Reconfigurable Voxels: A New Representation for LiDAR-Based Point Clouds
Supplementary Materials

Tai Wang, Xinge Zhu, Dahua Lin
The Chinese University of Hong Kong
{wt019, zx018, dhlin}@ie.cuhk.edu.hk

1 Algorithm Specification

Here we give the specification for the algorithm of contructing reconfigurable voxels in the multi-resolution case (Algorithm 1). To sum up, we can complete this in one traversal of all point clouds for every sample: for every point, locate its coordinates in two-resolution voxel maps at first and record the adjacency of relevant voxels; then for all neighbor voxels, compute transition distributions and implement reconfiguration; finally, resample points in larger voxels.

2 Supplementary Results

2.1 Quantitative Results

In this section, we present more quantitative analysis on Lyft and detailed results compared with other state-of-the-art methods on KITTI.

First, given the similarity of data format on nuScenes [1] and Lyft [2], it is convenient to compare performance of different methods on the Lyft under nuScenes metrics. Considering that we do not need to predict velocity and attribute on the Lyft, NDS is not suitable for evaluation. So we test their distance-based mAPs by categories on our split Lyft validation set, where *v1* and *v2* represent different feature fusion methods in Reconfigurable PointPillars.[1] See more details in Sec. 4.

Results of 7 categories based on our reproduced Fast PointPillars are shown in Tab. 1. Two other categories, animal and emergency vehicle, are not shown here because their training data is too limited (only 186 animal instances and 132 emergency instances compared with 534911 cars). It can be seen that our representation can greatly enhance the detection performance, especially for small objects. For example, our model can increase mAPs of pedestrian and bicycle by up to 5.7% and 13.6%. Note that motorcycle also has limited 818 instances, so the improvement is not much noteworthy. Finally, the original model can be improved by 3.2% mAP for all objects and 5.7% mAP for small objects. It further demonstrates the efficacy of our representation.

Then detailed results on the KITTI benchmark are shown in Tab. 2, which lists several methods using point clouds as input, including multi-modal data fusion methods, point-based methods and voxel-based methods. Note that only most of the published methods which provide test results for all 3 types of objects are listed here. In the same way, our method improves the detection performance of cyclist and pedestrian. The improvement for PointPillars is mainly focused on the pedestrian. We speculate that there may be differences in the network details between the baseline and the original paper. In fact, the reproduced baseline cannot fully achieve the decent performance of cyclist as the original paper. In addition, we find that for the detection of multi-class objects, there are certain mutual constraints between different categories, so maybe the performance improvement of cyclist and pedestrian can maintain a certain level together, but one can influence the other. The specific solution may include adopting different detection heads, which will be our future work.

[1]To use our metrics, we split the official training data into train/val set by the same ratio as nuScenes.

Table 1: Distance-based mAP by categories compared to PointPillars on the Lyft val 3D detection benchmark. Note that the results here are from the experiment with Fast PP as baseline in the paper. According to the average size of all the bounding boxes, we consider the first 4 categories (car, other vehicle, bus and truck) as large objects while the last 3 categories (pedestrian, bicycle and motorcycle) as small objects. We compute the mAP of all the small objects and record it as mSAP in the table

Method	Car	Other Veh.	Bus	Truck	Ped	Bicycle	Moto	**mAP**	**mSAP**
PointPillars	93.5	63.1	**46.9**	43.1	48.1	37.4	3.9	48.0	29.8
Reconfig PointPillars (sing-res v1)	93.7	65.6	44.1	46.1	**53.8**	46.6	2.1	50.3	34.2
Reconfig PointPillars (sing-res v2)	92.8	61.5	42.1	42.7	50.9	47.4	**5.6**	49.0	34.6
Reconfig PointPillars (multi-res)	**93.8**	**65.7**	45.7	**46.8**	52.4	**51.0**	3.1	**51.2**	**35.5**

Table 2: Results on the KITTI dataset

Method	Reference	Modality	**mAP**	Car			Cyclist			Pedestrian		
				Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
AVOD-FPN [7]	IROS 2018	LiDAR+RGB	56.84	83.07	71.76	65.73	63.76	50.55	44.93	50.46	42.27	39.04
F-PointNet [8]	CVPR 2018	LiDAR+RGB	57.86	82.19	69.79	60.59	72.27	56.12	49.01	50.53	42.15	38.08
F-ConvNet [9]	IROS 2019	LiDAR+RGB	63.15	87.36	76.39	66.69	**81.98**	**65.07**	56.54	52.16	**43.38**	38.80
PointRCNN [10]	CVPR 2019	LiDAR	60.33	86.96	75.64	70.70	74.96	58.82	52.53	47.98	39.37	36.01
STD [12]	ICCV 2019	LiDAR	63.60	**87.95**	**79.71**	**75.09**	78.69	61.59	55.30	**53.29**	42.47	38.35
VoxelNet [5]	CVPR 2018	LiDAR	50.99	77.47	65.11	57.73	61.22	48.36	44.37	39.48	33.69	31.51
Part A2 [11]	TPAMI 2020	LiDAR	**63.99**	87.81	78.49	73.51	79.17	63.52	**56.93**	53.10	43.35	**40.06**
SECOND [13]	Sensors 2018	LiDAR	58.35	83.13	73.66	66.20	70.51	53.85	46.90	51.07	42.56	37.29
+Reconfig	-	LiDAR	+1.31	+0.88	-0.33	+1.53	+1.08	+2.00	+2.68	+1.05	+1.14	+1.75
PointPillars [6]	CVPR 2019	LiDAR	60.80	82.58	74.31	68.99	77.10	58.65	51.92	51.45	41.92	38.89
+Reconfig	-	LiDAR	+0.68	+0.78	+0.21	+0.29	+0.43	-0.23	+0.27	+1.80	+1.4	+1.14

We can also see that point-based methods dominate the KITTI benchmark in terms of performance currently. So the trade-off between performance and efficiency as well as how to address the irregularity distribution problem of LiDAR data in point-based methods are worthy of further study.

2.2 Qualitative Results

In this section, we give some examples of detection results on nuScenes. Through these examples, we can intuitively observe the detection results and see the improvement of our model in detecting small and distant objects.

In Fig. 1, the near barriers in the first group of samples and the far-away little occluded cars in the second group of samples shows the improvement when detecting small and distant objects, while the last 2 groups of samples show that the improved model reduces false positive detections of large objects in the distance and small objects in the near.

Besides, we can see some interesting phenomena from the failure examples in these results. For instance, the vehicle detected by mistake in the last sample is closely related to the roadside building, which indicates that the detector sometimes cannot distinguish the corner of car and building. In the third sample, both models detect an obstacle that was not annotated.

3 Detailed Analysis of Change in Distribution

As mentioned in the paper, our reconfigurable voxels effectively mitigate the difficulty caused by sparsity and irregularity, which can be reflected in the more balanced distribution with respect to number of points per voxel. So in this section, we give quantitative and qualitative results in this respect.

3.1 Quantitative Analysis

To make this result more convincing, we conduct the study on the validation set of nuScenes instead of just on several samples. Considering the number of points involved in every voxel can be regarded as a kind of *ratio scale* in some sense, we use the *coefficient of variation* as the indicator to measure their dispersion of the distribution. Tab. 3 shows the result, which is consistent with our observation on the qualitative results.

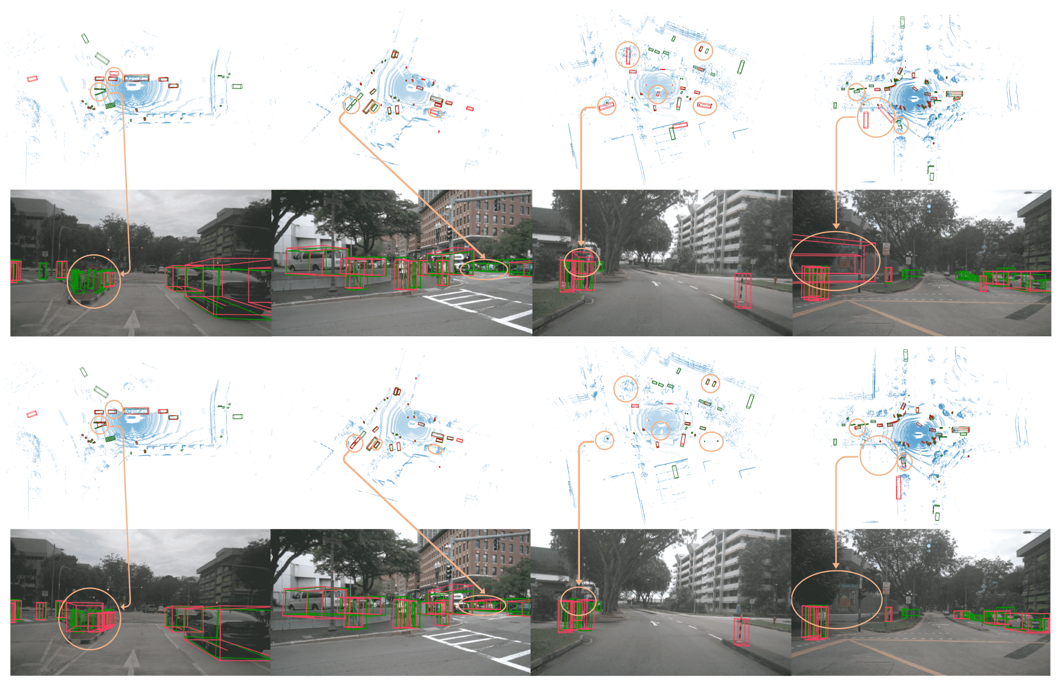
Figure 1: Qualitative analysis of nuScenes results. We show 3D bounding boxes, predicted results in red and ground truth in green, both in LiDAR point cloud and their projection into the image for visualization. The top 2 and bottom 2 rows are the results of our baseline and the model improved by reconfigurable pillars respectively. Less false and more correct detection of small and distant objects shows the improvement, which are marked with orange circles. Note that apart from the objects can be seen in images, there are more samples marked with orange circles in bird view

Table 3: Coefficient of variation w.r.t. points in original voxels and reconfigurable voxels

Method	Coefficient of Variation
PointPillars	0.9766
Reconfig PointPillars (sing-res)	0.7695
Reconfig PointPillars (multi-res)	**0.6796**

3.2 Qualitative Analysis

Fig. 2 shows the change in distribution with respect to number of points per voxel.[2] It reveals that the imbalance problem is alleviated and the reconfiguration in the multi-resolution case works even better than the single-resolution case. Specifally, voxels containing only one point are reduced and some of them are changed into voxels with decent number of points meanwhile.

4 Implementation Details

We basically follow two voxel-based frameworks, SECOND [13] and PointPillars [6], in the experiments. Different settings of these two frameworks are listed in Tab. 4. We follow the original method of PointPillars and SECOND in the setting of detection range and other details such as anchors and matching strategy except a few adjustments for network designs. Next in this section, we first present details of implementing random walk in different frameworks, and then elaborate the network designs in terms of three different modules shown in Fig. 3.

Firstly, in terms of specific parameter setting of random walk, in order to ensure that the neighbor voxels will not go too far away due to random walk, we divide the number of points by 4 and round

[2]Note that number of points per reconfigurable voxel refers to the average number of points in 5 voxels contained in the representation.

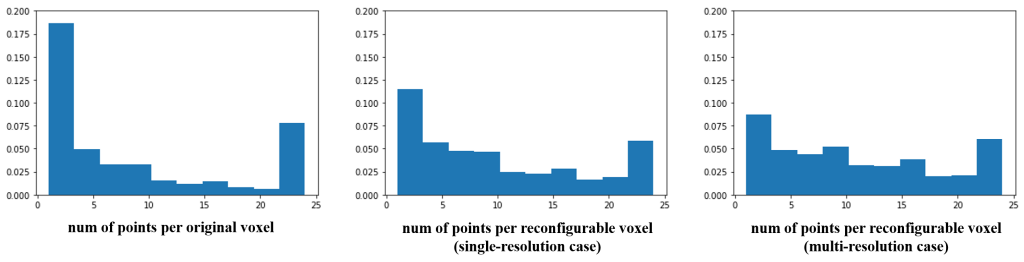

Figure 2: Change in distribution w.r.t number of points per voxel.

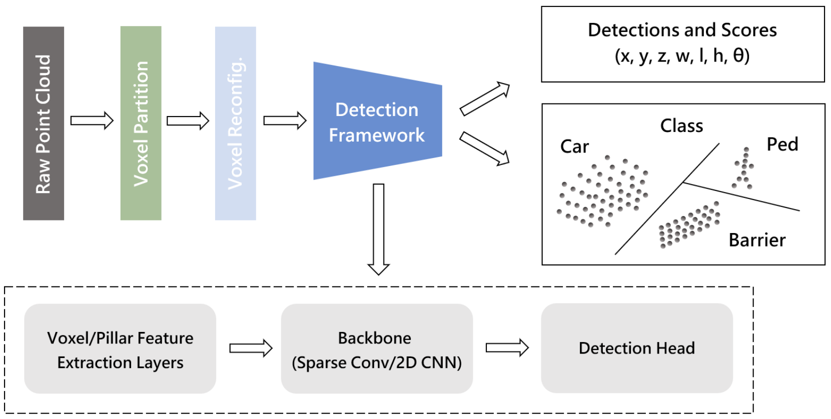

Figure 3: Overview of our framework.

them up when we calculate Eqn. 1 and 2 in PointPillars, then the resulting steps range and the probability space of random walk are similar with the situation of SECOND. Details of Eqn. 1 and 2 can be referred to the main paper.

Then for network designs, we have stated basic ideas of the first module, feature extraction layers, in the paper (Sec. 3.3). To review, we briefly summarize it as follows. The voxel encoder in SECOND takes all the $5n$ points as input and processes their features by average pooling. In comparison, we tried two kinds of pillar feature extractors. Both of them encode features of original pillar and neighbor pillars respectively and then concatenate them along feature channels. The difference lies in the way of encoding the neighbor pillar features. The first version takes a weighted sum of these 4 neighbor pillar features at first while the second version directly encodes these $4n$ point features. As is shown in the results on the Lyft, the first version works better on the whole possibly because of the benefit from adaptive weights, and so it is adopted as our final implementation method. Specific parameters are shown in Fig. 4.

The second module, convolutional backbone is shown in Fig. 5. In SECOND, the sparse convolutional backbone takes the 4D feature map as input and processes it by several submanifold convolution and sparse convolution layers. In Fig. 5, the 4 downsampling steps consist of (2,1), (2,1), (3,1), (4,1)

Table 4: Different settings in Reconfigurable PointPillars (Fast version) and SECOND experiments. To make experiments more comparable, PointPillars and SECOND in our experiments are both reproduced version, which share similar hyper parameters with the published ones while are improved on some details, thus have better performance in most cases. Our reconfigurable methods follow the same settings as well

Method	PointPillars	SECOND
Pillar/Voxel Input Features	$d, z, t, x_c, y_c, z_c, x_p, y_p$	d, z, r
Pillar/Voxel Resolution (m)	0.25×0.25	$0.05 \times 0.05 \times 0.1$
Max number of Pillars/Voxels	25000	30000
Max number of points per voxel	25	4
Convolutional Backbone	2D CNN	3D Sparse Convolution
Region Proposal Network	multi-group head	original RPN

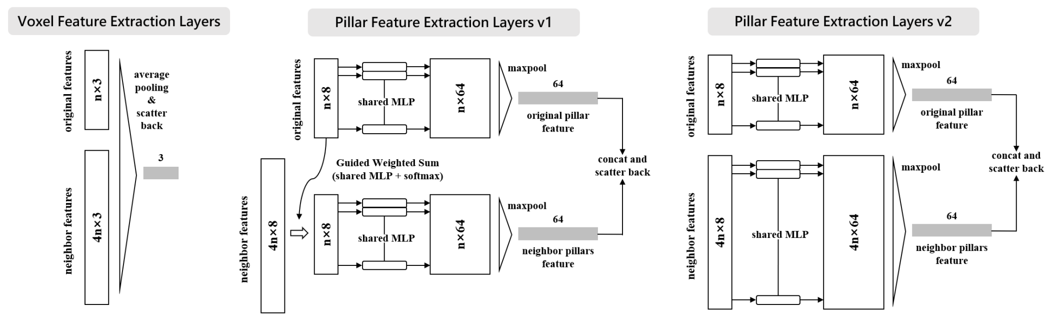

Figure 4: Voxel/Pillar Feature Extraction Layers.

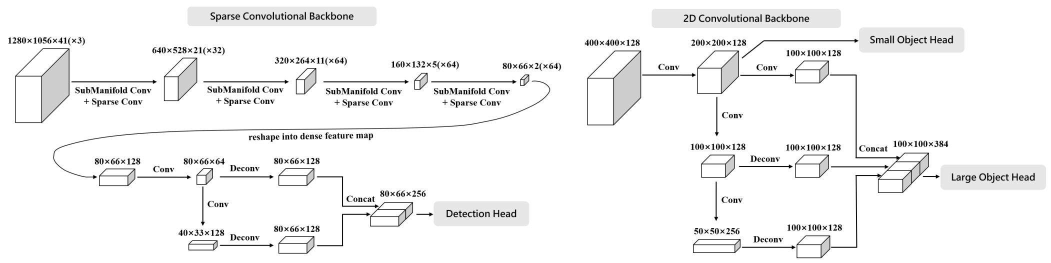

Figure 5: Backbone in SECOND and PointPillars.

submanifold convolution and sparse convolution layers respectively. Then we utilize a top-down network to produce feature maps in two resolutions, perform upsampling and concatenation to derive the final feature map used in the subsequent detection head. In PointPillars, instead of preprocessing by sparse convolution layers, we just perform top-down convolution, upsampling and concatenation to get the input of large object head. In comparison, the shallow feature map is used to detect small objects. All the top-down convolutional layers contain 1 downsampling layer with stride = 2 and 5 layers with stride = 1 except the first convolution in PointPillars, which has 1 downsampling layer with stride = 2 and 3 layers with stride = 1.

The last module, detection head, just derives box map, class map and attribute map (if necessary) from the input feature map like SSD [14]. In PointPillars, the large object head is set the same as the head in SECOND while the small object head takes 3 extra convolutional layers to compress the feature map to 64 feature channels first.

So far, we have introduced the details of network architectures. In conclusion, our implementation of SECOND is mainly different from that of PointPillars in terms of encoding methods, preprocessing in convolutional backbone and whether to design specific heads for different object categories. See the detailed structures and intermediate results in Fig. 4 and Fig. 5.

Algorithm 1 Multi-resolution Reconfig. Voxel Partition

```
Require: (1) point cloud data $P = \{p_i, i = 1, ..., n\}$; (2) maximum number of points per voxel $m$;
    (3) maximum number of voxels $N$ (in resolution 1);
Ensure: (1) voxel coordinates of two resolutions ($C_1$, $C_2$); (2) voxel features of two resolutions ($F_1$,
    $F_2$); (3) number of points in voxels of two resolutions ($N_1$, $N_2$); (4) graph $G$ (to record indices
    and resolutions of voxel neighbors);
 1: for every point $p_i$ do
 2:     if $p_i$ is not in the detection range then
 3:         continue;
 4:     end if
 5:     // Denote variables of smaller voxel with index 1
 6:     // and larger one with index 2
 7:     Locate its voxel coordinates $c_1$ in resolution 1;
 8:     Locate its voxel coordinates $c_2$ in resolution 2;
 9:     if voxel1 at $c_1$ not recorded yet then
10:         if number of recorded voxels1 $\geq N$ then
11:             break;
12:         end if
13:         create a new voxel1;
14:         if voxel2 at $c_2$ not recorded yet then
15:             create a new voxel2;
16:             if left/right/back/front neighbor exists then
17:                 record their adjacency (in resolution2);
18:             end if
19:         end if
20:         record the parent index for voxel1;
21:         record the children index for voxel2;
22:         if left/right/back/front neighbor exists then
23:             record their adjacency (in resolution1);
24:         end if
25:     end if
26:     if number of points in that voxel $< m$ then
27:         add the point features of voxel1 into $F_1$;
28:     end if
29: end for
30: compute transition distribution probabilities according to number of points in voxels;
31: for every voxel1 $v_i$ do
32:     for every neighbor of $v_i$ do
33:         compute number of steps $S$;
34:         conduct random walk for $S$ times;
35:     end for
36:     record final adjacency into $G$;
37: end for
38: resample points of voxels2 (up to $m$ points per voxel) and record them into $F_2$
```

Acknowledgments

This work is partially supported by the SenseTime Collaborative Grant on Large-scale Multi-modality Analysis (CUHK Agreement No. TS1610626 & No. TS1712093), the General Research Fund (GRF) of Hong Kong (No. 14236516 & No. 14203518).

References

[1] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. *CoRR*, abs/1903.11027, 2019. URL http://arxiv.org/abs/1903.11027.

[2] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Lyft level 5 av dataset 2019. https://level5.lyft.com/dataset/, 2019.

[3] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *CoRR*, abs/1908.09492, 2019. URL http://arxiv.org/abs/1908.09492.

[4] Lyft 3d object detection for autonomous vehicles. https://www.kaggle.com/c/3d-object-detection-for-autonomous-vehicles/leaderboard.

[5] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[6] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[7] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander. Joint 3d proposal generation and object detection from view aggregation. In *International Conference on Intelligent Robots and Systems*, 2018.

[8] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[9] Z. Wang and K. Jia. Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection. In *International Conference on Intelligent Robots and Systems*, 2019.

[10] S. Shi, X. Wang, and H. Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[11] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *arXiv preprint arXiv:1907.03670*, 2019.

[12] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia. Std: Sparse-to-dense 3d object detector for point cloud. In *IEEE International Conference on Computer Vision*, 2019.

[13] Y. Yan, Y. Mao, and B. Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18 (10), 2018.

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision*, 2016.