Recurrent Slice Networks for 3D Segmentation on Point Clouds

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

Abstract

In this paper, we present a conceptually simple and powerful framework, Recurrent Slice Network (RSNet), for 3D semantic segmentation on point clouds. Performing 3D segmentation on point clouds is computationally efficient. And it is free of the quantitation artifact problems which exists in other 3D data formats such as voxelized volumes and multi view renderings. However, existing point clouds based methods either do not model local dependencies [13] or rely on heavy extra computations [10, 15]. In contrast, our RSNet is equipped with a lightweight local dependency module, which 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 into an ordered sequence of feature vectors. Then, RNNs are applied to model dependencies for the sequence. We validate the importance of local contexts and the effectiveness of our RSNet on the S3DIS[1], ScanNet[3], and ShapeNet [23] dataset. Without bells and whistles, RSNet surpasses all previous state-of-the-art methods on these benchmarks. Moreover, additional computation analysis demonstrates the efficiency of RSNet.

1. Introduction

3D semantic segmentation is an important task for scene understanding in 3D. Most of the 3D data capturing devices (like LiDAR and depth sensor) produce point clouds as raw outputs. However, there are very few state-of-the-art 3D scene analysis algorithms taking point clouds as inputs. The main reason is that point clouds are unstructured and unordered. This makes it hard to apply powerful end-to-end learning algorithms which require structured inputs. As a compromise, many existing works transformed point clouds into other data representations such as voxelized volumes [20, 22, 12, 14, 21] and multi-view renderings [14, 19].

Unfortunately, there is usually information lost and quantitation artifacts in data format transformation. Local details usually disappear in this process. This hurts the performances in tasks that require rich local context, such as 3D segmentation. Moreover, the 3D CNN [20, 22, 12, 14] and 2D multi view CNN [19, 14] built for these data formats are usually time- and memory- consuming.

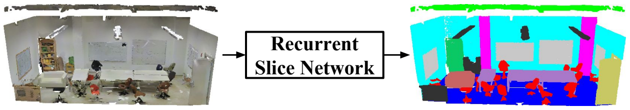


Figure 1: Our RSNet takes raw point clouds as inputs and outputs semantic labels for each of them.

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

The main challenge in handling point clouds is how to model local geometric dependencies between points. As the points come in an unstructured and unordered manner, powerful 2D segmentation methods like Convolutional Neural Networks (CNN) can not be directly generalized for them.

In RSNet, this problem is solved by projecting unordered points into an ordered sequence of their features via a slice pooling layer. Then, Recurrent Neural Networks (RNN) are used to model structures in the sequence. At the end, a slice unpooling layer is designed to assign features in the sequence back to points. The combination of the slice pooling layer, RNN layers, and slice unpooling layer forms the local dependency module in our RSNet. The key part is the slice pooling layer. It is able to project unordered point clouds into an ordered format, which makes the application of traditional deep learning algorithms (the RNNs) feasible. And in order to guarantee the efficiency, we adopt a conceptually simple design for the slice pooling layer. As shown in Section 3.2, its time complexity is $O(n)$ w.r.t the number of input points and $O(1)$ w.r.t the local context resolutions.

The performances of the RSNet are validated on three challenging benchmarks. Two of them are large scale realistic datasets, the S3DIS dataset [1] and the ScanNet dataset [3]. And another one is the ShapeNet dataset [23], a synthetic dataset. Without bells and whistles, RSNet can outperform all previous state-of-the-arts and significantly improve the performances on the S3DIS and ScanNet dataset.

In the remaining part 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 the conclusions.

2. Related Works

Voxelized Volumes. [22, 12, 14, 9] made the early attempts of applying end-to-end deep learning algorithms for 3D data analysis, including 3D shape recognition, 3D urban scene segmentation [9]. They all converted raw point cloud data into voxelized occupancy grids and then applied 3D deep Convolutional Neural Networks to them. Due to the memory and speed constraints of 3D convolutions, the size of input cubes in these methods were limited to 60^3 and the depth of the CNNs are relatively shallow. Many works have been proposed to ease the memory and computational intensities. One main direction is to exploit the sparsity in voxel grids. In [5], the authors proposed to calculate convolutions at sparse input locations by pushing values to their target locations. Benjamin Graham designed a sparse convolution network [6, 7] and applied it for 3D segmentation tasks [25]. [11] tried to reduce computation by sampling 3D data at sparse points before feeding them into networks. In [16], 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 [20], 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 vexel 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. In [18], 3D shapes were transformed into panoramic views, i.e., a cylinder project around its principle axis. [19] designed a 2D CNN for 3D shape recognition by taking multi-view images as inputs. In [14], the authors conducted comprehensive experiments to compare the recognition performances of 2D multi-view CNNs againt 3D volumetric CNNs. More recently, multi-view 2D CNNs have been applied to 3D shape segmentation and achieved promising results. Compared with volumetric methods, multi-view based methods are more efficient in terms of computational costs. However, there is also information lost in the multi-view rendering process.

Point Clouds. In the seminar work of PointNet [13], 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 classification and segmentation tasks. However, there is no local geometric contexts modeled in PointNet. In the following work, PointNet++ [15], the authors improved PointNet by incorporating local dependencies and hierarchical feature learning in the network. It was achieved by grouping input points using iterative farthest point sampling and ball query. In another direction, [10] proposed a KD-network for 3D point cloud recognitions. 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 [15] and the KD-tree construction in [10]. 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 [15] and results in more computations in iterative farthest point sampling. And higher resolutions will enlarge the kd-tree in [10] 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 RSNet, 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 RSNet. The input feature block consumes input points X (of size $n \times d$) and produce features F_i^{in} (of size $n \times d^{in}$). Output feature blocks take processed features F^{su} (of size $n \times d^{su}$) as inputs and produce final predictions for each point. The superscript *in* and *su* indicate the features are

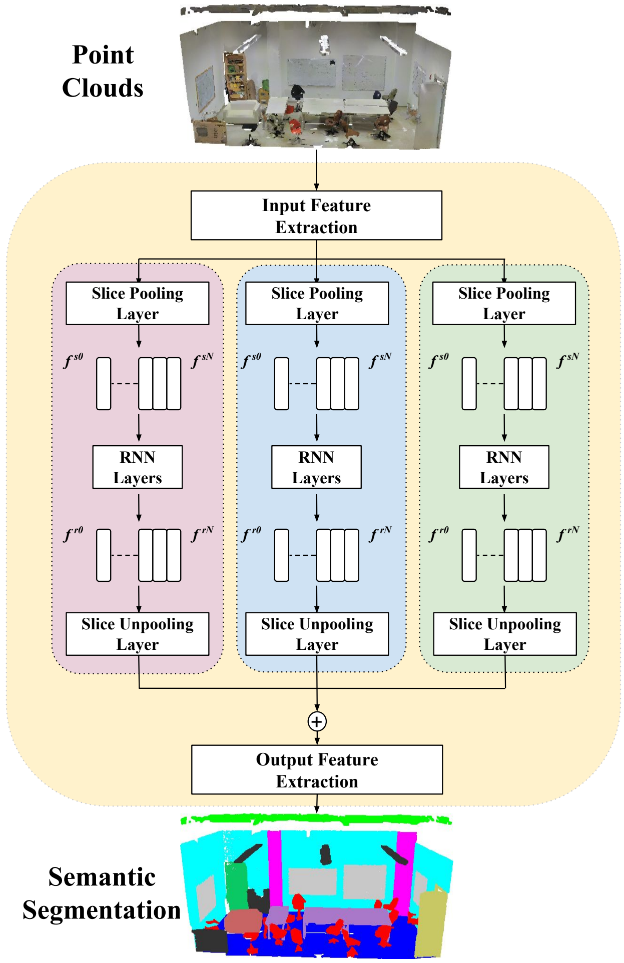


Figure 2: Diagram of our RSNet. The three parallel branches denote the slicing direction along x, y, and z axis.

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 RSNet is the local dependency module which is a combination of a slice pooling layer, RNN layers, and a slice unpooling layer. It offers an efficient solution for the 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 for the sequence. At the end, the slice unpooling layer reverses the projection and assign 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 RSNet. 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 $X = \{x_1, x_2, ..., x_i, ..., x_n\}$ are first splitted into slices by their spatial coordinates in z axis. The resolution of each slice is controlled by a hyper-parameter r. Assume input points are distributed 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 the x_i's coordinate in z axis. And there are N slices in total where $N = \lceil (z_{max} - z_{min})/r \rceil$. Here $\lceil \, \rceil$ and $\lfloor \, \rfloor$ indicate the ceil and floor function. In this way, 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 RSNet. It is formally defined in equation (1).

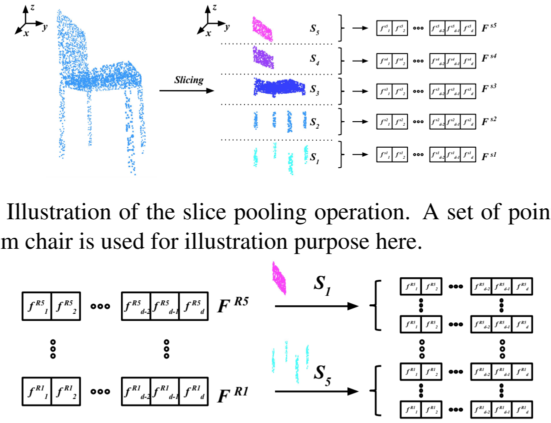
$$f^{si} = \max_{x_j \in S_i} \{f_j^{in}\} \tag{1}$$

The slice pooling layer designed above enjoys several interesting properties:

1. **Order and Structure**. $F^s = \{f^{s1}, f^{s2}, ..., f^{si}, ..., f^{sN}\}$ 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 models 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 preserved while larger r produces less slices with coarse local contexts;



(a) Illustration of the slice pooling operation. A set of points from 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.

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

In RSNet, the input of RNN layers is F^s. In order to guarantee information from one slice would impact on all other slices, RSNet utilizes the bidirectional RNN units [17] 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 in 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. 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 RSNet and compare with state-of-the-arts, we benchmark RSNet on three datasets, the Stanford 3D dataset (S3DIS) [1], ScanNet dataset [3], and the ShapeNet dataset [23]. The first two are large scale realistic 3D segmentation datasets and the last one is a synthetic 3D part segmentation dataset.

The training strategies in [13, 15] are adopted in this paper. For the S3DIS and ScanNet datasets, the scenes are first divided into smaller cubes using a sliding window of a fixed size and 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 [13]. During testing, the scene is similarly splitted into cubes. We first run RSNets to get point-wise predictions for each cube, then merge predictions of cubes in a 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 on 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 brach. 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. At 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 [20], spectral CNN based method [24], and point clouds based methods [13, 15, 10]. And there are more experimental results in the Appendices.

4.1. Segmentation on the S3DIS Dataset

We first show the performance of a baseline RSNet on the S3DIS dataset. The training/testing split in [20] is used here to better measure the generalization ability of all meth-

Method	mIOU	mAcc	ceiling	floor	wall	beam	column	window	door	chair	table	bookcase	sofa	board	clutter
PointNet^A [13]	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 [20]	43.67	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3D-CNN^A [20]	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} [20]	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.

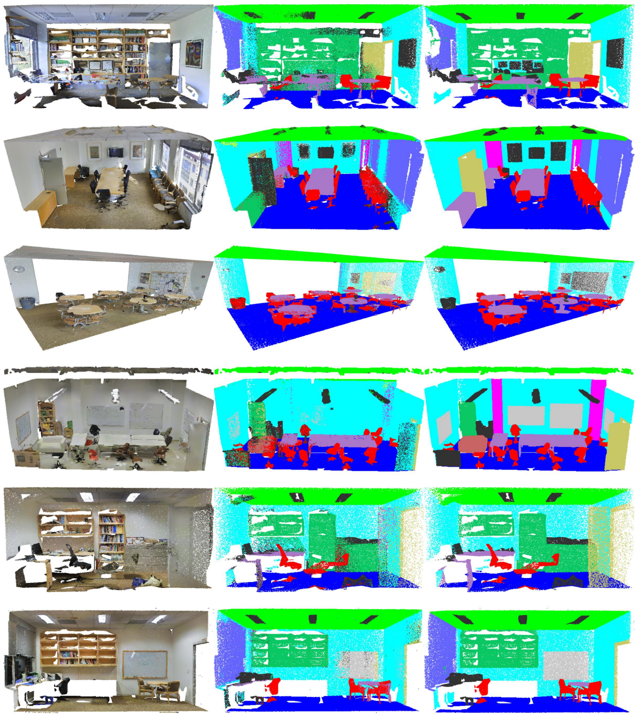


Figure 4: Sample segmentation results on the S3DIS dataset. From left to right are the input scenes, results produced by our RSNet, and ground truth. Best viewed with zoom in.

ods. The slicing resolutions r along the x, y, z axis are all set as $2cm$. 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 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 for most of the scenes in the S3DIS dataset.

During testing, the sliding stride is set as $1m$ to produce 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-arts [13, 20] are reported in Table.1 as well. In [20], the data representation is volexized 3D volumes and a 3D CNN is built for segmenting objects in the 3D volumes. Several geometric data augmentation strategies and end-to-end Conditional Random Filed (CRF) are utilized in their work. The PointNet [13] 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 our 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 [20] by 3.01 in mean IOU and 2.07 in mean accuracy. Compared with previous state-of-the-art point clouds based method [13], it improves the mean IOU by 10.84 and mean accuracy by 10.44. The detailed per-category IOU results show that our 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 modeling modules in our RSNet. Given the same input data, point clouds, our RSNet mainly differs from PointNet [13] in the local dependency module. While PointNet only relies on global features, our 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 can effectively model spatial dependencies among unstructured and unordered points. Moreover, the performance improvements against the previous state-of-the-art 3D volumes based method prove that directly handling point clouds can benefit 3D segmentation task a lot as there are no quantitation artifacts and no local details lost.

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

4.2. Ablation Studies

In this section, we validate the effects of various architecture choices and testing schemes by control experiments. In particular, several key parameters are considered: 1). the slicing resolution r in RSNet; 2). the size of sliding block; 3). the sliding stride during testing; 4). the type of RNN units. All settings 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 RSNet. 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 range in spatial region. Thus, there is a trade-off of selecting the slicing resolution r.

Several experiments are conducted to show how different slicing resolutions would impact the final performance of RSNet. 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 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 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 the RSNet 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 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 [13] , non-overlapping splitting is used while PointNet++ [15] adopted overlapping splitting. For our RSNet, 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. Ex-

Method	mIOU	mAcc	wall	floor	chair	table	desk	bed	bookshelf	sofa	sink	bathtub	toilet	curtain	counter	door	window	shower curtain	refridgerator	picture	cabinet	other furniture
PointNet [13]	14.69	19.90	69.44	88.59	35.93	32.78	2.63	17.96	3.18	32.79	0.00	0.17	0.00	0.00	5.09	0.00	0.00	0.00	0.00	0.00	4.99	0.13
PointNet++ [15]	34.26	43.77	77.48	92.50	64.55	46.60	12.69	51.32	52.93	52.27	30.23	42.72	31.37	**32.97**	20.04	2.02	3.56	27.43	18.51	0.00	23.81	2.20
Ours	**39.35**	**48.37**	**79.23**	**94.10**	**64.99**	**51.04**	**34.53**	**55.95**	**53.02**	**55.41**	**34.84**	**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.

perimental results show that using overlapped division can slightly increase the performance (0.4∼1.9 in mean IOU and 1.1∼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 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 our 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 RSNet is reported in Table.6.

In the ScanNet dataset, the previous state-of-the-art method is PointNet++ [15]. It only uses the xyz information of point clouds as inputs. To make a fair comparison, we also only use xyz information in our RSNet. [15] 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[13] and Pointnet++[15] (the single scale version) on the ScanNet dataset [1] and report them in Table.6 as well. As shown in Table.6, our RSNet has also achieved state-of-the-art results on the ScanNet dataset. Compared with the PointNet++, our 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, our 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 our RSNet with some other methods [10, 23, 24], we also report the segmentation results of our RSNet on the ShapeNet part segmentation dataset. The same RSNet as in Section 4.1 is used here. Our RSNet only takes the xyz information as convention. Its performance are reported in Table.7. Table.7 also presents the results of other state-of-the-art methods including PointNet, PointNet++, KD-net, and spectral CNN. Our RSNet outperforms all other methods except the PointNet++[15] which utilized extra normal information as inputs. However, our RSNet can also outperform PointNet++ when it only takes xyz information. This validates the effectiveness of our RSNet.

4.5. Computation Analysis

We now demonstrate the efficiency of RSNet in terms of inference speed and GPU memory consumption. We follow the same time and space complexity measurement strategy as [15]. We record the inference time and GPU memory consumption of a batch of 8 4096 points for vanilla PointNet and our RSNet using PyTorch on a K40 GPU. Since [15] reported the inference speed in TensorFlow, we use the relatively speed w.r.t vanilla PointNet to compare speeds with each other. The speed and memory measurements are reported in Table.8. There is no published information about the memory consumption of other networks. So we only compare the memory consumption with vanilla PointNet here.

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

[1] 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.

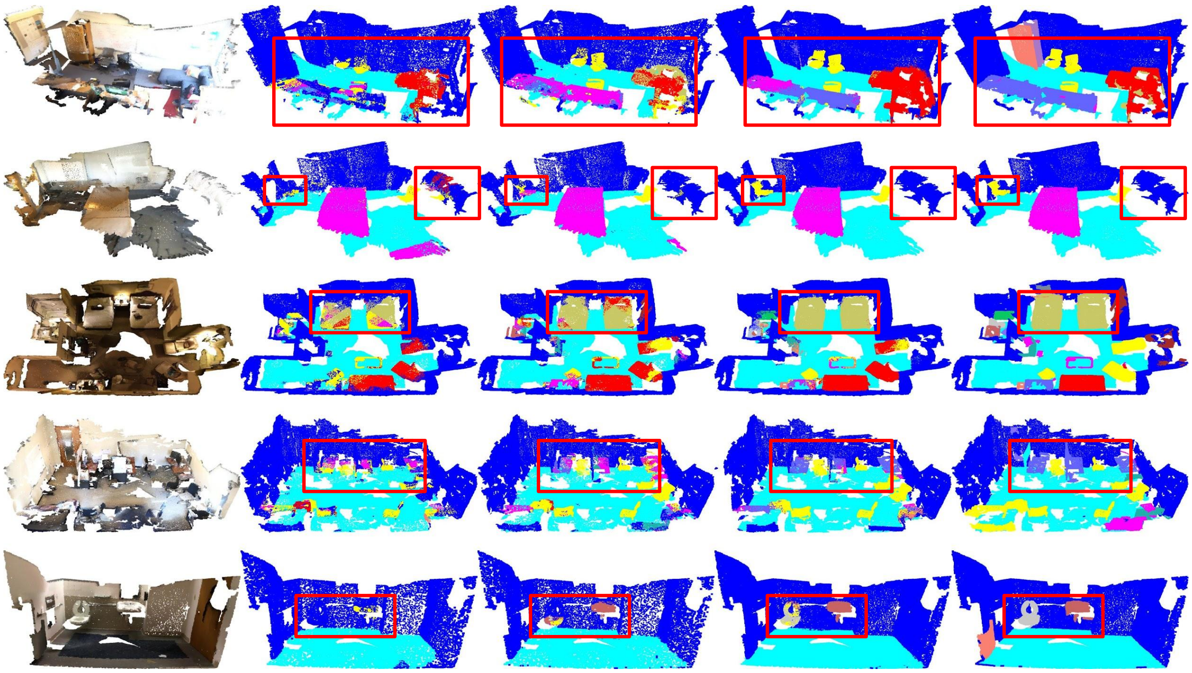


Figure 5: Sample segmentation results on the ScanNet dataset. From left to right are the input scenes, results produced by PointNet, PointNet++, our RSNet, and ground truth. Interesting areas have been highlighted by red bounding boxes. Best viewed with zoom in.

Method	mean	aero	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor	mug	pistol	rocket	skate board	table
Yi [23]	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 [10]	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 [13]	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++ * [15]	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 [24]	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++ [15]	**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.

5. Conclusion

This paper introduces a powerful and efficient 3D segmentation framework, Recurrent Slice Network (RSNet). 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.

Appendices

A. More Results and Discussions on the S3DIS dataset

In the S3DIS dataset, there are 272 indoor scenes captured from 6 areas in 3 buildings. The points are annotated in 13 categories. To process this dataset, our RSNet takes points with 9 dimensional features as inputs as in [13]. The first three, middle three, and last three dimensions represent the xyz coordinates, RGB intensities, and normalized xyz coordinates, respectively.

	PointNet (vanilla) [13]	PointNet [13]	PointNet++ (SSG) [15]	PointNet++ (MSG) [15]	PointNet++ (MRG) [15]	RSNet
Speed	1.0 ×	2.2 ×	7.1 ×	14.1 ×	7.5 ×	4.5 ×
Memory	844 MB	-	-	-	-	756 MB

Table 8: Computation comparisons between vanilla PointNet, PointNet++, and RSNet.

Method	mIOU	mAcc	ceiling	floor	wall	beam	column	window	door	chair	table	bookcase	sofa	board	clutter
PointNet [13]	47.71	-	-	-	-	-	-	-	-	-	-				
Ours	**56.47**	66.45	92.48	92.83	78.56	32.75	34.37	51.62	68.11	59.72	60.13	16.42	50.22	44.85	52.03

Table 9: 6-fold validation results on the Large-Scale 3D Indoor Spaces Dataset (S3DIS). IOU of each category is also reported.

In the main text, we used the training/testing split in [20] is used to avoid dividing areas from same building to both training and testing sets. However, in [13], the authors reported their performances using 6-fold validation. In order to comprehensively compare with [13], we also present the 6-fold validation performances of RSNet in Table.9. The results show that our RSNet outperforms PointNet by a large margin while requiring less memories and reasonable extra inference times.

Both Table.9 and the Table.1 in the main text show that while all the methods work well on some categories like ceiling, floor and wall, they all fail to achieve the same level of performances on the categories like beam, column, and bookcase. This is because the S3DIS dataset is a highly unbalanced dataset. From the data portion statistics in Table.10 we notice that ceiling, floor and wall are the dominant classes which have 7 $\sim$ 50 times more training data than the rare classes. This makes the segmentation algorithms fail to generalize well on the rare classes. In order to alleviate this problem, we adopt the median frequency balancing strategy [4] in our RSNet training. The results are compared with the baseline in Table.13. It shows that using median frequency balancing improves performances in terms of the mean accuracy. However, there is a slight decrease in mean IOU.

B. More Results and Discussions on the ScanNet dataset

The ScanNet dataset contain 1,513 scenes captured by the Matterport 3D sensor. We follow the official training/testing split [3] in this paper. The points are annotated in 20 categories and one background class. As shown in Table.11, the ScanNet dataset is also highly unbalanced. Thus, we use the mean IOU and mean accuracy as evaluation metrics in the main text to better measure the performances for this dataset. To process the ScanNet dataset, out RSNet takes points with 3 dimensional features (xyz coordinates) as inputs as in [15].

In order to further improve the performances on the ScanNet dataset, we train a RSNet taking not only xyz coordinates but also RGB intensities as inputs. The results are reported in Table.12. It shows that RGB information can slightly improve the performances of our baseline model. The mean IOU and mean accuracy are improved by 1.81 and 1.97. Moreover, detailed per-class IOUs show that the RGB information is particularly helpful for categories like door, window, and picture. These classes can be easily confused with walls when only geometric information (xyz coordinate) is present. However, RGB information helps the network distinguish them from each other.

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, 2, 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, 9

[4] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015. 9

[5] 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

	ceiling	floor	wall	beam	column	window	door	chair	table	bookcase	sofa	board	clutter
Data Per-centage (%)	25.3	23.3	17.3	2.42	1.6	1.1	4.6	3.4	5.3	0.5	3.3	0.7	11.2

.

Table 10: Data portion of each category in the training set of the S3DIS dataset.

	wall	floor	chair	table	desk	bed	bookshelf	sofa	sink	bathtub
Data Per-centage (%)	36.8	24.9	4.6	2.5	1.7	2.6	2.0	2.6	0.3	0.3
	toilet	curtain	counter	door	window	shower-curtain	refridgerator	picture	cabinet	other furniture
Data Per-centage (%)	0.3	1.5	0.6	2.3	0.9	0.2	0.4	0.4	2.6	2.5

.

Table 11: Data portion of each category in the training set of the ScanNet dataset.

Method	mIOU	mAcc	wall	floor	chair	table	desk	bed	book-shelf	sofa	sink
RSNet	39.35	48.37	79.23	94.10	**64.99**	**51.04**	34.53	**55.95**	53.02	**55.41**	34.84
RSNet with RGB	**41.16**	**50.34**	**79.38**	**94.21**	63.65	48.67	**35.27**	53.09	**53.67**	51.06	**41.00**
Method	bathtub	toilet	curtain	counter	door	window	shower curtain	refrid-gerator	picture	cabinet	other furniture
RSNet	49.38	54.16	6.78	**22.72**	3.00	8.75	**29.92**	37.90	0.95	**31.29**	18.98
RSNet with RGB	**60.37**	**63.20**	**8.30**	20.90	**15.32**	**15.67**	**24.36**	**39.76**	**4.30**	30.06	**20.98**

Table 12: Results on the ScanNet dataset. IOU of each category is also reported here.

Method	mIOU	mAcc
RSNet	**51.93**	59.42
RSNet-*median*	48.68	**62.09**

.

Table 13: Results of different training strategies on the S3DIS dataset.

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

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

[10] 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

[11] 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

[12] 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

[13] 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, 9

[14] 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

[15] 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, 9

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

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

[18] 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

[19] 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

[20] 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, 9

[21] 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

[22] 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

[23] 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

[24] 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

[25] 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