IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 26, 2021, accepted June 3, 2021, date of publication June 7, 2021, date of current version June 18, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3087179

Real-Time 3D Object Detection From Point Cloud Through Foreground Segmentation

BO WANG[1,2], MING ZHU[1], YING LU[1,2], JIARONG WANG[1], WEN GAO[1], AND HUA WEI[1,2]
[1]Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, Changchun 130033, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China

Corresponding author: Ming Zhu (zhu_mingca@163.com)

This work was supported by the Education Department of Jilin Province, China, under Grant JJKH20200780KJ.

ABSTRACT This paper aims to apply real-time light-weight high-precision 3D detection for autonomous driving. We propose LIDAR-based 3D object detection based on foreground segmentation using a fully sparse convolutional network (FS^{2}3D). We design a sparse convolutional backbone network and a sparse convolutional detection head to efficiently use the computing and memory resources and accelerate the inference. Instead of using the anchor-based method, we convert the detection problem into a foreground segmentation problem on a bird's-eye view. The sparse convolutional detection head predicts the objectness and bounding box on each active point on the sparse feature map. We design a new oriented bounding box coding method and corresponding loss functions. We predict the endpoints of two mutually perpendicular lines that pass through the foreground active points and indirectly predict the objects' oriented bounding box from these four endpoints. We use the indirectly calculated object center, size, and orientation as inputs of loss functions in the training step. Experiments on the KITTI dataset show that the sparse backbone network we designed is 2.2 times faster and 18.4 times fewer FLOPs than the dense backbone network. The average improvement of the loss functions based on the bounding box code is 1.1% and 0.8% on the BEV and 3D detection, respectively, compared to no addition of these losses. Moreover, FS^{2}3D outperforms the state-of-the-art LIDAR-based method in speed and precision for both cars and cyclists.

INDEX TERMS 3D object detection, LIDAR, real-time, sparse convolutional neural network.

I. INTRODUCTION

Three dimension object detection is a fundamental capability of perception systems in autonomous driving. It helps autonomous vehicles recognize and locate objects in the 3D scene, such as cars, pedestrians, and cyclists. Compared with 2D object detection providing pixel location and pixel size of objects, 3D object detection provides the location, dimension, and pose of objects in the real 3D world. The autonomous driving decision system uses this information to guide the vehicle to drive safely. Therefore, the 3D object detection algorithm's speed and precision directly affect the safety of autonomous vehicles. To accurately locate objects, LIDAR has become an indispensable sensor, which provides reliable depth information. LIDAR measures the distance and reflection of objects' surface in the 3D world under a certain horizontal and vertical angular resolution. The point cloud from LIDAR is a sample of the surface of the scene. Limited by LIDAR's angular resolution, the point cloud is sparse and disordered, making it difficult to detect objects from the point cloud.

In recent years, the convolutional neural network (CNN)-led deep learning method has achieved great success in 2D object detection, object tracking, semantic segmentation, etc. CNN also has been introduced in the field of 3D object detection. Compared with the 2D ordered and densely arranged image, the point cloud is sparse, disordered, and has a large difference of local density in 3D space [1], [6]. Those unexpected properties bring significant challenges to processing the point cloud using 2D CNN. However, there are many methods that can convert the disordered point cloud to an ordered representation. The first method [4], [5], [20] is converting the point cloud to the 3D voxel and using the 3D convolutional neural network to extract the 3D feature from the 3D voxel. The second method [1], [18], [19] is projecting the point cloud into the Front View (FV) or the Native Range View (NRV), FV and NRV is a dense 2D pseudo-image. The third method [1], [7], [28] is projecting the point cloud onto

The associate editor coordinating the review of this manuscript and approving it for publication was Syed Islam.

Bird's Eye View (BEV) to produce a sparse pseudo-image. FV, NRV, and BEV can be processed by 2D CNN.

The front view is denser than the bird's eye view and 3D voxels, which means that 2D convolution can be used efficiently for feature extraction. However, object occlusion and perspective problems are unavoidable.3D voxel can re-represent the point cloud data more accurately, but 3D voxel feature is a 4D tensor, and extracting features with 3D convolution is more difficult and slower compared with extracting feature from FV and BEV, but the performance of 3D voxel-based target detection methods are better than the front view method. Both BEV and 3D retain information about the location, size, and orientation of the objects, but BEV is a 2D feature map, making the BEV-based detection method faster than 3D Voxel-based methods. Also, all objects do not overlap, and the scale of objects is consistent across all locations on the bird's eye view. A lot of works focus on the BEV method.

Suppose that in the autonomous vehicle coordinate system, the X-axis points forward from the vehicle, the Y-axis points to the left, and the Z-axis points up from the ground. All vehicles on the road are driving on the X-Y plane. In this context, the information of surrounding objects along the X-axis and Y-axis is more valuable than the information along the Z-axis. And the yaw around the Z-axis is the movement direction of objects. The X-Y plane can be representing as the bird's eye view.

Projecting a point cloud onto a bird's-eye view is a process that converts a point cloud into a multichannel pseudo image. Firstly, the point cloud is limited in the range $[y_{min}, y_{max}]$ and $[x_{min}, x_{max}]$ for the Y-axis and X-axis. Then, set a grid size and discretize the point cloud into the corresponding grid. For every non-empty grid in the BEV map, we extract the feature. In early works, researchers design hand-craft feature extractors to get the BEV input feature. For example, in MV3D [1], the bird's eye view representation is encoded by height, intensity, and density. An intensity map is the reflection value of the point which has the maximum height in each cell. The density map is the number of points in each cell. The point cloud is divided equally into M slices along the Z-axis to get more detailed height features. For the i-th slice, its height feature is computed as the maximum height of points in each cell. Thus the disordered point cloud is encoded as $(M + 2)$-channel features (pseudo-image). In Complex-YOLO [6], only one slice of height is computed to generate a 3-channels RGB feature combined with intensity map and density map. Though the hand-craft feature is simple to apply and interpretable, it is shallow and unlearnable, which limits the performance of 3D object detection. In 2018, Qi proposed the PointNet [26], which can process the disordered point cloud directly. The PointNet consists of several linear layers followed by a Relu layer, and it can extract point-wise features from the point cloud. After employing the element-wise max-pooling, a global feature is gotten. PointPillars [28] introduced the PointNet to convert the point cloud into a BEV pseudo-image.

The BEV map is quite sparse because most of the grids on the BEV map do not have points. The BEV map's sparsity is defined as the ratio of the number of non-empty grids to the total number of grids [9]. The sparsity depends on the grid size. The smaller the grid size, the greater the sparsity. While present BEV methods employ 2D CNN to extract the feature of the pseudo image, the 2D CNN kernel sliding everywhere on the feature map, including empty grids. Though the CNN is optimized to be computed efficiently, a large amount of memory is consumed when processing the BEV map due to saving useless information, and a large number of computing resources are used to process useless grids. This is a waste of time, space, and power.

In this paper, we propose a novel 3D detection network, named FS^{2}3D (**3D** object detection based on **F**oreground **S**egmentation using **F**ully **S**parse convolutional networks) in which all convolution layers consist of normal sparse convolutions and submanifold sparse convolutions. Because the output prediction map is also sparse and few non-empty grids are assigned a foreground label makes the anchor-based detector training difficult in the training step, we predict objects by grid-wise foreground segmentation rather than the anchor-based method. Inspired by [10], we represent objects as two mutually perpendicular lines across the foreground grids. We regress endpoints of two lines, and these values are used to calculate the center, dimension, and orientation of objects indirectly. For every foreground grid, we regress a series of bounding boxes in N directions. We predict the score of every direction and get the bounding box with the maximum direction score. In [10], besides the endpoints regression loss, the collinear loss, and the vertical loss are introduced to constrain the endpoints of every line to be collinear and the two lines to be perpendicular to each other. To regress the bounding box better, we design a serial of loss functions for regressing objects' center, dimension, and orientation. The center, the dimension, and the orientation of the object are calculated from the endpoints of two lines.

We evaluate our FS^{2}3D Network on the KITTI dataset [11]. Results of our detection network achieve comparable performance to the state-of-the-art method, and our detection network can run at 55.1 FPS in PyTorch [12] with lower computation cost.

The main contributions of our work are summarized as follows:

- We propose a novel end-to-end 3D object detection network architecture, in which all convolution layers consist of sparse convolutions.
- We propose a novel oriented bounding box regression method by predicting endpoints of the vertical lines across the foreground grid on the BEV map.
- We propose three auxiliary loss functions to constrain regression of the center, the orientation, and the dimension of 3D objects.
- The proposed 3D object detection network can run at 55.1 FPS with comparable detection precision and lower

computation cost compared with the state of the art methods.

II. RELATED WORK

LIDAR and camera are essential sensors for autonomous driving vehicles. Most 3D object detection networks are based on LIDAR and camera. In this section, we briefly review three types of existing works on 3D object detection, including camera-based, LIDAR-based, and fusion-based methods. We also review sparse convolutional networks.

A. CAMERA-BASED METHODS

Mono3D [16] is an early work of monocular-camera-based 3D object detection. In Mono3D, an energy minimization approach is proposed to generate 3D candidate boxes. Then, each 3D candidate box projected to a 2D image is scored via several hand-craft features such as semantic segmentation, contextual information, size, location priors, and typical object shape. In cite3D Box Estimation, the network first regresses the 2D and 3D dimension and orientation of an object, and then the real object pose is selected by geometric constraints. In [17], the coarse 3D object bounding box is predicted by the modified faster R-CNN framework. Then it is refined by the guide of its surface feature extracted from the 2D bounding box. Except for monocular-based methods, there are several works based on stereo methods. 3DOP [13] has the same pipeline as Mono3D, while 3DOP has another stream network to process stereo depth information, and 3DOP gets better performance than Mono3D. Stereo R-CNN [14] takes a pair of right and left images as the input of the network and simultaneously detects objects in these two images to produce a coarse 3D object bounding box using stereo Region Proposal Network. The coarse 3D object bounding box is refined by the dense 3D Box Alignment. Tested on the KITTI dataset, Stereo R-CNN outperforms other monocular-based method and 3DOP in terms of precision of object detection. However, the distance information loses when the 3D object is projected onto the image plane. It is hard to recover an accurate depth of the 3D scene by monocular or stereo depth estimation. The performance of image-based methods is limited and depends on the distance of objects—the longer distance of the object, the worse result of object detection.

B. LIDAR-BASED METHODS

The point cloud generated by LIDAR is a sample of information about the surface of objects in the 3D space, including spatial coordinates and surface reflectance. Moreover, limited by the scanning resolution of LIDAR, it is essentially sparse and disordered. In order to process the point cloud using deep learning methods, the point cloud can be converted into Front View (FV), Bird's Eye View (BEV), and 3D voxel representation. There are also many methods that process the point cloud directly.

1) METHODS CONVERTING THE POINT CLOUD INTO FV

VeloFCN [19] encodes the point cloud's distance and height and projects it into a 2-channel image discrete by an average horizontal and vertical angle resolution between consecutive LIDAR beam. The image's feature is extracted by a CNN network and feeds it into two branches to predict the object confidence and bounding box. In MV3D [1], the front view image is encoded by height, distance, and intensity. In [18], the point cloud is encoded and projected into native range view built by mapping the laser id to rows and discretizing azimuth into columns. The NRV images are denser than BEV and 3D voxel representations. LMNet [38] encodes the point cloud into a 5-channel image with five representations (reflectance, range, forward, side, and height) and proposes a detection network that uses dilated convolution and proposes a detection network that uses dilated convolution. Though LMNet achieves real-time multiclass object detection on CPU, the object detection precision is significantly lower than MV3D [1]. The above methods are one-stage methods. FVNet [39] and RangeRCNN [40] are two-stage methods. FVNet first generates proposals on FV feature maps using a CNN and uses a parameter estimation network extended from PointNet [26] to regress the final bounding box. RangeRCNN follows a Range View to Point View to Bird's Eye View pipeline and achieves better performance compared with the above methods.

2) METHODS CONVERTING THE POINT CLOUD INTO BEV

In MV3D, the point cloud is divided into several slices along the height to generate height feature maps. The height maps are combined with the intensity map and the density map obtained from the point cloud to obtain channel BEV maps. Complex YOLO [6] reduces the number of channels of height maps to one channel, combined with the intensity map and the density map to obtain a 3-channel RGB-like BEV image. Complex YOLO follows the YOLO network and encodes object orientation as a complex angle to increase object detection speed and precision. PIXOR [7] modifies the encoding method based MV3D and uses a faster SSD detector to balance high accuracy and real-time efficiency. Pointpillars [28] uses a simple version of PointNet to extract local features of each pillar on the BEV map to make the feature of each BEV's grid learnable and achieves higher accuracy and speed. However, these methods use 2D dense CNN in their network that wastes a lot of computation resources because the BEV map is very sparse, and extracting features of empty regions is useless.

3) METHODS CONVERTING THE POINT CLOUD INTO 3D VOXELS

3D-based methods convert the point cloud into 3D voxels containing feature vectors. Vote3D [3] uses a convolution-like voting-based algorithm and a sliding window detector based on SVM. Vote3Deep [2] proposes a novel convolutional layer

leveraging a feature-centric voting scheme to exploit the sparsity of 3D voxels. Vote3D and Vote3Deep encode the 3D voxel using hand-craft methods, which feature is shallow cannot adapt complex autonomous driving task. VoxelNet [5] first proposes the voxel feature encoding (VFE) layer to make voxel feature learnable from training data. The voxel feature is 3D dense tensor and processed by 3D convolutional middle layers to obtain a 2D BEV feature map. VoxelNet follows the SSD network for detection. Though the accuracy of VoxelNet is better than previous methods, it cannot achieve real-time detection because the computation of 3D convolutional middle layers is huge. To overcome the drawback of 3D convolution layers dealing with 3D dense tensor, SECOND [4] introduces 3D sparse convolution layers to replace normal 3D convolutional layers. The voxel feature is encoded as a sparse tensor containing features and corresponding coordinates. SECOND follow the pipeline of VoxelNet and achieve real-time detection performance and better detection precision.

4) OTHER METHODS

PointRCNN [32] proposes a two-stage detection network. In the first stage, a PointNet is used to extract the point-wise feature. Then, an MLP computes 3D proposals based on every point-wise vector. In the second stage, the local spatial point combined with the expended 3D proposal's semantic feature is calculated to predict the final 3D bounding box. STD [33] is another type of two-stage 3D object detection, which is more complex than PointRCNN. The first stage of STD is a bottom-up proposal generation network that uses raw point clouds as input to generate accurate proposals by seeding each point with a new spherical anchor. Then, PointsPool transforms interior point features from sparse expression to compact representation. The second stage predicts bounding boxes with the IoU branch. The performance of STD is better than PointRCNN in aspects of precision and speed.

C. FUSION-BASED METHODS

In MV3D, the point cloud is converted into the BEV map and the FV map, then both feature maps extracted by different CNN networks are fused with the feature map from the image. In AVOD [25], only the BEV feature map and image feature maps are used to generate proposals and refine the final 3D bounding box. AVOD performs better than MV3D. F-PointNet [22], F-ConvNet [21] and RoarNet [24] use frustum proposals from 2D object detection as 3D proposals and regresses the 3D bounding box based on points in 3D proposals. The performance of 2D driven 3D relies on 2D object detection precision. ContFuse [23] proposes a two-stream detection network architecture using continuous convolutions to fuse image and LIDAR feature maps deeply in multiple scales. The 3D bounding boxes of objects are predicted based on the bird's eye view. The fusion-based method performs better than image-based methods and LIDAR-based methods. However, the complex network and multimodal fusion bring a huge computation, limiting the detection speed and implementation in autonomous driving systems.

D. SPARSE CONVOLUTIONAL NETWORKS

Sparse Convolution Neural Network (SCNN) [8] was proposed to speed up convolutional operations on sparse tensors. The main idea of SCNN is that output points are not computed if there are no related input points. However, the output sparsity is higher than the input sparsity. The increasing sparsity can lead to a decrease in speed in subsequent convolution layers due to a large number of active points. B. Graham proposed the submanifold convolution [9] to keep the sparsity after sparse convolution layers by restricting an output location to be active if and only if the corresponding input location is active. Yan proposed a GPU-based rule generation algorithm to replace the CPU-based rule generation algorithm using a hash table propose by B. Graham to speed up sparse convolutions [4].

III. FS^{2}3D NETWORK

In this paper, we propose a real-time 3D detection network with higher accuracy and less computation. We follow the pipeline of PointPillars, a single-stage detection network, while we redesign the architecture of the backbone network, the encoding method of 3D bounding boxes, and corresponding bounding box regression loss functions. We generate proposals through foreground segmentation instead of the anchor-based method. We regression a serial of bounding boxes based on every foreground grid on the BEV feature map. Our network consists of BEV input representation, sparse backbone network, and sparse detection head. An overview of our FS^{2}3D Network is shown in Figure 1.

A. FROM POINT CLOUD TO BEV PSEUDO-IMAGE

To convert the point cloud into BEV representation, the points in the point cloud are projected into the BEV plane and discretized by the 2D grid with a certain size. We first set a detection range $L \times W \times H$ on the BEV plane. Then, the 3D points within this detection range are discretized with a resolution of $d_L \times d_W$ of the per grid. For non-empty grids, we keep T points. If the number of points is larger than T, we randomly sample T points. If the number of points is less than T, we use zero-padding to keep the input dimension. After discretization we get a tensor with a shape $\frac{L}{d_L} \times \frac{W}{d_W}$. While this tensor is sparse that only a few parts of grids are occupied by points. We sample M non-empty grids. If the number of non-empty is less than M, we use zero-padding. Thus, after sampling and padding, we get an input tensor with a shape $M \times T \times D$, where D is the input dimension of point in non-empty grids. We take the vector $[x, y, z, x - \overline{x}, y - \overline{y}, z - \overline{z}, x_p, y_p, r]$ as input, where $\overline{x}, \overline{y}$ and $\overline{z}$ are coordinate average value of points in the grid and x_p and y_p are the center's coordinates of each grid. And, r is the reflectance of each point in the grid.

We use a simple version of PointNet [26] mentioned in PointPillars [28], a single linear layer followed by a

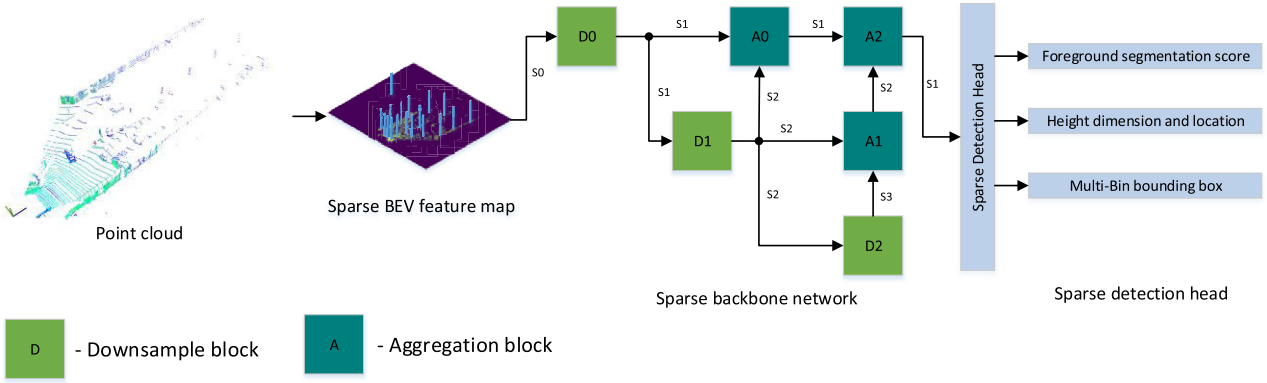


FIGURE 1. The architecture of FS^{2}3D. FS^{2}3D only takes a LIDAR point cloud as input. The sparse BEV feature map coded from the point cloud is inputted into a sparse backbone network, called S-DLA. The S-DLA has three downsampling blocks and three aggregation blocks. S0 is the initial scale of the sparse BEV feature map. S1, S2, and S3 denote scales of output feature maps from corresponding blocks. The scale S1 of the output feature map from S-DLA depends on the stride of block D0. The Sparse detection head has three branches sharing a common feature map outputted from S-DLA. Arrows in the figure represent the direction of data flow.

BatchNorm [41] layer and a ReLU [42] layer to extract the point-wise feature for each grid and apply max-pooling on the point-wise feature to get the element-wise maximum value as the grid feature. For each grid, we extract C dimensions feature vector, thus combining with the corresponding coordinates of each grid. Finally, we get a sparse BEV feature map with a shape $M \times C$.

B. SPARSE BACKBONE NETWORK

Previous BEV-based 3D object detection applies the dense 2D convolutional network to extract the feature from the pseudo-image. However, when the kernel of convolution slides on the pseudo-image, all the empty grids are computed. According to our statistics, the sparsity of the input tensor is less than 0.1 if the resolution of the grid is less than 0.24 m. Moreover, the features extracted from empty grids are useless but take up a lot of memory, which wastes both computing and memory resources. To overcome the drawback of dense convolution, we replace the dense convolutions with sparse convolutions and design a Sparse Deep Layer Aggregation network (S-DLA) as our detector's backbone network. The deep layer aggregation network architecture is proposed to effectively extract and combine multiscale feature [36]. The S-DLA network consists of two types of sparse convolution blocks: downsample blocks and aggregation blocks. Downsample blocks consist of a sparse convolution layer and several submanifold sparse convolution layers, shown as Figure 2. The first sparse convolutional layer of downsampling block has a stride S, and other submanifold convolution layers have a stride 1. Downsampling blocks take a fine feature map as input data and output a coarse feature map with a larger scale if the stride s is bigger than 1. Aggregation blocks fuse feature maps from two adjacent scales and keep the output scale the same as the feature map with the small scale (fine feature map), shown in Figure 2. We use an inverse

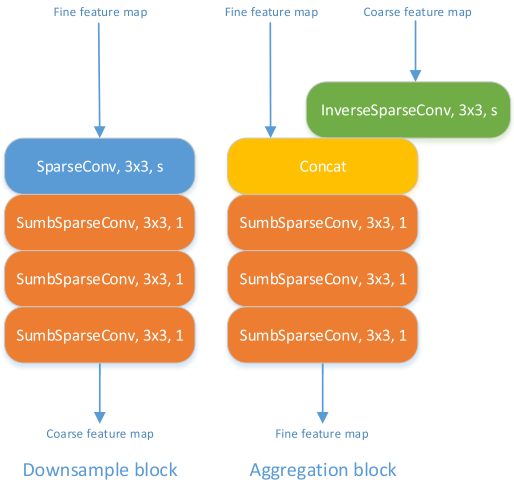


FIGURE 2. The structure of the downsampling block and the aggregation block.

sparse convolution to upsample the coarse feature map with a larger scale and recover the sparse tensor's active points. After that, we use a concatenation layer to fuse features. Finally, the output from A2 has the same scale as the output from D0.

C. DETECTION HEAD

In this paper, we propose a novel oriented bounding box encoding-decoding method. We represent the bounding box as two mutually perpendicular lines that pass through the object's foreground grid on the BEV. Both lines are also perpendicular to the boundaries of objects. The bounding box representation is shown in Figure 3.

In the stage of predicting the bounding box, the network regresses coordinate offsets of endpoints of two mutually

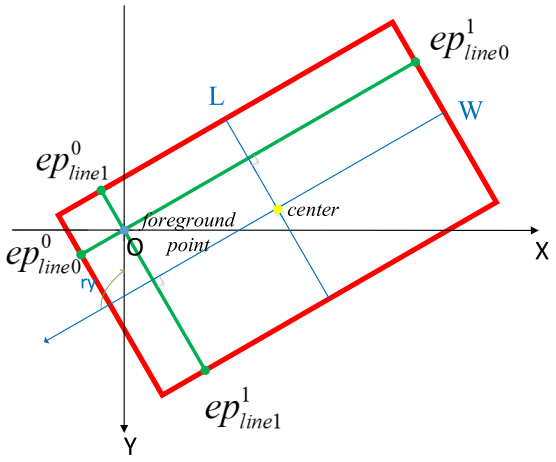


FIGURE 3. Representation of bounding box. The blue point is a foreground point. The two green lines are two mutually perpendicular lines which endpoints are predicted by the detector. The red box is the object's oriented bounding box in the BEV. The yellow point is the object's center. The two blue lines are the axis of symmetry of the object, and one of them with the arrow represents the object's orientation.

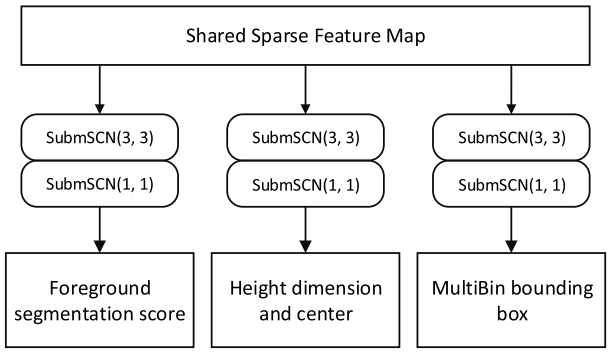


FIGURE 4. The structure of sparse detection head.

perpendicular lines based on every foreground point to calculate the final bounding box mediately. Typically, the bounding box of the object is denoted by its center, dimension, and orientation. We consider the center of the final bounding box is the intersection point of the perpendicular bisectors of two lines, the dimension of the bounding box is the length of two lines, and the orientation is the direction of ray from ep^1_{line0} to ep^0_{line0}. To regression a more accurate oriented bounding box, we leverage a MultiBin method [15] to discretize the orientation angle and divide it into N bins. For each bin, we predict a 9-channel vector that contains a bin's confidence score and 8 offsets of endpoints of two perpendicular lines from the foreground point. Then, we predict the height and the height dimension and combine it with other parameters mentioned above to get a 3D bounding box.

The detection head we proposed is a multitask network that contains three branches to predict the 3D bounding box of the object, and its architecture is shown in Figure 4. All branches have same structure - two submanifold sparse convolutional layers. The first submanifold sparse convolutional layer has a 3×3 kernal. and the second submanifold sparse convolutional layer has a 1×1 kernal. The foreground segmentation branch outputs 1-channel sparse score map followed with a sigmoid activation layer. The height dimension and location branch outputs 2-channel sparse map to regression the object's height and height center. The MultiBin bounding box branch outputs $N \times 9$-channel sparse map. All branches of the detection head share a common sparse feature map extracted from the sparse backbone network S-DLA.

D. LOSS

We leverage the commonly used task loss to train our detector. The total loss has three parts, including foreground segmentation loss, height regression loss, and MultiBin bounding box regression loss. We use modified focal loss on the foreground segmentation score $Y'_{\hat{xy}}$. To get the label of foreground segmentation, we first compute the set of active points (non-empty grids) on the output feature map from the backbone network and project 3D bounding boxes into BEV. We assign the active point $Y_{\hat{xy}}$ inside projected bounding box a foreground label 1 and outside active point a value 0.

$$L_{fs} = -\frac{1}{N}\sum_{\hat{xy}} \begin{cases} (1 - Y'_{\hat{xy}})^{\alpha} log Y'_{\hat{xy}}, & if\ Y_{\hat{xy}} = 1 \\ {Y'_{\hat{xy}}}^{\alpha} log(1 - Y'_{\hat{xy}}), & otherwise, \end{cases} \tag{1}$$

where α is the hyper-parameter of modified focal loss. $\hat{xy}$ is the active point on the sparse feature map. N is the number of active points on the output feature map.

We use the smooth $l1$ loss on the regression of height and height center. For regression targets of height h_t and height center z_t, we use the following encoding functions:

$$h_t = \frac{h_g}{h_a}, \quad z_t = z_g - z_a \tag{2}$$

where h_g and z_t are ground truth of height and height center. h_a and z_a are average values of height and height center.

The regression loss of height and height center has following form:

$$L_h = \frac{1}{N}\sum_{q\in\{h,z\}}\sum_{\hat{xy}\in \boldsymbol{f}} SmoothL1(q'_{\hat{xy}}, q_{\hat{xy}}) \tag{3}$$

where $\boldsymbol{f}$ is the set of active points with the foreground label, and N is the size of this set. h denotes the object's height. z denotes the object's height center. q' represents the predicted value, and q represents the ground truth.

The MultiBin bounding box loss L_{mbb} has two parts including bin classification loss L_{bc} and endpoints regression loss L_{er}. We use the cross-entropy loss on bin classification. To make the endpoints regression more accurate, we design a serial of loss functions that constrain the endpoints regression from multiple aspects. We extend the center regression loss, dimension regression loss, and rotation regression loss based on the middle line loss of [10]. The first loss is to regress endpoints of the two mutually perpendicular lines, and it is

shown as follows:

$$L_1 = \frac{1}{N}\sum_{ep=0}^{2}\sum_{\hat{xy}}[SmoothL1\left(\Delta x^*_{\hat{xy}}, \Delta x_{\hat{xy}}\right) + SmoothL1\left(\Delta y^*_{\hat{xy}}, \Delta y_{\hat{xy}}\right)]_{ep} \quad (4)$$

where N is the number of active points with foreground label. $\hat{xy}$ is the coordinate of the active point with foreground label. Δx and Δy are predicted endpoints offsets, and Δx^* and Δy^* are the ground truth. ep is the endpoints of the corresponding line.

The second loss is to solve the endpoints of one of lines being not collinear caused by that the endpoints are predicted independently. The loss is as follows:

$$L_2 = \frac{1}{N}\sum_{l}^{2}\sum_{\hat{xy}}[SmoothL1(\Delta x^{ep0}_{\hat{xy}} \times \Delta y^{ep1}_{\hat{xy}}, \Delta x^{ep1}_{\hat{xy}} \times \Delta y^{ep0}_{\hat{xy}})]_l \quad (5)$$

where l denotes the two lines. $ep0$ and $ep1$ are the two endpoints of a line.

The two lines are perpendicular to each other. To make the network learn to predict perpendicular lines, the following loss is needed:

$$L_3 = \frac{1}{N}\sum_{\hat{xy}}[SmoothL1(\Delta x^{ep0_{l0}}_{\hat{xy}} \times \Delta x^{ep0_{l1}}_{\hat{xy}} + \Delta y^{ep0_{l0}}_{\hat{xy}} \times \Delta y^{ep0_{l1}}_{\hat{xy}}), 0]_l \quad (6)$$

where $ep0_{l0}$ and $ep0_{l1}$ are the 1st endpoint of line 0 and the 1st endpoint of line 1.

We calculate the object's center, dimension and orientation angle through the endpoints of the two lines. Those value have the corresponding ground truth and can therefore be used to calculate losses. To constrain the center regression, we design the center regression loss as follows:

$$L_4 = \frac{1}{N}\sum_{\hat{xy}}[SmoothL1(\Delta x^*_{center}, \Delta x_{center}) + SmoothL1(\Delta y^*_{center}, \Delta y_{center})] \quad (7)$$

where Δx and Δy are the predicted coordinate of the object's center. Δx^* and Δy^* are the found truth.

The object's length and width are the length of line 0 and line 1. To make the detector learn to predict object's dimensions, we design the dimension regression loss as follows:

$$L_5 = \frac{1}{N}\sum_{\hat{xy}}[SmoothL1(w^*_{\hat{xy}}, w_{\hat{xy}}) + SmoothL1(l^*_{\hat{xy}}, l_{\hat{xy}})] \quad (8)$$

where w and l are predicted values. w^* and l^* are the ground truth.

The object's rotation angle is the direction of ray from ep^1_{line0} to ep^0_{line0}. To make the network learn to predict rotation angle, we introduce the rotation regression loss as follows:

$$L_6 = \frac{1}{N}\sum_{\hat{xy}}[SmoothL1(sin(r^*_{\hat{xy}}, r_{\hat{xy}}), 0)] \quad (9)$$

where r is the predicted value. r^* are ground truth.

The L_1, L_2, L_3, L_4, L_5 and L_6 make up the endpoint regression loss:

$$L_{er} = \sum_{i=1}^{6} w_i L_i \quad (10)$$

where w_i are the weights of losses.

The MultiBin bounding box loss is as follows:

$$L_{mbb} = w_0 L_{bc} + L_{er} \quad (11)$$

where w_0 is the weight of the bin classification loss.

The total loss of our model can be expressed as:

$$L_{total} = \alpha L_{fs} + \beta L_h + L_{mbb} \quad (12)$$

where α, β are the weights of losses.

IV. IMPLEMENTATION

In this section, we describe the implementation detail of our network.

A. NETWORK ARCHITECTURE

In the spare BEV feature coding network, we set the linear layer's input/output channel to (9, 64). In the backbone network, our S-DLA network consists of 3 downsampling blocks and 3 aggregation blocks.

The downsampling block has a normal sparse convolution with 3 sparse submanifold convolution layers. The parameters of downsampling block are $(C_{in}, C_{out}, K, S, N_{subm})$. In the downsampling block, the input sparse tensor is downsampled by a normal sparse convolution with the input channel C_{in}, the output channel C_{out}, the kernel size $K \times K$ and the downsampling stride S. The output feature is fed into the sparse submanifold convolution with the input channel C_{out}, the output channel C_{out}, the kernel size K. We cascade N_{subm} sparse submanifold convolution layers. The aggregation block takes two feature maps as inputs, one of them is a coarse feature map from the downsampling block with a larger scale, and another one is a fine feature map from the block with a smaller scale. The parameters of aggregation layer are $(C^l_{in}, C^s_{in}, C_{out}, K, S, N_{subm})$. In aggregation block, we first upsample the coarse feature map using sparse inverse convolution layer with the input channel C^l_{in}, the output channel C^l_{in}, the kernel size K and the upsampling stride S to make its size and active point same with the fine feature map, then we concatenate channel-wise feature vectors of upsampled feature map with feature vectors from fine feature map using concatenation layer to produce a $C^l_{in} + C^s_{in}$ feature map. We cascade N_{subm} submanifold sparse convolution layer. The first of them compresses the channel of feature map from $C^l_{in} + C^s_{in}$ to C_{out} with a kernel size $K \times K$. The following

TABLE 1. Evaluation of Lidar-based 3D object detectors in KITTI validation dataset.

Class	Method	Inference Time(ms)	AP-BEV				AP-3D			
			Easy	Moderate	Hard	Mean	Easy	Moderate	Hard	Mean
Car	MV3D [1]	360	86.18	77.32	76.33	79.94	71.19	56.60	55.30	61.03
	PIXOR [7]	93	86.79	80.75	76.60	81.38	N/A	N/A	N/A	N/A
	VoxelNet [5]	225	89.60	84.81	78.57	84.33	81.97	65.46	62.85	70.09
	SECOND [4]	50	89.96	87.07	79.66	85.56	**87.43**	76.48	69.10	77.67
	PointPillars [28]	16(24)	**90.07**	**87.44**	84.78	87.43	85.22	**76.52**	69.73	77.16
	$FS^2$3D	**18.1**	90.04	87.19	**85.95**	**87.73**	87.12	75.76	**73.91**	**78.93**
pedestrian	VoxelNet [5]	225	69.95	61.05	56.98	62.66	57.86	53.42	48.87	53.38
	PointPillars [28]	16(24)	**71.18**	**65.92**	**61.36**	**66.15**	**63.87**	**58.08**	**52.80**	**58.25**
	$FS^2$3D	**19.5**	68.08	61.05	56.88	62.00	60.47	55.88	49.89	55.41
Cyclist	VoxelNet [5]	225	74.41	52.18	50.49	59.03	67.17	47.65	45.11	53.31
	PointPillars [28]	16(24)	82.20	62.81	59.77	68.26	77.16	59.22	55.91	64.10
	$FS^2$3D	**19.5**	**88.08**	**69.77**	**64.80**	**74.22**	**86.47**	**66.10**	**63.08**	**71.88**

sparse submanifold convolution layers have the same parameters (C_{out}, C_{out}, K). We add a BatchNorm layer after every convolution layer.

We use three two-layer-submanifold-convolution branches in sparse detection head to predict foreground score, height and height center, and MultiBin bounding box. We use a 3×3 kernel convolution for every branch, followed by a BatchNorm layer to extract the branch-special feature from the shared sparse feature map. For foreground segmentation task, we add a submanifold convolution with parameters (C_{in}, 1, 1) followed by a sigmoid layer to get the probability. For height and height center regression task we add a submanifold convolution with parameters (C_{in}, 2, 1). And, for bounding box regression task we add a submanifold convolution with parameters (C_{in}, $9 \times N_b$, 1).

B. CAR DETECTION

Following the detection range for cars in PointPillars [28], we only take the point cloud within x, y, z range $[(0, 70.4), (-40, 40), (-3, 1)]$ meters as network's input. In input representation network we set the grid size to $d_L = 0.16$ and $d_W = 0.16$ meter, the maximum number of point in grid T to 1000. We only keep $M = 12000$ non-empty grids. For every point in a grid we generate a 9-channel input vector and feed it into a linear layer with 64-channel output followed by a BatchNorm layer and a ReLU layer to extract 64-channel point-wise features. After element-wise max-pooling we get M grid feature combining with the corresponding coordinates we get a sparse feature map with size of $H = 496$ and $W = 432$. For car detection in the backbone network We assign block D0(64, 128, 3, 2, 3), block D1(128, 128, 3, 2, 3), block D2(128, 128, 3, 2, 3), block A0(128, 128, 256, 3, 2, 3) and block A1(256, 256, 256, 3, 2, 3), block A2(256, 256, 256, 3, 2, 3). In the detection head network we set the first convolution layer (256, 128, 3) in every branch. And all the following 1×1 convolution layers in every branch has 128-channel input. We divide the orientation angle into $N = 8$ bins. For car detection, we define the mean of height h_a and the mean height center z_a as 1.56 and -1.0 meters.

C. PEDESTRIAN AND CYCLIST DETECTION

Following the detection range for pedestrians and cyclists in PointPillars [28], we set the input point cloud range to $[(0, 48), (-20, 20), (-2.5, 0.5)]$ meters for x, y, z respectively. We use the same parameters in the stage of input representation. And we get a feature map that has size of $H = 296$ and $W = 248$. We follow the architecture settings for car detection while we set the stride of block D0 to 1 to get a final feature map with high resolution. We use different detection heads for pedestrians and cyclists, and the parameters of detection head networks are the same as the parameter of the detection head network for cars. The two heads share a common feature map as input. For pedestrian and cyclist detection, we define the mean of height h_a and the mean of height center z_a as 1.73 and -0.6 meters.

D. HYPER-PARAMETERS

We use Adam optimizer together with one-cycle policy with LR max $1 * 10^{-4}$, division factor 10, momentum ranges from 0.95 to 0.85, and fixed weight decay 0.01. The model is trained for 200 epoch with batch size 8 on a NVIDIA RTX2080Ti with 11 GB memory. During inference, we keep top 1000 proposals for each object class, then apply NMS with score threshold 0.3 and IoU threshold 0.5. The parameters for total loss are set to $\alpha = 2$, $\beta = 1$, $w_0 = 1$, $w_1 = 2$ and $w_i = 0.2$ for $i = 2, 3, 4, 5, 6$.

E. DATA AUGMENTATION

Data augmentation is verified useful in training a 3D object detection model [4], [5], [7]. In this paper, it is necessary to introduce the data augmentation strategy to generate more samples of objects for training the model. Our data augmentation pipeline is shown as follows. First, we follow the method of SECOND [4]. We collect the ground truths, including labels, 3D bounding boxes, and the point cloud inside 3D bounding boxes from the training dataset. For every training point cloud and targets fed into the model, we randomly select some labels and corresponding points and put them into the training pairs. We randomly select 15, 10, 10 ground

truth samples for cars, pedestrians, and cyclists. We apply a collision test to avoid overlap between added objects and objects in the original training data.

Next, all ground-truth boxes and corresponding points are individually rotated around the z-axis and translated along X, Y, Z axes [28]. The rotation angle is from a uniform distribution $\mathcal{U}[-\pi/20, \pi/20]$. And the amount of translation is from a Gaussian distribution $\mathcal{N}[0, 0.25]$. After that, we remove background points if they are in a ground truth box.

Finally, we apply random flip along the x-axis [7], random global rotation from $\mathcal{U}[-\pi/4, \pi/4]$), random global translation along X, Y, Z axes from $\mathcal{N}[0, 0.2]$ and random scaling from $\mathcal{U}[0.95, 1.05]$ [4], [5], [28].

V. EXPERIMENTS

We evaluate the performance of proposed FS^{2}3D on the KITTI-object detection benchmark for 3D object detection and BEV object detection tasks on the car, pedestrian, and cyclist. In the KITTI dataset, each class's objects are divided into three different difficulty levels: easy, moderate, and hard, accounted for the size, occlusion, and truncation of objects. The KITTI dataset contains 7481 samples for training and 7518 samples for testing. We only perform experiments on the original training data because access to the official KITTI test server is limited. We split 7481 samples into a training subset with 3712 samples and a validation subset with 3769 samples, according to [13]. We follow the official KITTI evaluation protocols, evaluating the 3D object detection and BEV object detection at a 0.7 IoU threshold for cars and a 0.5 IoU threshold for pedestrians and cyclists. The average precision (AP) is computed as the performance measure. Meanwhile, we measure the inference time of our model on an NVIDIA RTX2080Ti GPU.

A. QUANTITATIVE ANALYSIS

We compare the 3D localization and 3D detection performance of our model with state-of-the-art Lidar-based methods, including MV3D [1], PIXOR [7], VoxelNet [5], SECOND [4] and PointPillars [28]. The validation performance of methods including MV3D, PIXOR, VoxelNet, SECOND is copied from published papers if there are provided. Because the performance of PointPillars isn't available in the article, we get the evaluation results for the car categories from [37] and evaluation results for pedestrian and cyclist category are produced by their released code. The results of the evaluation on the validation dataset are shown in Table 1.

On the car class, FS^{2}3D achieves the best results for the hard instances in both BEV and 3D object detection, increasing 1.17% and 4.18%, respectively, compared with the second-best method PointPillars. Although our model performs slightly worse than other methods on easy and moderate instances, the means of average precision of three levels of difficulty on the BEV detection and 3D detection increase 0.3% and 1.77%, respectively, compared with the second-best performing method. For the pedestrian class, FS^{2}3D is slightly weaker than PointPillars, but it outperforms VoxelNet for 3D detection. In the ablation study section VI.D., we discuss the effect of grid size on detector performance. When the grid size is 0.12 meters, the detection performance of our model for pedestrians outperforms that of Pointpillars on easy and moderate objects, though the inference time is 2 ms longer. For the cyclist class, FS^{2}3D outperforms other methods in all metrics. Compared with PointPillars, FS^{2}3D increases the mean of average precision of three levels of difficulty on the BEV detection and 3D detection by 5.96% and 6.22%, respectively. Because the NVIDIA TensorRT does not support sparse convolution, we only test our FS^{2}3D in the Pytorch pipeline. We measure the inference time of FS^{2}3D on an RTX2080Ti GPU and achieve 18.1 ms of inference time. Compared with Pointpillars running at 42.4 Hz, our FS^{2}3D can run at 55.1 Hz in the Pytorch [12] pipeline.

B. QUALITATIVE ANALYSIS

We visualize the 3D detection boxes on the KITTI validation set in Fig. 5 and Fig. 6.The detection boxes are shown in the LIDAR view and projected onto the image view for better visualization.

Fig. 5 shows high-quality detection results produced by FS^{2}3D. From the figure, it is observed that our model could locate and classify the objects in the complex scene (see the subfigure in the first column of the first row and the subfigure in the fourth column of the first row). Our FS^{2}3D even detects objects not being labeled in the ground truth. For hard-level objects that are heavily occluded or far away from LIDAR, our model could keep the performance of detection if there are enough points for extracting the objectness feature (see the subfigure in the second column of the first row). As observed from the figure, our model could predict the high-accurate orientation of objects, which is necessary for the autonomous driving system to predict the object's motion direction.

Fig. 6 shows the failure case including false positives and false negatives. False positives are most likely to occur on background objects with features similar to object features learned by FS^{2}3D network. For example, the motorcycle is misclassed as a car, and the traffic sign on the sidewalk is misclassed as a pedestrian in the Fig. 6(c). And the pole is identified as a pedestrian in the Fig. 6(d). False negatives mostly occur when the LIDAR points of objects are not enough to extract distinguishable features for detection. The main reasons are distance and occlusion. For example, in the Fig. 6(b), the car on the top in the lidar view provides only a few points that make it difficult to extract feature from those points. The car is misclassed as background on the left of the LIDAR view because of heavy occlusion, see Fig. 6(a). The cyclist occluded by the traffic light pole is misclassed as background, see Fig. 6(c). Two nearby pedestrians only get one

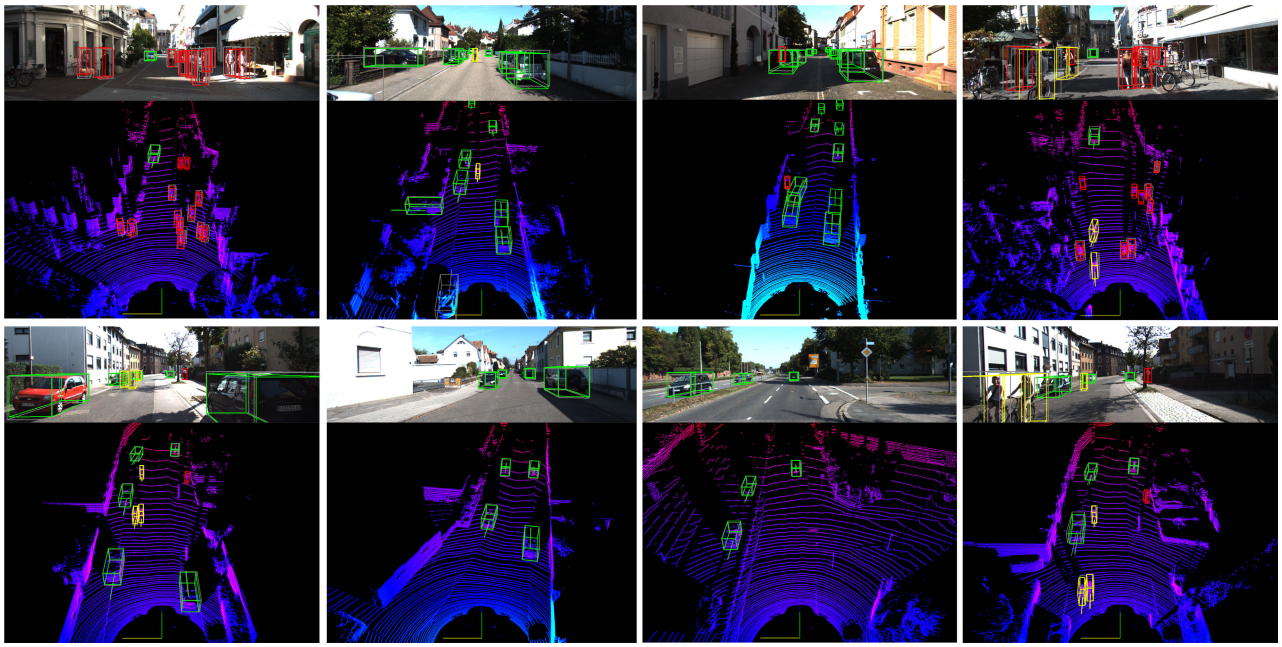
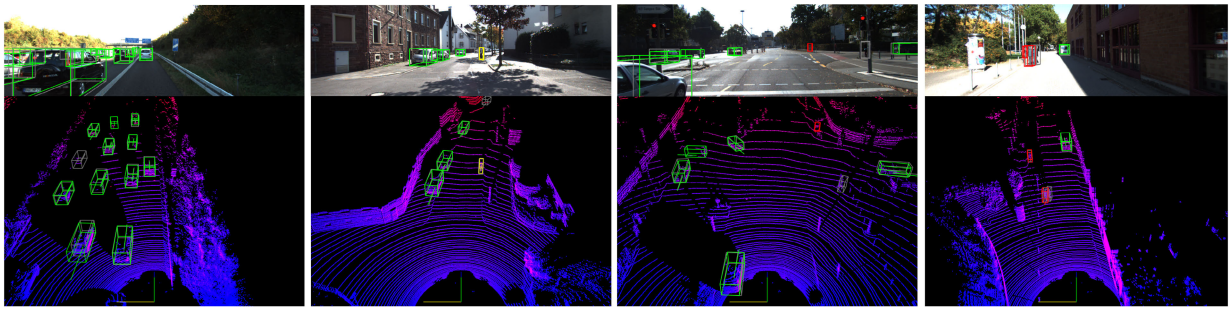


FIGURE 5. Results of 3D detection on the KITTI validation set. In the figure 8 samples are visualized. For each sample, the 3D detection boxes and ground-truth boxes are shown in the Lidar view (the lower part) and projected into the image-view (the upper part). We show the detection boxes for cars (green), pedestrians (red), and cyclists (yellow). The ground-truth boxes are shown in gray. The orientation of boxes is shown by a ray from the bottom center to the front of the box.



(a) (b) (c) (d)

FIGURE 6. The failure case of results on the KITTI validation set. We follow the same visiualiztion setup from Fig.5.

TABLE 2. A comparison of the performance of different backbone networks on the KITTI validation set.

Class	Method	Time (ms)	FLOPs (GMac)	AP-BEV			AP-3D		
				Easy	Moderate	Hard	Easy	Moderate	Hard
Car	Dense-CNN	40.0	455.2	89.23	85.15	84.97	84.53	74.77	73.19
	Sparse-CNN	**18.1**	24.7	**90.04**	**87.19**	**85.95**	**87.12**	**75.76**	**73.91**
Pedestrain	Dense-CNN	61.3	617.0	**66.02**	**59.27**	**55.86**	**58.18**	**54.05**	**48.38**
	Sparse-CNN	**19.5**	25.6	65.6	59.24	55.84	57.16	51.07	47.76
Cyclist	Dense-CNN	61.3	617.0	87.51	65.81	62.77	87.01	64.90	61.58
	Sparse-CNN	**19.5**	25.6	**87.63**	**66.64**	**63.65**	**87.04**	**65.86**	**62.46**

detection box because the pedestrian in the back is occluded, see Fig.6(d).

VI. ABLATION STUDY

A. BACKBONE NETWORK

This section compares the detection performance of models with the sparse convolutional backbone network and the dense convolutional backbone network. Shown in the Table. 2, the detection performance of the method with the sparse convolutional backbone network outperform that of the method with the dense convolutional backbone network for cars and cyclists. For pedestrian detection, the introduction of sparse convolution reduces the performance of BEV detection and 3D detection. This is because the sparse feature

TABLE 3. A comparison of the performance of models with different box coding method and loss function.

Box Coding Method	AP-BEV			AP-3D			AOS		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Traditional	88.89	84.28	84.35	85.24	72.40	67.75	89.76	86.38	83.17
Two line + L_1	89.20	85.48	85.31	85.83	75.11	73.5	90.58	88.57	87.05
Two line + $L_1 + L_2 + L_3$	89.82	86.76	85.92	86.69	75.32	73.84	**90.63**	88.53	86.71
Two line + $L_1 + L_2 + L_3 + L4$	89.74	86.64	85.71	86.52	75.49	73.84	90.57	**88.65**	86.70
Two Line +$L_1 + L_2 + L_3 + L_4 + L_5$	89.11	86.07	85.43	85.84	75.07	73.67	90.62	88.63	**87.10**
Two Line +$L_1 + L_2 + L_3 + L_4 + L_5 + L_6$	**90.04**	**87.19**	**85.95**	**87.12**	**75.76**	**73.91**	90.59	88.41	86.86

TABLE 4. A comparison of the performance of models with different configurations of the number of bins.

Num_Bin	AP-BEV			AP-3D			AOS		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
1	88.69	78.49	77.63	75.21	64.25	61.59	89.89	79.62	78.71
2	89.07	84.27	78.8	81.68	67.37	66.47	90.35	85.54	85.38
4	89.55	85.02	85.12	82.1	71.78	66.76	**90.66**	87.69	85.9
8	**90.04**	**87.19**	**85.95**	**87.12**	**75.76**	**73.91**	90.59	88.41	**86.86**
12	89.86	86.24	85.74	86.55	75.47	73.72	90.58	**88.62**	86.71

map output from the sparse convolutional backbone network has fewer proposals for the detection head than the dense feature map output from the dense convolutional backbone network. In particular, pedestrians occupy fewer grids than cars and cyclists. This effect causes the reduction of the number of foreground proposals for training and detection. So, the performance of pedestrian detection gets worse after the introduction of the sparse convolution.

B. BOX CODING METHODS

As shown in Table 3, we compare our box coding method with box coding method proposed in [15]. Our method with the endpoint regression loss L_1 outperforms the method we compare in all the difficult levels on the BEV detection, 3D detection, and AOS. Adding the endpoint collinear loss and the line vertical loss the BEV and 3D detection performances increase obviously. After adding the center regression loss L_4, the dimension regression loss L_5 and the rotation regression loss L_6, the BEV and 3D detection performances achieve the best results compared with the above three methods. The ablation study shows that the loss functions L_4, L_5, and L_6 we designed could help to increases detection performance for 3D detectors.

C. NUMBER OF BINS FOR ROTATION PREDICTION

The Classification-regression method proposed in [15] has many applications in 3D object detection and has previously only been used to predict target orientation. In this paper, we propose a method that first predicts the orientation angle classification and then predicts the corresponding 3D bounding box of objects. Like predicting the orientation angle influenced by the number of bins, our model is affected by the number of bins. As shown in Table 4, the detection performance of the bounding box increases as the bin increases, and the BEV and 3D detection performances get the best results when the number of $bins = 8$.

TABLE 5. Effect of the grid size for car detection.

Grid Size	FPS	AP-BEV		
		Easy	Moderate	Hard
0.10	39.4	89.48	86.39	85.29
0.12	46.5	89.44	85.39	85.47
0.14	52.8	89.65	85.98	85.45
0.16	55.1	**90.04**	**87.19**	**85.95**
0.18	64.0	88.91	85.00	85.15
0.20	**69.9**	89.75	85.48	85.28

TABLE 6. Effect of the grid size for pedestrian detection.

Grid Size	FPS	AP-BEV		
		Easy	Moderate	Hard
0.12	38.5	**71.20**	**66.56**	59.42
0.16	**51.3**	65.6	59.24	55.84
PointPillars [28]	42.5	71.18	65.92	**61.36**

D. GRID SIZE

The effect of the grid size for car detection is shown in Table 5. The grid size directly determines the number of non-empty grids. In the pseudo-image, the larger grid size brings fewer non-empty grids and faster detection speed and reduces the detection precision for trade-off because the larger grid size can obscure the extracted features inputted to the backbone network, the precision of object classification, and 3D bounding box prediction. The grid size of 0.16 m gets the best results, while the grid size of 0.20 m brings the fastest detection speed. For pedestrian detection, the small grid size of 0.12 meters improves the detection precision compared to the grid size of 0.16 meters shown in Table 6. Simultaneously, when the grid size is 0.12 meters, our model outperforms Pointpillars [28] on easy and moderate objects, although the inference time is 2ms less.

VII. CONCLUSION

In this paper, we introduce FS^{2}3D, a novel end-to-end fully sparse convolutional 3D detection network. We design the S-DLA backbone network and sparse detection head using sparse convolutional layers that only process non-empty grids to reduce the inference runtime and FLOPs. And, the Multi-Bin box coding method can predict 3D bounding box and orientation effectively. Experiments on the KITTI dataset show that our FS^{2}3D outperforms other state-of-the-art methods for detecting cars and cyclists. Meanwhile, our FS^{2}3D realizes real-time 3D detection for cars, pedestrians, and cyclists.

REFERENCES

[1] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1907–1915.

[2] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 1355–1361.

[3] D. Z. Wang and I. Posner, "Voting for voting in online point cloud object detection," in *Proc. Robot., Sci. Syst.*, 2015.

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

[5] Y. Zhou and O. Tuzel, "VoxelNet: End-to-end learning for point cloud based 3D object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4490–4499.

[6] M. Simon, S. Milz, K. Amende, and H. M. Gross, "Complex-YOLO: An Euler-region-proposal for real-time 3D object detection on point clouds," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Munich, Germany, Sep. 2018, pp. 197–209.

[7] B. Yang, W. Luo, and R. Urtasun, "PIXOR: Real-time 3D object detection from point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7652–7660.

[8] B. Graham, "Spatially-sparse convolutional neural networks," 2014, *arXiv:1409.6070*. [Online]. Available: http://arxiv.org/abs/1409.6070

[9] B. Graham, M. Engelcke, and L. V. D. Maaten, "3D semantic segmentation with submanifold sparse convolutional networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9224–9232.

[10] H. Wei, Y. Zhang, Z. Chang, H. Li, H. Wang, and X. Sun, "Oriented objects as pairs of middle lines," 2019, *arXiv:1912.10694*. [Online]. Available: http://arxiv.org/abs/1912.10694

[11] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3354–3361.

[12] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *Proc. Future Gradientbased Mach. Learn. Softw. Techn. (Autodiff), 29th Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017.

[13] X. Chen, K. Kundu, Y. Zhu, H. Ma, S. Fidler, and R. Urtasun, "3D object proposals using stereo imagery for accurate object class detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 5, pp. 1259–1272, May 2018.

[14] P. Li, X. Chen, and S. Shen, "Stereo R-CNN based 3D object detection for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 7636–7644.

[15] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecká, "3D bounding box estimation using deep learning and geometry," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5632–5640.

[16] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3D object detection for autonomous driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2147–2156.

[17] B. Li, W. Ouyang, L. Sheng, X. Zeng, and X. Wang, "GS3D: An efficient 3D object detection framework for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 1019–1028.

[18] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington, "LaserNet: An efficient probabilistic 3D object detector for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 12669–12678.

[19] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3D lidar using fully convolutional network," in *Proc. 12th Robot. Sci. Syst.*, 2016.

[20] B. Li, "3D fully convolutional network for vehicle detection in point cloud," 2016, *arXiv:1611.08069*. [Online]. Available: http://arxiv.org/abs/1611.08069

[21] Z. Wang and K. Jia, "Frustum ConvNet: Sliding frustums to aggregate local point-wise features for amodal 3D object detection," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 1742–1749.

[22] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D object detection from RGB-D data," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 918–927.

[23] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3D object detection," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 641–656.

[24] K. Shin, Y. P. Kwon, and M. Tomizuka, "RoarNet: A robust 3D object detection based on RegiOn approximation refinement," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2019, pp. 2510–2515.

[25] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3D proposal generation and object detection from view aggregation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 1–8.

[26] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 77–85.

[27] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 5099–5108.

[28] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 12689–12697.

[29] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[30] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 21–37.

[31] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 779–788.

[32] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D object proposal generation and detection from point cloud," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 770–779.

[33] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "STD: Sparse-to-dense 3D object detector for point cloud," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1951–1960.

[34] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, 2017.

[35] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Med. Image Comput. Comput.-Assist. Intervent.*, 2015, pp. 234–241.

[36] F. Yu, D. Wang, E. Shelhamer, and T. Darrell, "Deep layer aggregation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2403–2412.

[37] Z. Wang, H. Fu, L. Wang, L. Xiao, and B. Dai, "SCNet: Subdivision coding network for object detection based on 3D point cloud," *IEEE Access*, vol. 7, pp. 120449–120462, 2019.

[38] K. Minemura, H. Liau, A. Monrroy, and S. Kato, "LMNet: Real-time multiclass object detection on CPU using 3D LiDAR," in *Proc. 3rd Asia–Pacific Conf. Intell. Robot Syst. (ACIRS)*, Jul. 2018, pp. 28–34.

[39] J. Zhou, X. Tan, Z. Shao, and L. Ma, "FVNet: 3D front-view proposal generation for real-time object detection from point clouds," in *Proc. 12th Int. Congr. Image Signal Process., Biomed. Eng. Informat. (CISP-BMEI)*, Oct. 2019, pp. 1–8.

[40] Z. Liang, M. Zhang, Z. Zhang, X. Zhao, and S. Pu, "RangeRCNN: Towards fast and accurate 3D object detection with range image representation," 2020, *arXiv:2009.00206*. [Online]. Available: http://arxiv.org/abs/2009.00206

[41] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[42] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. ICML*, 2010, pp. 807–814.

BO WANG received the B.S. degree in microelectronics from Jilin University, in 2016. He is currently pursuing the Ph.D. degree with the Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, China. His research interests include object detection and 3D object detection.

MING ZHU is currently a Research Fellow and a Supervisor of Ph.D. candidates with the Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences. His research interests include digital image processing, television tracking, and automatic target recognition technology.

YING LU received the B.S. degree in mechanical engineering from Shandong University, in 2016. She is currently pursuing the Ph.D. degree with the Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, China. Her research interests include optical mechanical structure design and remote sensing.

JIARONG WANG was born in Changchun, Jilin, China, in 1989. She received the B.S. degree in optical engineering from the Changchun University of Science and Technology and the M.S. degree in circuits and systems from Jilin University. She is currently pursuing the Ph.D. degree with the Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, China. Her research interests include 2D and 3D object detections and stereo vision.

WEN GAO is an Associate Research Fellow with the Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences. Her research interests include digital image processing, television tracking, and automatic target recognition technology.

HUA WEI received the B.S. degree in automation from Shandong University. She is currently pursuing the Ph.D. degree with the Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, China. Her research interests include object detection and fine-grained recognition.