Recurrent Slice Networks for 3D Segmentation of Point Clouds

Qiangui Huang Weiyue Wang Ulrich Neumann
University of Southern California
Los Angeles, California
{qianguih,weiyuewa,uneumann}@usc.edu

Abstract

Point clouds are an efficient data format for 3D data. However, existing 3D segmentation methods for point clouds either do not model local dependencies [21] or require added computations [14, 23]. This work presents a novel 3D segmentation framework, RSNet[1], to efficiently model local structures in point clouds. The key component of the RSNet is a lightweight local dependency module. It is a combination of a novel slice pooling layer, Recurrent Neural Network (RNN) layers, and a slice unpooling layer. The slice pooling layer is designed to project features of unordered points onto an ordered sequence of feature vectors so that traditional end-to-end learning algorithms (RNNs) can be applied. The performance of RSNet is validated by comprehensive experiments on the S3DIS[1], ScanNet[3], and ShapeNet [34] datasets. In its simplest form, RSNets surpass all previous state-of-the-art methods on these benchmarks. And comparisons against previous state-of-the-art methods [21, 23] demonstrate the efficiency of RSNets.

1. Introduction

Most 3D data capturing devices (like LiDAR and depth sensors) produce point clouds as raw outputs. However, there are few state-of-the-art 3D segmentation algorithms that use point clouds as inputs. The main obstacle is that point clouds are usually unstructured and unordered, so it is hard to apply powerful end-to-end learning algorithms. As a compromise, many researchers transform point clouds into alternative data formats such as voxels [31, 33, 16, 22, 32] and multi-view renderings [22, 30, 19].

Unfortunately, information loss and quantitation artifacts often occur in data format transformations. These can lead to 3D segmentation performance drops as a result due to loss of local contexts. Moreover, the 3D CNNs [31, 33, 16, 22, 13] and 2D multi-view CNNs [30, 22] designed for these data formats are often time- and memory- consuming.

[1]Codes are released here https://github.com/qianguih/RSNet

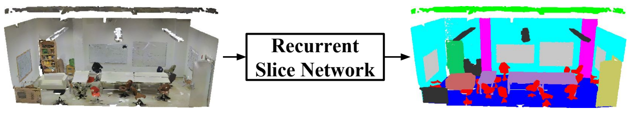


Figure 1: The RSNet takes raw point clouds as inputs and outputs semantic labels for each point.

In this paper, we approach 3D semantic segmentation tasks by directly dealing with point clouds. A simple network, a Recurrent Slice Network (RSNet), is designed for 3D segmentation tasks. As shown in Fig.1, the RSNet takes as inputs raw point clouds and outputs semantic labels for each of them.

The main challenge in handling point clouds is modeling local geometric dependencies. Since points are processed in an unstructured and unordered manner, powerful 2D segmentation methods like Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNNs) cannot be directly generalized to them.

In RSNets, the local context problem is solved by first projecting unordered points into ordered features and then applying traditional end-to-end learning algorithms. The projection is achieved by a novel slice pooling layer. In this layer, the inputs are features of unordered points and the output is an ordered sequence of aggregated features. Next, RNNs are applied to model dependencies in this sequence. Finally, a slice unpooling layer assigns features in the sequence back to points. In summary, the combination of the slice pooling layer, RNN layers, and the slice unpooling layer forms the local dependency module in RSNets. We note that the local dependency module is highly efficient. As shown in Section 3.2, the time complexity of the slice pooling/unpooling layer is $O(n)$ w.r.t the number of input points and $O(1)$ w.r.t the local context resolutions.

The performances of RSNets are validated on three challenging benchmarks. Two of them are large-scale real-world datasets, the S3DIS dataset [1] and the ScanNet dataset [3]. Another one is the ShapeNet dataset [34], a synthetic dataset. RSNets outperform all prior results and significantly improve performances on the S3DIS and ScanNet datasets.

In following parts of the paper, we first review related works in Section 2. Then, details about the RSNet are presented in Section 3. Section 4 reports all experimental results and Section 5 draws conclusions.

2. Related Works

Traditional 3D analysis algorithms are based on handcrafted features [19, 18, 20, 17, 7, 26, 25, 24, 9, 10, 11]. Recently, there are some works that utilize end-to-end learning algorithms for 3D data analysis. They are categorized by their input data formats as follows.

Voxelized Volumes. [33, 16, 22, 12, 13] made the early attempts of applying end-to-end deep learning algorithms for 3D data analysis, including 3D shape recognition, 3D urban scene segmentation [13]. They converted raw point cloud data into voxelized occupancy grids and then applied 3D deep Convolutional Neural Networks to them. Due to the memory constraints of 3D convolutions, the size of input cubes in these methods was limited to 60^3 and the depth of the CNNs was relatively shallow. Many works have been proposed to ease the computational intensities. One direction is to exploit the sparsity in voxel grids. In [4], the authors proposed to calculate convolutions at sparse input locations by pushing values to their target locations. Benjamin Graham designed a sparse convolution network [5, 6] and applied it for 3D segmentation tasks [36]. [15] tried to reduce computation by sampling 3D data at sparse points before feeding them into networks. In [27], the authors designed a memory efficient data structure, hybrid grid-octree, and corresponding convolution/pooling/unpooling operations to handle higher resolution 3D voxel grids (up to 256^3). In [31], the authors managed to consume 3D voxel inputs of higher resolution (100^3) and build deeper networks by adopting early down-sampling and efficient convolutional blocks like residual modules. While most of these works were focusing on reducing computational requirements of 3D voxel inputs, few of them tried to deal with the quantitation artifacts and information loss in voxelization.

Multi-view Renderings. Another popular data representation for 3D data is its multi-view rendering images. [19] designed a multi-view CNN for object detection in point clouds. In [29], 3D shapes were transformed into panoramic views, i.e., a cylinder project around its principal axis. [30] designed a 2D CNN for 3D shape recognition by taking as inputs multi-view images. In [22], the authors conducted comprehensive experiments to compare the recognition performances of 2D multi-view CNNs against 3D volumetric CNNs. More recently, multi-view 2D CNNs have been applied to 3D shape segmentation and achieved promising results. Compared to volumetric methods, multi-view based methods require less computational costs. However, there is also information loss in the multi-view rendering process.

Point Clouds. In the seminal work of PointNet [21], the authors designed a network to consume unordered and unstructured point clouds. The key idea is to process points independently and then aggregate them into a global feature representation by max-pooling. PointNet achieved state-of-the-art results on several 3D classification and segmentation tasks. However, there were no local geometric contexts in PointNet. In the following work, PointNet++ [23], the authors improved PointNet by incorporating local dependencies and hierarchical feature learning in the network. It was achieved by applying iterative farthest point sampling and ball query to group input points. In another direction, [14] proposed a KD-network for 3D point clouds recognition. In KD-network, a KD-tree was first built on input point clouds. Then, hierarchical groupings were applied to model local dependencies in points.

Both works showed promising improvements on 3D classification and segmentation tasks, which proved the importance of local contexts. However, their local context modeling methods all relied on heavy extra computations such as the iterate farthest point sampling and ball query in [23] and the KD-tree construction in [14]. More importantly, their computations will grow linearly when higher resolutions of local details are used. For example, higher local context resolutions will increase the number of clusters in [23] and result in more computations in iterative farthest point sampling. And higher resolutions will enlarge the kd-tree in [14] which also costs extra computations. In contrast, the key part of our local dependency module, the slice pooling layer, has a time complexity of $O(1)$ w.r.t the local context resolution as shown in Section 3.2.

3. Method

Given a set of unordered point clouds $X = \{x_1, x_2, ..., x_i, ..., x_n\}$ with $x_i \in \mathbb{R}^d$ and a candidate label set $L = \{l_1, l_2, ..., l_K\}$, our task is to assign each of input points x_i with one of the K semantic labels. In RSNets, the input is raw point clouds X and output is $Y = \{y_1, y_2, ..., y_i, ..., y_n\}$ where $y_i \in L$ is the label assigned to x_i.

A diagram of our method is presented in Fig.2. The input and output feature extraction blocks are used for independent feature generation. In the middle is the local dependency module. Details are illustrated below.

3.1. Independent Feature Extraction

There are two independent feature extraction blocks in an RSNet. The input feature block consumes input points

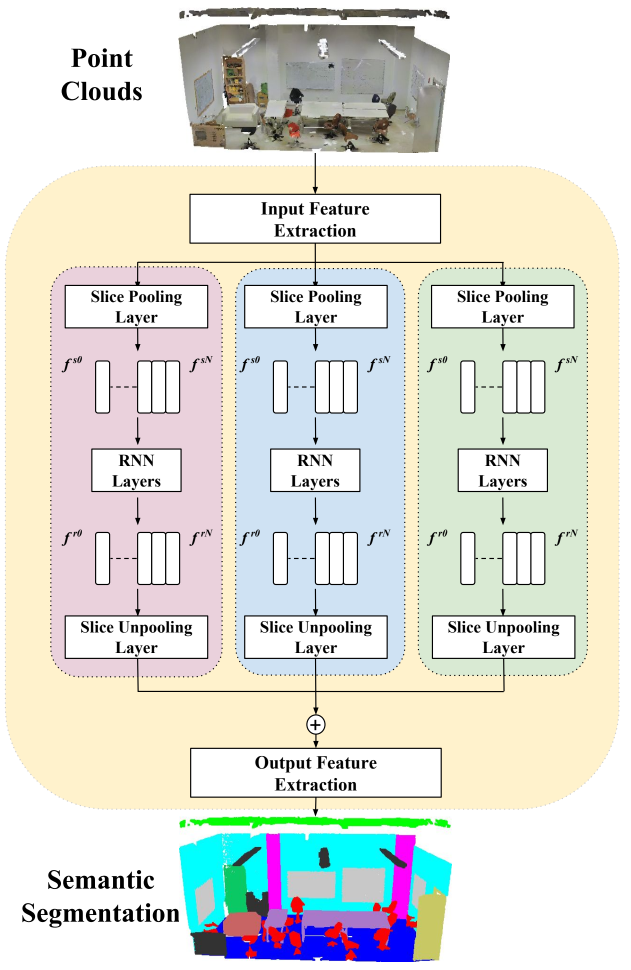


Figure 2: Diagram of an RSNet. The three parallel branches denote the slicing direction along x, y, and z axis.

$X \in \mathbb{R}^{n\times d^{in}}$ and produce features $F^{in} \in \mathbb{R}^{n\times d^{in}}$. Output feature blocks take processed features $F^{su} \in \mathbb{R}^{n\times d^{su}}$ as inputs and produce final predictions for each point. The superscript *in* and *su* indicate the features are from the input feature block and the slice unpooling layer, respectively. Both blocks use a sequence of multiple 1×1 convolution layers to produce independent feature representations for each point.

3.2. Local Dependency Module

The key part of an RSNet is the local dependency module which is a combination of a slice pooling layer, RNN layers, and a slice unpooling layer. It supports efficient and effective local context modeling. The slice pooling layer is designed to project features of unordered points onto an ordered sequence. RNNs are then applied to model dependencies in the sequence. In the end, the slice unpooling layer reverses the projection and assigns updated features back to each point.

Slice Pooling Layer. The inputs of a slice pooling layer are features of *unordered* point clouds $F^{in} = \{f_1^{in}, f_2^{in}, ..., f_i^{in}, ..., f_n^{in}\}$ and the output is an *ordered* sequence of feature vectors. This is achieved by first grouping points into slices and then generating a global representation for each slice via aggregating features of points within the slice.

Three slicing directions, namely slicing along x, y, and z axis, are considered in RSNets. We illustrate the details of slice pooling operation by taking z axis for example. A diagram of the slice pooling layer is presented in Fig.3. In a slice pooling layer, input points X are first split into slices by their spatial coordinates in z axis. The resolution of each slice is controlled by a hyper-parameter r. Assume input points span in the range $[z_{min}, z_{max}]$ in z axis. Then, the point x_i is assigned to the k^{th} slice, where $k = \lfloor (z_i - z_{min})/r \rfloor$ and z_i is x_i's coordinate in z axis. In total, there are N slices where $N = \lceil (z_{max} - z_{min})/r \rceil$. Here $\lceil\,\rceil$ and $\lfloor\,\rfloor$ indicate the ceil and floor function. After this process, all input points are grouped into N slices. They are also treated as N sets of points $S = \{S_1, S_2, ..., S_i, ..., S_N\}$, where S_i denotes the set of points assigned to i^{th} slice. In each slice, features of points are aggregated into one feature vector to represent the global information about this slice. Formally, after aggregation, a slice pooling layer produces an ordered sequence of feature vectors $F^s = \{f^{s1}, f^{s2}, ..., f^{si}, ..., f^{sN}\}$, where f^{si} is the global feature vector of slice set S_i. The max-pooling operation is adopted as the aggregation operator in RSNets. It is formally defined in equation (1).

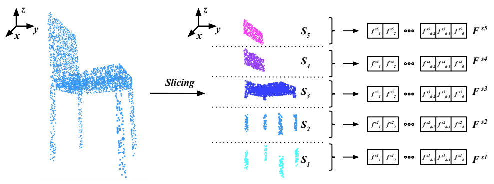
$$f^{si} = \max_{x_j \in S_i} \{f_j^{in}\} \tag{1}$$

The slice pooling layer has several important properties:

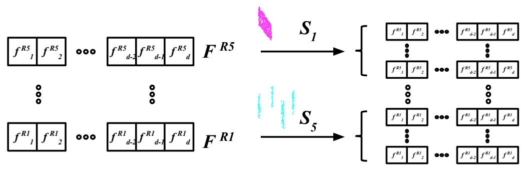
1. **Order and Structure**. F^s is an *ordered* and *structured* sequence of feature vectors. In the aforementioned case, F^s is ordered in the z axis. f^{s1} and f^{sN} denote the feature representations of the bottom-most and top-most set of points, respectively. Meanwhile, f^{si} and $f^{s(i-1)}$ are features representing adjacent neighbors. This property makes traditional local dependency modeling algorithms applicable as F^s is structured and ordered now.

2. **Efficiency**. The time complexity of the slice pooling layer is $O(n)$ (n is the number of the input points). And it is $O(1)$ w. r. t the slicing resolution r.

3. **Local context trade-off**. Given a fixed input, smaller r will produce more slices with richer local contexts



(a) Illustration of the slice pooling operation. A set of points from a chair is used for illustration purpose here.



(b) Illustration of the slice unpooling operation. Global feature representation for one point set is replicated back to all points in the set.

Figure 3: Illustration of slice pooling and slice unpooling operation and RNN modeling for slices.

preserved while larger r produces fewer slices with coarse local contexts;

RNN Layer. As mentioned above, the slice pooling layer is essentially projecting features of unordered and unstructured input points onto an ordered and structured sequence of feature vectors. RNNs are a group of end-to-end learning algorithms naturally designed for a structured sequence. Thus, they are adopted to model dependencies in the sequence. By modeling one slice as one timestamp, the information from one slice interacts with other slices as the information is flowing through timestamps in RNN units. This enables contexts in slices impact with each other which in turn models the dependencies in them.

In an RSNet, the input of RNN layers is F^s. In order to guarantee information from one slice could impact on all other slices, RSNets utilize the bidirectional RNN units [28] to help information flow in both directions. After processing the inputs with a stack of bidirectional RNNs, the final outputs are $F^r = \{f^{r1}, f^{r2}, ..., f^{ri}, ..., f^{rN}\}$ with superscript r denoting the features are from RNN layers. Compared with F^s, F^r has been updated by interacting with neighboring points.

Slice Unpooling Layer. As the last part of an RSNet's local dependency module, the slice unpooling layer takes updated features F^r as inputs and assigns them back to each point by reversing the projection. This can be easily achieved by storing the slice sets S. A diagram of the slice unpooling layer is presented in Fig.3. We note that the time complexity of slice unpooling layer is $O(n)$ w. r. t the number of input points and is $O(1)$ w. r. t slicing resolution as well.

4. Experiments

In order to evaluate the performance of RSNets and compare with state-of-the-art, we benchmark RSNets on three datasets, the Stanford 3D dataset (S3DIS) [1], ScanNet dataset [3], and the ShapeNet dataset [34]. The first two are large-scale realistic 3D segmentation datasets and the last one is a synthetic 3D part segmentation dataset.

We use the strategies in [21, 23] to process all datasets. For the S3DIS and ScanNet datasets, the scenes are first divided into smaller cubes using a sliding window of a fixed size. A fixed number of points are sampled as inputs from the cubes. In this paper, the number of points is fixed as 4096 for both datasets. Then RSNets are applied to segment objects in the cubes. Note that we only divide the scene on the xy plane as in [21]. During testing, the scene is similarly split into cubes. We first run RSNets to get point-wise predictions for each cube, then merge predictions of cubes in the same scene. Majority voting is adopted when multiple predictions of one point are present.

We use one unified RSNet architecture for all datasets. In the input feature extraction block, there are three 1×1 convolutional layers with output channel number of 64, 64, and 64, respectively. In the output feature extraction block, there are also three 1×1 convolutional layers with output channel number of 512, 256, and K, respectively. Here K is the number of semantic categories. In each branch of the local dependency module, the first layer is a slice pooling layer and the last layer is a slice unpooling layer. The slicing resolution r varies for different datasets. There is a comprehensive performance comparison of different r values in Section 4.2. In the middle are the RNN layers. A stack of 6 bidirectional RNN layers is used in each branch. The numbers of channels for RNN layers are 256, 128, 64, 64, 128, and 256. In the baseline RSNet, Gated Recurrent Unit (GRU) [2] units are used in all RNNs.

Two widely used metrics, mean intersection over union (mIOU) and mean accuracy (mAcc), are used to measure the segmentation performances. We first report the performance of a baseline RSNet on the S3DIS dataset. Then, comprehensive studies are conducted to validate various architecture choices in the baseline. In the end, we show state-of-the-art results on the ScanNet and ShapeNet dataset. Through experiments, the performances of RSNets are compared with various state-of-the-art 3D segmentation methods including 3D volumes based methods [31], spectral CNN based method [35], and point clouds based methods [21, 23, 14].

4.1. Segmentation on the S3DIS Dataset

We first present the performances of a baseline RSNet on the S3DIS dataset. The training/testing split in [31] is used here to better measure the generalization ability of all methods. The slicing resolutions r along the x, y, z axis are

Method	mIOU	mAcc	ceiling	floor	wall	beam	column	window	door	chair	table	bookcase	sofa	board	clutter
PointNet^A [21]	41.09	48.98	88.80	97.33	69.80	**0.05**	3.92	**46.26**	10.76	52,61	58.93	40.28	5.85	26.38	33.22
3D-CNN [31]	43.67	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3D-CNN^A [31]	47.46	54.91	90.17	96.48	70.16	0.00	11.40	33.36	21.12	**76.12**	70.07	57.89	37.46	11.16	41.61
3D-CNN^{AC} [31]	48.92	57.35	90.06	96.05	69.86	0.00	**18.37**	38.35	23.12	75.89	**70.40**	**58.42**	40.88	12.96	41.60
Ours	**51.93**	**59.42**	**93.34**	**98.36**	**79.18**	0.00	15.75	45.37	**50.10**	65.52	67.87	22.45	**52.45**	**41.02**	**43.64**

Table 1: Results on the Large-Scale 3D Indoor Spaces Dataset (S3DIS). Superscripts A and C denote data augmentation and post-processing (CRF) are used.

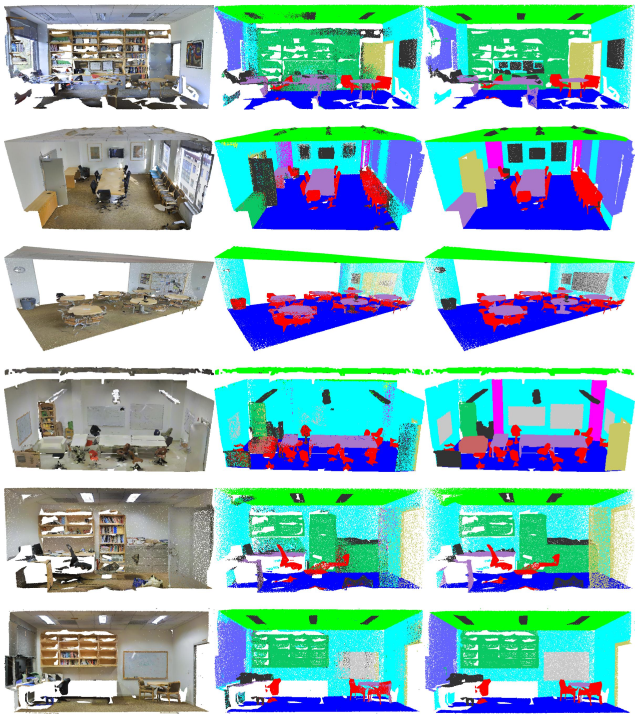


Figure 4: Sample segmentation results on the S3DIS dataset. From left to right are the input scenes, results produced by the RSNet, and ground truth. Best viewed with zoom in.

all set at $2cm$. And the block size in x and y axis of each cube is $1m \times 1m$. Given these settings, there are 50 slices ($N = 50$) in x and y branch in an RSNet after the slice pooling layer. As we do not limit the block size in z axis, the number of slices along z axis varies on different inputs. In the S3DIS dataset, most of the scenes have a maximum z coordinate around $3m$ which produces around 150 slices.

During testing, the sliding stride is set at $1m$ to generate non-overlapping cubes. The performance of our baseline network is reported in Table.1. Besides the overall mean IOU and mean accuracy, the IOU of each category is also presented. Meanwhile, some segmentation results are visualized in Fig.4.

Previous state-of-the-art results [21, 31] are reported in Table.1 as well. In [31], the data representation is voxelized 3D volumes and a 3D CNN is built for segmenting objects in the volumes. Several geometric data augmentation strategies and end-to-end Conditional Random Filed (CRF) are utilized in their work. The PointNet [21] takes the same inputs, point clouds, as our method. It adopted rotation along z axis to augment data. In contrast, our baseline RSN does not use any data augmentations.

The results in Table.1 show that the RSNet has achieved state-of-the-art performances on the S3DIS dataset even without using any data augmentation. In particular, it improves previous 3D volumes based methods [31] by 3.01 in mean IOU and 2.07 in mean accuracy. Compared with prior point clouds based method [21], it improves the mean IOU by 10.84 and mean accuracy by 10.44. The detailed per-category IOU results show that the RSNet is able to achieve better performances in more than half of all categories (7 out of 13).

We argue that the great performance improvements come from the local dependency module in the RSNet. While PointNet only relies on global features, the RSNet is equipped with local geometric dependencies among points. In summary, the significant performance gains against PointNet demonstrate: 1). local dependency modeling is crucial for 3D segmentation; 2). the combination of the novel slice pooling/unpooling layers and RNN layers is able to support effective spatial dependencies modeling in point clouds. Moreover, the performance improvements against 3D volumes based methods prove that directly handling point clouds can boost the 3D segmentation performances a lot as there are no quantitation artifacts and no local details lost anymore.

4.2. Ablation Studies

In this section, we validate the effects of various architecture choices and testing schemes. In particular, several key parameters are considered: 1). the slicing resolution r in RSNets; 2). the size of the sliding block; 3). the sliding stride during testing; 4). the type of RNN units. All set-

r_x (cm)	r_y (cm)	r_z (cm)	mIOU	mAcc
2	2	1	49.12	56.63
2	2	2	**51.93**	**59.42**
2	2	5	51.20	58.97
2	2	8	49.16	56.91
1	1	2	49.23	56.90
2	2	2	**51.93**	**59.42**
4	4	2	48.97	57.10
6	6	2	47.86	56.82

.

Table 2: Varying slice resolutions for RSNs on the S3DIS dataset. r_x, r_y, and r_z indicate the slicing resolution along x, y, and z axis, respectively.

bs (m)	r_x (cm)	r_y (cm)	r_z (cm)	mIOU	mAcc
1	2	2	2	**51.93**	**59.42**
	4	4	2	48.97	57.10
	6	6	2	47.86	56.82
2	2	2	2	44.15	52.39
	4	4	2	**44.59**	52.62
	6	6	2	43.15	**53.07**
3	2	2	2	**39.08**	**49.61**
	4	4	2	37.77	47.89
	6	6	2	37.55	49.01
	8	8	2	37.21	46.35
	16	16	2	35.25	44.70

.

Table 3: Varying sizes of sliding blocks for RSNs on the S3DIS dataset. bs indicates the block size.

sliding stride during testing	mIOU	mAcc
0.2	52.39	60.52
0.5	**53.83**	**61.81**
1.0	51.93	59.42

.

Table 4: Varying the testing stride on the S3DIS dataset

RNN unit	mIOU	mAcc
vanilla RNN	45.84	54.82
GRU	**51.93**	**59.42**
LSTM	50.08	57.80

.

Table 5: Varying RNN units for RSNs on the S3DIS dataset

tings remain unchanged as the baseline RSNet in following control experiments except explicitly specified.

Slicing resolution. The slicing resolution r is an important hyper-parameter in RSNets. It controls the resolution of each slice which in turn controls how much local details are kept after slice pooling. By using a small slicing resolution, there are more local details preserved as the feature aggregation operation is executed in small local regions. However, a small slicing resolution will produce a large number of slices which requires RNN layers to consume a longer sequence. This may hurt the performance of RSNets as the RNN units may fail to model dependencies in the long sequence due to the "gradient vanishing" problem [8]. On the other hand, a large slicing resolution will eliminate a lot of local details in input data as the feature aggregation is conducted on a wide spatial range. Thus, there is a trade-off of selecting the slicing resolution r.

Several experiments are conducted to show the impacts of different slicing resolutions. Two groups of slicing resolutions are tested. In the first group, we fix the slicing resolutions along x and y axis to be $2cm$ and vary the resolution along the z axis. In the second group, the slicing resolution along z axis is fixed as $2cm$ while varying resolutions along x and y axis. Detailed performances are reported in Table.2. Results in Table.2 show that the slicing resolution of $2cm$, $2cm$, $2cm$ along x, y, z axis works best for the S3DIS dataset. Both larger or smaller resolutions decrease the final performances.

Size of sliding block. The size of the sliding block is another key factor in training and testing. Small block sizes may result in too limited contexts in one cube. Large block sizes may put RSNets in a challenging trade-off between slicing resolutions as large block size will either produce more slices when the slicing resolution is fixed or increase the slicing resolution. In Table.3, we report the results of three different block sizes, $1m$, $2m$, and $3m$, along with different slicing resolution choices. The results show that larger block sizes actually decrease the performance. That is because larger block sizes produce a longer sequence of slices for RNN layers, which is hard to model using RNNs. Among various settings, the optimal block size for the S3DIS dataset is $1m$ on both x and y axis.

Stride of sliding during testing. When breaking down the scenes during testing, there are two options, splitting it into non-overlapping cubes or overlapping cubes. In PonintNet [21] , non-overlapping splitting is used while PointNet++ [23] adopted overlapping splitting. For RSNets, both options are tested. Specifically, we set the sliding stride into three values, $0.2m$, $0.5m$, and $1m$. The first two produce overlapping cubes and the last one produces non-overlapping cubes. All results are reported in Table.4. Experimental results show that using overlapped division can slightly increase the performance (0.4$\sim$1.9 in mean IOU and 1.1$\sim$2.4 in mean accuracy on the S3DIS dataset). However, testing using overlapped division requires more computations as there are more cubes to process. Thus, we select the non-overlap sliding in our baseline RSNet.

RNN units. Due to the "gradient vanishing" problem in

Method	mIOU	mAcc	wall	floor	chair	table	desk	bed	book-shelf	sofa	sink
PointNet [21]	14.69	19.90	69.44	88.59	35.93	32.78	2.63	17.96	3.18	32.79	0.00
PointNet++ [23]	34.26	43.77	77.48	92.50	64.55	46.60	12.69	51.32	52.93	52.27	30.23
Ours	**39.35**	**48.37**	**79.23**	**94.10**	**64.99**	**51.04**	**34.53**	**55.95**	**53.02**	**55.41**	**34.84**
Method	bathtub	toilet	curtain	counter	door	window	shower curtain	refrid-gerator	picture	cabinet	other furniture
PointNet [21]	0.17	0.00	0.00	5.09	0.00	0.00	0.00	0.00	0.00	4.99	0.13
PointNet++ [23]	42.72	31.37	**32.97**	20.04	2.02	3.56	27.43	18.51	0.00	23.81	2.20
Ours	**49.38**	**54.16**	6.78	**22.72**	**3.00**	**8.75**	**29.92**	**37.90**	**0.95**	**31.29**	**18.98**

Table 6: Results on the ScanNet dataset. IOU of each category is also reported here.

the vanilla RNN unit, two RNN variants, LSTM and GRU, are proposed to model long-range dependencies in inputs. The effects of different RNN units are compared in Table.5. They show that GRU has the best performance for RSNets.

4.3. Segmentation on the ScanNet dataset

We now show the performances of RSNets on the ScanNet dataset. The exact same RSNet as Section 4.1 is used to process the ScanNet dataset. The performance of the RSNet is reported in Table.6.

In the ScanNet dataset, the previous state-of-the-art method is PointNet++ [23]. It only uses the xyz information of point clouds as inputs. To make a fair comparison, we also only use xyz information in the RSNet. [23] only reported the global accuracy on the ScanNet dataset. However, as shown in the supplementary, the ScanNet dataset is highly unbalanced. In order to get a better measurement, we still use mean IOU and mean accuracy as evaluation metrics as previous sections. We reproduced the performances of PointNet[21] and Pointnet++[23] (the single scale version) on the ScanNet dataset [2] and report them in Table.6 as well. As shown in Table.6, the RSNet has also achieved state-of-the-art results on the ScanNet dataset. Compared with the PointNet++, the RSNet improves the mean IOU and mean accuracy by 5.09 and 4.60. Some comparisons between different methods are visualized in Fig.5. These visualizations show that as a benefit of the local dependency module, the RSNet is able to handle small details such the chairs, desks, and toilets in inputs.

4.4. Segmentation on the ShapeNet Dataset

In order to compare the RSNet with some other methods [14, 34, 35], we also report the segmentation results of RSNets on the ShapeNet part segmentation dataset. The same RSNet as in Section 4.1 is used here. It only takes the xyz information as the convention. Its performance are reported in Table.7. Table.7 also presents the results of other state-of-the-art methods including PointNet [21], PointNet++ [23], KD-net [14], and spectral CNN [35]. The RSNet outperforms all other methods except the PointNet++[23] which utilized extra normal information as inputs. However, the RSNet can also outperform PointNet++ when it only takes xyz information. This validates the effectiveness of the RSNet.

[2] We reproduced the PonitNet and PointNet++ training on ScanNet by using the codes here and here which are published by the authors. The global accuracy of our version of PointNet and PointNet++ are 73.69% and 81.35%, respectively.

4.5. Computation Analysis

We now demonstrate the efficiency of RSNets in terms of inference speed and GPU memory consumption. We follow the same time and space complexity measurement strategy as [23]. We record the inference time and GPU memory consumption of a batch of 8 4096 points for vanilla PointNet and the RSNet using PyTorch on a K40 GPU. Since [23] reported the inference speed in TensorFlow, we use the relative speed w.r.t vanilla PointNet to compare speeds with each other. The speed and memory measurements are reported in Table.8.

Table.8 show that the RSNet is much faster than PointNet++ variants. It is near 1.6 $\times$ faster than the single scale version of PointNet++ and 3.1 $\times$ faster than its multi-scale version. Moreover, the GPU memory consumption of the RSNet is even lower than vanilla PointNet. These prove that the RSNet is not only powerful but also efficient.

5. Conclusion

This paper introduces a powerful and efficient 3D segmentation framework, Recurrent Slice Network (RSNet). An RSNet is equipped with a lightweight local dependency modeling module which is a combination of a slice pooling, RNN layers, and a slice unpooling layer. Experimental results show that RSNet can surpass previous state-of-the-art methods on three widely used benchmarks while requiring less inference time and memory.

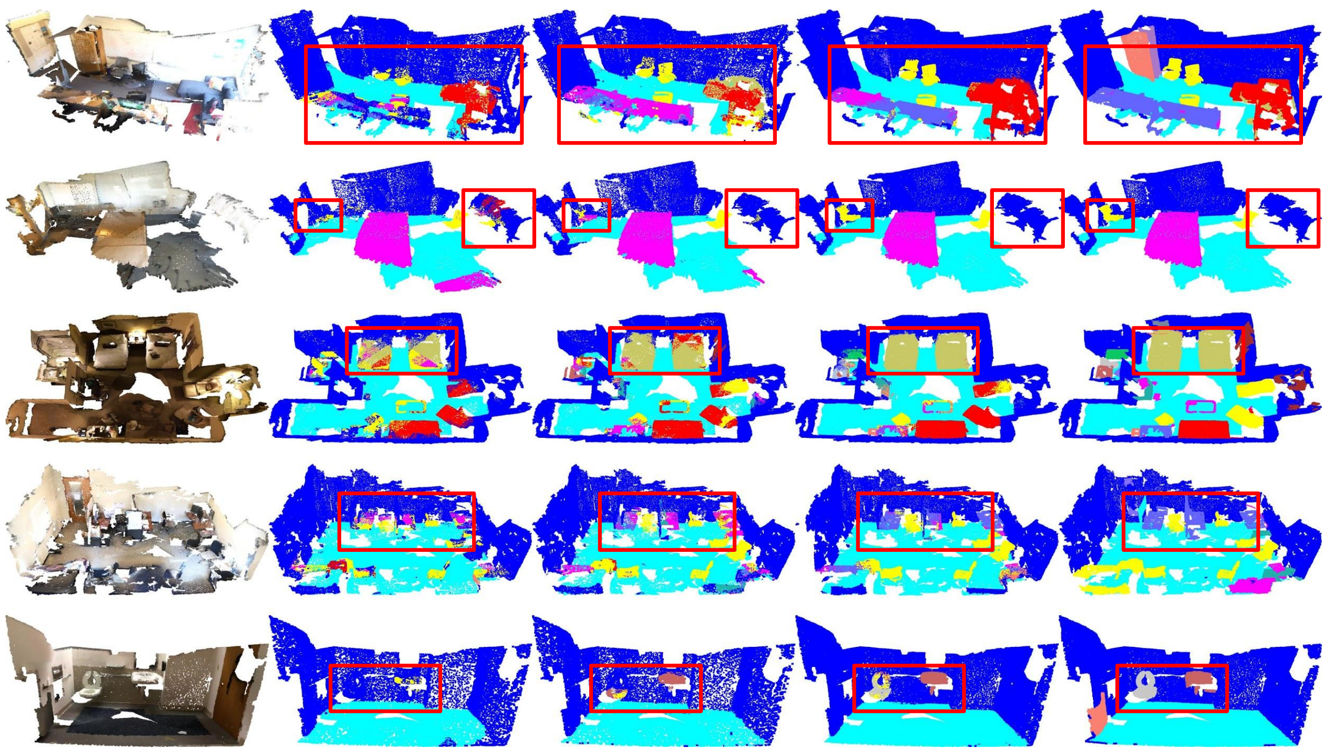


Figure 5: Sample segmentation results on the ScanNet dataset. From left to right are the input scenes, results produced by PointNet, PointNet++, RSNet, and ground truth. Interesting areas have been highlighted by red bounding boxes. Best viewed with zoom in.

Method	mean	aero	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor	mug	pistol	rocket	skate board	table
Yi [34]	81.4	81.0	78.4	77.7	75.7	87.9	61.9	92.0	85.4	82.5	95.7	70.6	91.9	85.9	53.1	69.8	75.3
KD-net [14]	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
PN [21]	83.7	**83.4**	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
PN++ * [23]	84.6	80.4	80.9	60.0	76.8	88.1	**83.7**	90.2	82.6	76.9	94.7	68.0	91.2	**82.1**	59.9	78.2	**87.5**
SSCNN [35]	84.7	81.6	81.7	81.9	75.2	90.2	74.9	**93.0**	86.1	**84.7**	**95.6**	66.7	92.7	81.6	**60.6**	**82.9**	82.1
PN++ [23]	**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
Ours	**84.9**	82.7	**86.4**	84.1	**78.2**	90.4	69.3	91.4	87.0	83.5	95.4	66.0	92.6	81.8	56.1	75.8	82.2

Table 7: Results on the ShapeNet dataset. PN++ * denotes the PointNet++ trained by us which does not use extra normal information as inputs.

	PointNet (vanilla) [21]	PointNet [21]	PointNet++ (SSG) [23]	PointNet++ (MSG) [23]	PointNet++ (MRG) [23]	RSNet
Speed	1.0 ×	2.2 ×	7.1 ×	14.1 ×	7.5 ×	4.5 ×
Memory	844 MB	-	-	-	-	756 MB

.

Table 8: Computation analysis between PointNet, PointNet++, and RSNet.

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 *Proceedings of the IEEE Conference*

on Computer Vision and Pattern Recognition, pages 1534–1543, 2016. 1, 4

[2] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 4

[3] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. *arXiv preprint arXiv:1702.04405*, 2017. 1, 2, 4

[4] 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 *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1355–1361. IEEE, 2017. 2

[5] B. Graham. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014. 2

[6] B. Graham. Sparse 3d convolutional neural networks. *arXiv preprint arXiv:1505.02890*, 2015. 2

[7] W. Guan, S. You, and G. Pang. Estimation of camera pose with respect to terrestrial lidar data. In *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*, pages 391–398. IEEE, 2013. 2

[8] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001. 6

[9] J. Huang and S. You. Point cloud matching based on 3d self-similarity. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 41–48. IEEE, 2012. 2

[10] J. Huang and S. You. Detecting objects in scene point cloud: A combinational approach. In *3D Vision-3DV 2013, 2013 International Conference on*, pages 175–182. IEEE, 2013. 2

[11] J. Huang and S. You. Pole-like object detection and classification from urban point clouds. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3032–3038. IEEE, 2015. 2

[12] J. Huang and S. You. Point cloud labeling using 3d convolutional neural network. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 2670–2675. IEEE, 2016. 2

[13] J. Huang and S. You. Vehicle detection in urban point clouds with orthogonal-view convolutional neural network. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 2593–2597. IEEE, 2016. 1, 2

[14] 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, 4, 7, 8

[15] 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. 2

[16] D. Maturana and S. 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. 1, 2

[17] G. Pang and U. Neumann. Training-based object recognition in cluttered 3d point clouds. In *3D Vision-3DV 2013, 2013 International Conference on*, pages 87–94. IEEE, 2013. 2

[18] G. Pang and U. Neumann. Fast and robust multi-view 3d object recognition in point clouds. In *3D Vision (3DV), 2015 International Conference on*, pages 171–179. IEEE, 2015. 2

[19] G. Pang and U. Neumann. 3d point cloud object detection with multi-view convolutional neural network. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 585–590. IEEE, 2016. 1, 2

[20] G. Pang, R. Qiu, J. Huang, S. You, and U. Neumann. Automatic 3d industrial point cloud modeling and recognition. In *Machine Vision Applications (MVA), 2015 14th IAPR International Conference on*, pages 22–25. IEEE, 2015. 2

[21] 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, 5, 6, 7, 8

[22] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5648–5656, 2016. 1, 2

[23] 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

[24] R. Qiu and U. Neumann. Exemplar-based 3d shape segmentation in point clouds. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 203–211. IEEE, 2016. 2

[25] R. Qiu and U. Neumann. Ipdc: Iterative part-based dense correspondence between point clouds. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*, pages 1–9. IEEE, 2016. 2

[26] R. Qiu, Q.-Y. Zhou, and U. Neumann. Pipe-run extraction and reconstruction from point clouds. In *European Conference on Computer Vision*, pages 17–30. Springer, 2014. 2

[27] G. Riegler, A. O. Ulusoys, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. *arXiv preprint arXiv:1611.05009*, 2016. 2

[28] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. 4

[29] B. Shi, S. Bai, Z. Zhou, and X. Bai. Deeppano: Deep panoramic representation for 3-d shape recognition. *IEEE Signal Processing Letters*, 22(12):2339–2343, 2015. 2

[30] H. Su, S. Maji, E. Kalogerakis, and E. 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. 1, 2

[31] L. P. Tchapmi, C. B. Choy, I. Armeni, J. Gwak, and S. Savarese. Segcloud: Semantic segmentation of 3d point clouds. *arXiv preprint arXiv:1710.07563*, 2017. 1, 2, 4, 5

[32] W. Wang, Q. Huang, S. You, C. Yang, and U. Neumann. Shape inpainting using 3d generative adversarial network and recurrent convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2298–2306, 2017. 1

[33] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. 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. 1, 2

[34] 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. 1, 2, 4, 7, 8

[35] L. Yi, H. Su, X. Guo, and L. Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. *arXiv preprint arXiv:1612.00606*, 2016. 4, 7, 8

[36] L. Yi, H. Su, L. Shao, M. Savva, H. Huang, Y. Zhou, B. Graham, M. Engelcke, R. Klokov, V. Lempitsky, et al. Large-scale 3d shape reconstruction and segmentation from shapenet core55. *arXiv preprint arXiv:1710.06104*, 2017. 2