Computers & Graphics (2021)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Feature interpolation convolution for point cloud analysis

ARTICLE INFO

Article history:
Received April 15, 2021

Keywords: Convolutional Neural Network, Sparse Data, Irregular Domains

ABSTRACT

Point clouds, as the native output of many real-world 3D sensors, can not be trivially consumed by convolutional networks in the same way with the 2D image. This is mainly caused by the irregular organization of points. In this paper, we propose a new convolution operation, named Feature Interpolation Convolution (FI-Conv), which is computationally efficient, invariant to the order of points, and robust to different samples and varying densities. First, point clouds are viewed as discrete samples of continuous space. The feature corresponding to one point is seen as a sampled point of continuous feature function. We desire a set of points, named key points, to describe the important locations of the convolution and relatively stable points of the feature function, such as extreme or inflection points. In our method, the positions of key points are trainable parameters of the networks, i.e., we can optimize the positions of key points. Then, we interpolate point features onto the learned key points. Finally, a standard convolution operation is applied to these estimated features. We use FI-Conv to replace the convolution operations of some cutting-edge networks. Experiments show that FI-Conv effectively improves the performance of these networks and achieves on par or better performance than state-of-the-art methods on multiple challenging benchmark datasets and tasks.

1. Introduction

With the recent proliferation of applications employing 3D depth sensors such as autonomous navigation, robotics, and virtual reality, there is an increasing demand for algorithms to effectively analyze point clouds. Driven by the remarkable success of deep learning on regular grid data (*e.g.*, image) [1, 2], there has been a fresh wave of deep network architectures proposed to tackle the new challenge: unlike image grids, point clouds are sampled sparsely and non-uniformly with continuous spatial coordinates, and they are equivalent up to permutations.

One common approach is transforming point clouds into regular voxels [3, 4, 5] or multi-view images [6, 7, 8], for easy application of classic grid CNN. However, these methods are either unable to scene segmentation or limited by resolution and not suitable for fine-grained analysis. Another difficult yet attractive solution is to learn directly from irregular point clouds. PointNet [9] is a pioneer in this direction. It learns over each point independently and then accumulates features by performing the max-pooling operation. However, PointNet ignores local structures which have been proven to be important for abstracting high-level visual concepts in image CNN. To solve this problem, some works [10, 11, 12, 13, 14] built a hierarchy neural network to learn contextual representation from local to global. They partition the set of points into overlapping local regions by the distance metric of the underlying space and extract local features capturing fine geometric structures from small neighborhoods; such local features are further grouped into larger units and processed to produce higher level features. This process is repeated until we obtain the feature of the whole point set. These methods have to address one issue: how to abstract local features through a local feature learner.

One popular method is to define a general learnable continuous convolution layer by predicting the continuous weight or feature function of the convolution operator [12, 13, 15, 16]. To present the idea clearly, we first review the definition of the convolution: $(f * w)(x) = \int_{y \in G} f(x+y)w(y)dy$, where f is the

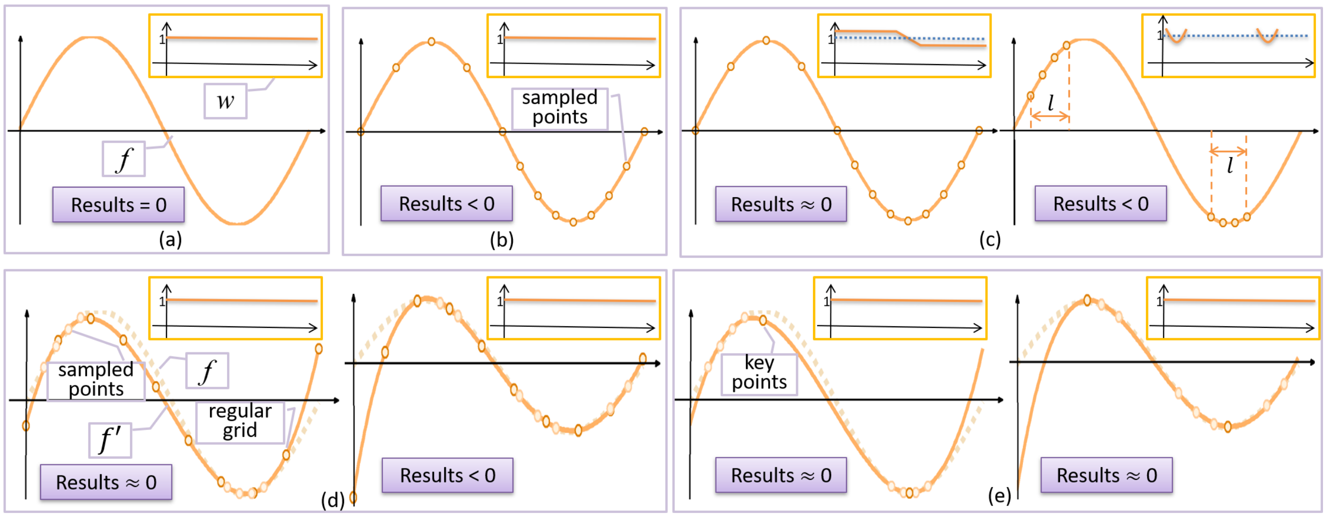


Fig. 1: Different convolution operations. (a): Convolution of continuous signal function $f = sin(x)$ and weight function $w = 1, x \in [0, 2 * \pi]$. The Convolution result should be 0. (b), (c), (d), and (e) illustrate different ways to compute the convolution results. (b) and (c) predict the continuous weight function w. (d) and (e) estimate the continuous feature function f. (b): The weight corresponding to each sampled point is computed by the estimated weight function w. Such convolution formulation is vulnerable to non-uniform sampling. (c): The density scale is used to revise the learned function w. However, the non-uniform sampling effects are only alleviated rather than eliminated. (d) interpolates point features onto a regular grid. Some unfaithful convolution results may be caused by the sampling imperfections. (e): Our FI-Conv interpolates point features onto some key points. This will result in a new convolution form that is robust to different samples and varying densities.

feature function on $\mathbb{R}^3$ to be convolved, w is the convolution weight function, and G is the domain of w. This formula can be discretized as follows: $(f*w)(x) = \sum_{v_i \in \hat{G}} f(x+v_i)w(v_i)$, where $\hat{G}$ is a sampling of G. Point clouds can be viewed as non-uniform samples of continuous $\mathbb{R}^3$ space. Furthermore, the feature corresponding to one point can be seen as a sampled point of function f.

Some methods try to represent the weight function w using an interpolation function [17, 16, 18], a multi-layer perceptron (MLP) [19], or a polynomial function [14]. Then $w(v_i)$, denoted by w_{v_i}, is obtained by estimated function w. However, in the case of non-uniformly sampling, this kind of methods may deviate from the desired filter response, since it does not optimize the positions of sampled points, see Fig. 1 (b). Additional technology aids, such as density function [20, 12], are employed to compensate for the non-uniform sampling. However, the non-uniform sampling effects are only alleviated rather than eliminated, see Fig. 1 (c).

Another line of work attempts to estimate features of some special points that are reasonably distributed and have been given a fixed order. PointCNN [13] learns a $\mathcal{X}$-transformation from the coordinates of sampled points by an MLP, then uses the $\mathcal{X}$-transformation to predict features of some special points. With ideal $\mathcal{X}$-transformations, pointCNN is independent of point orders. In practice, the learned $\mathcal{X}$-transformations are far from ideal, especially in terms of the permutation equivalence aspect. SPLATNet [15] and PCNN [21] interpolate point features onto a regular grid; then 3D convolution is applied on the grid; finally, they project convolved features back to point locations. However, the positions of grids are handcrafted, which reduces the degree of freedom. Some small interpolation errors caused by the sampling imperfections may affect the convolution results, see Fig. 1 (d).

In this paper, we propose a new way to perform convolution on 3D point clouds, called Feature Interpolation Convolution (FI-Conv), which is invariant to the order of points and robust to different samples and varying densities. Unlike SPLATNet and PCNN, we interpolate point features onto a set of learnable points, named key points, whose locations can be optimized by networks. The optimized locations of key points can be understood as some relatively stable points, such as extreme or inflection points of the feature function f, see Fig. 1 (e). Since such points are more robust to the typical sampling imperfections, such as noise, outliers, and sampling anisotropy, this will result in a new convolution form that is robust to different samples and varying densities. This is also proved through experimental results, see Section 4.2. Moreover, we can guarantee that the results of FI-Conv are invariant to permutations in theory, due to the interpolation results are not affected by the order of points.

2. Related work

Previous works on processing 3D data can be roughly categorized into image-based methods, voxel-based methods, and point-based methods.

Image-based and voxel-based methods. Image-based methods [8, 4, 22, 23] often project 3D point clouds or shapes into several 2D images and then apply convolutions on these images [8, 24, 25]. These approaches have achieved dominating performances on shape classification and retrieval. However, for scene segmentation, they suffer from occluded surfaces and density variations.

In the case of voxel-based methods, the points are projected on the 3D grid in Euclidean space and 3D convolutions are applied on the grid [3, 4, 26]. Reference [4] attempts to convert 3D point clouds into regular voxel grids. However, this method is limited by the high computational and memory cost associated with 3D convolutions. Some recent works [27, 5, 28, 22, 23] propose solutions to the above problems. Octnet [5] saves computation significantly by skipping convolution in empty space. SSCN [22] keeps the sparsity in convolved layers by computing the convolution only at activated points whose number does not increase when the convolution layers are stacked. However, the kernels used in these approaches are still high dimension and dense. Sparse kernels are designed by FPNN [27], but this approach cannot be applied recursively and used to learn hierarchical features.

Point-based methods. In recent years, with the rapid advances and demands in 3D sensing, some works directly take raw point clouds as input, instead of converting it to other formats. PointNet [9] is an early attempt at using deep networks to directly process point clouds. For each point, a multi-dimensional feature is extracted by a shared MLP, and the final global feature is obtained by max-pooling, which is a symmetric aggregation operation that does not rely on the order of input point sequence. PointNet does not capture local structure effectively, because the max-pooling layers are applied across all the points in the point cloud. Aiming at this problem, PointNet++ [10] proposes a hierarchical neural network to capture local structures, which can extract local features at different scales. It groups the set of points into overlapping local regions and extracts local features from local regions by PointNet. SpecGCN [29] proposes to leverage the power of spectral graph CNNs in the pointnet++ framework, while adopting a different pooling strategy. Shen *et al.* [30] present two new operations to improve PointNet with more efficient exploitation of local structures. The first one is a kernel correlation layer which computes an affinity between each data point's neighbors and kernels of learnable point sets. The second one is a graph-based pooling layer which recursively computes aggregation features by a nearest-neighbor-graph computed from 3D positions. Dynamic Graph CNN [31] generates edge features that describe the relationships between a point and its neighbors, and then apply a channel-wise symmetric aggregation operation (*e.g.* sum or max) on the edge features associated with all the edges emanating from each vertex. While above methods ([9, 10, 30, 31]) have theoretical guarantee in terms of achieving permutation-invariance, the symmetric aggregation operations come with the price of throwing away information.

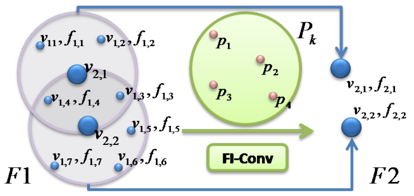
Some latest works view point clouds as discrete samples of a continuous function in $\mathbb{R}^3$ space. Various parameterizations of learnable continuous filter functions have thus been proposed. Some works use the multi-layer perceptron networks to estimate continuous weight functions [20, 12, 19, 32]. Instead of MLPs, a special family of filters is proposed by SpiderCNN [14] to approximate the weight function. Besides, references [17, 16, 18] compute the weight corresponding to each sample by the weighted combination of weight values of all kernel elements. We convert the operations performed in kernel space into feature space, which is easier to implement and robust to sampling density and distribution.

An alternative approach treats an attempt to give the unordered point set an order. PointSIFT [33] extracts the information of each point in different directions by an orientation-encoding unit, and stacks the orientation-encoding units for multi-scale representation. PointCNN [13] learns a χ-transformation from the input points and uses it to simultaneously weight and permute the input features associated with the points. Comparing to our method, PointCNN is unable to guarantee permutation-invariance, which is desired for point clouds. SPLATNet [15] and PCNN [21] interpolate input features onto a regular grid and do convolution over this grid. The filtered signal is then interpolated back onto the input signal. However, our method maps input features onto learnable key points which provides a more flexible framework.

3. Method

We introduce the FI-Conv in Section 3.1. In Section 3.2, we prove that the results of FI-Conv are invariant to permutations. The bright spots of FI-Conv are discussed in Section 3.3. Based on the FI-Conv, we design our FI-Conv layer (Section 3.4).

The input of FI-Conv is $F1 = \{(v_{1,i}, f_{1,v_i}) | i = 1, 2, \cdots, N_1\}$, i.e., a set of points $V1 = \{v_{1,i} | v_{1,i} \in R^3\}$, each associated with a feature $\{f_{1,v_i} | f_{1,v_i} \in R^{C1}\}$. $V2 = \{v_{2,i} | v_{2,i} \in R^3, i = 1, 2, \cdots, N_2\}$ is a set of representative points of $V1$. The representative points $V2$ can be at arbitrary locations and the common ways to construct it are random down-sampling or farthest point sampling of $V1$. We segment the $V1$ into N_2 groups, where each group corresponds to a local region composed by the neighbors of $v_{2,i}$. Key points are regarded as a set of learnable points $P_k = \{p_i | i = 1, 2, \cdots, N_k\}$. For each sub-group of $V1$, we would like to interpolate point features onto P_k; then 3D convolution is applied on the interpolated features. Finally, we get a higher level representation $F2 = \{(v_{2,i}, f_{v_{2,i}}) | i = 1, 2, \cdots, N_2\}$.



3.1. Feature Interpolation Convolution

Since the output features are associated with the representative points $V2$, FI-Conv takes their neighborhood points in $V1$, as well as the associated features, as input to convolve with. For simplicity, we denote a specific representative point in $V2$ as v, and its neighborhood with size K in $V1$ as $P_S = \{v_i | v_i \text{ is one of the } K \text{ nearest neighbors of } v\}$. Thus the FI-Conv input for this specific v is $S = \{(v_i, f_{v_i}) | v_i \in P_S, f_{v_i} \in R^{C1}\}$ and $P_k = \{p_i | i = 1, 2, \cdots, N_k\}$. The overall procedure of FI-Conv is shown in Fig. 2. We now explain the details of applying FI-conv on S.

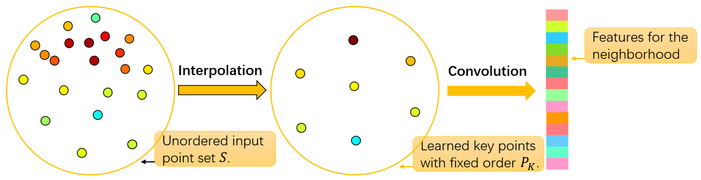

Interpolation
Convolution
Features for the neighborhood
Unordered input point set S.
Learned key points with fixed order P_K.

Fig. 2: Overview of the proposed FI-Conv. The input of FI-Conv is a set of points each associated with a feature (Left). Note that these points are unordered. Then, we interpolate point features onto learned key points that are ordered (Middle). Finally, a standard convolution operation is performed on P_K (Right).

First, we compute the feature f_{p_k} corresponding to each key point p_k by interpolating the features $\{f_{v_i} | v_i \in P_S\}$. Among the many choices for interpolation, we use Gaussian weights associated with distance:

$$f_{p_k} = \frac{1}{X_{p_k}} \sum_{v_i \in P_S} e^{\frac{-\|p_k - v_i\|^2}{\sigma^2}} f_{v_i} \tag{1}$$

where $X_{p_k} = \sum_{v_i \in \mathcal{N}} e^{\frac{-\|p_k - v_i\|^2}{\sigma^2}}$ is the normalization factor.

The features corresponding to points in S can be cast into the matrix $F_S = [f_{v_1}, f_{v_2}, \cdots, f_{v_K}]^T \in \mathbb{R}^{K \times C_1}$. Note that S is an unordered set, thus features in F_S can be in arbitrary order. The point set P_k is used for all the sub-groups of $V1$. If we give an order to points in P_k, the interpolated features of points in P_k can be cast into a fixed matrix $F_k = [f_{p_1}, f_{p_2}, \cdots, f_{p_{N_k}}]^T \in \mathbb{R}^{N_k \times C_1}$. The transformation matrix T between F_S and F_κ can be defined as:

$$T = \begin{bmatrix} \frac{1}{X_{p_1}} e^{\frac{-\|p_1 - v_1\|^2}{\sigma^2}} & \cdots & \frac{1}{X_{p_1}} e^{\frac{-\|p_1 - v_K\|^2}{\sigma^2}} \\ \frac{1}{X_{p_2}} e^{\frac{-\|p_2 - v_1\|^2}{\sigma^2}} & \cdots & \frac{1}{X_{p_2}} e^{\frac{-\|p_2 - v_K\|^2}{\sigma^2}} \\ \vdots & \ddots & \vdots \\ \frac{1}{X_{p_{N_k}}} e^{\frac{-\|p_{N_k} - v_1\|^2}{\sigma^2}} & \cdots & \frac{1}{X_{p_{N_k}}} e^{\frac{-\|p_{N_k} - v_K\|^2}{\sigma^2}} \end{bmatrix} \in \mathbb{R}^{N_k \times K}. \tag{2}$$

Thus, we obtain the following equation:

$$F_k = T \times F_S. \tag{3}$$

We call the operation $T \times F_S$ Interpolation Transformation (IT). After IT, the unordered point set will have a order which is identical to the order of points in P_k. A standard convolution operation can be performed on F_k:

$$\text{FI-Conv}(\kappa, F_S) = \text{conv}(\kappa, T \times F_S), \tag{4}$$

where $\kappa \in \mathbb{R}^{N_k \times C_1 \times C_2}$ is a convolution kernel, $conv(\cdot, \cdot)$ is the standard convolution operation, the output of FI-Conv(κ, F_S) is a $1 \times C_2$ local feature vector.

The elements of transformation matrix T are computed by Gaussian function which is a continuous differentiable function. Therefore, the positions of key points can be a fixed disposition, such as the centers of a 3D voxel grid, or optimized by networks. Experiments show that better performance can be achieved by learning the positions of key points by networks, see Section 4

3.2. *Invariance to the order of points*

Differently from images, point clouds are sets of unordered 3D points. The main challenge in building point cloud network is to make it ”indifferent” to the order of points, that is, the network is equivariant. Given a set $X = \{x_1, x_2, \cdots, x_n\}$, an equivariant function $f : X \to \mathbb{R}$ satisfies:

$$f(X) = f(\pi(X)), \tag{5}$$

where π is an arbitrary permutation and $\pi(X) = \{x_{\pi(1)}, x_{\pi(2)}, \cdots, x_{\pi(n)}\}$. Since the interpolation results are not affected by the order of points, we can guarantee that FI-Conv is equivariant in theory.

Theorem 1. *The FI-conv defined in* (4) *is equivariant,* i.e., *FI-Conv(κ, F_S) = FI-Conv$(\kappa, \pi(F_S))$, for all permutations π.*

Proof. The property follows from the definition of FI-Conv.

$$\begin{aligned} &\text{FI-Conv}(\kappa, \pi(F_S)) \\ &= \text{conv}(\kappa, [\frac{1}{X_{p_1}} \sum_{v_{\pi(i)} \in \mathcal{N}} e^{\frac{-\|p_1 - v_{\pi(i)}\|^2}{\sigma^2}} f_{v_{\pi(i)}}, \cdots, \frac{1}{X_{p_k}} \sum_{v_{\pi(i)} \in \mathcal{N}} e^{\frac{-\|p_k - v_{\pi(i)}\|^2}{\sigma^2}} f_{v_{\pi(i)}}]) \\ &= \text{conv}(\kappa, [\frac{1}{X_{p_1}} \sum_{v_i \in \mathcal{N}} e^{\frac{-\|p_1 - v_i\|^2}{\sigma^2}} f_{v_i}, \cdots, \frac{1}{X_{p_k}} \sum_{v_i \in \mathcal{N}} e^{\frac{-\|p_k - v_i\|^2}{\sigma^2}} f_{v_i}]) \\ &= \text{FI-Conv}(\kappa, F_S) \end{aligned}$$

□

3.3. *Related Results*

The form of FI-Conv is closely related with some methods which also built a continuous convolutional layer. Here, we quickly review some of these connections.

PointCNN. The core component of PointCNN is the $\mathcal{X}$-Conv which learns a $\mathcal{X}$-transformation from the input points and then uses it to simultaneously weigh 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. In details, the $\mathcal{X}$-Conv between features $F_S \in \mathbb{R}^{K \times C_1}$ and kernel $\kappa \in \mathbb{R}^{K \times C_1 \times C_2}$ is:

$$\mathcal{X}\text{-Conv}(\kappa, F_S) = \text{conv}(\kappa, MLP(P_S) \times F_S), \tag{6}$$

where the output of multi-layer perceptron $MLP(\cdot)$ is a transformation matrix with size $K \times K$, defined as $\mathcal{X}$-transformation by [13]. Note that for PointCNN, the number of neighborhood points is the same as that of kernel points. $MLP(P_S) \times F_S$ is supposed to predict features of some special points. With ideal $\mathcal{X}$-transformations, pointCNN is independent of point orders. In practice, the learned $\mathcal{X}$-transformations are far from ideal, especially in terms of the permutation equivalence aspect.

We can find the similarity between $\mathcal{X}$-Conv and FI-Conv, in the sense that both of them provide a mechanism to ”transform”

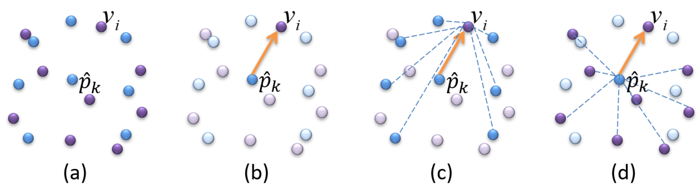


Fig. 3: Different methods to construct the relation between neighboring points and kernel points. (a) The distribution of neighboring points (purple) and kernel points (blue). Estimation the relation between v_i and $\hat{p}_k$ by the information of v_i and $\widehat{p_k}$ (b), the distribution of kernel points (c), or the distribution of neighboring points (d).

input into latent canonical forms for further standard convolution operation. However, our studies are set apart by two visible advantages: order invariance and flexible neighborhood size. Firstly, The IT operation uses linear interpolation weighted by distances to "transform" the features of unordered neighbors into that of ordered kernel points. Since the features of kernel points are computed by interpolation which is not affected by the order of neighbors, we can guarantee that the results of FI-Conv are invariant to permutations in theory. Second, the size of transformation matrix T is $K \times N_k$. Thus the neighborhood size K can vary across groups and the succeeding IT operation can convert the flexible number of neighbors into the fixed number N_k of points included in the P_k. In general, a larger value of N_k represents more accurate results and higher computational costs.

Representation the weight function by weighted methods. References [17, 16, 18] treat the kernel as a set of points (kernel points) with filter weights. The weight w_{v_i} of neighbor v_i is a weighted combination of filter weights corresponding to kernel points:

$$w_{v_i} = \sum_{k=1}^{\widehat{N}_k} t_{i,k} w_{\widehat{p_k}}, \tag{7}$$

where $\widehat{N}_k$ is the number of kernel points, $\widehat{p_k}$ represents a kernel point, $w_{\widehat{p_k}}$ is the filter weight corresponding to $\widehat{p_k}$, and $t_{i,k}$ is the relation between $\widehat{p_k}$ and v_i. Reference [17] and [18] use the distance between v_i and $\widehat{p_k}$ to compute $t_{i,k}$, *i.e.*, only the information of v_i and $\widehat{p_k}$ is considered. Boulch *et al.* [16] introduce the difference between v_i and all kernel points to estimate $t_{i,k}$, *i.e.*, the distribution of kernel points is used, see Fig. 3 (c). However, it ignores the distribution of neighboring points which is more important in the convolution. If we change the positions of $\{v_j | j \neq i\}$ (lilac points in Fig. 3 (c)), the values of $t_{i,k}$ and w_{v_i} are not changing. However, the value of w_{v_i} should be connected with the positions of $\{v_j | j \neq i\}$.

As we know, the interpolation performed in features can be translated into the operation on filter weights. Bringing the features interpolation formula (1) into convolution (4), we can get:

$$\begin{aligned} \text{FI-Conv}(\kappa, F_S) &= \text{conv}(\kappa, T \times F_S) \\ &= \sum_{k=1}^{N_k} \frac{1}{X_{p_k}} \Big(\sum_{v_i \in \mathcal{N}} e^{\frac{-\|p_k - v_i\|^2}{\sigma^2}} f_{v_i} \Big) \times w_k \\ &= \sum_{k=1}^{N_k} \Big(\sum_{v_i \in \mathcal{N}} \frac{1}{X_{p_k}} e^{\frac{-\|p_k - v_i\|^2}{\sigma^2}} f_{v_i} \times w_{p_k} \Big) \\ &= \sum_{v_i \in \mathcal{N}} \Big(\sum_{k=1}^{N_k} \frac{1}{X_{p_k}} e^{\frac{-\|p_k - v_i\|^2}{\sigma^2}} w_k \Big) \times f_{v_i}. \end{aligned} \tag{8}$$

The key point p_k can be seen as the kernel point $\hat{p}_k$ and the weight w_k can be see as the weight of p_k. Therefore, the relation $t_{i,k}$ is defined as:

$$t_{i,k} = \frac{1}{X_{p_k}} e^{\frac{-\|p_k - v_i\|^2}{\sigma^2}}. \tag{9}$$

The normalization factor $X_{p_k} = \sum_{v_i \in \mathcal{N}} e^{\frac{-\|p_k - v_i\|^2}{\sigma^2}}$ takes the difference between p_k and all neighboring points, *i.e.*, the distribution of neighboring points, into considering.

3.4. FI-Conv for Point Cloud Analysis

Using FI-Conv as a basic operator, we define a continuous convolution layer (FI-Conv layer). RS-CNN [32] and PointCNN [13] also define a continuous convolution layer by predicting the filter weight function (the former) or feature function (the latter). To evaluate the performance of FI-Conv, we use the architectures proposed by PointCNN and RS-CNN, and replace the continuous convolution layer in their networks by our FI-Conv layer. We depict the FI-Conv layer in Fig. 4.

3.4.1. PointCNN+FI-Conv

```
Algorithm 1: FI-Conv layer
  Input: $P_k$, $v$, $P_S$, $F_S$, $\kappa$.
  Output: Features "projected", or "aggregated", to $v$: $f_v$
    1. Move $P_S$ to local coordinate system of $v$ :
           $P'_S \leftarrow P_S - v$.
    2. Individually lift each point into $C_\delta$ dim. space:
           $F_\delta \leftarrow MLP_\delta(P'_S)$.
    3. Concatenate $F_\delta$ and $F_s$:
           $F_* \leftarrow [F_\delta, F_s]$.
    4. Compute the transformation matrix $T$:
           $T \leftarrow Fun(P_k, P'_S)$.
    5. Transform $F_*$ with the matrix $T$:
           $F_\kappa = T \times F_*$
    6. Finally, typical convolution between $\kappa$ and $F_k$:
           $f_v = Conv(\kappa, F_k)$.
```

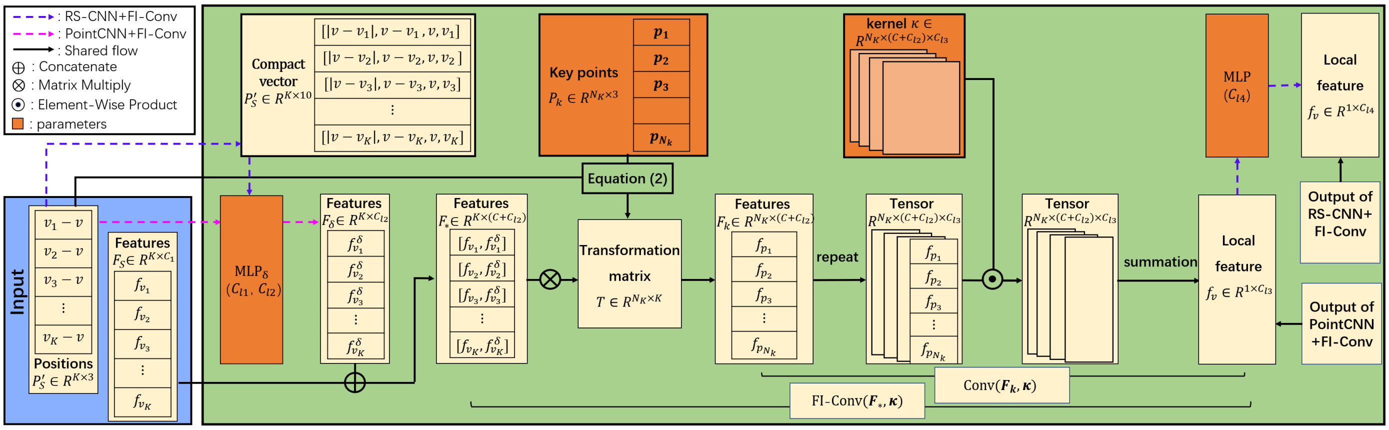

: RS-CNN+FI-Conv
: PointCNN+FI-Conv
: Shared flow
: Concatenate
: Matrix Multiply
: Element-Wise Product
: parameters
Input
Positions $P_S' \in R^{K\times3}$
Features $F_S \in R^{K\times C_1}$
Compact vector $P_S' \in R^{K\times10}$
MLP_δ (C_{l1}, C_{l2})
Features $F_\delta \in R^{K\times C_{l2}}$
Features $F_* \in R^{K\times(C+C_{l2})}$
Key points $P_k \in R^{N_K\times3}$
Equation (2)
Transformation matrix $T \in R^{N_K\times K}$
Features $F_k \in R^{N_K\times(C+C_{l2})}$
repeat
kernel $\kappa \in R^{N_K\times(C+C_{l2})\times C_{l3}}$
Tensor $R^{N_K\times(C+C_{l2})\times C_{l3}}$
summation
Local feature $f_v \in R^{1\times C_{l3}}$
MLP (C_{l4})
Local feature $f_v \in R^{1\times C_{l4}}$
Output of RS-CNN+ FI-Conv
Output of PointCNN +FI-Conv
Conv(F_k, κ)
FI-Conv(F_*, κ)

Fig. 4: Our pipeline for the FI-Conv layer in PointCNN+FI-Conv and RS-CNN+FI-Conv.

The only difference between our FI-Conv and $\mathcal{X}$-Conv provided by PointCNN is in the mechanism to ”transform” input into latent canonical forms (see Section 3.3). Therefore, we take the same operations in the convolution layer of pointCNN, except for the transformation mechanism. To make our paper self-contained, we give a brief introduction here and details are referred to [13]. Algorithm 1 depicts the FI-Conv layer.

First, a local coordinate system at the representative point v is built and we translate the neighboring points P_S to P_S' centered around the origin v (Line 1). Besides the associated features F_S, the local coordinates themselves are part of the input features as well. However, the local coordinates are of quite a different dimensionality and representation than the associated features. We first lift the coordinates into a higher dimensional and more abstract representation F_δ (Line 2), and then combine it with the associated features (Line 3). The lifting of coordinates into features is through a point-wise MLP_δ, which is the same as that in PointNet. In line 4, the transformation matrix T is defined by (2). Finally, we use IT operation to transform the input unordered features F_* into latent canonical form F_κ (Line 5) for further standard convolution operation (Line 6).

3.4.2. RS-CNN+FI-Conv

For RS-CNN, we make two changes to the convolution layer introduced above, see the dashed purple line in Fig. 4. First, the input of MLP_δ in Line 2 is changed into a compact vector with 10 channels, i.e., (3D Euclidean distance, $v - v_j$, v, v_j). This vector is used by RS-CNN to predict continuous filter weight functions. Second, we add a channel-raising mapping, provided by RS-CNN, on f_v.

4. Experiments

Our experiment consists of three parts: introducing the results of PointCNN+FI-Conv and RS-CNN+FI-Conv for classification (Section 4.1) and segmentation (Section 4.2), analyzing the proposed model (Section 4.3), and visualizing the positions of key points (Section 4.4). We conduct the evaluation for classification on 2D objects (MNIST [34]) and 3D objects (ModelNet40 [4]), semantic segmentation on ShapeNet Parts [35].

4.1. Object classification

Datasets: We evaluate our networks on both 2D and 3D point clouds. For 2D object classification, we evaluate our model on MNIST which is widely used for the sanity check of image CNNs. It contains images of handwritten digits with 60k training and 10k testing samplings. The original size of an image is 28 × 28. PointCNN [13] randomly samples 160 foreground pixels and converts them into point cloud representation, with the gray-scale value as the input feature. KCNet [30] transforms non-zero pixels in each image to points. PointNet++ [10] randomly samples 512 pixels and converts them into the point cloud. GDCNN [16] considers the input image as points (pixel coordinates) associated with color features (grey value). We create the input point cloud by the way of PointCNN and GDCNN. For 3D object classification, we evaluate our model on 40-categories benchmark ModelNet40 which is composed of 12311 3D mesh models with a 9843/2468 training/testing official split. We train a model for testing with 1024 points on the classification task by sampling points from the point cloud conversion of ModelNet40 provided by [9], where 2048 points are sampled from each mesh. During testing, similar to [9, 10, 32], we perform ten voting tests with random scaling and average the predictions. For more details for the strategy in training and testing, please refer to PointCNN and RS-CNN for PointCNN+FI-Conv and RS-CNN+FI-Conv, respectively.

Implementation details: For ModelNet40, PointCNN+ FI-Conv uses the 4-layers architecture provided by PointCNN and RS-CNN+FI-Conv uses the 2-layers single-scale-neighborhood architecture provided by RS-CNN. For Minst with 160 randomly sampled foreground points, PointCNN also provides an architecture in which the number of samples in the input layer and four convolution layers is 160, 160, 160, 120, and 120, respectively. We use this architecture for PointCNN and PointCNN+FI-Conv with 160 foreground samples as input. When each pixel in the image is treated as a point, we change these number into 734, 734, 734, 160, 128. For RS-CNN and RS-CNN+FI, we modify the 2-layers single-scale-neighborhood architecture to incorporate the new input. In this architecture, the number of samples in the input layer and two convolution layers is 1024, 512, and 128, respectively. For Minst with 160 randomly sampled foreground points, we

Table 1: Comparisons of classification accuracy (%) on ModelNet40 (nor: normal, -: unknown).

Methods	Input	#points	Accuracy
Pointwise-CNN [36]	xyz	1k	86.1
Deep Sets [37]	xyz	1k	87.1
ECC [11]	xyz	1k	87.4
PointNet [9]	xyz	1k	89.2
SCN [38]	xyz	1k	90.0
Flex-Conv [39]	xyz	1k	90.2
Kd-Net(depth=10)[40]	xyz	1k	90.6
PointNet++ [10]	xyz	1k	90.7
KCNet [30]	xyz	1k	90.8
MRTNet [41]	xyz	1k	91.2
Spec-GCN [29]	xyz	1k	91.5
DGCNN [31]	xyz	1k	92.2
MC-Conv [12]	xyz	1k	90.9
PCNN [21]	xyz	1k	92.3
GDCNN [16]	xyz	1k	91.6
Linked-DGCNN [42]	xyz	1k	91.6
DensePoint [43]	xyz	1k	93.2
SO-Net [44]	xyz	2k	92.9
KPConv [17]	xyz	7k	92.9
Kd-Net(depth=15)[40]	xyz	32k	91.8
O-CNN [23]	xyz,nor	-	90.6
Spec-GCN [29]	xyz,nor	2k	92.1
PointNet++ [10]	xyz,nor	5k	91.9
SpiderCNN [14]	xyz,nor	5k	92.4
PointConv [20]	xyz,nor	1k	92.5
SO-Net [44]	xyz,nor	5k	93.4
PointCNN [13]	xyz	1k	92.2
+ FI-Conv (ours)	xyz	1k	92.9
RS-CNN [32]	xyz	1k	92.9
+ FI-Conv (ours)	xyz	1k	93.4

change the number of samples in the input layer and the first convolution layer into 160. When each pixel in the image is treated as a point, we change the number of samples in the input layer into 734. For each layer, the number of key points is the same as that of neighboring points.

Results and discussion: We compare the classification results on ModelNet40 with several methods that are designed to consume these data with different core operators. The results are summarized in Tab. 1. The performance of RS-CNN+FI-Conv with input data size (1024 points) and features (coordinates only) is better than (or comparable with) PointConv (or SO-Net) which uses additional features (normal vectors) and requires denser point clouds (5000 points). Furthermore, we observe a reasonable increase in performance by simply replacing the $\mathcal{X}$-Conv in PointCNN and RS-Conv in RS-CNN with our FI-Conv. The results demonstrate the advantage of FI-Conv.

The classification results on MNIST are reported in Tab. 2, where we compare PointCNN+FI-Conv and RS-CNN+FI-Conv with the competitive PointNet++, KCNet, SpecGCN, PointCNN, GDCNN, and RS-CNN. The results of RS-CNN+FI-Conv with randomly sampled 160 foreground pixels

Table 2: Classification accuracy (%) on MNIST.

Methods	Input	Accuracy
KCNet [30]	Foreground points	99.2
PointNet++ [10]	512 Samples	99.49
Spec-GCN [29]	All points	99.58
GDCNN [16]	All points	99.61
PointCNN [13]	160 Foreground samples	99.54
+FI-Conv (ours)	160 Foreground samples	99.59
RS-CNN [32]	160 Foreground samples	99.63
+FI-Conv (ours)	160 Foreground samples	99.68
PointCNN [13]	All points	99.61
+FI-Conv (ours)	All points	99.67
RS-CNN [32]	All points	99.70
+FI-Conv (ours)	All points	99.74

are much better than the other methods. The classification accuracy can be further improved by considering each pixel as a point.

4.2. Semantic segmentation

Datasets: We evaluate FI-Conv for part segmentation on ShapeNet Parts which contains 16880 models (14006/2874 training/testing split) from 16 categories. Each point in an object corresponds to a part label (50 parts in total). On average each object consists of less than 6 parts. We sample points from each point cloud to train a model for testing with 2048 input points. The category label for each model is used for trimming irrelevant predictions, the same as that in [13, 22]. Note that RGB information is not used in previous methods. To make fair comparisons, we do not use them either. Other details of the strategy in training and testing, please refer to PointCNN and RS-CNN.

To further evaluate the effectiveness of FI-Conv, we compare the performance of PointNet, PointNet++, PointCNN, and PointCNN+FI-Conv on ScanNet. This is a semantic voxel labeling dataset with a total of 1513 scanned and reconstructed indoor scenes. We use 1201 scenes for training, 312 scenes for testing without RGB information. First, the data is split by room. Then the rooms are sliced into 1.5m by 1.5m blocks, with 0.3m padding. We sample $\mathcal{N}(2048, 256^2)$ points from each sliced block to train a model for testing with 2048 input points.

Implementation details: For RS-CNN+FI-Conv, we use the 4-layers multi-scale-neighborhood architecture provided by RS-CNN. The neighborhood size of RS-CNN is 30 $\sim$ 50. To balance the segmentation quality and computational cost, the number of key points is set to 10 for each FI-Conv layer. PointCNN also provides a Conv-DeConv architecture. Note that, in this segmentation network, both the "Conv" and "DecConv" use the $\mathcal{X}$-Conv. In FI-Conv layer, we need to compute a transformation matrix T to transform feature matrix F_* (features corresponding to neighbors P_S) to F_k (features corresponding to key points). However, the construction method of T is based on the assumption that the feature function is smooth. This makes the obtained features F_k confused over the edge. Therefore, in the Conv part, we replace the $\mathcal{X}$-Conv layer

Table 3: Shape part segmentation results (%) on ShapeNet Parts benchmark (nor: normal, -: unknown).

method	input	class mIoU	instance mIoU	air plane	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor bike	mug	pistol	rocket	skate board	table
Kd-Net [40]	4k	77.4	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
PointNet [9]	2k	80.4	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
RS-Net [45]	-	81.4	84.9	82.7	86.4	84.1	78.2	90.4	69.3	91.4	87.0	83.5	95.4	66.0	92.6	81.8	56.1	75.8	82.2
SCN [38]	1k	81.8	84.6	83.8	80.8	83.5	79.3	90.5	69.8	91.7	86.5	82.9	96.0	69.2	93.8	82.5	62.9	74.4	80.8
PCNN [21]	2k	81.8	85.1	82.4	80.1	85.5	79.5	90.8	73.2	91.3	86.0	85.0	95.7	73.2	94.8	83.3	51.0	75.0	81.8
SPLATNet [15]	-	82.0	84.6	81.9	83.9	88.6	79.5	90.1	73.5	91.3	84.7	84.5	**96.3**	69.7	95.0	81.7	59.2	70.4	81.3
KCNet [30]	2k	82.2	84.7	82.8	81.5	86.4	77.6	90.3	76.8	91.0	87.2	84.5	95.5	69.2	94.4	81.6	60.1	75.2	81.3
DGCNN [31]	2k	82.3	85.1	**84.2**	83.7	84.4	77.1	90.9	78.5	91.5	87.3	82.9	96.0	67.8	93.3	82.6	59.7	75.5	82.0
Linked-DGCNN [42]	2k	82.2	85.1	84.0	83.0	84.9	78.4	90.6	74.4	91.0	88.1	83.4	95.8	67.4	94.9	82.3	59.2	76.0	81.9
DensePoint [43]	2k	84.2	86.4	84.0	85.4	**90.0**	79.2	91.1	**81.6**	91.5	87.5	84.7	95.9	74.3	94.6	82.9	**64.6**	76.8	83.7
PointNet++ [10]	2k,nor	81.9	85.1	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
SyncCNN [46]	mesh	82.0	84.7	81.6	81.7	81.9	75.2	90.2	74.9	93.0	86.1	84.7	95.6	66.7	92.7	81.6	60.6	82.9	82.1
SO-Net [44]	1k,nor	80.8	84.6	81.9	83.5	84.8	78.1	90.8	72.2	90.1	83.6	82.3	95.2	69.3	94.2	80.0	51.6	72.1	82.6
SpiderCNN [14]	2k,nor	82.4	85.3	83.5	81.0	87.2	77.5	90.7	76.8	91.1	87.3	83.3	95.8	70.2	93.5	82.7	59.7	75.8	82.8
PointCNN [13]	2k	83.2	85.2	82.4	83.3	89.8	79.4	90.1	73.0	91.4	87.9	84.1	95.7	73.5	94.7	82.9	61.0	79.4	82.6
+FI-Conv(ours)	2k	83.6	85.4	83.4	82.3	87.8	**80.7**	90.2	77.0	**91.9**	87.2	84.5	96.0	71.9	**95.6**	83.8	63.6	**80.1**	81.9
RS-CNN [32]	2k	84.0	86.2	83.5	84.8	88.8	79.6	91.2	81.1	91.6	88.4	**86.0**	96.0	73.7	94.1	83.4	60.5	77.7	83.6
+FI-Conv(ours)	2k	**84.2**	**86.5**	83.8	**86.7**	89.4	80.6	**91.6**	76.6	91.5	**88.8**	85.3	96.0	**75.1**	95.5	**83.9**	62.3	76.6	**84.0**

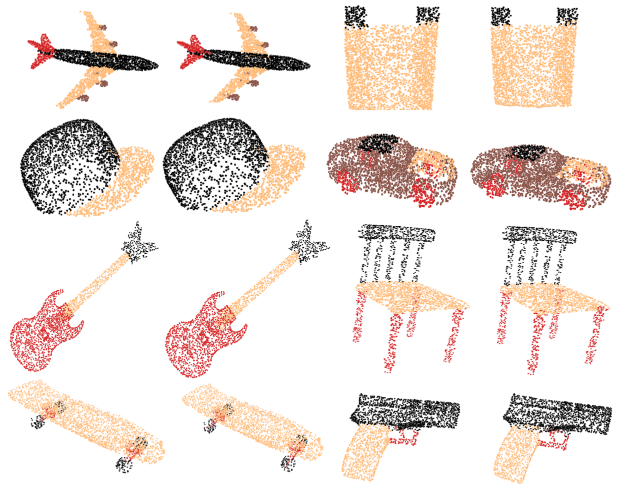


Fig. 5: Examples of semantic segmentation results on ShapeNet Parts. For each pair of objects, the left one is the ground truth and the right one is predicted by PointCNN+FI-Conv.

by our FI-Conv layer. However, in the DeConv part, we still use the $\mathcal{X}$-Conv layer proposed by PointCNN. The neighborhood size of PointCNN is 8 ∼ 16. For each FI-Conv layer, the number of key points is the same as the number of neighbors.

Results and discussion: For ShapeNet Parts, except for standard IoU (Interover-Union) on each category, we also report two types of mean IoU (mIoU) that are averaged across all classes and all instances, respectively. Tab. 3 summarizes the quantitative comparisons with the state-of-the-art methods. We visualize example semantic labeling results in Fig. 5.

RS-CNN+FI-Conv achieves the best performance with class mIoU of 84.2% and instance mIoU of 86.5%. Furthermore, it sets new state of the arts in the xyz-based methods over six categories. For the same input and architecture, the class mIoU/instance mIoU of RS-CNN+FI-Conv and PointCNN+FI-Conv is slightly higher than that of RS-CNN and PointCNN, respectively. These improvements demonstrate the effectiveness of FI-Conv to diverse shapes.

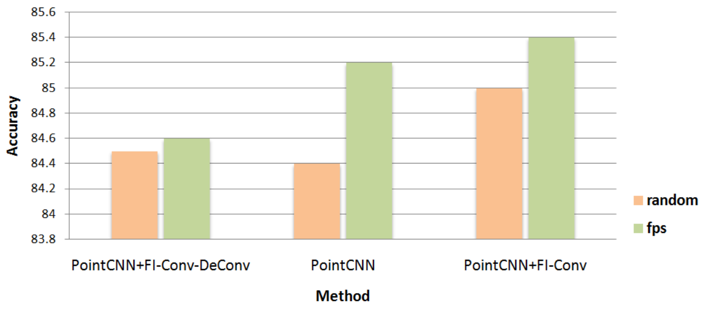


Fig. 6: Comparisons of instance mIoU (%) on ShapeNet Parts with random down-sampling and farthest point sampling (fps).

Moreover, we find the FI-Conv is more robust for the sampling quality. In both Conv and DeConv part of PointCNN segmentation architecture, we replace the $\mathcal{X}$-Conv layer by our FI-Conv layer and named the obtain network PointCNN+FI-Conv-DeConv. Fig. 6 shows the results of ShapeNet Parts obtained by PointCNN, PointCNN+FI-Conv, and PointCNN+FI-Conv-DeConv with random down-sampling and farthest point sampling. In the case of farthest point sampling, PointCNN outperforms PointCNN+FI-Conv-DeConv with a 0.6% gap. The reason could be that we suppose the feature function is smooth. Better performance can be achieved by PointCNN+FI-Conv which only uses FI-Conv in the Conv part. Furthermore, since farthest point sampling obtains a more uniform point distribution, the results using farthest point sampling are better than those using random down-sampling. But PointCNN+FI-Conv and PointCNN+FI-Conv-DeConv are only slightly affected and achieve higher accuracy than PointCNN when random down-sampling is employed.

For ScanNet, we convert point cloud label prediction into voxel labeling following [9, 47]. The results of ScanNet dataset are shown in Tab. 4. We can see that FI-Conv also slightly increases the performance on ScanNet dataset.

4.3. Model analysis

Stress test on small number of input points. We evaluate the performance of PointCNN+FI-Conv and RSCNN+FI-

Table 4: Segmentation comparisons on ScanNet in per voxel accuracy (%).

PointNet [9]	PointNet++ [10]	PointCNN [13]	PointCNN+FI-Conv
73.9	84.5	84.9	85.1

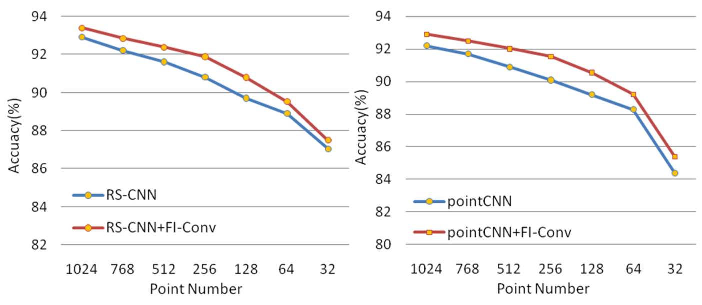


Fig. 7: Stress test on ModelNet40 classification.

Conv with different number of input points on ModelNet40 classification task and summarize the results in Fig. 7. The results show that our FI-Conv performs well in this testing. RS-CNN+FI-Conv (or PointCNN+FI-Conv) achieves higher accuracy than RS-CNN (or PointCNN) at each density. Moreover, when the input point number is reduced to 128/64, PointCNN+FI-Conv (90.56%/89.22 accuracy) and RS-CNN+FI-Conv (90.8%/89.52 accuracy) achieve comparable results with PointNet++/PointNet with 1024 input points. In such settings, the inference runs at 1.2ms-0.6ms per sample on N-Vidia GTX 1080Ti GPU, making the networks quite promising for real-time recognition applications with low-resolution point clouds input, such as autonomous driving.

Different construction methods for kernel points. In this section, we investigate several construction methods for the key points, including the random method (RAN), regulation method (REG), random with optimization method (RAN+OPT), and regulation with optimization method (REG+OPT). Specifically, the RAN generates key points randomly. The REG divides the unit cube by a 3D voxel grid and then takes the centers as the key points. RAN+OPT (or REG+OPT) generates key points randomly (or regularly), and then optimizes their positions by networks.

The classification results of PointCNN+FI-Conv on ModelNet40 are summarized in Tab. 5. All the construction methods achieve better performance than PointCNN (92.2% accuracy). Although REG obtains better results than RAN, it is not free in selecting the number of key points. Positions optimized by networks achieve better performance. When the number of key points used in each layer is the same, REG+OPT can get better performance than RAN+OPT, see the first row of Tab. 5. But, for REG+OPT, the number of key points is not free, since key points are initialized on the centers of a 3D voxel grid. In our implementation, we use RAN+OPT to obtain the final key points.

Table 5: Comparisons of classification accuracy (%) on ModelNet40 with different construction methods for key points. Num means the number of key points used in each layer.

Num	RAN	REG	RAN+OPT	REG+OPT
8 + 8 + 27 + 27	92.37	92.41	92.69	92.82
8 + 12 + 16 + 16	92.32	-	92.85	-

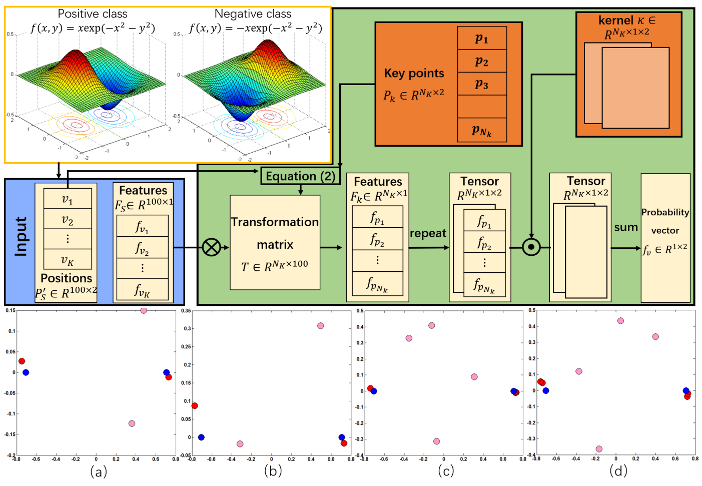

Positive class
$f(x,y)=x\exp(-x^2-y^2)$
Negative class
$f(x,y)=-x\exp(-x^2-y^2)$
Key points $P_k \in R^{N_K\times 2}$
kernel $\kappa \in R^{N_K\times 1\times 2}$
Input
Positions $P_S' \in R^{100\times 2}$
Features $F_S \in R^{100\times 1}$
Equation (2)
Transformation matrix $T \in R^{N_K\times 100}$
Features $F_k \in R^{N_K\times 1}$
repeat
Tensor $R^{N_K\times 1\times 2}$
sum
Probability vector $f_v \in R^{1\times 2}$
(a)
(b)
(c)
(d)

Fig. 8: Our pipeline for the toy example (top row) and learned key points (bottom row). Pink points and red points are the initialized and optimized positions of key points, respectively. Blue points are the extreme points of function $f(x, y) = xexp(-x^2 - y^2)$ and $f(x, y) = -xexp(-x^2 - y^2)$.

4.4. Visualization

In our implementation, the positions of key points are optimized by networks. Ideally, the optimized locations of key points should be some relatively stable points, such as extreme or inflection points of the feature function. We give a toy example to illustrate this point.

We construct a binary classification problem. The positive class is sampled from surface $f(x, y) = xexp(-x^2 - y^2)$, and the negative class is sampled from surface $f(x, y) = -xexp(-x^2 - y^2)$. For the positive and negative classes, we collect 6000 data for training and 2000 data for testing, including non-uniform sampling and occlusion. We randomly sample 100 points from region $[-2, 2] \times [-2, 2]$ in xoy plane. The function value corresponding to each sample point is regarded as its input feature. We construct a one layer neural network which includes a FI-conv with 2 (or 4) key points and 2 convolution kernel for data classification. We depict the used network in the top row of Fig. 8. For such a simple problem, the accuracy can achieve 100%.

The bottom row of Fig. 8 visualizes the initialized and optimized positions of key points. It can be seen that the optimized points are near $(\frac{\sqrt{2}}{2}, 0)$ and $(-\frac{\sqrt{2}}{2}, 0)$, which are the extreme points of function $f(x, y) = xexp(-x^2 - y^2)$ and $f(x, y) = -xexp(-x^2 - y^2)$.

5. Conclusion

In this work, we propose a novel way to perform convolution operation on 3D point clouds, called FI-Conv. FI-Conv learns a set of key points which are some relatively stable points such as extreme points and inflection points. Then, we estimate the key points' features by interpolating the features of neighboring points. Finally, a standard convolution operation is applied to these estimated features. We can guarantee the results of FI-Conv are invariant to permutations in theory. Applying FI-Conv to build deep convolutional networks (PointCNN+FI-Conv and RS-CNN+FI-Conv) achieves great performance on multiple challenging benchmark datasets and tasks.

We suppose the feature function is smooth. This hypothesis is false for the boundaries of the segmented regions. The idea of bilateral filtering may be helpful to get the boundaries features. Besides, we would like to adopt FI-Conv into more mainstream image convolutional network architectures. This will further improve the performance of point cloud convolutional networks.

References

[1] Krizhevsky, A, Sutskever, I, Hinton, GE. Imagenet classification with deep convolutional neural networks. Communications of the ACM 2017;60(6):84–90.

[2] Simonyan, K, Zisserman, A. Very deep convolutional networks for large-scale image recognition. International Conference on Learning Representations 2015;.

[3] Maturana, D, Scherer, SA. Voxnet: A 3d convolutional neural network for real-time object recognition. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. 2015, p. 922–928.

[4] Wu, Z, Song, S, Khosla, A, Yu, F, Zhang, L, Tang, X, et al. 3d shapenets: A deep representation for volumetric shapes. In: IEEE Conference on Computer Vision and Pattern Recognition. 2015, p. 1912–1920.

[5] Riegler, G, Ulusoy, AO, Geiger, A. Octnet: Learning deep 3d representations at high resolutions. In: IEEE Conference on Computer Vision and Pattern Recognition. 2017, p. 6620–6629.

[6] Feng, Y, Zhang, Z, Zhao, X, Ji, R, Gao, Y. GVCNN: group-view convolutional neural networks for 3d shape recognition. In: IEEE Conference on Computer Vision and Pattern Recognition. 2018, p. 264–272.

[7] Guo, H, Wang, J, Gao, Y, Li, J, Lu, H. Multi-view 3d object retrieval with deep embedding network 2016;25(12):5526–5537.

[8] Su, H, Maji, S, Kalogerakis, E, Learned-Miller, E. Multi-view convolutional neural networks for 3d shape recognition. In: IEEE International Conference on Computer Vision. 2015, p. 945–953.

[9] Qi, CR, Su, H, Mo, K, Guibas, LJ. Pointnet: Deep learning on point sets for 3d classification and segmentation. In: IEEE Conference on Computer Vision and Pattern Recognition. 2017, p. 77–85.

[10] Qi, CR, Yi, L, Su, H, Guibas, LJ. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: Advances in Neural Information Processing Systems. 2017, p. 5099–5108.

[11] Simonovsky, M, Komodakis, N. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: IEEE Conference on Computer Vision and Pattern Recognition. 2017, p. 29–38.

[12] Hermosilla, P, Ritschel, T, Vázquez, P, Vinacua, À, Ropinski, T. Monte carlo convolution for learning on non-uniformly sampled point clouds. ACM Transactions on Graphics 2018;37(6):235:1–235:12.

[13] Li, Y, Bu, R, Sun, M, Wu, W, Di, X, Chen, B. Pointcnn: Convolution on x-transformed points. In: Advances in Neural Information Processing Systems. 2018, p. 828–838.

[14] Xu, Y, Fan, T, Xu, M, Zeng, L, Qiao, Y. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In: European Conference on Computer Vision; vol. 11212. 2018, p. 90–105.

[15] Su, H, Jampani, V, Sun, D, Maji, S, Kalogerakis, E, Yang, M, et al. Splatnet: Sparse lattice networks for point cloud processing. In: IEEE Conference on Computer Vision and Pattern Recognition. 2018, p. 2530–2539.

[16] Boulch, A. Generalizing discrete convolutions for unstructured point clouds. In: Eurographics Workshop on 3D Object Retrieval. 2019, p. 71–78.

[17] Thomas, H, Qi, CR, Deschaud, J, Marcotegui, B, Goulette, F, Guibas, LJ. Kpconv: Flexible and deformable convolution for point clouds. In: IEEE International Conference on Computer Vision. 2019, p. 6410–6419.

[18] Xiong, Y, Ren, M, Liao, R, Wong, K, Urtasun, R. Deformable filter convolution for point cloud reasoning. In: NeurIPS Workshop on Sets & Partitions. 2019,.

[19] Wang, S, Suo, S, Ma, W, Pokrovsky, A, Urtasun, R. Deep parametric continuous convolutional neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition. 2018, p. 2589–2597.

[20] Wu, W, Qi, Z, Li, F. Pointconv: Deep convolutional networks on 3d point clouds. In: IEEE Conference on Computer Vision and Pattern Recognition. 2019, p. 9621–9630.

[21] Atzmon, M, Maron, H, Lipman, Y. Point convolutional neural networks by extension operators. ACM Transactions on Graphics 2018;37(4):71:1–71:12.

[22] Graham, B, Engelcke, M, van der Maaten, L. 3d semantic segmentation with submanifold sparse convolutional networks. In: IEEE Conference on Computer Vision and Pattern Recognition. 2018, p. 9224–9232.

[23] Wang, P, Liu, Y, Guo, Y, Sun, C, Tong, X. O-CNN: octree-based convolutional neural networks for 3d shape analysis. ACM Transactions on Graphics 2017;36(4):72:1–72:11.

[24] Boulch, A, Saux, BL, Audebert, N. Unstructured point cloud semantic labeling using deep segmentation networks. In: Eurographics Workshop on 3D Object Retrieval. 2017,.

[25] Ma, C, Guo, Y, Yang, J, An, W. Learning multi-view representation with lstm for 3d shape recognition and retrieval. IEEE Transactions on Multimedia 2019;21(5):1169–1182.

[26] Qi, CR, Su, H, Nießner, M, Dai, A, Yan, M, Guibas, LJ. Volumetric and multi-view cnns for object classification on 3d data. In: IEEE Conference on Computer Vision and Pattern Recognition. 2016, p. 5648–5656.

[27] Li, Y, Pirk, S, Su, H, Qi, CR, Guibas, LJ. FPNN: field probing neural networks for 3d data. In: Advances in Neural Information Processing Systems. 2016, p. 307–315.

[28] Tatarchenko, M, Dosovitskiy, A, Brox, T. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In: IEEE International Conference on Computer Vision. 2017, p. 2107–2115.

[29] Wang, C, Samari, B, Siddiqi, K. Local spectral graph convolution for point set feature learning. In: European Conference on Computer Vision; vol. 11208. 2018, p. 56–71.

[30] Shen, Y, Feng, C, Yang, Y, Tian, D. Mining point cloud local structures by kernel correlation and graph pooling. In: IEEE Conference on Computer Vision and Pattern Recognition. 2018, p. 4548–4557.

[31] Wang, Y, Sun, Y, Liu, Z, Sarma, SE, Bronstein, MM, Solomon, JM. Dynamic graph CNN for learning on point clouds. ACM Transactions on Graphics 2019;38(5):146:1–146:12.

[32] Liu, Y, Fan, B, Xiang, S, Pan, C. Relation-shape convolutional neural network for point cloud analysis. In: IEEE Conference on Computer Vision and Pattern Recognition. 2019, p. 8895–8904.

[33] Jiang, M, Wu, Y, Lu, C. Pointsift: A sift-like network module for 3d point cloud semantic segmentation. arXiv preprint arXiv: 80700652 2018;.

[34] Lecun, Y, Bottou, L, Bengio, Y, Haffner, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE 1998;86(11):2278– 2324.

[35] Yi, L, Shao, L, Savva, M, Huang, H, Zhou, Y, Wang, Q, et al. Large-scale 3d shape reconstruction and segmentation from shapenet core55. arXiv preprint arXiv:171006104 2017;.

[36] Hua, B, Tran, M, Yeung, S. Pointwise convolutional neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition. 2018, p. 984–993.

[37] Zaheer, M, Kottur, S, Ravanbakhsh, S, Póczos, B, Salakhutdinov, R, Smola, AJ. Deep sets. In: Advances in Neural Information Processing Systems. 2017, p. 3391–3401.

[38] Xie, S, Liu, S, Chen, Z, Tu, Z. Attentional shapecontextnet for point cloud recognition. In: IEEE Conference on Computer Vision and Pattern Recognition. 2018, p. 4606–4615.

[39] Groh, F, Wieschollek, P, Lensch, HPA. Flex-convolution - million-

scale point-cloud learning beyond grid-worlds. In: Asian Conference on Computer Vision; vol. 11361. 2018, p. 105–122.

[40] Klokov, R, Lempitsky, VS. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In: International Conference on Computer Vision. 2017, p. 863–872.

[41] Gadelha, M, Wang, R, Maji, S. Multiresolution tree networks for 3d point cloud processing. In: European Conference on Computer Vision; vol. 11211. 2018, p. 105–122.

[42] Zhang, K, Hao, M, Wang, J, de Silva, CW, Fu, C. Linked dynamic graph CNN: learning on point cloud via linking hierarchical features. CoRR 2019;abs/1904.10014.

[43] Liu, Y, Fan, B, Meng, G, Lu, J, Xiang, S, Pan, C. Densepoint: Learning densely contextual representation for efficient point cloud processing. In: International Conference on Computer Vision. 2019, p. 5238–5247.

[44] Li, J, Chen, BM, Lee, GH. So-net: Self-organizing network for point cloud analysis. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. 2018, p. 9397–9406.

[45] Huang, Q, Wang, W, Neumann, U. Recurrent slice networks for 3d segmentation of point clouds. In: Conference on Computer Vision and Pattern Recognition. 2018, p. 2626–2635.

[46] Yi, L, Su, H, Guo, X, Guibas, LJ. Syncspeccnn: Synchronized spectral CNN for 3d shape segmentation. In: Conference on Computer Vision and Pattern Recognition. 2017, p. 6584–6592.

[47] Dai, A, Chang, AX, Savva, M, Halber, M, Funkhouser, TA, Nießner, M. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In: IEEE Conference on Computer Vision and Pattern Recognition. 2017, p. 2432–2443.