PointCNN: Convolution On \( \mathcal{X} \)-Transformed Points

Yangyan Li†* Rui Bu† Mingchao Sun‡ Wei Wu† Xinhan Di‡ Baoquan Chen§
†Shandong University ‡Huawei Inc. §Peking University

Abstract

We present a simple and general framework for feature learning from point clouds. 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 clouds are irregular and unordered, thus directly convolving kernels against features associated with the points, will result in desertion of shape information and variance to point ordering. To address these problems, we propose to learn an \( \mathcal{X} \)-transformation from the input points, to simultaneously promote two causes. The first is the weighting of the input features associated with the points, and the second is the permutation of the points into a latent and potentially canonical order. Element-wise product and sum operations of the typical convolution operator are subsequently applied on the \( \mathcal{X} \)-transformed features. The proposed method is a generalization of typical CNNs to feature learning from point clouds, 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.

1 Introduction

Spatially-local correlation is a ubiquitous property of various types of data that is independent of the data representation. For data that is represented in regular domains, such as images, the convolution operator has been shown to be effective in exploiting that correlation as the key contributor to the success of CNNs on a variety of tasks [25]. However, for data represented in point cloud form, which is irregular and unordered, the convolution operator is ill-suited for leveraging spatially-local correlation in the data.

![Convolution input from regular grids (i) and point clouds (ii-iv). In (i), each grid cell is associated with a feature. In (ii-iv), the points are sampled from local neighborhoods, in analogy to local patches in (i), and each point is associated with a feature, an order index, and coordinates.](image)

\[ f_{ii} = \text{Conv}(K, [f_a, f_b, f_c, f_d]^T), \]
\[ f_{iii} = \text{Conv}(K, [f_a, f_b, f_c, f_d]^T), \]
\[ f_{iv} = \text{Conv}(K, [f_c, f_a, f_b, f_d]^T). \]

(1a)

\[ f_{ii} = \text{Conv}(K, \mathcal{X}_{ii} \times [f_a, f_b, f_c, f_d]^T), \]
\[ f_{iii} = \text{Conv}(K, \mathcal{X}_{iii} \times [f_a, f_b, f_c, f_d]^T), \]
\[ f_{iv} = \text{Conv}(K, \mathcal{X}_{iv} \times [f_c, f_a, f_b, f_d]^T). \]

(1b)

We illustrate the problems and challenges of applying convolutions on point clouds in Figure 1. Suppose the unordered set of the \( C \)-dimensional input features is the same \( \mathcal{F} = \{f_a, f_b, f_c, f_d\} \)

*Part of the work was done during Yangyan’s Autodesk Research 2017 summer visit.
in all the cases ((i) – (iv)), and we have one kernel $K = [k_\alpha, k_\beta, k_\gamma, k_\delta]^T$ of shape $4 \times C$. In (i), by following the canonical order given by the regular grid structure, the features in the local $2 \times 2$ patch can be cast into $[f_{a}, f_{b}, f_{c}, f_{d}]^T$ of shape $4 \times C$, for convolving with $K$, yielding $f_i = \text{Conv}(K, [f_{a}, f_{b}, f_{c}, f_{d}])$, where $\text{Conv}(\cdot, \cdot)$ is simply an element-wise product followed by a sum. In (ii), (iii), and (iv), the points are sampled from local neighborhoods, and thus their ordering may be arbitrary. By following orders as illustrated in the figure, the input feature set $\mathcal{F}$ can be cast into $[f_{a}, f_{b}, f_{c}, f_{d}]^T$ in (ii) and (iii), and $[f_{a}, f_{c}, f_{d}, f_{b}]^T$ in (iv). Based on this, if the convolution operator is directly applied, the output features for the three cases could be computed as depicted in Eq. 1.

Note that $f_{i} \equiv f_{iii}$ holds for all cases, while $f_{i} \equiv f_{iv}$ holds for most cases. This example illustrates that a direct convolution results in deserting shape information (i.e., $f_{i} \equiv f_{iii}$), while retaining variance to the ordering (i.e., $f_{iii} \neq f_{iv}$).

In this paper, we propose to learn a $K \times K \mathcal{X}$-transformation for the coordinates of $K$ input points $(p_1, p_2, \ldots, p_K)$, with a multilayer perceptron \cite{8}, i.e., $\mathcal{X} = \text{MLP}(p_1, p_2, \ldots, p_K)$. Our aim is to use it to simultaneously weight and permute the input features, and subsequently apply a typical convolution on the transformed features. We refer to this process as $\mathcal{X}$-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 formulated as in Eq. 15 where the $\mathcal{X}$’s are $4 \times 4$ matrices, as $K = 4$ in this figure. Note that since $\mathcal{X}_{i}ii$ and $\mathcal{X}_{iii}$ are learned from points of different shapes, they can differ so as to weight the input features accordingly, and achieve $f_{i} \neq f_{iii}$. For $\mathcal{X}_{i}ii$ and $\mathcal{X}_{iv}$, if they are learned to satisfy $\mathcal{X}_{i}ii = \mathcal{X}_{iv} \times \Pi$, where $\Pi$ is the permutation matrix for permuting $(c, a, b, d)$ into $(a, b, c, d)$, then $f_{i} \equiv f_{iv}$ 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 invariant to ordering. In practice, we find that the learned $\mathcal{X}$-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 convolutions on point clouds, and on par or better than state-of-the-art neural networks designed for point cloud input data, such as PointNet++ \cite{53}.

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

2 Related Work

Feature Learning from Regular Domains. CNNs have been very successful for leveraging spatially-local correlation in images — pixels in 2D regular grids \cite{26}. There has been work in extending CNNs to higher dimensional regular domains, such as 3D voxels \cite{32}. However, as both the input and convolution kernels are of higher dimensions, the amount of both computation and memory inflates dramatically. Octree \cite{57, 58}, Kd-Tree \cite{22} and Hash \cite{41} based approaches have been proposed to save computation by skipping convolution in empty space. The activations are kept sparse in \cite{15} to retain sparsity in convolved layers. \cite{17} and \cite{14} partition point cloud into grids and represent each grid with grid mean points and Fisher vectors respectively for convolving with 3D kernels. In these approaches, the kernels themselves are still dense and of high dimension. Sparse kernels are proposed in \cite{28}, 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 \cite{39} and Deep Sets \cite{58} proposed to achieve input order invariance by the use of symmetric function over inputs. PointNet++ \cite{53} and SO-Net \cite{27} applies PointNet hierarchically for better capturing of local structures. Kernel correlation and graph pooling are proposed for improving PointNet-like methods in \cite{52}. RNN is used in \cite{18} for processing features aggregated by pooling from ordered point cloud slices. \cite{50} proposed to leverage neighborhood structures in both point and feature spaces. While these symmetric pooling based approaches, as well as those in \cite{11, 52, 56}, have guarantee in achieving order invariance, they come with a price of throwing away information.

2

\footnote{Actually, this is a special instance of convolution — a convolution that is applied in one spatial location. For simplicity, we call it convolution as well.}

\textbf{2}
31 propose to first “interpolate” or “project” features into predefined regular domains, where typical CNNs can be applied. In contrast, the regular domain is latent in our method. CNN kernels are represented as parametric functions of neighborhood point positions to generalize CNNs into point cloud in 3145524. The kernels associated with each point are parametrized individually in these methods, while the $\mathcal{X}$-transformations in our method are learned from each neighborhoods, thus could potentially by more adaptive to local structures.

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 315550. We refer the interested reader to 3 for a comprehensive survey of work along these directions. Spectral graph convolution on a local graph is used for processing point cloud in 46.

**Invariance vs. Equivariance.** A line of pioneering work aiming at achieving equivariance have been proposed to address the information loss problem of pooling in achieving invariance 1650. 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 20, in the sense that both of them provided a mechanism to “transform” input into latent canonical forms for being further processed, with no explicit loss or constraint in enforcing the canonicalization. In practice, it turns out that the networks find their ways to leverage the mechanism for learning better. 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 3, where a permutation matrix is the desired output, and is approximated by a doubly stochastic matrix.

3 **PointCNN**

The hierarchical application of convolutions is essential for learning hierarchical representations via CNNs. PointCNN shares the same design and generalizes it to point clouds. First, we introduce hierarchical convolutions in PointCNN, in analogy to that of image CNNs, then, we explain the core $\mathcal{X}$-Conv operator in detail, and finally, present PointCNN architectures geared toward various tasks.

### 3.1 Hierarchical Convolution

![Figure 2: Hierarchical convolution on regular grids (upper) and point clouds (lower). In regular grids, convolution is recursively applied on local grid patches, which often reduces the grid resolution $(4 \times 4 \rightarrow 3 \times 3 \rightarrow 2 \times 2)$, while increasing the channel number (visualized by dot thickness). Similarly, in point clouds, $\mathcal{X}$-Conv is recursively applied to “project”, or “aggregate”, information from neighborhoods into fewer representative points $(9 \rightarrow 5 \rightarrow 2)$, but each with richer information.](image)

Before we introduce the hierarchical convolution in PointCNN, we briefly go through its well known version for regular grids, as illustrated in Figure 2 upper. The input to grid-based CNNs is a feature map $F_1$ of 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 $K$ of shape $K \times K \times C_1 \times C_2$ against local patches of shape $K \times K \times C_1$ from $F_1$, yields another feature map $F_2$ of 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 $F_1$, $F_2$ is often 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 with decreasing spatial resolution $(4 \times 4 \rightarrow 3 \times 3 \rightarrow 2 \times 2$ in Figure 2 upper), but deeper channels (visualized by increasingly thicker dots in Figure 2 upper).

The input to PointCNN is $F_1 = \{(p_{1,i}, f_{1,i}) : i = 1,2,...,N_1\}$, i.e., a set of points $\{p_{1,i} : p_{1,i} \in \mathbb{R}^{D_{in}}\}$, each associated with a feature $\{f_{1,i} : f_{1,i} \in \mathbb{R}^{C_1}\}$. Following the hierarchical construction of grid-based CNNs, we would like to apply $\mathcal{X}$-Conv on $F_1$ to obtain a higher level representation $F_2 = \{(p_{2,i}, f_{2,i}) : f_{2,i} \in \mathbb{R}^{C_2}, i = 1,2,...,N_2\}$, where $\{p_{2,i}\}$ is a set of representative points of
We detail the To leverage spatially-local correlation, similar to convolution in grid-based CNNs, the representative points are then individually lifted and combined with the associated features.

\{p_{1,i}\} and \(F_2\) is of a smaller spatial resolution and deeper feature channels than \(F_1\), i.e., \(N_2 < N_1\), and \(C_2 > C_1\). When the \(X\)-Conv process of turning \(F_1\) into \(F_2\) is recursively applied, the input points with features are “projected”, or “aggregated”, into fewer points (9 \(\rightarrow\) 5 \(\rightarrow\) 2 in Figure 2, lower), but each with increasingly richer features (visualized by increasingly thicker dots in Figure 2, lower).

The representative points \(\{p_{2,i}\}\) should be the points that are beneficial for the information “projection” or “aggregation”. In our implementation, they are generated by random down-sampling of \(\{p_{1,i}\}\) in classification tasks, and farthest point sampling in segmentation tasks, since segmentation tasks are more demanding on a uniform point distribution. We suspect special point selections which have shown promising performance in geometry processing, such as Deep Points [51], could fit in here as well. We leave the exploration of better representative point generation methods for future work.

### 3.2 \(X\)-Conv Operator

\(X\)-Conv is the core operator for turning \(F_1\) into \(F_2\). In this section, we first introduce the input, output and procedure of the operator, and then explain the rationale behind the procedure.

**ALGORITHM 1: \(X\)-Conv Operator**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(K, p, P, F)</td>
<td>(F_p)</td>
</tr>
</tbody>
</table>

1. \(P' \leftarrow P - p\)  \(\triangleright\) Features “projected”, or “aggregated”, into representative point \(p\)
2. \(F_\delta \leftarrow MLP_\delta(P')\)  \(\triangleright\) Move \(P\) to local coordinate system of \(p\)
3. \(F_* \leftarrow [F_\delta, F]\)  \(\triangleright\) Individually lift each point into \(C_\delta\) dimensional space
4. \(X \leftarrow MLP(X')\)  \(\triangleright\) Concatenate \(F_\delta\) and \(F_*\), a \(K \times (C_\delta + C_1)\) matrix
5. \(F_X \leftarrow X \cdot F_*\)  \(\triangleright\) Learn the \(K \times K X\)-transformation matrix
6. \(F_p \leftarrow Conv(K, F_X)\)  \(\triangleright\) Finally, typical convolution between \(K\) and \(F_X\)

To leverage spatially-local correlation, similar to convolution in grid-based CNNs, \(X\)-Conv operates in local regions. Since the output features are supposed to be associated with the representative points \(\{p_{2,i}\}\), \(X\)-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 \(p\), the features with \(p\) as \(f\) and its \(K\) neighbors in \(\{p_{1,i}\}\) as \(N\), thus the \(X\)-Conv input for this specific \(p\) is \(S = \{(p_i, f_i) : p_i \in N\}\). Note that \(S\) is an unordered set. Without loss of generality, \(S\) can be cast into a \(K \times Dim\) matrix \(P = (p_1, p_2, ..., p_K)^T\), and a \(K \times C_1\) matrix \(F = (f_1, f_2, ..., f_K)^T\), and \(K\) denotes the trainable convolution kernels. With these inputs, we would like to compute the features \(F_p\), which are the “projection”, or “aggregation”, of the input features into the representative point \(p\).

We detail the \(X\)-Conv operator in Algorithm 1 and summarize it concisely as:

\[
F_p = X^{-\text{Conv}}(K, p, P, F) = Conv(K, MLP(P - p) \times [MLP_\delta(P - p), F]),
\]

(2)

where \(MLP_\delta(\cdot)\) is a multilayer perceptron applied individually on each point, as in PointNet [33]. Note that all the operations involved in building \(X\)-Conv, i.e., \(Conv(\cdot, \cdot), MLP(\cdot),\) matrix multiplication \((\cdot) \times (\cdot),\) and \(MLP_\delta(\cdot),\) are differentiable. Accordingly, \(X\)-Conv is differentiable, and can be plugged into a neural network for training by back propagation.

Lines 4-6 in Algorithm 1 are the core \(X\)-transformation as described in Eq. [15] in Section 1. Here, we explain the rationale behind lines 1-3 of Algorithm 1 in detail. \(X\)-Conv is designed to work on local point regions, and the output should not be dependent on the absolute position of \(p\) and its neighboring points, but on their relative positions. To that end, we position local coordinate
systems at the representative points (line 1 of Algorithm 1, Figure 3b). It is the local coordinates of neighboring points, together with their associated features, that define the output features. However, the local coordinates are of a different dimensionality and representation than the associated features. To address this issue, we first lift the coordinates into a higher dimensional and more abstract representation (line 2 of Algorithm 1), and then combine it with the associated features (line 3 of Algorithm 1) for further processing (Figure 3c).

To address this issue, we first lift the coordinates into a higher dimensional and more abstract representation, vs. predictions. Note that, in this way, we can train the top neighboring points, where always taking the ratio of training samples for the top point number in the previous layer. With this definition, the final point “sees” all the points from the previous layer, thus has a receptive field of \( \frac{K}{N} \). PointCNN can thus handle point clouds with or without additional features in a robust uniform fashion.

For more details about the \( \mathcal{X} \)-Conv operator, including the actual definition of \( MLP_\delta(\cdot) \), \( MLP(\cdot) \) and \( \text{Conv}(\cdot, \cdot) \), please refer to Supplementary Material Section I.

### 3.3 PointCNN Architectures

From Figure 2, we can see that the Conv layers in grid-based CNNs and \( \mathcal{X} \)-Conv layers in PointCNN only differ in two aspects: the way the local regions are extracted (\( K \times K \) patches vs. \( K \) neighboring points around representative points) and the way the information from local regions is learned (Conv vs. \( \mathcal{X} \)-Conv). Otherwise, the process of assembling a deep network with \( \mathcal{X} \)-Conv layers highly resembles that of grid-based CNNs.

Figure 4 depicts a simple PointCNN with two \( \mathcal{X} \)-Conv layers that gradually transform the input points (with or without features) into fewer representation points, but each with richer features. After the second \( \mathcal{X} \)-Conv layer, there is only one representative point left, and it aggregates 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 \( \frac{K}{N} \), where \( K \) is the neighboring point number, and \( N \) is the point number in the previous layer. With this definition, the final point “sees” all the points from the previous layer, thus has a receptive field of 1.0 — it has a global view of the entire shape, and its features are informative for semantic understanding of the shape. We can add fully connected 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 drops rapidly (Figure 4b), making it inefficient to train them thoroughly. To address this problem, we propose PointCNN with denser connections (Figure 4b), where more representative points are kept in the \( \mathcal{X} \)-Conv layers. However, we aim to maintain the depth of the network, while keeping the receptive field growth rate, such that the deeper representative points “see” increasingly larger portions of the entire shape. We achieve this goal by employing the dilated convolution idea from grid-based CNNs in PointCNN. Instead of always taking the \( K \) neighboring points as input, we 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 \( \frac{K}{N} \) to \( \frac{(K \times D)}{N} \), without increasing actual neighboring point count or kernel size.

In the second \( \mathcal{X} \)-Conv layer of PointCNN in Figure 4b, 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 to PointCNN in Figure 4. In test time, the output from the multiple representative points is averaged right before the softmax to stabilize the prediction. This design is similar to that of Network in Network [29]. The denser version of PointCNN (Figure 4) is the one we used for classification tasks.

For segmentation tasks, high resolution point-wise output is required, and this can be realized by building PointCNN following Conv-DeConv architecture, where the DeConv part is responsible for propagating global information into high resolution predictions (see Figure 4c). Note that both the “Conv” and “DeConv” in the PointCNN segmentation network are the same $\mathcal{X}$-Conv operator. The only differences between the “Conv” and “DeConv” layers is that the latter has more points but less feature channels in its output vs. its input, and its higher resolution points are forwarded from earlier “Conv” layers, following the design of U-Net [38].

Dropout is applied before the last fully connected layer for reducing over-fitting. We also employed the “subvolume supervision” idea from [34], to further address the over-fitting problem. In the last $\mathcal{X}$-Conv layers, the receptive field is set to be less than 1, such that only partial information is “seen” by the representative points. The network is pushed to learn harder from the partial information during training, and performs better at test time. In this case, the global coordinates of the representative points matter, thus they are lifted into feature space by the representative points. The network is pushed to learn harder from the partial information during training.

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

4 Experiments

We conducted an extensive evaluation of PointCNN for shape classification on six datasets (ModelNet40 [52], ScanNet [49], TU-Berlin [111], Quick Draw [15], MNIST, CIFAR10), and segmentation task on three datasets (ShapeNet Parts [54], S3DIS [9], and ScanNet [29]). The details of the datasets and how we convert and feed data into PointCNN, are described in Supp. Material Section 3 and the PointCNN architectures for the tasks on these datasets can be found in Supp. Material Section 5.

4.1 Classification and Segmentation Results

We summarize our 3D point cloud classification results on ModelNet40 and ScanNet in Table 1 and compare to several neural network methods designed for point clouds. Note that a large portion of the 3D models from ModelNet40 are pre-aligned to the common up direction and horizontal facing direction. If a random horizontal rotation is not applied on either the training or testing sets, then the relatively consistent horizontal facing direction is leveraged, and the metrics based on this setting is not directly comparable to those with the random horizontal rotation. For this reason, we ran PointCNN and reported its performance in both settings. Note that PointCNN achieved top performance on both ModelNet40 and ScanNet.

Table 1: Comparisons of mean per-class accuracy (mA) and overall accuracy (OA) (%) on ModelNet40 [52] and ScanNet [29]. The reported performances are based on 1024 input points, unless otherwise noted by P# (# input points) or PN# (# input points with normals).
We evaluate PointCNN on the segmentation of ShapeNet Parts, S3DIS, and ScanNet datasets, and summarize the results in Table 2. More detailed segmentation result comparisons can be found in Supplementary Material Section 4. We note that PointCNN outperforms all the compared methods, including SSCN [12], SPGraph [24] and SGPN [49], which are specialized segmentation networks with state-of-the-art performance. Note that the part averaged IoU metric for ShapeNet Parts is the one used in [55]. Compared with mean IoU, the part averaged IoU puts more emphasis on the correct prediction of small parts.

Sketches are 1D curves in 2D space, thus can be more effectively represented with point clouds, rather than with 2D images. We evaluate PointCNN on TU-Berlin and Quick Draw sketches, and present results in Table 3, where we compare its performance with the competitive PointNet++, as well as new CNN based methods. PointCNN outperforms PointNet++ on both datasets, with a more prominent advantage on Quick Draw (25M data samples), which is significantly larger than TU-Berlin (0.02M data samples). On the TU-Berlin dataset, while the performance of PointCNN is slightly better than the generic image CNN AlexNet [23], there is still a gap with the specialized Sketch-a-Net [57]. It is interesting to study whether architectural elements from Sketch-a-Net can be adopted and integrated into PointCNN to improve its performance on the sketch datasets.

Since  \( \mathcal{X} \)-Conv is a generalization of Conv, ideally, PointCNN should perform on par with CNNs, 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, and show results in Table 4. For MNIST data, PointCNN achieved comparable performance with other methods, indicating its effective learning of 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. From this experiment, we can conclude that CNNs are still the better choice for general images.

<table>
<thead>
<tr>
<th>Method</th>
<th>TU-Berlin</th>
<th>Quick Draw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sketch-a-Net</td>
<td>77.95</td>
<td></td>
</tr>
<tr>
<td>AlexNet</td>
<td>68.60</td>
<td></td>
</tr>
<tr>
<td>PointNet++</td>
<td>66.53</td>
<td>51.58</td>
</tr>
<tr>
<td>PointCNN</td>
<td>70.57</td>
<td><strong>59.13</strong></td>
</tr>
</tbody>
</table>

Table 3: Sketch classification results.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST</th>
<th>CIFAR10</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet</td>
<td>99.20</td>
<td>84.07</td>
</tr>
<tr>
<td>Network in Network</td>
<td>99.53</td>
<td><strong>91.20</strong></td>
</tr>
<tr>
<td>PointNet+++</td>
<td>99.49</td>
<td><strong>10.01</strong></td>
</tr>
<tr>
<td>PointCNN</td>
<td><strong>99.54</strong></td>
<td>80.22</td>
</tr>
</tbody>
</table>

Table 4: Image classification results.

4.2 Ablation Experiments and Visualizations

Ablation test of the core  \( \mathcal{X} \)-Conv operator. To verify the effectiveness of the  \( \mathcal{X} \)-transformation, we propose PointCNN without it as a baseline, where lines 4-6 of Algorithm 1 are replaced by  \( F_p \leftarrow \text{Conv}(K, F) \). Compared with PointCNN, the baseline has less trainable parameters, and is more “shallow” due to the removal of  \( \text{MLP}(\cdot) \) in line 4 of Algorithm 1. For a fair comparison, we further propose PointCNN w/o  \( \mathcal{X} \)-W/D, which is wider/deeper, and has approximately the same amount of parameters as PointCNN. The model depth of PointCNN w/o  \( \mathcal{X} \) (deeper) also compensates for  \( \mathcal{X} \)-W/D.

\(^3\)PointNet++ performs no better than random choice on CIFAR10. We suspect the reason is that, in PointNet++, the RGB features become in-discriminative after being processed by the max-pooling. Together with the lack of “shape” information, PointNet++ fails completely on this task.
the decrease in depth caused by the removal of $MLP(\cdot)$ from PointCNN. The comparison results are summarized in Table 5. Clearly, PointCNN outperforms the proposed variants by a significant margin, and the gap between PointCNN and PointCNN w/o $\mathcal{X}$ is not due to model parameter number, or model depth. With these comparisons, we conclude that $\mathcal{X}$-Conv is the key to the performance of PointCNN.

**Visualization of $\mathcal{X}$-Conv features.** 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 $C = C_3 + C_1$. For the same representative point, if its neighboring points in different orders are fed 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 $\mathcal{F}_o$. Clearly, $\mathcal{F}_*$ can be quite scattering in the $\mathbb{R}^{K \times C}$ space, since differences in input point order will result in a different $\mathbf{F}_*$. On the other hand, if the learned $\mathcal{X}$ 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 picked representative points from the ModelNet40 dataset in Figure 5, each with one color, and consistent in the sub-figures. Note that $\mathcal{F}_o$ is quite “blended”, which indicates that the features from different representative points are not discriminative against each other (Figure 5a). While $\mathcal{F}_*$ is better than $\mathcal{F}_o$, it is still “fuzzy” (Figure 5b). In Figure 5c, $\mathcal{F}_\mathcal{X}$ are “concentrated” by $\mathcal{X}$, and the features of each representative point become highly discriminative. To give an quantitative reference of the “concentration” effect, we firstly compute the feature centers of different representative points, then classify all the feature points to the representative points they belong to, based on nearest search to the centers. The classification accuracies are 76.83%, 89.29% and 94.72% for $\mathcal{F}_o$, $\mathcal{F}_*$ and $\mathcal{F}_\mathcal{X}$, respectively. With the qualitative visualization and quantitative investigation, we conclude that though the “concentration” is far from reaching a point, the improvement is significant, and it explains the performance of PointCNN in feature learning.

**Table 5: Ablation tests on ModelNet40.**

<table>
<thead>
<tr>
<th>Core Layers</th>
<th>$\mathcal{X}$-Conv</th>
<th>$\mathcal{X}$-W</th>
<th>$\mathcal{X}$-D</th>
</tr>
</thead>
<tbody>
<tr>
<td># Parameter</td>
<td>0.6M</td>
<td>0.54M</td>
<td>0.63M</td>
</tr>
<tr>
<td>Accuracy (%)</td>
<td>92.2</td>
<td>90.7</td>
<td>90.8</td>
</tr>
</tbody>
</table>

**Table 6: Parameter number, FLOPs and running time comparisons.**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Parameters</th>
<th>FLOPs</th>
<th>Time Training</th>
<th>Time Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet</td>
<td>3.48M</td>
<td>43.82B</td>
<td>0.006s</td>
<td>0.015s</td>
</tr>
<tr>
<td>PointNet++</td>
<td>1.48M</td>
<td>67.94B</td>
<td>0.091s</td>
<td>0.027s</td>
</tr>
<tr>
<td>3DmFV-Net</td>
<td>45.77M</td>
<td>48.57B</td>
<td>0.101s</td>
<td>0.039s</td>
</tr>
<tr>
<td>DGCNN</td>
<td>1.84M</td>
<td>131.37B</td>
<td>0.171s</td>
<td>0.064s</td>
</tr>
<tr>
<td>SpecGCN</td>
<td>2.05M</td>
<td>49.97B</td>
<td>14.64s</td>
<td>0.276s</td>
</tr>
<tr>
<td>PCNN</td>
<td>8.2M</td>
<td>6.49B</td>
<td>11.25s</td>
<td>0.226s</td>
</tr>
<tr>
<td>PointCNN</td>
<td>0.6M</td>
<td>93.03B</td>
<td>0.476s</td>
<td>0.012s</td>
</tr>
</tbody>
</table>

**Optimizer, model size, memory usage and timing.** We implemented PointCNN in tensorflow and use ADAM optimizer with an initial learning rate 0.01 for the training of our models. As shown in Table 6, we summarize our running statistics based with the model for classification with batch size 16, 1024 input points on nVidia Tesla P100 GPU, in comparison with several other methods. PointCNN achieves 0.031/0.012 second per batch for training/inference on this setting. In addition, the model for segmentation with 2048 input points has 4.4M parameters runs on nVidia Tesla P100 with batch size 12 at 0.61/0.25 second per batch for training/inference.
5 Conclusion

We proposed PointCNN, which is a generalization of CNN into leveraging spatially-local correlation from data represented in point cloud. The core of PointCNN is the $\mathcal{X}$-Conv operator that weights and permutes input points and features before they are processed by a typical convolution. While $\mathcal{X}$-Conv is empirically demonstrated to be effective in practice, a rigorous understanding of it, especially when being composited into a deep neural network, is still an open problem for future work. It is also interesting to study how to combine PointCNN and image CNNs to jointly process paired point clouds and images, probably at the early stages. We open source our code at https://github.com/yangyanl/PointCNN to encourage further development.

Acknowledgments

Yangyan would like to thank Leonidas Guibas from Stanford University and Mike Haley from Autodesk Research for insightful discussions, and Noa Fish from Tel Aviv University for proof reading. The work is supported in part by National Key Research and Development Program of China grant No. 2017YFB1002603, the National Basic Research grant (973) No. 2015CB352501, National Science Foundation of China General Program grant No. 61772317, and “Qilu” Young Talent Program of Shandong University.

References


