

Interpolated Convolutional Networks for 3D Point Cloud Understanding

Jiageng Mao Xiaogang Wang Hongsheng Li

CUHK-SenseTime Joint Laboratory, The Chinese University of Hong Kong

Fmaoj i ageng@l i nk, xgwang@ee, hsl i @eeG. cuhk. edu. hk

Abstract

Point cloud is an important type of 3D representation. However, directly applying convolutions on point clouds is challenging due to the sparse, irregular and unordered data structure. In this paper, we propose a novel Interpolated Convolution operation, InterpConv, to tackle the point cloud feature learning and understanding problem. The key idea is to utilize a set of discrete kernel weights and interpolate point features to neighboring kernel-weight coordinates by an interpolation function for convolution. A normalization term is introduced to handle neighborhoods of different sparsity levels. Our InterpConv is shown to be permutation and sparsity invariant, and can directly handle irregular inputs. We further design Interpolated Convolutional Neural Networks (InterpCNNs) based on InterpConv layers to handle point cloud recognition tasks including shape classification, object part segmentation and indoor scene semantic parsing. Experiments show that the networks can capture both fine-grained local structures and global shape context information effectively. The proposed approach achieves state-of-the-art performance on public benchmarks including ModelNet40, ShapeNet Parts and S3DIS.

1. Introduction

Point cloud is an important data format obtained by 3D sensors and has shown extensive usage in many real-world tasks including autonomous driving [7], robotics [32], etc. Efficient learning from point cloud data remains a challenge to the research community, given the fact that point clouds are usually irregular, unordered and sparse.

In view of the great success of convolutional neural networks (CNNs) on 2D images, many endeavors have been made to adapt the convolution operation to 3D point clouds. Currently there are two main approaches to tackle this problem. The first type of attempts [24] is to directly rasterize irregular point clouds into regular voxel grids, and adopts standard 3D convolutions to learn shape features. However, the transformation of irregular inputs leads to a loss of geometric information, and convolutions on dense voxel grids lead to heavy computational burden. Other ap-

(a) 3D convolutions with rasterization

(b) Graph neural networks

(c) InterpConv with trilinear interpolation

(d) InterpConv with Gaussian interpolation

Figure 1. Illustration of different types of convolutions on point clouds. Red points denote point clouds and green points denote spatially-discrete kernel weights. Green lines in (b) denote continuous convolutional kernels. Our InterpConv directly takes irregular point clouds as inputs and interpolates point features to the neighboring kernel weights by an interpolation function.

proaches [43, 45, 28, 49, 34, 20, 46, 39] build local graphs in the neighborhood of each point in the Euclidean or feature space, then a continuous convolutional kernel is applied on each edge of the graph to learn geometric features. Continuous kernels are commonly modeled by Multi-layer Perceptrons (MLPs). These graph-based methods are able to directly process irregular data structure but have some drawbacks. The construction of local graphs is not sparsity invariant. Namely, different point cloud densities sampled from the same object surface lead to different neighborhood selections, thus it may produce various graph construction results. Besides, compared with discrete convolutions, uti-

lizing MLPs to learn an arbitrary continuous function does not work well in practice [49].

In this paper, we propose a novel Interpolated Convolution operation (InterpConv) to address the existing problems in graph and 3D convolutional neural networks. Key to our approach is the use of discrete convolutional kernels and an interpolation function to explicitly measure geometric relations between input point clouds and kernel-weight coordinates. Unlike 3D convolutions which have to transform inputs into regular grids, our InterpConv directly takes irregular point clouds as inputs. Each $n \times n \times c$ convolutional kernel is split into n^2 kernel weights, each of which has a $1 \times c$ weight vector and its own coordinate p relative to the kernel center. The center of discrete convolutional kernels can be placed at any location in the 3D space and then the kernel-weight absolute coordinates can be determined for each kernel. Input points are interpolated to neighboring kernel-weight coordinates by an interpolation function. To guarantee InterpConv to be sparsity invariant, normalization on points is adopted in the neighborhood of each kernel weight vector. Finally, weighted convolutions can be calculated between kernel weight vectors and point cloud features that are associated to them. With spatially-discrete convolutional kernel weights and an explicitly defined interpolation function, our approach performs better than graph-based methods, which use continuous functions as convolutional kernels and implicitly learn geometric relations. See Figure 1 for illustration.

We further propose Interpolated Convolutional Neural Networks (InterpCNNs) based on InterpConvs. The classification network is composed of multi-layer and multi-receptive-field InterpConv blocks, which can capture both fine-grained geometric structures and context information. The segmentation network explores a deeper architecture to predict semantic labels for all input points. We evaluate our networks on several benchmark datasets, including ModelNet40 [5], ShapeNet Parts [50] and S3DIS [1]. Experiments show that our approach achieves state-of-the-art performances on those datasets.

The key contributions of our work are as follows:

- We propose a novel Interpolated Convolution operation (InterpConv) to effectively deal with point cloud recognition problems. Such an operation is permutation and sparsity invariant, and can directly handle irregular point clouds;
- We design Interpolated Convolutional Neural Networks based on InterpConvs. The networks perform better than Graph Neural Networks (GNNs) and 3D Convolutional Neural Networks (3D ConvNets) on point cloud recognition and segmentation problems.

2. Related Work

Our approach is closely related to other deep learning methods on point clouds. We introduce literatures on point

cloud feature learning by regular grids and irregular inputs.

Learning from point clouds by regular grids. When facing irregular point clouds as inputs, an intuitive way is to transform this irregular data structure into regular grids. Some approaches [15, 37, 40] transform 3D objects or point clouds into 2D regular grids, namely, images by multi-view projection, then 2D CNNs are utilized to learn from these images. Those methods work well owing to the great success of 2D CNNs on images. However, not all the geometric information is kept during projection, and those approaches are usually inefficient and time-consuming when handling sparse point cloud data.

An alternative approach is to rasterize point clouds into 3D regular grids. VoxNet [24] transforms original point cloud data into occupancy grids, which store binary values to indicate whether the spaces are occupied. Then a 3D CNN is applied to learn from these voxel grids. The rasterization process loses some fine-grained geometric features, and 3D convolutions are both time and memory consuming. OctNet [30] exploits the sparsity of voxel grids and uses unbalanced octrees to hierarchically partition the space, which saves much memory. Some other efforts [21, 42, 29] have also been made to ease the computational burden but still cannot solve the loss of geometric information during rasterization.

Compared with the above-mentioned approaches, our approach directly takes irregular point clouds as inputs without rasterization, which is time-saving and accurate.

Learning from point clouds by irregular inputs. Recently there are many works trying to directly process irregular point cloud data. Pioneering work PointNet [25] utilizes shared MLPs and a maxpooling layer, which is permutation invariant, to tackle unordered inputs and learn a global representation. PointNet++ [27] exploits local structures by grouping and sampling point clouds, then a PointNet is applied in each group to aggregate local features. However, how to effectively partition and select point clouds remain a challenge. Many approaches [16, 14, 12, 19] explore new grouping and sampling strategies. In [23, 8], new modules are added to original PointNet++ in order to gain a better performance.

Graph Neural Networks (GNNs) [33] have been widely used to deal with irregular data structure. There are also a bunch of works trying to apply GNNs to solve point cloud processing problem. Those approaches [43, 45, 28, 22, 49, 34, 20] usually build local graphs in the neighborhood of the Euclidean or feature space, utilize MLPs as continuous convolutional kernel functions, and aggregate local features by weighted sum or pooling from neighborhood to center. DGCNN [45] proposes an EdgeConv operation which concatenates central and neighboring point features and learns new features by MLP and maxpooling. 3DGNN [28] applies gated graph neural networks [22] on semantic segmentation task. SpiderCNN [49] defines a continuous kernel function as a product of step function and a Taylor polyno-

mial. KCNet [34] proposes a kernel correlation and graph pooling layer to exploit local structures. PointCNN [20] applies γ -transform operator on local graphs.

GNN-based methods still have some problems. First, the graph construction process based on K-nearest neighbors (KNN) is sensitive to point cloud density. Second, using MLPs to directly learn from point coordinates is inefficient, as it ignores some explicitly defined geometric relations. Different from those methods, our approach is not sensitive to point cloud density due to the proposed normalization term, and geometric relations between discrete kernel weights and point clouds are explicitly defined by an interpolation function.

3. Method

In this section, we first revisit convolutions on different types of point sets. We then introduce our proposed Interpolated Convolution operation (InterpConv) and key elements of the InterpConv algorithm. Finally, we show details for our network architectures for 3D object recognition and semantic segmentation.

3.1. Convolutions on Point Sets

Standard 2D and 3D convolutions have achieved great success in handling regularly-arranged data such as images and voxel grids. When it comes to sparse and irregular point sets such as 3D point clouds, multiple variants of convolutions have been proposed. In this section, we review those convolutions to motivate the design of our InterpConv operation.

Considering a standard 3D convolution, let 3D voxel grids or features be denoted by $F : Z^3 \rightarrow R^c$, and the convolutional kernel weights W be a series of $1 \times c$ weight vectors, where c is the number of channels. Standard convolution at location \hat{p} can be formulated as

$$F \cdot W(\hat{p}) = \sum_p F(\hat{p} + p) \cdot W(p), \quad (1)$$

where $\cdot = \{p \in Z^3 : (-n, -n, -n), \dots, (n, n, n)\}$ is the set of kernel weight vectors' coordinates relative to the kernel center. The kernel size is assumed to be $2n + 1$, and \cdot denotes dot production between two vectors.

When it comes to irregular inputs, points are no longer regularly-arranged and distances between points become irregular. Some approaches [43, 46] adopt a continuous weight function $W(p)$, which takes relative coordinates p of neighboring points $\hat{p} + p$ to the central point \hat{p} as inputs, to predict the convolutional weights. The continuous function $W(p)$ is no longer a $1 \times c$ weight vector but a mapping $R^3 \rightarrow R^c$ commonly implemented by MLPs. Then the continuous convolution can be formulated as

$$F \cdot W(\hat{p}) = \sum_p F(\hat{p} + p) \cdot W(p). \quad (2)$$

It is worth noting that applying graph neural networks [45, 49, 34] to handle point clouds essentially shares the same idea with continuous convolutions.

Replacing discrete kernel weights $W(p)$ with continuous functions $W(p)$ remains some problems. Simply learning continuous functions by MLPs cannot always work in practice [49]. The predicted parameters might be too many, and the learning process is inefficient and sometimes unstable. Knowledge on the great success of discrete kernels in images cannot be transferred to point clouds recognition tasks as well.

3.2. Interpolated Convolution for 3D Point Clouds

In our approach, we adopt the design of discrete convolutional weights while maintaining the characteristics of continuous distances, by decoupling $W(p)$ into two parts: spatially-discrete kernel weights $W(p) \in R^c$ and an interpolation function $T(p, \hat{p})$. We note that a spatially-discrete kernel weight $W(p)$ is a $1 \times c$ vector, which can be initialized and updated during training, and \hat{p} is the relative coordinate of this kernel weight vector to the kernel center. The interpolation function $T(p, \hat{p}) : R^3 \times R^3 \rightarrow R$ takes the coordinate of a kernel weight vector p and a neighboring input point \hat{p} as inputs, and computes a weight by certain interpolation algorithm. Our approach takes every input point in the neighborhood of a kernel weight vector into account. In order to make convolutions sparsity invariant, a density normalization term N_p , which sums the interpolation weights or number of input points in the neighborhood of \hat{p} , is needed for each kernel weight vector $W(p)$. Finally, our InterpConv centered at location \hat{p} can be formulated as

$$F \cdot W(\hat{p}) = \sum_p \frac{1}{N_p} T(p, \hat{p}) F(\hat{p} + p) \cdot W(p). \quad (3)$$

We note that unlike standard convolutions, where kernel weights are regularly-arranged, kernel-weight coordinates p in InterpConv can be set flexibly or even learned during training.

There are three key parts of our proposed InterpConv operation: spatially-discrete kernel weights W , an interpolation function T , and a normalization term N . We first discuss those three parts separately, and then introduce the complete algorithm.

Discrete kernel weights. In 2D convolution [17], one kernel can be represented as an $n \times n \times c$ tensor, where n denotes the kernel size and c denotes the number of channels. In [6, 9], one kernel is split into $n \times n$ weight vectors, each of which is of size $1 \times c$. By doing so, kernel weights no longer have to be regularly-arranged, but can be flexibly placed on 2D grids.

In our approach, we further improve this idea by defining a set of kernel weight vectors for each convolutional kernel in the 3D Euclidean space. Each kernel weight vector $W(p)$ has a 3D coordinate p to store its location relative to

the kernel center, and its weights are stored in a $1 \times c$ vector, which will be initialized and updated during training. The vector coordinate \mathbf{p} can either be fixed or updated during training. To simplify the problem, we fix kernel-weight coordinates in most experiments and organize them as a cube, namely, kernel weight vectors are arranged at $3 \times 3 \times 3$ 3D regular grids if the total number of kernel weight vectors is 27. We note that this is an analogy of standard $3 \times 3 \times 3$ discrete convolutions while kernel weight vectors can theoretically be placed at arbitrary locations in the 3D space.

As we arrange kernel weight vectors as a cube, we define two important hyperparameters: the kernel size $n \times n \times n$ and kernel length l . The coordinate set of spatially-discrete kernel weight vectors can be formulated as

$$\mathbf{p} = (\mathbf{x}, \mathbf{y}, \mathbf{z}) \quad \mathbf{x}, \mathbf{y}, \mathbf{z} = kl, \quad (4)$$

$$\mathbf{k} = \left[\frac{n-1}{2}, \dots, \frac{n-1}{2} \right],$$

where $\mathbf{p} = (\mathbf{x}, \mathbf{y}, \mathbf{z})$. Similar to the definition of kernel size in standard convolutions, kernel size $n \times n \times n = 2^3$ means that there are n kernel weight vectors on each edge of one kernel, and the total number of kernel weight vectors is n^3 . Kernel length $l = R$ is the distance between two adjacent weight vectors. It determines the actual 3D size of a kernel in the Euclidean space and is defined to control the receptive field, from which one convolutional kernel can “see” input point clouds. If l is small, the convolutional kernel is able to capture fine-grained local structures, otherwise it encodes more global shape information.

Interpolation functions. One problem to apply discrete kernels on irregular point clouds is that kernel weight vectors’ spatial locations generally do not align with input points. Naively rasterizing point clouds into regular grids [24, 11] solves part of the problem, but at the cost of losing local structures. In our approach, we solve this problem while keep all fine-grained structures by adopting an interpolation function. Namely, we first find a set of input points near each kernel weight vector, and then interpolate their features to be assigned to the kernel weight vectors for convolution. We propose two interpolation functions: trilinear interpolation and Gaussian interpolation.

Trilinear interpolation is a commonly-used method to approximate the value of an intermediate point in a 3D grid by values of adjacent lattice points. The intermediate point’s value is calculated by a weighted sum of lattice points’ values, and the weights characterize closeness between intermediate and lattice points. In our approach, we adopt the inverse process of trilinear interpolation. Namely, we first compute the weights that lattice points (kernel-weight coordinates) contribute to the intermediate point (input point) and then we inversely assign the input point feature to adjacent kernel-weight coordinates with those weights.

For trilinear interpolation, we find 8 adjacent kernel-weight coordinates \mathbf{p} for each input point \mathbf{p} in the kernel,

and then we normalize input point and kernel weights into a unit-length cube. Finally we compute the trilinear interpolation weights by

$$T_{tr}(\mathbf{p}, \mathbf{p}) = (1 - |\mathbf{x} - \mathbf{x}|)(1 - |\mathbf{y} - \mathbf{y}|)(1 - |\mathbf{z} - \mathbf{z}|), \quad (5)$$

where input point $\mathbf{p} = (\mathbf{x}, \mathbf{y}, \mathbf{z})$ is the relative point coordinate to the kernel center, and kernel-weight coordinate $\mathbf{p} = (\mathbf{x}, \mathbf{y}, \mathbf{z})$. We further note that Eq. (5) is a simplified format for normalized points. One property of trilinear interpolation is self-normalization, namely, all 8 weights which an input point assigns to can sum up to 1.

In Gaussian interpolation, we assign each input point \mathbf{p} to each kernel weight vector at \mathbf{p} with a weight factor calculated by the following Gaussian function,

$$T_G(\mathbf{p}, \mathbf{p}) = e^{-\frac{(\mathbf{x} - \mathbf{x})^2 + (\mathbf{y} - \mathbf{y})^2 + (\mathbf{z} - \mathbf{z})^2}{2}}, \quad (6)$$

where the hyperparameter σ controls the decay rate. To save computation, if a 3D point is 3 away from a weight vector, its assignment coefficient to the vector is directly set to 0 and will not be calculated. It is worth noting that other functions, for example, linear basis functions, can also be adopted as the interpolation function.

Normalization terms. Given the fact that we take all neighboring points of a kernel weight vector into calculation, normalization is necessary to keep convolutions invariant to points density. There are two ways of normalization. We can aggregate and normalize the point features by

$$\mathbf{f}_{\text{aggregate}} = \frac{\sum_{i=1}^N t_i \mathbf{f}_i}{N}, \quad (7)$$

where N is the number of neighboring points, \mathbf{f}_i is the i th point feature, and t_i denotes its interpolation weight. Apart from normalizing according to the number of points, we can also normalize the sum of interpolation weights:

$$\mathbf{f}_{\text{aggregate}} = \frac{\sum_{i=1}^N t_i \mathbf{f}_i}{\sum_{i=1}^N t_i}. \quad (8)$$

We can perform normalization either on each kernel weight vector or on the whole convolutional kernel. We argue that normalization on each kernel weight vector is more accurate, since input points are not uniformly distributed in the whole kernel.

The InterpConv Algorithm. An InterpConv operation takes point cloud coordinates and their features as input, and outputs new point coordinates and features. We note that output point coordinates can be set as the same as the input points, or downsampled from input point clouds. The center of convolutional kernels is placed at each output point coordinate, and kernel-weight coordinates are further determined by the relative coordinates of the weight vectors following Eq. (4). We calculate interpolation weights between kernel weight vectors and adjacent input points, and then

Algorithm 1 The InterpConv Algorithm

Input: point coordinates $p \in \mathbb{R}^3$, point features $f \in \mathbb{R}^c$
Output: output coordinates $\hat{p} \in \mathbb{R}^3$, new features $\hat{f} \in \mathbb{R}^c$
Parameter: c kernels with n weight vectors $w \in \mathbb{R}^c$ and shared weight coordinates $p \in \mathbb{R}^3$ in each kernel

- 1: Sample \hat{p} from p or $\hat{p} = p$
- 2: **for each** \hat{p} **do**
- 3: **for each** p **do**
- 4: **for each** neighboring p , feature f_p **do**
- 5: $t = p - \hat{p}$
- 6: $t = T(p, p)$
- 7: $f_i = f_i + t f_p$
- 8: $f_i = \text{Normalize}(f_i)$
- 9: $F = [f_1, \dots, f_n]$
- 10: **for each** kernel k **do**
- 11: $W_k = [w_1^k, \dots, w_n^k]$
- 12: $v_k = F \cdot W_k$
- 13: $\hat{f} = [v_0, \dots, v_c]$
- 14: **return** \hat{p}, \hat{f}

aggregate feature by the weighted sum of all neighboring point features. The aggregated features are further normalized to keep it sparsity invariant. Finally dot production is applied between the normalized features and kernel weight vectors. A convolutional kernel sums all the results and c kernels constitute a $1 \times c$ new feature vector at the output coordinate. See the InterpConv algorithm for details.

3.3. Network Architectures

In this section, we introduce details for two deep architectures based on our InterpConv approaches. We explore embedding multi-scale context features in the classification network and a deep encoder-decoder architecture in the segmentation network. See Figure 2 for details.

The classification network consists of a series of InterpConv blocks which is mainly composed of three InterpConv layers. In the InterpConv block, the first and last layer are of kernel size $1 \times 1 \times 1$ and the middle layer has a kernel size $3 \times 3 \times 3$. The first InterpConv layer reduces channel dimensions and the last InterpConv layer increases channel dimensions, leaving the middle InterpConv layer with relatively small input and output channels. One Batch-Norm [13] and ReLU [48] layer also follow each InterpConv layer in the block.

Apart from this, we propose the PointInception module to encode multi-scale geometric features. Similar to the Inception module [38] in 2D CNNs, our PointInception module also concatenates multi-branch features. However, we design each branch as one InterpConv block with a different kernel length l . The hyperparameter l determines the distances between adjacent kernel weight vectors in the Euclidean space and controls the receptive field of the convolution. So the PointInception module is able to capture both

fine-grained local structures and shape context information by combining multi-branch outputs. We further explore a deeper model by stacking two PointInception modules.

In the segmentation network, we share the similar spirit as U-Net [31] and build a deep encoder-decoder architecture. We stack multiple $3 \times 3 \times 3$ InterpConv layers in the encoder, and in each layer output points are downsampled. In the first $3 \times 3 \times 3$ InterpConv layer, we set the kernel length l as a small value in order to capture fine-grained geometric structures, which is important in semantic segmentation. We then gradually enlarge the kernel length l in the following blocks to capture context information. For the upsampling layers in the decoder, we utilize feature propagation layers following [27]. Skip connections are added between layers that have the same number of output points. The decoder outputs are then fed into an InterpConv layer with kernel size $1 \times 1 \times 1$ to obtain the final predictions.

4. Experiments

In this section, we evaluate the efficacy of Interpolated Convolutional Neural Networks on multiple tasks, including shape classification, object part segmentation and indoor scene semantic parsing. In all experiments, we implement the models using CUDA and PyTorch on NVIDIA TITAN X GPUs, and we use the Adam optimizer. We first demonstrate the performances of our approach on those tasks. Then we discuss key components of our method in ablation study.

4.1. Shape Classification

Dataset. We evaluate the 3D shape classification performance of our network on the benchmark dataset ModelNet40 [5]. ModelNet40 is composed of 12,311 CAD models which belong to 40 categories with 9,843 for training and 2,468 for testing. We use the point cloud conversion of ModelNet40, where 2,048 points are sampled from each CAD model. We further sample 1,024 points for training and testing following [25].

Implementation details. We adopt the classification network in Figure 2(a). We use Gaussian interpolation as our interpolation function and fix the Gaussian bandwidth 3 to 0.1 in all InterpConv blocks. Point clouds are downsampled to half of the input number after each $3 \times 3 \times 3$ InterpConv layer. The input point clouds are randomly scaled by a factor ranging from 0.8 to 1.2 , then jittered by a zero-mean Gaussian noise with 0.02 standard deviation. We trained the network for 480 epoches with initial learning rate 0.001 and decay rate 0.7 every 80 epoches with batch size 16.

Results. We report the overall accuracy on this dataset. In Table 1, we compare our InterpCNN with other approaches. We demonstrate that the deep architecture based on InterpConvs performs much better than graph-based and voxel-based counterparts, with 0.8% improvement on the best graph-based network DGCNN [45]. Our approach performs even better than Point2Seq [23] and 3DCapsule [8]

(a) Classification network

(b) Segmentation network

Figure 2. Interpolated Convolutional Neural Networks (InterpCNNs). Gray boxes indicate the size of input and output data, and other boxes are all network layers. In the classification network (a), we extend the idea of Inception module [38] and stack two multi-branch, multi-receptive-field PointInception modules to capture both local and context geometric information. We note that kernel length l varies at different branches. In the segmentation network (b), we share similar spirit as U-Net [31] and build an InterpConv-based deep encoder-decoder architecture. Kernel length l begins with a small value and becomes larger as the network goes deeper.

	Input	Acc.
Subvolume [26]	voxels	89.2%
VRN Single [4]	voxels	91.3%
OctNet [30]	hybrid grid octree	86.5%
ECC [35]	graphs	87.4%
PointwiseCNN [11]	1024 points	86.1%
PointNet [25]	1024 points	89.2%
PointNet++ [27]	1024 points	90.7%
PointNet++ [27]	5000 points+normal	91.9%
Kd-Network [16]	1024 points	91.8%
ShapeContextNet [47]	1024 points	90.0%
KCNet [34]	1024 points	91.0%
PointCNN [20]	1024 points	92.2%
DGCNN [45]	1024 points	92.2%
SO-Net [19]	2048 points	90.9%
SpiderCNN [49]	1024 points+normal	92.4%
Point2Seq [23]	1024 points	92.6%
3DCapsule [8]	1024 points	92.7%
PointConv [46]	1024 points+normal	92.5%
InterpCNN (ours)	1024 points	93.0%

Table 1. Classification results on ModelNet40. Overall accuracy is reported.

in which many modules and model compacity are added on top of PointNet++ [27] to gain a better performance.

4.2. Object Part Segmentation

Dataset. We evaluate our segmentation network on the part segmentation dataset ShapeNet Parts [50]. ShapeNet Parts consists of 16,880 models from 16 shape categories, with 14,006 for training and 2,874 for testing. Each model is annotated with 2 to 6 parts and there are 50 different parts in total. Each point sampled from the models is annotated with a part label.

Implementation details. We use the segmentation net-

	Cat. mIOU	Ins. mIOU
PointNet [25]	80.4%	83.7%
PointNet++ [27]	81.9%	85.1%
FCPN [29]	-	84.0%
SyncSpecCNN [51]	82.0%	84.7%
SSCN [10]	83.3%	86.0%
SPLATNet [36]	83.7%	85.4%
SpiderCNN [49]	81.7%	85.3%
SO-Net [19]	81.0%	84.9%
PCNN [2]	81.8%	85.1%
KCNet [34]	82.2%	83.7%
ShapeContextNet [47]	-	84.6%
SpecGCN [41]	-	85.4%
3DmFV [3]	81.0%	84.3%
RSNet [12]	81.4%	84.9%
PointCNN [20]	84.6%	86.1%
DGCNN [45]	82.3%	85.1%
SGPN [44]	82.8%	85.8%
PointConv [46]	82.8%	85.7%
Point2Seq [23]	-	85.2%
InterpCNN (ours)	84.0%	86.3%

Table 2. Segmentation results on ShapeNet Parts. Mean IoU over categories (Cat.) and instances (Ins.) are reported.

work in Figure 2(b). During training we randomly sample 2,048 points from each object and use the original point clouds for testing. Different from the classification network, we utilize a trilinear interpolation function with a smaller kernel length l , which is shown to perform much better. The kernel length l starts with 0.05 in the first InterpConv layer and doubles in the following layers. We use a minibatch of 32 in each GPU and 4 GPUs to train a model. We set the initial learning rate to be 0.005. Data augmentation is the same as classification.

Figure 3. Visualization of object part segmentation results on ShapeNet Parts. The first row is ground truth and the second row is our predictions. From left to right are cars, motorbikes, lamps and chairs.

	Overall Accuracy	Cat. mIOU
PointNet [25]	78.5%	47.6%
ShapeContextNet [47]	81.6%	52.7%
RSNet [12]	-	56.5%
PointCNN [20]	88.1%	65.4%
DGCNN [45]	84.1%	56.1%
SGPN [44]	80.8%	50.4%
SPGraph [18]	85.5%	62.1%
InterpCNN (ours)	88.7%	66.7%

Table 3. 6-fold validation results on S3DIS. Overall accuracy and mean IOU over categories are reported.

Results. We report mean IOU over categories and instances in Table 2. It is worth noting that mean IOU over instances is more realistic. Our approach performs better than compared methods on mean IOU over instances.

4.3. Indoor Scene Segmentation

Dataset. S3DIS [1] is an indoor scene semantic parsing dataset which contains 271 rooms in 6 areas. Each room is scanned by Matterport scanners and every point in the scan is annotated with one semantic label from 13 categories. We follow [25] and split rooms into $1\text{m} \times 1\text{m}$ blocks for training and testing.

Implementation details. Similar to the part segmentation task we use the same architecture in Figure 2(b). The difference is that we take 4,096 points from each $1\text{m} \times 1\text{m}$ block as inputs during training. We construct a 9D vector (XYZ, RGB, and the normalized location) for each input. Other configurations are the same as that in the object part segmentation task.

Results. Following [25], we adopt 6-fold validations on 6 areas, and we report overall accuracy and mean IOU over categories in Table 3. Our approach significantly outperforms state-of-the-art methods on both accuracy and mean IOU.

Figure 4. Qualitative evaluation on S3DIS compared with PointNet++ and DGCNN.

4.4. Ablation Study

We perform ablation studies to investigate components of our InterpCNNs on ModelNet40 and ShapeNet Parts.

Effectiveness of kernel size n and kernel length l . We explore different settings of hyperparameter n and l for the 1st and 2nd PointInception module in Table 4 and Table 5. We first try the setting of all InterpConvs with kernel size $1 \times 1 \times 1$ and only use a maxpooling layer to aggregate global features. We note that this architecture is similar to PointNet [25] in which the network cannot capture local structures, and the result is much worse. This indicates the great power of InterpConvs with kernel size more than 1. Simply replacing one $1 \times 1 \times 1$ InterpConv with $3 \times 3 \times 3$ for each InterpConv block obtains a performance gain of 3%. We also try InterpConvs with a larger kernel size $5 \times 5 \times 5$ but the performance does not improve. We demonstrate that utilizing $3 \times 3 \times 3$ InterpConvs is sufficiently effective and this also reduces model parameters compared with the $5 \times 5 \times 5$ counterparts. We also explore different kernel lengths and we show that this hyperparameter has a significant effect on the final performance. Either too small or too large the kernel length l will harm the accuracy.

Effectiveness of interpolation functions. We try both Gaussian and trilinear interpolation functions in all tasks. In Table 6, the results show that Gaussian interpolation performs better in classification while trilinear interpolation is better in segmentation. We argue that trilinear interpolation can capture fine-grained geometric structures better than Gaussian counterpart, which is more important for segmentation. Gaussian interpolation is able to obtain global shape information more effectively.

Effectiveness of normalization methods. We try normalization according to the number of neighboring points (Eq. (7)) and the sum of interpolation weights (Eq. (8)). The results in Table 7 indicate that both two methods are effective and show comparable performances. It is worth noting that in extreme cases where there are only a few close points but many far away points in the neighborhood of a kernel-weight coordinate, normalization on the sum of interpolation weights is more appropriate.

1st module	2nd module	Accuracy
$1 \times 1 \times 1$	$1 \times 1 \times 1$	89.9%
$3 \times 3 \times 3$	$5 \times 5 \times 5$	92.9%
$5 \times 5 \times 5$	$3 \times 3 \times 3$	92.8%
$3 \times 3 \times 3$	$3 \times 3 \times 3$	93.0%

Table 4. Results of different kernel sizes on ModelNet40.

1st module	2nd module	Accuracy
$0.05 - 0.1 - 0.2$	$0.1 - 0.2 - 0.4$	92.4%
$0.1 - 0.2 - 0.4$	$0.2 - 0.4 - 0.8$	93.0%
$0.2 - 0.4 - 0.8$	$0.4 - 0.8 - 1.6$	91.9%

Table 5. Results of different kernel lengths on ModelNet40.

Interpolation function	ModelNet40	ShapeNet Parts
Gaussian	93.0%	85.0%
Trilinear	92.5%	86.3%

Table 6. Results of different interpolation functions on ModelNet40 and ShapeNet Parts.

Normalization method	Accuracy
interpolation weights	92.8%
number of points	93.0%

Table 7. Results of different normalization methods on ModelNet40.

Method	Parameters	Accuracy
Subvolume [26]	16.6M	89.2%
PointNet [25]	3.5M	89.2%
PointNet++ (MSG) [27]	12M	90.7%
InterpCNN (ours)	12.8M	93.0%

Table 8. Model parameters and performance comparisons on ModelNet40.

Method	Inference time	Accuracy
PointNet++ (MSG) [27]	26.8ms	90.7%
DGCNN [45]	89.7ms	92.2%
InterpCNN (ours)	31.4ms	93.0%

Table 9. Inference time comparisons on ModelNet40.

Model parameters analysis. We report the number of parameters in our classification network on ModelNet40. Results in Table 8 show that even with the comparable model parameters, PointNet++ still performs much worse than our approach. InterpCNNs also have fewer parameters than other 3D convolution methods.

Runtime analysis. We summarize average inference time based on the classification network with batch size 16, 1024 points on an NVIDIA TITAN X GPU, and compare it with pioneering work PointNet++ and DGCNN under the same settings. In Table 9, average inference time of our ap-

Figure 5. Visualization of feature activations learned by different InterpConv kernels on ModelNet40.

Figure 6. Visualization of failure cases on ShapeNet Parts.

proach is slightly slower than PointNet++, but much faster than graph-based approach DGCNN.

Visualization. We visualize activations by different kernels in the first $3 \times 3 \times 3$ InterpConv layer in Figure 5 and some failure cases in Figure 6.

5. Conclusion

We propose a novel convolution InterpConv and Interpolated Convolutional Neural Networks (InterpCNNs) for 3D classification and segmentation. Experiments on ModelNet40, ShapeNet Parts and S3DIS show promising results compared with existing methods. For future work, we plan to explore new deep architectures based on learnable kernel-weight coordinates, and apply our approach on other point cloud processing tasks including 3D detection and instance segmentation.

Acknowledgements

This work is supported in part by SenseTime Group Limited, in part by the General Research Fund through the Research Grants Council of Hong Kong under Grants CUHK14202217, CUHK14203118, CUHK14205615, CUHK14207814, CUHK14213616, CUHK14208417, CUHK14239816, in part by CUHK Direct Grant.

References

- [1] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1534–1543, 2016. [2](#), [7](#)
- [2] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*, 2018. [6](#)
- [3] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 3d point cloud classification and segmentation using 3d modified fisher vector representation for convolutional neural networks. *arXiv preprint arXiv:1711.08241*, 2017. [6](#)
- [4] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016. [6](#)
- [5] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. [2](#), [5](#)
- [6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018. [3](#)
- [7] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017. [1](#)
- [8] Ali Cheraghian and Lars Petersson. 3dcapsule: Extending the capsule architecture to classify 3d point clouds. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1194–1202. IEEE, 2019. [2](#), [5](#), [6](#)
- [9] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017. [3](#)
- [10] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, 2018. [6](#)
- [11] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Point-wise convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 984–993, 2018. [4](#), [6](#)
- [12] Qiangui Huang, Weiye Wang, and Ulrich Neumann. Recurrent slice networks for 3d segmentation of point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2626–2635, 2018. [2](#), [6](#), [7](#)
- [13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. [5](#)
- [14] Mingyang Jiang, Yiran Wu, and Cewu Lu. Pointsift: A sift-like network module for 3d point cloud semantic segmentation. *arXiv preprint arXiv:1807.00652*, 2018. [2](#)
- [15] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5010–5019, 2018. [2](#)
- [16] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 863–872, 2017. [2](#), [6](#)
- [17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. [3](#)
- [18] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018. [7](#)
- [19] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018. [2](#), [6](#)
- [20] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pages 828–838, 2018. [1](#), [2](#), [3](#), [6](#), [7](#)
- [21] Yangyan Li, Soeren Pirk, Hao Su, Charles R Qi, and Leonidas J Guibas. Fpnn: Field probing neural networks for 3d data. In *Advances in Neural Information Processing Systems*, pages 307–315, 2016. [2](#)
- [22] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015. [2](#)
- [23] Xinhai Liu, Zhizhong Han, Yu-Shen Liu, and Matthias Zwicker. Point2sequence: Learning the shape representation of 3d point clouds with an attention-based sequence to sequence network. *arXiv preprint arXiv:1811.02565*, 2018. [2](#), [5](#), [6](#)
- [24] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015. [1](#), [2](#), [4](#)
- [25] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. [2](#), [5](#), [6](#), [7](#), [8](#)
- [26] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016. [6](#), [8](#)
- [27] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017. [2](#), [5](#), [6](#), [8](#)
- [28] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. 3d graph neural networks for rgb-d semantic seg-

- mentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5199–5208, 2017. **1, 2**
- [29] Dario Rethage, Johanna Wald, Jurgen Sturm, Nassir Navab, and Federico Tombari. Fully-convolutional point networks for large-scale point clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 596–611, 2018. **2, 6**
- [30] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017. **2, 6**
- [31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. **5, 6**
- [32] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 56(11):927–941, 2008. **1**
- [33] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. **2**
- [34] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4548–4557, 2018. **1, 2, 3, 6**
- [35] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3693–3702, 2017. **6**
- [36] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2530–2539, 2018. **6**
- [37] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015. **2**
- [38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. **5, 6**
- [39] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3887–3896, 2018. **1**
- [40] Chu Wang, Marcello Pelillo, and Kaleem Siddiqi. Dominant set clustering and pooling for multi-view 3d object recognition. In *Proceedings of British Machine Vision Conference (BMVC)*, volume 12, 2017. **2**
- [41] Chu Wang, Babak Samari, and Kaleem Siddiqi. Local spectral graph convolution for point set feature learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–66, 2018. **6**
- [42] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems*, volume 1, pages 10–15607, 2015. **2**
- [43] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018. **1, 2, 3**
- [44] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2569–2578, 2018. **6, 7**
- [45] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018. **1, 2, 3, 5, 6, 7, 8**
- [46] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. *arXiv preprint arXiv:1811.07246*, 2018. **1, 3, 6**
- [47] Saining Xie, Sainan Liu, Zeyu Chen, and Zhuowen Tu. Attentional shapecontextnet for point cloud recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4606–4615, 2018. **6, 7**
- [48] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015. **5**
- [49] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102, 2018. **1, 2, 3, 6**
- [50] Li Yi, Vladimir G Kim, Duygu Ceylan, I Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, Leonidas Guibas, et al. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (TOG)*, 35(6):210, 2016. **2, 6**
- [51] Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Sync-specncnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017. **6**