wu et al. 2015a], could fit in here as well. However, we leave the exploration of better representative points generation methods as future work.

3.2 $\mathcal{X}$-Conv Operator

$\mathcal{X}$-Conv is the core operator for turning $\mathbb{F}_1$ into $\mathbb{F}_2$. To leverage spatially-local correlation, similar to convolution in image CNNs, $\mathcal{X}$-Conv works with local regions. Since the output features are supposed to be associated with the representative points $\{p_{2,i}\}$, $\mathcal{X}$-Conv takes their neighborhood points in $\{p_{1,i}\}$, as well as the associated features, as input to convolve with.

For simplicity, we denote a representative point in $\{p_{2,i}\}$ as p, and its K neighbors in $\{p_{1,i}\}$ as $\mathbb{N}$, thus the $\mathcal{X}$-Conv input for this specific p is $\mathbb{S} = \{(p_i, f_i) : p_i \in \mathbb{N}\}$. Note that $\mathbb{S}$ is an unordered set. Without loss of generality, $\mathbb{S}$ can be casted into a $K \times D$ matrix $\mathbf{P} = (p_1, p_2, ..., p_K)^T$, and a $K \times C_1$ matrix $\mathbf{F} = (f_1, f_2, ..., f_K)^T$. The trainable parameters of $\mathcal{X}$-Conv is a $K \times (C_1 + C_\delta) \times C_2$ tensor $\mathbf{K}$. With these inputs, we would like to compute feature $\mathbf{F}_p$, which is the input features "projected", or "aggregated" into the representative point p. We depict the $\mathcal{X}$-Conv operator in Algorithm 1, or maybe more concisely, it can be summarized as:

ALGORITHM 1: $\mathcal{X}$-Conv Operator

Input : $\mathbf{K}$, p, $\mathbf{P}$, $\mathbf{F}$
Output: $\mathbf{F}_p$ ▷ Features "projected", or "aggregated", to p

1: $\mathbf{P}' \leftarrow \mathbf{P} - p$ ▷ Move $\mathbf{P}$ to local coordinate system of p
2: $\mathbf{F}_\delta \leftarrow MLP_\delta(\mathbf{P}')$ ▷ **Individually** lift each point into C_δ dim. space
3: $\mathbf{F}_* \leftarrow [\mathbf{F}_\delta, \mathbf{F}]$ ▷ Concatenate $\mathbf{F}_\delta$ and $\mathbf{F}$, $\mathbf{F}_*$ is a $K \times (C_\delta + C_1)$ matrix
4: $\mathcal{X} \leftarrow MLP(\mathbf{P}')$ ▷ Learn the $K \times K$ $\mathcal{X}$-transformation matrix
5: $\mathbf{F}_\mathcal{X} \leftarrow \mathcal{X} \times \mathbf{F}_*$ ▷ Weight and permute $\mathbf{F}_*$ with the learnt $\mathcal{X}$
6: $\mathbf{F}_p \leftarrow \text{Conv}(\mathbf{K}, \mathbf{F}_\mathcal{X})$ ▷ Finally, typical convolution between $\mathbf{K}$ and $\mathbf{F}_\mathcal{X}$

$$\begin{aligned}\mathbf{F}_p &= \mathcal{X}\text{–}Conv(\mathbf{K}, p, \mathbf{P}, \mathbf{F}) \\ &= Conv(\mathbf{K}, MLP(\mathbf{P} - p) \times [MLP_\delta(\mathbf{P} - p), \mathbf{F}]),\end{aligned} \tag{3}$$

where $MLP_\delta(\cdot)$ is a multilayer perceptron applied individually on each point, same to that in PointNet. Note that all the operations involved in building $\mathcal{X}$-Conv, i.e., $\text{Conv}(\cdot, \cdot)$, $MLP(\cdot)$, matrix multiplication $(\cdot)\times(\cdot)$, and $MLP_\delta(\cdot)$, are differentiable. In this case, clearly, $\mathcal{X}$-Conv is differentiable, thus can be plugged into neural network for training by back propagation.

In our implementation, K nearest neighbor search is applied for extracting the K neighboring points. This assumes a more or less uniform distribution of input points. For point cloud with non-uniform point distribution, a radius search can be applied first, and then randomly sample K points out of the radius search results.

Note that trainable kernel $\mathbf{K}$ of $\mathcal{X}$-Conv is a $K \times (C_1 + C_\delta) \times C_2$ tensor. The trainable parameter number is proportional to the number of neighboring points K, instead of being quadratic in image CNNs, or cubic in 3D CNNs. In this sense, we consider our PointCNN sparse in both the input representation and kernels, and it saves both memory and computation. The sparse kernels enables the coupling of long range information without dramatic growth of trainable parameter numbers.

Since Line 4-6 of Algorithm 1 have been covered in the Introduction, here we explain the rationale behind Line 1-3 of Algorithm 1 in detail. Since $\mathcal{X}$-Conv is designed to work on local point regions, the output should not be dependent on the absolute position of p and its neighboring points, but on their relative positions, thus we build local coordinate systems at the representative points and the neighboring points are translated to center around the origins, i.e., $\mathbf{P}' \leftarrow \mathbf{P} - p$ (Line 1 of Algorithm 1). Note that one point may be in the neighborhood of multiple representative points, for example, $p_{1,1}$ is neighboring to both $p_{2,1}$ and $p_{2,2}$ in Figure 3 a and b, thus one point can be at different relative positions in local coordinate systems of different representative points.

It is the local coordinates of neighboring points, together with their associated features, that defines the output features. In other word, besides the associated features, the local coordinates themselves are part of the input features as well. However, the local coordinates are of quite different dimensionality and representation than the associated features. We first lift the coordinates into an higher dimensional and more abstract representation ($\mathbf{F}_\delta \leftarrow MLP_\delta(\mathbf{P}')$, Line 2 of Algorithm 1), and then combine it with the associated features ($\mathbf{F}_* \leftarrow [\mathbf{F}_\delta, \mathbf{F}]$, Line 3 of Algorithm 1) for being further processed (Figure 3 c).

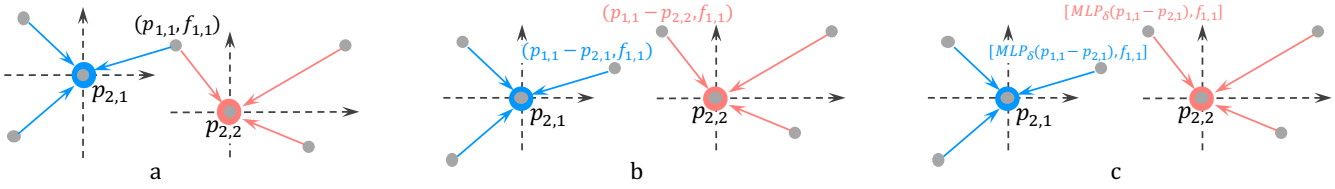

Figure 3: The process for converting point coordinates to features. The neighboring points of representative points are transformed to local coordinate systems of the representative points (a and b). Then the local coordinates of each point are individually lifted into features, and combined with the associated features (c).

The lifting of coordinates into features is through a point-wise $MLP_\delta(\cdot)$, which is the same as that in PointNet and PointNet++. However, the lifted features are not processed by a symmetric function in PointCNN. Instead, they are weighted and potentially permuted, together with the associated features, by the learnt $\mathcal{X}$-transformation. Note that, unlike $MLP_\delta(\cdot)$, $MLP(\cdot)$ is applied on the entire neighboring point coordinates. Thus the resulting $\mathcal{X}$ **is** dependent on the order of the points, and this **is** desired, as $\mathcal{X}$ is supposed to permute $\mathbf{F}_*$ according to the input points, thus it has to be aware of the specific input order.

One nice property of $\mathcal{X}$-Conv is that it handles point cloud with or without additional features in a quite uniform fashion. For input point cloud without any additional features, i.e., $\mathbf{F}$ is empty, the first $\mathcal{X}$-Conv layer uses only $\mathbf{F}_\delta$.

Note that, in theory, $\mathcal{X}$-transformation can be applied on either the features, or the kernels. We opt to apply it on the features, in which way, the follow up operation is a standard Conv operation — an operation that is highly optimized by popular deep learning frameworks. Otherwise, it will result in a convolution between features and the kernels "spawned" by $\mathcal{X}$, which is not common, thus probably not fully optimized.

3.3 PointCNN Architectures

From Figure 2, we can see that the convolution layers in image CNNs and $\mathcal{X}$-Conv layers in PointCNN only differs in two aspects: the way the local regions are extracted ($K \times K$ patches in image CNNs vs. K neighboring points around representative points.) and the way the information from local regions is learnt (Conv in image CNNs vs. $\mathcal{X}$-Conv). Otherwise, there is no much difference in assembling a deep network with the $\mathcal{X}$-Conv layers than that with convolution layers in image CNNs.

In Figure 4 (a), we show a simple PointCNN, with two $\mathcal{X}$-Conv layers that gradually turn the input points (with or without features) into less representation points, but each with richer feature. After the second $\mathcal{X}$-Conv layer, there is only one representative point left, and it received information from all the points from the previous layer. In PointCNN, we can roughly define the receptive field of each representative point as the ratio K/N, where K is the neighboring point number, and N is the point number in the previous layer. With this definition, the only one left point "sees" all the points from previous layer, thus has receptive field 1.0 — it has a global view of the entire shape, thus its features are informative for semantic understanding of the shape. We can add some fully connected

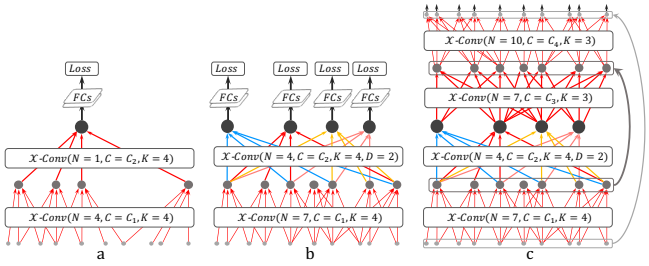

Figure 4: PointCNN architecture for classification (a and b) and segmentation (c), where N and C denote the output representative point number and feature dimensionality, K is the neighboring point number for each representative point, and D is the $\mathcal{X}$-Conv dilation rate.

layers on top of the last $\mathcal{X}$-Conv layer output followed by a loss for training the network.

Note that the number of training samples for the top $\mathcal{X}$-Conv layers of PointCNN in Figure 4 (a) drops rapidly, making it inefficient to train the top $\mathcal{X}$-Conv layers thoroughly. To address this problem, we propose the PointCNN in Figure 4 (b), where more representative points are kept in the $\mathcal{X}$-Conv layers. However, we want to maintain the depth of the network, while keeping the receptive field growth rate, such that the deeper representative points "see" larger and larger portion of the entire shape. We achieve this goal by employing the dilated convolution idea from image CNNs into PointCNN. Instead of always taking the K neighboring points as input, we may uniformly sample K input points from $K \times D$ neighboring points, where D is the dilation rate. In this case, the receptive field increases from K/N to $(K \times D)/N$, without the increase of the actual neighboring point number, nor the kernel size.

In the second $\mathcal{X}$-Conv layer of PointCNN in Figure 4 (b), dilation rate $D = 2$ is used, thus all the four remaining representative points "see" the entire shape, and all of them are suitable for making predictions. Note that, in this way, we can train the top $\mathcal{X}$-Conv layers more thoroughly, as much more connections are involved in the network, compared with that in PointCNN of Figure 4 (a). In testing time, the output from the multiple representative points are averaged right before the $softmax$ to stabilize the prediction. This design is quite similar to that of Network in Network [Lin et al.

Method	Input	Core Operator	ModelNet40	ScanNet
MVCNN [Su et al. 2015]	Images	2D Conv	90.1	-
FPNN [Li et al. 2016]	3D Dist. Field	1D Conv	87.5	-
Vol. CNN [Qi et al. 2016]	Voxels	3D Conv	89.9	74.9
O-CNN [Wang et al. 2017]	Octree Voxels	Sparse 3D Conv	90.6	-
PointNet [Qi et al. 2017a]	Point Cloud	Pointwise MLP	89.2	-
PointNet++ [Qi et al. 2017b]	Point Cloud	Multiscale Pointwise MLP	90.7	76.1
PointCNN	Point Cloud	$\mathcal{X}$-Conv	**91.7**	**77.9**

Table 1: Comparisons of classification accuracy (%) on ModelNet40 [Wu et al. 2015b] and ScanNet [Dai et al. 2017].

2014]. PointCNN in the denser style (Figure 4 (b)) is the one we used for classification tasks.

For segmentation tasks, high resolution pointwise output is required, and this can be realized by building PointCNN following Conv-DeConv [Noh et al. 2015] architecture, where the DeConv part is responsible of propagating global information into high resolution predictions (see Figure 4 (c)). Note that both the "Conv" and "DecConv" in PointCNN segmentation network are the same $\mathcal{X}$-Conv operator. For "DeConv" layers, the only difference with the "Conv" layers is that there are more points, but less feature channels, in the output than that in the input. And the higher resolution points for the "DeConv" layers are forwarded from earlier "Conv" layers, following the design of U-Net [Ronneberger et al. 2015].

ELU [Clevert et al. 2016] is the nonlinear activation function used in PointCNN, as we found it is more stable and performs slightly better than ReLU [Glorot et al. 2011]. Batch normalization [Ioffe and Szegedy 2015] is applied on $\mathbf{P}'$, $\mathbf{F}_p$ and the fully connected layer outputs (except for that of the last fully connected layer) for reducing internal covariate shift. It is important to note that batch normalization should not be applied in MLP_δ and MLP, since $\mathbf{F}_*$ and $\mathcal{X}$, especially $\mathcal{X}$, are supposed to be quite specific for a particular representative point. For the Conv in Line 6 of Algorithm 1, separable convolution [Chollet 2016] is used for reducing parameter number and computation than that of typical convolution. We use ADAM optimizer [Kingma and Ba 2014] with initial learning rate 0.01 for the training of PointCNN.

Dropout is applied before the last fully connected layer for reducing over-fitting. We also employed the "subvolume supervision" idea from [Qi et al. 2016] for addressing over-fitting problem. In the last $\mathcal{X}$-Conv layers, the receptive field is set to be less than 1, such that only a partial information is "seen" by the representative points in the last $\mathcal{X}$-Conv layers. The network is pushed to learn harder from the partial information at training time, and performs better in testing time.

In this paper, PointCNN is demonstrated with simple feed forward networks on classification tasks, and simple feed forward layers plus skip-links in segmentation network. However, since the interface $\mathcal{X}$-Conv exposed to its input and output layers is quite similar to that of Conv, we think many advanced neural network techniques from image CNNs can be adopted to work with $\mathcal{X}$-Conv, e.g., recurrent PointCNN. We leave the exploration along these directions as future work.

Data augmentation. For the training of the parameters in $\mathcal{X}$-Conv, clearly, it is not beneficial if the neighboring points are always the same set in the same order for a specific representative point. To improve the generalizability, we propose to randomly sample and shuffle the input points, such that both the neighboring point sets and order can be different from batch to batch. To train a model that takes N points as input, $\mathcal{N}(N, (N/8)^2)$ points are used for the training, where $\mathcal{N}$ denotes Gaussian distribution. We found this strategy is crucial for the training of PointCNN.

4 EXPERIMENTS

4.1 Datasets and Tasks

We conducted extensive evaluation of PointCNN for classification task on six benchmark datasets, and segmentation task on three benchmark datasets. The PointCNN architecture details for the tasks on these datasets can be found in the Appendix A.1. Before we dive into the experimental results, we introduce the datasets:

- Object datasets: ModelNet40 [Wu et al. 2015b] and ShapeNet Parts [Yi et al. 2016].
  - ModelNet40 is composed of 12,311 3D mesh models from 40 categories, with a 9,843/2,468 training/testing split. Both the gravity and "facing" directions of the models are aligned in the dataset. However, the "facing" direction is ignored by random horizontal rotations to better approximate the scenarios in real world applications. We use the point cloud conversion of ModelNet40 provided by [Qi et al. 2017a] as our input, where 2,048 points are sampled from each mesh, and we further sample $\mathcal{N}(10, 24, 128^2)$ points to train a model for testing with 1,024 points on the classification task.
  - ShapeNet Parts contains 16,880 models (14,006/2,874 training/testing split) from 16 shape categories, each annotated with 2 to 6 parts and there are 50 different parts in total. Each point sampled from the models is associated with a part label. The task is to predict the part label for each point, thus a segmentation task, and can be treated as a dense point-wise classification problem. The category label for each model is given, and can be used for trimming irrelevant predictions, same as that in [Graham et al. 2017]. $\mathcal{N}(2048, 256^2)$ points are sampled from each point cloud to train a model for testing with 2,048 input points on the segmentation task. Each testing point cloud is sampled mutliple times to make sure all the points are evaluated at least r ($r = 10$ in our experiments) times at testing time.
- Indoor scene datasets: S3DIS [Armeni et al. 2016] and ScanNet [Dai et al. 2017]. While ModelNet40 and ShapeNet models

Method	ShapeNet Parts	S3DIS	ScanNet
PointNet [Qi et al. 2017a]	83.7	47.6	73.9
PointNet++ [Qi et al. 2017b]	85.1	-	84.5
SyncSpecCNN [Yi et al. 2017a]	84.74	-	-
Pd-Network [Klokov and Lempitsky 2017]	85.49	-	-
SSCN [Graham et al. 2017]	85.98	-	-
SegCloud [Tchapmi et al. 2017]	-	48.92	-
SPGraph [Landrieu and Simonovsky 2017]	-	54.06	-
PointCNN	**86.13**[2]	**62.74 (54.1, w/o RGB)**	**85.1**

Table 2: Segmentation comparisons on ShapeNet Parts [Yi et al. 2016] in part averaged IoU (%), S3DIS [Armeni et al. 2016] in mean IoU (%), and ScanNet [Dai et al. 2017] in per voxel accuracy (%).

are mostly made by 3D modeling tools, S3DIS and ScanNet are from real scans of indoor environments.

- S3DIS contains 3D scans from Matterport scanners in 6 areas including 271 rooms. Each point with RGB features in the scan is annotated with one of the semantic labels from 13 categories. The task is segmentation The data is firstly split by room, and then the rooms are sliced into 1.5m by 1.5m blocks, with 0.3m padding on each side. The sliced blocks are handled in the same way as the object point clouds in ShapeNet Parts. The points in the pading areas serve as context of the internal points, and themselves are not linked to loss in the training phase, nor used for prediction in the testing phase.
- ScanNet contains $1,513$ scanned and reconstructed indoor scenes, with $1,201/312$ scenes for training/testing in semantic voxel labeling of 17 categories. We firstly prepare data in the same way as that of S3DIS to train a segmentation model, and the segmentation results on testing data are then converted into semantic voxel labeling, as that in [Qi et al. 2017b], for a fair comparison with previous methods. The $9,305/2,606$ training/testing object instances from the 17 categories in ScaneNet are also used for evaluating classification task. Note that ScanNet comes with RGB information for each point. However, they are not used in previous methods. To make fair comparisons, we do not use them either.

- 2D sketch datasets: TU-Berlin [Eitz et al. 2012] and Quick Draw [Ha and Eck 2017]. Similar to surfaces in 3D space, line sketches in 2D are inherently of less dimension than the ambient space, and can be represented as point cloud, thus we consider 2D sketches good arena for evaluating neural networks that are designed to consume point cloud data. TU-Berlin has sketches from 250 categories, with 80 sketches from each category, where 2/3 are used for training and the rest 1/3 for testing. Quick Draw is the largest available sketch dataset, with sketches from 345 categories, each with $70,000/2,500$ training/testing samples. We sample $\mathcal{N}(512, 64^2)$ points from the sketch stokes to train a model for testing with 512 points on sketch classification task.
- Image datasets: MNIST and CIFAR10. MNIST and CIFAR10 are widely used for sanity check of image CNNs. Since PointCNN is a generalization of CNNs, we would like to evaluate PointCNN on the point cloud representation of MNIST and CIFAR10. For MNIST, we randomly sample 160 **foreground** pixels and convert them into point cloud representation, with the gray-scale pixel value as the input feature. For CIFAR10, we randomly sampled 512 pixels out of the 32×32 pixels for converting into point cloud with RGB features. Note that there is "shape" information in the MNIST point cloud, sine the point cloud follow the digits' structure, but this is not the case for the CIFAR10 point cloud, where the points are mostly the same blob for all the data samples.

Method	TU-Berlin	Quick Draw
Sketch-a-Net [Yu et al. 2017]	**77.95**	-
AlexNet [Krizhevsky et al. 2012]	68.60	-
PointNet++ [Qi et al. 2017b]	66.53	51.58
PointCNN	67.72	**56.75**

Table 3: Accuracy (%) comparisons on Tu-Berlin [Eitz et al. 2012] and Quick Draw [Ha and Eck 2017] classification.

4.2 Classification and Segmentation Results

Classification results. Classification task is generally considered as the touchstone for the evaluation of a neural network, as a neural network which shows strong performance on classification task can often be adapted to achieve strong performance on other tasks. We evaluate PointCNN on the classification of ModelNet40 and ScanNet 3D objects, and TU-Berlin and Quick Draw 2D sketches.

Since ModelNet40, ScanNet and TU-Berlin datasets are rather small, strong over-fitting is observed if the receptive field of the last $\mathcal{X}$-Conv layers is set to 1. To address this problem, we set the receptive field of the last $\mathcal{X}$-Conv layers to 1/3 for PointCNN of ModelNet40 and ScanNet, 5/8 for that of TU-Berlin and Quick Draw.

We summarize our results on ModelNet40 and ScanNet in Table 1, in comparison with several methods that are designed to consume these data in different representations with different core operators. Compare with the other representations, point cloud probably is the most direct output of 3D sensing. The consistent better performance of PointCNN on these two datasets makes PointCNN a natural choice for endowing 3D sensing pipeline with recognition capability.

[2] An earlier version of this work participated ShapeNet Parts segmentation challenge at ICCV 2017 and was briefly described in the technical report [Yi et al. 2017b]. The major reasons for why it performs less well are three folds: 1. it did not assume the aligned "facing" directions, thus added random horizontal rotations to the data, which made the task harder; 2. random sampling was used for generating the representative points, rather than farthest point sampling; 3. it was using ReLU for activation, and batch normalization was used in MLP_δ and MLP.

	PointCNN	w/o $\mathcal{X}$	w/o $\mathcal{X}$ (wider)	w/o $\mathcal{X}$ (deeper)
Core Layers	$\mathcal{X}$-Conv ×4	*Conv* × 4	*Conv* × 4	*Conv* × 6
# Parameter	0.45M	0.23M	0.49M	0.4M
Accuracy (%)	**91.7**	89.1	86.1	88.3

Table 4: Ablation test of PointCNN variants on ModelNet40 classification. $\mathcal{X}$-Conv is the key to PointCNN performance.

PointCNN results on the classification task of TU-Berlin and Quick Draw sketches are presented in Table 3, where we compare it with the competitive PointNet++, as well as image CNN based methods. PointCNN is better than PointNet++ on both of the two datasets, and the advantage is more prominent on Quick Draw (25M data samples), which is significantly larger than TU-Berlin (0.02M data samples). On TU-Berlin dataset, while the performance of PointCNN is comparable with the widely used generic AlexNet [Krizhevsky et al. 2012] image CNN, there is still a big gap with the specialized Sketch-a-Net [Yu et al. 2017] image CNN. It is interesting to study whether the specialized designs in Sketch-a-Net can be adopted into PointCNN for improving its performance on the sketch datasets.

Segmentation results. Segmentation is a more challenging task than classification, as it requires a finer understanding of the data. We evaluate PointCNN on the segmentation of ShapeNet Parts, S3DIS and ScanNet datasets, and summarize the results in Table 2.

Note that the part averaged IoU metric for ShapeNet Parts is the one used for ICCV 2017 ShapeNet Parts segmentation challenge [Yi et al. 2017b]. It is a weighted average of per category IoU, with the number of shapes in each category as the weights, and the per category IoU is computed for each category first by averaging across all parts on all shapes with the certain category label. Compared with mean IoU, the part averaged IoU puts more emphasis on the correct prediction of small parts.

From Table 2, we can see that PointCNN is better than all the comparison methods, including SSCN [Graham et al. 2017], SegCloud [Tchapmi et al. 2017] and SPGraph [Landrieu and Simonovsky 2017], which are specialized networks designed for segmentation tasks with very competitive performance. Note that the performance of PointCNN on S3DIS is on par with other methods, even if the RGB features are not used, which is a strong indication that PointCNN can effectively leverage "shape" information in point cloud. And the additional performance brought by using RGB features shows that PointCNN can benefit from extra features than the point coordinates to learn even better (from 54.1% to 62.74%, 8.64% improvement).

4.3 Ablation Experiments and Visualizations

Ablation test of the core $\mathcal{X}$-Conv operator. To verify the effectiveness of the core $\mathcal{X}$-Conv operator, we propose PointCNN w/o $\mathcal{X}$ as a baseline method, where Line 4-6 of Algorithm 1 is replaced by $\mathbf{F}_p \leftarrow \text{Conv}(\mathbf{K}, \mathbf{F}_*)$, i.e., the input features are convolved without the transformation of the learnt $\mathcal{X}$. This is all and the only difference between PointCNN and PointCNN w/o $\mathcal{X}$. Compared with PointCNN, the baseline has less trainable parameters, and is more "shallow" due to the removal of $MLP(\cdot)$ in Line 4 of Algorithm 1. To make the comparison fair, based on the baseline, we further propose PointCNN w/o $\mathcal{X}$ (wider) and w/o $\mathcal{X}$(deeper). As indicated by their names, they are wider/deeper than PointCNN w/o $\mathcal{X}$, and have approximately same amount of parameters as PointCNN. The model depth of PointCNN w/o $\mathcal{X}$(deeper) also compensate the depth decrease introduced by the removal of $MLP(\cdot)$ from PointCNN.

The comparison results are summarized in Table 4. Clearly, PointCNN outperforms the proposed variants with a significant margin, and the gap between PointCNN and PointCNN w/o $\mathcal{X}$ is neither due to model parameter number, nor model depth. While the widening and deepening of PointCNN w/o $\mathcal{X}$ bring extra parameters (and model depth) and increase accuracy on training time, they do not generalized well to testing data. In contrast, the extra parameters in $MLP(\cdot)$ of PointCNN introduced for learning $\mathcal{X}$ turned out to be an high-return investment. With these comparisons, we can conclude that $\mathcal{X}$-Conv is the key to the performance of PointCNN.

Note that after the removal $MLP(\cdot)$, there is still $MLP_\delta(\cdot)$ (Line 2 of Algorithm 1, the same module used in PointNet) in the network. Remember that we use separable convolution [Chollet 2016] for $\text{Conv}(\cdot,\cdot)$. In this case, if the depthwise filters are learnt to be some constants, then the separable convolution approximates the mean function, which is symmetric, thus the results can be made independent of the input point order, similar to the max-pooling in PointNet. We suspect this might be the reason why the performance of PointCNNs w/o $\mathcal{X}$ is on par with PointNet.

Stress test on small number of input points. We evaluate PointCNN performance with different number of input points on ModelNet40 classification task, in comparison with PointNet and PointNet++, and summarize the results in Table 6. Note that the performance of PointCNN at 512 (90.7% accuracy) input points is on par with PointNet++ with $1,024$ input points. Moreover, when input point number is further reduced into 64 (88.3% accuracy) and 32 (84.4% accuracy), PointCNN outperforms PointNet/PointNet++ with a 3.9% and 18.3% gap respectively. In such settings, the inference runs at **0.6ms** and **0.3ms** per sample on NVidia GTX 1080 GPU[3], making PointCNN quite promising for real time recognition applications with low resolution point cloud input, such as autonomous driving.

How about if we per-order the neighboring points? PointCNN is designed to consume unordered points, and it is up to the network for permuting the points into a latent potentially canonical order. One interesting question to ask is that whether PointCNN can benefit from some sort of pre-ordering. We verified this hypothesis by sorting the extracted neighboring points according to their coordinates, one axis by one axis. We found that while this strategy sometimes brings a small amount of gain on the training dataset, it is often harmful for the performance on testing dataset, even through the same pre-ordering is applied in both training and testing dataset.

T-SNE visualization of $\mathcal{X}$-Conv features. Note that each representative point, with its neighboring points in a particular order, has a corresponding $\mathbf{F}_*$ and $\mathbf{F}_\mathcal{X}$ in $\mathbb{R}^{K\times C}$, where $C = C_\delta + C_1$. For the same representative point, if its neighboring points in different orders are feed into the network, we get a set of $\mathbf{F}_*$ and $\mathbf{F}_\mathcal{X}$, and we denote them as $\mathcal{F}_*$ and $\mathcal{F}_\mathcal{X}$. Similarly, we define the set of $\mathbf{F}_*$ in PointCNN w/o $\mathcal{X}$ as $\mathcal{F}_o$.

[3] Averaged from 0.12 and 0.06 second per batch with batch size 200.

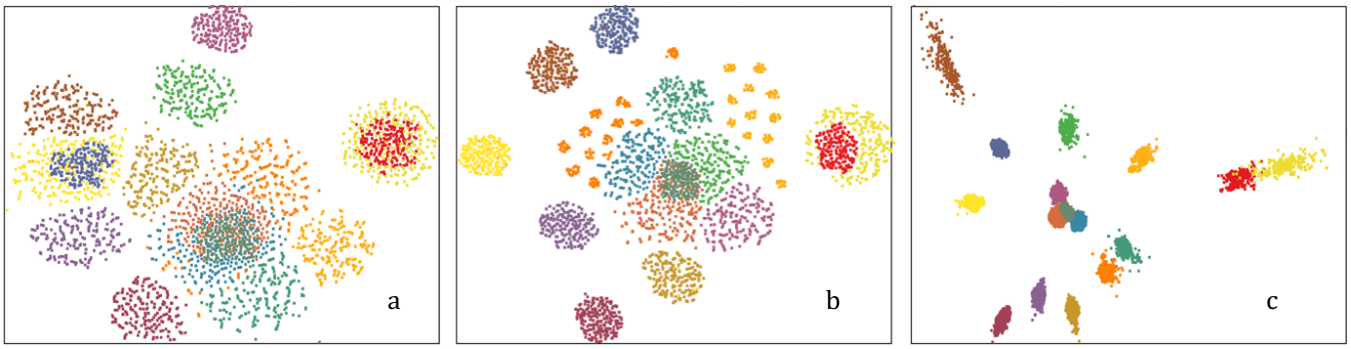

Figure 5: T-SNE visualization of features in PointCNN w/o $\mathcal{X}$ (a), and before (b) and after (c) $\mathcal{X}$-transformation in PointCNN.

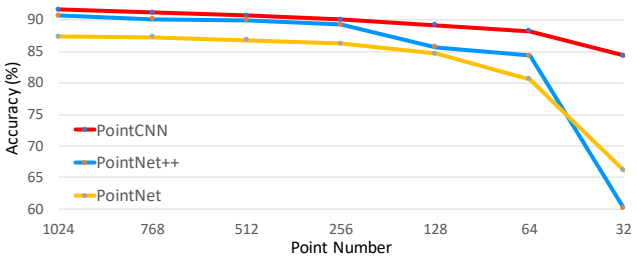

Figure 6: Stress test on ModelNet40 classification.

Clearly, $\mathcal{F}_*$ can be quite scattering in the $\mathbb{R}^{K\times C}$ space, since different of input point orders will result in a different $\mathbf{F}_*$. On the other hand, if the learnt $\mathcal{X}$ can perfectly canonize $\mathbf{F}_*$, $\mathcal{F}_\mathcal{X}$ is supposed to stay at a canonical point in the space.

To verify this, we show T-SNE visualization of $\mathcal{F}_o$, $\mathcal{F}_*$ and $\mathcal{F}_\mathcal{X}$ of 15 randomly picking representative points from ModelNet40 dataset in Figure 5, each with one color, and same color in the sub-figures. Note that $\mathcal{F}_o$ is quite "blended", which indicates that the features from different representative points are not discriminative to each other (Figure 5 (a)). $\mathcal{F}_*$ while being better than $\mathcal{F}_o$, is still "fuzzy" (Figure 5 (b)). In (Figure 5 (c)), $\mathcal{F}_*$ are "concentrated" by $\mathcal{X}$, and the features of each representative points become highly discriminative from each other. Note that even though the "concentration" is far from reaching a point, the improvement is significant, and this visualization explains the extraordinary performance of PointCNN in feature learning.

Distance visualization of features at different hierarchies. Hierarchical feature representation is an important feature of PointCNN. In image CNNs, such hierarchies can be visualized from the 2D grid kernels. However, due to the lack of grid structure, the kernels of PointCNN cannot be observed in such a way. Instead, we opt to visualize PointCNN features at different hierarchies by examining the feature distances of representative points at different hierarchies.

In Figure 7, each point is associated with the feature from a certain hierarchy, and the points with an arrow indication are set as the query points, the feature distance between these points and all the rest points are computed. The points with distances less than 40% of the maximal distances are highlighted with blue, and the darker the color, the more similar the points are to the query points. It is clear that while the low level features of PointCNN capture local geometry, the high level features faithfully encode semantic information.

Model size, memory usage and timing. We implement PointCNN in tensorflow [Abadi et al. 2015]. The actual model size, memory usage and timing depends on the model complexity. Here we report the statistics for two typical settings. The PointCNN for classification with 1024 input points has 0.5M parameters, and can run on NVidia GTX 1080 GPU (8GB GPU memory) with batch size 200, at 0.9/0.23 second per batch at training/inference stage. The PointCNN for segmentation with 2048 input points has 4.5M parameters, and can run on NVidia Tesla P100 GPU (16GB GPU memory) with batch size 32, at 0.44/0.41 second per batch at training/inference stage.

4.4 Discussions and Future Work

Better understanding of $\mathcal{X}$-transformation. While $\mathcal{X}$-Conv is designed aiming at addressing the problems in applying convolution directly on point cloud, and it is demonstrated to outperform state-of-the-art methods, the rigorous theoretical rationale behind this operation, especially when it is composited into a deep neural network, remains barely understood. We proposed probably the most straight forward way of learning the $\mathcal{X}$-transformations as general matrices simply with MLP. While the general matrices are capable of realizing weighting and permutation, it is not clear whether it is the minimalist for our goal. Actually, we found that PointCNN can exhibit strong over-fitting on some small datasets, such as ModelNet40 and TU-Berlin sketches, even though the parameter number in PointCNN is rather small compared with alternative methods. It seems that while powerful representation capability is brought by $\mathcal{X}$-transformations, extra degrees of freedom smuggled in. It is extremely intriguing to study whether there are some structural constraints in $\mathcal{X}$, rather than being a general matrix, and to propose specific methods for learning it.

From Line 1 of Algorithm 1 ($\mathbf{P}' \leftarrow \mathbf{P} - p$), clearly, $\mathcal{X}$-Conv is independent of translation. However, it is not clear whether $\mathcal{X}$-Conv is capable of achieving equivariance on the rigid rotation of the local points. Or if it is capable, whether the learnt degree of equivariance on the rigid rotation is up to different tasks, i.e., the

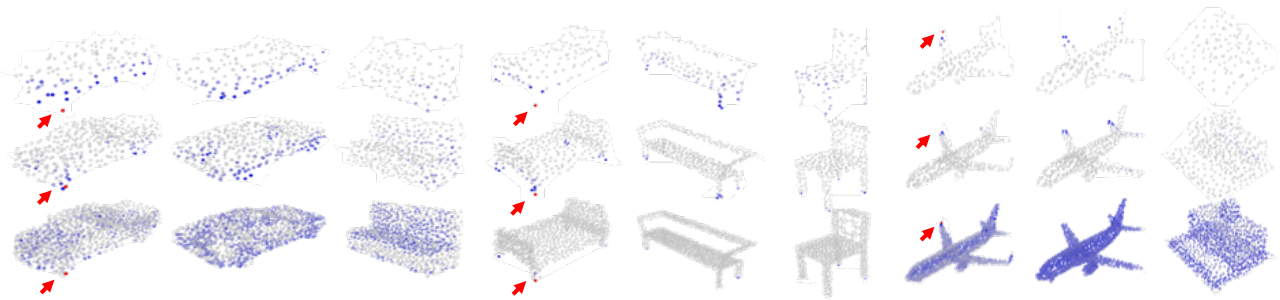

Figure 7: Distance visualization of features at different hierarchies.

network can learn to ignore local rigid rotations if they are not useful to solve the task, and capture them if they are necessary.

PointCNN for shape analysis. We have demonstrated the feature learning effectiveness of PointCNN for classification and segmentation tasks. We suspect the PointCNN learnt features might outperform hand-crafted features in various shape analysis tasks, such as shape key point matching, registration, and retrieval, as is the case with CNN features for various image tasks. Note that in PointNet and PointNet++, the learnt features are features of regions, whereas in PointCNN, the learnt features are "projected", or "aggregated" at the specific representative points, thus PointCNN might outperform at tasks where the locations matter more.

Fully convolutional PointCNN. The way we process point cloud in different scales, such as S3DIS and ScanNet, is sub-optimal. Following the fully convolutional idea in processing images in different sizes, PointCNN should be applied in a fully convolutional way for extracting features from point cloud in different scales. The block slicing we currently used is a brutal approximation of the fully convolutional approach. We leave the extension of PointCNN into supporting fully convolutional $\mathcal{X}$-Conv, which would require highly efficient point cloud indexing and memory management, as future work.

PointCNN or CNN? Since $\mathcal{X}$-Conv is a generalization of Conv, ideally, PointCNN is supposed to perform as good as, if not better than, CNN, if the underlying data is the same but only represented differently. To verify this, we evaluate PointCNN on the point cloud representation of MNIST and CIFAR10. We show the our PointCNN classification results, in comparison with PointNet++, as well as image CNNs in Table 5. For MNIST data, PointCNN achieved the best performance out of the comparison methods, which indicates that PointCNN performs quite well in learning the digits' shape information. For CIFAR10 data, where there is mostly no "shape" information, PointCNN has to learn mostly from the spatially-local correlation in the RGB features, and it performed reasonably well on this task, though there is a large gap between PointCNN and the mainstream image CNNs. Note that PointNet++ performs no better than random choice on CIFAR10. We suspect the reason is that, in PointNet++, the RGB features, after being process by the max-pooling, become in-discriminative. Together with the lack of "shape" information, PointNet++ failed completely on this task.

Method	MNIST (%)	CIFAR10 (%)
LeNet [LeCun et al. 1998]	99.20	84.07
Network in Network [Lin et al. 2014]	99.53	**91.20**
PointNet++ [Qi et al. 2017a]	99.49	10.0
PointCNN	**99.54**	76.69

Table 5: Classification accuracies on MNIST and CIFAR10.

From the CIFAR10 experiment, we can conclude that CNNs are still the choice over PointCNN for general images for now, before the ideal PointCNN, if it exists, is developed. Note that this is not contradictory to the fact that PointCNN outperforms MVCNN [Su et al. 2015] on ModelNet40. MVCNN project 3D meshes into multi-view images, and then apply image CNNs, while PointCNN directly operates on the points sampled from the meshes, but not the image pixel points. It seems that the sparser the data is, the more prominent the advantage of PointCNN can be observed.

It is interesting to study the principle criteria for making the choice of CNN+dense representation vs. PointCNN+point cloud representation. Meanwhile, some seemly dense data might be represented sparsely. For example, videos are commonly represented as dense 3D volume, which might be an overkill since usually only a small portion of pixels are non-stationary in the frames. PointCNN+sparse, but irregular, represented videos seems to be an interesting direction.

The combination of PointCNN and CNNs. Due to the rapid advancement of 3D sensor, more and more data will be captured with point cloud and images hand in hand. In such case, processing point cloud and images with PointCNN and CNNs independently, and then merge them for the final inference might not be the optimal solution. It is interesting to study how to combine PointCNN and CNNs to jointly process paired point cloud and images, probably at early convolution stages.

5 CONCLUSION

We proposed PointCNN, which is a generalization of CNN into leveraging spatially-local correlation from data represented as point cloud. We demonstrated its strong performance on multiple challenging benchmark datasets and tasks. The core of PointCNN is

the $\mathcal{X}$-Conv operator that weights and permutes input points and features before they are process by a typical convolution.

As point cloud data is becoming more accessible, we envision it is of great importance to develop methods that can effectively leverage spatially-local correlation from such data, and our method is just a starting point in the important undertaking. We open source our code at https://github.com/yangyanli/PointCNN for encouraging future developments.

ACKNOWLEDGMENTS

Yangyan would like to thank Leonidas Guibas from Stanford University and Mike Haley from Autodesk Research for insightful discussions.

The work is supported in part by National Science Foundation of China General Program grant No. 61772317, the National Basic Research grant (973) No. 2015CB352501, and National Key Research and Development Program of China grant No. 2017YFB1002603.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.

Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 2016. 3d semantic parsing of large-scale indoor spaces. In *CVPR*. 1534–1543.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

François Chollet. 2016. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv preprint arXiv:1610.02357* (2016).

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*.

Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. 2017. DeepPermNet: Visual Permutation Learning. In *CVPR*.

Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. 2017. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *CVPR*.

Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. 2016. Exploiting Cyclic Symmetry in Convolutional Neural Networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 1889–1898. http://dl.acm.org/citation.cfm?id=3045390.3045590

Mathias Eitz, James Hays, and Marc Alexa. 2012. How Do Humans Sketch Objects? *ToG* 31, 4 (2012), 44:1–44:10.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep Sparse Rectifier Neural Networks. In *International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Geoffrey Gordon, David Dunson, and Miroslav DudÃąk (Eds.), Vol. 15. PMLR, Fort Lauderdale, FL, USA, 315–323.

Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 2017. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. *arXiv preprint arXiv:1711.10275* (2017).

Benjamin Graham and Laurens van der Maaten. 2017. Submanifold Sparse Convolutional Networks. *arXiv preprint arXiv:1706.01307* (2017).

David Ha and Douglas Eck. 2017. A Neural Representation of Sketch Drawings. *arXiv preprint arXiv:1704.03477* (2017).

Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. 2011. Transforming auto-encoders. In *International Conference on Artificial Neural Networks*. Springer, 44–51.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*. 448–456.

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

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. In *ICLR*.

Roman Klokov and Victor Lempitsky. 2017. Escape from Cells: Deep Kd-Networks for The Recognition of 3D Point Cloud Models. In *ICCV*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.

Loïc Landrieu and Martin Simonovsky. 2017. Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs. *CoRR* abs/1711.09869 (2017). arXiv:1711.09869 http://arxiv.org/abs/1711.09869

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

Yangyan Li, Sören Pirk, Hao Su, Charles R Qi, and Leonidas J Guibas. 2016. FPNN: Field probing neural networks for 3d data. In *NIPS*. 307–315.

Min Lin, Qiang Chen, and Shuicheng Yan. 2014. Network in network. In *ICLR*.

Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G. Kim, and Yaron Lipman. 2017. Convolutional Neural Networks on Surfaces via Seamless Toric Covers. *ACM Trans. Graph.* 36, 4, Article 71 (July 2017), 10 pages. https://doi.org/10.1145/3072959.3073616

Federico Monti, Davide Boscaini, and Jonathan Masci. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*.

Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. 2015. Learning Deconvolution Network for Semantic Segmentation. In *ICCV (ICCV '15)*. IEEE Computer Society, Washington, DC, USA, 1520–1528. https://doi.org/10.1109/ICCV.2015.178

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017a. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*. 77–85. https://doi.org/10.1109/CVPR.2017.16

Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. 2016. Volumetric and multi-view CNNs for object classification on 3d data. In *CVPR*. 5648–5656.

Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017b. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NIPS*. 5105–5114.

Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. 2016. Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500* (2016).

Gernot Riegler, Ali Osman Ulusoys, and Andreas Geiger. 2017. Octnet: Learning deep 3d representations at high resolutions. In *CVPR*.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *MICCAI*, Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi (Eds.). Springer International Publishing, Cham, 234–241.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. MIT Press, Cambridge, MA, USA, Chapter Learning Internal Representations by Error Propagation, 318–362. http://dl.acm.org/citation.cfm?id=104279.104293

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic routing between capsules. In *NIPS*. 3859–3869.

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. 2015. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*. 945–953.

Lyne P. Tchapmi, Christopher B. Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. 2017. SEGCloud: Semantic Segmentation of 3D Point Clouds. In *3DV*.

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. 2017. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM Trans. Graph.* 36, 4, Article 72 (July 2017), 11 pages. https://doi.org/10.1145/3072959.3073608

Shihao Wu, Hui Huang, Minglun Gong, Matthias Zwicker, and Daniel Cohen-Or. 2015a. Deep Points Consolidation. *ToG* 34, 6, Article 176 (Oct. 2015), 13 pages. https://doi.org/10.1145/2816795.2818073

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

Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. 2016. A Scalable Active Framework for Region Annotation in 3D Shape Collections. *ToG* 35, 6, Article 210 (Nov. 2016), 12 pages. https://doi.org/10.1145/2980179.2980238

Li Yi, Hao Su, Xingwen Guo, and Leonidas Guibas. 2017a. SyncSpecCNN: Synchronized spectral CNN for 3d shape segmentation. In *CVPR*. 6584–6592. https://doi.org/10.1109/CVPR.2017.697

Li Yi, Hao Su, Lin Shao, Manolis Savva, Haibin Huang, Yang Zhou, Benjamin Graham, Martin Engelcke, Roman Klokov, Victor Lempitsky, et al. 2017b. Large-Scale 3D Shape Reconstruction and Segmentation from ShapeNet Core55. *arXiv preprint arXiv:1710.06104* (2017).

Qian Yu, Yongxin Yang, Feng Liu, Yi-Zhe Song, Tao Xiang, and Timothy M. Hospedales. 2017. Sketch-a-Net: A Deep Neural Network That Beats Humans. *IJCV* 122, 3 (May 2017), 411–425. https://doi.org/10.1007/s11263-016-0932-3

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. 2017. Deep Sets. In *NIPS*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.).

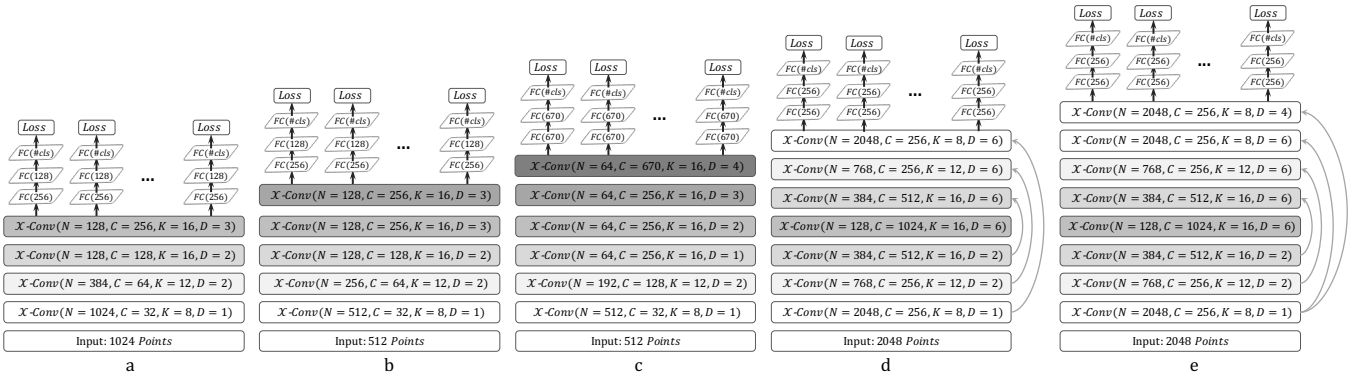

Figure 8: PointCNN model zoo, where (a) is used for ModelNet40 and ScanNet classification, (b) is used for TU-Berlin sketch classification, (c) is used for Quick Draw sketch classification, (d) is used for ScanNet and S3DIS segmentation, and (e) is used for ShapeNet Parts segmentation.

3394–3404.

A APPENDIX

A.1 PointCNN Model Zoo

In Figure 8, we list the PointCNNs used for classification and segmentation tasks on multiple benchmark datasets. PointCNNs are easy to implement, setup, and tune. Larger C are used for layers with more abstract/semantic information, such as the top layers in classification networks, and middle layers in "Conv-DeConv" segmentation networks. To relax the memory demand, smaller Ks are used at layers with large number of representative points, such as bottom layers of classification networks, and top and bottom layers of segmentation networks. Deeper PointCNN with larger receptive field in the last $\mathcal{X}$-Conv layer are used for larger or harder datasets. The skip-links, together with the dilation parameter D, make it easy to fuse information from different scales (receptive fields), as illustrated in (d) and (e), which is essential for segmentation tasks.