SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters

Yifan Xu[1,2], Tianqi Fan[2], Mingye Xu[2], Long Zeng[1], and Yu Qiao[2(✉)]

[1] Tsinghua University, Beijing, China
xuyf16@mails.tsinghua.edu.cn, zenglong@sz.tsinghua.edu.cn
[2] Guangdong Key Lab of Computer Vision and Virtual Reality, SIAT-SenseTime
Joint Lab, Shenzhen Institutes of Advanced Technology, Chinese Academy
of Sciences, Shenzhen, People's Republic of China
{tq.fan,my.xu,yu.qiao}@siat.ac.cn

Abstract. Deep neural networks have enjoyed remarkable success for various vision tasks, however it remains challenging to apply CNNs to domains lacking a regular underlying structures such as 3D point clouds. Towards this we propose a novel convolutional architecture, termed SpiderCNN, to efficiently extract geometric features from point clouds. SpiderCNN is comprised of units called SpiderConv, which extend convolutional operations from regular grids to irregular point sets that can be embedded in $\mathbb{R}^n$, by parametrizing a family of convolutional filters. We design the filter as a product of a simple step function that captures local geodesic information and a Taylor polynomial that ensures the expressiveness. SpiderCNN inherits the multi-scale hierarchical architecture from classical CNNs, which allows it to extract semantic deep features. Experiments on ModelNet40 demonstrate that SpiderCNN achieves state-of-the-art accuracy 92.4% on standard benchmarks, and shows competitive performance on segmentation task.

Keywords: Convolutional neural network
Parametrized convolutional filters · Point clouds

1 Introduction

Convolutional neural networks are powerful tools for analyzing data that can naturally be represented as signals on regular grids, such as audio and images [10]. Thanks to the translation invariance of lattices in $\mathbb{R}^n$, the number of parameters in a convolutional layer is independent of the input size. Composing convolution layers and activation functions results in a multi-scale hierarchical learning

---

Y. Xu and T. Fan contributed equally.
Work done during Yifan Xu's internship at SIAT.

pattern, which is shown to be very effective for learning deep representations in practice.

With the recent proliferation of applications employing 3D depth sensors [23] such as autonomous navigation, robotics and virtual reality, there is an increasing demand for algorithms to efficiently analyze point clouds. However, point clouds are distributed irregularly in $\mathbb{R}^3$, lacking a canonical order and translation invariance, which prohibits using CNNs directly. One may circumvent this problem by converting point clouds to 3D voxels and apply 3D convolutions [13]. However, volumetric methods are computationally inefficient because point clouds are sparse in 3D as they usually represent 2D surfaces. Although there are studies that improve the computational complexity, it may come with a performance trade off [2,18]. Various studies are devoted to making convolution neural networks applicable for learning on non-Euclidean domains such as graphs or manifolds by trying to generalize the definition of convolution to functions on manifolds or graphs, enriching the emerging field of geometric deep learning [3]. However, it is challenging theoretically because convolution cannot be naturally defined when the space does not carry a group action, and when the input data consists of different shapes or graphs, it is difficult to make a choice for convolutional filters.[1]
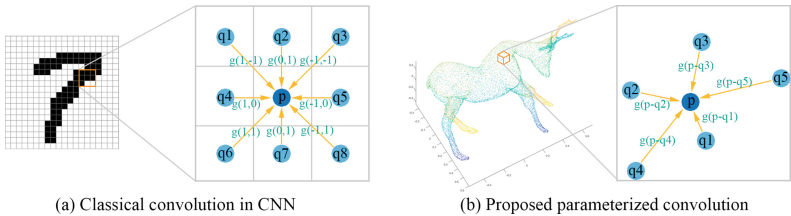

(a) Classical convolution in CNN   (b) Proposed parameterized convolution

Fig. 1. The integral formula for convolution between a signal f and a filter g is $f * g(p) = \int_{q \in \mathbb{R}^n} f(q)g(p-q)dq$. Discretizing the integral formula on a set of points P in $\mathbb{R}^n$ gives $f * g(p) = \sum_{q \in P, \|p-q\| \leq r} f(q)g(p-q)$ if g is supported in a ball of radius r. **(a)** when P can be represented by regular grids, only 9 values of a filter g are needed to compute the convolution due to the translation invariance of the domain. **(b)** when the signal is on point clouds, we choose the filter g from a parameterized family of function on $\mathbb{R}^3$.

In light of the above challenges, we propose an alternative convolutional architecture, SpiderCNN, which is designed to directly extract features from point clouds. We validate its effectiveness on classification and segmentation benchmarks. By discretizing the integral formula of convolution as shown in Fig. 1, and using a special family of parametrized non-linear functions on $\mathbb{R}^3$ as filters, we introduce a novel convolutional layer, SpiderConv, for point clouds.

The family of filters is designed to be expressive while still being feasible to optimize. We combine simple step functions, which are used to capture the coarse

---

[1] There is no canonical choice of a domain for these filters.

geometry described by local geodesic distance, with order-3 Taylor expansions, which ensure the filters are complex enough to capture intricate local geometric variations. Experiments in Sect. 4 show that SpiderCNN with a relatively simple network architecture achieves the state-of-the-art performance for classification on ModelNet40 [4], and shows competitive performance for segmentation on ShapeNet-Part [4].

2   Related Work

First we discuss deep neural network based approaches that target point clouds data. Second, we give a partial overview of geometric deep learning.

Point Clouds as Input: PointNet [15] is a pioneering work in using deep networks to directly process point sets. A spatial encoding of each point is learned through a shared MLP, and then all individual point features aggregate to a global signature through max-pooling, which is a symmetric operation that doesn't depend on the order of input point sequence.

While PointNet works well to extract global features, its design limits its efficacy at encoding local structures. Various studies addressing this issue propose different grouping strategies of local features in order to mimic the hierarchical learning procedure at the core of classical convolutional neural networks. Point-Net++ [17] uses iterative farthest point sampling to select centroids of local regions, and PointNet to learn the local pattern. Kd-Network [9] subdivides the space using K-d trees, whose hierarchical structure serves as the instruction to aggregate local features at different scales. In SpiderCNN, no additional choice for grouping or sampling is needed, for our filters handle the issue automatically.

The idea of using permutation-invariant functions for learning on unordered sets is further explored by DeepSet [22]. We note that the output of SpiderCNN does not depend on the input order by design.

Voxels as Input: VoxNet [13] and Voxception-ResNet [2] apply 3D convolution to a voxelization of point clouds. However, there is a high computational and memory cost associated with 3D convolutions. A variety of work [6,7,18] has aimed at exploiting the sparsity of voxelized point clouds to improve the computational and memory efficiency. OctNet [18] modified and implemented convolution operations to suit a hybrid grid-octree data structure. Vote3Deep [6] uses a feature-centric voting scheme so that the computational cost is proportional to the number of points with non-zero features. Sparse Submanifold CNN [7] computes the convolution only at activated points whose number does not increase when the convolution layers are stacked. In comparison, SpiderCNN can use point clouds as input directly and can handle very sparse input.

Convolution on Non-euclidean Domain: There are two main philosophically different approaches to define convolutions for non-Euclidean domains: one is spatial and the other is spectral. The recent work ECC [20] defines convolution-like operations on graphs where filter weights are conditioned on edge labels. Viewing point clouds as a graph, and taking the filters to be MLPs, SpiderCNN

and ECC [20] result in similar convolution. However, we show that our proposed family of filters outperforms MLPs.

Spatial Methods: GeodesicCNN [12] is an early attempt at applying neural networks to shape analysis. The philosophy behind GeodesicCNN is that for a Riemannian manifold, the exponential map identifies a local neighborhood of a point to a ball in the tangent space centered at the origin. The tangent plane is isomorphic to $\mathbb{R}^d$ where we know how to define convolution.

Let M be a mesh surface, and let $F : M \to \mathbb{R}$ be a function, GeodesicCNN first uses the patch operator D to map a point p and its neighbors $N(p)$ to the lattice $\mathbb{Z}^2 \subseteq \mathbb{R}^2$, and applies Eq. 2. Explicitly, $F * g(p) = \sum_{j \in J} g_j(\sum_{q \in N(p)} w_j(u(p,q))F(q))$, where $u(p,q)$ represents the local polar coordinate system around p, $w_j(u)$ is a function to model the effect of the patch operator $D = \{D_j\}_{j \in J}$. By definition $D_j = \sum_{q \in N(p)} w_j(u(p,q))F(q)$. Later, AnisotropicCNN [1] and MoNet [14] further explore this framework by improving the choice for u and w_j. MoNet [14] can be understood as using mixtures of Gaussians as convolutional filters. We offer an alternative viewpoint. Instead of finding local parametrizations of the manifold, we view it as an embedded submanifold in $\mathbb{R}^n$ and design filters, which are more efficient for point clouds processing, in the ambient Euclidean space.

Spectral Methods: We know that Fourier transform takes convolutions to multiplications. Explicitly, If $f, g : \mathbb{R}^n \to \mathbb{C}$, then $\widehat{f * g} = \hat{f} \cdot \hat{g}$. Therefore, formally we have $f * g = (\hat{f} \cdot \hat{g})^{\vee}$,[2] which can be used as a definition for convolution on non-Euclidean domains where we know how to take Fourier transform.

Although we do not have Fourier theory on a general space without any equivariant structure, on Riemannian manifolds or graphs there are generalized notions of Laplacian operator. Taking Fourier transform in $\mathbb{R}^n$ could be formally viewed as finding the coefficients in the expansion of the eigenfunctions of the Laplacian operator. To be more precise, recall that

$$\hat{f}(\xi) = \int_{\mathbb{R}^n} f(x) \exp\left(-2\pi i x \cdot \xi\right) d\xi, \tag{1}$$

and $\{\exp\left(-2\pi i x \cdot \xi\right)\}_{\xi \in \mathbb{R}^n}$ are eigen-functions for the Laplacian operator $\Delta = \sum_{i=1}^{n} \frac{\partial}{\partial x_i}$. Therefore, if U is the matrix whose columns are eigenvectors of the graph Laplacian matrix and Λ is the vector of corresponding eigenvalues, for F, g two functions on the vertices of the graph, then $F * g = U(U^T F \odot U^T g)$, where U^T is the transpose of U and $\odot$ is the Hadamard product of two matrices. Since being compactly supported in the spatial domain translates into being smooth in the spectral domain, it is natural to choose $U^T g$ to be smooth functions in Λ. For instance, ChebNet [5] uses Chebyshev polynomials that reduces the complexity of filtering, and CayleyNet [11] uses Cayley polynomials which allows efficient computations for localized filters in restricted frequency bands of interest.

---

[2] If h is a function, then $\hat{h}$ is the Fourier transform, and $h^{\vee}$ is its inverse Fourier transform.

When analyzing different graphs or shapes, spectral methods lack abstract motivations, because different spectral domains cannot be canonically identified. SyncSpecCNN [21] proposes a weight sharing scheme to align spectral domains using functional maps. Viewing point clouds as data embedded in $\mathbb{R}^3$, SpiderCNN can learn representations that are robust to spatial rigid transformations with the aid of data augmentation.

3   SpiderConv

In this section, we describe SpiderConv, which is the fundamental building block for SpiderCNN. First, we discuss how to define a convolutional layer in neural network when the inputs are features on point sets in $\mathbb{R}^n$. Next we introduce a special family of convolutional filters. Finally, we give details for the implementation of SpiderConv with multiple channels and the approximations used for computational speedup.

3.1   Convolution on Point Sets in $\mathbb{R}^n$

An image is a function on regular grids $F : \mathbb{Z}^2 \to \mathbb{R}$. Let W be a $(2m + 1) \times (2m + 1)$ filter matrix, where m is a positive integer, the convolution in classical CNNs is

$$F * W(i, j) = \sum_{s=-m}^{m} \sum_{t=-m}^{m} F(i - s, j - t)W(s, t), \tag{2}$$

which is the discretization of the following integration

$$f * g(p) = \int_{\mathbb{R}^2} f(q)g(p - q)dq, \tag{3}$$

if $f, g : \mathbb{R}^2 \to \mathbb{R}$, such that $f(i, j) = F(i, j)$ for $(i, j) \in \mathbb{Z}^2$ and $g(s, t) = W(s, t)$ for $s, t \in \{-m, -m + 1, ..., m - 1, m\}$ and g is supported in $[-m, m] \times [-m, m]$.

Now suppose that F is a function on a set of points P in $\mathbb{R}^n$. Let $g : \mathbb{R}^n \to \mathbb{R}$ be a filter supported in a ball centered at the origin of radius r. It is natural to define SpiderConv with input F and filter g to be the following:

$$F * g(p) = \sum_{q \in P, \|q-p\| \le r} F(q)g(p - q). \tag{4}$$

Note that when $P = \mathbb{Z}^2$ is a regular grid, Eq. 4 reduces to Eq. 3. Thus the classical convolution can be seen as a special case of SpiderConv. Please see Fig. 1 for an intuitive illustration.

In SpiderConv, the filters are chosen from a parametrized family $\{g_w\}$ (See Fig. 2 for a concrete example) which is piece-wise differentiable in w. During the training of SpiderCNN, the parameters $w \in \mathbb{R}^d$ are optimized through SGD algorithm, and the gradients are computed through the formula $\frac{\partial}{\partial w_i} F * g_w(p) = \sum_{q \in P, \|q-p\| \le r} F(q) \frac{\partial}{\partial w_i} g_w(p - q)$, where w_i is the i-th component of w.

3.2 A Special Family of Filters $\{g_w\}$

A natural choice is to take g_w to be a multilayer perceptron (MLP) network, because theoretically an MLP with one hidden layer can approximate an arbitrary continuous function [8]. However, in practice we find that MLPs do not work well. One possible reason is that MLP fails to account for the geometric prior of 3D point clouds. Another possible reason is that to ensure sufficient expressiveness the number of parameters in a MLP needs to be sufficiently large, which makes the optimization problem difficult.
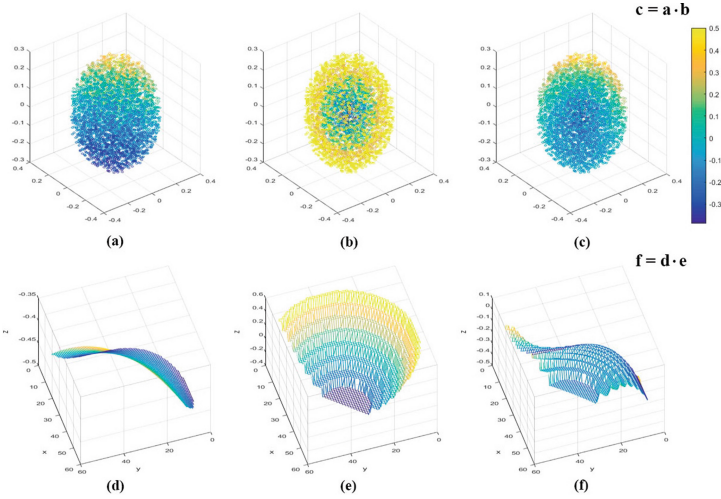
Fig. 2. Visualization of a filter in the family $\{g_w\}$. **(a)** is the scatter plot (color represents the value of the function) of $g^{Taylor}(x, y, z) = 1+x+y+z+xy+xz+yz+xyz$. **(b)** is the scatter plot of $g^{step}(x, y, z) = \frac{i+1}{8}$ if $\frac{i}{8} \leq \sqrt{x^2 + y^2 + z^2} < \frac{i+1}{8}$, when $i = 0, 1, ..., 7$. **(c)** is the scatter plot of the product $g = g^{Taylor} \cdot g^{step}$. In the second row, **(d) (e) (f)** are the graphs of g^{Taylor}, g^{step} and g respectively when restricting their domain to the plane $z = 0$ (the Z-axis represents the value of the function). (Color figure online)

To address the above issues, we propose the following family of filters $\{g_w\}$:

$$g_w(x, y, z) = g_{w^S}^{Step}(x, y, z) \cdot g_{w^T}^{Taylor}(x, y, z), \tag{5}$$

with $w = (w^S, w^T)$ is the concatenation of two vectors $w^S = (w_i^S)$ and $w^T = (w_i^T)$,[3] where

$$g_{w^S}^{Step}(x, y, z) = w_i^S \text{ if } r_i \leq \sqrt{x^2 + y^2 + z^2} < r_{i+1}, \tag{6}$$

---

[3] Here we use the notation $v = (v_i)$ to represent that $v_i \in \mathbb{R}$ is the i-th component of the vector v.

with $r_0 = 0 < r_1 < r_2 ... < r_N$, and

$$
\begin{aligned}
g_{w^T}^{Taylor}(x, y, z) = {} & w_0^T + w_1^T x + w_2^T y + w_3^T z + w_4^T xy + w_5^T yz + w_6^T xz + w_7^T x^2 \\
& + w_8^T y^2 + w_9^T z^2 + w_{10}^T xy^2 + w_{11}^T x^2 y + w_{12}^T y^2 z + w_{13}^T yz^2 \qquad (7) \\
& + w_{14}^T x^2 z + w_{15}^T xz^2 + w_{16}^T xyz + w_{17}^T x^3 + w_{18}^T y^3 + w_{19}^T z^3.
\end{aligned}
$$

The first component $g_{w^S}^{Step}$ is a step function in the radius variable of the local polar coordinates around a point. It encodes the local geodesic information, which is a critical quantity to describe the coarse local shape. Moreover, step functions are relatively easy to optimize using SGD.

The order-3 Taylor term $g_{w^T}^{Taylor}$ further enriches the complexity of the filters, complementary to $g_{w^S}^{Step}$ since it also captures the variations of the angular component. Let us be more precise about the reason for choosing Taylor expansions here from the perspective of interpolation. We can think of the classical 2D convolutional filters as a family of functions interpolating given values at 9 points $\{(i, j)\}_{i,j \in \{-1,0,1\}}$, and the 9 values serve as the parametrization of such a family. Analogously, in 3D consider the vertices of a cube $\{(i, j, k)\}_{i,j,k=0,1}$, assume that at the vertex (i, j, k) the value $a_{i,j,k}$ is assigned. The trilinear interpolation algorithm gives us a function of the form

$$
f_{w^T}(x, y, z) = w_0^T + w_1^T x + w_2^T y + w_3^T z + w_4^T xy + w_5^T yz + w_6^T xz + w_{16}^T xyz, \quad (8)
$$

where w_i^T's are linear functions in c_{ijk}. Therefore f_{w^T} is a special form of $g_{w^T}^{Taylor}$, and by varying w^T, the family $\{g_{w^T}^{Taylor}\}$ can interpolate arbitrary values at the vertexes of a cube and capture rich spatial information.

3.3 Implementation

The following approximations are used based on the uniform sampling process constructing the point clouds:

1. K-nearest neighbors are used to measure the locality instead of the radius, so the summation in Eq. 4 is over the K-nearest neighbors of p.
2. The step function $g_{w^T}^{Step}$ is approximated by a permutation. Explicitly, let X be the $1 \times K$ matrix indexed by the K-nearest neighbors of p including p, and $X(1, i)$ is a feature at the i-th K-nearest neighbors of p. Then $F * g_{w^T}^{Step}(p)$ is approximated by Xw, where w is a $K \times 1$ matrix with $w(i, 1)$ corresponds to w_i^T in Eq. 6.

Later in the article, we omit the parameters w, w^S and w^T, and just write $g = g^{Step} \cdot g^{Taylor}$ to simplify our notations.

The input to SpiderConv is a c_1-dimensional feature on a point cloud P, and is represented as $F = (F_1, F_2, ..., F_{c_1})$ where $F_v : P \to \mathbb{R}$. The output of a SpiderConv is a c_2-dimensional feature on the point cloud $\tilde{F} = (\tilde{F}_1, \tilde{F}_2, ..., \tilde{F}_{c_2})$ where $\tilde{F}_i : P \to \mathbb{R}$. Let p be a point in the point cloud, and $q_1, q_2, ..., q_K$ are its K-nearest neighbors in order. Assume $g_{i,v,t}^{Step}(p - q_j) = w_j^{(i,v,t)}$, where

$t = 1, 2, ..., b$ and $v = 1, 2, ..., c_1$ and $i = 1, 2, ...c_2$. Then a SpiderConv with c_1 in-channels, c_2 out-channels and b Taylor terms is defined via the formula: $\tilde{F}_i(p) = \sum_{v=1}^{c_1} \sum_{j=1}^{K} g_i(p - q_j) F_v(q_j)$, where $g_i(p - q_j) = \sum_{t=1}^{b} g_t^{Taylor}(p - q_j) w_j^{(i,v,t)}$, and g_t^{Taylor} is in the parameterized family $\{g_{w^T}^{Taylor}\}$ for $t = 1, 2, ..., b$.

4 Experiments

We analyze and evaluate SpiderCNN on 3D point clouds classification and segmentation. We empirically examine the key hyper-parameters of a 3-layer SpiderCNN, and compare our models with the state-of-the-art methods.

Implementation Details: All models are prototyped with Tensorflow 1.3 on 1080Ti GPU and trained using the Adam optimizer with a learning rate of 10^{-3}. A dropout rate of 0.5 is used with the fully connected layers. Batch normalization is used at the end of each SpiderConv with decay set to 0.5. On a GTX 1080Ti, the forward-pass time of a SpiderConv layer (batch size 8) with in-channel 64 and out-channel 64 is 7.50 ms. For the 4-layer SpiderCNN (batch size 8), the total forward-pass time is 71.68 ms.

4.1 Classification on ModelNet40

ModelNet40 [4] contains 12,311 CAD models which belong to 40 different categories with 9,843 used for training and 2,468 for testing. We use the source code for PointNet [15] to sample 1,024 points uniformly and compute the normal vectors from the mesh models. The same data augmentation strategy as [15] is applied: the point cloud is randomly rotated along the up-axis and the position of each point is jittered by a Gaussian noise with zero mean and 0.02 standard deviation. The batch size is 32 for all the experiments in Sect. 4.1. We use the (x, y, z)-coordinates and normal vectors of the 1,024 points as the input for SpiderCNN for the experiments on ModelNet40 unless otherwise specified.
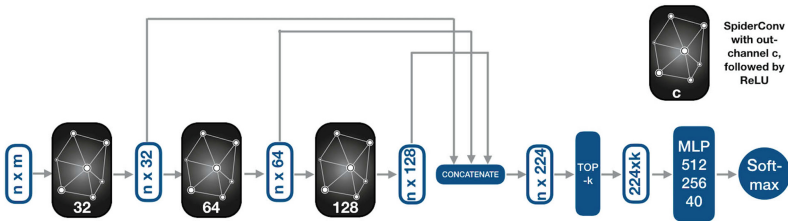

Fig. 3. The architecture of a 3-layer SpiderCNN used in ModelNet40 classification.

3-Layer SpiderCNN: Figure 3 illustrates a SpiderCNN with 3 layers of SpiderConvs each with 3 Taylor terms, and the respective out-channels for each layer

being 32, 64, 128.[4] ReLU activation function is used here. The output features of the three SpiderConvs are concatenated in the end. Top-k pooling among all the points is used to extract global features.
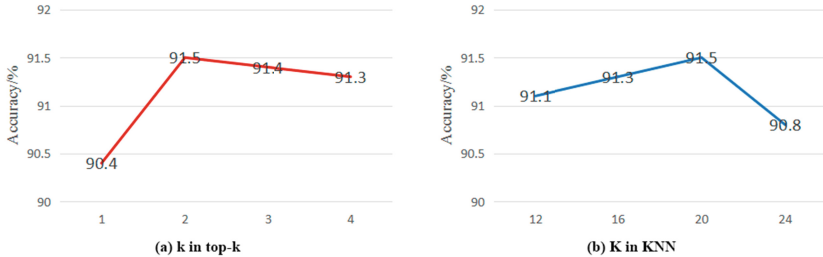

Fig. 4. On ModelNet40 **(a)** shows the effect of number of pooled features on the accuracy of 3-layer SpiderCNN with 20-nearest neighbors. **(b)** shows the effect of nearest neighbors in SpiderConv on the accuracy of 3-layer SpiderCNN with top-2 pooling.

Two important hyperparameters in SpiderCNN are studied: the number of nearest neighbors K chosen in SpiderConv, and the number of pooled features k after the concatenation. The results are summarized in Fig. 4. The number of nearest-neighbors K is analogous to size of the filter in the usual convolution. We see that 20 is the optimal choice among 12, 16, 20, and 24-nearest neighbors. In Fig. 5 we provide visualization for top-2 pooling. The points that contribute to the top-2 pooling features are plotted. We see that similar to PointNet, Spider CNN picks up representative critical points.

SpiderCNN + PointNet: We train a 3-layer SpiderCNN (top-2 pooling and 20-nearest neighbors) and PointNet with only (x, y, z)-coordinates as input to predict the classical robust local geometric descriptor FPFH [19] on point clouds in ModelNet40. The training loss of SpiderCNN is only $\frac{1}{4}$ that of PointNet's. As a result, we believe that a 3-layer SpiderCNN and PointNet are complementary to each other, for SpiderCNN is good at learning local geometric features and Point-Net is good at capturing global features. By concatenating the 128 dimensional features from PointNet with the 128 dimensional features from SpiderCNN, we improve the classification accuracy to 92.2%.

4-Layer SpiderCNN: Experiments show that 1-layer SpiderCNN with a SpiderConv of 32 channels can achieve classification accuracy 85.5%, and the performance of SpiderCNN improves with the increasing number of layers of SpiderConv. A 4-layer SpiderCNN consists of SpiderConv with out-channels 32, 64, 128, and 258. Feature concatenation, 20-nearest neighbors and top-2 pooling are used. To prevent overfitting, while training we apply the data augmentation

---

[4] See Sect. 3.3 for the definition of a SpiderConv with c_1 in-channels, c_2 out-channels and b Taylor terms.
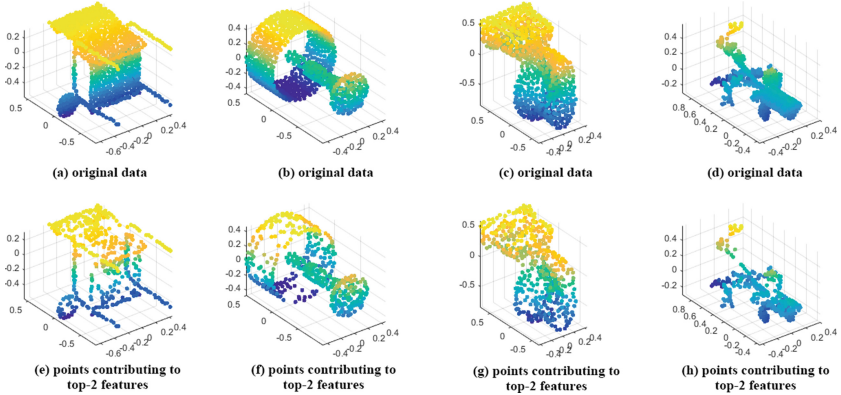
Fig. 5. Visualization of the effect of top-k pooling. Edge points and points with non-zero curvature are preserved after pooling. **(a)**, **(b)**, **(c)**, **(d)** are the original input point clouds. **(e)**, **(f)**, **(g)**, **(h)** are points contributing to features extracted via top-2 pooling.

method DP (random input dropout) introduced in [17]. Table 1 shows a comparison between SpiderCNN and other models. The 4-layer SpiderCNN achieves accuracy of 92.4% which improves over the best reported result of models with input 1024 points and normals. For 5 runs, the mean accuracy of a 4-layer SpiderCNN is 92.0%.

Table 1. Classification accuracy of SpiderCNN and other models on ModelNet40.

Method	Input	Accuracy
Subvolume [16]	Voxels	89.2
VRN Single [2]	Voxels	91.3
OctNet [18]	Hybrid grid octree	86.5
ECC [20]	GWe compute three intrinsicraphs	87.4
Kd-Network [9] (depth 15)	1024 points	91.8
PointNet [15]	1024 points	89.2
PointNet++ [17]	5000 points+normal	91.9
SpiderCNN + PointNet	1024 points+normal	92.2
SpiderCNN (4-layer)	1024 points+normal	**92.4**

Ablative Study: Compared to max-pooling, top-2 pooling enables the model to learn richer geometric information. For example, in Fig. 6, we see top-2 pooling preserves more points where the curvature is non-zero. Using max-pooling, the classification accuracy is 92.0% for a 4-layer SpiderCNN, and is 90.4% for a

3-layer SpiderCNN. In comparison, using top-2 pooling, the accuracy is 92.4% for a 4-layer SpiderCNN, and is 91.5% for a 3-layer SpiderCNN.
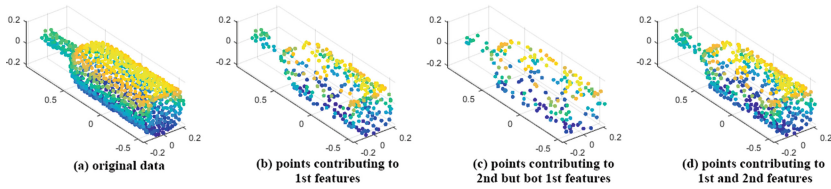

Fig. 6. Top-2 pooling learns rich features and fine geometric details.

MLP filters do not perform as well in our setting. The accuracy of a 3-layer SpiderCNN is 71.3% with $g_w = \mathrm{MLP}(16, 1)$, and is 72.8% with $g_w = \mathrm{MLP}(16, 32, 1)$.

Without normals, the accuracy of a 4-layer SpiderCNN using only the 1,024 points is 90.5%. Using normals extracted from the 1,024 input points via orthogonal distance regression, the accuracy of a 4-layer SpiderCNN is 91.8%.
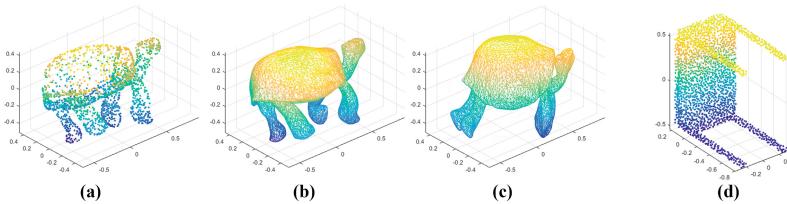

Fig. 7. (b) and **(c)** are shapes in SHREC15. **(d)** is a shape in ModelNet40. **(a)** is the point cloud sampled from **(b)**.

4.2   Classification on SHREC15

SHREC15 is a dataset for non-rigid 3D shape retrieval. It consists of 1,200 watertight triangle meshes divided in 50 categories. On average 10,000 vertices are stored in one mesh model. Comparing to ModelNet40, SHREC15 contains more complicated local geometry and non-rigid deformation of one object. See Fig. 7 for a comparison. 1,192 meshes are used with 895 for training and 297 for testing. We compute three intrinsic shape descriptors (Heat Kernel Signature, Wave Kernel Signature and Fast Point Feature Histograms) for deformable shape analysis from the mesh models. 1,024 points are sampled uniformly randomly from the vertices of a mesh model, and the (x, y, z)-coordinates are used as the input for SpiderCNN, PointNet and PointNet++. We use SVM with linear kernel when the inputs are classical shape descriptors. Table 2 summarizes the results. We see that SpiderCNN outperforms the other methods.

Table 2. Classification accuracy on SHEREC15.

Method	Input	Accuracy
SVM + HKS	Features	56.9
SVM + WKS	Features	87.5
SVM + FPFH	Features	80.8
PointNet	Points	69.4
PointNet++ [17]	Points	60.2
PointNet++ (our implementation)	Points	94.1
SpiderCNN (4-layer)	Points	**95.8**

4.3   Segmentation on ShapeNet Parts

ShapeNet Parts consists of 16,880 models from 16 shape categories and 50 different parts in total, with a 14,006 training and 2,874 testing split. Each part is annotated with 2 to 6 parts. The mIoU is used as the evaluation metric, computed by taking the average of all part classes. A 4-layer SpiderCNN whose architecture is shown in Fig. 8 is trained with batch of 16. We use points with their normal vectors as the input and assume that the category labels are known. The results are summarized in Table 3. For 4 runs, the mean of mean IoU of SpiderCNN is 85.24. We see that SpiderCNN achieves competitive results despite a relatively simple network architecture (Fig. 9).
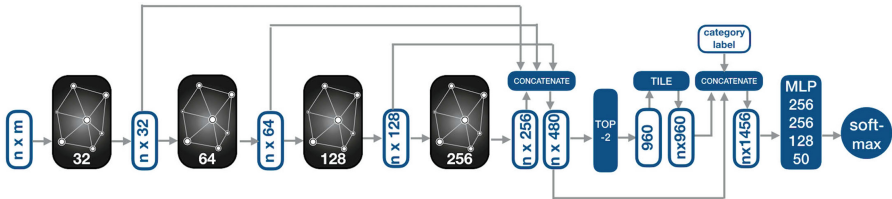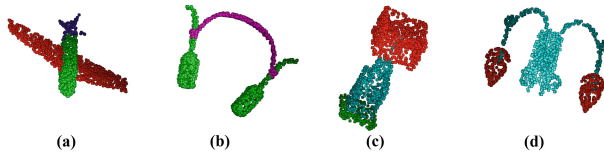

Fig. 8. The SpiderCNN architecture used in the ShapeNet Part segmentation task.

Fig. 9. Some examples of the segmentation results of SpiderCNN on ShapeNet Part.

Table 3. Segmentation results on ShapeNet Part dataset. Mean IoU and IoU for each categories are reported.

	Mean	Aero	Bag	Cap	Car	Chair	Ear ph	Guitar	Knife	Lamp	Laptop	Motor	Mug	Pistol	Rocket board	Skate	Table
PN [15]	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
PN++[17]	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
Kd-Net [9]	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
SSCNN [21]	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
SpiderCNN	**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**

5   Analysis

In this section, we conduct additional analysis and evaluations on the robustness of SpiderCNN, and provide visualization for some of the typical learned filters from the first layer of SpiderCNN.
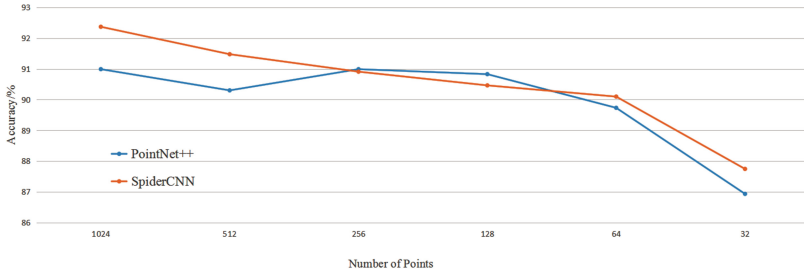
Fig. 10. Classification accuracy of SpiderCNN and PointNet++ with different number of input points on ModelNet40.

Robustness: We study the effect of missing points on SpiderCNN. Following the setting for experiments in Sect. 4.1, we train a 4-layer SpiderCNN and Point-Net++ with 512, 248, 128, 64 and 32 points and their normals as input. The results are summarized in Fig. 10. We see that even with only 32 points, Spider-CNN obtains 87.7% accuracy.

Visualization: In Fig. 11, we scatter plot the convolutional filters $g_w(x, y, z)$ learned in the first layer of SpiderCNN and the color of a point represents the value of g_w at the point.

In Fig. 12 we choose a plane passing through the origin, and project the points that lie on one side of the plane of the scatter graph onto the plane. We see some similar patterns that appear in 2D image filters. The visualization gives some hints about the geometric features that the convolutional filters in SpiderCNN learn. For example, the first row in Fig. 12 corresponds to 2D image filters that can capture boundary information.
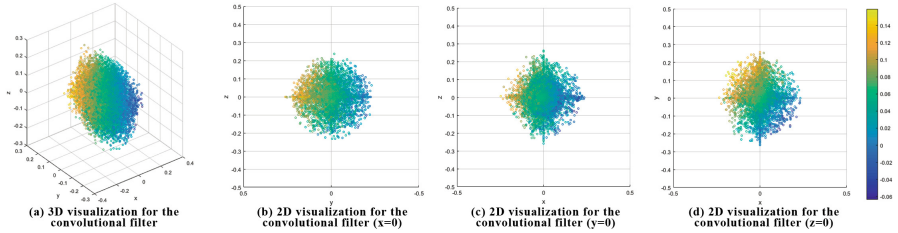
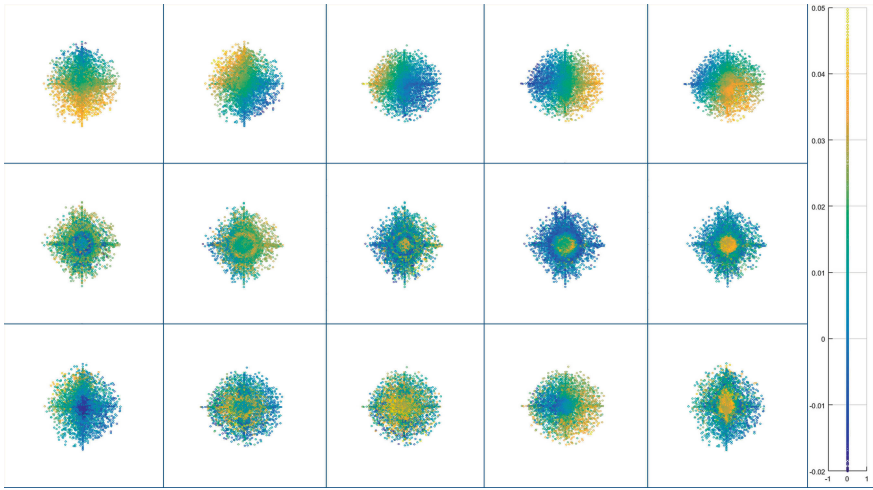**Fig. 11.** Visualization of for the convolutional filters learned in the first layer of SpiderCNN.

Fig. 12. Visualization for the convolutional filters learned in the first layer of Spider-CNN. The 3D filters are shown as scatter plots projected on to the planes $x = 0$ or $y = 0$ or $z = 0$.

6   Conclusions

A new convolutional neural network SpiderCNN that can directly process 3D point clouds with parameterized convolutional filters is proposed. More complex network architectures and more applications of SpiderCNN can be explored.

References

1. Boscaini, D., Masci, J., Rodolà, E., Bronstein, M.: Learning shape correspondence with anisotropic convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 3189–3197 (2016)
2. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: Generative and discriminative voxel modeling with convolutional neural networks. arXiv preprint arXiv:1608.04236 (2016)
3. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond Euclidean data. IEEE Signal Process. Mag. **34**(4), 18–42 (2017)
4. Chang, A.X., et al.: Shapenet: an information-rich 3D model repository. arXiv preprint arXiv:1512.03012 (2015)
5. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, pp. 3844–3852 (2016)
6. Engelcke, M., Rao, D., Wang, D.Z., Tong, C.H., Posner, I.: Vote3deep: fast object detection in 3D point clouds using efficient convolutional neural networks. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 1355–1361. IEEE (2017)
7. Graham, B., Engelcke, M., van der Maaten, L.: 3D semantic segmentation with submanifold sparse convolutional networks. arXiv preprint arXiv:1711.10275 (2017)
8. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Netw. **4**(2), 251–257 (1991)
9. Klokov, R., Lempitsky, V.: Escape from cells: deep Kd-networks for the recognition of 3D point cloud models. In: IEEE International Conference on Computer Vision (ICCV), pp. 863–872. IEEE (2017)
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
11. Levie, R., Monti, F., Bresson, X., Bronstein, M.M.: Cayleynets: graph convolutional neural networks with complex rational spectral filters. arXiv preprint arXiv:1705.07664 (2017)
12. Masci, J., Boscaini, D., Bronstein, M., Vandergheynst, P.: Geodesic convolutional neural networks on Riemannian manifolds. In: Proceedings of the IEEE International Conference on Computer Vision Workshops, pp. 37–45 (2015)
13. Maturana, D., Scherer, S.: Voxnet: a 3D convolutional neural network for real-time object recognition. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 922–928. IEEE (2015)
14. Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model CNNs. In: Proceedings of CVPR, vol. 1, p. 3 (2017)
15. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: deep learning on point sets for 3D classification and segmentation. In: Proceedings of the Computer Vision and Pattern Recognition (CVPR), vol. 1, no. 2, p. 4. IEEE (2017)
16. Qi, C.R., Su, H., Nießner, M., Dai, A., Yan, M., Guibas, L.J.: Volumetric and multi-view CNNs for object classification on 3D data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5648–5656 (2016)
17. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: deep hierarchical feature learning on point sets in a metric space. In: Advances in Neural Information Processing Systems, pp. 5105–5114 (2017)

18. Riegler, G., Ulusoy, A.O., Geiger, A.: Octnet: learning deep 3D representations at high resolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, vol. 3 (2017)
19. Rusu, R.B., Blodow, N., Beetz, M.: Fast point feature histograms (FPFH) for 3D registration. In: IEEE International Conference on Robotics and Automation, ICRA 2009, pp. 3212–3217. IEEE (2009)
20. Simonovsky, M., Komodakis, N.: Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: Proceedings of CVPR (2017)
21. Yi, L., Su, H., Guo, X., Guibas, L.: Syncspeccnn: synchronized spectral CNN for 3D shape segmentation. In: Computer Vision and Pattern Recognition (CVPR) (2017)
22. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J.: Deep sets. In: Advances in Neural Information Processing Systems, pp. 3394–3404 (2017)
23. Zhang, Z.: Microsoft kinect sensor and its effect. IEEE Multimed. **19**(2), 4–10 (2012)