3D Object Detection from Point Cloud and RGB Image

Semester Project

Author Kang Peilin

Supervisor Prof. Mathieu Salzmann

Computer Vision Laboratory – CVLAB
Faculté Informatique et Communications – IC
Ecole Polytechnique Fédérale de Lausanne – EPFL

Submitted on January 10, 2020
Lausanne, EPFL

Abstract

In this report we consider the problem of 3D object detection in robotics applications such as autonomous cars. PointRCNN is the state-of-art method to detect 3D objects by using point cloud directly. However, RGB image also provides detailed semantic information which will benefit the 3D object detection. In this project, we try to further improve the performance of PointRCNN by leveraging some image information properly. We tried several different methods to add image information to PointRCNN: (1) Adding RGB values to each corresponding LIDAR point in the input dataloader. We project the point cloud into the corresponding image and use bilinear interpolation to get the RGB value for each point. (2) Instead of using the bilinear interpolation to get RGB value, we consider a patch around the point and use the mean and covariance matrix of this patch. (3) Apart from adding simple image information, we also tried to add image features learned by PSPNet to PointRCNN. Evaluated on KITTI 3D object detection benchmark, we find that adding simple image information such as RGB value make the performance worse especially on hard level object detection. Adding more robust image information, mean and covariance, slightly improve the performance of easy level objects, but cause the hard level objects detection performance worse. And when we add finetuned image features learned by PSPNet and train the RCNN and PSPNet together, we are delighted to find that the performance of 3D object detection at all difficult levels improved.

Contents

Chapter 1

Introduction

Over the last decade, deep neural networks have made great progress on 2D computer vision applications, such as image classification, object detection and instance segmentation. However, in robotics applications such as autonomous drivings, we are interested in 3D object detection. These 2D object detection methods cannot be applied to 3D object detection directly since typical convolutional neural network requires regular input image format while raw LIDAR point clouds are sparse and unordered.

Most works on 3D object detection try to form a structured representation of point clouds and then use mature 2D detection architectures. There are mainly two types of methods to achieve this: one is projecting the point clouds to Front View and/or Bird's Eye View [4][2][9] and the other is partitioning the point clouds to the regular 3D voxels and then use 3D CNN[11] to detect the objects. However, since point clouds are sparse and have variable point density, we will suffer from information loss and bias after projection and discretization.

In order to fully leverage the spatial information of the point clouds, recently, a number of papers have proposed to learn point cloud features directly without converting them to other regular formats. [1][7] proposed a novel type of neural network named PointNet and PointNet++ that directly consumes point clouds, which well respects the permutation invariance of points in the input and is able to learn deep point set features efficiently and robustly. [6][8] are two methods based on PointNet/PointNet++ to detection 3D object.

[6] proposed a framework named Frustum PointNet which first uses the RGB image to generate the object proposals and then extrude to frustums according to depth information. Finally, based on 3D point clouds in those frustum regions, the network achieves 3D instance segmentation and amodal 3D bounding box estimation, using PointNet/PointNet++ networks. The performance of the Frustum PointNet highly relies on the performance of 2D CNN region proposal network and no 3D object will be detected given no 2D detection and the 3D information such as depth cannot be used during the region proposal stage.

[8] proposed a framework named PointRCNN which is a two-stage framework using only point clouds to detect 3D objects. In the first stage, PointRCNN generates accurate 3D object proposals from raw point clouds by using the PointNet to learn the point-wise features of point clouds. Then in the second stage, the proposed regions are further refined in the canonical

coordinate by the proposed bin-based 3D box regression loss. PointRCNN is evaluated on KITTI 3D object detection challenge and outperforms the state-of-art methods. However, PointRCNN uses only point clouds and doesn't leverage the corresponding RGB images which are dense and provide much more detailed semantic information and may further improve the average precision(AP) of the 3D object detection.

In this semester project, we aim to add RGB image information to the PointRCNN framework to further improve the Average Precision (AP) of 3D object detection. We evaluate our network on KITTI 3D object detection challenges which require detection of cars, pedestrians and cyclists. First, we reproduce the result of PointRCNN and use this as the baseline of the whole project. Then, we try three ways to add image information to this framework and compare with the baseline to see whether we get benefits by adding this image information. (1) First, we try to add RGB values to each point cloud in the input dataloader. We project the point cloud to the corresponding RGB image and use bilinear interpolation to get the RGB value for each point. (2) Then, instead of using the bilinear interpolation to get RGB value, we consider a patch around the point and use the mean and covariance matrix of this patch. (3) Apart from adding simple image information, we also try to add image features learned by CNN. After testing all these methods, we are delighted to find that when we add finetuned image features learned by PSPNet and train the RCNN and PSPNet together, the performance of 3D object detection at all difficult levels improved.

Chapter 2 of this report details the architecture and evaluation results of the PointRCNN. Chapter 3 describes the detailed implementations of adding image information to the PointRCNN. Chapter 4 presents evaluation results on KITTI 3D object detection benchmark. The final chapter is the conclusion.

Chapter 2

Baseline PointRCNN

In this chapter we look more detail into the structure of PointRCNN and show the reproduction results which we will use as the baseline.

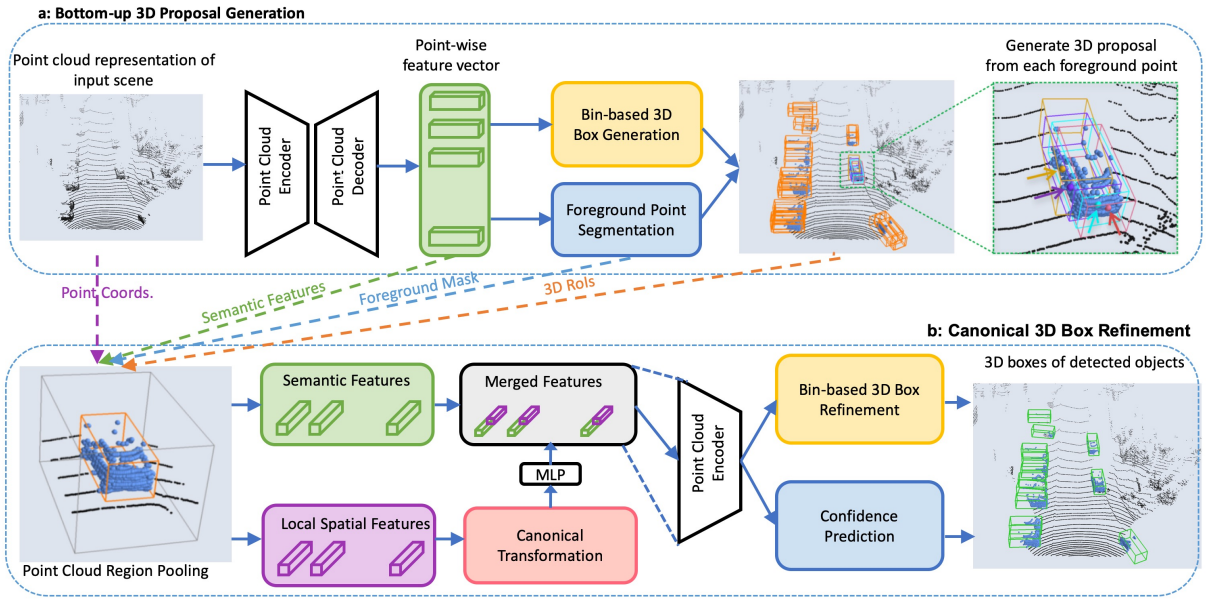


Figure 2.1: The PointRCNN architecture for 3D object detection from point cloud. The whole network consists of two parts: (a) for generating 3D proposals from raw point cloud in a bottom-up manner. (b) for refining the 3D proposals in canonical coordinate.

As shown in Figure 2.1, PointRCNN consists of two stages. The **first stage (RPN)** aims at generating high-recall 3D proposals. First, we feed the raw point clouds into a feature extraction network to get point-wise feature vector, here we use the PointNet++[7] as the feature extraction network. Based on the learned point-wise feature vector, we add two heads one for foreground point segmentation and the other for bin-based 3D box generation from the foreground points simultaneously. The generated 3D proposals will then flow into the second stage. The **second stage (RCNN)** aims at refining the box locations and orientations generated by the first stage. The features we feed into the second stage are: point cloud coordinates (x, y, z), its laser reflection intensity R, its foreground mask and C-dimentional feature vector learned from the first stage. Based on these features, we also add two heads one for bin-based

3D box refinement and the other for confidence prediction.

The training scheme of PointRCNN is two steps: (1) Train the first stage (we use **RPN** interchangeably in the later chapter) to get high-recall 3d boxes (2) Train the second stage (we use **RCNN** interchangeably in the later chapter) to refine the boxes. There are two ways to train the RCNN: (1) Train the RCNN offline: after training the RPN (first stage), we save the 3D proposals and their features for the RCNN. (2) Train the RCNN online: instead of saving the proposals and features, we fixed the RPN during the second stage and generate the proposals and features using the fixed RPN at the real time. Since the training samples of KITTI dataset are relatively small, so the author of PointRCNN also use the Ground Truth Augmentation (GT_AUG), which means put several new ground-truth boxes and their inside points from other scenes to the same locations of current training scene by randomly selecting non-overlapping boxes. So we can also train the PointRCNN with/without the GT_AUG.

We evaluate PointRCNN on 3D object detection benchmark of KITTI dataset and use the results as the baseline of our project. KITTI 3D object detection challenge requires to detect Cars, Pedestrians and Cyclists and define the difficult levels as bellow: (1) Easy: Min. bounding box height: 40 Px, Max. occlusion level: Fully visible, Max. truncation: 15%. (2) Moderate: Min. bounding box height: 25 Px, Max. occlusion level: Partly occluded, Max. truncation: 30%. (3) Hard: Min. bounding box height: 25 Px, Max. occlusion level: Difficult to see, Max. truncation: 50%. The PointRCNN detect one class at a time, here, we fucus on the detection of Cars.

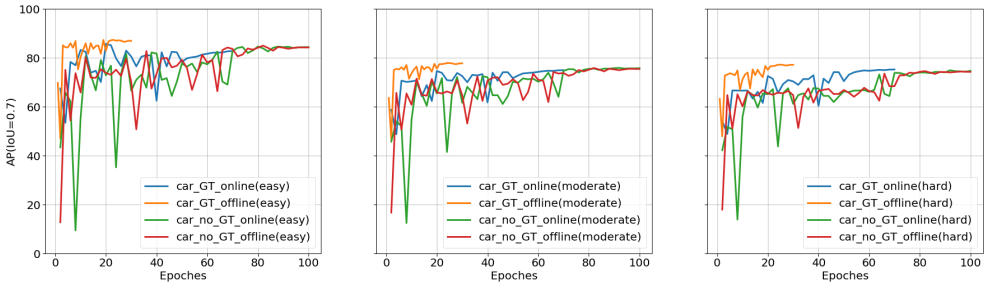


Figure 2.2: Performance of 3D object detection with PointRCNN on the car class of KITTI val split set.

		Car (IoU=0.7)		
		Easy	Moderate	Hard
With GT_AUG	RCNN online	82.77	74.93	75.23
	RCNN offline	86.90	77.73	77.13
Without GT_AUG	RCNN online	84.37	75.76	74.65
	RCNN offline	84.23	75.43	74.26

Table 2.1: Performance of 3D object detection with PointRCNN on the car class of KITTI val split set.

After comparing different ways of training the PointRCNN, we finally decided to use the training strategy RCNN online without GT_AUG as our baseline. Since this strategy is more elegant and convenient while the performance is also acceptable. Besides, what we want to do is to compare the performance before and after adding the image information.

Chapter 3

Implementation

We tried several ways to add image information to the PointRCNN framework to further improve the Average Precision (AP) of 3D object detection. First, we tried to add RGB values to each Lidar point in the input dataloader. We projected the point cloud to the corresponding image and used bilinear interpolation to get the RGB value for each point. Then, instead of using the bilinear interpolation to get RGB values, we considered a patch around the point and used the RGB mean and covariance matrix of the patch. Apart from adding these simple image information, we also tried to add image features learned by CNN.

3.1 Add RGB Data

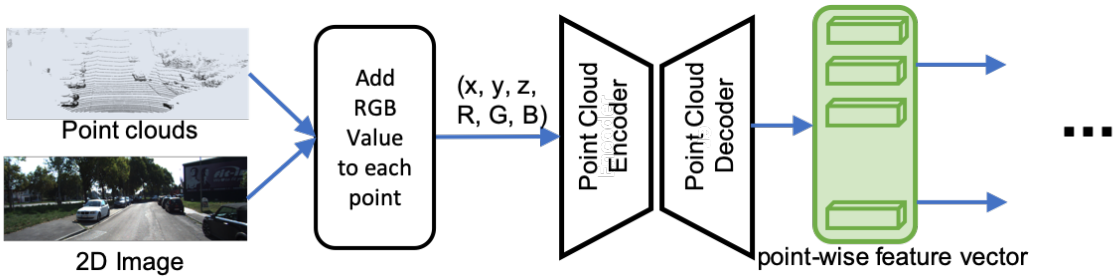
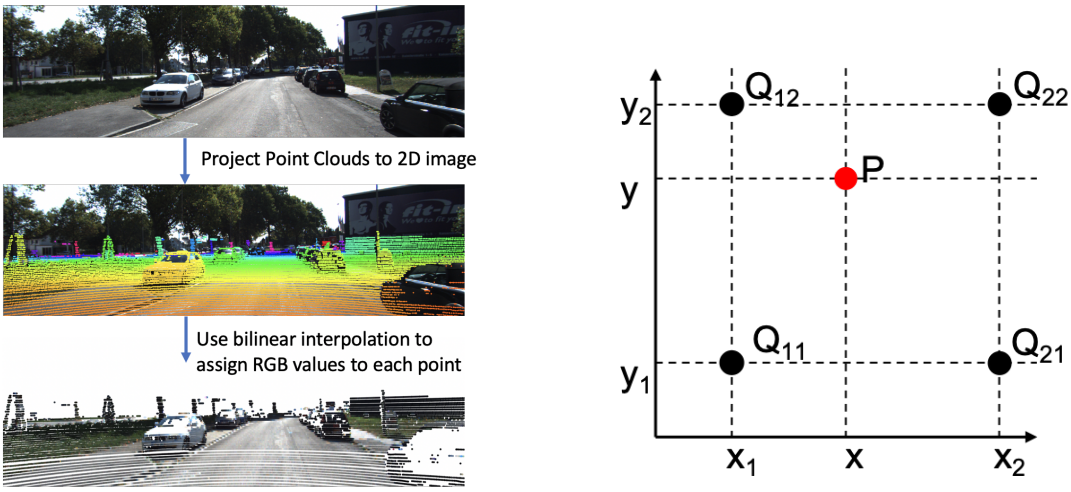


Figure 3.1: The partial model architecture: we add the RGB values before the first stage and leave the other parts of model unchanged

We add the RGB value to each Lidar point. The architecture is shown on figure 3.1. Now the input data we feed into the PointNet is the point cloud coordinates (x, y, z) plus the corresponding (R, G, B) values. The second stage RCNN is left unchanged because we assume that the RGB information has been encoded in the learned point-wise feature vector which will be used in the second stage RCNN.



(a) (b)

Figure 3.2: (a)Project point cloud to 2D image, (b) Bilinear interpolation

Figure 3.2 shows the process of getting the corresponding (R, G, B) value for each point. First we project the point cloud into the corresponding 2D image using the calibration files provided by KITTI. Since the projected point may not lie exactly on a pixel point of image, we use the bilinear interpolation to assign the RGB value to this point. As shown on Figure 3.2b, the red point is the projected LIDAR point and the four black points are the surrounding image pixels. we calculate the RGB value for P according to equation3.1.

$$\begin{aligned} P = &\frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} * [R,G,B]_{Q11} + \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} * [R,G,B]_{Q12} \\ &+ \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} * [R,G,B]_{Q21} + \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} * [R,G,B]_{Q22} \end{aligned} \tag{3.1}$$

3.2 Add Mean and Covariance

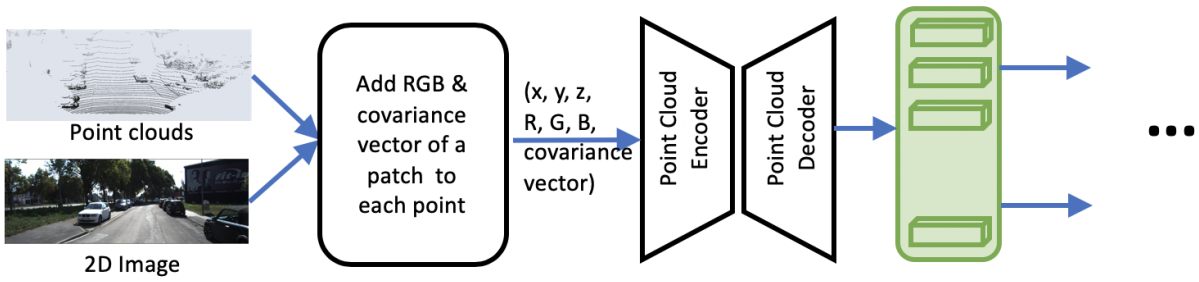


Figure 3.3: The partial model architecture: we add the mean and covariance vector of a patch to each Lidar point before the first stage and leave the other parts of the model unchanged

Instead of using the bilinear interpolation to get RGB values, we consider a patch around the projected point and use the mean and covariance vector of the patch for the reason that the

mean and covariance information of a patch are more robust to the number of points, sequence of image pixels and illumination. The architecture is shown on figure 3.3. Now the input data we feed into the feature extractor(PointNet++) is 12-dimension: 3d point cloud coordinates (x, y, z), 3d mean (r, g, b) values and 6d covariance vector (Cov(R,R), Cov(R,G), Cov(R,B), Cov(G,G), Cov(G,B), Cov(B,B)) of the patch. The second stage RCNN is left unchanged because we assume that the mean and covariance information has been encoded in the learned point-wise feature vectors which will be used in the second stage RCNN.

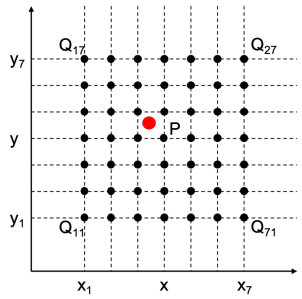


Figure 3.4: The black points are pixels on image and red point is the LIDAR point projected to the image. We take a 7x7 patch and calculate the mean and covariance vector of the patch

The patch size we choose is 7x7. Each patch has three channels R, G and B, each channel has 49 values (7*7). We calculate the mean of each channel and also calculate the covariance of three channels in pairs of two using equation 3.2 and get the covarance vector with 6 values.

$$Cov(X,Y) = \frac{1}{N}\sum_{i=0}^{N-1}(X_i - \mathbb{E}[X])(Yi - \mathbb{E}[Y]) \tag{3.2}$$

3.3 Add Image Features

We use the PSPNet[10] to extract the image features. PSPNet first use CNN to get the global context information and then a pyramid parsing module[5] is applied to harvest different sub-region representations to get local context information. Finally, the representation is concatenated and fed into a convolution layer to get the final pixel-wise prediction.

Since KITTI 3b object detection benchmark doesn't have the pixel-wise annotated image needed to train the PSPNet, and the KITTI semantic segmentation benchmark consists only 200 semantically annotated train as well as 200 test images, which is not enough to train the PSPNet from the scratch. In this way, we use the pretrained weights of PSPNet on CityScapes Dataset [3](The pretrained weight we use is according to `https://github.com/shahabty/PSPNet-Pytorch`). The pretrained model segments 19 classes, but here we only need to focus on Cars. So we finetune the last convolution layer of PSPNet use the KITTI semantic segmentation benchmark to segment cars from the background.

3.3.1 Add to the first stage: RPN

As shown on Figure 3.5, We add one more branch to extract point-wise image features. Then we concatenate the image features learned by finetuned PSPNet with point cloud features learned by PointNet++. Then the concatenated features is used both in the first stage and in the second stage. When training the PointRCNN model we will fix the PSPNet.

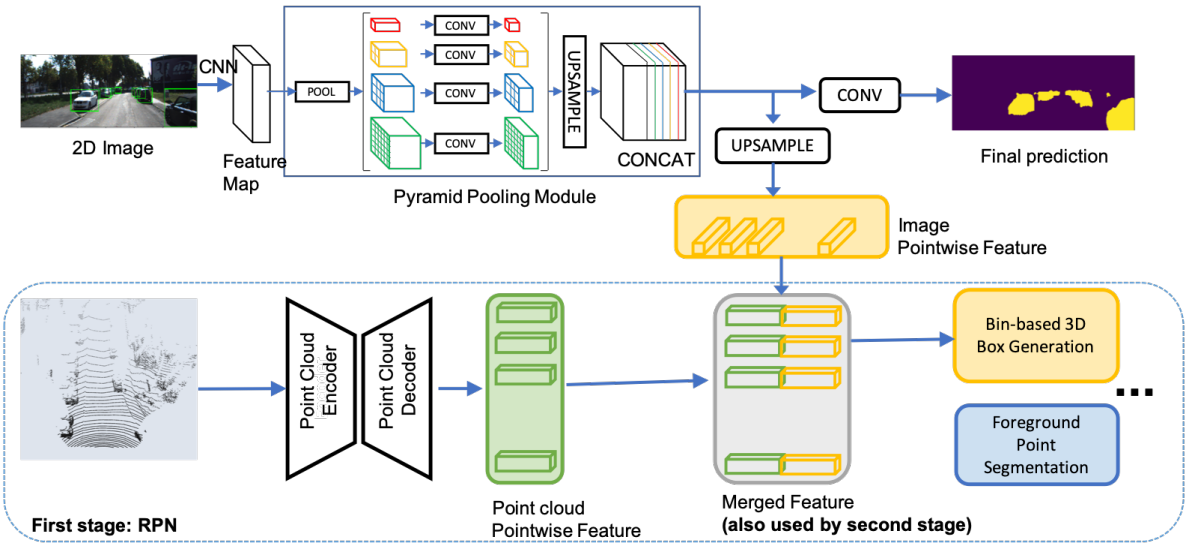


Figure 3.5: The partial model architecture: we concatenate the point-wise image features with point-wise point cloud features and then the concatenated feature is used to generate 3D proposal. The second stage remains unchanged except that the second stage uses the concatenated features instead of the point cloud point-wise features.

3.3.2 Add to the second stage: RCNN

As shown on Figure 3.6, we leave the first stage RPN unchanged and merge the image features with point cloud features before the first layer of the second RCNN stage and then we train the PSPNet with the RCNN together.

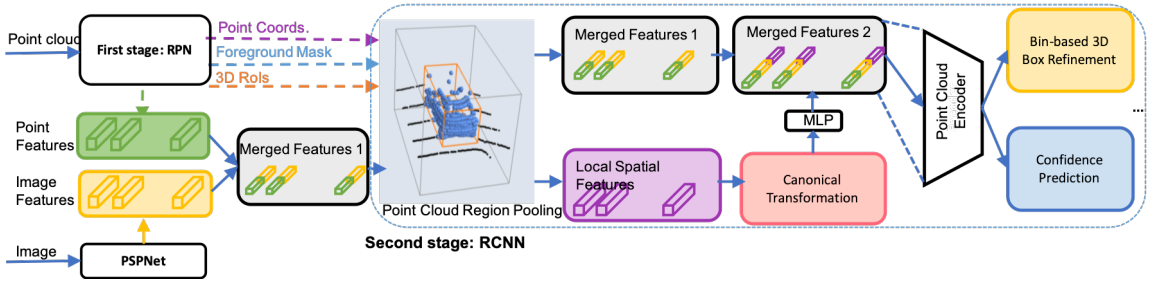


Figure 3.6: Model architecture: we remain the first stage RPN unchanged and concatenate the point-wise image features with point-wise point cloud features before feeding into the second stage RCNN

Chapter 4

Evaluation

4.1 Evaluate on Car

We evaluated those architectures on the KITTI object detection benchmark and the training strategy is the same as baseline: RCNN online without GT_AUG. We use the default hyper-parameters provided on the official PointRCNN paper[8]. The first stage is trained for 200 epochs while the second stage is trained for 100 epochs.

	Car (IoU=0.7)		
	Easy	Moderate	Hard
Baseline	84.37	75.76	74.65
Add RGB	83.93	74.88	68.49
Add mean and cov	85.17	75.37	68.78
Add img feature rpn	84.05	75.64	74.58
Add img feature rcnn	**86.82**	**76.99**	**76.33**

Table 4.1: Performance of 3D object detection after adding image information to point clouds on the car class of KITTI val split set.

Table 4.1 shows the Average Precision(AP) of 3D object detection after adding image information. We can observe that after adding RGB values, the performance for all the three difficult levels becomes worse, especially for the hard level. We think this may be caused by the fact that the RGB information we added is not so robust. For example, the RGB value will be influenced by the illumination: when in the shadow, the color of the car may becomes similar to the color of the shadow of the tree. Also, for the hard level, the point cloud is sparse and as a result we only add sparse RGB values to the hard level cars and may induce some noise instead.

When adding mean and covariance, the performance of the easy level increases a little bit compared to baseline. We infer that this is because the mean and covariance information are more robust and have the same property for cars which helps a little bit to detect cars at easy level. But we observe that the performance of the hard level is almost as worse as adding RGB information. We think this may caused by two reasons: (1) The LIDAR points in the hard level cars are very few and sparse, cause even the mean and covariance act like noise; (2) The patch

size we used is not suitable for the hard level cars. In the next step we can try to use difference patch size for different difficult levels.

When adding the image features at the first stage RPN and then used by both RPN and RCNN with fixed PSPNet, we find the performance of all three levels is almost the same as the baseline. When adding the image features at first layer of the second stage RCNN and then train the PSPNet together with the RCNN, we can observe that the performance of all the three difficult levels become better. This means by adding finetuned image features will help to detect objects. But if the image features are not accurate enough, they may become useless.

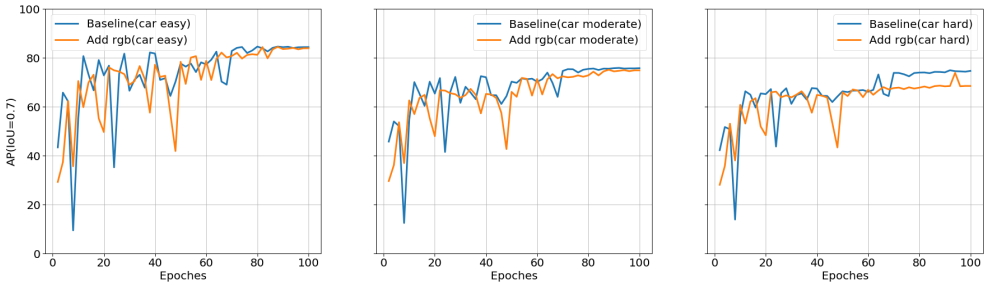


Figure 4.1: The Average precise(AP) of 3D object detection vs Epochs after adding RGB values to point clouds on the car class of KITTI val split set.

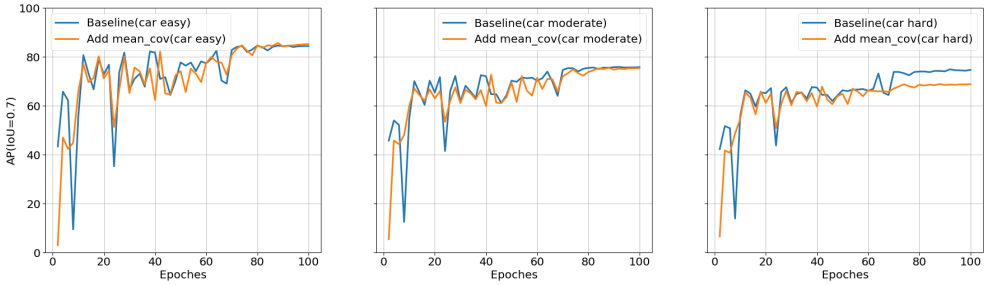


Figure 4.2: The Average precise(AP) of 3D object detection vs Epochs after adding mean and covariance vector to point clouds on the car class of KITTI val split set.

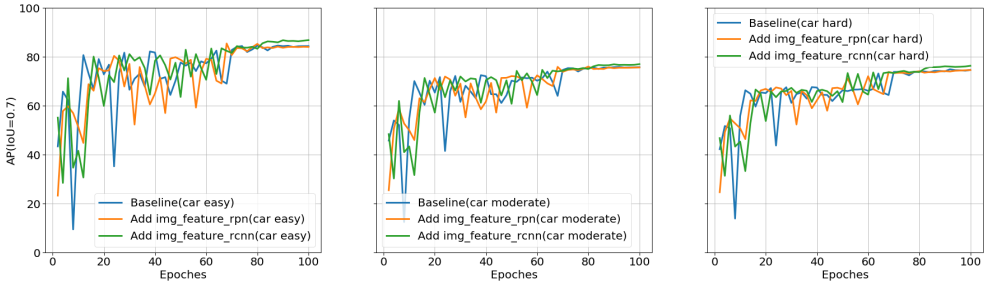


Figure 4.3: The Average precise(AP) of 3D object detection vs Epochs after adding image features on the car class of KITTI val split set.

Figure 4.1, 4.2, 4.3 show the Average Precise(AP) of 3D car detection for each epochs when using different ways to add the image information.

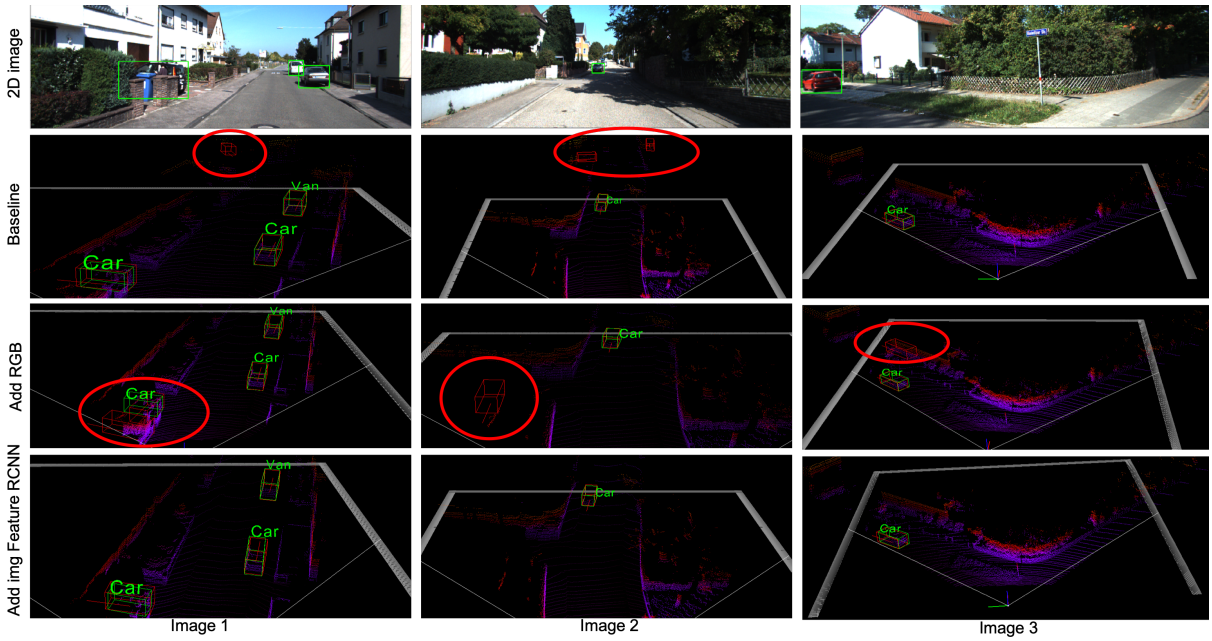


Figure 4.4: Visualizations of different methods' results on KITTI val set. 3D instance masks on point cloud are shown in color. Ground truth boxes are in green, while the predicted boxes are in red. And the red circles show the false positive boxes.

In Figure 4.4, we visualize representative outputs of our PointRCNN baseline model, add_RGB model and add_img_feature_RCNN model. For image 1, add_img_feature_RCNN model can predict the boxes correctly, while the baseline model has one false positive boxes and add_RGB model fails to predict the car occluded. For image 2, though all three models predict the boxes correctly, but both baseline model and add_RGB model has false positive boxes. For image 3, only add_RGB model has false positive box.

4.2 Evaluate on Pedestrian and Cyclist

Based on the evaluation results on Car class, we can conclude that adding image features learned by PSPNet to RCNN and joint train RCNN and PSPNet will improve the 3D Car detection Average Precision. In this section we want to evaluate this method on Pedestrian and Cyclist classes. First we clarify the meaning of the term *PSPNet not finetuned* and *PSPNet finetuned*. (1) **PSPNet not finetuned** means we use the pretrained weights of PSPNet on CityScapes Dataset [3] directly. (2) **PSPNet finetuned** means we finetune the last convolution layer of PSPNet use the KITTI semantic segmentation benchmark to segment pedestrian from the background. Since there is not cyclist class in the KITTI semantic segmentation benchmark, for cyclist we only use the PSPNet that is not finetuned.

	Pedestrian (IoU=0.5)		
	Easy	Moderate	Hard
PointRCNN(Baseline)	58.49	49.37	43.22
Add img feature rcnn(PSPNet not finetuned)	57.42	49.54	43.44
Add img feature rcnn(PSPNet finetuned)	**60.10**	**51.14**	**44.41**

Table 4.2: Performance of 3D Pedestrian detection by adding image features generated by PSPNet on KITTI val split set.

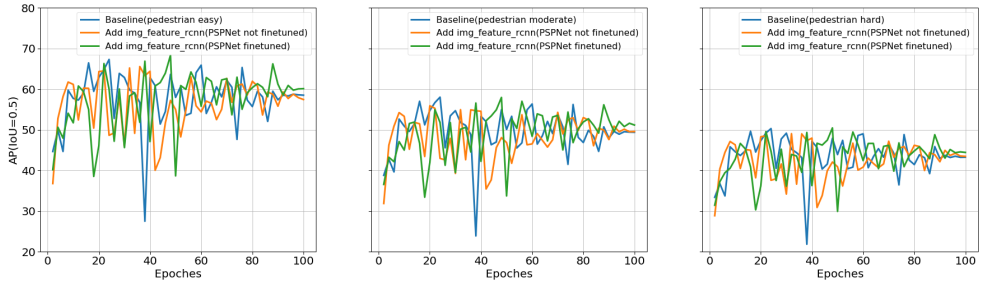


Figure 4.5: The Average precise(AP) of 3D Pedestrian detection vs Epochs after adding image features generated by PSPNet on KITTI val split set.

	Cyclist (IoU=0.5)		
	Easy	Moderate	Hard
PointRCNN(Baseline)	**77.77**	**59.60**	**55.83**
Add img feature rcnn(PSPNet not finetuned)	75.58	57.68	53.64

Table 4.3: Performance of 3D Cyclist detection by adding image features generated by PSPNet on KITTI val split set.

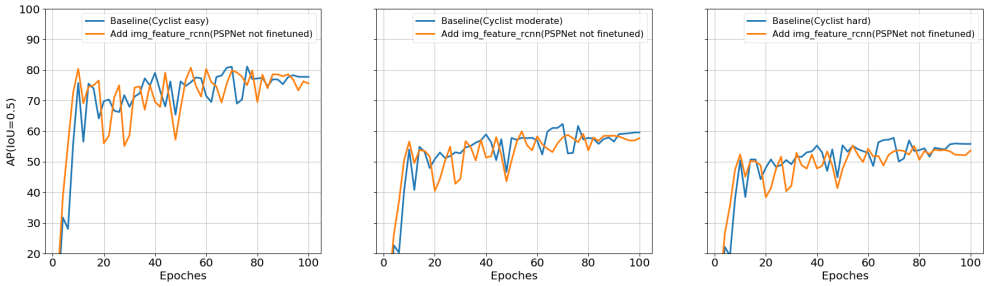


Figure 4.6: The Average precise(AP) of 3D Cyclist detection vs Epochs after adding image features generated by PSPNet on KITTI val split set.

Table 4.2 and Figure 4.5 show the performance of 3D Pedestrian detection. Table 4.3 and Figure 4.6 show the performance of 3D Cyclist detection. We can observe that when we add the image features generated by PSPNet that is not finetuned by KITTI semantic segmentation benchmark, we will get worse result compared to the baseline for both Pedestrian and Cyclist classes. When we use finetuned PSPNet, the performance of 3D pedestrian detection becomes better than the baseline. This means the image features we added to the RCNN should be well finetuned, otherwise, the image features will become useless or even harmful to the final performance.

Chapter 5

Conclusion

In this work we improved the performance of 3D object detection results of PointRCNN by leveraging the RGB image information. We tried several methods to leverage image information, such as adding RGB values, adding mean and covariance and merging the image features with point cloud features. The experiments show that adding simple image information such as RGB values cause the performance worse especially on hard level object detection. When Adding more robust image information, mean and covariance of a patch, slightly improve the performance of easy level objects compared to baseline, but the hard level objects detection performance is still worse as adding RGB values. And when we add finetuned image features learned by PSPNet and train the RCNN and PSPNet together, we are delighted to find that the performance of 3D object detection at all difficult levels improved.

Bibliography

[1] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 77–85. DOI: 10.1109/CVPR.2017.16.

[2] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. *Multi-View 3D Object Detection Network for Autonomous Driving*. 2016. arXiv: 1611.07759 [cs.CV].

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[4] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. *Joint 3D Proposal Generation and Object Detection from View Aggregation*. 2017. arXiv: 1712.02294 [cs.CV].

[5] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. *Feature Pyramid Networks for Object Detection*. 2016. arXiv: 1612.03144 [cs.CV].

[6] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. *Frustum PointNets for 3D Object Detection from RGB-D Data*. 2017. arXiv: 1711.08488 [cs.CV].

[7] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. 2017. arXiv: 1706.02413 [cs.CV].

[8] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. *PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud*. 2018. arXiv: 1812.04244 [cs.CV].

[9] Bin Yang, Wenjie Luo, and Raquel Urtasun. *PIXOR: Real-time 3D Object Detection from Point Clouds*. 2019. arXiv: 1902.06326 [cs.CV].

[10] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. "Pyramid Scene Parsing Network". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017). DOI: 10.1109/cvpr.2017.660. URL: http://dx.doi.org/10.1109/CVPR.2017.660.

[11] Yin Zhou and Oncel Tuzel. *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*. 2017. arXiv: 1711.06396 [cs.CV].