

# Pointwise Convolutional Neural Networks

Binh-Son Hua

Minh-Khoi Tran

Sai-Kit Yeung

The University of Tokyo

Singapore University of Technology and Design

## Abstract

Deep learning with 3D data such as reconstructed point clouds and CAD models has received great research interests recently. However, the capability of using point clouds with convolutional neural network has been so far not fully explored. In this paper, we present a convolutional neural network for semantic segmentation and object recognition with 3D point clouds. At the core of our network is pointwise convolution, a new convolution operator that can be applied at each point of a point cloud. Our fully convolutional network design, while being surprisingly simple to implement, can yield competitive accuracy in both semantic segmentation and object recognition task.

## 1. Introduction

Deep learning with 3D data has received great research interests recently, which leads to noticeable advances in typical applications including scene understanding, shape completion, and shape matching. Among these, scene understanding is considered as one of the most important tasks for robots and drones as it can assist exploratory scene navigations. Tasks such as semantic scene segmentation and object recognition are often performed to predict contextual information about objects for both indoor and outdoor scenes.

Unfortunately, deep learning in 3D was deemed difficult due to the fact that there are several ways to represent 3D data such as volumes, point clouds, or multi-view images. Volume representation is a true 3D representation and straightforward to implement but often requires a large amount of memory for data storage. By contrast, multi-view representation is not a true 3D representation but shows promising prediction accuracy as existing pre-trained weights from 2D networks can be utilized. Among such representations, point clouds have been the most flexible as they are compact and

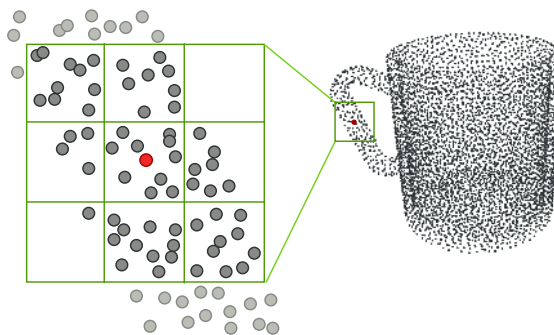


Figure 1: Pointwise convolution. We define a new convolution operator for point cloud input. For each point, nearest neighbors are queried on the fly and binned into kernel cells before convolving with kernel weights. By stacking pointwise convolution operators together, we can build fully convolutional neural networks for scene segmentation and object recognition for point clouds.

could be exported from a wide range of CAD modelling and 3D reconstruction software. However, the capability of using point clouds with neural network has been so far not fully explored.

In this paper, we present a convolutional neural network for semantic segmentation and object recognition with 3D point clouds. At the core of our network is a new convolution operator, called *pointwise convolution*, which can be applied at each point in a point cloud to learn pointwise features. This leads to surprisingly simple and fully convolutional network designs for scene segmentation and object recognition. Our experiments show that pointwise convolution can yield competitive accuracy to previous techniques while being much simpler to implement. In summary, our contributions are:

- A pointwise convolution operator that can output features at each point in a point cloud;
- Two pointwise convolutional neural networks for semantic scene segmentation and object recognition.

<sup>0</sup>This work was done when Binh-Son Hua was a postdoctoral researcher in Singapore University of Technology and Design in 2017.

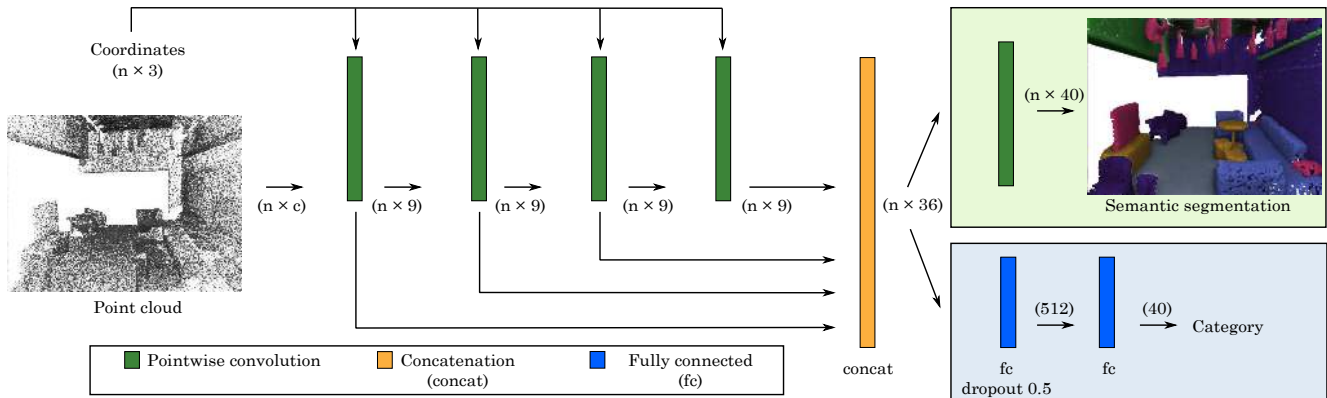


Figure 2: Pointwise convolutional neural network. The input point cloud is fed into each convolution operator, and all outputs are concatenated before being fed to a final convolution layer for dense semantic segmentation, or to fully connected layers for object recognition. In this figure, we assume point cloud with  $n$  points and  $c$  attributes (colors, normals, coordinates, etc.). We use 9 output channels for each convolution operator before concatenation. Source code is available at our homepage [13].

## 2. Related Works

Recently, there has been a great number of works about deep learning with 3D data. Let us focus on those for scene understanding tasks such as semantic segmentation and object recognition.

### 2.1. Shape descriptors

Hand-crafted shape descriptors were widely used in computer vision and graphics applications before the era of deep learning. For example, 3D shapes can be projected into 2D images and represented by a set of 2D descriptors on such images. Shapes can then be represented as histograms or bag-of-feature models which can be constructed from surface normals and curvatures [12]. 3D shapes can also be represented by their inherent statistical properties, such as distance distribution [25] and harmonic descriptors [15]. Heat kernel signatures extract shape descriptions by simulating a heat diffusion process on 3D shapes [38]. The Light Field Descriptor (LFD) is another popular descriptor useful in the shape classification tasks. It extracts geometric and Fourier descriptors from object silhouettes rendered from several different viewpoints [4]. Despite their long history and being widely used, hand-crafted 3D shape descriptors do not generalize well across different domains.

### 2.2. Object recognition

Convolutional neural networks (CNNs) [17] has been successfully applied in various areas of computer vision and artificial intelligence. Recently, significant achievements have been reached in understanding images through learning features by CNNs. Large RGB image datasets like ImageNet [7] can be used in training a CNN, which is in turn able to learn general purpose image descriptors from such datasets. Image descriptors generated by CNNs are proved to greatly

outperform other hand-crafted features for various tasks, including object detection [9], scene recognition [8], texture recognition [31, 6] and classification [10].

Recently, several approaches to using 3D convolutional networks to extract shape descriptor have been proposed, ranging from voxel-based representation [42, 20] panorama [32], feature pooling from 2D projections from multiple viewpoints [37, 27], to point set [27]. Among these, Qi *et al.* [26]’s PointNet is one of the first network architectures that can handle point cloud data. PointNet is robust as it can learn an order-invariance function to canonicalize input point clouds. Subsequently, PointCNN [18] explored the idea of equivariance instead of invariance and demonstrated competitive performance to PointNet. To achieve scalability, it is also possible to learn representations on unstructured point clouds by building computational graphs based on hierarchical data structures such as octree [30] and kd-tree [16].

Despite their competitive performance, network structures based on PointNet [26] are rather complex. In this work, we show that it is possible to perform scene understanding tasks such as semantic segmentation and object recognition on *ordered* point clouds. We design pointwise convolution, a simple convolution operator for 3D point cloud and use it to make (fully) convolutional neural networks for object recognition and semantic segmentation. With the availability of our pointwise convolution, we aim to pave the way towards adapting many existing network architecture designed for scene understanding with color and RGB-D images [34, 19, 35] to the 3D domain.

### 2.3. Semantic segmentation

There are considerably great numbers of related works in semantic segmentation. Since the introduction of the NYUv2

dataset from Silberman *et al.* [33], there has been a spark in the direction of RGBD semantic segmentation. The work from Long *et al.* [19] showed how to adopt a conventional classification network for the semantic segmentation problem. Since then, different techniques have been proposed to further improve the segmentation results. Some notable examples are SegNet [2] which employs an encoder-decoder architecture, or the dilation filter [43].

In the 3D domain, interactive semantic segmentation [40, 39] relied on user strokes to propagate segmentation. McCormac *et al.* [21] explored transferring semantic segmentation from 2D predictions to the 3D domain. An advantage of such methods is that they can produce high-resolution segmentation. However, none of the predictions can be performed directly in the 3D domain.

SSCNet [36] applied convolutional neural network to a 3D volume representation to classify each voxel in the scene. This could be flexible as real-time scene reconstruction techniques such as KinectFusion [23] and voxel hashing [24] are often based on volumes. PointNet [26] can also be used for semantic segmentation with minor modifications from their object recognition network.

Recently, Qi *et al.* [29] proposed to build a graph neural network for semantic segmentation on a point cloud, where each graph node is a group of points and graph edges are constructed by nearest neighbor search on the point cloud. Their results are shown with RGB-D images, where color features from a pre-trained VGG-16 network [34, 5] are used to initialize the prediction. Here, we demonstrate a fully convolutional neural network for 3D point cloud segmentation. Compared to the method by Qi *et al.* [29], we train our network from scratch. The input point cloud is also more general such as CAD models or 3D meshes reconstructed from RGB-D sensors.

### 3. Pointwise Convolution

Before presenting pointwise convolution, we briefly revise a few possibilities to represent 3D data for neural network. The most straightforward approach is perhaps to employ volumetric representation. For example, VoxNet [20] represents each object by a volume up to  $64 \times 64 \times 64$  resolution. This is natural because almost existing network architecture for image applications can be adopted. However, a significant drawback is that volumetric representation requires a large amount of memory while the number of non-zero values in a volume only accounts for a very small percentage. This could be addressed by a sparse representation [30].

A second possibility is to use point clouds. This is a direct representation as point cloud is often the output of many applications such as RGB-D reconstruction and CAD modeling. However, mapping point cloud to neural network is not natural because traditional convolution operators are only

designed for grid and volumes. PointNet [26] implements point feature learning by fully connected layers.

The previous limitations motivate us to design fully convolutional networks for point clouds. The basic building block of our architecture is a convolution operator applied at each point in a point cloud, which we term the *pointwise convolution*. This operator works as follows.

**Convolution.** A convolution kernel is centered at each point of a point cloud. Neighbor points within the kernel support can contribute to the center point. Each kernel has a size or radius value, which can be adjusted to account for different number of neighbor points in each convolution layer. Figure 1 shows a diagram that demonstrates this idea.

Formally, pointwise convolution can be written as

$$x_i^\ell = \sum_k w_k \frac{1}{|\Omega_i(k)|} \sum_{p_j \in \Omega_i(k)} x_j^{\ell-1}, \quad (1)$$

where  $k$  iterates over all sub-domains in the kernel support;  $\Omega_i(k)$  is the  $k$ -th sub-domain of the kernel centered at point  $i$ ;  $p_j$  is the coordinate of point  $j$ ;  $|\cdot|$  counts all points within the sub-domain;  $w_k$  is the kernel weight at the  $k$ -th sub-domain,  $x_i$  and  $x_j$  the value at point  $i$  and  $j$ , and  $\ell - 1$  and  $\ell$  the index of the input and output layer.

**Gradient backpropagation.** To make pointwise convolution trainable, it is necessary to compute the gradients with respects to the input data and the kernel weights. Let  $L$  is the loss function. The gradient with respect to input could be defined as

$$\frac{\partial L}{\partial x_j^{\ell-1}} = \sum_{i \in \Omega_j} \frac{\partial L}{\partial x_i^\ell} \frac{\partial x_i^\ell}{\partial x_j^{\ell-1}} \quad (2)$$

where we iterate over all neighbor points  $i$  of a given point  $j$ . In the chain rule,  $\partial L / \partial x_i^\ell$  is the gradient up to layer  $\ell$ , which is known during back propagation. The derivative  $\partial x_i^\ell / \partial x_j^{\ell-1}$  could be written as

$$\frac{\partial x_i^\ell}{\partial x_j^{\ell-1}} = \sum_k w_k \frac{1}{|\Omega_i(k)|} \sum_{p_j \in \Omega_i(k)} 1 \quad (3)$$

Similarly, the gradient with respect to kernel weights could be defined by iterating over all points  $i$ :

$$\frac{\partial L}{\partial w_k} = \sum_i \frac{\partial L}{\partial x_i^\ell} \frac{\partial x_i^\ell}{\partial w_k} \quad (4)$$

where

$$\frac{\partial x_i^\ell}{\partial w_k} = \frac{1}{|\Omega_i(k)|} \sum_{p_j \in \Omega_i(k)} x_j^{\ell-1} \quad (5)$$

Note that the above formula does not assume a specific shape for convolution kernel. Here we simply use a uniform grid kernel. In conjunction with an acceleration structure for neighbor query, e.g., grid, the convolution operator can be efficiently implemented on both CPU and GPU. In this paper, we use convolution kernels of size  $3 \times 3 \times 3$ . All points within each kernel cell have the same weights.

Unlike convolution in volumes, in our design, we do not use pooling. There are some advantages of doing so. First, it is no longer required to deal with point cloud downsampling and upsampling, which is not straightforward when the point attributes become high dimensional when the point cloud is processed in the network. Second, by keeping the point cloud unchanged in the entire network, acceleration structures for neighbor query only need to be built once. This significantly speeds up computation and simplifies network design.

**Point order.** A notable difference between our design and PointNet [26] is how points are ordered before being fed to the network. In PointNet, point cloud is orderless, and the training process of PointNet learns a symmetric function to turn an ordered point cloud into order invariant. However, we argue that this might not be necessary. In our method, we input points sorted in a specific order, e.g., XYZ or Morton curve [22], to the network and can still achieve competitive performance in the object recognition task. In this task, the order of the points only affects the final global feature vector used to predict the object category. In semantic segmentation, in principle we can leverage local features at each point, and hence point order is not necessary.

**À-trous convolution.** The original pointwise convolution can be easily extended to à-trous convolution by including a stride parameter that determines the gaps between kernel cells. The benefit of pointwise à-trous convolution is that it is possible to extend the kernel size, and hence the perceptive field, without actually processing too many points in the convolution. This yields significant speed up without sacrificing accuracy as to be demonstrated in our experiments.

**Point attributes.** For easy housekeeping in the implementation of our convolution operator, we separately store point coordinates and other point attributes such as colors, normals, or other high-dimensional features output from preceding convolutional layers. Point coordinates can be passed to any layer despite the layer depth so that they can be used for neighbor queries to determine which points can participate in the convolution at a particular point. Point attributes can then be retrieved accordingly.

**Relevance to geometric deep learning.** Our pointwise convolution is relevant to geodesic convolution in geometric

deep learning [3], which is more robust for tasks such as non-rigid shape correspondences and retrieval. To compute a geodesic convolution at a particular point, only neighbor points on its local surface manifold are considered. This is achieved by definition because the filter support in geodesic convolution is directly defined on the surface manifold. By contrast, our pointwise convolution operates adaptively in the 3D Euclidean space, and does not require any surface definition to operate.

## 4. Evaluations

**Semantic segmentation.** We evaluate our pointwise convolutional neural network with semantic scene segmentation and object recognition. For scene segmentation, we first experiment with the S3DIS dataset [1], which has 13 categories of indoor scene objects. Each point has 9 attributes: XYZ coordinates, RGB color, and normalized coordinates w.r.t. the room space it belongs to. To perform segmentation of a scene, each squared-meter block of the scene (measured on the floor), sampled to 4096 points, are fed into the network. The predictions of all blocks are then assembled to obtain the prediction of the entire scene.

We report per-point accuracy of the semantic segmentation. As shown in Table 1, our network is able to produce comparable accuracy to PointNet [26], with the accuracy of 81.5%. Table 2 reports per-class accuracy. Figure 3 shows visualization of predictions and ground truths of the scenes in the evaluation dataset.

Network	Accuracy (per class)	Accuracy
PointNet [26]	-	87.0
Ours	56.5	81.5

Table 1: Comparison of scene segmentation on S3DIS dataset [1].

To further test semantic segmentation with more categories and more complex indoor scenes, we annotate 76 scenes from the SceneNN dataset [13] with 40 categories defined by the NYU v2 dataset [33]. Scenes in this dataset appear to be more cluttered, which poses great challenges to semantic segmentation. We use 56 scenes for training, and 20 scenes for evaluation. In each scene, a  $2 \times 2$  sqm. window with stride 0.2 meter and height 2 meters is used to scan the floor area, resulting in approximately 30K scene blocks for training and 15K blocks for testing. Each block is sampled to 4096 points.

For SceneNN dataset, we additionally compare with VoxNet [20], a voxel-based representation technique, and SemanticFusion [21], a multi-view 2D-3D semantic segmentation with RGB-D images. For VoxNet [20], we apply their

Network	ceiling	floor	wall	column	
PointNet [26]	98.3	98.8	83.3	63.4	
Ours	97.4	99.1	89.1	56.2	
	door	table	chair	sofa	clutter
PointNet [26]	84.6	70.3	66.0	56.7	69.0
Ours	62.9	73.7	68.4	54.6	65.2

Table 2: Per-class accuracy of semantic segmentation on S3DIS dataset [1].

network to predict labels of scene blocks as described above and gather all outputs into a final scene prediction. For SemanticFusion [21], we perform 2D semantic segmentation on the RGB-D images independently and then integrate all 2D predictions to a 3D point cloud to generate the final segmentation.

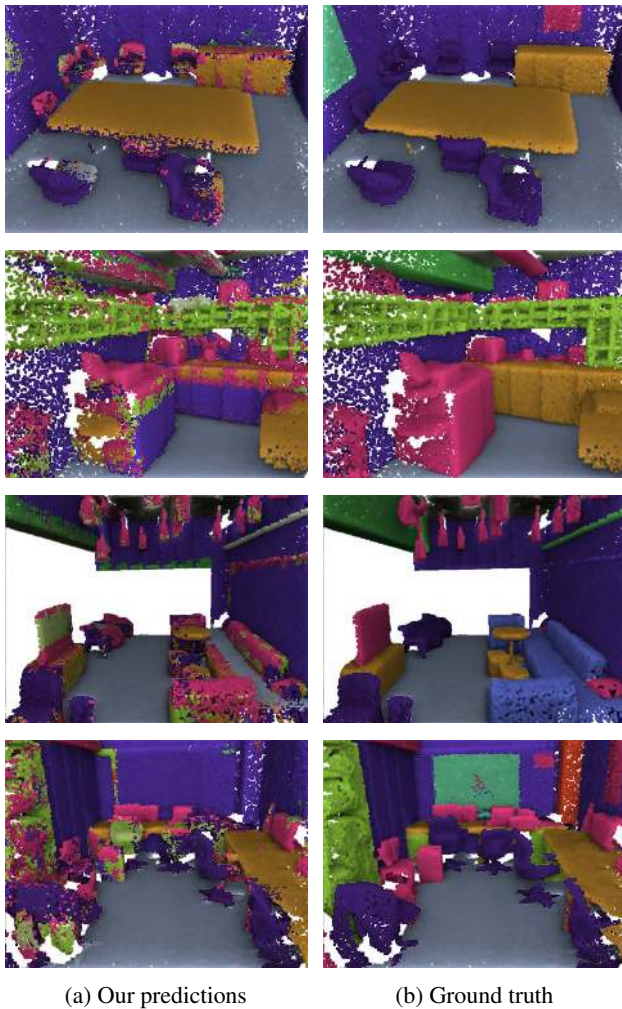


Figure 3: Semantic segmentation on the S3DIS dataset [1].

The visualization of the predictions and ground truth are shown in Figure 4. It can be seen that structures like wall and floor have very good accuracy, and small objects are moderately well segmented. A notable issue is noise due to prediction inconsistency in the overlap regions of the blocks. This could be addressed by a conditional random field and would be an interesting future work.

Table 3 reports the accuracy of a few common categories. While structures and chairs are quite accurate, table and desk are often ambiguous, resulting in lower accuracy for both classes. In general, the performance of VoxNet [20] is inferior to ours and SemanticFusion [21] due to limited resolution (we used  $64^3$  volume). Our method works competitively to SemanticFusion, but note that our method does not apply any label smoothing while SemanticFusion has a conditional random field to remove noise after propagating predictions from 2D to 3D.

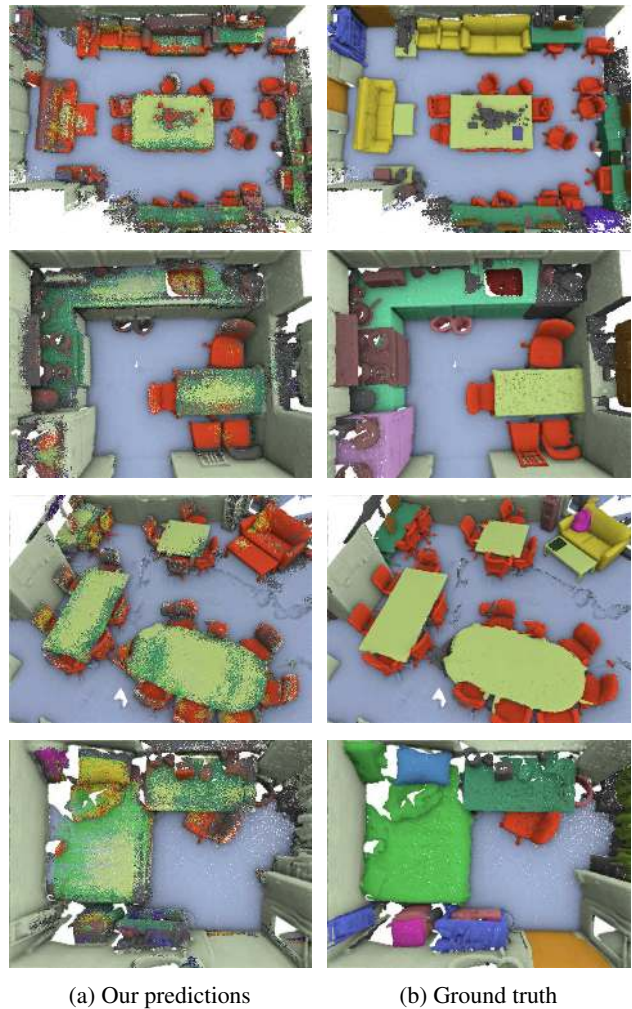


Figure 4: Semantic segmentation on SceneNN dataset [13].

Network	wall	floor	chair	table	desk
VoxNet [20]	82.8	74.3	3.1	0.8	5.4
SemanticFusion [21]	72.8	94.4	46.3	70.1	28.1
Ours	93.8	88.6	58.6	23.5	29.5

Table 3: Per-class accuracy of semantic segmentation on SceneNN dataset [13].

**Object recognition.** We evaluate object recognition with two datasets, ModelNet40 [42] and ObjectNN [14]. ModelNet40 is a CAD model dataset of 40 categories which has served as a standard benchmark for object recognition in the recent years. On the other hand, ObjectNN is an object dataset from RGB-D scene reconstruction mixed with CAD models for studying 3D object retrieval. Objects in ObjectNN is particularly difficult to classify because they are reconstructed from noisy RGB-D data and often has missing parts.

For object recognition, our point attributes are simple XYZ coordinates. In fact, we also trained the network with point attributes set to one, making the convolution equivalent to density estimation, and found no significant change in accuracy. Our results on ModelNet40 are shown in Table 4. As can be seen, our network performs comparably to state-

Network	Accuracy (per class)	Accuracy
VoxNet [20]	83	-
MO-SubvolumeSup [27]	86	89.2
PointNet [26]	86.2	89.2
PointNet++ [28]	-	90.7
Ours	81.4	86.1

Table 4: Comparison of performance of network architectures using 3D object representations on the ModelNet40 dataset [42].

of-the-art methods. Note that compared to VoxNet [20], our input point cloud is more compact. Our network is also significantly simpler in design compared to PointNet [26] and PointNet++ [28] while being close to their accuracy.

The results on ObjectNN are shown in Table 5. In this dataset, again our method performs comparably to PointNet,

Network	Accuracy (per class)	Accuracy
PointNet [26]	57.1	65.6
Ours	57.1	65.1

Table 5: Comparison of object recognition accuracy on the ObjectNN dataset [14].

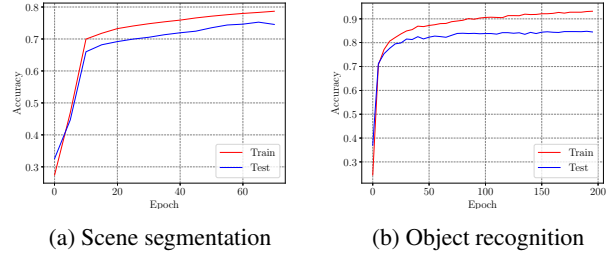


Figure 5: Train and test accuracy over time.

but overall both methods are less effective due to the ambiguity in learning features from both CAD models and RGB-D objects.

Table 9 and Table 10 further provide per-class accuracy on the ModelNet40 and the ObjectNN dataset, respectively.

**Convergence.** Figure 5 shows a plot of the training and test accuracy of our networks over time. The graph shows that our pointwise convolutional neural network can be trained effectively.

**Ablation experiments.** Here we analyze the effectiveness of pointwise convolution. We first start with with a basic 4-layer model as in Figure 2. The accuracy improvement when more features are added are presented in Table 6. As can be seen, feature concatenation, à-trous convolution, SELU activation, and dropout each contributes a small improvement to the final result.

Base	Concat.	À-trous	SELU	Dropout	Accuracy
✓					78.6
✓	✓				78.0
✓		✓			75.0
✓	✓	✓			82.5
✓			✓		81.7
✓	✓		✓		81.9
✓	✓		✓	✓	85.2
✓	✓	✓	✓	✓	86.1

Table 6: Ablation experiment. Accuracy improvement is achieved when pointwise convolution is combined with feature concatenation (Concat.), à-trous convolution, self-normalization activation function (SELU), and dropout.

**Point order.** In object recognition, the order of the input points determine the orders of the features in the fully connected layers. As long as this layer has an order, it is sufficient to discriminate their features and predict the categories. We experiment with different orders of the input

point set and report the results in Table 7(a). We found that point orders sorted by space filling curve techniques such as Morton curve [22] yields comparable accuracy, which means that it is sufficient to just follow an order, but not a particular one. However, a benefit is that space filling curves organize points such that nearby points in space are stored close to each other in memory, allowing more memory coherence.

**Neighborhood radius.** So far we have been setting the radius for neighbor query as constant in each convolution layer. In our experience, this works well for both tasks. We also explore the capability of adaptive radius using k-nearest neighbors. The modification for the convolution operator is as follows.

At each point, a k-nearest neighbor is performed, and the query radius is set to the distance to the furthest neighbor. This radius is used each time neighbor points have to be queried for convolution. To compute gradients for backpropagation for this operator, it is worth noting that in this case, neighbor lookup is no longer symmetric. Therefore, at a point  $j$ , it is required to look up all points  $i$  such that point  $i$  can contribute to point  $j$  in the forward convolution.

We compare the performance of the k-nearest neighbor and the fixed radius convolution for object recognition task. The result is shown in Table 7(b). In general, we found no significant difference in terms of accuracy.

Order	Accuracy	Neighbor query	Accuracy
ZYX	86.1	Fixed-size radius	86.1
Morton	86.0	K-nearest neighbor	85.7

(a) (b)

Table 7: (a) Object recognition with different ways of ordering the input point cloud. (b) Object recognition with convolution using neighbor queries with adaptive radius.

**Deeper networks.** Finally, we study the capability of learning with deeper networks using pointwise convolution. From the basic model, we increase the number of layers from 4 to 8 and 16, and then retrain from scratch. The performance are reported in Table 8 below. Generally, it takes

Network	Accuracy
4 layers	86.1
8 layers	82.1
16 layers	82.6

Table 8: Deep pointwise convolutional neural network. We compare object recognition performance with 4-, 8-, and 16-layer architecture.

longer to train networks with 8 and 16 layers, resulting in slightly slower accuracy. Experimenting the training with residual learning [11] would be an interesting future work.

**Running time.** A key challenge when implementing pointwise convolution is how to perform fast nearest neighbor query without impacting too much the network training and prediction time. To make the training feasible, we choose to use a grid for neighbor query because this is a lightweight and GPU-friendly data structure to build and query *on the fly*. In fact, we experimented with kd-tree, but found that on modern CPUs and GPUs, a kd-tree query does not outperform a grid unless the number of points are more than 16K points, not to mention extra time needed for tree construction that has  $O(n \log n)$  complexity.

Our pointwise convolution is currently implemented with Tensorflow. We report the running time, including grid build and query each time convolution is invoked, as follows. For a batch size of 128 point clouds, each with 2048 points, a forward convolution of our network takes 1.272 seconds on an Intel Core i7 6900K with 16 threads, and a backward propagation takes 2.423 seconds to compute the gradients. Our GPU implementation on an NVIDIA TITAN X can further improve the running time for about 10%. Compared to PointNet [26] and VoxNet [20] which leverage Tensorflow’s optimized convolution operators, our pointwise convolution is not yet engineering optimized. Our training time is about  $2\times$  slower which we currently compensate by using multiple CPUs and GPUs.

## 5. Conclusion

In this paper, we proposed pointwise convolution and leveraged it to build convolutional neural networks for scene understanding with point cloud data. We demonstrated two scene understanding applications including scene segmentation and object recognition. We showed that it is practical to simply sort input point clouds in a specific order before feature learning for object classification. Our pointwise convolution can offer competitive accuracy while being simple to implement, allowing us to create effective and simple neural networks for learning local features of point clouds.

There are several research avenues to be further explored. For example, finding a robust solution to handle large-scale point clouds for scene understanding would be an interesting future work. Currently, we just circumvent the large-scale issue in semantic segmentation by simply dividing the scene into blocks and resample each block to fixed number of points for prediction. In addition, it would be of great interest to extend pointwise convolutional neural networks to geometry point cloud processing [44], or explore the connection of pointwise convolution to tensor voting [41], which was used in the literature to detect structures in a local point neighborhood.

Network	airplane	bathtub	bed	bench	bookshelf	bottle	bowl	car	chair	cone
PointNet [26]	100	80.0	94.0	75.0	93.0	94.0	100.0	97.9	96.0	100.0
Ours	100	82.0	93.0	68.4	91.8	93.9	95.0	95.6	96.0	80.0
	cup	curtain	desk	door	dresser	flower pot	glass box	guitar	keyboard	lamp
PointNet [26]	70.0	90.0	79.0	95.0	65.1	30.0	94.0	100.0	100.0	90.0
Ours	60.0	80.0	76.7	75.0	67.4	10.0	80.8	98.0	100.0	83.3
	laptop	mantel	monitor	night stand	person	piano	plant	radio	range hood	sink
PointNet [26]	100.0	96.0	95.0	82.6	85.0	88.8	73.0	70.0	91.0	80.0
Ours	95.0	93.9	92.9	70.2	89.5	84.5	78.8	65.0	88.9	65.0
	sofa	stairs	stool	table	tent	toilet	tv stand	vase	wardrobe	xbox
PointNet [26]	96.0	85.0	90.0	88.0	95.0	99.0	87.0	78.8	60.0	70.0
Ours	96.0	80.0	83.3	90.9	90.0	94.9	84.5	81.3	30.0	75.0

Table 9: Per-class accuracy of object recognition on the ModelNet40 dataset. Average: PointNet: 86.3. Ours 81.4.

Network	chair	display	desk	book	storage	box	table	bin	bag	keyboard
PointNet [26]	84.2	85.4	56.7	30.1	62.5	23.8	80.0	75.0	47.4	82.4
Ours	83.1	85.4	70.0	57.7	45.8	23.8	60.0	65.0	36.8	88.2
	sofa	bookshelf	pillow	machine	pc case	light	oven	cup	printer	bed
PointNet [26]	76.5	23.1	84.6	18.2	36.4	77.8	60.0	37.5	50.0	28.6
Ours	88.2	38.5	76.9	18.2	54.5	88.9	30.0	75.0	12.5	42.9

Table 10: Per-class accuracy of object recognition on the ObjectNN dataset. Average: PointNet: 56.0. Ours: 57.1.

## A. Layer Visualization

Intuitively, pointwise convolution works by summarizing local spatial point distributions to build feature vectors for each point in a point cloud. As shown in per-class accuracy tables, local features work the most effectively in classifying structures such as ceiling, floor, or walls and common furniture such as tables and chairs. In our observation, it is quite challenging to differentiate between tables (for dining) and desks (for study and work).

We visualize the filters of the first four layers in the object recognition network in Figure 6. Here we display each  $3 \times 3 \times 3$  filter on a row in the visualization. The number of rows is equal to the product of the total number of input and output channels of each filter (27 for the first layer, and 81 for the subsequent layers). In the visualization, blue and red represent positive and negative values, respectively. White represents zero. This shows that the filters in the network are relatively sparse and smooth. We also observed that positive and negative values dominate the filters interchangeably in each layer.

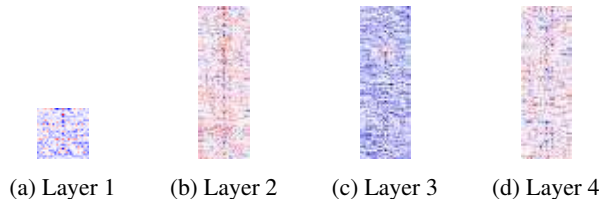


Figure 6: Visualization of the filters in pointwise convolution network for object recognition.

**Acknowledgement.** We thank Quang-Hieu Pham for helping with the 2D-to-3D semantic segmentation experiment and proofreading the paper, Quang-Trung Truong and Benjamin Kang Yue Sheng for their kind support for the neural network training experiments.

Binh-Son Hua and Sai-Kit Yeung are supported by the SUTD Digital Manufacturing and Design Centre which is supported by the Singapore National Research Foundation (NRF). Sai-Kit Yeung is also supported by Singapore MOE Academic Research Fund MOE2016-T2-2-154, Heritage Research Grant of the National Heritage Board, Singapore, and Singapore NRF under its IDM Futures Funding Initiative and Virtual Singapore Award No. NRF2015VSGAA3DCM001-014.



## References

- [1] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. In *CVPR*, 2016. 4, 5
- [2] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv:1511.00561*, 2015. 3
- [3] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *arXiv:1611.08097*, 2016. 4
- [4] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*, volume 22, pages 223–232. Wiley Online Library, 2003. 2
- [5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv:1606.00915*, 2016. 3
- [6] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *CVPR*, 2014. 2
- [7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 2
- [8] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning*, 2014. 2
- [9] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 2
- [10] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *CVPR*, 2015. 2
- [11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 7
- [12] B. K. P. Horn. Extended gaussian images. *Proceedings of the IEEE*, 72(12):1671–1686, 1984. 2
- [13] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung. Scenenn: A scene meshes dataset with annotations. In *International Conference on 3D Vision (3DV)*, 2016. <http://www.scenenn.net>. 2, 4, 5, 6
- [14] B.-S. Hua, Q.-T. Truong, M.-K. Tran, Q.-H. Pham, A. Kanazaki, T. Lee, H. Chiang, W. Hsu, B. Li, Y. Lu, et al. Shrec17: Rgb-d to cad retrieval with objectnn dataset. 6
- [15] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3 d shape descriptors. In *Symposium on geometry processing*, volume 6, pages 156–164, 2003. 2
- [16] R. Klokov and V. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *ICCV*, 2017. 2
- [17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2
- [18] Y. Li, R. Bu, M. Sun, and B. Chen. Pointcnn. *arXiv:1801.07791*, 2018. 2
- [19] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 2, 3
- [20] D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *IROS*, 2015. 2, 3, 4, 5, 6, 7
- [21] J. McCormac, A. Handa, A. Davison, and S. Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. *arXiv:1609.05130*, 2016. 3, 4, 5, 6
- [22] G. M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966. 4, 7
- [23] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *The IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011. 3
- [24] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*, 2013. 3
- [25] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Shape distributions. *ACM Transactions on Graphics (TOG)*, 21(4):807–832, 2002. 2
- [26] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CVPR*, 2017. 2, 3, 4, 5, 6, 7, 8
- [27] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *CVPR*, 2016. 2, 6
- [28] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv:1706.02413*, 2017. 6
- [29] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun. 3d graph neural networks for RGBD semantic segmentation. In *ICCV*, 2017. 3
- [30] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *CVPR*, 2017. 2, 3
- [31] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014. 2
- [32] B. Shi, S. Bai, Z. Zhou, and X. Bai. Deeppano: Deep panoramic representation for 3-d shape recognition. *Signal Processing Letters, IEEE*, 22(12):2339–2343, 2015. 2
- [33] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 3, 4
- [34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014. 2, 3
- [35] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *CVPR*, 2015. 2
- [36] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic scene completion from a single depth image. *CVPR*, 2017. 3

- [37] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*, 2015. [2](#)
- [38] J. Sun, M. Ovsjanikov, and L. Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, volume 28, pages 1383–1392. Wiley Online Library, 2009. [2](#)
- [39] D. Thanh Nguyen, B.-S. Hua, L.-F. Yu, and S.-K. Yeung. A robust 3d-2d interactive tool for scene segmentation and annotation. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2017. [3](#)
- [40] J. Valentin, V. Vineet, M.-M. Cheng, D. Kim, J. Shotton, P. Kohli, M. Nießner, A. Criminisi, S. Izadi, and P. Torr. Semanticpaint: Interactive 3d labeling and learning at your fingertips. *ACM Transactions on Graphics*, 2015. [3](#)
- [41] T.-P. Wu, S. K. Yeung, J. Jia, C.-K. Tang, and G. G. Medioni. A closed-form solution to tensor voting: Theory and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012. [7](#)
- [42] Z. Wu, S. Song, A. Khosla, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. [2](#), [6](#)
- [43] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016. [3](#)
- [44] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Pu-net: Point cloud upsampling network. In *CVPR*, 2018. [7](#)