IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 12, 2019, accepted August 22, 2019, date of publication August 26, 2019, date of current version September 9, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2937676

SCNet: Subdivision Coding Network for Object Detection Based on 3D Point Cloud

ZHIYU WANG[1], HAO FU[1], LI WANG[1], LIANG XIAO[2], AND BIN DAI[1,2]
[1]College of Intelligence Science and Technology, National University of Defense Technology, Changsha 410073, China
[2]Unmanned Systems Research Center, National Innovation Institute of Defense Technology, Beijing 100071, China

Corresponding author: Bin Dai (bindai.cs@gmail.com)

This work was supported by the National Natural Science Foundation of China under Grant 61790565 and Grant 61803380.

ABSTRACT High-precision real-time 3D object detection based on the LiDAR point cloud is an important task for autonomous driving. Most existing methods utilize grid-based convolutional networks to handle sparse and cluttered point clouds. However, the performance of object detection is limited by the coarse grid quantization and expensive computational cost. In this paper, we propose a more efficient representation of 3D point clouds and propose SCNet, a single-stage, end-to-end 3D subdivision coding network that learns finer feature representations for vertical grids. SCNet divides each grid into smaller sub-grids to preserve more point cloud information and converts points in the grid to a uniform feature representation through 2D convolutional neural networks. The 3D point cloud is encoded as the fine 2D sub-grid representation, which helps to reduce the computational cost. We validate our SCNet on the KITTI object benchmark in which we show that the proposed object detector produces state-of-the-art results with more than 20 FPS.

INDEX TERMS Autonomous driving, 3D object detection, point rearrangement, sub-grid.

I. INTRODUCTION

Recently, great progress has been made on object detection for autonomous driving. The unmanned vehicle relies on the environmental perception system to obtain external information in which the LiDAR and camera are the major sensors. The image from the camera can provide rich information with textures and object shapes in 2D [1]. Thus, a lot of works have been done based on image, especially based on the convolutional neural network (CNN) [2], [3]. However, it is difficult to make an accurate 3D estimation from an image at long distance, no matter it is monocular or stereo image [4]. Compared with the 2D image, LiDAR could generate a high-precision 3D point cloud of the environment, which makes it another commonly used sensor for autonomous driving.

In processing the point cloud, one has to deal with the sparsity and the disordering of the LiDAR points. To tackle these challenges, many approaches have been proposed. Traditionally, object detection based on the LiDAR point cloud is divided into three steps: the removal of the ground plane, the bottom-up clustering, and the followed classification step [5]. These three steps are usually performed sequentially, which may cause the errors produced in the previous steps, especially the over-segmentation and under-segmentation errors, propagated to the final classification step. Recently, deep learning approaches have become popular for object detection [6]–[9]. Since the convolution operator in deep learning requires the input data to have a regular format, the point cloud is usually converted to 3D voxel grids. Besides the 3D voxel grid representation, the MaxPooling operator could also transform the unordered point cloud into regular a format. This novel idea is firstly proposed in Qi *et al.* [10]. However, this approach requires that the point cloud has already been segmented into potential objects.

In the 3D voxel representation, some features, such as the occupancy state or the hand-crafted statistics, are extracted for each 3D voxel [11], [12]. 3D CNN could then be utilized to learn the efficient feature representation. However, there is too much redundancy in the 3D CNN as the point cloud is usually very sparse. 3D CNN is also very computationally expensive and hard to run in real-time [11], [13].

Another form of point representation is the 2D planar projection which projects points to a plane and then converts them to discretized grids. The hand-crafted or statistical feature of the 2D grid is similar to the pixel feature for the 2D image. Therefore, 2D CNN commonly used in the image processing area can be directly applied to the 2D grid projection image. Typically, there are two forms of 2D grid

The associate editor coordinating the review of this article and approving it for publication was Sudipta Roy.

projection: the range image projection and the bird's eye view projection. In this paper, we focus on the bird's eye view image.

There has been a large literature on utilizing deep learning approaches to learn efficient features for point cloud data. VoxelNet [14] is an end-to-end trainable deep network which divides point cloud into 3D voxels and uses PointNet [10] to learn voxel features. Since the VoxelNet is performed in 3D space, it can only run at 4.4 Hz and could not meet the real-time requirement. To improve the computational efficiency of VoxelNet, SECOND [15] adopts sparse convolution to accelerate the 3D convolutional speed. Following these methods, a simplified method named PointPillars [16] is proposed recently. PointPillars projects the point cloud into 2D grids and then uses PointNet to learn the pillar features.

In this paper, we propose a novel method called SCNet, which performs 3D object detection based on the 2D convolutional network. Due to the limited grid resolution, a lot of information might get lost in the grid quantization stage. To overcome this, we propose a subdivision coding method, which assigns the points organized in grids to smaller sub-grids. SCNet is a single-stage object detector that relies on an efficient end-to-end network to extract sub-grid features and output the final object detection results. The grid coding of the bird's eye view ensures that we can perform 2D convolutions efficiently and utilize the priors of object shape and size. Unlike traditional hand-crafted features, the learned grid feature captures more information about the original point cloud. Inspired by the slice segmentation model [17], the proposed network architecture uses a novel grid coding method that can reduce the feature dimension. The finer structure ensures that most points can be arranged in the regular format and preserve most of the 3D structure information. Since all the operations in SCNet are performed in 2D, the inference time is much faster than those approaches that rely on 3D convolution.

We validate the effectiveness of SCNet on the KITTI dataset [18]. Results show that the performance of our SCNet is comparable to the state-of-the-art methods while being much faster than existing approaches.

The main contributions of this paper are summarized as follows:

- We propose a novel end-to-end neural network architecture for 3D object detection based on the point cloud.
- We propose a feature learning network that learns representative features by rearranging the points in a finer structure and thus overcomes the sparsity of the point cloud.
- We evaluate our SCNet on the KITTI benchmark and show that the results of our method are comparable to the state-of-the-art methods while being much faster.

II. RELATED WORK

Plenty of works have been done to find the appropriate representation of the 3D point cloud. Efficient data representation is beneficial for many environmental perception tasks, such as object detection, semantic segmentation, etc. In this section, we briefly review related works on object detection and data representation for the 3D point cloud.

A. OBJECT DETECTION

Before the deep learning era, object detection approaches usually consist of several independent stages such as ground plane removal, object clustering, classification, etc. In these approaches, the errors that occurred in the previous stages will propagate to later stages [19]. With the help of deep learning approaches, all these stages can be learned by the neural network and even be merged into a single stage. Existing deep learning approaches for object detection could be roughly divided into two types: single-stage network and two-stage network.

The two-stage network consists of a region proposal network (RPN) [20] that generates object proposals and a classification network that classify the object proposals into different categories. MV3D [21] extends the 2D RPN to 3D and generates 3D proposals based on the bird's eye view. A deep fusion network is then used to combine the multimodal information to predict object class. AVOD [22] is a typical two-stage network. Its RPN can generate proposals on high-resolution feature map which is important for small object detection. Qi *et al.* [23] proposed a novel two-stage network named F-PointNet. Unlike traditional methods, the data input of the proposal network is different from that of the subsequent detection network. The proposals are generated by projecting the detection results of the image to the 3D frustum. F-PointNets combines the detection results of 2D image and feature extraction of the 3D point cloud to precisely estimate 3D localization. It thus enjoys the merits of both 2D image detection and the classification ability of PointNet.

In contrast to the two-stage network, a single-stage network unifies object classification network and the object proposal network into a single network. Among the single-stage methods, VeloFCN [13] proposed by Li et al. is a 2D end-to-end fully convolutional network that projects point cloud to the range image and then outputs the object confidence score and the bounding boxes. 3D-FCN [24] extends the fully convolutional network to 3D space and performs 3D convolution on voxels. This method could generate more accurate bounding boxes. A great improvement on single-stage architecture is made by Lin *et al.* [25]. They propose a new loss function named focal loss to overcome the imbalance between positive samples and negative ones. It proves that single-stage architecture could achieve better performance than two-stage methods in terms of efficiency and accuracy. PIXOR [26] is another interesting single-stage network in which all components are specifically designed to achieve a balance between efficiency and accuracy.

B. FEATURE REPRESENTATION OF POINT CLOUD

The intrinsic difference between the 2D image and 3D point cloud requires designing specific data representation of point

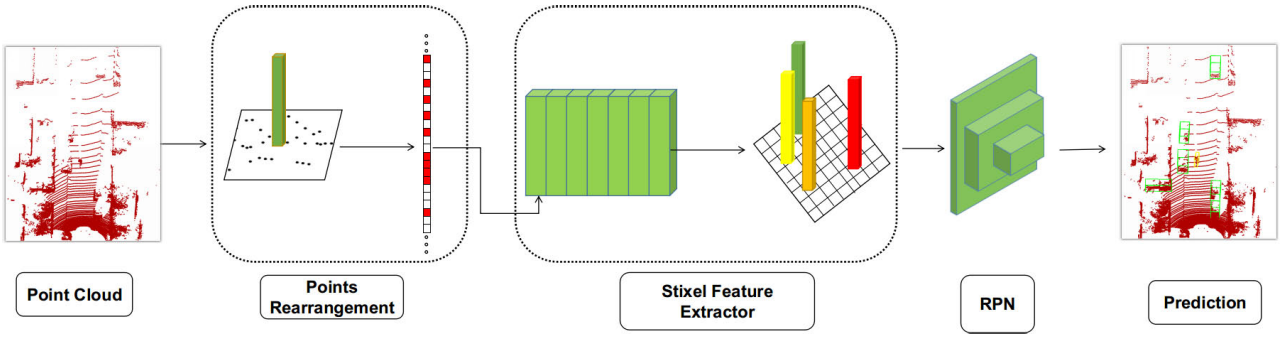


FIGURE 1. SCNet Architecture. SCNet consists of a point rearrangement stage, a grid feature extractor and a region proposal network. This network takes one frame raw point data as input, then points rearrangement stage assigns points to the corresponding grids and arranges the points in smaller sub-grids. The grid feature extractor stacks the non-empty grids and learns the general features. Later, the grid feature is assigned back to a sparse feature map. Finally, the RPN extracts high-level features and outputs the object classification and 3D bounding boxes.

cloud when extending the approaches from 2D to 3D. Earlier methods utilize hand-crafted features to represent the point cloud. For a segmented object, a variety of object-level features designed by researchers are used to represent the object. However, these features are usually unable to capture the complex object attributes. Several methods attempt to convert the point cloud to grids or voxels and then extract hand-crafted features in each grid cell to represent the point cloud [27]–[30]. MV3D [21] encodes the BEV grids by height, intensity, density and encodes the frontal view grids with height, distance and intensity. In an extreme case, 3D-FCN [24] directly use the occupancy state as grid features.

As learning-based approaches have become more popular, some recent approaches try to learn the efficient representation of the point cloud. Qi *et al.* [10] proposed a network called PointNet in which they designed a network to extract point-wise features. These features could then be used for object classification, part segmentation and scene semantic understanding. They further proposed an improved version named PointNet++ [31] to learn local features. VoxelNet [14] builds a new end-to-end network that replaces the hand-crafted channel features with learned features obtained from the PointNet. Later, SECOND [15] makes a series of improvements based on VoxelNet and achieves better performance in terms of accuracy and efficiency. Recently, Lang *et al.* [16] extends the VoxelNet further by projecting point cloud to pillars instead of voxels. The pillar features can be learned by PointNet and all the operations involved are 2D convolutions which ensures a faster runtime. Since the pillar features learned by PointNet ignores the vertical structure information, it is hard to make full use of the 3D information, especially along the vertical direction.

In this paper, we propose subdivision coding network (SCNet), a single-stage end-to-end neural network. We rearrange the point cloud and extract features in a more efficient way. Our feature extractor clusters points into grids and arranges these points into smaller sub-grids. We then use 2D CNN to learn grid features without any pooling operation. Due to the sub-division coding stage, the learned features could capture more 3D information than existing approaches.

III. SCNET

In this section, we will introduce SCNet which outputs accurate classification and 3D bounding boxes based on the LiDAR point cloud.

A. SCNET ARCHITECTURE

SCNet is a single-stage end-to-end trainable network. As shown in Fig. 1, SCNet consists of three blocks: (1) a point rearrangement block that divides the points into grids and assign points to smaller sub-grids, (2) a grid feature extractor that use 2D CNN to extract grid features, and (3) a region proposal network that converts grid feature map to high-level feature map and then outputs the labels and bounding boxes of the objects.

1) POINTS REARRANGEMENT

The point cloud is represented in the typical LiDAR coordinate system where x-axis points to the forward, y-axis points to the left and z-axis points upward. Given one frame of point cloud data with range (D, H, W) along axes (z, y, x), we divide the X-Y plane into equal 2D grids and all of the points are projected into grids. Defining the height and width of the grid as S_H, S_W, we group the points in each grid and count the number of points in it. Due to the sparsity of the point cloud, the number of points in each grid is not uniform. Nearer grids usually contain more points than further ones. A natural thought is to sample a fixed number of points [14] for each grid to overcome the imbalance of point distribution. However, it will cause an information loss as some points are discarded.

For each grid, we split it into different slices along the z axis. Each slice is then divided into smaller 2D sub-grids in the X-Y plane. The points in each grid are assigned to associated sub-grids. The slice is then encoded along x and y axes, respectively. To encode in the x-direction, all points in grids with the same x coordinate are collected to calculate

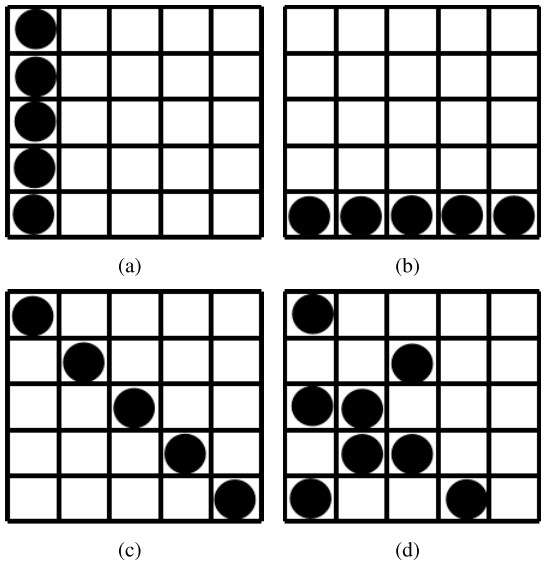


FIGURE 2. Several typical distributions of points in a grid slice.

TABLE 1. The corresponding encoding of the point distribution in Fig. 2.

Type	X	Y
(a)	$5-0-0-0-0$	$1-1-1-1-1$
(b)	$1-1-1-1-1$	$5-0-0-0-0$
(c)	$1-1-1-1-1$	$1-1-1-1-1$
(d)	$3-2-2-1-0$	$2-2-2-1-1$

the density. The same procedure is also performed along the y-direction. As shown in Fig. 2 and Table 1, there are several typical point distributions. In Fig. 2(a), the points are distributed along the y axis, so the encoding along y is uniform. In Fig. 2(b), there is a peak along the y-axis. And in (c), both directions are encoded uniformly. Fig. 2(d) presents an example of point cloud distribution in real scenario. This encoding method reduces the feature dimension while simultaneously retains most information of the original point cloud. In the traditional grid-based coding method, the feature dimension grows quadratically with the number of grids. However, the feature dimension of our coding method only grows linearly.

Specifically, each grid is divided into $N_x \times N_y \times N_z$, where N_x, N_y and N_z are the smaller sub-grid numbers of the grid divided along each axis. It is worth mentioning that the encoding sub-grid sizes in the x and y directions could be different. For x-direction, the encoding sub-grid size is $1 \times N_y$, and for the y-direction, the grid size is $N_x \times 1$. For each encoding sub-grid, we randomly sample a point as the feature representation and combine it with the grid density. It is reasonable to sample only one point in the encoding sub-grid because adjacent points have similar properties. This arrangement may overcome the shortcomings of sampling a fixed number of points and makes full use of the raw data by considering effective spatial distribution.

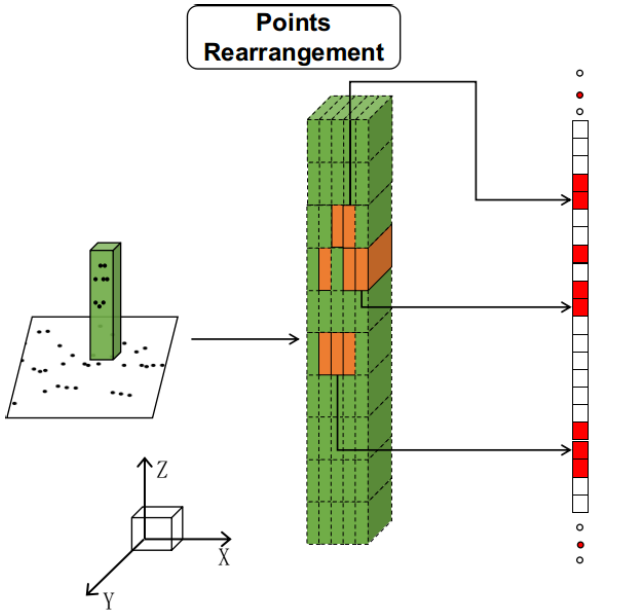


FIGURE 3. The points rearrangement. First, we divide the space in the X-Y plane into equal 2D grids and assign points to the corresponding grids. For each grid, we split it into different slices along the z axis. Each slice is then divided into smaller 2D sub-grids in the X-Y plane. The sub-grid is encoded along the X and Y axes, respectively. Finally, according to its relative position, the encoding sub-grids are rearranged into 2D vectors. For better visualization, only the x-direction encoding process is shown.

Let $G = [x, y, z, r, n]^T$ represents a non-empty encoding sub-grid, where x, y, z are the coordinates of a randomly sampled point in the grid, r is the reflectivity, and n is the density of the encoding sub-grid.

In Fig. 3, we illustrate the x-direction point arrangement process. Define a non-empty grid S as an array with size $N \times C$, where N (equals to $N_x \times N_z$) is the size of the grid feature and C is the feature dimension of the encoding sub-grid. Since we have encoded each slice along x axes, there are N_x encodings per slice and N_z slices per grid. Then, according to its relative position I_i, the feature of each encoding sub-grid is arranged by the following formula:

$$I_i = \lfloor [x_i - x_{si}]/stride_{gx} \rfloor + N_x \times \lfloor z_i/stride_{gz} \rfloor \qquad (1)$$

where $\lfloor\ \rfloor$ means the round down operator and x_i, z_i denote the X, Z coordinates of the point for the i-th encoding sub-grid. The x_{si} is the X coordinate of the corresponding grid. $stride_{gx}$ and $stride_{gz}$ represent the grid resolution.

The y-direction encoding procedure is the same as x-direction. The two codes are then merged into the final grid representation.

We assign a fixed buffer of size $(N_x + N_y) \times N_z$ for each grid. All point clouds are arranged orderly with the proposed method.

2) GRID FEATURE EXTRACTION

An overview of the grid feature extractor is shown in Fig. 4. All non-empty grid features are stored together, forming a $K \times N \times C$ tensor, where K is the number of non-empty

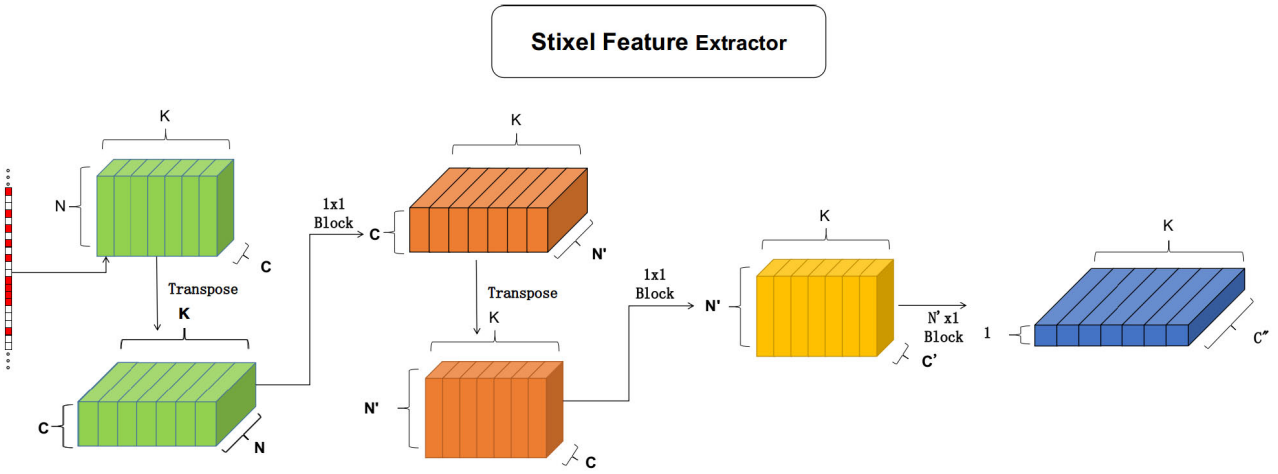


FIGURE 4. Grid Feature Extractor. All non-empty grids are gathered together to form a new tensor of size $K \times N \times C$. By transposing the tensor, the network can use depth-wise convolutions to learn the grid feature through 1×1 convolution blocks. First, the features of each channel are learned. Then, the channel features are fused by 1×1 convolution. Finally, grid features are integrated into the tensor of size $K \times 1 \times C''$.

grids, N is the number of encoding sub-grids, and C is the feature dimension of each encoding sub-grid. The proportion of non-empty grids is less than 10%, which suggests that the computational cost is significantly reduced. Similar to Xception [32] and MobileNets [33], we also introduce a depth-wise separable convolution operator into our grid feature extraction framework. The depth-wise separable convolution performs spatial convolution in separate feature channel spaces to optimize the network latency.

We use 2D CNN to simplify the calculation process and reduce the computation. We firstly transpose the tensor from size $K \times N \times C$ to $K \times C \times N$, and then we adopt several 1×1 blocks to extract features of each channel. The 1×1 block consists of a 1×1 convolutional layer, a batch normalization (BatchNorm) layer and a rectified linear unit (ReLU) layer. After the 1×1 block, the size of the tensor becomes $K \times C \times N'$. This tensor is then transformed back to $K \times N' \times C$. Finally, through the learning of $1 \times N'$ blocks, the final grid features are generated.

Due to the sparsity of the point cloud, the occupied grids only account for less than 10% of all the grids. Therefore, a lot of computational loads have been saved during the forward and backward propagation. All grids share the same learning parameters, which makes it possible to extract features that are generic and consistent. Each feature is associated with a spatial coordinate of a non-empty grid. We can assign them to the relevant spatial coordinates to form a sparse tensor of size $C'' \times H \times W$. The learned feature is then used in the subsequent object detection stage.

3) REGION PROPOSAL NETWORK

Region proposal network has been widely used in object detection tasks [34]. In this paper, we use a one-stage RPN similar to single shot multibox detector (SSD) [35] architecture for object classification and localization. The input of the network is the sparse tensor obtained from the grid feature extractor. Convolutional Neural Networks are mainly composed of convolutional layers and pooling layers. The grid-wise features are learned by the convolutional layers while the pooling layers are used to extract more stable features and downsample the feature map to save computation. As the network gets deeper, the receptive field becomes larger and the resolution becomes lower. Lower resolutions may result in inaccurate localization and classification of small objects. In order to maintain sufficient feature representation and resolution, as shown in Fig. 5, we designed a feature extraction network that fuses the results from different resolution output. Features for each resolution are obtained by convolution, pooling, and upsampling operations. Features obtained from three different resolutions are then concatenated to construct the final feature map. A 1×1 convolutional layer is then applied to output the probability score and the position of the object.

B. LOSS

In the networks, we use the multi-task loss to learn the network parameters. It consists of three different kinds of loss functions. The classification loss is used to identify the category of the object. the regression loss is used to learn the shape of the object. We also introduce a direction loss as proposed in SECOND [15] to identify the direction of the object which is a special case of the regression loss.

By combining three loss functions, the total loss is defined as:

$$L_{total} = \beta_{reg} L_{reg} + \beta_{dir} L_{dir} + \beta_{cls} L_{cls} \tag{2}$$

where loss weights β_{reg}, β_{dir}, β_{cls} are empirically set to 2.0, 0.2, 1.0. For the setting of the hyperparameters, we refer to the setting of SECOND and adjust them based on it. This is a multi-task problem. The weights of the three tasks are

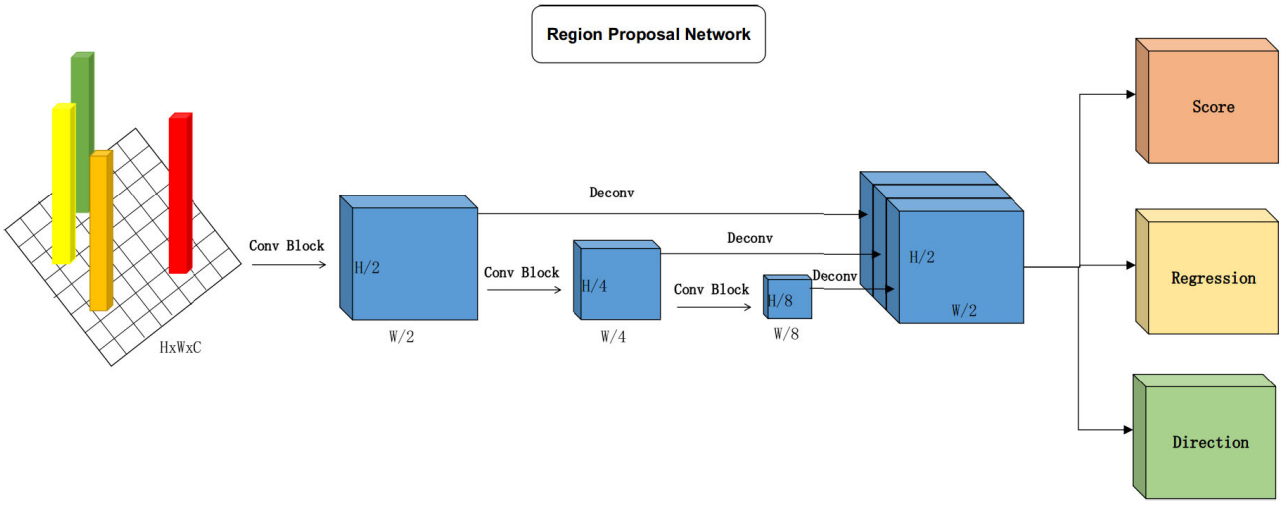


FIGURE 5. Region Proposal Network. All grid features are assigned to the associated spatial coordinates to form a sparse tensor. We define a conv-block with several convolutional layers, with the first layer with stride 2 and the other layers with stride 1. The final high-level feature map consists of three different resolution parts. The first part downsamples the feature map by half through a conv-block. The second part is downsampled again by a conv-block on the basis of the first part and the third part is downsampled based on the second part. Then, all parts with different resolution capabilities are upsampled to the same size and concatenated to form the final high-level feature map.

mutually influential. When β_{cls} and β_{dir} are fixed, the relative increase of the bounding box regression weight does not affect the performance too much. However, when β_{reg} and β_{dir} are fixed, an increase in the weight of the category prediction can make it difficult for the network to learn appropriate regression parameters. That is to say, the increase of the classification weight will result in a significant performance degradation. In terms of the task difficulty, the bounding box regression task is larger than the category prediction task, and the category prediction task is larger than the direction prediction task, which is the reason for the weight setting.

1) REGRESSION LOSS

Since point clouds capture the real size of the objects, and objects in the same category have nearly the same size, we use fixed-size anchors to model the detected objects. Let $B_a = \{x_a, y_a, z_a, w_a, l_a, h_a, \theta_a\}$ represent the detected 3D object bounding box, where x_a, y_a, z_a are the center coordinates, w_a, l_a, h_a represent the width, length and height of the object, θ_a is the yaw angle around Z-axis. The ground-truth object is also parameterized as $B_g = \{x_g, y_g, z_g, w_g, l_g, h_g, \theta_g\}$. In the regression branch of RPN, we define the regression target as $\{\delta x, \delta y, \delta z, \delta w, \delta l, \delta h, \delta\theta\}$:

$$\delta x = \frac{x_g - x_a}{d_a}, \delta y = \frac{y_g - y_a}{d_a}, \delta z = \frac{z_g - z_a}{h_a}, \tag{3}$$

$$\delta w = log(\frac{w_g}{w_a}), \delta l = log(\frac{l_g}{l_a}), \delta h = log(\frac{h_g}{h_a}), \tag{4}$$

$$\delta\theta = sin(\theta_g - \theta_a) \tag{5}$$

where $d_a = \sqrt{(l_a)^2 + (w_a)^2}$ is used to normalized the width and length of the object.

We use Smooth L1 loss for 3D box regression:

$$L_{reg} = \sum_{\delta b \in (\delta x, \delta y, \delta z, \delta w, \delta l, \delta h, \delta\theta)} SmoothL1(\delta b) \tag{6}$$

As pointed out in the SECOND [15], the direction of the object is a noteworthy attribute that is important for predicting subsequent movements. Since $\delta\theta$ cannot distinguish the flip of the object, we add another direction classifier. The direction classifier uses a softmax loss function. When the yaw angle is higher than 0, it is considered positive, otherwise, it is negative.

2) CLASSIFICATION LOSS

The network may generate tens of thousands of anchors, and only a few dozen of them are positive. For a single-stage detection network, the foreground-background imbalance problem must be addressed when training the detector. The analysis in RetinaNet [25] shows that the failure of training mostly comes from the easy negative samples. The authors of RetinaNet introduced a new loss function called Focal Loss [25] to solve the single-stage training problem. Following RetinaNet, we use the focal loss to handle the class imbalance problem. The classification loss is formulated as:

$$L_{cls} = -\alpha_t(1 - p_t)^{\gamma} log(p_t) \tag{7}$$

where α and γ are the hyper-parameters of the focal loss. p_t is the probability score of the RPN classification output.

IV. IMPLEMENTATION

In this section, we introduce the implementation details of the network.

A. NETWORK ARCHITECTURE

In the grid feature extraction network, we first learn the channel features using two 1×1 blocks with Input/Output size of (100, 64) and (64, 16). In the second step, the relationship between channels is learned through two 1×1 blocks with (5, 32) and (32, 8). Then a 16×1 block with (8, 128) is used to output the grid feature. Finally, after projecting each grid feature back to the corresponding position on the x-y plane, we get a sparse tensor of size (B, 128, H, W), where B is the input batchsize, H and W are the height and width.

In our RPN, we concatenate features from different resolutions to build more expressive features. Three consecutive conv-blocks convert the input tensor into feature maps of different sizes. Then, each feature map is upsampled to obtain a uniform tensor and then merged into the final output. The three conv-blocks contain (3, 5, 5) convolution layers. For car detection, the first strides of the three blocks are (2, 2, 2). For pedestrian and cyclist detection, the first strides of the three blocks are (1, 2, 2). The strides of other layers are all set to 1. The upsampling layer strides are (1, 2, 4).

1) CAR DETECTION

In the KITTI dataset, since the annotations only exist in the frontal view, we only processed the point cloud within $[-3, 1] \times [-40, 40] \times [0, 70.4]$ meters on the Z, Y and X axes. When projected onto the image plane, all the points outside the image boundary are removed. We set the grid size to $S_H = 0.2$ and $S_W = 0.2$ meter, the input feature map then has a size of $H = 400$ and $W = 352$. In each grid, we define the $stride_{gx}$, $stride_{gy}$, $stride_{gz}$ as 0.04, 0.04, 0.4 meters, and $range_z$ is set to 10. The encoding sub-grids in the grid are arranged as $(5 + 5) \times 10$ and then concatenated to a 100-dimensional vector. $(5 + 5)$ represents a combination of 5 x-direction codes and 5 y-direction codes.

For car detection, we use fixed-size anchors. w_a, l_a, h_a are set to 1.6, 3.9, 1.56 meters respectively, and the center of the car in Z-direction is set to -1 meter. The rotation parameters around the z-axis contain two values: 0° and 90°.

When defining the positive and negative samples, we consider intersection-over-union (IOU) [36] as the criteria. If the IOU between the anchor and the ground truth is above 0.6, the anchor is regarded as a positive sample. It is considered a negative sample when the IOU is less than 0.45. During the training, we ignore all anchors with the IOU between 0.45 and 0.6.

2) PEDESTRIAN AND CYCLIST DETECTION

Since pedestrian and cyclists have fewer points than cars, we only detect objects within $[-2.5, 0.5] \times [-20, 20] \times [0, 48]$ meters on the Z, Y and X axes. The size of the grid is also set to $S_H = 0.2$ and $S_W = 0.2$ meter, and the input feature map has a size of $H = 200$ and $W = 240$. In each grid, we define the $stride_{gx}$, $stride_{gy}$, $stride_{gz}$ as 0.04, 0.04, 0.3 meters, so the $range_z$ is 10. The encoding sub-grids in grid are arranged as $(5 + 5) \times 10$ and then concatenated to a 100-dimensional vector.

For pedestrian detection, the anchor dimensions are $w_a = 0.6, l_a = 0.8, h_a = 1.73$ meters, and the center of the pedestrian in Z-direction is set to -0.6 meters depending on the location of the LiDAR relative to the ground plane. For the cyclist detection, the cyclist anchor has width, length and height of $w_a = 0.6, l_a = 1.76, h_a = 1.73$ meters and the height center is -0.6 meters. The rotation of the pedestrian and cyclist anchors around z-axis contains two values: 0° and 90°. The positive and negative IOU thresholds are set to 0.5 and 0.35, respectively.

3) HYPER-PARAMETERS

In the training phase, we use stochastic gradient descent (SGD). The initial learning rate is 0.0002, and the decay weight is set to 0.8 per 15 periods. We set the batch size to 3 based on the graphics memory. The parameters for focal loss are set to $\alpha_t = 0.25$ and $\gamma = 2$.

B. DATA AUGMENTATION

Our training dataset only contains less than 4000 frames, which makes it easy to overfit during training. It is, therefore, necessary to introduce the data augmentation strategy to increase the variants of the object. We use three different ways for data augmentation.

Firstly, as introduced in SECOND [15], in order to get more object labels during the training phase, we collect all the ground truth labels in the training data set. The ground truth consists of object labels and object points. In each training step, we randomly add some labeled objects to the original point cloud. Those manually added positive objects could ease the problem of imbalanced training loss. We also perform a collision test to ensure that there is no overlap between the added objects and existing objects..

Secondly, motivated by VoxelNet [14], we firstly use the uniformly distributed random variable $\delta\theta \in [-\pi/10, \pi/10]$ around the z-axis to rotate the bounding box and the points within the bounding box. Then the bounding box is randomly translated along the X, Y and Z axes. The amount of translation is randomly sampled from a Gaussian distribution with a mean value of 0.25 and a standard deviation of 0.25.

Thirdly, we adjust all point clouds and ground truth boxes to get more samples. We scale the bounding box size and the coordinates of all point clouds using random sampling from a uniform distribution [0.95, 1.05]. The scaling operation allows the learning network to adapt to the scale changes of the object. To simulate the turning of the vehicle, we rotate all ground truths and points around the z-axis. The rotation value is sampled from a uniform distribution in the range of $[-\pi/4, \pi/4]$.

V. EXPERIMENTS

We evaluate the performance of the proposed SCNet on the KITTI-Object benchmark dataset which contains 7481 training data and 7518 testing data. We perform experiments on

TABLE 2. Bird's eye view detection performance in the KITTI validation dataset.

Method	Time	AP-BEV		
		Easy	Moderate	Hard
MV3D [21]	360ms	86.18	77.32	76.33
PIXOR [26]	93ms	86.79	80.75	76.60
VoxelNet [14]	225ms	89.60	84.81	78.57
SECOND [15]	50ms	89.96	87.07	79.66
PointPillars [16]	16ms	90.07	87.44	84.78
SCNet	40ms	**90.35**	**88.09**	**87.30**

TABLE 3. 3D object detection performance in the KITTI validation dataset.

Method	Time	AP-3D		
		Easy	Miderate	Hard
MV3D [21]	360ms	71.19	56.60	55.30
VoxelNet [14]	225ms	81.97	65.46	62.85
SECOND [15]	50ms	87.43	76.48	69.10
PointPillars [16]	16ms	85.22	76.52	69.73
SCNet	40ms	**87.83**	**77.77**	**75.97**

the following three categories: cars, pedestrians and cyclists. For each category, there are three different difficulty levels: easy, moderate and hard. The difficulty levels account for different sizes, occlusions, and truncation, which are key factors influencing the detection results. The original training data is divided into a training subset and a validation subset according to [37], with 3712 and 3769 samples respectively. We evaluate the object detection performance both in the 3D view and the bird's eye view.

We compare our performance with state-of-the-art approaches. Since we do not use the image information, we compare SCNet with algorithms that purely based on point clouds, such as MV3D [21], PIXOR [26], VoxelNet [14], SECOND [15] and PointPillars [16].

A. EVALUATION METRIC

We adopt the KITTI evaluation protocols, where the IOU threshold is set to 0.7. The average accuracy (AP) is calculated as the performance measure. The same threshold is applied both in the bird's eye view and the 3D view. For car detection, we ignore similar categories such as trucks, vans, trams and those labeled as don't care.

B. EVALUATION ON THE VALIDATION SET

Since most of the methods used for comparison do not provide detection results for pedestrians and cyclists, we only compare the performance of car detection on the validation dataset.

1) BIRD'S EYE VIEW (BEV)

The results of BEV evaluation on the validation dataset are shown in Table 2. Since PointPillars [16] does not provide validation results in the article, the results of PointPillars in Table 2 and Table 3 are reproduced by their released code. From the results, it is observed that SCNet proposed in this paper obtains the best AP in all three categories. Its runtime is also faster than most of the comparing approaches.

2) 3D OBJECT DETECTION

The 3D object detection result is shown in Table 3. Unlike BEV object detection tasks that only require 2D bounding boxes of the objects, 3D object detection is a more difficult task. Results in Table 3 indicates that SCNet still performs best among all the competing approaches. Its faster running time relies on its avoidance of 3D convolution.

C. EVALUATION ON THE TESTING SET

To compare our method to other methods in the official KITTI website, we evaluate our method by submitting the results on the testing set to the official server. The results of BEV and 3D are shown in Table 4 and Table 5. Our method is still among the best performing approaches. In particular, our results are superior to VoxelNet [14] and SECOND [15] which have similar loss function and region proposal networks. For smaller objects such as pedestrian and cyclist, our results are slightly worse than the PointPillar approach but still better than the other approaches.

D. ANALYSIS OF RESULTS

All experiments were run on a computer with a GTX 1080Ti GPU and an Intel i7 CPU@3GHZ. We present several

TABLE 4. Bird's eye view detection performance on the KITTI testing dataset.

Method	Time	Car			Pedestrian			Cyclist		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
MV3D [21]	360ms	85.82	77.00	68.94	N/A	N/A	N/A	N/A	N/A	N/A
PIXOR [26]	93ms	81.70	77.05	72.95	N/A	N/A	N/A	N/A	N/A	N/A
VoxelNet [14]	225ms	**89.35**	79.26	77.39	46.13	40.74	38.11	66.70	54.76	50.55
SECOND [15]	50ms	88.07	79.37	77.95	55.10	46.27	**44.76**	**73.67**	56.04	48.78
SCNet	40ms	88.87	**86.46**	**78.81**	**57.63**	**49.50**	44.20	72.25	**56.93**	**50.88**

TABLE 5. 3D object detection performance on the KITTI testing dataset.

Method	Time	Car			Pedestrian			Cyclist		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
MV3D [21]	360ms	66.77	52.73	51.31	N/A	N/A	N/A	N/A	N/A	N/A
VoxelNet [14]	225ms	77.47	65.11	57.73	39.48	33.69	31.5	61.22	48.36	44.37
SECOND [15]	50ms	**83.13**	73.66	66.20	**51.07**	**42.56**	37.29	**70.51**	**53.85**	46.90
SCNet	40ms	82.84	**73.90**	**66.91**	49.32	41.54	**39.52**	67.66	51.06	**47.54**

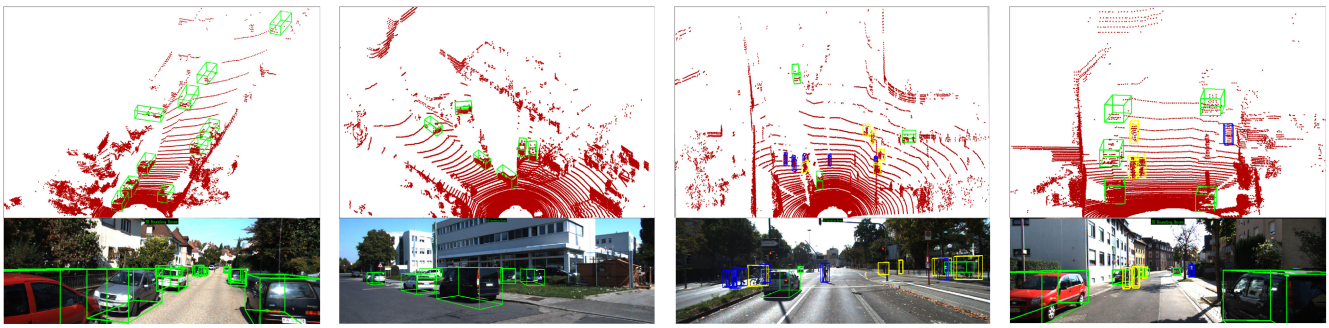
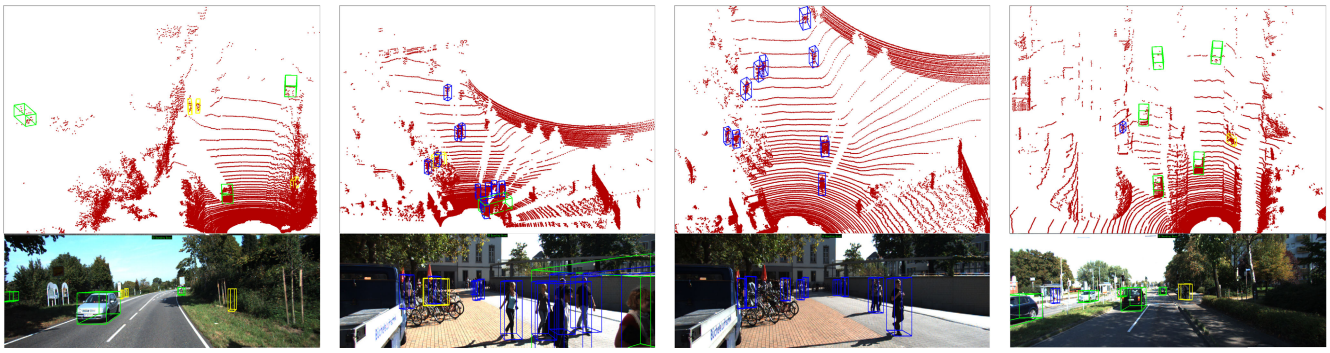


FIGURE 6. Results of 3D object detection on the KITTI validation dataset. Our results are only from LiDAR. For better visualization, the 3D bounding boxes are projected onto the images. In each frame of data, the green box indicates the result of the car, the blue box indicates the pedestrian and the yellow box indicates the cyclist.



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

FIGURE 7. The failure cases of results on the KITTI validation dataset.

qualitative results on the validation dataset in Fig. 6 and Fig. 7. For better visualization, the 3D bounding boxes are projected onto images.

Fig. 6 shows typical object detection results. From the figure, it is observed that our SCNet could produce accurate results for cars of varying difficulty levels. For cars that are far away or heavily occluded, SCNet can still achieve excellent results. Failures in car detection include false negatives and false positives. False negatives mostly occur at further distances. An example of failure is shown in Fig. 7a. Fig. 7b shows another failure case where several nearby pedestrians are detected as a car due to the lack of texture information for LiDAR points.

Since pedestrians and cyclists are smaller than cars, and there are fewer samples in the KITTI dataset, the detection of them is a more challenging task. The results also indicate that the performance of pedestrians and cyclists is worse than that of cars. Besides, due to the smaller size of pedestrians and cyclists, the downsampling operation would cause a more severe information loss. Therefore, pedestrians and cyclists are more likely to be confused with vertical objects such as poles or trees, see Fig. 7a for an example. Nearby pedestrians may be misclassified as a cyclist (as shown in Fig. 7b). Pedestrians and cyclists tend to get together, so the mutual occlusion between them can cause some false negatives (see Fig. 7d), and two nearby pedestrians may be treated as one, as shown in Fig. 7c.

TABLE 6. Performance of the object detection for different grid features in the KITTI validation dataset.

Feature	AP-BEV			AP-3D		
	Easy	Moderate	Hard	Easy	Moderate	Hard
$f1$	**90.35**	**88.09**	**87.30**	**87.83**	**77.77**	**75.97**
$f2$	90.18	87.26	84.36	87.15	76.91	70.31
$f3$	90.01	86.78	83.98	87.08	77.16	72.53

TABLE 7. Performance of the object detection with different grid representations in the KITTI validation dataset.

Method	AP-BEV			AP-3D		
	Easy	Moderate	Hard	Easy	Moderate	Hard
DA-baseline	90.21	87.42	84.13	80.62	74.93	69.08
BEV-baseline	90.01	86.19	80.78	86.88	76.61	70.68
SCNet	**90.35**	**88.09**	**87.30**	**87.83**	**77.77**	**75.97**

VI. ABLATION STUDY

A. GRID FEATURE REPRESENTATION

We compare different combinations of features to evaluate the performance of object detection. Some basic properties of the grid include the absolute coordinates $(x,\ y,\ z)$, the relative coordinates $(\delta x,\ \delta y,\ \delta z)$, the reflectivity r and the number of points n in each grid. In related works, PointNet++ [31] only uses relative coordinates as the input feature, whilst VoxelNet [14], SECOND [15] and PointPillars [16] use a combination of relative and absolute coordinates. In this section, we construct three different feature combinations and analyze their influence on object detection. These feature combinations include $f_1 = [x,\ y,\ z,\ r,\ n]^T$, $f_2 = [\delta x,\ \delta y,\ \delta z,\ r,\ n]^T$ and $f_3 = [x,\ y,\ z,\ \delta x,\ \delta y,\ \delta z,\ r,\ n]^T$.

The object detection results are listed in Table 6. From the results, it is observed that the relative coordinates do not play an important role in our object detection tasks. The likely reason is that our subdivision encoding structure has contained the relative position information. Therefore, the usage of the relative coordinates will not introduce any performance boost. In addition, if we combine the relative and absolute coordinates together, the performance may even be deteriorated, as the redundant features make it difficult for the network to extract meaningful features.

Based on the results, we choose the absolute coordinates $f_1 = [x,\ y,\ z,\ r,\ n]^T$ as the input feature for our SCNet.

B. GRID REPRESENTATION

To evaluate the effectiveness of our network, we compare some different grid feature representations. Since the network only performs 2D convolution operations on the bird's eye view, we only need to construct point data representations on the 2D X-Y plane.

We implemented two types of baselines, named as a bird's eye baseline (BEV-baseline) and a disordered arrangement baseline (DA-baseline). Similar to the MV3D [21] structure, we replaced the grid feature with hand-crafted features in a bird's eye view. To get a more detailed height representation, we increased the number of height slices to 20, thus producing a 20-dimensional height feature. We also include the reflectance and the number of points in the grid as two other features. The grid encoding of DA-baseline is the same as SCNet, but the encoding sub-grids are randomly arranged. We set the sub-grid resolution of SCNet to $(5 + 5) \times 10$. To make a fair comparison, the RPNs of SCNet and the two baselines are kept the same.

As shown in Table 7, our SCNet shows a better performance than the BEV-baseline and the DA-baseline in both BEV and 3D detection. The BEV detection results indicate that hand-crafted features cannot fully express the data information compared to the learning features. The relatively poor performance of DA-baseline for 3D detection suggests the importance of point arrangement.

C. GRID FEATURE EXTRACTOR

In this section, we have designed several different structures for the grid feature extractor. All structures are based on the grid size of $(5 + 5) \times 10$. We consider four different network structures, as shown in Fig. 8.

It is worth mentioning that the input to the grid feature extractor is a tensor of size $(K,\ N,\ C)$, where K is the number of non-empty grids, N is the size of the encoding sub-grid, and C is the feature dimension.

The first network structure is similar to the voxel feature encoding (VFE) layer [14], which we called grid feature encoding (GFE). A GFE layer takes all the points in each grid as input and then constructs a fully connected network

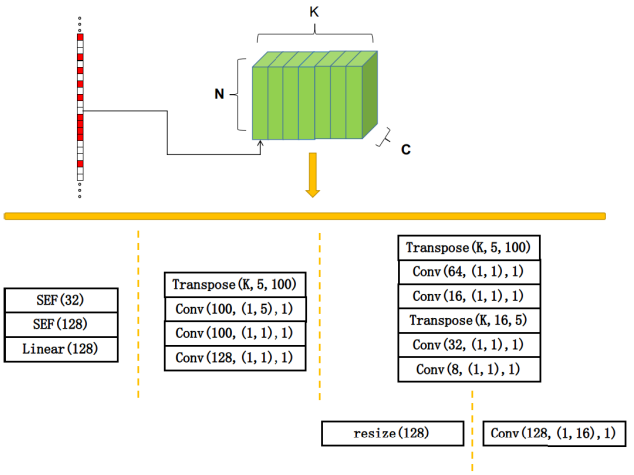


FIGURE 8. Four network structures for grid feature extraction.

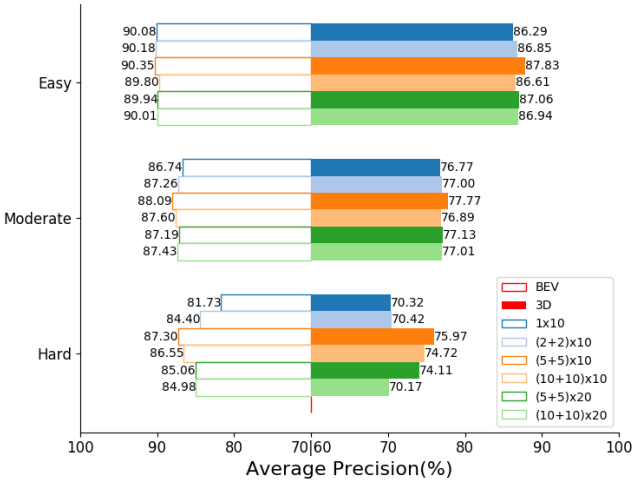


FIGURE 9. The comparison of different grid structures.

(FCN) to extract pointwise features. Each FCN contains a linear layer, a BatchNorm layer and a ReLU layer. It then uses the MaxPooling operation to extract the global feature in each grid. The GFE then tiles the learned feature and concatenates them with pointwise features. The first structure consists of GFE(32), GFE(128) and a linear(128) layer. The GFE(M) and linear(M) denote the blocks that convert the input to M-dimensional output.

The second structure is a traditional convolution operation. The Conv-block consists of a 2D CNN, a BatchNorm and a Relu layer. *Conv*(M, k, s) represents a network operation block with output dimension of M, the convolution kernel of k, and the stride size of s. The second structure contains three blocks: *Conv*(100, (1, 5), 1), *Conv*(100 , 1, 1), *Conv*(128, 1, 1). This network structure combines channel features and structural features.

The third structure separates the channel features from the structural features as proposed in Xception [32]. The input tensor is firstly transposed to (K, C, N). The structural information is considered as the channel features. Two Conv-blocks *Conv*(64, 1, 1) and *Conv*(16, 1, 1) are used to learn different structural features. The output tensor is then transposed back to (K, N', C). Two Conv-blocks with (32, 1, 1) and (8, 1, 1) are used to learn the combination of different features. Finally, the output size of the network is resized to $K \times 1 \times 128$.

The fourth structure has the same basic network as the third structure but replaces the final resizing operation with the Conv-block(128, (1, 16), 1).

As shown in Table 8, the average precision of the BEV detection performance of the four network structures is quite similar. It suggests that the learned features on the X-Y plane are sufficient to represent the object. However, for 3D object detection tasks, there is a significant gap between the performance of different structures. The GFE with the MaxPooling operation is not sufficient to extract structural features, while Xception with resizing operation also loses the structural information due to the resizing operation. The traditional CNN mixes the features and structure of the grid, but this kind of mixed learning still couldn't get a satisfactory performance. Experiments show that Xception with conv-block structure performs best among the four network structures.

D. GRID STRUCTURE

We designed a finer grid structure to retain most of the point cloud information. The form of the subdivision grid is critical to the rearrangement of the points. As the LiDAR resolution varies with distance, we adjust the resolution of the encoding sub-grid accordingly. Since the size of the grid is set to 0.2×0.2, we only need to modify the number of grids along the X and Y axes to split the points.

TABLE 8. Performance of the object detection with different feature extraction networks in the KITTI validation dataset.

Method	AP-BEV			AP-3D		
	Easy	Moderate	Hard	Easy	Moderate	Hard
GFE	89.94	87.15	83.44	80.74	75.76	69.76
CNN	89.79	87.19	84.04	86.22	76.57	70.05
Xception(resize)	90.12	87.48	84.97	86.83	76.83	70.16
Xception	**90.35**	**88.09**	**87.30**	**87.83**	**77.77**	**75.97**

TABLE 9. Performance of the object detection with different grid size in the KITTI validation dataset.

Grid Size	AP-BEV			AP-3D		
	Easy	Moderate	Hard	Easy	Moderate	Hard
$(1+1)\times 10$	90.08	86.74	81.73	86.29	76.77	70.32
$(2+2)\times 10$	90.18	87.26	84.40	86.85	77.00	70.42
$(5+5)\times 10$	**90.35**	**88.09**	**87.30**	**87.83**	**77.77**	**75.97**
$(10+10)\times 10$	89.80	87.60	86.55	86.61	76.89	74.72
$(5+5)\times 20$	89.94	87.19	85.06	87.06	77.13	74.11
$(10+10)\times 20$	90.01	87.43	84.98	86.94	77.01	70.17

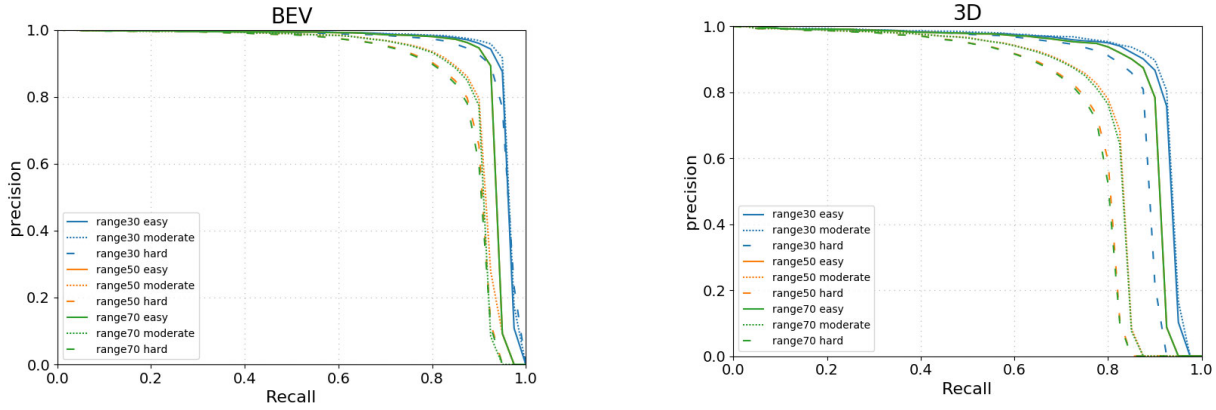


FIGURE 10. Performance at different distances.

TABLE 10. Performance of the object detection at different distance in the KITTI validation dataset.

Difficulty Levels	AP-BEV			AP-3D		
	0-30m	0-50m	0-70m	0-30m	0-50m	0-70m
Easy	90.46	90.35	90.35	88.94	87.83	87.83
Moderate	90.47	88.30	88.09	89.01	77.98	77.77
Hard	89.79	87.40	87.30	87.13	76.09	75.97

The details are shown in Table 9 and Fig. 9. We select different split values for X, Y coordinates to evaluate the performance of various grid sizes. The $(5+5)\times 10$ gets the best results across all the different grid sizes. The subdivision structure is used to rearrange points at a higher resolution. In theory, as the number of grids increases, the detection performance will get better. The experiment shows a similar trend: with the increase of horizontal and vertical resolution, the performance of BEV and 3D detection gets better. It is also observed that grid numbers larger than $(5+5)\times 10$ could not get better performance. We finally set grid number to $(5+5)\times 10$.

E. RANGE OF OBJECT DETECTION

In practical applications of object detection, the effective detection range is an important aspect. It is necessary to evaluate the performance of different ranges. Since pedestrians and cyclists have fewer data at longer distances, we only compare the performance of the car category. We show the Precision-Recall (PR) curves of SCNet in Fig. 10. From the figure, it is observed that at close range, we can get satisfactory performance, and our approach could overcome the truncation and occlusion of objects and achieves similar performance at different levels of difficulty. However, in the further range, the degree of occlusion and truncation can have a larger impact on performance. It is also observed that the detection performance between 50 and 70 meters is quite stable. We also list the average precision in Table 10.

VII. CONCLUSION

In this paper, we present SCNet, a new end-to-end trainable network for object detection. SCNet uses a subdivision

structure to project points onto a higher-resolution grid. The subdivision grid is then encoded in a way that effectively expresses the structural information. Our SCNet has successfully applied 2D convolution operations to capture 3D point information. Experiments on the KITTI dataset show that the proposed network outperforms other state-of-the-art methods. In addition, SCNet also shows remarkable performance in detecting pedestrians and cyclists. Our network structure also ensures its real-time operation on 3D detection tasks.

REFERENCES

[1] 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.*, Jun. 2016, pp. 2147–2156.

[2] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, Jun. 2015, pp. 1440–1448.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[4] 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.

[5] M. Himmelsbach, F. V. Hundelshausen, and H.-J. Wuensche, "Fast segmentation of 3D point clouds for ground vehicles," in *Proc. IEEE Intell. Vehicles Symp.*, Jun. 2010, pp. 560–565.

[6] B. Yang, M. Liang, and R. Urtasun, "HDNET: Exploiting HD maps for 3D object detection," in *Proc. Conf. Robot Learn.*, 2018, pp. 146–155.

[7] 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.

[8] K. Shin, Y. P. Kwon, and M. Tomizuka, "RoarNet: A robust 3D object detection based on region approximation refinement," 2018, *arXiv:1811.03818*. [Online]. Available: https://arxiv.org/abs/1811.03818

[9] M. Simon, S. Milz, K. Amende, and H.-M. Gross, "Complex-YOLO: Real-time 3D object detection on point clouds," 2018, *arXiv:1803.06199*. [Online]. Available: https://arxiv.org/abs/1803.06199

[10] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3d classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2017, pp. 652–660.

[11] D. Z. Wang and I. Posner, "Voting for voting in online point cloud object detection," *Robot., Sci. Syst.*, vol. 1, no. 3, 2015, Art. no. 15607.

[12] 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/Jun. 2017, pp. 1355–1361.

[13] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3D lidar using fully convolutional network," 2016, *arXiv:1608.07916*. [Online]. Available: https://arxiv.org/abs/1608.07916

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

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

[16] 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 Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 12697–12705.

[17] Q. Huang, W. Wang, and U. Neumann, "Recurrent slice networks for 3D segmentation of point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2626–2635.

[18] 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. (CVPR)*, Jun. 2012, pp. 3354–3361.

[19] J. Cheng, Z. Xiang, T. Cao, and J. Liu, "Robust vehicle detection using 3D lidar under complex urban environment," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May/Jun. 2014, pp. 691–696.

[20] 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.

[21] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Proc. IEEE CVPR*, vol. 1, Jun. 2017, pp. 1907–1915.

[22] 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.

[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 *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 918–927.

[24] B. Li, "3D fully convolutional network for vehicle detection in point cloud," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 1513–1518.

[25] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis.*, Jun. 2017, pp. 2980–2988.

[26] 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.

[27] G. Pang and U. Neumann, "Fast and robust multi-view 3D object recognition in point clouds," in *Proc. Int. Conf. 3D Vis.*, Oct. 2015, pp. 171–179.

[28] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view CNNs for object classification on 3D data," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 5648–5656.

[29] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *Proc. IEEE Int. Conf. Comput. Vis.*, Jun. 2015, pp. 945–953.

[30] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1912–1920.

[31] 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. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5099–5108.

[32] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 1251–1258.

[33] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: https://arxiv.org/abs/1704.04861

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

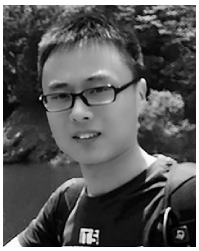
[35] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 21–37.

[36] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010.

[37] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun, "3D object proposals for accurate object class detection," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 424–432.

ZHIYU WANG received the B.S. degree in simulation engineering and the M.S. degree in control science and engineering from the College of Mechatronic Engineering and Automation, National University of Defense Technology, in 2013 and 2016, respectively, where he is currently pursuing the Ph.D. degree in unmanned system institution. His research interests include machine learning and 3-D object detection.

HAO FU received the Ph.D. degree from the University of Nottingham, in 2012. He is currently an Associate Professor with the National University of Defense Technology, Changsha, China. His research interests include computer vision and pattern recognition. He has published several articles in respective fields, including the IEEE TRANSACTIONS ON MEDICAL IMAGING, the European Conference on Computer Vision, and the British Machine Vision Conference.



LI WANG received the B.S. degree in automation from Shandong University, in 2014, and the M.S. degree in control science and engineering from the National University of Defense Technology, in 2016, where he is currently pursuing the Ph.D. degree. His research interests include computer vision, machine learning, and intelligent vehicles.

LIANG XIAO received the Ph.D. degree in control science and engineering from the National University of Defense Technology (NUDT), Changsha, China, in 2017. He is currently a Research Associate with the Unmanned Systems Research Center, National Innovation Institute of Defense Technology (NIIDT), Beijing, China. His research interests include computer vision, machine learning, and robotics.

BIN DAI received the Ph.D. degree in control science and engineering from the National University of Defense Technology, Changsha, China, in 1998. He was a Visiting Scholar with the Intelligent Process Control and Robotics Laboratory, Karlsruhe Institute of Technology, in 2006. He is currently a Professor with the Unmanned Systems Research Center, National Innovation Institute of Defense Technology (NIIDT), Beijing, China. He is also an Adjunct Professor with the National University of Defense Technology. His research interests include pattern recognition, computer vision, and intelligent vehicles.