PointConv: Deep Convolutional Networks on 3D Point Clouds

Wenxuan Wu, Zhongang Qi, Li Fuxin
CORIS Institute, Oregon State University
`wuwen, qiz, lif@oregonstate.edu`

Abstract

Unlike images which are represented in regular dense grids, 3D point clouds are irregular and unordered, hence applying convolution on them can be difficult. In this paper, we extend the dynamic filter to a new convolution operation, named PointConv. PointConv can be applied on point clouds to build deep convolutional networks. We treat convolution kernels as nonlinear functions of the local coordinates of 3D points comprised of weight and density functions. With respect to a given point, the weight functions are learned with multi-layer perceptron networks and density functions through kernel density estimation. The most important contribution of this work is a novel reformulation proposed for efficiently computing the weight functions, which allowed us to dramatically scale up the network and significantly improve its performance. The learned convolution kernel can be used to compute translation-invariant and permutation-invariant convolution on any point set in the 3D space. Besides, PointConv can also be used as deconvolution operators to propagate features from a subsampled point cloud back to its original resolution. Experiments on ModelNet40, ShapeNet, and ScanNet show that deep convolutional neural networks built on PointConv are able to achieve state-of-the-art on challenging semantic segmentation benchmarks on 3D point clouds. Besides, our experiments converting CIFAR-10 into a point cloud showed that networks built on PointConv can match the performance of convolutional networks in 2D images of a similar structure.

1. Introduction

In recent robotics, autonomous driving and virtual/augmented reality applications, sensors that can directly obtain 3D data are increasingly ubiquitous. This includes indoor sensors such as laser scanners, time-of-flight sensors such as the Kinect, RealSense or Google Tango, structural light sensors such as those on the iPhoneX, as well as outdoor sensors such as LIDAR and MEMS sensors. The capability to directly measure 3D data is invaluable in those applications as depth information could remove a lot of the segmentation ambiguities from 2D imagery, and surface normals provide important cues of the scene geometry.

In 2D images, convolutional neural networks (CNNs) have fundamentally changed the landscape of computer vision by greatly improving results on almost every vision task. CNNs succeed by utilizing translation invariance, so that the same set of convolutional filters can be applied on all the locations in an image, reducing the number of parameters and improving generalization. We would hope such successes to be transferred to the analysis of 3D data. However, 3D data often come in the form of point clouds, which is a set of unordered 3D points, with or without additional features (e.g. RGB) on each point. Point clouds are unordered and do not conform to the regular lattice grids as in 2D images. It is difficult to apply conventional CNNs on such unordered input. An alternative approach is to treat the 3D space as a volumetric grid, but in this case, the volume will be sparse and CNNs will be computationally intractable on high-resolution volumes.

In this paper, we propose a novel approach to perform convolution on 3D point clouds with non-uniform sampling. We note that the convolution operation can be viewed as a discrete approximation of a continuous convolution operator. In 3D space, we can treat the weights of this convolution operator to be a (Lipschitz) continuous function of the local 3D point coordinates with respect to a reference 3D point. The continuous function can be approximated by a multi-layer perceptron(MLP), as done in [33] and [16]. But these algorithms did not take non-uniform sampling into account. We propose to use an inverse density scale to re-weight the continuous function learned by MLP, which corresponds to the Monte Carlo approximation of the continuous convolution. We call such an operation **PointConv**. PointConv involves taking the positions of point clouds as input and learning an MLP to approximate a weight function, as well as applying a inverse density scale on the learned weights to compensate the non-uniform sampling.

The naive implementation of PointConv is memory inefficient when the channel size of the output features is very large and hence hard to train and scale up to large networks. In order to reduce the memory consumption of PointConv,

we introduce an approach which is able to greatly increase the memory efficiency using a reformulation that changes the summation order. The new structure is capable of building multi-layer deep convolutional networks on 3D point clouds that have similar capabilities as 2D CNN on raster images. We can achieve the same translation-invariance as in 2D convolutional networks, and the invariance to permutations on the ordering of points in a point cloud.

In segmentation tasks, the ability to transfer information gradually from coarse layers to finer layer is important. Hence, a deconvolution operation [24] that can fully leverage the feature from a coarse layer to a finer layer is vital for the performance. Most state-of-the-art algorithms [26, 28] are unable to perform deconvolution, which restricts their performance on segmentation tasks. Since our PointConv is a full approximation of convolution, it is natural to extend PointConv to a PointDeconv, which can fully untilize the information in coarse layers and propagate to finer layers. By using PointConv and PointDeconv, we can achieve improved performance on semantic segmentation tasks.

The contributions of our work are:
- We propose PointConv, a density re-weighted convolution, which is able to fully approximate the 3D continuous convolution on any set of 3D points.
- We design a memory efficient approach to implement PointConv using a change of summation order technique, most importantly, allowing it to scale up to modern CNN levels.
- We extend our PointConv to a deconvolution version(PointDeconv) to achieve better segmentation results.

Experiments show that our deep network built on Point-Conv is highly competitive against other point cloud deep networks and achieve state-of-the-art results in part segmentation [2] and indoor semantic segmentation benchmarks [5]. In order to demonstrate that our PointConv is indeed a true convolution operation, we also evaluate PointConv on CIFAR-10 by converting all pixels in a 2D image into a point cloud with 2D coordinates along with RGB features on each point. Experiments on CIFAR-10 show that the classification accuracy of our PointConv is comparable with a image CNN of a similar structure, far outperforming previous best results achieved by point cloud networks. As a basic approach to CNN on 3D data, we believe there could be many potential applications of PointConv.

2. Related Work

Most work on 3D CNN networks convert 3D point clouds to 2D images or 3D volumetric grids. [36, 27] proposed to project 3D point clouds or shapes into several 2D images, and then apply 2D convolutional networks for classification. Although these approaches have achieved dominating performances on shape classification and retrieval tasks, it is nontrivial to extend them to high-resolution scene

segmentation tasks [5]. [43, 23, 27] represent another type of approach that voxelizes point clouds into volumetric grids by quantization and then apply 3D convolution networks. This type of approach is constrained by its 3D volumetric resolution and the computational cost of 3D convolutions. [31] improves the resolution significantly by using a set of unbalanced octrees where each leaf node stores a pooled feature representation. Kd-networks[18] computes the representations in a feed-forward bottom-up fashion on a Kd-tree with certain size. In a Kd-network, the input number of points in the point cloud needs to be the same during training and testing, which does not hold for many tasks. SSCN [7] utilizes the convolution based on a volumetric grid with novel speed/memory improvements by considering CNN outputs only on input points. However, if the point cloud is sampled sparsely, especially when the sampling rate is uneven, for the sparsely sampled regions on may not be able to find any neighbor within the volumetric convolutional filter, which could cause significant issues.

Some latest work [30, 26, 28, 35, 37, 13, 9, 39] directly take raw point clouds as input without converting them to other formats. [26, 30] proposed to use shared multi-layer perceptrons and max pooling layers to obtain features of point clouds. Because the max pooling layers are applied across all the points in point cloud, it is difficult to capture local features. PointNet++ [28] improved the network in PointNet [26] by adding a hierarchical structure. The hierarchical structure is similar to the one used in image CNNs, which extracts features starting from small local regions and gradually extending to larger regions. The key structure used in both PointNet [26] and PointNet++ [28] to aggregate features from different points is max-pooling. However, max-pooling layers keep only the strongest activation on features across a local or global region, which may lose some useful detailed information for segmentation tasks. [35] presents a method that projects the input features of the point clouds onto a high-dimensional lattice, and then apply bilateral convolution on the high-dimensional lattice to aggregate features, which called "SPLATNet". The SPLATNet [35] is able to give comparable results as PointNet++ [28]. The tangent convolution [37] projects local surface geometry on a tangent plane around every point, which gives a set of planar-convolutionable tangent images. The pointwise convolution [13] queries nearest neighbors on the fly and bins the points into kernel cells, then applies kernel weights on the binned cells to convolve on point clouds. Flex-convolution [9] introduced a generalization of the conventional convolution layer along with an efficient GPU implementation, which can applied to point clouds with millions of points. FeaStNet [39] proposes to generalize conventional convolution layer to 3D point clouds by adding a soft-assignment matrix. PointCNN [21] is to learn a $\chi-$transformation from

the input points and then use it to simultaneously weight and permute the input features associated with the points. Comparing to our approach, PointCNN is unable to achieve permutation-invariance, which is desired for point clouds.

The work [33, 16, 41, 12, 40] and [44] propose to learn continuous filters to perform convolution. [16] proposed that the weight filter in 2d convolution can be treated as a continuous function, which can be approximated by MLPs. [33] firstly introduced the idea into 3d graph structure. [40] extended the method in [33] to segmentation tasks and proposed an efficient version, but their efficient version can only approximate depth-wise convolution instead of real convolution. Dynamic graph CNN [41] proposed a method that can dynamically updating the graph. [44] presents a special family of filters to approximate the weight function instead of using MLPs. [12] proposed a Monta Carlo approximation of 3D convolution by taking density into account. Our work differ from those in 3 aspects. Most importantly, our efficient version of a real convolution was never proposed in prior work. Also, we utilize density differently than [12], and we propose a deconvolution operator based on PointConv to perform semantic segmentation.

3. PointConv

We propose a convolution operation which extends traditional image convolution into the point cloud called **Point-Conv**. PointConv is an extension to the Monte Carlo approximation of the 3D continuous convolution operator. For each convolutional filter, it uses MLP to approximate a weight function, then applies a density scale to re-weight the learned weight functions. Sec. 3.1 introduces the structure of the PointConv layer. Sec. 3.2 introduces PointDeconv, using PointConv layers to deconvolve features.

3.1. Convolution on 3D Point Clouds

Formally, convolution is defined as in Eq.(1) for functions $f(\mathbf{x})$ and $g(\mathbf{x})$ of a d-dimensional vector $\mathbf{x}$.

$$(f * g)(\mathbf{x}) = \iint_{\boldsymbol{\tau} \in \mathbb{R}^d} f(\boldsymbol{\tau}) g(\mathbf{x} + \boldsymbol{\tau}) d\boldsymbol{\tau} \qquad (1)$$

Images can be interpreted as 2D discrete functions, which are usually represented as grid-shaped matrices. In CNN, each filter is restricted to a small local region, such as $3 \times 3, 5 \times 5$, etc. Within each local region, the relative positions between different pixels are always fixed, as shown in Figure 1(a). And the filter can be easily discretized to a summation with a real-valued weight for each location within the local region.

A point cloud is represented as a set of 3D points $\{p_i | i = 1, ..., n\}$, where each point contains a position vector (x, y, z) and its features such as color, surface normal, etc. Different from images, point clouds have more flexible
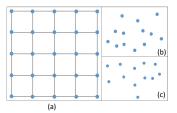

Figure 1. **Image grid vs. point cloud.** (a) shows a 5×5 local region in a image, where the distance between points can only attain very few discrete values; (b) and (c) show that in different local regions within a point cloud, the order and the relative positions can be very different.
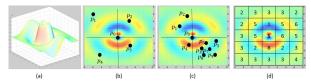

Figure 2. **2D weight function for PointConv.** (a) is a learned continuous weight function; (b) and (c) are different local regions in a 2d point cloud. Given 2d points, we can obtain the weights at particular locations. The same applies to 3D points. The regular discrete 2D convolution can be viewed as a discretization of the continuous convolution weight function, as in (d).

shapes. The coordinates $p = (x, y, z) \in \mathbb{R}^3$ of a point in a point cloud are not located on a fixed grid but can take an arbitrary continuous value. Thus, the relative positions of different points are diverse in each local region. Conventional discretized convolution filters on raster images cannot be applied directly on the point cloud. Fig. 1 shows the difference between a local region in a image and a point cloud.

To make convolution compatible with point sets, we propose a permutation-invariant convolution operation, called **PointConv**. Our idea is to first go back to the continuous version of 3D convolution as:

$$Conv(W, F)_{xyz} =$$
$$\iiint_{(\delta_x, \delta_y, \delta_z) \in G} W(\delta_x, \delta_y, \delta_z) F(x + \delta_x, y + \delta_y, z + \delta_z) d\delta_x \delta_y \delta_z$$
$$(2)$$

where $F(x + \delta_x, y + \delta_y, z + \delta_z)$ is the feature of a point in the local region G centered around point $p = (x, y, z)$. A point cloud can be viewed as a non-uniform sample from the continuous $\mathbb{R}^3$ space. In each local region, $(\delta_x, \delta_y, \delta_z)$ could be any possible position in the local region. We define *PointConv* as the following:

$$PointConv(S, W, F)_{xyz} =$$
$$\sum_{(\delta_x, \delta_y, \delta_z) \in G} S(\delta_x, \delta_y, \delta_z) W(\delta_x, \delta_y, \delta_z) F(x + \delta_x, y + \delta_y, z + \delta_z)$$
$$(3)$$

where $S(\delta_x, \delta_y, \delta_z)$ is the inverse density at point $(\delta_x, \delta_y, \delta_z)$. $S(\delta_x, \delta_y, \delta_z)$ is required because the point cloud can be sampled very non-uniformly[1]. Intuitively, the

---

[1] To see this, note the Monte Carlo estimate with a biased sample:

number of points in the local region varies across the whole point cloud, as in Figure 2 (b) and (c). Besides, in Figure 2 (c), points $p_3, p_5, p_6, p_7, p_8, p_9, p_{10}$ are very close to one another, hence the contribution of each should be smaller.

Our main idea is to approximate the weight function $W(\delta_x, \delta_y, \delta_z)$ by multi-layer perceptrons from the 3D coordinates $(\delta_x, \delta_y, \delta_z)$ and and the inverse density $S(\delta_x, \delta_y, \delta_z)$ by a kernelized density estimation [38] followed by a nonlinear transform implemented with MLP. Because the weight function highly depends on the distribution of input point cloud, we call the entire convolution operation *Point-Conv*. [16, 33] considered the approximation of the weight function but did not consider the approximation of the density scale, hence is not a full approximation of the continuous convolution operator. Our nonlinear transform on the density is also different from [12].

The weights of the MLP in PointConv are shared across all the points in order to maintain the permutation invariance. To compute the inverse density scale $S(\delta_x, \delta_y, \delta_z)$, we first estimate the density of each point in a point cloud offline using kernel density estimation (KDE), then feed the density into a MLP for a 1D nonlinear transform. The reason to use a nonlinear transform is for the network to decide adaptively whether to use the density estimates.

Figure 3 shows the PointConv operation on a K-point local region. Let C_{in}, C_{out} be the number of channels for the input feature and output feature, k, c_{in}, c_{out} are the indices for k-th neighbor, c_{in}-th channel for input feature and c_{out}-th channel for output feature. The inputs are the 3D local positions of the points $P_{local} \in \mathbb{R}^{K \times 3}$, which can be computed by subtracting the coordinate of the centroid of the local region and the feature $F_{in} \in \mathbb{R}^{K \times C_{in}}$ of the local region. We use 1×1 convolution to implement the MLP. The output of the weight function is $W \in \mathbb{R}^{K \times (C_{in} \times C_{out})}$. So, $\mathbf{W}(k, c_{in}) \in \mathbb{R}^{C_{out}}$ is a vector. The density scale is $S \in \mathbb{R}^K$. After convolution, the feature F_{in} from a local region with K neighbour points are encoded into the output feature $\mathbf{F}_{out} \in \mathbb{R}^{C_{out}}$, as shown in Eq.(4).

$$\mathbf{F}_{out} = \sum_{k=1}^{K} \sum_{c_{in}=1}^{C_{in}} S(k)\mathbf{W}(k, c_{in})F_{in}(k, c_{in}) \quad (4)$$

PointConv learns a network to approximate the continuous weights for convolution. For each input point, we can compute the weights from the MLPs using its relative coordinates. Figure 2(a) shows an example continuous weight function for convolution. With a point cloud input as a discretization of the continuous input, a discrete convolution can be computed by Fig. 2(b) to extract the local features, which would work (with potentially different approximation accuracy) for different point cloud samples (Figure 2(b-

---

$\int f(x)dx = \int \frac{f(x)}{p(x)} p(x)dx \approx \sum_i \frac{f(x_i)}{p(x_i)}$, for $x_i \sim p(x)$. Point clouds are often biased samples because many sensors have difficulties measuring points near plane boundaries, hence needing this reweighting
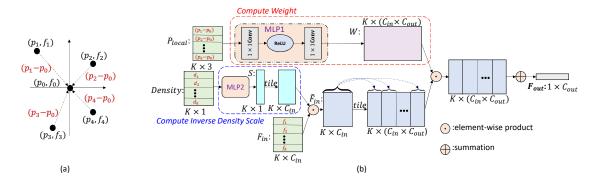
---

d)), including a regular grid (Figure 2(d)). Note that in a raster image, the relative positions in local region are fixed. Then PointConv (which takes only relative positions as input for the weight functions) would output the same weight and density across the whole image, where it reduces to the conventional discretized convolution.

In order to aggregate the features in the entire point set, we use a hierarchical structure that is able to combine detailed small region features into abstract features that cover a larger spatial extent. The hierarchical structure we use is composed by several *feature encoding modules*, which is similar to the one used in PointNet++ [28]. Each module is roughly equivalent to one layer in a convolutional CNN. The key layers in each *feature encoding module* are the sampling layer, the grouping layer and the PointConv. More details can be found in the supplementary material.

The drawback of this approach is that each filter needs to be approximated by a network, hence is very inefficient. In Sec.4, we propose an efficient approach to implement PointConv.

3.2. Feature Propagation Using Deconvolution

For the segmentation task, we need point-wise prediction. In order to obtain features for all the input points, an approach to propagate features from a subsampled point cloud to a denser one is needed. PointNet++ [28] proposes to use distance-based interpolation to propagate features, which is reasonable due to local correlations inside a local region. However, this does not take full advantage of the deconvolution operation that captures local correlations of propagated information from the coarse level. We propose to add a PointDeconv layer based on the PointConv, as a deconvolution operation to address this issue.

As shown in Fig. 4, PointDeconv is composed of two parts: interpolation and PointConv. Firstly, we employ an interpolation to propagate coarse features from previous layer. Following [28], the interpolation is conducted by linearly interpolating features from the 3 nearest points. Then, the interpolated features are concatenated with features from the convolutional layers with the same resolution using skip links. After concatenation, we apply Point-Conv on the concatenated features to obtain the final deconvolution output, similar to the image deconvolution layer [24]. We apply this process until the features of all the input points have been propagated back to the original resolution.

4. Efficient PointConv

The naive implementation of the PointConv is memory consuming and inefficient. Different from [33], we propose a novel reformulation to implement PointConv by reducing it to two standard operations: matrix multiplication and 2d convolution. This novel trick not only takes advantage of the parallel computing of GPU, but also can be easily

Figure 3. **PointConv**. (a) shows a local region with the coordinates of points transformed from global into local coordinates, p is the coordinates of points, and f is the corresponding feature; (b) shows the process of conducting PointConv on one local region centered around one point (p_0, f_0). The input features come form the K nearest neighbors centered at (p_0, f_0), and the output feature is $\mathbf{F}_{out}$ at p_0.
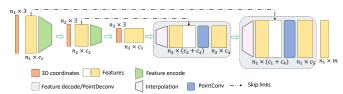

Figure 4. **Feature encoding and propagation.** This figure shows how the features are encoded and propagated in the network for a m classes segmentation task. n is the number of points in each layer, c is the channel size for the features. Best viewed in color.

implemented using main-stream deep learning frameworks. Because the inverse density scale does not have such memory issues, the following discussion mainly focuses on the weight function.

Specifically, let B be the mini-batch size in the training stage, N be the number of points in a point cloud, K be the number of points in each local region, C_{in} be the number of input channels, and C_{out} be the number of output channels. For a point cloud, each local region shares the same weight functions which can be learned using MLP. However, weights computed from the weight functions at different points are different. The size of the weights filters generated by the MLP is $B \times N \times K \times (C_{in} \times C_{out})$. Suppose $B = 32$, $N = 512$, $K = 32$, $C_{in} = 64$, $C_{out} = 64$, and the filters are stored with single point precision. Then, the memory size for the filters is $8GB$ for only one layer. The network would be hard to train with such high memory consumption. [33] used very small network with few filters which significantly degraded its performance. To resolve this problem, we propose a memory efficient version of PointConv based on the following lemma:

Lemma 1 *The PointConv is equivalent to the following formula:* $\mathbf{F}_{out} = Conv_{1\times1}(\mathbf{H}, (\mathbf{S} \cdot \mathbf{F}_{in})^T \otimes \mathbf{M})$ *where* $\mathbf{M} \in \mathbb{R}^{K \times C_{mid}}$ *is the input to the last layer in the MLP for computing the weight function, and* $\mathbf{H} \in \mathbb{R}^{C_{mid} \times (C_{in} \times C_{out})}$ *is the weights of the last layer in the same MLP,* $Conv_{1\times1}$ *is* 1×1 *convolution.*

Proof: Generally, the last layer of the MLP is a linear layer.

In one local region, let $\widetilde{\mathbf{F}}_{in} = \mathbf{S} \cdot \mathbf{F}_{in} \in \mathbb{R}^{K \times C_{in}}$ and rewrite the MLP as a 1×1 convolution so that the output of the weight function is $\mathbf{W} = Conv_{1\times1}(H, M) \in \mathbb{R}^{K \times (C_{in} \times C_{out})}$. Let k is the index of the points in a local region, and c_{in}, c_{mid}, c_{out} are the indices of the input, middle layer and the filter output, respectively. Then $\mathbf{W}(k, c_{in}) \in \mathbb{R}^{C_{out}}$ is a vector from $\mathbf{W}$. And the $\mathbf{H}(c_{mid}, c_{in}) \in \mathbb{R}^{C_{out}}$ is a vector from $\mathbf{H}$. According to Eq.(4), the PointConv can be expressed in Eq.(5).

$$\mathbf{F}_{out} = \sum_{k=0}^{K-1} \sum_{c_{in}=0}^{C_{in}-1} (\mathbf{W}(k, c_{in})\widetilde{\mathbf{F}}_{in}(k, c_{in})) \quad (5)$$

Let's explore Eq.(5) in a more detailed manner. The output of the weight function can be expressed as:

$$\mathbf{W}(k, c_{in}) = \sum_{c_{mid}=0}^{C_{mid}-1} (\mathbf{M}(k, c_{mid})\mathbf{H}(c_{mid}, c_{in})) \quad (6)$$

Substituting Eq.(6) into Eq.(5).

$$\begin{aligned}\mathbf{F}_{out} &= \sum_{k=0}^{K-1} \sum_{c_{in}=0}^{C_{in}-1} (\widetilde{\mathbf{F}}_{in}(k, c_{in}) \sum_{c_{mid}=0}^{C_{mid}-1} (\mathbf{M}(k, c_{mid})\mathbf{H}(c_{mid}, c_{in}))) \\ &= \sum_{c_{in}=0}^{C_{in}-1} \sum_{c_{mid}=0}^{C_{mid}-1} (\mathbf{H}(c_{mid}, c_{in}) \sum_{k=0}^{K-1} (\widetilde{\mathbf{F}}_{in}(k, c_{in})\mathbf{M}(k, c_{mid}))) \\ &= Conv_{1\times1}(\mathbf{H}, \widetilde{\mathbf{F}}_{in}^T \mathbf{M})\end{aligned} \quad (7)$$

Thus, the original PointConv can be equivalently reduced to a matrix multiplication and a 1×1 convolution. Figure 5 shows the efficient version of PointConv.

In this method, instead of storing the generated filters in memory, we divide the weights filters into two parts: the intermediate result M and the convolution kernel H. As we can see, the memory consumption reduces to $\frac{\mathbf{C_{mid}}}{\mathbf{K} \times \mathbf{C_{out}}}$ of the original version. With the same input setup as the Figure 3 and let $C_{mid} = 32$, the memory consumption is $0.1255GB$, which is about $1/64$ of the original PointConv.
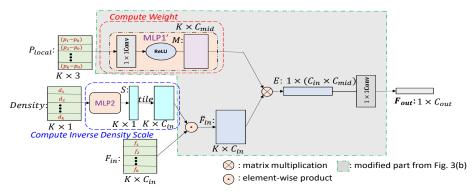
Figure 5. **Efficient PointConv.** The memory efficient version of PointConv on one local region with K points.

Table 1. **ModelNet40 Classification Accuracy**

Method	Input	Accuracy(%)
Subvolume [27]	voxels	89.2
ECC [33]	graphs	87.4
Kd-Network [18]	1024 points	91.8
PointNet [26]	1024 points	89.2
PointNet++ [28]	1024 points	90.2
PointNet++ [28]	5000 points+normal	91.9
SpiderCNN [44]	1024 points+normal	92.4
PointConv	1024 points+normal	**92.5**

5. Experiments

In order to evaluate our new PointConv network, we conduct experiments on several widely used datasets, ModelNet40 [43], ShapeNet [2] and ScanNet [5]. In order to demonstrate that our PointConv is able to fully approximate conventional convolution, we also report results on the CIFAR-10 dataset [19]. In all experiments, we implement the models with Tensorflow on a GTX 1080Ti GPU using the Adam optimizer. ReLU and batch normalization are applied after each layer except the last fully connected layer.

5.1. Classification on ModelNet40

ModelNet40 contains 12,311 CAD models from 40 manmade object categories. We use the official split with 9,843 shapes for training and 2,468 for testing. Following the configuration in [26], we use the source code for PointNet [26] to sample 1,024 points uniformly and compute the normal vectors from the mesh models. For fair comparison, we employ the same data augmentation strategy as [26] by randomly rotating the point cloud along the z-axis and jittering each point by a Gaussian noise with zero mean and 0.02 standard deviation. In Table 1, PointConv achieved state-of-the-art performance among methods based on 3D input. ECC[33] which is similar to our approach, cannot scale to a large network, which limited their performance.
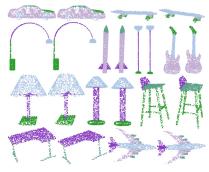

Figure 6. **Part segmentation results.** For each pair of objects, the left one is the ground truth, the right one is predicted by PointConv. Best viewed in color.

Table 2. **Results on ShapeNet part dataset.** Class avg. is the mean IoU averaged across all object categories, and inctance avg. is the mean IoU across all objects.

	class avg.	instance avg.
SSCNN [45]	82.0	84.7
Kd-net [18]	77.4	82.3
PointNet [26]	80.4	83.7
PointNet++[28]	81.9	85.1
SpiderCNN [44]	82.4	85.3
SPLATNet$_{3D}$ [35]	82.0	84.6
SSCN [7]	-	**86.0**
PointConv	**82.8**	85.7

5.2. ShapeNet Part Segmentation

Part segmentation is a challenging fine-grained 3D recognition task. The ShapeNet dataset contains 16,881 shapes from 16 classes and 50 parts in total. The input of the task is shapes represented by a point cloud, and the goal is to assign a part category label to each point in the point cloud. The category label for each shape is given. We follow the experiment setup in most related work [28, 35, 44, 18]. It is common to narrow the possible part labels to the ones specific to the given object category by using the known input 3D object category. And we also compute the normal direction on each point as input features to better describe the underlying shape. Figure 6 visualizes some sample results.

We use point intersection-over-union(IoU) to evaluate our PointConv network, same as PointNet++ [28], SPLAT-Net [35] and some other part segmentation algorithms [45, 18, 44, 7]. The results are shown in Table 2. Point-Conv obtains a class average mIoU of 82.8% and an instance average mIoU of 85.7%, which are on par with the state-of-the-art algorithms which only take point clouds as input. According to [35], the SPLATNet$_{2D-3D}$ also takes rendered 2D views as input. Since our PointConv only takes 3D point clouds as input, for fair comparison, we only compare our result with the SPLATNet$_{3D}$ in [35].

5.3. Semantic Scene Labeling

Datasets such as ModelNet40 [43] and ShapeNet [2] are man-made synthetic datasets. As we can see in the previous section, most state-of-the-art algorithms are able to obtain relatively good results on such datasets. To evaluate the capability of our approach in processing realistic point clouds, which contains a lot of noisy data, we evaluate our PointConv on semantic scene segmentation using the Scan-Net dataset. The task is to predict semantic object labels on each 3D point given indoor scenes represented by point clouds. The newest version of ScanNet [5] includes updated annotations for all 1513 ScanNet scans and 100 new test scans with all semantic labels publicly unavailable and we submitted our results to the official evaluation server to compare against other approaches.

We compare our algorithm with Tangent Convolutions [37], SPLAT Net [35], PointNet++ [28] and ScanNet [5]. All the algorithm mentioned reported their results on the new ScanNet dataset to the benchmark, and the inputs of the algorithms only uses 3D coordinates data plus RGB. In our experiments, we generate training samples by randomly sample $3m \times 1.5m \times 1.5m$ cubes from the indoor rooms, and evaluate using a sliding window over the entire scan. We report intersection over union (IoU) as our main measures, which is the same as the benchmark. We visualize some example semantic segmentation results in Figure 7. The mIoU is reported in Table 3. The mIoU is the mean of IoU across all the categories. Our PointConv outperforms other algorithm by a significant margin (Table 3). The total running time of PointConv for training one epoch on Scan-Net on one GTX1080Ti is around 170s, and the evaluation time with 8×8192 points is around 0.5s.

5.4. Classification on CIFAR-10

In Sec.3.1, we claimed that PointConv can be equivalent with 2D CNN. If this is true, then the performance of a network based on PointConv should be equivalent to that of a raster image CNN. In order to verify that, we use the CIFAR-10 dataset as a comparison benchmark. We treat each pixel in CIFAR-10 as a 2D point with xy coordinates and RGB features. The point clouds are scaled onto the unit
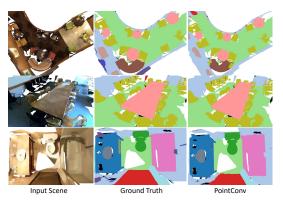

Figure 7. **Examples of semantic scene labeling.** The images from left to right are the input scenes, the ground truth segmentation, and the prediction from PointConv. For better visualization, the point clouds are converted into mesh format. Best viewed in color.

Table 3. **Semantic Scene Segmentation results on ScanNet**

Method	mIoU(%)
ScanNet [5]	30.6
PointNet++ [28]	33.9
SPLAT Net [35]	39.3
Tangent Convolutions [37]	43.8
PointConv	**55.6**

Table 4. **CIFAR-10 Classification Accuracy**

	Accuracy(%)
Image Convolution	88.52
AlexNet [20]	89.00
VGG19 [34]	93.60
PointCNN [21]	80.22
SpiderCNN [44]	77.97
PointConv(5-layer)	89.13
PointConv(VGG19)	93.19

ball before training and testing.

Experiments show that PointConv on CIFAR-10 indeed has the same learning capacities as a 2D CNN. Table 4 shows the results of image convolution and Point-Conv. From the table, we can see that the accuracy of PointCNN[21] on CIFAR-10 is only 80.22%, which is much worse than image CNN. However, for 5-layer networks, the network using PointConv is able to achieve 89.13%, which is similar to the network using image convolution. And, PointConv with VGG19 [34] structure can also achieve on par accuracy comparing with VGG19.

6. Ablation Experiments and Visualizations

In this section, we conduct additional experiments to evaluate the effectiveness of each aspect of PointConv. Besides the ablation study on the structure of the PointConv, we also give an in-depth breakdown on the performance

of PointConv on the ScanNet dataset. Finally, we provide some learned filters for visualization.

6.1. The Structure of MLP

In this section, we design experiments to evaluate the choice of MLP parameters in PointConv. For fast evaluation, we generate a subset from the ScanNet dataset as a classification task. Each example in the subset is randomly sampled from the original scene scans with 1,024 points. There are 20 different scene types for the ScanNet dataset. We empirically sweep over different choices of C_{mid} and different number of layers of the MLP in PointConv. Each experiment was ran for 3 random trials.The results can be find in supplementary. From the results, we find that larger C_{mid} does not necessarily give better classification results. And the different number of layers in MLP does not give much difference in classification results. Since C_{mid} is linearly correlated with the memory consumption of each PointConv layer, this results shows that we can choose a reasonably small C_{mid} for greater memory efficiency.

6.2. Inverse Density Scale

In this section, we study the effectiveness of the inverse density scale S. We choose ScanNet as our evaluation task since the point clouds in ScanNet are generated from real indoor scenes. We follow the standard training/validation split provided by the authors. We train the network with and without the inverse density scale as described in Sec. 3.1, respectively. Table 5 shows the results. As we can see, Point-Conv with inverse density scale performs better than the one without by about 1%, which proves the effectiveness of inverse density scale. In our experiments, we observe that inverse density scale tend to be more effective in layers closer to the input. In deep layers, the MLP tends to diminish the effect of the density scale. One possible reason is that with farthest point sampling algorithm as our sub-sampling algorithm, the point cloud in deeper layer tend to be more uniformly distributed. And as shown in Table 5, directly applying density without using the nonlinear transformation gives worse result comparing with the one without density on ScanNet dataset, which shows that the nonlinear transform is able to learn the inverse density scale in the dataset.

6.3. Ablation Studies on ScanNet

As one can see, our PointConv outperforms other approaches with a large margin. Since we are only allowed to submit one final result of our algorithm to the benchmark server of ScanNet, we perform more ablation studies for PointConv using the public validation set provide by [5]. For the segmentation task, we train our PointConv with 8,192 points randomly sampled from a $3m \times 1.5m \times 1.5m$, and evaluate the model with exhaustively choose all points in the $3m \times 1.5m \times 1.5m$ cube in a sliding window fashion
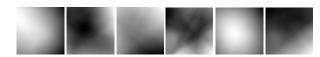

Figure 8. **Learned Convolutional Filters.** The convolution filters learned by the MLPs on ShapeNet.For better visualization, we take all weights filters from $z = 0$ plane.

through the xy-plane with different stride sizes. For robustness, we use a majority vote from 5 windows in all of our experiments. From Table 5, we can see that smaller stride size is able to improve the segmentation results, and the RGB information on ScanNet does not seem to significantly improve the segmentation results. Even without these additional improvements, PointConv still outperforms baselines by a large margin.

Table 5. **Ablation study on ScanNet.** With and without RGB information, inverse density scale and using different stride size of sliding window.

Input	Stride Size(m)	mIoU	mIoU No Density	mIoU Density (no MLP)
	0.5	61.0	60.3	60.1
xyz	1.0	59.0	58.2	57.7
	1.5	58.2	56.9	57.3
	0.5	60.8	58.9	-
xyz+RGB	1.0	58.6	56.7	-
	1.5	57.5	56.1	-

6.4. Visualization

Figure 8 visualizes the learned filters from the MLPs in our PointConv. In order to better visualize the filters, we sample the learned functions through a plane $z = 0$. From the Figure 8, we can see some patterns in the learned continuous filters.

7. Conclusion

In this work, we proposed a novel approach to perform convolution operation on 3D point clouds, called PointConv. PointConv trains multi-layer perceptrons on local point coordinates to approximate continuous weight and density functions in convolutional filters, which makes it naturally permutation-invariant and translation-invariant. This allows deep convolutional networks to be built directly on 3D point clouds. We proposed an efficient implementation of it which greatly improved its scalability. We demonstrated its strong performance on multiple challenging benchmarks and capability of matching the performance of a grid-based convolutional network in 2D images. In future work, we would like to adopt more mainstream image convolution network architectures into point cloud data using PointConv, such as ResNet and DenseNet. The code can be found here: https://github.com/DylanWusee/pointconv.

References

[1] Michael M Bronstein and Iasonas Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1704–1711. IEEE, 2010.

[2] 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, 6, 7

[3] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*, volume 22, pages 223–232. Wiley Online Library, 2003.

[4] Hang Chu, Wei-Chiu Ma3 Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Surfconv: Bridging 3d and 2d convolution for rgbd images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3002–3011, 2018.

[5] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 1, 2017. 2, 6, 7, 8

[6] Yi Fang, Jin Xie, Guoxian Dai, Meng Wang, Fan Zhu, Tiantian Xu, and Edward Wong. 3d deep shape descriptor. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2319–2328, 2015.

[7] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017. 2, 6, 7

[8] Adrien Gressin, Clément Mallet, Jérôme Demantké, and Nicolas David. Towards 3d lidar point cloud registration improvement using optimal neighborhood knowledge. *ISPRS journal of photogrammetry and remote sensing*, 79:240–251, 2013.

[9] Fabian Groh, Patrick Wieschollek, and Hendrik Lensch. Flex-convolution (deep learning beyond grid-worlds). *arXiv preprint arXiv:1803.07289*, 2018. 2

[10] Kan Guo, Dongqing Zou, and Xiaowu Chen. 3d mesh labeling via deep convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 35(1):3, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. In *SIGGRAPH Asia 2018 Technical Papers*, page 235. ACM, 2018. 3, 4

[13] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 984–993, 2018. 2

[14] Qiangui Huang, Weiyue 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.

[15] Jörn-Henrik Jacobsen, Jan van Gemert, Zhongyou Lou, and Arnold WM Smeulders. Structured receptive fields in cnns. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2610–2619. IEEE, 2016.

[16] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, pages 667–675, 2016. 1, 3, 4

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 863–872. IEEE, 2017. 2, 6, 7

[19] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 6

[20] 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. 7

[21] Yangyan Li, Rui Bu, Mingchao Sun, and Baoquan Chen. Pointcnn. *arXiv preprint arXiv:1801.07791*, 2018. 2, 7

[22] Haibin Ling and David W Jacobs. Shape classification using the inner-distance. *IEEE transactions on pattern analysis and machine intelligence*, 29(2):286–299, 2007.

[23] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015. 2

[24] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015. 2, 4

[25] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. *arXiv preprint arXiv:1711.08488*, 2017.

[26] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017. 2, 6

[27] 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. 2, 6

[28] 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 5105–5114, 2017. 2, 4, 6, 7

[29] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. 3d graph neural networks for rgbd semantic segmentation. In *Proceedings of theqi IEEE Conference on*

Computer Vision and Pattern Recognition, pages 5199–5208, 2017.

[30] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500*, 2016. 2

[31] 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*, volume 3, 2017. 2

[32] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009.

[33] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*, 2017. 1, 3, 4, 5, 6

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 7

[35] 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. 2, 6, 7

[36] 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

[37] 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. 2, 7

[38] Berwin A Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*. Citeseer, 1993. 4

[39] Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *CVPR 2018-IEEE Conference on Computer Vision & Pattern Recognition*, 2018. 2

[40] 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. 3

[41] 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. 3

[42] Zizhao Wu, Ruyang Shou, Yunhai Wang, and Xinguo Liu. Interactive shape co-segmentation via label propagation. *Computers & Graphics*, 38:248–254, 2014.

[43] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 2, 6, 7

[44] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. *arXiv preprint arXiv:1803.11527*, 2018. 3, 6, 7

[45] Li Yi, Hao Su, Xingwen Guo, and Leonidas Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Computer Vision and Pattern Recognition (CVPR)*, 2017. 6, 7

[46] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, volume 2, page 7, 2017.

[47] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *arXiv preprint arXiv:1711.06396*, 2017.