A Lightweight One-Stage 3D Object Detector Based on LiDAR and Camera Sensors

Li-Hua Wen and Kang-Hyun Jo
Intelligent Systems Laboratory, Department of Electrical, Electronic
and Computer Engineering, University of Ulsan, Ulsan 44610, Korea
wenlihua@islab.ulsan.ac.kr, acejo@ulsan.ac.kr

***Abstract*—Currently, many kinds of LiDAR-camera-based 3D object detectors have been developed with two individual heavy branches, one is used to extract features from LiDAR data, and the other one is utilized to extract features from camera data. In contrast, this paper proposes an early-fusion method to exploit both LiDAR and camera data for fast 3D object detection using only one backbone, achieving a good balance between accuracy and efficiency. Massive experiments evaluated on the KITTI benchmark suite show that the proposed approach outperforms state-of-the-art LiDAR-camera-based methods on the three classes. Additionally, the proposed model runs at 23 frames per second (FPS), which is almost 2× faster than state-of-the-art fusion methods for LiDAR and camera.**

***Index Terms*—LiDAR-camera-based 3D detector, single stage, one backbone, point-wise fusion, KITTI benchmark**

I. Introduction

With the rapid development of autonomous vehicles, three-dimensional (3D) object detection has become more important, whose purpose is to perceive the size and accurate location of objects in the real world. Currently, an intelligent car is equipped with at least one LiDAR apparatus, one radar and one RGB camera. Note that radar is now widely used in companies, however, only a few researchers use it to validate a new algorithm. Hence, this paper focuses on LiDAR and camera for 3D object detection. LiDAR is employed to collect the surrounding 3D data, referred to as a point cloud, and the camera is used to capture a high-resolution RGB image. The two devices provide two important and different types of data. However, it is non-trivial to highly efficiently and quickly extract and fuse the features of the point cloud and RGB image.

Recently, feature extraction with deep learning has drawn much attention. For the RGB image, a general 2D convolutional neural network (CNN) can be used to extract its features. For the point cloud however, it is difficult to extract its features due to its irregular distribution and sparse contributions. Before the advent of highly-efficient graphics processing units (GPUs), representative studies [1]–[7] have converted point clouds into 2D dense images or structured voxel-grid representations and utilized 2D neural networks to extract the corresponding feature from the converted 2D image. With the development of computer technology, the authors in [8]–[11] directly utilized a multi-layer perceptron (MLP) to aggregate features from point clouds. Shi et al. [12] encoded the point cloud natively in a graph using the points as the graph vertices.

To leverage the mutual advantages of point clouds and the RGB image, some researchers have attempted to fuse view-specific region of interest (ROI) features. Currently, there are two mainstream fusion methods. The first is to fuse two view-specific features. The other method is pointwise feature fusion. Chen et al. [1] and Ku et al. [2] directly fuse the ROI feature maps output with the two backbones of the point cloud and RGB image, respectively. On the other hand, Xu et al. [13] and Sindagi et al. [14] fuse pointwise features. These methods achieve better performance compared with LiDAR-based methods; however, their inference time is usually intolerable for application in real-time autonomous driving systems.

To deal with the above issues, this paper proposes a novel point-wise fusion strategy between point clouds and RGB images. The proposed method directly extracts pointwise features from the raw RGB image based on the raw point cloud first. Then, it fuses the two pointwise features and feeds them into a 3D neural network. The structure, as shown in Figure 1, has only one backbone to extract features, making the proposed model much faster than state-of-the-art LiDAR and camera fusion methods.

The key contributions of this work are as follows:

- This paper presents an early-fusion method to exploit both LiDAR and camera data for fast multi-class 3D object detection with only one backbone, achieving a good balance between accuracy and efficiency.
- This paper proposes a highly-efficient pointwise feature fusion module, which directly extracts the RGB image point feature based on a point cloud and fuses the extracted RGB image point feature with the corresponding feature of the point cloud.

The presented one-stage 3D multi-class object detection framework outperforms state-of-the-art LiDAR-camera-based methods on the KITTI benchmark [15] both in terms of the speed and accuracy.

II. Proposed Approach

The proposed model, as shown in Figure 1, takes point clouds and RGB images as inputs and predicts oriented 3D bounding boxes for cyclists, pedestrians, and cars. This model includes four main parts: (1) A point feature fusion module that extracts the point features from the RGB image and fuses the extracted features with the corresponding point cloud features, (2) a voxel feature encoder (VFE) module and a 3D

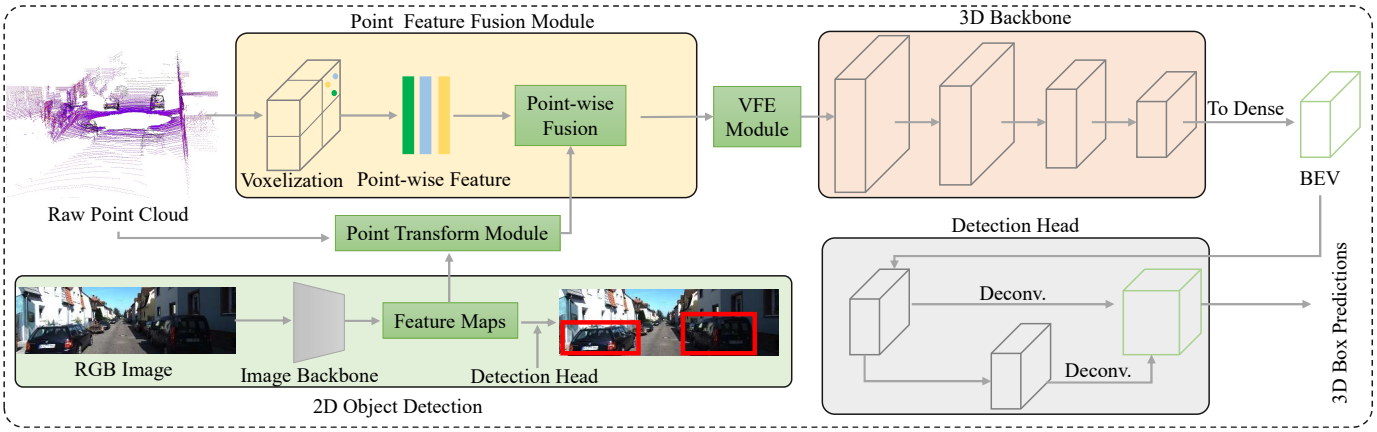

Figure 1: The architecture of the proposed one-stage 3D object detection network for the LiDAR and camera. It mainly includes the input data, the point feature fusion module, the 3D backbone, the 2-D object detection and the detection head. The gray box and green box represent the convolutional block and feature map, respectively.

backbone to process the fused pointwise features into a high-level representation, (3) a detection head that regresses and classifies the 3D bounding boxes, and (4) a loss function.

A. Point Feature Fusion Module

The fusion module, shown in Figure 2, consists of three submodules: the point transform module, the voxelization of point clouds, and the pointwise fusion module. Since this module involves the input of raw data, before introducing the module, the input data is first introduced.

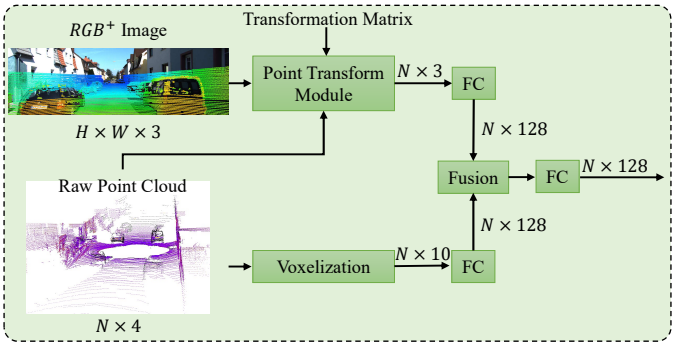

Figure 2: Visualization of the point feature fusion module. N is the number of points in a point cloud, and FC denotes one fully connected layer.

Input Data. This model accepts point clouds and RGB images as the input. To reduce the loss of raw point-cloud information during voxelization, a LiDAR point cloud is projected onto an RGB image and embedded into the image to generate a new image with three channels, called RGB$^+$. The RGB$^+$ object has two typical representations: the RGB$^{\mathbf{I}}$ portion that embeds the intensity of point clouds into an RGB image, and the RGB$^{\mathbf{D}}$ representation that embeds the Z-axis value of point clouds into the image [6], [7].

Point Transform Module. This module extracts point features from the RGB$^+$ image $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$ based on the raw point cloud. First, a point cloud $\mathbf{P} \in \mathbb{R}^{N \times 3}$ is projected onto its corresponding image by Eq. 1 to obtain the corresponding image coordinates (u_i, v_i). Second, the RGB$^+$ and the (u_i, v_i) are fed into the image sampler [16], outputting the image point feature $\mathbf{P_i} \in \mathbb{R}^{N \times 3}$, where N is the number of points in the point cloud.

$$\begin{pmatrix} u & v & 1 \end{pmatrix}^T = \mathbf{M} \cdot \begin{pmatrix} X & Y & Z & 1 \end{pmatrix}^T, \tag{1}$$

Voxelization. Voxelization divides the point cloud into evenly spaced voxel grids and then generates a many-to-one mapping between 3D points and their corresponding voxels. Currently, there exist two voxelization methods: hard voxelization [3], [17], [18] and dynamic voxelization [19]. Compared with the former, dynamic voxelization makes the detection more stable by preserving all the raw points and voxel information. This work applies the dynamic voxelization method. Given a point cloud $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N\}$, the process assigns N points to a buffer of size $N \times F$, where N is the number of points and F denotes the feature dimension. Specifically, each point $p_i = [x_i, y_i, z_i, r_i]$ (containing the XYZ coordinates and the reflectance value) in a voxel is denoted by its inherent information (x_i, y_i, z_i, r_i), its relative offsets (x_v, y_v, z_v) with respect to the centroid of the points in the voxel, and its relative offsets (x_p, y_p, z_p) with respect to the centroid of the points in the pillar. Finally, the output point-wise feature is $\mathbf{P_v} \in \mathbb{R}^{N \times 10}$, and the resulting size of the 3D voxel grid is $\left(\frac{W}{s_y}, \frac{H}{s_x}, \frac{D}{s_z}\right)$, where (s_y, s_x, s_z) gives the voxel sizes, and (W, H, D) are the ranges along the Y-axis, X-axis, Z-axis, respectively.

Point-wise Fusion. This module fuses the pointwise features $\mathbf{P_i}$ and $\mathbf{P_v}$. Since the dimensions of the two features are different, two fully connected (FC), one for each feature, are used to adjust their dimensions to be the same. There are two common fusion methods for ROIs: addition and concatenation. Therefore, this paper will analyze which fusion method is the most suitable for the pointwise features in Table II in the

ablation section. After the fusion operation, one FC layer is utilized to further merge the fused features and output the result as $\mathbf{P_f}$.

B. Voxel Feature Encoder Module and 3D Backbone

This section introduces the voxel feature encoder module and the 3D backbone, in that order.

Voxel Feature Encoder Module. Upon completing the point-wise fusion, the fused feature $\mathbf{P_f}$ is transformed through the VFE layer which is composed of a fully connected network (FCN), into a feature space, where information from the point features $\mathbf{f_i} \in \mathbb{R}^m$ can be aggregated to encode the shape of the surface contained within the voxel [3], [17], [18], where $i \in [1, N]$ and m is the feature dimension of a point. The FCN consists of a linear layer followed by a batch normalization layer, and a ReLU layer. An elementwise max-pooling process is used to locally aggregate the transformed features and output a feature $\vec{\mathbf{f}}$ for $\mathbf{P_f}$. Finally, the max-pooled feature $\vec{\mathbf{f}}$ is concatenated with each point feature $\mathbf{f_i}$ to generate the final feature $\mathbf{P_{vfe}}$. his work stacks two such VFE layers and both of the output lengths are 128. This means the shape of $\mathbf{P_{vfe}}$ is $N \times 128$.

3D Backbone. The 3D backbone takes the feature $\mathbf{P_{vfe}}$ and it's corresponding index of 3D coordinates $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ as inputs. The backbone is widely used in [20], [21] and has twelve 3D sparse convolutional layers and is divided into four stages according to feature resolution. The four-stage feature resolutions in the order of (W, H, D) are $(1600, 1408, 41)$, $(800, 704, 21)$, $(400, 352, 11)$, and $(200, 176, 2)$. Specifically, each stage has two kinds of 3D convolutional layers: the submanifold convolution [17] and the sparse convolution. The former does not generate new points and shares the point coordinate indices in each stage; hence, the submanifold convolution runs very fast. The latter is a sparse version of the dense 3D convolution. Usually, these two convolutions are used in conjunction to achieve the speed/accuracy balance. The sparse feature map after the 3D sparse convolution needs to be converted into the dense feature map $\mathbf{F_d} \in \mathbb{R}^{200\times176\times256}$.

C. Detection Head

The input data of the detection head is the dense feature map $\mathbf{F_d}$. The detection head is comprised of three convolution blocks. Block 1 has five 2D convolutional layers and outputs the feature map $\mathbf{F_1} \in \mathbb{R}^{100\times88\times128}$. Similarly, block 2 also has five 2D convolutional layers and takes the feature map $\mathbf{F_1}$ as input and outputs the feature map $\mathbf{F_2} \in \mathbb{R}^{50\times44\times256}$. Block 3 has two transpose layers and one 2D convolutional layer. $\mathbf{F_1}$ and $\mathbf{F_2}$ are transposed as the feature map $\mathbf{F_3} \in \mathbb{R}^{100\times88\times256}$ and the feature map $\mathbf{F_4} \in \mathbb{R}^{100\times88\times256}$, respectively. Finally, the feature maps $\mathbf{F_3}$ and $\mathbf{F_4}$ are concatenated as the feature map $\mathbf{F} \in \mathbb{R}^{100\times88\times512}$. The feature map $\mathbf{F}$ is mapped to three desired learning targets: (1) a classification score map $\mathbf{F_{score}} \in \mathbb{R}^{100\times88\times18}$, (2) a box regression map $\mathbf{F_{box}} \in \mathbb{R}^{100\times88\times42}$, and (3) a direction regression map $\mathbf{F_{dir}} \in \mathbb{R}^{100\times88\times12}$.

D. Loss Function

This work utilizes the same loss functions in PointPillars [18] and SECOND [17]. The 3D ground truth boxes and anchors are parameterized as $(x, y, z, l, w, h, \theta)$, where (x, y, z) denote the box's center, (l, w, h) represent the box's size, and θ is the yaw rotation around the Z-axis. The corresponding regression residuals between the 3D anchors and ground truth are defined as follows:

$$\begin{aligned} \Delta x &= \frac{x^g - x^a}{d^a}, & \Delta y &= \frac{y^g - y^a}{d^a}, \\ \Delta z &= \frac{z^g - z^a}{h^a}, & \Delta l &= log\left(\frac{l(g)}{l(a)}\right), \\ \Delta w &= log\left(\frac{w(g)}{w(a)}\right), & \Delta h &= log(\frac{h^g}{h^a}), \\ \Delta\theta &= sin(\theta^g - \theta^a), \end{aligned} \tag{2}$$

where the superscripts g and a represent the ground truth box and the anchor, respectively. The variable $d^a = \sqrt{(w^a)^2 + (l^a)^2}$ is the diagonal of the base of the anchor box.

The regression loss function is as follows:

$$\mathcal{L}_{reg} = \sum_b Smooth_{\mathcal{L}1}(\Delta b), \tag{3}$$

where the input dimensions are $b \in (x, y, z, w, l, h, \theta)$ and $Smooth_{\mathcal{L}1}$ is the smooth $\mathcal{L}1$ loss function in the Fast R-CNN module.

Since the yaw angle $\theta \in [-\Pi, \Pi]$ has two directions $\{+, -\}$, and the angle regression loss cannot distinguish the directions. A softmax classification loss is utilized to compute the discretized direction loss [17], $\mathcal{L}_{dir}$. If the yaw angle θ around the Z-axis of the ground truth is greater than zero, the direction is positive; otherwise, the direction is negative.

For the object classification loss, the focal loss [22] is used:

$$\mathcal{L}_{cls} = -\alpha_a(1 - p^a)^\gamma log(p^a), \tag{4}$$

where p^a is the class probability of an anchor, $\alpha = 0.25$, and $\gamma = 2$. The total loss can be formulated as follows:

$$\mathcal{L}oss = \frac{1}{N_{pos}}(\beta_1\mathcal{L}_{box} + \beta_2\mathcal{L}_{cls} + \beta_3\mathcal{L}_{dir}), \tag{5}$$

where N_{pos} is the number of positive anchors and $\beta_1 = 2.0$, $\beta_2 = 1.0$, and $\beta_3 = 0.2$. For the car class, an anchor is defined as positive if it has a 2D IoU greater than 0.60 (pedestrian/cyclist is 0.35) with its paired ground truth. If it has a 2D IoU less than 0.45 (pedestrian/cyclist is 0.2), the anchor is labeled as negative. The other anchors are ignored when computing the loss.

III. Experiments

A. Dataset

The proposed model is trained and evaluated on the KITTI dataset [15]. The KITTI object dataset possesses 7,518 testing frames and 7,481 training frames. Each frame is comprised of a point cloud, stereo RGB images (the left image and the right image), and calibration data. In this research, only a point cloud and the left image with their calibration data

Table I: Performance comparison using the KITTI testing dataset. The results of cars are evaluated by the mean Average Precision with 40 recall positions. The top performance is highlighted in bold only for the mAP columns and FPS column, and the second-best is shown in blue.

Method	FPS	AP_{BEV} (IoU = 0.7)				AP_{3D} (IoU = 0.7)				$AP_{2D}(IoU = 0.7)$			
		Easy	Moderate	Hard	mAP	Easy	Moderate	Hard	mAP	Easy	Moderate	Hard	mAP
MV3D [1]	2.8	86.00	76.90	68.50	77.13	71.10	62.40	55.10	62.87	96.47	90.83	78.63	88.64
F-PointNet [23]	5.9	88.70	84.00	75.30	82.67	81.20	70.40	62.20	71.27	95.85	95.17	85.42	92.15
AVOD [2]	12.5	86.80	85.40	77.70	83.30	73.60	65.80	58.40	65.93	95.17	89.88	82.83	89.29
AVOD-FPN [2]	10.0	88.50	83.80	77.90	83.40	81.90	71.90	66.40	73.40	94.70	88.92	84.13	89.25
ContFusion [24]	16.7	94.07	85.35	75.88	85.10	83.68	68.78	61.67	71.38	-	-	-	-
MVX-Net [14]	6.7	89.20	85.90	78.10	84.40	83.20	72.70	65.20	73.70	-	-	-	-
PFF3D [7]	18.5	89.61	85.08	80.42	85.04	81.11	72.93	67.24	73.76	95.37	92.15	87.54	91.69
Proposed	**23**	90.05	86.14	80.91	**85.70**	83.96	73.83	68.19	**75.33**	95.97	93.30	88.75	**92.67**

Table II: Effect of the point feature fusion module. The results are from the 'Moderate' difficulty category. The best result is highlighted in bold for each column.

Method				Cars (%)				Pedestrians (%)				Cyclists (%)			
Addition	Concatenation	FC	PA [25]	2D	AOS	BEV	3D	2D	AOS	BEV	3D	2D	AOS	BEV	3D
				89.27	88.72	85.04	77.00	70.69	33.91	62.74	56.20	64.13	59.55	59.19	55.71
√				89.74	89.39	85.84	76.60	70.26	56.75	62.80	57.65	64.64	58.40	61.34	59.72
	√			89.54	85.84	86.14	77.01	69.99	56.59	62.71	57.74	65.19	62.08	61.74	60.51
√		√		89.72	89.29	86.97	77.60	71.87	60.20	64.22	60.16	67.21	63.95	63.50	60.07
	√	√		89.70	89.27	86.27	77.12	69.51	58.02	65.89	59.74	64.74	61.50	60.87	59.37
√			√	89.76	89.23	86.05	76.32	72.23	53.15	67.89	60.18	65.18	60.50	60.50	60.60

are used. To impartially compare the proposed approach with existing methods, the training dataset is divided into two subsets (training subset and validation subset) based on the same criteria, and the ratio of the two subsets is 1:1.

B. Experimental Settings

The proposed model is an end-to-end 3D detector for three classes: the car, pedestrian, and cyclist. When designing the anchors for the three classes, different classes employ different sizes (w, l, h). The sizes $(1.6, 3.9, 1.56)$, $(0.6, 0.8, 1.73)$, and $(0.6, 1.76, 1.73)$ are for the car, the pedestrian, and the cyclist, respectively. Note that each anchor has two directions $\{0°, 90°\}$, which means that each location has six anchors. The detection area in the point cloud is $\{(x, y, z) \mid x \in [0, 70.4], y \in [-40, 40], z \in [-3, 1]\}$.

The framework is based on Pytorch and programmed by the python language. This model is trained from scratch based on Adam optimizer. The whole network is trained with a batch of size 10 and the initial learning rate is 0.003 for 80 epochs on one TITAN RTX GPU. This work also adopts the cosine annealing learning rate for the learning rate decay. The entire training time is around 12 hours.

For data augmentation, this work employs the widely used augmentations found in [7], [19], [20], including global scaling $[0.95, 1.05]$, global rotation around the Z-axis $[-45°, 45°]$, and the random flipping along the X-axis.

C. Results

The proposed model is also evaluated using the more challenging dataset: the KITTI testing dataset. In Table I, this part only compares the proposed method with state-of-the-art methods in three aspects: BEV, 3D, and 2D. Since it requires a great deal of data to compare these three performances, here, the results are simply compared based on the mean average precision (mAP). For the 3D performance, the proposed model has the best performance. For the BEV and 2D performances, the proposed method is the second-best, but the overall performance of the proposed method outperforms state-of-the-art methods when taking accuracy and speed into account. The results of the proposed method can be retrieved on the KITTI website based on the name of the proposed method, PFF3D.

IV. Ablation Studies

This section analyzes the proposed methods individually by conducting ablation experiments using the KITTI validation dataset.

1) Effect of the Point Feature Fusion Module: This section analyzes the point feature fusion module based on the three classes in detail. In Table II, the 'Addition' and 'Concatenation' represent the respective addition and concatenation fusion methods. The parameter'FC' means the fully connected layer followed after the fusion operation, as shown in Figure 2. The experimental results show that the combination of the addition operation and FC of the proposed module is best for the three classes: the car, pedestrian, and cyclist. The data in the first row give the results of the proposed method when only taking a point cloud as input. Compared with the LiDAR-based method (the first row), the proposed method (the fourth row) achieves 0.45%, 0.57%, 1.83%, and 0.6% gains in the 2D, AOS, BEV, and 3D performance, respectively. Compared with the performance improvement of cars, the proposed model is more helpful for improving the identification of pedestrians and cyclists. Additionally, we integrate the pointwise attention (PA) module proposed by Huang et al. [25] into our point feature fusion module. In Table II, our method is better than the PA module.

Figure 3: Qualitative results of the proposed method using the KITTI validation dataset. In the RGB images, the red, cyan, and yellow color represent the predictions for the car, pedestrian, cyclist, respectively. In the point cloud images, the green color denotes the ground truth, and the red color represents the prediction. The results in the point cloud images are used for a qualitative comparison.

Table III: Effect of the proposed framework. The 'Time' column denotes the training time and the 'Memory' is the memory needed when the model is run for four batch sizes. The 'Rtime' column denotes the runtime. 'R' and 'V1' represent ResNet and ResNetV1d, respectively. '2D Image Branch' denotes that if the model use a full 2D image detection branch. The results of the cars are in the 'Moderate' difficulty category for the BEV and 3D.

2D Image Branch	Method	Time (hour)	Memory (MB)	Rtime (FPS)	BEV (%)	3D (%)
Yes	R101	28.0	19,500	9.5	85.95	76.82
	R50	23.5	12,550	11.0	86.29	76.92
	V1-50	25.0	12,700	10.6	85.51	76.48
	VGG11	16	11900	12.0	85.96	76.44
No	Ours	**11.5**	**4200**	**18.0**	86.17	**76.93**

2) Effect of the Proposed Framework: The proposed 3D object detection framework is the first to directly project the raw RGB point features to a point cloud, as shown in Figure 2. The proposed approach is not without precedent but was discovered through experiments. Inspired by MVX-Net [14], we simply wanted to implement a lightweight design based on two backbones. One backbone was intended for 2D detection and the other one for 3D detection. First, ResNet-101 [26] was chosen as the backbone to extract features from RGB images. The results were as expected but the testing model ran very slowly. Then, ResNet-101 was replaced by ResNet-50 [26], and the model ran a little faster but the accuracy was almost the same. When using ResNetV1d-50 [27], the result was almost the same as the result for ResNet-50 [26]. These results are thought-provoking. Hence, we boldly propose to map the raw point features of the RGB image to the point cloud without the 2D detection branch. The experimental results in Table 5 demonstrate that the proposed method is feasible. As can be seen in Table III, the proposed approach not only drastically reduces the memory requirements for model operation, but

also reduces the time of model training by half. It can be said that the proposed framework is lightweight, memory-saving, and energy-saving.

Figure 3 presents some qualitative results. As can be seen in the figures, each object can be detected by the proposed model and the predicted bounding boxes are well-matched with their corresponding ground truth boxes. Even in very complex scenes, the proposed model can detect objects quite well, as shown in the last two rows of Figure 3.

V. Conclusion

This paper is the first to propose a lightweight, memory-saving, and energy-saving framework for 3D object detection based on LiDAR and an RGB camera. Different from the existing frameworks, the proposed framework only employs one backbone to extract features from a point cloud and RGB image. The framework benefits from the proposed module, i.e., the point feature fusion module. The fusion module directly extracts the point features of RGB images and fuses them with the corresponding point cloud features. The experimental results using both the KITTI validation dataset and testing dataset demonstrate that the proposed method significantly improves the speed (23 FPS) of LiDAR-camera-based 3D object detection compared with other state-of-the-art approaches. Note that the proposed native model can achieve an inferring speed 23 FPS.

In the future, the proposed method will be directly used in the point-based methods [10], thereby achieving breakthroughs in both accuracy and speed.

References

[1] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6526–6534.

[2] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–8.

[3] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4490–4499.

[4] L. Wen and K.-H. Jo, "Fully convolutional neural networks for 3d vehicle detection based on point clouds," in *Intelligent Computing Theories and Application*. Cham: Springer International Publishing, 2019, pp. 592–601.

[5] L. Wen and J. Kang-Hyun, "Lidar-camera-based deep dense fusion for robust 3d object detection," in *Intelligent Computing Methodologies*. Cham: Springer International Publishing, 2020, pp. 133–144.

[6] L.-H. Wen and K.-H. Jo, "Three-attention mechanisms for one-stage 3d object detection based on lidar and camera," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2021.

[7] L. H. Wen and K. H. Jo, "Fast and accurate 3d object detection for lidar-camera-based autonomous vehicles using one shared voxel-based backbone," *IEEE Access*, vol. 9, pp. 22 080–22 089, 2021.

[8] 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)*, 2017, pp. 77–85.

[9] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 5099–5108.

[10] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3dssd: Point-based 3d single stage object detector," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 037–11 045.

[11] L. Wen, X. T. Vo, and K.-H. Jo, "3d saccadenet: A single-shot 3d object detector for lidar point clouds," in *2020 20th International Conference on Control, Automation and Systems (ICCAS)*, 2020, pp. 1225–1230.

[12] W. Shi and R. Rajkumar, "Point-gnn: Graph neural network for 3d object detection in a point cloud," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 1708–1716.

[13] D. Xu, D. Anguelov, and A. Jain, "Pointfusion: Deep sensor fusion for 3d bounding box estimation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 244–253.

[14] V. A. Sindagi, Y. Zhou, and O. Tuzel, "Mvx-net: Multimodal voxelnet for 3d object detection," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7276–7282.

[15] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[16] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems*, vol. 28, 2015, pp. 2017–2025.

[17] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, Oct 2018.

[18] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 689–12 697.

[19] Y. Zhou, P. Sun, Y. Zhang, D. Anguelov, J. Gao, T. Ouyang, J. Guo, J. Ngiam, and V. Vasudevan, "End-to-end multi-view fusion for 3d object detection in lidar point clouds," in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 100, 30 Oct–01 Nov 2020, pp. 923–932.

[20] C. He, H. Zeng, J. Huang, X. S. Hua, and L. Zhang, "Structure aware single-stage 3d object detection from point cloud," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 870–11 879.

[21] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "Pv-rcnn: Point-voxel feature set abstraction for 3d object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 10 526–10 535.

[22] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2020.

[23] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 918–927.

[24] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3d object detection," in *Computer Vision – ECCV 2018*. Cham: Springer International Publishing, 2018, pp. 663–678.

[25] T. Huang, Z. Liu, X. Chen, and X. Bai, "Epnet: Enhancing point features with image semantics for 3d object detection," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 35–52.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, pp. 630–645.