EasyChair Preprint

№ 4474

BVNet: A 3D End-to-End Model Based on Point Cloud

Nuo Cheng, Xiaohan Li, Shengguang Lei and Pu Li

October 26, 2020

BVNet: A 3D End-to-end Model Based on Point Cloud

Nuo Cheng[1,2], Xiaohan Li[2], Shengguang Lei[2], and Pu Li[1]

[1] Process Optimization Group, Technische Universität Ilmenau
98693 Ilmenau, Germany
nuocheng1992@gmail.com Pu.li@tu-ilmenau.de
[2] LiangDao GmbH, 12099 Berlin, Germany
{xiaohao.li, shengguang.lei}@liangdao.de

Abstract. Point cloud LiDAR data are increasingly used for detecting road situations for autonomous driving. The most important issues here are the detection accuracy and the processing time. In this study, we propose a new model which can improve the detection performance based on point cloud. A well-known difficulty in processing 3D point cloud is that the point data are unordered. To address this problem, we define 3D point cloud features in the grid cells of the bird's view according to the distribution of the points. In particular, we introduce the average and standard deviation of the heights as well as a distance-related density of the points as new features inside a cell. The resulting feature map is fed into a conventional neural network to obtain the outcomes, thus realizing an end-to-end real-time detection framework, called BVNet (**B**ird's-**V**iew-**N**et). The proposed model is tested on the KITTI benchmark suit and the results show considerable improvement for the detection accuracy compared with the models without the newly introduced features.

Keywords: Autonomous driving · point cloud · feature extraction · 3D object detection · CNN · KITTI.

1 Introduction

Object detection plays an important role in an advanced driver assistance system (ADAS). Compared with 2D images, 3D data have many advantages because of its depth information. However, with the rapid hardware development, the improvement of the performance of 3D object detection based on deep learning is not as fast as that of 2D object detection. For example, in the tests of the KITTI benchmark suite[3], the performance gap of the average precision between 2D and 3D models for car detection is higher than 10%. Therefore, improving the 3D detection precision remains a significant challenge.

The available 2D object detection methods can be classified into two approaches: the one-stage methods [13,18–20] and the two-stage methods [6–8,21]. A two-stage method first trains a region proposal network (RPN) to generate

[3] *http://www.cvlibs.net/datasets/kitti/eval_3dobject.php*

several interested regions, and then performs the classification and regression of the bounding boxes. On the other hand, the one-stage detection method predicts bounding boxes and object categories in one step. The latest versions of the software using these two approaches both show extremely high performance [8, 20].

With the fast development and wide application of LiDAR, 3D detection-based point cloud data becomes more and more important. Point cloud gives 3D information which can be used for 3D object detection. However, unlike the 2D arrangement of pixels in an image, the data points of a 3D point cloud are unordered. For this reason, point cloud cannot be processed directly by a typical convolutional architecture. Currently, the existing 3D detection methods could be classified into three categories to solve this problem.

First, the unordered point cloud is processed using a symmetric function. Qi et al. were the first to propose a detection method based on point cloud called PointNet [16, 17]. In [15] they applied it for detecting objects for autonomous driving and tested on the KITTI benchmark suite. However, this method needs a pre-processing using a 2D image and a 2D detection model. Therefore, the precision of this method depends on the performance of the 2D detection approach. In addition, it is difficult to achieve a real-time detection, since successively running the 2D and 3D detection models will take a large amount of computation time.

Using PointNet [16, 17] as the backbone network, PointRCNN [22] has recently shown higher performance on the KITTI Benchmark Suite. It is the first two-stage 3D object detection approach with raw point cloud as input. However, in the first detection stage, a bounding box is needed for each foreground point of the detected object, which requires high computation expense. In addition, PointNet [16, 17] does not show high-speed performance on large-scale point cloud [15, 17]. Furthermore, PointRCNN is not suitable to detect large objects such as trucks and special vehicles.

Second, unordered point cloud is projected to a 2D image before processing. Chen et al. [3] proposed another highly ranked model (MV3D) by projecting LiDAR point cloud to a bird's eye view map. In this way, unordered 3D points are transferred into a 2D image which is further processed for object detection. After that, more and more detection models [1–4, 10, 11, 24, 25, 28] were proposed to extract features on the bird's eye view map to improve the quality of the object detection. Table 1 shows the features extracted in the grid cells by these detection models. Some models [1, 2, 4, 11, 24, 25, 28] extract features based only on the bird's eye view map and others combine it together with image [3, 10]. The features mostly extracted are the height, intensity and density of the point cloud for each grid cell. Although 2D approaches could achieve a high frame-rate, the average precision is low [1, 2, 4, 11, 24, 25, 28].

Third, unordered point cloud is transformed to voxels. The basic idea is to divide the point cloud into many small voxels and uses a 3D-CNN to extract features. VoxelNet [29] was developed based on this idea. In comparison to the models shown in Table 1, VoxelNet achieves a high average precision but cannot be used for real-time detection (only 4 fps on TitanX GPU [29]).

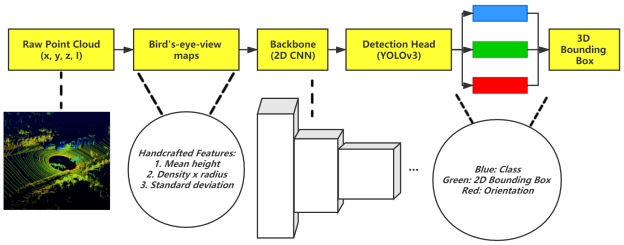


Fig. 1. BVNet Pipeline. We present a sample detection model. The input is raw point cloud from Velodyne HDL-64 LiDAR. The output is 3D bounding box with classes. We connect three new handcrafted features with modified State-of-the-Art 2D detector.

From the hardware implementation point of view, the first and the third approaches discussed above cannot be readily applied in real circumstances due to the fact that they usually run in GPUs [9, 14, 15, 17, 22, 23, 26, 29]. High performance GPUs consume a large amount of power and thus are difficult to run stably for vehicle-mounted detection systems. Existing dedicated hardware or embedded devices (e.g. Huawei Atlas 200 and 300 etc.) support 2D-detectors better than 3D-detectors[4]. In such cases, the second approach is suitable.

In addition, detection of multiple objects for autonomous driving is required. Compared with 3D detectors, 2D detectors are more appropriate for addressing multi-classification problems [18–20]. 3D detectors such as MV3D [3] and PointRCNN [22] can only detect a single category at a time. Therefore, it is favorable to use the second approach, i.e. to project the point cloud to a 2D image and combine it with an efficient 2D detection model, and finally convert the 2D results back into the 3D detection results. However, the shortcoming of this approach is that the height information is lost. Therefore, proper measures have to be taken to compensate the height information in the case of autonomous driving.

In this study, we propose a simple 3D detection model, BVNet (**B**ird's-**V**iew-**N**et), which is only based on raw point cloud, to achieve real-time detection for multi-categories for autonomous driving. The proposed BVNet pipeline is shown in Fig.1, being a detection method based on the bird's eye view map. Unlike the models listed in Table 1, we introduce the average and standard deviation of the heights as new features inside each cell. In addition, we define a distance-related density feature of the points. The backbone detection network in BVNet is an extended CNN based on the Complex-YOLO [24, 25] and YOLOv3 [20]. The proposed BVNet is evaluated on the KITTI benchmark suite[3]. Compared with the models in Table 1, better results in terms of accuracy are achieved by using our model. In addition, in comparison to models with 3D approaches like

[4] Huawei atlas support: https://support.huawei.com/enterprise/en

PointNet [16,17] and VoxelNet [29], we can achieve similarly accurate results for detecting cars with considerably lower computation time.

Table 1. Features extracted by different detection models

Models	Features			Number of the height feature maps	Camera
MV3D [3]	Max Intensity	Density	Max height	M	+
Complex-YOLO [25]	Max Intensity	Density	Max height	1	-
AVOD [10]	-	Density	Max height	5	+
YOLO3D [1,4]	-	Density	Max height	1	-
RT3D [28]	Mean height	Min Height	Max Height	3	-
BirdNet [2]	Mean Intensity	Density	Max height	1	-

2 Proposed Model

2.1 Introduction of new features

We consider the point cloud data from Velodyne HDL64 Lidar [5]. At first, we represent the point cloud by projecting it to a bird's eye view map. The detection range is selected as $60m \times 60m \times 3.25m$. Inspired by Complex-YOLO [24,25], we discretize and project the raw point cloud into a 2D grid cell with the resolution of 8 cm. Therefore, the final size of the grid map is 768×768.

Since the features of the cells used for detection are important, we introduce following three new features to improve the performance for detection.

First, as mentioned in the last section, the models listed in Table 1 extract the maximum height information of the point cloud inside grid cells as a feature. However, the measurement of the maximum height by Lidar may include noises. In order to reduce the influence of the noises, we use the average height of each grid cell, as follows:

$$f_h = \frac{h_{mean}}{3.25} \tag{1}$$

where h_{mean} is the average of the height values at the points reflecting the object under consideration,and the 3.25 is the maximum height of the points.

Second, it is a well-known fact that the raw point cloud will become sparser as the distance increases. It means that the detection accuracy will be lowered if the cloud data are taken from a large distance. To address this problem, we introduce a distance-related density of the point cloud as a new feature. To this end, we multiply the grid density by the Euclidean distance between the grid cell and the origin point (i.e. the LiDAR sensor position), as shown in Fig.2. For this purpose, we modify the normalized density formula $min(\frac{log(N+1)}{log64})$ used in MV3D [3], as follows:

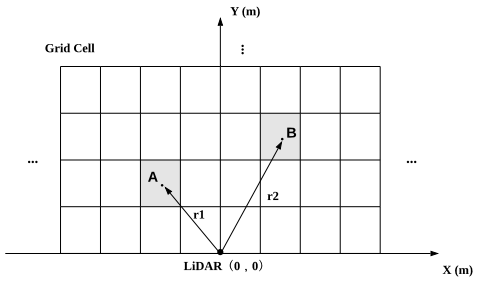


Fig. 2. For the two grid cells (A and B) in this bird's eye view map, we multiply the number of point clouds in each grid by r1 and r2 to balance the point clouds sparse region.

$$r = \sqrt{(x_{grid} - x_{origin})^2 + (y_{grid} - y_{origin})^2} \tag{2}$$

$$\vartheta = N \times r \tag{3}$$

$$f_d = min(1.0, \frac{log(\vartheta + 1) - a}{b}) \tag{4}$$

where N is the number of points in the grid cell, x_{grid}, y_{grid} denote the coordinates of the grid cell and (x_{origin}, y_{origin}) is defined as (0, 0). Fig.3 shows the distribution of the values of $log(\vartheta+1)$, based on which, it can be found that most of the values are between 3.0 to 9.0. Thus, in the normalized density formula (4), we take $a = 3, b = 6$. Attention should be paid that for different LiDAR data, different values for a and b should be chosen.

Third, some detection models [2, 3, 24, 25] extract the intensity as a feature inside a grid cell. However, based on the BirdNet's experimental results [2], it is found that only using this feature, the intensity of point cloud is extremely unstable. Thus, using the intensity as a feature can improve a little or nothing of the detection performance. Therefore, we propose to use the standard deviation of the height of the point cloud as a new feature to replace the intensity. This is because the main disadvantage of bird's-eye-view-based detection models is the loss of the height information. However, it is difficult to reflect the distribution of point clouds on the Z-axis only by the mean point cloud height of each grid cell. However, the standard deviation can supplement the distribution of the point clouds in the cell, which is computed as follows

$$S_n = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(h_i - h_{mean})^2} \tag{5}$$

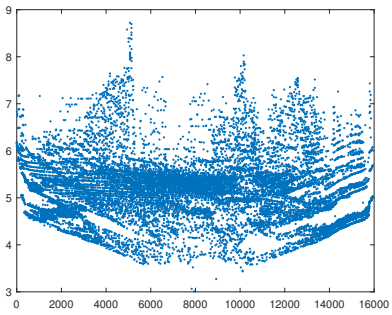


Fig. 3. The distribution of the distance-related density of the point cloud (The horizontal ordinate is the number of point cloud and the vertical ordinate is the values of $log(\vartheta + 1)$).

where h_i is the height values of the points. However, the standard deviation in most grids may be too small (e.g. 0-0.1). To make the distribution of the standard deviation smooth between the cells, we normalize the standard deviation by

$$f_{std} = \sqrt{1 - (\frac{S_n}{max(S_n)} - 1)^2} \tag{6}$$

Finally, the three newly introduced features f_d, f_h, f_{std} are used as inputs to the CNN, as shown in Fig.1.

2.2 Network Architecture

Our BVNet model uses a modified Darknet53 [20] architecture as the backbone, which can predict not only the dimension and classes of the bounding boxes, but also the orientation. Fig.4 shows the complete network structure. Because the BVNet consists of feature pyramid networks [12], it extracts features from three different scales of a feature map. The down-sampling steps of the three scales are 8, 16 and 32, respectively. In addition, as YOLOv3 [20], BVNet also has 5 residual modules, which extends the network to 53 layers

Orientation Prediction. In the past two years, many methods [1, 4, 24, 25] were proposed to modify YOLO [18–20] to enable it to predict orientation. YOLO3D [1] and YOLO4D [4]can predict the offset of orientation and compute the mean squared error between the ground truth and the predicted orientation directly in the loss function expressed as $\sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} (\theta_i - \theta_i^{'})^2$, where θ_i is the predicted orientation and $\theta_i^{'}$ is the ground truth orientation of the bounding box. However, since using one value could create singularities, in Complex-YOLO

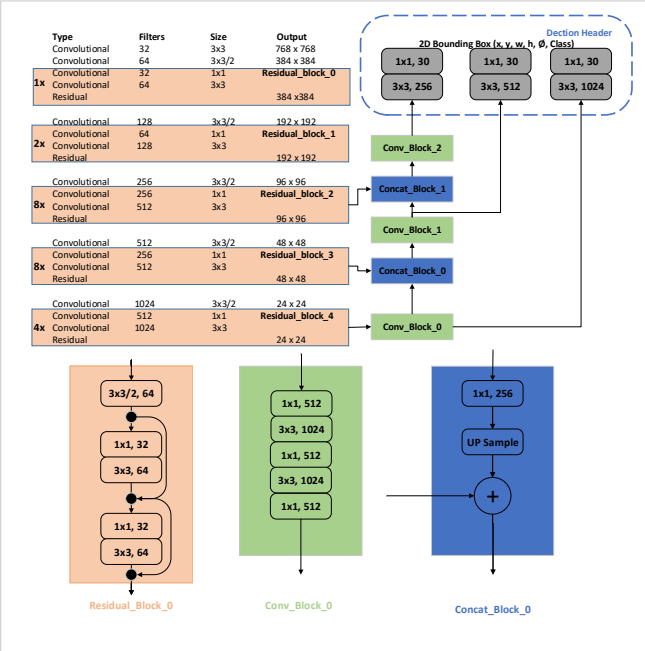


Fig. 4. The detection backbone is the same as YOLOv3's and we add an orientation regression at the end, so that the network can predict the orientation of the object. The bottom shows the residual block, the convolutional block and the feature pyramid network of the network, respectively.

[24,25], the Grid-RPN approach in YOLOv2 [19] was modified by adding two responsible regression parameters (t_{im}, t_{re}): $b_\theta = arg(|z|\, e^{ib_\theta}) = arctan_2(t_{im}, t_{re})$. In BVNet, We employ this modified method so that it can predict the orientation of bounding boxes.

Loss Function. The loss function $\mathcal{L}$ of our network is based on YOLOv3 [20] and Complex-YOLO [25] and the ground true boxes are defined by (x, y, w, h, θ, class):

$$\mathcal{L}_{BVNet} = \mathcal{L}_{YOLOv3} + \mathcal{L}_{Complex-YOLO} \tag{7}$$

$$\begin{aligned}\mathcal{L}_{BVNet} = \lambda_{coord} \sum_{i=0}^{N\times N} \sum_{j=0}^{K} 1_{ij}^{obj} [(t_x - t_x^{'})^2 + (t_y - t_y^{'})^2] \\ +\lambda_{coord} \sum_{i=0}^{N\times N} \sum_{j=0}^{K} 1_{ij}^{obj} [(t_w - t_w^{'})^2 + (t_h - t_h^{'})^2] \\ +\lambda_{coord} \sum_{i=0}^{N\times N} \sum_{j=0}^{K} 1_{ij}^{obj} [(t_{re} - t_{re}^{'})^2 + (t_{im} - t_{im}^{'})^2]\end{aligned} \tag{8}$$

$$
\begin{aligned}
&-\sum_{i=0}^{N\times N}\sum_{j=0}^{K} 1_{ij}^{obj}[c_i^{'}log(c_i)+(1-c_i^{'})log(1-c_i)] \\
&-\lambda_{noobj}\sum_{i=0}^{N\times N}\sum_{j=0}^{K} 1_{ij}^{noobj}[c_i^{'}log(c_i)+(1-c_i^{'})log(1-c_i)] \\
&-\sum_{i=0}^{N\times N} 1_{ij}^{obj}\sum_{c\in classes}[P_i^{'}(c)log(P_i(c))+(1-P_i^{'}(c))log(1-P_i(c))]
\end{aligned}
$$

where λ_{coord} is the weight of the coordinate loss. λ is used to control the imbalance of the predicted bounding boxes with and without objects inside. $t_x, t_y, t_w, t_h, t_{re}, t_{im}$ are the predicted values of the 2D bounding box. $t_x^{'}, t_y^{'}, t_w^{'}, t_h^{'}, t_{re}^{'}, t_{im}^{'}$ are the truth values of the 2D bounding box. c is the confidence of the prediction bounding box. P_i is the probability of the object class. Cross entropy is used here to calculate the loss. 1_{ij}^{obj} takes the value of 0 and 1 based on whether there is a ground truth box in the i-th and j-th location (i.e.,1 if there is a ground truth box, and 0 otherwise). In contrast, 1_{ij}^{noobj} takes the value of 0 if there is no object, and 1 otherwise.

Anchor Box Calculation. As reported above, YOLOv3 [20] uses a feature pyramid network [12] to detect objects on three different scale feature maps. For each feature map, an object detector predicts three anchors on every grid cell. The size of the anchors is calculated in different datasets by the k-means clustering. Since the position of Lidar is fixed, the size of the object in the bird's view map does not change with distance, we use the predefined length and width from cars, cyclists and pedestrians to determine the sizes of the 3 anchors in each grid cell, with width, and length of $(1.6, 3.9)$, $(0.6, 1.76)$ and $(0.6, 0.8)$ meters [11], respectively.

Detection Header Network. We predict 3 bounding boxes in each grid cell on one feature map. The dimension of the tensor is $[3 \times (6 + 1 + N_{class})]$, where 6 stands for the number of bounding box offsets, 1 stand for the confidence prediction, and N_{class} is the number of the classes. In our model, we choose three KITTI-classes (i.e., car, pedestrian and cyclist) to perform the detection.

3D Bounding Box Regression. A 2D bounding box is obtained through our improved detection model. Quit a few methods [24,25] transfer 2D bounding boxes to 3D by a predefined height for each class. However, this method cannot accurately generate the 3D bounding boxes. In order to improve the accuracy, we take the maximum average height from the 2D bounding box as the height of the 3D bounding box.

3 Training and Experiments

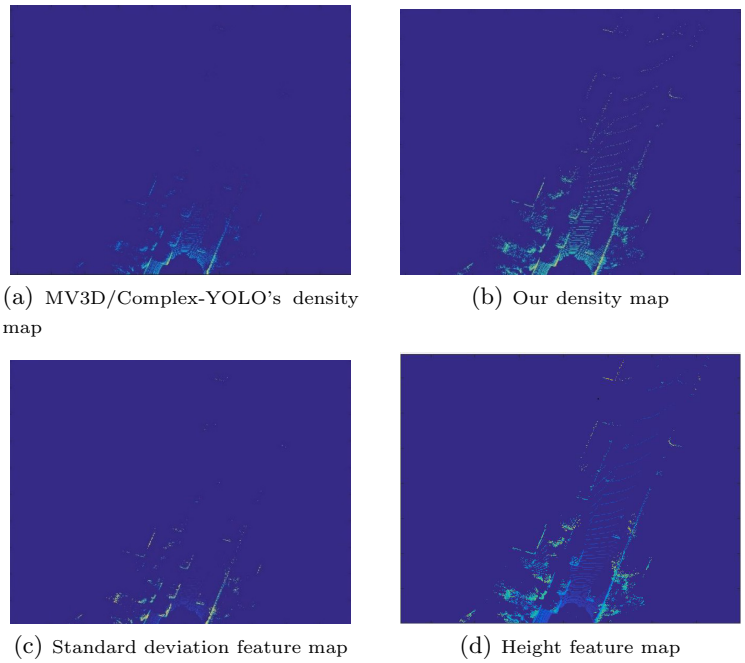
3.1 KITTI Dataset

The KITTI dataset [5] (with 7,481 samples for the training and 7,518 samples for testing) is used to test our model. We follow the frequently used method [14]

[17] to divide the KITTI training dataset into train split (3712 samples) and validation split (3769 samples).

3.2 Optimization of feature maps

As stated in the second section, we at first extract the features on the bird's eye view map. Fig.5. (a) and (b) show the density map from our model and that from Complex-YOLO [25]/ MV3D's [3]. It can be seen that since we use a distance-related density of point cloud instead of the normally used density, the improvement is obvious.



(a) MV3D/Complex-YOLO's density map

(b) Our density map

(c) Standard deviation feature map

(d) Height feature map

Fig. 5. Comparison of different feature maps

In addition, Fig.5 (c) and (d) show the mean height and the standard deviation feature map. It is shown that, compared with the height and the density feature map, the deviation feature map provides the contour of the object better and remove some useless points. Finally, we encode the three features as RGB (**R**ed-**G**reen-**B**lue) and fed them into the CNN.

3.3 Training and Implementation

Our model is trained by the stochastic gradient descent method with a weight decay of 0.0005 and a momentum of 0.9. Since the backbone network is based

on YOLOv3 [20], the training parameters are taken as the same as YOLOv3's. We trained the network for 500 epochs, with a batch size of 4.

At the first few epochs, we used a small learning rate to ensure the convergence (0.001), and then we scaled the learning rate up in the middle epochs(0.005). In order to avoid gradient divergence, we slowly reduce the learning rate at the end of the training(0.005 to 0.001). Batch normalization for regularization is performed and Leaky ReLU is used as the activation function. To remove overlapped bounding boxes, we use the non-maximal suppression (NMS) on the bird's eye view maps with a threshold of 0.4.

The training environment used is Ubuntu 16.04 with an NVIDIA 2080ti GPU and an i7-9700K 3.6GHz CPU. During the training, we randomly crop and flip the bird's eye view maps for data augmentation. The total training time is around 50 hours.

3.4 Result Evaluation

We evaluate our model by following the KITTI evaluation protocol, where the IoU thresholds for the car class is 0.7 and for others are 0.5. Results are given on the validation set. The average precision (AP) is used to compare the performances of different models.

Bird's eye view map detection performance on validation set using different features as input. We use our detection network to test all extracted features on the bird's view map. We trained each network for 150 epochs.

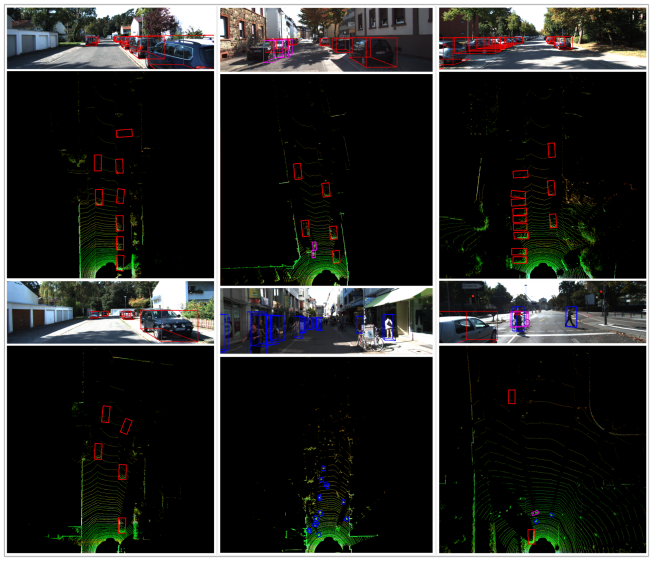


Fig. 6. Visualization of BVNet results. Examples of detection results of BVNet on KITTI validation set. (red: car, blue: pedestrian, pink: Cyclist)

The combinations of features are listed in Table 1. All the methods are tested on our validation dataset. Our method is compared with Complex-YOLO [25], YOLO3D [1, 4] and BirdNet [2], which are based only on bird's-eye-view map feature extraction. It can be seen that our model shows the best performance. (see Table 2 and Fig.6).

Table 2. Results of testing different features on the validation dataset.

Features			Car	Pedestrian	Cyclist
Max Intensity	Max Height	Density	78.18	42.21	61.81
Mean Intensity	Max Height	Density	78.77	**42.57**	61.21
-	Max Height	Density	76.21	41.78	60.00
Max height	Mean Height	Min Height	72.94	38.92	56.89
Our Features			**82.78**	42.50	**63.71**

Evaluation of the bird's view map with other state-of-the-art models. All other results are taken from the KITTI benchmark suite[3]. Following the KITTI setting, we also divide the results into three difficulty regimes: easy, moderate and hard. For the car class, our model uses a 2D detector and shows similar performance compared with the models using a 3D detector. On the other hand, BVNet has an advantage in terms of efficiency (see Table 3).

Table 3. Results of bird's view detection

Method	Modality	speed	Car			Pedestrian			Cyclist		
			Easy	Moderate	hard	Easy	Moderate	hard	Easy	Moderate	hard
MV3D [3]	Lid.+Img.	360ms	86.62	78.93	69.80	-	-	-	-	-	-
VoxelNet [29]	LiDAR	230ms	89.35	76.26	77.39	46.13	40.74	38.11	66.70	54.76	50.55
F-PointNet [15]	LiDAR	170ms	91.17	84.67	74.77	**57.13**	**49.57**	**45.48**	77.26	61.37	53.78
PointRCNN [22]	LiDAR	100ms	**92.13**	87.36	**82.72**	54.77	46.13	42.84	82.56	67.24	60.28
F-ConvNet [26]	LiDAR	470ms	91.51	85.84	76.11	57.04	48.96	44.33	**84.16**	**68.88**	60.05
PointRGCN [27]	LiDAR	260ms	91.63	**87.49**	80.73	-	-	-	-	-	-
Ours	LiDAR	**50ms**	89.45	84.65	78.52	48.97	42.53	38.12	73.87	64.68	**60.98**

3.5 Failure Cases

Although our model results in an improved detection performance, it shows some failure cases. The most error cases are road signs and trees mistakenly detected as pedestrians. The reason is due to the fact that the three handcrafted features for these classes are too similar. In practical applications, this issue can be addressed by using target tracking after the detection so as to remove such false bounding boxes.

4 Conclusion

In this paper, we propose BVNet, an end-to-end model based on point cloud only. We compensate the loss of height information on the bird's eye view map with newly introduced features to improve the detection accuracy. In addition, since our backbone network is YOLOv3 [20] which is able to detect objects of multiple classes, our model can also be easily extended for detecting new classes. Moreover, since BVNet only uses point cloud coordinate information to extract features, it works not only on Velodyne HDL-64 LiDAR, our model can also be adapted to various LiDAR (e.g. Innovusion LiDAR or Ibeo LiDAR etc.) with different parameters. Furthermore, since we use the classical 2D network as the backbone, our model can be readily applied to real autopilot.

References

1. Ali, W., Abdelkarim, S., Zidan, M., Zahran, M., El Sallab, A.: Yolo3d: End-to-end real-time 3d oriented object bounding box detection from lidar point cloud. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 0–0 (2018)
2. Beltran, J., Guindel, C., Moreno, F.M., Cruzado, D., Garcia, F., De La Escalera, A.: Birdnet: a 3d object detection framework from lidar information. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC). pp. 3517–3523. IEEE (2018)
3. Chen, X., Kundu, K., Zhang, Z., Ma, H., Fidler, S., Urtasun, R.: Monocular 3d object detection for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2147–2156 (2016)
4. El Sallab, A., Sobh, I., Zidan, M., Zahran, M., Abdelkarim, S.: Yolo4d: A spatio-temporal approach for real-time multi-object detection and classification from lidar point clouds (2018)
5. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition. pp. 3354–3361. IEEE (2012)
6. Girshick, R.: Fast r-cnn. In: Proceedings of the IEEE international conference on computer vision. pp. 1440–1448 (2015)
7. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2014)
8. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: Proceedings of the IEEE international conference on computer vision. pp. 2961–2969 (2017)
9. Jiang, M., Wu, Y., Zhao, T., Zhao, Z., Lu, C.: Pointsift: A sift-like network module for 3d point cloud semantic segmentation. arXiv preprint arXiv:1807.00652 (2018)
10. Ku, J., Mozifian, M., Lee, J., Harakeh, A., Waslander, S.L.: Joint 3d proposal generation and object detection from view aggregation. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1–8. IEEE (2018)
11. Lang, A.H., Vora, S., Caesar, H., Zhou, L., Yang, J., Beijbom, O.: Pointpillars: Fast encoders for object detection from point clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 12697–12705 (2019)

12. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2117–2125 (2017)
13. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: European conference on computer vision. pp. 21–37. Springer (2016)
14. Qi, C.R., Litany, O., He, K., Guibas, L.J.: Deep hough voting for 3d object detection in point clouds. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 9277–9286 (2019)
15. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum pointnets for 3d object detection from rgb-d data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 918–927 (2018)
16. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 652–660 (2017)
17. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: Advances in neural information processing systems. pp. 5099–5108 (2017)
18. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2016)
19. Redmon, J., Farhadi, A.: Yolo9000: Better, faster, stronger. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (July 2017)
20. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
21. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. pp. 91–99 (2015)
22. Shi, S., Wang, X., Li, H.: Pointrcnn: 3d object proposal generation and detection from point cloud. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
23. Shi, S., Wang, Z., Wang, X., Li, H.: Part-aˆ 2 net: 3d part-aware and aggregation neural network for object detection from point cloud. arXiv preprint arXiv:1907.03670 (2019)
24. Simon, M., Amende, K., Kraus, A., Honer, J., Samann, T., Kaulbersch, H., Milz, S., Michael Gross, H.: Complexer-yolo: Real-time 3d object detection and tracking on semantic point clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 0–0 (2019)
25. Simony, M., Milzy, S., Amendey, K., Gross, H.M.: Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 0–0 (2018)
26. Wang, Z., Jia, K.: Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection. arXiv preprint arXiv:1903.01864 (2019)
27. Zarzar, J., Giancola, S., Ghanem, B.: Pointrgcn: Graph convolution networks for 3d vehicles detection refinement. arXiv preprint arXiv:1911.12236 (2019)
28. Zeng, Y., Hu, Y., Liu, S., Ye, J., Han, Y., Li, X., Sun, N.: Rt3d: Real-time 3-d vehicle detection in lidar point cloud for autonomous driving. IEEE Robotics and Automation Letters **3**(4), 3434–3440 (2018)
29. Zhou, Y., Tuzel, O.: Voxelnet: End-to-end learning for point cloud based 3d object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4490–4499 (2018)