SO-Net: Self-Organizing Network for Point Cloud Analysis

Jiaxin Li Ben M. Chen Gim Hee Lee
National University of Singapore

Abstract

This paper presents SO-Net, a permutation invariant architecture for deep learning with orderless point clouds. The SO-Net models the spatial distribution of point cloud by building a Self-Organizing Map (SOM). Based on the SOM, SO-Net performs hierarchical feature extraction on individual points and SOM nodes, and ultimately represents the input point cloud by a single feature vector. The receptive field of the network can be systematically adjusted by conducting point-to-node k nearest neighbor search. In recognition tasks such as point cloud reconstruction, classification, object part segmentation and shape retrieval, our proposed network demonstrates performance that is similar with or better than state-of-the-art approaches. In addition, the training speed is significantly faster than existing point cloud recognition networks because of the parallelizability and simplicity of the proposed architecture. Our code is available at the project website.[1]

1. Introduction

After many years of intensive research, convolutional neural networks (ConvNets) is now the foundation for many state-of-the-art computer vision algorithms, *e.g.* image recognition, object classification and semantic segmentation etc. Despite the great success of ConvNets for 2D images, the use of deep learning on 3D data remains a challenging problem. Although 3D convolution networks (3D ConvNets) can be applied to 3D data that is rasterized into voxel representations, most computations are redundant because of the sparsity of most 3D data. Additionally, the performance of naive 3D ConvNets is largely limited by the resolution loss and exponentially growing computational cost. Meanwhile, the accelerating development of depth sensors, and the huge demand from applications such as autonomous vehicles make it imperative to process 3D data efficiently. Recent availability of 3D datasets including ModelNet [37], ShapeNet [8], 2D-3D-S [2] adds on to the popularity of research on 3D data.

[1] https://github.com/lijx10/SO-Net

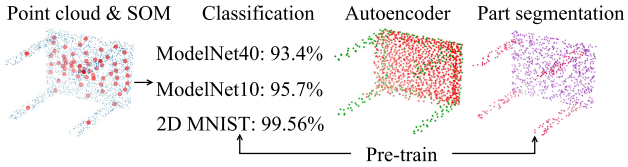


Figure 1. Our SO-Net applies hierarchical feature aggregation using SOM. Point clouds are converted into SOM node features and a global feature vector that can be applied to classification, autoencoder reconstruction, part segmentation and shape retrieval etc.

To avoid the shortcomings of naive voxelization, one option is to explicitly exploit the sparsity of the voxel grids [35, 21, 11]. Although the sparse design allows higher grid resolution, the induced complexity and limitations make it difficult to realize large scale or flexible deep networks [30]. Another option is to utilize scalable indexing structures including kd-tree [4], octree [25]. Deep networks based on these structures have shown encouraging results. Compared to tree based structures, point cloud representation is mathematically more concise and straight-forward because each point is simply represented by a 3-vector. Additionally, point clouds can be easily acquired with popular sensors such as the RGB-D cameras, LiDAR, or conventional cameras with the help of the Structure-from-Motion (SfM) algorithm. Despite the widespread usage and easy acquisition of point clouds, recognition tasks with point clouds still remain challenging. Traditional deep learning methods such as ConvNets are not applicable because point clouds are spatially irregular, and can be permutated arbitrarily. Due to these difficulties, few attempts has been made to apply deep learning techniques directly to point clouds until the very recent PointNet [26].

Despite being a pioneer in applying deep learning to point clouds, PointNet is unable to handle local feature extraction adequately. PointNet++ [28] is later proposed to address this problem by building a pyramid-like feature aggregation scheme, but the point sampling and grouping strategy in [28] does not reveal the spatial distribution of the input point cloud. Kd-Net [18] build a kd-tree for the input point cloud, followed by hierarchical feature extractions from the leaves to root. Kd-Net explicitly utilizes the spatial distri-

bution of point clouds, but there are limitations such as the lack of overlapped receptive fields.

In this paper, we propose the SO-Net to address the problems in existing point cloud based networks. Specifically, a SOM [19] is built to model the spatial distribution of the input point cloud, which enables hierarchical feature extraction on both individual points and SOM nodes. Ultimately, the input point cloud can be compressed into a single feature vector. During the feature aggregation process, the receptive field overlap is controlled by performing point-to-node k-nearest neighbor (kNN) search on the SOM. The SO-Net theoretically guarantees invariance to the order of input points, by the network design and our permutation invariant SOM training. Applications of our SO-Net include point cloud based classification, autoencoder reconstruction, part segmentation and shape retrieval, as shown in Fig. 1.

The **key contributions** of this paper are as follows:

- We design a permutation invariant network - the SO-Net that explicitly utilizes the spatial distribution of point clouds.
- With point-to-node kNN search on SOM, hierarchical feature extraction is performed with systematically adjustable receptive field overlap.
- We propose a point cloud autoencoder as pre-training to improve network performance in various tasks.
- Compared with state-of-the-art approaches, similar or better performance is achieved in various applications with significantly faster training speed.

2. Related Work

It is intuitive to represent 3D shapes with voxel grids because they are compatible with 3D ConvNets. [24, 37] use binary variable to indicate whether a voxel is occupied or free. Several enhancements are proposed in [27] - overfitting is mitigated by predicting labels from partial subvolumes, orientation pooling layer is designed to fuse shapes with various orientations, and anisotropic probing kernels are used to project 3D shapes into 2D features. Brock *et al*. [6] propose to combine voxel based variational autoencoders with object recognition networks. Despite its simplicity, voxelization is able to achieve state-of-the-art performance. Unfortunately, it suffers from loss of resolution and the exponentially growing computational cost. Sparse methods [35, 21, 11] are proposed to improve the efficiency. However, these methods still rely on uniform voxel grids and experience various limitations such as the lack of parallelization capacity [21]. Spectral ConvNets [23, 5, 7] are explored to work on non-Euclidean geometries, but they are mostly limited to manifold meshes.

Rendering 3D data into multi-view 2D images turns the 3D problem into a 2D problem that can be solved using standard 2D ConvNets. View-pooling layer [33] is designed to aggregate features from multiple rendered images. Qi *et al*. [27] substitute traditional 3D to 2D rendering with multi-resolution sphere rendering. Wang *et al*. [34] further propose the dominant set pooling and utilize features like color and surface normal. Despite the improved efficiency compared to 3D ConvNets, multi-view strategy still suffers from information loss [18] and it cannot be easily extended to tasks like per-point labeling.

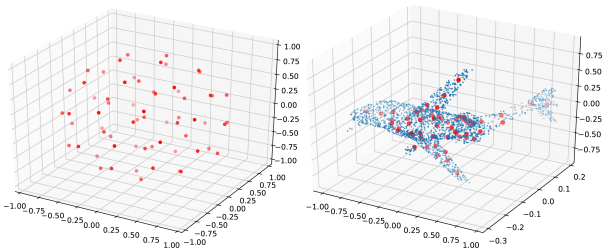
Indexing techniques such as kd-tree and octree are scalable compared to uniform grids, and their regular structures are suitable for deep learning techniques. To enable convolution and pooling operations over octree, Riegler *et al*. [30] build a hybrid grid-octree structure by placing several small octrees into a regular grid. With bit string representation, a single voxel in the hybrid structure is fully determined by its bit index. As a result, simple arithmetic can be used to visit the parent or child nodes. Similarly, Wang *et al*. [36] introduce a label buffer to find correspondence of octants at various depths. Klokov *et al*. propose the Kd-Net [18] that computes vectorial representations for each node of the prebuilt balanced kd-tree. A parent feature vector is computed by applying non-linearity and affine transformation on its two child feature vectors, following the bottom-up fashion.

PointNet [26] is the pioneer in the direct use of point clouds. It uses the channel-wise max pooling to aggregate per-point features into a global descriptor vector. PointNet is invariant to order permutation of input points because the per-point feature extraction is identical for every point and max pooling operation is permutation invariant. A similar permutation equivariant layer [29] is also proposed at almost the same time as [26], with the major difference that the permutation equivariant layer is max-normalized. Although the max-pooling idea is proven to be effective, it suffers from the lack of ConvNet-like hierarchical feature aggregation. PointNet++ [28] is later designed to group points into several groups in different levels, so that features from multiple scales could be extracted hierarchically.

Unlike networks based on octree or kd-tree, the spatial distribution of points is not explicitly modeled in PointNet++. Instead, heuristic grouping and sampling schemes, *e.g*. multi-scale and multi-resolution grouping, are designed to combine features from multiple scales. In this paper, we propose our SO-Net that explicitly models the spatial distribution of input point cloud during hierarchical feature extraction. In addition, adjustable receptive field overlap leads to more effective local feature aggregation.

3. Self-Organizing Network

The input to the network is a point set $P = \{p_i \in \mathbb{R}^3,\ i = 0, \cdots, N-1\}$, which will be processed into M



(a) (b)

Figure 2. (a) The initial nodes of an 8×8 SOM. For each SOM configuration, the initial nodes are fixed for every point cloud. (b) Example of a SOM training result.

SOM nodes $S = \{s_j \in \mathbb{R}^3,\ j = 0, \cdots, M-1\}$ as shown in Sec. 3.1. Similarly, in the encoder described in Sec. 3.2, individual point features are max-pooled into M node features, which can be further aggregated into a global feature vector. Our SO-Net can be applied to various computer vision tasks including classification, per-point segmentation (Sec. 3.3), and point cloud reconstruction (Sec. 3.4).

3.1. Permutation Invariant SOM

SOM is used to produce low-dimensional, in this case two-dimensional, representation of the input point cloud. We construct a SOM with the size of $m \times m$, where $m \in [5, 11]$, i.e. the total number of nodes M ranges from 25 to 121. SOM is trained with unsupervised competitive learning instead of the commonly used backpropagation in deep networks. However, naive SOM training schemes are not permutation invariant for two reasons: the training result is highly related to the initial nodes, and the per-sample update rule depends on the order of the input points.

The first problem is solved by assigning fixed initial nodes for any given SOM configuration. Because the input point cloud is normalized to be within $[-1, 1]$ in all three axes, we generate a proper initial guess by dispersing the nodes uniformly inside a unit ball, as shown in Fig. 2(a). Simple approaches such as the potential field can be used to construct such a uniform initial guess. To solve the second problem, instead of updating nodes once per point, we perform one update after accumulating the effects of all the points. This batch update process is deterministic [19] for a given point cloud, making it permutation invariant. Another advantage of batch update is the fact that it can be implemented as matrix operations, which are highly efficient on GPU. Details of the initialization and batch training algorithms can be found in our supplementary material.

3.2. Encoder Architecture

As shown in Fig. 3, SOM is a guide for hierarchical feature extraction, and a tool to systematically adjust the receptive field overlap. Given the output of the SOM, we search for the k nearest neighbors (kNN) on the SOM nodes S for each point p_i, i.e., point-to-node kNN search:

$$s_{ik} = \text{kNN}(p_i \mid s_j,\ j = 0, \cdots, M-1). \tag{1}$$

Each p_i is then normalized into k points by subtraction with its associated nodes:

$$p_{ik} = p_i - s_{ik}. \tag{2}$$

The resulting kN normalized points are forwarded into a series of fully connected layers to extract individual point features. There is a shared fully connected layer on each level l, where ϕ is the non-linear activation function. The output of level l is given by

$$p_{ik}^{l+1} = \phi(W^l p_{ik}^l + b^l). \tag{3}$$

The input to the first layer p_{ik}^0 can simply be the normalized point coordinates p_{ik}, or the combination of coordinates and other features like surface normal vectors.

Node feature extraction begins with max-pooling the kN point features into M node features following the above kNN association. We apply a channel-wise max pooling operation to get the node feature s_j^0 for those point features associated with the same node s_j:

$$s_j^0 = \max(\{p_{ik}^l, \forall s_{ik} = s_j\}). \tag{4}$$

Since each point is normalized into k coordinates according to the point-to-node kNN search, it is guaranteed that the receptive fields of the M max pooling operations are overlapped. Specifically, M nodes cover kN normalized points. k is an adjustable parameter to control the overlap.

Each node feature produced by the above max pooling operation is further concatenated with the associated SOM node. The M augmented node features are forwarded into a series of shared layers, and then aggregated into a feature vector that represents the input point cloud.

Feature aggregation as point cloud separation and assembly There is an intuitive reason behind the SOM feature extraction and node concatenation. Since the input points to the first layer are normalized with M SOM nodes, they are actually separated into M mini point clouds as shown in Fig. 3. Each mini point cloud contains a small number of points in a coordinate whose origin is the associated SOM node. For a point cloud of size 2048, and $M = 64$ and $k = 3$, a typical mini point cloud may consist of around 90 points inside a small space of $x, y, z \in [-0.3, 0.3]$. The number and coverage of points in a mini point cloud are determined by the SOM training and kNN search, i.e. M and k.

The first batch of fully connected layers can be regarded as a small PointNet that encodes these mini point clouds.

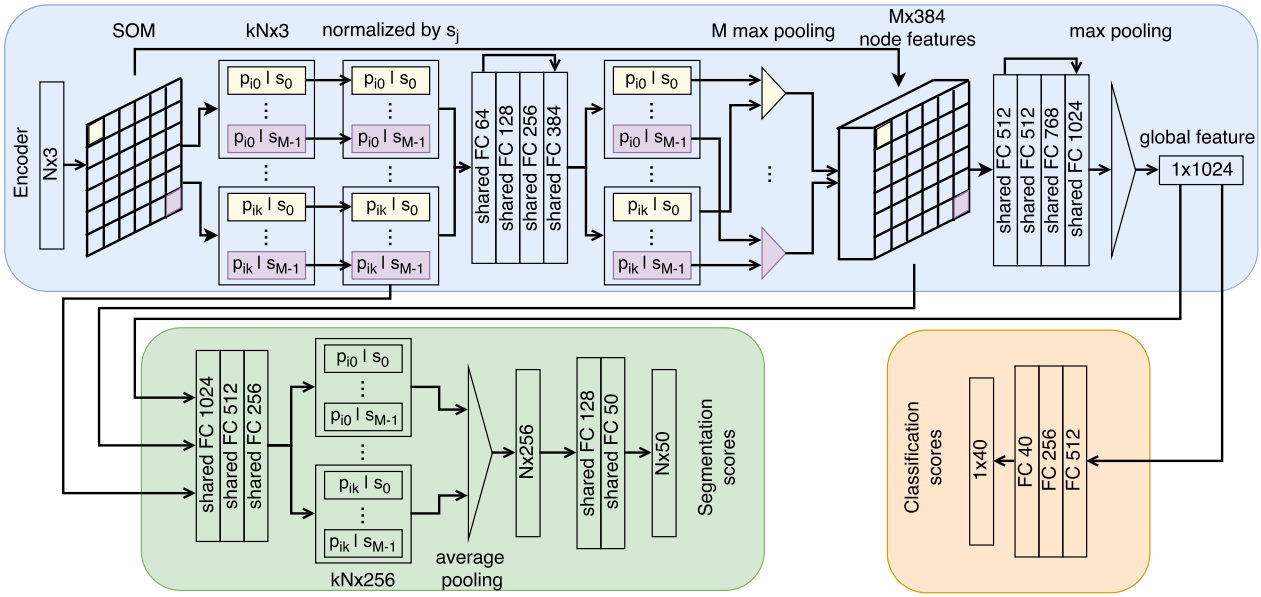


Figure 3. The architecture of the SO-Net and its application to classification and segmentation. In the encoder, input points are normalized with the k-nearest SOM nodes. The normalized point features are later max-pooled into node features based on the point-to-node kNN search on SOM. k determines the receptive field overlap. In the segmentation network, M node features are concatenated with the kN normalized points following the same kNN association. Finally kN features are aggregated into N features by average pooling.

The concatenation with SOM nodes plays the role of assembling these mini point clouds back into the original point cloud. Because the SOM explicitly reveals the spatial distribution of the input point cloud, our separate-and-assemble process is more efficient than the grouping strategy used in PointNet++ [28], as shown in Sec. 4.

Permutation Invariance There are two levels of feature aggregation in SO-Net, from point features to node features, and from node features to global feature vector. The first phase applies a shared PointNet to M mini point clouds. The generation of these M mini point clouds is irrelevant to the order of input points, because the SOM training in Sec. 3.1 and kNN search in Fig. 3 are deterministic. PointNet [26] is permutation invariant as well. Consequently, both the node features and global feature vector are theoretically guaranteed to be permutation invariant.

Effect of suboptimal SOM training It is possible that the training of SOM converges into a local minima with isolated nodes outside the coverage of the input point cloud. In some situations no point will be associated with the isolated nodes during the point-to-node kNN search, and we set the corresponding node features to zero. This phenomenon is quite common because the initial nodes are dispersed uniformly in a unit ball, while the input point cloud may occupy only a small corner. Despite the existence of suboptimal SOM, the proposed SO-Net still out-performs state-of-the-art approaches in applications like object classification. The effect of invalid node features is further investigated in Sec. 4 by inserting noise into the SOM results.

Exploration with ConvNets It is interesting to note that the node feature extraction has generated an image-like feature matrix, which is invariant to the order of input points. It is possible to apply standard ConvNets to further fuse the node features with increasing receptive field. However, the classification accuracy decreased slightly in our experiments, where we replaced the second batch of fully connected layers with 2D convolutions and pooling. It remains as a promising direction to investigate the reason and solution to this phenomenon.

3.3. Extension to Segmentation

The extension to per-point annotations, *e.g.* segmentation, requires the integration of both local and global features. The integration process is similar to the invert operation of the encoder in Sec. 3.2. The global feature vector can be directly expanded and concatenated with the kN normalized points. The M node features are attached to the points that are associated with them during the encoding process. The integration results in kN features that combine point, node and global features, which are then forwarded into a chain of shared fully connected layers.

The kN features are actually redundant to generate N per-point classification scores because of the receptive field overlap. Average or max pooling are methods to fuse the redundant information. Additionally, similar to many deep

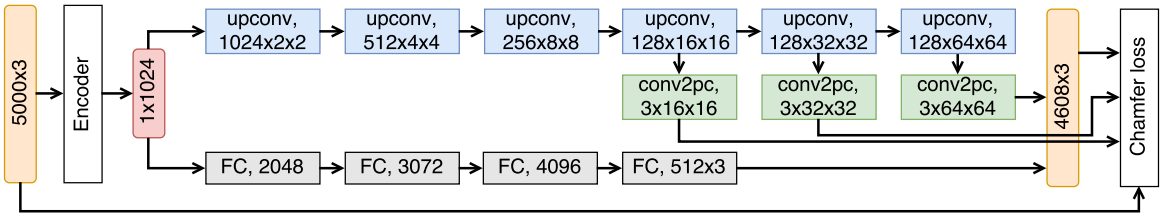


Figure 4. The architecture of the decoder that takes 5000 points and reconstructs 4608 points. The up-convolution branch is designed to recover the main body of the input, while the more flexible fully connected branch is to recover the details. The "upconv" module consists of a nearest neighbor upsampling layer and a 3×3 convolution layer. The "conv2pc" module consists of two 1×1 convolution layers.

networks, early, middle or late fusion may exhibit different performance [9]. With a series of experiments, we found that middle fusion with average pooling is most effective compared to other fusion methods.

3.4. Autoencoder

In this section, we design a decoder network to recover the input point cloud from the encoded global feature vector. A straightforward design is to stack series of fully connected layers on top of the feature vector, and generate an output vector of length $3\hat{N}$, which can be reshaped into $\hat{N} \times 3$. However, the memory and computation footprint will be too heavy if $\hat{N}$ is sufficiently large.

Instead of generating point clouds with fully connected layers [1], we design a network with two parallel branches similar with [13], i.e, a fully connected branch and a convolution branch as shown in Fig. 4. The fully connected branch predicts $\hat{N}_1$ points by reshaping an output of $3\hat{N}_1$ elements. This branch enjoys high flexibility because each coordinate is predicted independently. On the other hand, the convolution branch predicts a feature matrix with the size of $3 \times H \times W$, i.e. $\hat{N}_2 = H \times W$ points. Due to the spatial continuity of convolution layers, the predicted $\hat{N}_2$ point may exhibit more geometric consistency. Another advantage of the convolution branch is that it requires much less parameters compared to the fully connected branch.

Similar to common practice in many depth estimation networks [14, 15], the convolution branch is designed as an up-convolution (upconv) chain in a pyramid style. Instead of deconvolution layers, each upconv module consists of a nearest neighbor upsampling layer and a 3×3 convolution layer. According to our experiments, this design is much more effective than deconvolution layers in the case of point cloud autoencoder. In addition, intermediate upconv products are converted to coarse reconstructed point clouds and compared with the input. The conversion from upconv products to point clouds is a 2-layer 1×1 convolution stack in order to give more flexibility to each recovered point. The coarse-to-fine strategy produces another boost in the reconstruction performance.

To supervise the reconstruction process, the loss function should be differentiable, ready for parallel computation and robust against outliers [13]. Here we use the Chamfer loss:

$$d(P_s, P_t) = \frac{1}{|P_s|} \sum_{x \in P_s} \min_{y \in P_t} \|x - y\|_2 + \frac{1}{|P_t|} \sum_{y \in P_t} \min_{x \in P_s} \|x - y\|_2. \tag{5}$$

where P_s and $P_t \in \mathbb{R}^3$ represents the input and recovered point cloud respectively. The numbers of points in P_s and P_t are not necessarily the same. Intuitively, for each point in P_s, Eq. (5) computes its distance to the nearest neighbor in P_t, and vice versa for points in P_t.

4. Experiments

In this section, the performance of our SO-Net is evaluated in three different applications, namely point cloud autoencoder, object classification and object part segmentation. In particular, the encoder trained in the autoencoder can be used as pre-training for the other two tasks. The encoder structure and SOM configuration remain identical among all experiments without delicate finetuning, except for the 2D MNIST classification.

4.1. Implementation Detail

Our network is implemented with PyTorch on a NVIDIA GTX1080Ti. We choose a SOM of size 8×8 and $k = 3$ in most experiments. We optimize the networks using Adam [17] with an initial learning rate of 0.001 and batch size of 8. For experiments with 5000 or more points as input, the learning rate is decreased by half every 20 epochs, otherwise the learning rate decay is executed every 40 epochs. Generally the networks converge after around 5 times of learning rate decay. Batch-normalization and ReLU activation are applied to every layer.

4.2. Datasets

As a 2D toy example, we adopt the MNIST dataset [20] in Sec. 4.4. For each digit, 512 two-dimensional points are sampled from the non-zero pixels to serve as our input.

Two variants of the ModelNet [37], i.e. ModelNet10 and ModelNet40, are used as the benchmarks for the autoencoder task in Sec. 4.3 and the classification task in Sec. 4.4.

The ModelNet40 contains 13,834 objects from 40 categories, among which 9,843 objects belong to training set and the other 3,991 samples for testing. Similarly, the ModelNet10 is split into 2,468 training samples and 909 testing samples. The original ModelNet provides CAD models represented by vertices and faces. Point clouds are generated by sampling from the models uniformly. For fair comparison, we use the prepared ModelNet10/40 dataset from [28], where each model is represented by 10,000 points. Various sizes of point clouds, *e.g.*, 2,048 or 5,000, can be sampled from the 10k points in different experiments.

Object part segmentation is demonstrated with the ShapeNetPart dataset [38]. It contains 16,881 objects from 16 categories, represented as point clouds. Each object consists of 2 to 6 parts, and in total there are 50 parts in the dataset. We sample fixed size point clouds, *e.g.* 1,024, in our experiments.

Data augmentation Input point clouds are normalized to be zero-mean inside a unit cube. The following data augmentations are applied at training phase: (a) Gaussian noise $\mathcal{N}(0, 0.01)$ is added to the point coordinates and surface normal vectors (if applicable). (b) Gaussian noise $\mathcal{N}(0, 0.04)$ is added to the SOM nodes. (c) Point clouds, surface normal vectors (if applicable) and SOM nodes are scaled by a factor sampled from an uniform distribution $\mathcal{U}(0.8, 1.2)$. Further augmentation like random shift or rotation do not improve the results.

4.3. Point Cloud Autoencoder

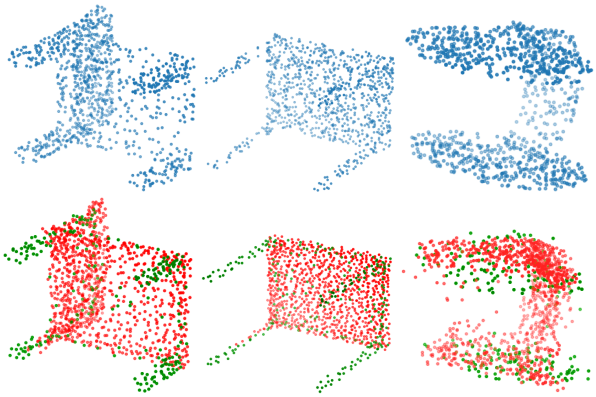


Figure 5. Examples of point cloud autoencoder results. First row: input point clouds of size 1024. Second row: reconstructed point clouds of size 1280. From left to right: chair, table, earphone.

In this section, we demonstrate that a point cloud can be reconstructed from the SO-Net encoded feature vector, *e.g.* a vector with length of 1024. The nearest neighbor search in Chamfer distance (Eq. 5) is conducted with Facebook's faiss [16]. There are two configurations for the decoder to reconstruct different sizes of point clouds. The first configuration generates 64×64 points from the convolution branch and 512 points from the fully connected branch. The other one produces 32×32 and 256 points respectively, by removing the last upconv module of Fig. 4.

It is difficult to provide quantitative comparison for the point cloud autoencoder task because little research has been done on this topic. The most related work is the point set generation network [13] and the point cloud generative models [1]. Examples of our reconstructed ShapeNetPart point clouds are visualized in Fig. 5, where 1024 points recovered from the convolution branch are denoted in red and the other 256 points in green. The overall testing Chamfer distance (Eq. 5) is 0.033. Similar to the results in [13], the convolution branch recovers the main body of the object, while the more flexible fully connected branch focuses on details such as the legs of a table. Nevertheless, many finer details are lost. For example, the reconstructed earphone is blurry. This is probably because the encoder is still not powerful enough to capture fine-grained structures.

Despite the imperfect reconstruction, the autoencoder enhances SO-Net's performance in other tasks by providing a pre-trained encoder, illustrated in Sec. 4.4 and 4.5. More results are visualized in the supplementary materials.

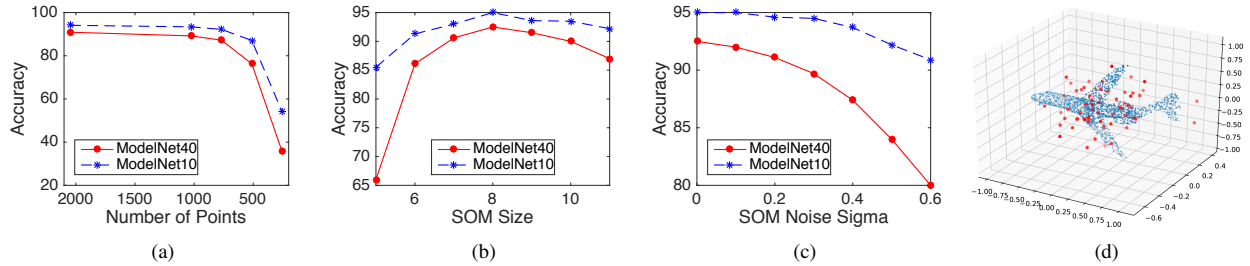
4.4. Classification Tasks

To classify the point clouds, we attach a 3-layer multilayer perceptron (MLP) on top of the encoded global feature vector. Random dropout is applied to the last two layers with keep-ratio of 0.4. Table 1 illustrates the classification accuracy for state-of-the-art methods using scalable 3D representations, such as point cloud, kd-tree and octree. In MNIST dataset, our network achieves a relative 13.7% error rate reduction compared with PointNet++. In ModelNet10 and ModelNet40, our approach out-performs state-of-the-art methods by 1.7% and 1.5% respectively in terms of instance accuracy. Our SO-Net even out-performs single networks using multi-view images or uniform voxel grids as input, like qi-MVCNN [27] (ModelNet40 at 92.0%) and VRN [6] (ModelNet40 at 91.3%). Methods that integrate multiple networks, i.e., qi-MVCNN-MultiRes [27] and VRN Ensemble [6], are still better than SO-Net in ModelNet classification, but their multi-view / voxel grid representations are far less scalable and flexible than our point cloud representation, as illustrated in Sec. 1 and 2.

Effect of pre-training The performance of the network can be improved with pre-training using the autoencoder in Sec. 3.4. The autoencoder is trained with ModelNet40, using 5000 points and surface normal vectors as input. The autoencoder brings a boost of 0.5% in ModelNet10 classification, but only 0.2% in ModelNet40 classification. This is not surprising because pre-training with a much larger

Method	Representation	Input	ModelNet10		ModelNet40			MNIST	
			Class	Instance	Class	Instance	Training	Input	Error rate
PointNet, [26]	points	1024×3	-	-	86.2	89.2	3-6h	256×2	0.78
PointNet++, [28]	points + normal	5000×6	-	-	-	91.9	20h	512×2	0.51
DeepSets, [29, 39]	points	5000×3	-	-	-	90.0	-	-	-
Kd-Net, [18]	points	$2^{15} \times 3$	93.5	94.0	88.5	91.8	120h	1024×2	0.90
ECC, [32]	points	1000×3	90.0	90.8	83.2	87.4	-	-	0.63
OctNet, [30]	octree	128^3	90.1	90.9	83.8	86.5	-	-	-
O-CNN, [36]	octree	64^3	-	-	-	90.6	-	-	-
Ours (2-layer)*	points + normal	5000×6	94.9	95.0	89.4	92.5	3h	-	-
Ours (2-layer)	points + normal	5000×6	94.4	94.5	89.3	92.3	3h	-	-
Ours (2-layer)	points	2048×3	93.9	94.1	87.3	90.9	3h	512×2	**0.44**
Ours (3-layer)	points + normal	5000×6	**95.5**	**95.7**	**90.8**	**93.4**	**3h**	-	-

Table 1. Object classification results for methods using scalable 3D representations like point cloud, kd-tree and octree. Our network produces the best accuracy with significantly faster training speed. * represents pre-training.



(a) (b) (c) (d)

Figure 6. Robustness test on point or SOM corruption. (a) The network is trained with point clouds of size 2048, while there is random point dropout during testing. (b) The network is trained with SOM of size 8×8, but SOMs of various sizes are used at testing phase. (c) Gaussian noise $\mathcal{N}(0, \sigma)$ is added to the SOM during testing. (d) Example of SOM with Gaussian noise $\mathcal{N}(0, 0.2)$.

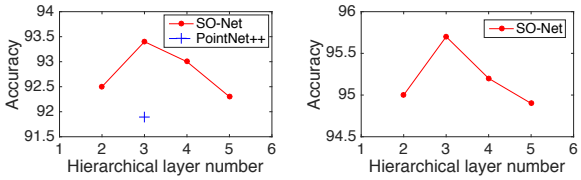


Figure 7. Effect of layer number on classification accuracy with ModelNet40 (left) and ModelNet10 (right).

dataset may lead to convergence basins [12] that are more resistant to over-fitting.

Robustness to point corruption We train our network with point clouds of size 2048 but test it with point dropout. As shown in Fig. 6(a), our accuracy drops by 1.7% with 50% points missing (2048 to 1024), and 14.2% with 75% points missing (2048 to 512). As a comparison, the accuracy of PN drops by 3.8% with 50% points (1024 to 512).

Robustness to SOM corruption One of our major concern when designing the SO-Net is whether the SO-Net relies too much on the SOM. With results shown in Fig. 6, we demonstrate that our SO-Net is quite robust to the noise or corruption of the SOM results. In Fig. 6(b), we train a network with SOM of size 8×8 as the noise-free version, but test the network with SOM sizes varying from 5×5 to 11×11. It is interesting that the performance decay is much slower if the SOM size is larger than training configuration, which is consistent with the theory in Sec. 3.2. The SO-Net separates the input point cloud into M mini point clouds, encodes them into M node features with a mini PointNet, and assembles them during the global feature extraction. In the case that the SOM becomes smaller during testing, the mini point clouds are too large for the mini PointNet to encode. Therefore the network performs worse when the testing SOM is smaller than expected.

In Fig. 6(c), we add Gaussian noise $\mathcal{N}(0, \sigma)$ onto the SOM during testing. Given the fact that input points have been normalized into a unit cube, a Gaussian noise with $\sigma = 0.2$ is rather considerable, as shown in Fig. 6(d). Even in that difficult case, our network achieves the accuracy of 91.1% in ModelNet40 and 94.6% in ModelNet10.

Effect of hierarchical layer number Our framework shown in Fig. 3 can be made to further out-perform state-of-the-art methods by simply adding more layers. The vanilla SO-Net is a 2-layer structure "grouping&PN(PointNet) - PN", where the grouping is based on SOM and point-to-node kNN. We make it a 3-layer structure by simply repeat-

	Intersection over Union (IoU)																
	mean	air	bag	cap	car	chair	ear.	gui.	knife	lamp	lap.	motor	mug	pistol	rocket	skate	table
PointNet [26]	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
PointNet++ [28]	85.1	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
Kd-Net [18]	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
O-CNN + CRF [36]	**85.9**	85.5	87.1	84.7	77.0	91.1	85.1	91.9	87.4	83.3	95.4	56.9	96.2	81.6	53.5	74.1	84.4
Ours (pre-trained)	84.9	82.8	77.8	88.0	77.3	90.6	73.5	90.7	83.9	82.8	94.8	69.1	94.2	80.9	53.1	72.9	83.0
Ours	84.6	81.9	83.5	84.8	78.1	90.8	72.2	90.1	83.6	82.3	95.2	69.3	94.2	80.0	51.6	72.1	82.6

Table 2. Object part segmentation results on ShapeNetPart dataset.

ing the SOM/kNN based "grouping&PN" with this protocol: for each SOM node, find $k' = 9$ nearest nodes and process the k' node features with a PointNet. The output is a new SOM feature map of the same size but larger receptive field. Shown in Table 1, our 3-layer SO-Net increases the accuracy to 1.5% higher (relatively 19% lower error rate) than PN++ on ModelNet40, and 1.7% higher (relatively 28% lower error rate) than Kd-Net on ModelNet10. The effect of hierarchical layer number is illustrated in Fig. 7, where too many layers may lead to over-fitting.

Training speed The batch training of SOM allows parallel implementation on GPU. Moreover, the training of SOM is completely deterministic in our approach, so it can be isolated as data preprocessing before network optimization. Compared to the randomized kd-tree construction in [18], our deterministic design provides great boosting during training. In addition to the decoupled SOM, the hierarchical feature aggregation based on SOM can be implemented efficiently on GPU. As shown in Table 1, it takes about 3 hours to train our best network on ModelNet40 with a GTX1080Ti, which is significantly faster than state-of-the-art networks that can provide comparable performance.

4.5. Part Segmentation on ShapeNetPart

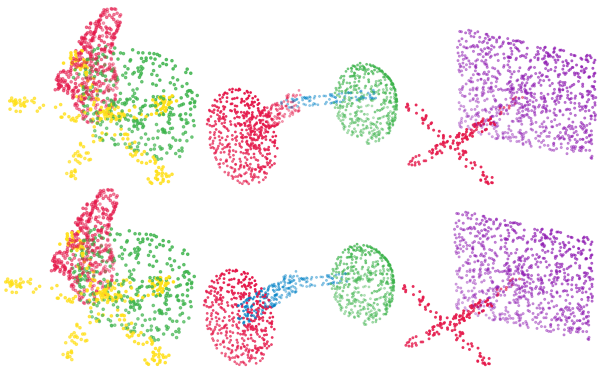


Figure 8. Visualization of object part segmentation results. First row: ground truth. Second row: predicted segmentation. From left to right: chair, lamp, table.

We formulate the object part segmentation problem as a per-point classification task, as illustrated in Fig. 3. The networks are evaluated using the mean Intersection over Union (IoU) protocol proposed in [26]. For each instance, IoU is computed for each part that belongs to that object category. The mean of the part IoUs is regarded as the IoU for that instance. Overall IoU is calculated as the mean of IoUs over all instances, and category-wise IoU is computed as an average over instances under that category. Similar with O-CNN [36] and PointNet++ [28], surface normal vectors are fed into the network together with point coordinates.

By optimizing per-point softmax loss functions, we achieve competitive results as reported in Table 2. Although O-CNN reports the best IoU, it adopts an additional dense conditional random field (CRF) to refine the output of their network while others do not contain this post-processing step. Some segmentation results are visualized in Fig. 8 and we further visualize one instance per category in the supplementary material. Although in some hard cases our network may fail to annotate the fine-grained details correctly, generally our segmentation results are visually satisfying. The low computation cost remains as one of our advantages. Additionally, pre-training with our autoencoder produces a performance boost, which is consistent with our classification results.

5. Conclusion

In this paper, we propose the novel SO-Net that performs hierarchical feature extraction for point clouds by explicitly modeling the spatial distribution of input points and systematically adjusting the receptive field overlap. In a series of experiments including point cloud reconstruction, object classification and object part segmentation, our network achieves competitive performance. In particular, we outperform state-of-the-art deep learning approaches in point cloud classification and shape retrieval, with significantly faster training speed. As the SOM preserves the topological properties of the input space and our SO-Net converts point clouds into feature matrice accordingly, one promising future direction is to apply classical ConvNets or graph-based ConvNets to realize deeper hierarchical feature aggregation.

Acknowledgment. This work is supported partially by a ODPRT start-up grant R-252-000-636-133 from the National University of Singapore.

References

[1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017. 5, 6

[2] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, Feb. 2017. 1

[3] S. Bai, X. Bai, Z. Zhou, Z. Zhang, and L. Jan Latecki. Gift: A real-time and scalable 3d shape search engine. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5023–5032, 2016. 12

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. 1

[5] D. Boscaini, J. Masci, S. Melzi, M. M. Bronstein, U. Castellani, and P. Vandergheynst. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. In *Computer Graphics Forum*, volume 34, pages 13–23. Wiley Online Library, 2015. 2

[6] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016. 2, 6

[7] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013. 2

[8] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 1

[9] Y. Cheng, R. Cai, Z. Li, X. Zhao, and K. Huang. Locality-sensitive deconvolution networks with gated fusion for rgb-d indoor semantic segmentation. In *CVPR*, 2017. 5

[10] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012. 14

[11] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017. 1, 2

[12] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010. 7

[13] H. Fan, H. Su, and L. Guibas. A point set generation network for 3d object reconstruction from a single image. 2017. 5, 6

[14] R. Garg, G. Carneiro, and I. Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *ECCV*, 2016. 5

[15] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *arXiv preprint arXiv:1609.03677*, 2016. 5

[16] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017. 6

[17] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[18] R. Klokov and V. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. *arXiv preprint arXiv:1704.01222*, 2017. 1, 2, 7, 8, 12, 13, 14

[19] T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1):1–6, 1998. 2, 3

[20] 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. 5, 13, 14

[21] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas. Fpnn: Field probing neural networks for 3d data. In *Advances in Neural Information Processing Systems*, pages 307–315, 2016. 1, 2

[22] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 13, 14

[23] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *ICCV Workshops*, 2015. 2

[24] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015. 2

[25] D. J. Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980. 1

[26] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016. 1, 2, 4, 7, 8, 12, 14

[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 preprint arXiv:1706.02413*, 2017. 1, 2, 4, 6, 7, 8, 12, 14

[29] S. Ravanbakhsh, J. Schneider, and B. Poczos. Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500*, 2016. 2, 7

[30] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *CVPR*, 2017. 1, 2, 7

[31] P. Y. Simard, D. Steinkraus, J. C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003. 14

[32] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. *arXiv preprint arXiv:1704.02901*, 2017. 7, 14

[33] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*, 2015. 2, 12

[34] C. Wang, M. Pelillo, and K. Siddiqi. Dominant set clustering and pooling for multi-view 3d object recognition. In *BMVC*, 2017. 2

[35] D. Z. Wang and I. Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems*, 2015. 1, 2

[36] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72, 2017. 2, 7, 8, 12

[37] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 1, 2, 5

[38] L. Yi, V. G. Kim, D. Ceylan, I. Shen, M. Yan, H. Su, A. Lu, Q. Huang, A. Sheffer, L. Guibas, et al. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (TOG)*, 35(6):210, 2016. 6

[39] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola. Deep sets. *arXiv preprint arXiv:1703.06114*, 2017. 7