arXiv:1807.06288v5 [cs.CV] 14 Sep 2018

PointSeg: Real-Time Semantic Segmentation Based on 3D LiDAR Point Cloud

Yuan Wang[1] Tianyue Shi[2] Peng Yun[1] Lei Tai[1] Ming Liu[1]

Abstract— **We propose *PointSeg*, a real-time end-to-end semantic segmentation method for road-objects based on spherical images. The spherical image is transformed from the 3D LiDAR point clouds with the shape $64 \times 512 \times 5$ and taken as input of the convolutional neural networks (CNNs) to predict the point-wise semantic mask. We build the model based on the light-weight network, *SqueezeNet*, with several improvements in accuracy. It also maintains a good balance between efficiency and prediction performance. Our model is trained on spherical images and label masks projected from the *KITTI* 3D object detection dataset. Experiments show that *PointSeg* can achieve competitive accuracy quickly with 90pfs on a single GPU, which makes this real-time semantic segmentation task quite compatible with the robot applications.**

I. Introduction

A. Motivation

3D real-time semantic segmentation plays an important role in the visual robotic perception application, such as in autonomous driving cars. Those robotic systems collect information from the real-time perception based on different modes of sensors, and understand which and where objects are on the road. Different applications make decisions based on different perceptions, such as camera, inertial measurement units (IMUs) and LiDAR. LiDAR scanner is one of the most essential components where we can directly get the space information from. It is also less influenced by light compared with cameras and has robust features in challenging environments. Therefore, a fast 3D semantic segmentation method will help robot understand the world information more directly. In addition, computation power on an autonomous driving system is quite limited to maintain those state-of-the-art sources consuming methods. Even in the workstations, the semantic segmentation is still difficult to achieve the real-time performance. Moreover, embedded computing devices, such as Jetson TX2 and FPGA, cannot provide the same level computation ability as those normal workstations. Because of this, a good perception method with high accuracy, low cost memory and compatible real-time performance has become a crucial problem, which has attracted many research attentions.

Previous approaches about point clouds recognition [1] [2] mainly rely on complicated hand-crafted features, such as surface normal or generated descriptors, and hard threshold decision rules based on a clustering algorithm. These approaches have two problems: (1) hand-crafted features cost much time and the results by hard threshold decision are not suitable for productions; (2) they can not recognise the pixel level object category as the same as the semantic segmentation, which makes it difficult to apply in some autonomous driving tasks.

To solve these problems, we design a light network architecture for the road-object segmentation task, which is called PointSeg. The pipeline is shown in Fig. 1. Our network predicts a point-wise map on the spherical image and transforms this map back to 3D space.

B. Contributions

To achieve compatible real-time performance, we take the light-weight network *SqueezeNet* [3] as our root structure. Then we take several ideas from the state-of-the-art RGB semantic segementation methods like PSPNet [4] and apply them in our network together to achieve the state of the art performance. Because 3D point cloud data is natrually sparse and large, it is arduous to build real-time semantic segmentation task. We solve this problem by transforming the point cloud data into a spherical image and make PointSeg accept the transformed data.

Generally, we propose a fast semantic segmentation system for autonomous driving with the following features:

- The model is quite light-weight. We extend the basic light-version network *SqueezeNet* [3] with new feature extract layers to improve balance between accuracy and computation efficiency in 3D semantic segmentation task.
- Our network can be applied directly on the autonomous driving system with limited memory cost and easily implementation with basic deep learning unit.

II. Related works

We will discuss some recent approaches about semantic segmentation network structure, deep learning in the 3D point cloud data, bounding box detection tasks and semantic segmentation tasks.

A. High Quality Semantic Segmentation for Image

FCN [5] was the pioneering method for semantic segmentation based on deep learning. It replaced the last fully-connected layers in classification with convolution layers. Recent approaches like DeeplabV3 [6] used a dilated convolutional layer [7] and the conditional random field (CRF) [8] to improve the predicition accuracy. SegNet [9] used an encoder-decoder architecture to fuse the feature maps from the deep layer with spatial information from lower

[1]The Hong Kong University of Science and Technology, {ywangeq, pyun, ltai, eelium}connect.ust.hk
[2]Nanjing University of Aeronautics and Astronautics iristainyue@gmail.com

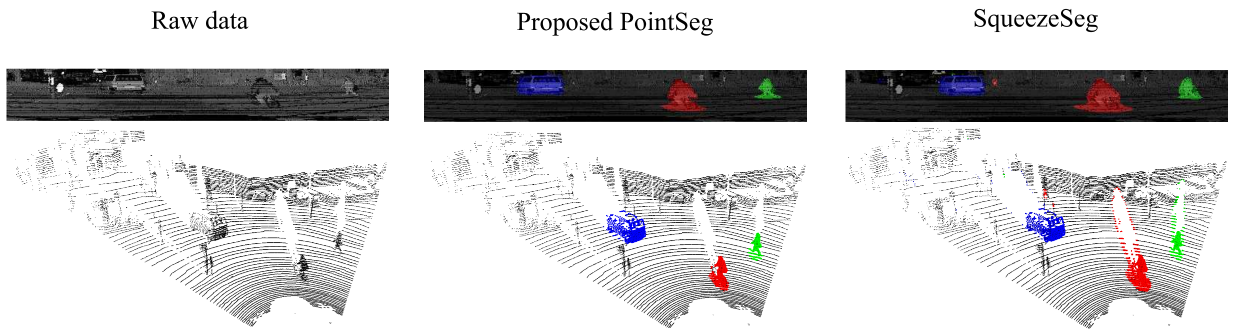

Fig. 1: The piepeline of PointSeg. The first column shows the input spherical data and its corresponding 3D point cloud data. The second column shows the predicted mask results and back projection results of PointSeg. The third column show the predicted mask results and back projection results of SqueezeSeg [21]. Cars, cyclists and pedestrians are shown in blue, red and green.

layers. Other approaches like ICnet [10], RefineNet [11] took multi-scale features into consideration and combined image features from multiple refined paths. Among those methods, Networks like Deeplabv3 and FCN were compelled to increase the performance. SegNet and ICNet are able to achieve real-time performance. Although, they have big improvement in speed or accuracy, these methods still have some influence on the other side.

B. *Convolutional neural networks with 3D point cloud data*

3D data has sufficient features and attracts much research attention. With the rapidly developing of deep learning, many methods apply convolutional neural networks (CNN) on the 3D point cloud data directly. 3DFCN [12] and VoxelNet [13] used a 3D-CNN [14] to extract features from width, height and depth simultaneously. MV3D [15] fused multi-perception from a bird's-eye view, a front view and a camera view to obtain a more robust feature representation. In addition, some works [16] considered the representation of three-dimensional data itself and divided it into voxels to undertake features such as intensity, distance, local mean and disparity. Although all of above methods have achieved a good accuracy, they still cost too much time in computation which limited their applications in real time tasks. In this paper, we are aiming to improve the real time performance and keep a good accuracy at same time.

C. *Segmentation for 3D Lidar point cloud data*

Previous works proposed several different algorithms for plane extractions from 3D point clouds, such as RANSAC-based (random sample consensus) [17] methods and region-grow-based methods [18]. However, RANSAC requires much computation on random plane model selection. Region-grow-based methods, depending on the manually designed threshold, are not adaptive. Other traditional approaches based on clustering algorithms just realized the segmentation work but not pixel-wise region classifications.

Recently, researchers started focusing on the semantic segmentation of 3D Lidar point cloud data. PointNet [19] explored a deep learning architecture to do the 3D classification and segmentation on raw 3D data. However, it only works well in indoor. Also Dubé [20] explored an incremental segmentation algorithm, based on region growing, to improve the 3D task performance. However, real-time performance is still challenging. *SqueezeSeg* [21] is similar with our task which used the *SqueezeNet* [3] as the backbone and perfomed compatible results. However, it only referred the CRF to improve the performance in the predicted 2D spherical masks, which could lose location information in the 3D space. Without considering the 3D constraints in the original point clound, the results of *SqueezeSeg* is exremely limited by this CRF post-process.

III. Method

We will discuss the generation of spherical image and key features of network structure in this section. Network structures and parameters are also included in the end of section.

A. *Spherical image generation from point cloud*

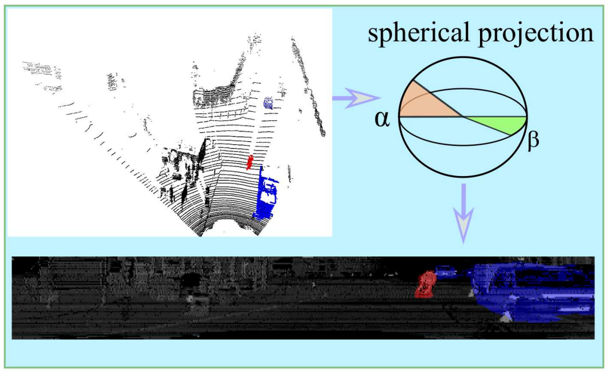

Fig. 2: The spherical projection process from a point cloud to an dense spherical image. The colored masks are cropped from ground truth boundary boxes of the KITTI dataset.

A 3D Lidar point cloud is often stored as a set of Cartesian coordinates (x, y, z). We can also easily obtain the extra feature, such as RGB values (if Lidar has been calibrated with a camera) and intensities. However, the 3D Lidar point cloud is usually sparse and huge. Therefore, transforming it into voxels and then feeding voxel representations into a 3D-CNN [14] would be computationally inefficient and memory-consuming because many voxels would be empty. To solve this problem, we transform the Lidar point cloud data by spherical projection to achieve a kind of dense representation as:

$$\alpha = arcsin(\frac{z}{\sqrt{x^2+y^2+z^2}}), \bar{\alpha} = \lfloor \frac{\alpha}{\Delta\alpha} \rfloor, \tag{1}$$

$$\beta = arcsin(\frac{y}{\sqrt{x^2+y^2}}), \bar{\beta} = \lfloor \frac{\beta}{\Delta\beta} \rfloor, \tag{2}$$

where α and β are the azimuth and zenith angles, as shown in Fig. 2; $\Delta\alpha$ and $\Delta\beta$ are the resolutions which can generate a fixed-shape spherical image; and $\bar{\alpha}$ and $\bar{\beta}$ are indexes which set the positions of points on the 2D spherical image. After applying Equation 1 and Equation2 on the point cloud data, we obtain an array as $H \times W \times C$. Here, the data is generated from Velodyne HDL-64E LiDAR with 64 vertical channels. Therefore, we set $H = 64$. Considering that in a real self-driving system most attentions are focusing on the front view, we filter the dataset to only consider the front view area $(-45°, 45°)$ and discretize it into 512 indexes, so W is 512. C is the channel of input. In our paper, we consider x, y, z coordinates, intensity for each point and distance $d = \sqrt{x^2+y^2+z^2}$ as five channels in total. Therefore, we can obtain the transformed data as $64 \times 512 \times 5$. By this transformation, we can input it into traditional convolutional layers.

We directly extract features from the transformed data, which has dense and regular distribution. Time cost is dramatically reduced compared with taking raw 3D point clound as inputs.

B. Network structure

The proposed *PointSeg* has three main functional layers: (1) fire layer (from *SqueezeNet* [3]), (2) squeeze reweighting layer and (3) enlargement layer. The network structure is shown in Fig. 3.

1) Fire layer: Assessing *SqueezeNet*, we find that its fire unit can construct a light-weight network which can achieve similar performance as AlexNet [23] but costing far fewer parameters than AlexNet. Therefore, we take the fire module as our basic network unit. We follow *SqueezeNet* to construct our feature extraction layer, which is shown in Fig. 3 (Fire1 to Fire9). The fire module is shown in Fig. 4 (a). During the feature extraction dowasampling process, we use the left one to replace the common convolutional layer. The fire module contains one squeeze module and one expand module. The squeeze module is a single 1×1 convolution layer which compresses the model's channel dimensions from C to $1/4C$. C is the channal number of the input tensor. And the expand module with one 1×1 convolutional layer and one 3×3 convolutional layer help the network to achieve more feature representations from different kernel sizes.

2) Enlargement layer: Pooling layers are set to expand the receptive field and discard the location information to aggregate the context information. However, the semantic segmentation still demands the location information. So, in *PointSeg*, we reduce the number of pooling layers to keep more location information. To solve this problem, it does not use the pooling layer to get a large receptive field after Fire9 (and SR-3). Instead, we use a dilated convolutional layer to achieve a larger receptive field. And similar as Atrous Spatial Pyramid Pooling (ASPP), which is proposed by L.Chen [24], we use different rates of the dilated convolutional layer to get a multi-scale features receptive field at the same time. The structure of the enlargement layer is shown in Fig. 5.

One 1×1 convolutional layer and one global average layer are also added in the enlargement layer. Because the input feature shape is 64×64, we set the rates as 6, 9 and 12 in three dilated convolutional layers. By doing this, we can avoid that too many zeros are added between two pixels in the traditional dilated convolutional layer and get more neighboring information. After concatenating them together, the 1×1 convolutional layer was used to compress the channel from the original size to $1/4$ to avoid too much time cost in computation.

3) Squeeze reweighting layer: To help the network get a more robust feature representation with the limitation of time cost, we proposed a reweighting layer to tackle this issue and exploit channel dependencies efficiently.

To get the squeeze global information descriptor, we simply use a global average pooling layer to achieve this. Calculating all elements χ through spatial dimensions $H \times W$ from C channels, we have:

$$\chi_n = \frac{1}{H \times W} \sum_{i=1,j=1}^{H,W} p_n(i,j), n \in [1, C], n \in int \tag{3}$$

After getting the channel-wise representations (which are shown in Fig. 6) with a shape of $1 \times 1 \times C$, which are expressive for the whole feature maps, we use two fully connected layers to generate channel-wise dependencies (called Scale in Fig. 6). To reweight the channel dependencies, Y is formulated as

$$Y_n = X_n \cdot S_n, n \in [1, C], n \in int \tag{4}$$

where $X_n = sigmod(\chi_n)$, S_n donates the output of fully connected layer as shown in Fig. 6. Here we use sigmod function to map χ_n between 0 and 1. After squeeze reweight layer, the channel weights can be adapted to the feature representation with its own global information descriptors.

4) Details in the network: We have limited information on height because the input shape is $64 \times 512 \times 5$. To solve this problem, we only do the downsampling process along the width and keep the same dimensions in height. To get the original scale resolution for point-wise prediction, we use a deconvolutional layer to upsample the feature maps from output features of SR-3 (squeeze reweighting layer) and EL layer (enlargement layer shown in Fig. 3). Although we can

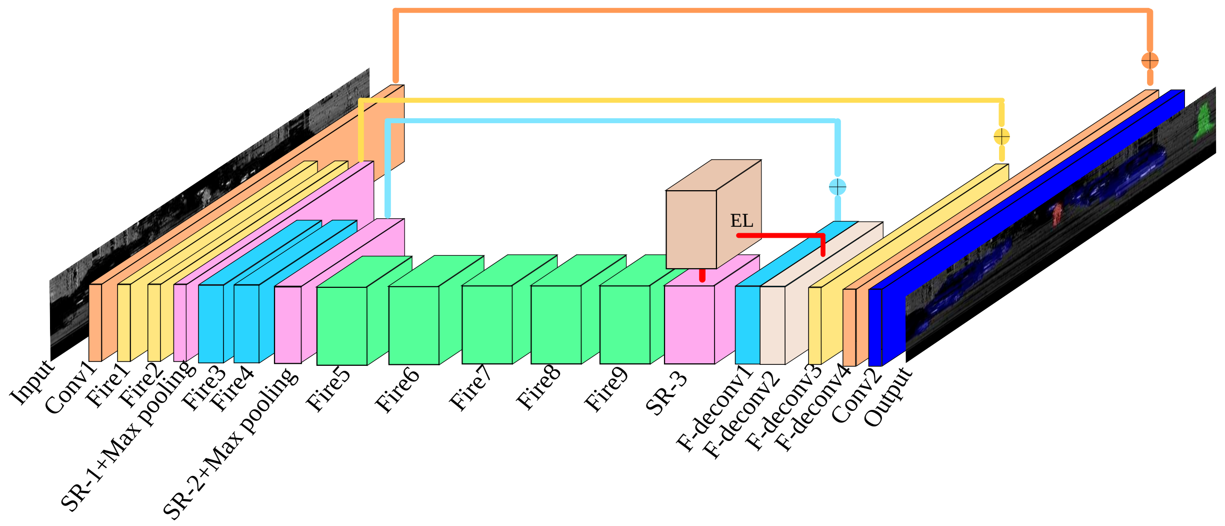

Fig. 3: The network structure of *PointSeg*. *Fire* is the fire layer. *EL* means the enlargement layer and *SR* is the squeeze reweighting layer.

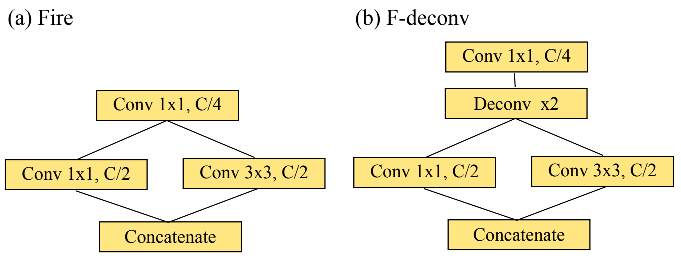

Fig. 4: (a) is the fire model in the downsampling process. (b) is the f-deconv model in the upsampling process.

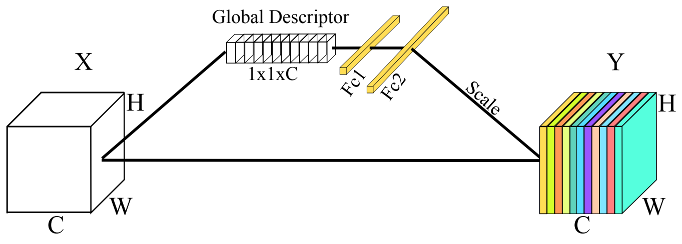

Fig. 6: The structure of Squeeze reweight layer. The *Global Descriptor* is generated from former feature maps ,and it will reweigh the channel-wise feature.

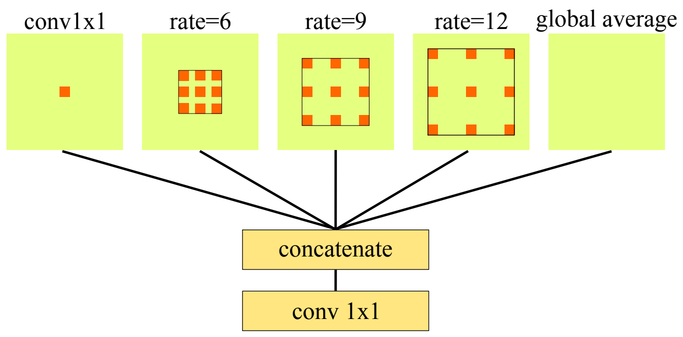

Fig. 5: The structure of the enlargement layer. Here rate means the skipping nodes between two sampled nodes. The orange points represent the sampled nodes from related feature outputs.

get a large receptive field from enlargement layer, this layer was not used any more in other layers because it will increase the parameters amounts. We also only add three squeeze reweight layers (from SR1 to SR3) before each pooling layer to help each fire block learn more robust features and reduce memory cost. We replace the deconvolutional layers with F-deconv as shown in Fig. 4. Because the feature of the enlargement layer is from a different receptive field, we concatenate the outputs from enlargement layer and SR layer after F-deconv1 (shown in Fig. 3). Skip connections are used to fuse low-level features from layers with high-level features in the other F-deconv layers. To reduce computation cost, we use add instead of $concat$ in these skip operations.

IV. Experimental Evaluation

All experiments are performed on a workstation with a 1080ti GPU under CUDA 9 and CUDNN V7. During training, we set the learning rate as 0.001 and use the Adagrad [25] as the optimizer. And we set the batch size as 32 to train the whole network for 50000 steps in around three hours

A. Dataset and evaluation metrics

We convert velodyne data from the KITTI 3D object detection dataset, and we split this dataset into a training set with around 8000 frames and a validation set with around 2800 frames for the evaluation. We evaluate the system performance on class-level segmentation tasks. The evaluation precision, recall and IoU are defined as follows:

$$P_n = \frac{|\rho_n \bigcap G_n|}{|\rho_n|}, R_n = \frac{|\rho_n \bigcap G_n|}{|G_n|}, IoU_n = \frac{|\rho_n \bigcap G_n|}{\rho_n \bigcup G_n}$$

where ρ_n is the predicted sets that belong to class-n, and G_n is the ground-truth sets.

B. Experimental results

1) Downsample times: The *SqueezeNet* [3] has four downsampling processes. The end feature shape is (1/16H×1/16W×C). In origin network structure, the end feature shape is $64 \times 32 \times$C without downsampling in height if our input is 64×512×C. But we remove last downsampling process to keep size with 64×64×C to restore more details in feature maps.

TABLE I: The comparsion of different downsampling times based on SqueezeNet, without other new layers

	downsample times	Precision	Recall	IOU
Car	4 (*SqueezeNet* [3])	0.637	0.913	0.629
	3	0.611	0.9	0.6
Pedestrian	4 (*SqueezeNet* [3])	0.13	0.195	0.008
	3	0.246	0.233	0.181
Cyclist	4 (*SqueezeNet* [3])	0.204	0.547	0.189
	3	0.238	0.611	0.237

In Table I, the different downsample times show different performance in pedestrian and cyclist which is a small object in the projected images. We would find that with a small downsample time, the performance would be better and the network would learn robust feature representation to distinguish the small objects.

2) Evaluation without reweight layer: Because the end feature map size is 64×64 in height and weight. To make enlargement layer achieve better performance, we set a different rate of enlargement layer according to the feature map size. A suitable rate set will help the overlap of layer eyesight be more effective.

TABLE II: The performance of the network with different rate set and without reweight layer.

	rate	Precision	Recall	IOU
Car	3,5,8	0.716	0.942	0.675
Pedestrian	3,5,8	0.167	0.187	0.143
Cyclist	3,5,8	0.381	0.556	0.663
Car	4,8,12	0.711	0.901	0.663
Pedestrian	4,8,12	0.376	0.233	0.181
Cyclist	4,8,12	0.278	0.611	0.237
Car	6,9,12	0.705	0.934	0.618
Pedestrian	6,9,12	0.339	0.32	0.198
Cyclist	6,9,12	0.347	0.550	0.331

In Table II, it shows the performance of network which does not add reweight layer to improve the channel-wise feature. We only implement the enlargement layer in fire 9 and its memory cost increases from 1.6G to 1.8G. If we add another one in fire4, memory cost will improve from 1.6G to 2.2G with little performance change. The different rate sets have the same memory cost, the only difference is the eyesight overlap. Because of this, the network chooses the rate(6,9,12), which achieves the best result. At this stage, we notice that it is difficult for the network to distinguish pedestrian and cyclist based on enlargement layer only because of the distortion and uncommon input image shape.

TABLE III: The comparsion of runtime performance

Methods		Time(ms)
SqueezeSeg [21]	w/ CRF	13.5
	w/o CRF	8.5
PointSeg	w RANSAC	14
	w/o RANSAC	12
PointSeg in TX2	w/o RANSAC	98

3) Evaluation without enlargement layer: : From the above discussion, we conclude that the improvement of feature representation cannot help us get better performance, which is the reason why our network takes the reweight layer into consideration. We proposed three methods to combine the network with reweight layer. (i)derive features from the encoder to get the global descriptor. (ii)derive features both from the encoder and the decoder and concatenate them to get the global descriptor. (iii)derive features from the decoder to get the global descriptor.

We name the downsampling process as the encoder, the upsampling process as the decoder. The only difference between the three methods is the feature maps come from which layer. For example, if we consider features from decoder, the reweight layer will reweight channel-wise feature based on those features which have more spatial information. In Table IV, both (i)encoder and (ii)encoder and decoder show the better performance than the baseline of *SqueezeNet* [3] which is shown in Table I. According to the experiment, we find that to classify objects in the spherical image, the most key features come from the encoder which is the best time to reweight the layer weight. If we implement the reweight layer which global descriptor is from decoder, the results will decrease obviously as reweighting feature weight from deconvolution layer will add noise on feature location. We add the reweight layer only before every downsampling process. We also implement the reweight layer after each layer in the downsampling process. Even though the accuracy improves slightly, the real time performance would be influenced.

TABLE IV: The performance of network with different reweight layer

	feature from	Precision	Recall	IOU
Car	encoder+decoder	0.636	0.952	0.616
Pedestrian	encoder+decoder	0.293	0.232	0.149
Cyclist	encoder+decoder	0.261	0.458	0.2
Car	encoder	0.692	0.927	0.657
Pedestrian	encoder	0.347	0.288	0.187
Cyclist	encoder	0.337	0.565	0.267
Car	decoder	0.532	0.827	0.495
Pedestrian	decoder	0.128	0.118	0.136
Cyclist	decoder	0.143	0.389	0.24

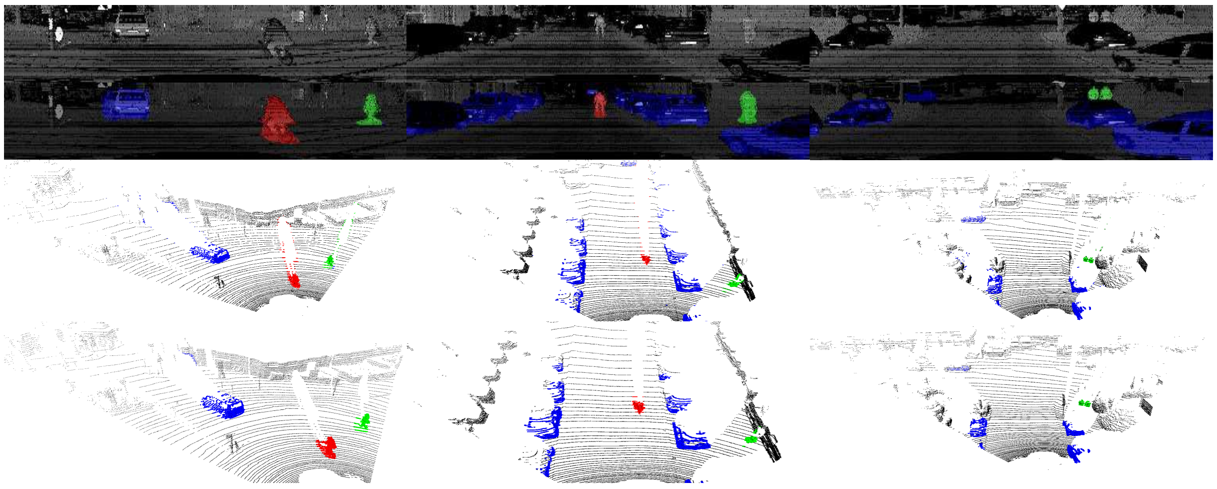

Fig. 7: Visualizations of raw inputs, *PointSeg* predictions and results after back projection with or without RANSAC refinements from up to down. The third row shows results which are projected back without RANSAC and the forth row shows resutls wich are projected back without RANSAC.

TABLE V: The performance comparison between *SqueezeSeg* and *PointSeg*

	method		Precision	Recall	IOU
Car	*SqueezeSeg* [21]	w/ CRF	66.7	95.4	64.6
	SqueezeSeg [21]	w/o CRF	62.7	**95.5**	60.9
	PointSeg		**74.8**	92.3	**67.4**
Pedestrian	*SqueezeSeg* [21]	w/ CRF	45.2	**29.7**	21.8
	SqueezeSeg [21]	w/o CRF	**52.9**	18.6	**22.8**
	PointSeg		41.4	29.3	19.2
Cyclist	*SqueezeSeg* [21]	w/ CRF	35.7	45.8	25.1
	SqueezeSeg [21]	w/o CRF	35.2	51.1	26.4
	PointSeg		**41.4**	**59.7**	**32.7**

We compare our results with *SqueezeSeg* [21], which is summarized in Table V. Our results for the pedestrian are similar to those for *SqueezeSeg* (without CRF) and show great improvement in performance for car and cyclist. We do not use any end-processes on the evaluation results which are shown in Tabel V, and our system can achieve 12 ms per frame in our workstation during the forward process with 2G memory cost. The comparison of runtime performance is shown in III. During the back projection from the mask on the spherical image to point cloud data, we use random sample consensus (RANSAC) to do the outlier remove. The operation can help us get a refined segmentation result as shown in Fig. 7, and only cost around 2 ms improvement in time. The evaluation result with RANSAC is shown in Table VI. The performance can still improve in the precision and IoU generally. We do not compare this with *SqueezeSeg* due to the randomness of RANSAC.

V. Conclusion

In this paper, we improved the feature-wise and channel-wise attention of the network to get the robust feature representation, which have shown a good improvement of performance in this task. We proposed the *PointSeg* system

TABLE VI: The performance of *PointSeg* with RANSAC adided

		Precision	Recall	IOU
Car	*PointSeg*+RANSAC	0.772	0.962	0.673
Pedestrian	*PointSeg*+RANSAC	0.486	0.294	0.239
Cyclist	*PointSeg*+RANSAC	0.463	0.633	0.387

which can be directly applied in autonomous driving systems and implemented in embedded AI computing device with limited memory cost, the interface time in TX2 was shown in Table III. It is an end-to-end method which does semantic segmentation in transformed 3D point cloud data.

Besides, we focus on the computation ability and memory cost during the implementation. Therefore, our approach can achieve a high accuracy at real-time speeds, and spare enough space and computation ability for other tasks in driving systems. In the projection data, it is quite difficult for the network to extract good features to distinguish the cyclist and pedestrian for the similar patterns, and this will influence the performance on both classes. There are also other light-version networks, such as *MobileNet* [26], *ShuffleNet* [27]. However, we do not use it because some operations in those network structure can not set different $stride$ in $height$ and $width$ in the process. Despite the limitations, the *PointSeg* still showed a good improvement in the balance of accuracy and real-time performance.

REFERENCES

[1] C. Feng, Y. Taguchi, and V. R. Kamat, "Fast plane extraction in organized point clouds using agglomerative hierarchical clustering," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6218–6225.

[2] M. Himmelsbach, A. Mueller, T. Lüttel, and H.-J. Wünsche, "Lidar-based 3d object perception," in *Proceedings of 1st international workshop on cognition for technical systems*, vol. 1, 2008.

[3] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[4] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," *CoRR*, vol. abs/1612.01105, 2016. [Online]. Available: http://arxiv.org/abs/1612.01105

[5] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[6] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *CoRR*, vol. abs/1706.05587, 2017. [Online]. Available: http://arxiv.org/abs/1706.05587

[7] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *CoRR*, vol. abs/1511.07122, 2015. [Online]. Available: http://arxiv.org/abs/1511.07122

[8] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," in *Advances in neural information processing systems*, 2011, pp. 109–117.

[9] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *CoRR*, vol. abs/1511.00561, 2015. [Online]. Available: http://arxiv.org/abs/1511.00561

[10] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "Icnet for real-time semantic segmentation on high-resolution images," *CoRR*, vol. abs/1704.08545, 2017. [Online]. Available: http://arxiv.org/abs/1704.08545

[11] G. Lin, A. Milan, C. Shen, and I. D. Reid, "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation," *CoRR*, vol. abs/1611.06612, 2016. [Online]. Available: http://arxiv.org/abs/1611.06612

[12] B. Li, "3d fully convolutional network for vehicle detection in point cloud," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 1513–1518.

[13] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," *CoRR*, vol. abs/1711.06396, 2017. [Online]. Available: http://arxiv.org/abs/1711.06396

[14] B. Graham, "Sparse 3d convolutional neural networks," *CoRR*, vol. abs/1505.02890, 2015. [Online]. Available: http://arxiv.org/abs/1505.02890

[15] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," *CoRR*, vol. abs/1611.07759, 2016. [Online]. Available: http://arxiv.org/abs/1611.07759

[16] R. Dubé, A. Cramariuc, D. Dugas, J. Nieto, R. Siegwart, and C. Cadena, "SegMap: 3d segment mapping using data-driven descriptors," in *Robotics: Science and Systems (RSS)*, 2018.

[17] R. Schnabel, R. Wahl, and R. Klein, "Efficient ransac for point-cloud shape detection," in *Computer graphics forum*, vol. 26, no. 2. Wiley Online Library, 2007, pp. 214–226.

[18] Z. Lin, J. Jin, and H. Talbot, "Unseeded region growing for 3d image segmentation," in *Selected papers from the Pan-Sydney workshop on Visualisation-Volume 2*. Australian Computer Society, Inc., 2000, pp. 31–37.

[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, vol. 1, no. 2, p. 4, 2017.

[20] R. Dubé, M. G. Gollub, H. Sommer, I. Gilitschenski, R. Siegwart, C. Cadena, and J. Nieto, "Incremental-segment-based localization in 3-d point clouds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1832–1839, 2018.

[21] B. Wu, A. Wan, X. Yue, and K. Keutzer, "Squeezeseg: Convolutional neural nets with recurrent CRF for real-time road-object segmentation from 3d lidar point cloud," *CoRR*, vol. abs/1710.07368, 2017. [Online]. Available: http://arxiv.org/abs/1710.07368

[22] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[24] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016. [Online]. Available: http://arxiv.org/abs/1606.00915

[25] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[27] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *CoRR*, vol. abs/1707.01083, 2017. [Online]. Available: http://arxiv.org/abs/1707.01083