3D Backbone Network for 3D Object Detection

Xuesong Li[1], Jose E Guivant[1], Ngaiming Kwok[1], Yongzhi Xu[2]
[1]*School of Mechanical and Manufacturing Engineering, University of New South Wales, NSW 2052, Australia*
[2]*School of Civil and Environmental Engineering, University of New South Wales, NSW 2052, Australia*
{*xuesong.li, j.guivant, nmkwok, y.xu*}*@unsw.edu.au*

Keywords: 3D Object Detection, Sparse CNN, Backbone Network

Abstract: The task of detecting 3D objects in point cloud has a pivotal role in many real-world applications. However, 3D object detection performance is behind that of 2D object detection due to the lack of powerful 3D feature extraction methods. In order to address this issue, we propose to build a 3D backbone network to learn rich 3D feature maps by using sparse 3D CNN operations for 3D object detection in point cloud. The 3D backbone network can inherently learn 3D features from almost raw data without compressing point cloud into multiple 2D images and generate rich feature maps for object detection. The sparse 3D CNN takes full advantages of the sparsity in the 3D point cloud to accelerate computation and save memory, which makes the 3D backbone network achievable. Empirical experiments are conducted on the KITTI benchmark and results show that the proposed method can achieve state-of-the-art performance for 3D object detection.

1 INTRODUCTION

Three-dimensional object detection in point cloud can provide accurate spatial location of targets in 3D space and their geometrical shapes, which is vital for various applications such as driverless vehicles and home-assist robots. Even though 2D Convolutional Neural Network (CNN) makes remarkable progress in image-based object detection, 3D CNN is not fully explored for 3D object detection due to the irregular 3D data format and the curse of dimensionality. In our work, we seek to investigate how the 3D CNN can be better applied for object detection.

Point-cloud-based 3D object detection has two main challenges: the irregular data format and expensive 3D computation such as 3D CNN. To address these issues, the irregular data is usually converted into the regular format [28, 3, 15], such as bird view, front view and 3D voxelized grids. Then, cheap and memory-friendly feature extractors, such as 2D CNN based network architecture and shallow 3D CNN architecture, are applied for detection. MV3D [3] generates 3D proposals by operating 2D CNN on projected bird view images. VoxelNet [15] builds 3 layers of 3D CNN to extract 3D features for region proposal network. However, compressing 3D data into 2D or shallow 3D layers make the network fail to extract reliable 3D features for detection. 2D backbone networks, such as VGG [22] and ResNet [9], can extract rich high-level features and are key to the success of 2D CNN based object detection. However, according to our knowledge, there is no corresponding 3D feature extraction backbone network for 3D object detection. To address this, we propose to build a 3D backbone network to extract better 3D features for object detection.

Our 3D backbone network mainly consists of sparse 3D CNN [8, 7] operations instead of traditional 3D ones. Sparse 3D CNN takes advantages of spatial sparsity of point cloud data, which can significantly reduce memory and computational resources for data processing and makes it possible to build complex 3D feature extraction networks. The backbone network includes two kinds: 3DBN-1 and 3DBN-2. The 3DBN-1 only has one forward path from bottom to top, which manages to generate 3D feature maps with different resolutions and abstract levels. In spite of the forward network, 3DBN-2 includes one extra backward path from top to bottom, which passes the high semantic level, but low-resolution, feature map back to feature map with low semantic level, then fuse them together for high-resolution and high semantic feature maps. Before building the 3D backbone, we apply voxelization on point cloud to generate regular 3D grids. After building the backbone network, 2D detection network is used to dilate the sparse 2D feature maps to generate 3D proposals. Except for conventional classification and bounding box regression

tasks, we add one extra classification task which tries to find the heading of predicted boxes. The opposite direction is treated the same in the heading regression task and will not lead to the sudden jumpy in the heading offset of predicted boxes with respect to anchors. We evaluate the proposed method with dataset provided by KITTI benchmark [5] for 3D object detection. Experimental results show that the proposed 3D backbone network is better at extracting 3D features for object detection without increasing significantly computation load, and the 3D backbone network based detection overpass other 3D object detection methods. The source code will be open[1]..

The contributions of this work are:

- proposing two types of 3D backbone network;
- comparing two different approaches to voxelization;
- designing 3D backbone network based detection network;
- using a new method to predict the orientation of the object;
- evaluating the proposed detection framework on KITTI benchmark dataset;

The rest of the paper is organized as follows. Section 2 is an introduction to the related work, followed by Section 3 which illustrates how to build the 3D backbone network with sparse CNN. Other procedures about 3D object detection are presented in Section 4. Experiments of proposed methods are given in Section 5. Section 6 concludes our work and summarizes the contributions.

2 RELATED WORK

2.1 Backbone network

The backbone network represents the main body of the feature extraction network. In the last few years, many backbone networks have been designed for computer vision tasks, such as classification and object detection. AlexNet [12] is the pioneering work to build the deep convolutional neural network including five layers for extracting features, then VGG [23] makes a deeper network of 16 CNN layers by stacking 3×3 convolution operation. Besides, GoogleNet [24] contains a novel inception block to increase feature diversity, and ResNet [9] employs bottleneck mechanism which has the residual sum operation in each stage to increase the deep CNN up to 1202 layers. There are also other backbone networks, such as Xception [4], ResNext [26] and denseNet [10]. They have been the de-facto components for extracting features in 2D object detection, while there is no corresponding 3D backbone network for 3D object detection. In this paper, we propose to build 3D backbone networks by using sparse 3D CNN for feature extraction.

2.2 3D object detection

2.2.1 RGB image based detection

2D object detection has achieved a great success in the RGB image for the last few years, and some researchers [2, 17, 1, 25] have approached to combine the well-developed 2D object detector and prior information of target objects, such as shape, context and occlusion pattern, to infer 3D objects from monocular image. Chen et al. [2] generate 3D proposals in 3D layouts with context priors and project 3D proposals back into Region of Interest in the image to crop corresponding convolutional features for further classification and bounding boxes regression. Mousavian et al. [17] estimate 3D object poses by using geometric transformation constraints imposed by perspective projection from 3D bounding boxes to 2D windows in the image. Deep MANTA [1] contains a coarse-to-fine object proposal framework and makes the final refining network to achieve multitasks: 2D object detection, part localization, visibility characterization and 3D template estimation. Since projecting a 3D scene onto a 2D plane is an irreversible process and lots of 3D information is lost in forming the 2D image, 3D object detector based on monocular image usually is not as good as methods using 3D point cloud in terms of detection accuracy. Instead of using RGB image, our method detects objects from 3D point cloud only.

2.2.2 3D point cloud based detection

The 3D point cloud can directly represent real-world scenes and often is regarded as an important information source to detect objects. The common procedure for processing the point cloud is to convert irregular point cloud into a regular data formats, such as 3D voxels or 2D view images, then apply the 2D convolutional operation to learn features. Li et al. [15] encode the point cloud into a Cylindric-like image where every valid element carries 2-channel data. The MV3D [3] includes a new data representation and convert 3D point cloud into birds eye representation including many height maps, one density map and intensity

[1]The source code can be found: `https://github.com/Benzlxs/3DBN`

map, then apply 2D CNN on birds view to propose 3D object candidates. These methods use the mature 2D CNN framework to perform detection efficiently, but fail to learn the feature in 3D space. Instead of compressing 3D data into 2D planes, other methods [14, 19] discretize the point cloud into 3D voxel grids and process them with 3D convolution. VoxelNet [28] and SECOND [27] adopt PointNet [21] to extract features from the raw point cloud for every voxel in low resolution, then employ a shallow 3D CNN network to convert 3D feature maps into 2D feature maps, so that a powerful 2D-CNN-based network can be used to further extract features and generate proposals. These methods attempt to learn features of the object in the original 3D space without dimensional reduction. However, the shallow 3D CNN layers are not enough to extract abstract 3D features and they almost still heavily rely on the 2D region proposal network. In our paper, we propose to build a deeper 3D backbone network to fully exploit feature learning in 3D feature space instead of heavily using a 2D CNN backbone network.

2.2.3 Detection by fusing RGB image 3D point cloud

Lots of methods [20, 3] are also adopted to fuse RGB image and 3D point cloud for object detection. The 3D proposals are usually generated using image or point cloud only, then feature maps extracted from both information sources are concatenated together to do further inference. The MV3D [3] predicts proposals on the LiDAR bird with conventional CNN, and and region-based fusion network merging features from three 2D image: LiDAR bird view, LiDAR front view and RGB image, for every proposal to output 3D bounding boxes. Based on 2D detection bounding boxes obtained from mature object detector in the image, Frustum PointNets [20] extrude 3D bounding frustums of objects in point clouds as proposals, then PointNet [21] is employed to process point cloud within every frustum to detect 3D objects. However, such approaches are very sensitive to synchronization error between two sensors: camera and 3D Lidar, and also complicate the sensory system. In contrast, our method only relies on the point cloud, but achieve better performance than these fusion based methods.

3 3D Backbone Network

The sparse CNN will be employed to construct the 3D backbone network. Before demonstrating the main work, we firstly introduce how the sparse CNN [8, 7] functions.

3.1 Sparse CNN

Define the input data as one from the current layer which can be the raw data in the input layer or feature maps in the hiddent layer under the common convention and its elements are input points. The output data is from the next layer which can be feature maps in the hidden layer or output layer under the common convention, and its elements are output points. Lower case, capital and bold capital represent the scalar, vector and matrix respectively. We assume that the size of input data is $c_{in} \times h_{in} \times w_{in} \times l_{in}$ where the 3D input field size is $h_{in} \times w_{in} \times l_{in}$, and c_{in} is the number of 3D feature map channels. The size of conventional 3D CNN filter is $(k \times k \times c_{in} \times c_{out},\ s)$ where k is convolutional kernel size, s is the stride size and c_{out} is the number of output data. If this filter is used to process the aforementioned input data, the size of the output data will be $c_{out} \times h_{out} \times w_{out} \times l_{out}$, where $h_{out} = (h_{in} - k)/s$ and so are w_{out} and l_{out}. The conventional 3D CNN is very computation and memory demanding, so it is not widely used in processing 3D data.

In real-world applications, most 3D data, such as point cloud, is very sparse. After being voxelized, the 3D input data is assumed to enclose $h_{in} \times w_{in} \times l_{in}$ 3D grids, among which there are a_{in} active points[2]. Usually, $a_{in} \ll h_{in} \times w_{in} \times l_{in}$, so that the majority of memory and computation are spent on empty data area. Taking full advantage of sparsity of 3D data, 3D sparse CNN remove the computation and memory cost of empty point in input data. To achieve this, the input data, 4D tensor with the size: $c_{in} \times h_{in} \times w_{in} \times l_{in}$ is represented with two matrices: the input feature matrix $\boldsymbol{M}_{in}$, and the hash table $\boldsymbol{H}_{in}$. The input feature matrix $\boldsymbol{M}_{in}$, whose size is $a_{in} \times c_{in}$, only carries the information of all active points, and each row vector with the size of $1 \times c_{in}$ is the feature value of one active point across all channels, c_{in}. The hash table contains the location map of active points in input data grid, whose size is $h_{in} \times w_{in} \times l_{in}$, to the corresponding row number of the input feature matrix $\boldsymbol{M}_{in}$, and it is represented as $\boldsymbol{M}_{in} = \{(KEY_i, value_{key_i}) : i \in \{0,1,2,....,(a_{in}-1)\}\}$, where $KEY_i = [x_i, y_i, z_i]$ is a 3×1 vector of integer coordinates indicating the location of active point in the input data grid, and $value_{key_i}$ is the row number in $\boldsymbol{M}_{in}$.

[2]The point in the current layer is active if it satisfies one of following criterions: 1. its vector in the input layer is not zero; 2. in the hidden layer it receives information from the non-zero vector input layer.

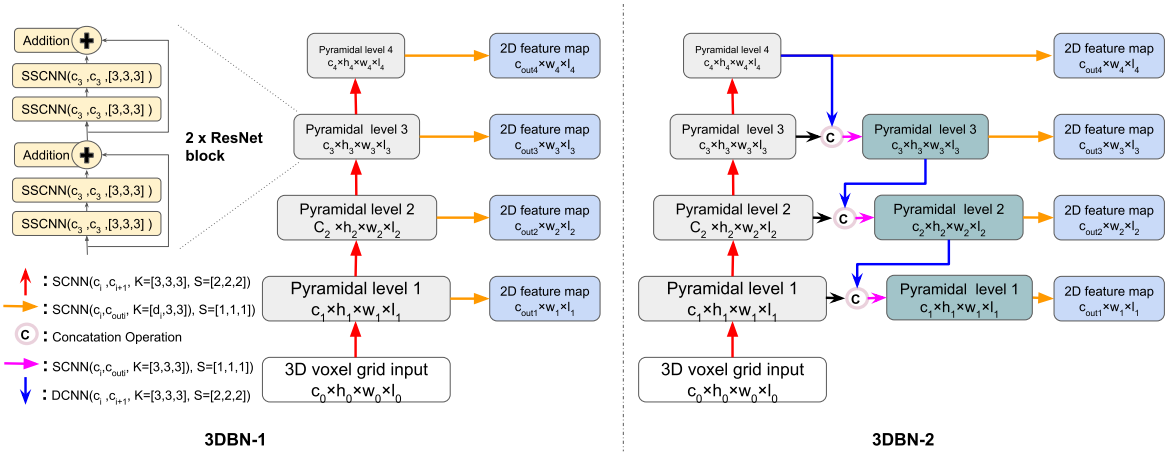

Figure 1: 3D Backbone Network: SSCNN denotes submanifold sparse 3D CNN, SCNN is the sparse 3D CNN, DCNN is the sparse 3D deconvolutional neural network, and K and S are the convolutional kernel and stride respectively.

Then, the rule book, ***Rule***, number of active point a_{out} and hash table $\boldsymbol{H}_{out}$ for the next layer will be generated, see Appendix A. The hash table can be built by iterating all output points which receive information from any active input point. If the output point is visited for the first time, its spatial location and index will be added to $\boldsymbol{H}_{out}$. The rule book, ***Rule***, depicting neuron connections from the current layer to the next layer is created based on the offset between input points and its corresponding output point. Keeping output point in the center of the 3D convolutional kernel, offset is used to describe the position of input point in the 3D kernel. For example, offset, (0,0,0), represent input point located in the most top-right-front corner of 3D kernel centered at one output input, and offset $(f-1, f-1, f-1)$ represent input point is located in the most bottom-left-back corner of the 3D kernel. Due to the sparsity of input data, it frequently happens that a lot of points in the 3D kernel for one output point are inactive, so an output point is usually only connected with several active input point in a 3D kernel.

The output feature matrix $\boldsymbol{M}_{out}$ with the size, $a_{out} \times c_{out}$, will lastly be computed according to algorithm 2 in Appendix B. The matrix $\boldsymbol{W}$ is the 3D convolutional parameter matrix with the size, $n_{in} \times f^3 \times n_{out}$, and $W^{(i,j,k)}$ with size $n_{in} \times n_{out}$ is the parameter weight for one offset or one type of connection between input and output point. $\boldsymbol{M}_{out}$ is initialized with zeros. For one offset (i,j,k), the feature matrix of all input point $\boldsymbol{Rule}^{(i,j,k)}(:,0)$ will be multiplied with the corresponding weighting parameters $W^{(i,j,k)}$, and added to the output feature matrix of corresponding output points $\boldsymbol{Rule}^{(i,j,k)}(:,1)$. After all offsets are gone through, bias vector B whose size is $1 \times c_{out}$, is added to $\boldsymbol{M}_{out}$, followed by an activation function, ReLU [18]. Unlike a regular convolution, sparse 3D CNN discards information from inactive points by assuming that the input from those points is zero instead of ground state. The hash table and rule book generation will be done on CPU and feature matrix computation will be on GPU. The submanifold sparse CNN restricts an output point to be active if and only if the point at the corresponding position in the input data is active. Therefore, the input hash table can be taken as the output hash table to increase the computation efficiency, and the sparsity of 3D data can be kept the same.

3.2 3D Backbone Network

By using sparse 3D CNN, we aim at building 3D backbone networks for high-resolution and stronger semantic 3D feature maps for 3D object detection. The input for 3D backbone network is voxelized 3D grids, and outputs are 2D feature maps which are proportionally sized at multiple levels. There are two types of 3D backbone network: 3DBN-1 and 3DBN-2, in Fig. 1.

3DBN − 1: It is the unidirectional feed-forward computation of sparse 3D CNN, which computes the pyramidal 3D feature hierarchy. Each pyramidal level usually consists of several 3D feature maps with the same size, and network layers generating these 3D feature maps are in the same network stage. Every network layers in one stage is a residual network (ResNet) block. The sparse 3D CNN with the stride size of (2,2,2) is applied to reduce 3D resolutions of feature maps and increase the channel number c_i in order to extract stronger semantic features. At the end of every pyramid level, a sparse 3D CNN kernel with the

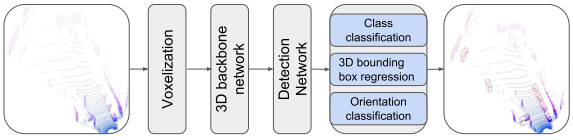

Figure 2: Framework of the whole 3D detection network.

size of $(d_i, 1, 1)$ is used to compress the 3D features into 2D feature maps for anchoring object proposals. The reason is that it is impossible to detect two 3D objects in the same vertical direction (Z-axis) for 3D objects in the traffic scenes. Therefore, we compress the 3D features into 2D maps to reduce the computation cost of generating 3D anchors, besides, 2D CNN is computationally cheaper than sparse 3D CNN. There is usually the large semantic gap among different 3D feature maps of pyramidal level in the unidirectional 3DBN-1, which is caused by different convolutional depths. The deeper the neural network is, the stronger semantic features are extracted.

3DBN – 2: To attain continuously semantic 3D feature map, we build another 3D backbone network, 3DBN-2. Its bottom-top path is the same as 3DBN-1, but it has one extra top-bottom path that passes the 3D semantic context down, so we can achieve high-level 3D semantic feature maps at all pyramidal levels. The low-resolution but semantically stronger feature maps are upsampled to the size of the upper pyramidal level by using the deconvolutional neural network with the stride size of (2,2,2), then concatenated to the corresponding feature maps of the same spatial size. The 3×3 sparse CNN is appended to concatenated feature maps in order to alias the effect of upsampling and concatenation operation. Similarly, a sparse 3D CNN kernel with the size of $(d_i, 1, 1)$ is used to compress the 3D features into 2D feature maps, so size and type of its outputs are the same as that of 3DBN-1.

4 3D Object Detection

The whole detection network consists of three main components: point cloud voxelization, 3D backbone network and detection network, as shown in Fig. 2. Voxelization converts raw point could into voxels which can be consumed by the 3D backbone. Detection network mainly generates 3D object proposals from feature maps provided by the 3D backbone network.

4.1 Voxelization

The input for sparse 3D CNN should be in the regular grid, and point cloud data is usually in the irregular format, so we cannot directly feed the point cloud into the aforementioned 3D backbone network. Voxelization is firstly required to preprocess the unordered and irregular point cloud data in order to obtain its voxel representation.

The 3D point cloud space range is supposed to be l, w and h along the X, Y and Z axes respectively. The size of voxel is set to v_l, v_w and v_h. Therefore, the size of 3D voxel grid is l/v_l, w/v_w and h/v_h. For the feature of every voxel, the simple method is to calcuate the binary value; 1 is assigned to the feature if there exists a point in the corresponding voxel, otherwise the voxel is empty and the feature is 0. This method is called Binary Voxel (BV) in our work. The drawback of this kind of method is that the local shape information of points within a voxel is lost, so voxel size in the BV is usually set very small to generate high-resolution 3D grids; for example, v_l, v_w and v_h are 0.025, 0.025 and 0.0375 meters respectively in our experimental settings. When using BV to produce the 3D voxel grid input, whose size is $c_0 \times h_0 \times w_0 \times l_0$, for 3D backbone network, the voxelization module in Fig. 2 also includes four layers of sparse 3D CNN with convolutional kernel size [3,3,3] and stride size [2,2,2] to decrease the resolution of 3D voxel grid from $(h/v_h) \times (w/v_w) \times (l/v_l)$ to $h_0 \times w_0 \times l_0$ ($h/v_h = 16 \times h_0$, $w/v_w = 16 \times w_0$ and $l/v_l = 16 \times l_0$), and the feature dimension per voxel will be increased from 1 to c_0.

An alternative method is to use PointNet [21] to extract the pointwise features, which is called the Voxel Feature Encoding (VFE) [28]. VFE enables inter-point interaction within a voxel by combining pointwise features with a locally aggregated feature in order to learn the local 3D shape information, and hence can alleviate information loss due to data quantization. Since the VFE requires an amount of points per valid voxel, the size of voxel in VFE should be large. For example, we set v_l, v_w and v_h to 0.2, 0.2 and 0.3 meters in the experiment, which are much larger than these in the BV. If this method is adopted, the voxelization consisted of two layers of VFE to learn the voxelwise feature with the size $1 \times c_0$, and 3D voxel grid shape remains the same. Therefore, $h/v_h, w/v_w$ and l/v_l equal h_0, w_0 and l_0 respectively.

4.2 Detection network

The detection network aims to detect objects from feature maps generated by the 3D backbone network, and it will fuse all the 2D feature maps together to form a final high-resolution and stronger semantic feature maps for classification and regression tasks. For object detection in the image, the scale of the

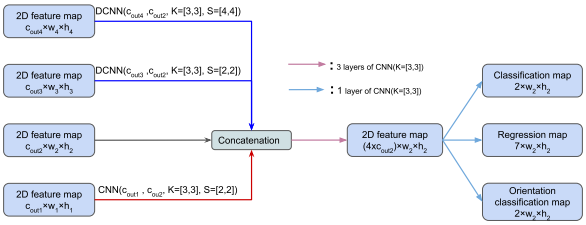

Figure 3: Detection Network: CNN is the 2D convolutional neural network, DCNN is the 2D deconvolutional neural network, and K and S are the convolutional kernel and stride respectively.

object is unknown, and object detection is usually conducted on multiple levels of feature map pyramid; higher spatial resolution feature maps accommodate small objects and lower spatial resolution feature maps are for large objects. However, the scale of 3D objects in the point cloud is almost fixed and scale variation of 3D bounding boxes in the 3D space is much smaller than that of 2D bounding boxes in the image. Therefore, we concatenate all 2D feature maps together, then employ 3 layers of conventional 2D CNN to fuse feature maps of different spatial resolution together. The classification, location and orientation predication maps are generated on the final layer of feature maps by using a layer of 2D CNN, as shown in Fig. 3.

4.3 Loss function

The loss consists of three main components: classes classification, 3D bounding box regression and orientation classification, as shown in equation 1. Variables κ, λ and μ are weighting parameters to balance the relative importance.

$$L_{all} = \kappa L_{cls} + \lambda L_{reg} + \mu L_{ori} \tag{1}$$

Classes Classification L_{cls}: Compared with anchors $\sim 140,000$, the number of detected objects, usually less than 30, in point cloud is extremely small, which leads to significant imbalance between positive labels and negative labels. To address this, we calculate the classification loss for positive anchors and negative anchors separately, besides, we applied the $\alpha-$balanced focal loss [16] for classification, as shown in equation 2, in which N_{pos} and N_{neg} are the number of positive and negative labels; p_i^{pos} and p_j^{neg} are the softmax output for positive anchors and negative anchors respectively; α is the balanced parameters for positive and negative samples; modulating factors $(1-p_i^{pos})^\gamma$ and $(p_j^{neg})^\gamma$ put less weights and reduce loss for well-classified examples.

$$\begin{aligned} L_{cls} = &-\frac{1}{N_{pos}}\sum_i \alpha(1-p_i^{pos})^\gamma log(p_i^{pos}) \\ &-\frac{1}{N_{neg}}\sum_j (1-\alpha)(p_j^{neg})^\gamma log(1-p_j^{neg}) \end{aligned} \tag{2}$$

3D Bounding Box Regression L_{reg}: The regression targets $\mathbf{r}^*$ are defined as a 1×7 vector, $[r_x, r_y, r_z, r_l, r_w, r_h, r_\theta]$. Vector $\mathbf{r_i^*}$ encodes the 3D ground truth bounding boxes based on anchor boxes correspondence and the computation method can be found in the Appendix C. For heading angle regression, Voxelnet [28] uses angle residual between ground truth and anchor boxes directly; AVOD [13] encodes heading into orientation vector by using trigonometric functions. Similarly, SECOND [27] employ trigonometric functions, *sin*, to encode the radian offset. Different from them, we propose the simple and effective angle loss function by preprocessing the yaw angle in ground truth labels making one angle to represent two opposite orientation, so the ground truth bounding boxes with opposite orientation will be treated as the same and the original range 2π is changed into one π. The reason for such preprocessing is that some angle residual r_θ sets, such as 0 and π, match the same ground truth bounding boxes but their losses are significantly different, and it makes regression network hard to learn. The orientation information will be abandoned in 3D bounding box regression but will be taken into consideration in orientation classification, so only small and continuous angle offset r_θ with regard to anchor boxes will be encoded. The $smoothL1$ function [6] is used to calculate the loss value, as given in equation 3.

$$L_{reg} = \frac{1}{N_{pos}}\sum_i SmoothL1(\mathbf{r_i^*}, \mathbf{r_i}) \tag{3}$$

Orientation Classification L_{ori}: To retain the orientation of objects, we adopt an extra softmax loss function, as shown in equation 4. The ground truth boxing boxes whose heading angle is larger than zero are trained as positive labels, otherwise negative labels. The p_i^{+pos} and p_j^{-neg} are the softmax output for positive labels and negative labels respectively.

$$L_{ori} = -\frac{1}{N_{pos}}(\sum_i log(p_i^{+pos}) + \sum_j log(1-p_j^{-neg})) \tag{4}$$

5 Experiments

5.1 Implementation details

5.1.1 Dataset

The KITTI benchmark dataset [5] is borrowed to validate our proposed methods. It includes 7481 training and 7518 testing sets of LIDAR laser data. Since the ground truth of the testing dataset is not publicly available, we split the training dataset in the 1:1 ratio for training and validation, and use them to conduct comparative experiments for tuning our models and comparisons with other methods. The adequately-trained network model with optimal configuration is deployed to process the testing dataset and their results are submitted to the KITTI benchmark.

5.1.2 Network details

The range of point cloud we take into consideration is [0, 70.2], [-39.9, 39.9] and [-3.25, 1.25] meters along X, Y and Z axis respectively. If BV is selected for the voxelization, 3D voxel size is [0.025, 0.025, 0.0375] meters along X, Y and Z axis respectively. The voxelizaiton consists of four layers of sparse 3D CNN block, and sizes of their corresponding feature maps are $16(c) \times 127(h) \times 3199(w) \times 2815(l)$, $32 \times 63 \times 1599 \times 1407$, $48 \times 31 \times 799 \times 703$ and $64 \times 15 \times 399 \times 351$, so $[c_0, h_0, w_0, l_0]$ equals $[64, 15, 399, 351]$. When VEF is used, the voxelization includes two layers of VEF to increase feature channels from 7 to 32 and from 32 to 128 and c_0 is 128; the 3D voxel size is [0.2, 0.2, 0.3] meters and $[h_0, w_0, l_0]$ is $[15, 399, 351]$; the maximum number of kept points per voxel is 35. As for 3D backbone networks and detection network, $[c_1, c_2, c_3, c_4]$ is set to $[64, 80, 96, 128]$; $[h_1, h_2, h_3, h_4]$ is $[15, 7, 3, 1]$; $[w_1, w_2, w_3, w_4]$ is $[399, 199, 99, 49]$; $[l_1, l_2, l_3, l_4]$ is $[351, 175, 87, 43]$. Dimensions of 2D feature maps $[c_{out1}, c_{out2}, c_{out3}, c_{out4}]$ are $[128, 128, 128, 128]$.

The first set of anchors is 3D cube boxes with size [1.6 (l), 3.9 (w), 1.56 (h)] meters and orientation [0, $\pi/2$], and the second set of anchors is 3D cube boxes with size [1.581, 3.513, 1.511] and [1.653, 4.234, 1.546] meters and orientation [0, $\pi/2$]. The two sets of size are obtained by using K-mean clustering among all 3D ground truth bounding boxes. The anchor whose Intersection over Union (IoU) with ground truth is over 0.7 for the car is treated as a positive label, and one whose IoU is less than 0.5 for the car is regarded as a negative label. α, β and δ in the loss function are 1, 2 and 1 respectively.

The AdamOptimizer [11] is employed to train our network, and it is configured with an initial learning rate of 0.0002 and exponential decay factor of 0.8 for every 18570 steps.

5.1.3 Data augmentation

To reduce the overfitting caused by the fact that the complex model is trained from scratch on the small dataset, four techniques of data augmentation are introduced in our method as following:

Rotation: every point cloud in the entire block is rotated along the Z-axis and around the origin [0,0,0] by θ, and θ is drawn from the uniform distribution [-π/4, π/4].

Scaling: similar with global scaling operation in image-based data augmentation, the whole point cloud block is zoomed in or out. The XYZ coordinates of points in the whole block are multiplied with a random variable which is drawn from the uniform distribution [0.95, 1.05].

Motion: in order to increase the variation of each ground truth 3D bounding box, the motion is applied to independently perturb each bounding box. The 3D bounding box is firstly rotated around Z-axis by an uniformly distributed random variable, $\delta \in [\pi/2, \pi/2]$. Then, a translation $[\triangle X, \triangle Y, \triangle Z]$ is used to move the rotated bounding box, and $[\triangle X, \triangle Y, \triangle Z]$ are independently sampled from a Gaussian distribution with a mean of zero and a standard deviation of 1.0. The collision test will be conducted after one motion.

Increment: there are usually only a few ground truth 3D bounding boxes in one point cloud block, which leads to a very small number of positive samples. To complicate the training scenarios and increase the number of positive samples, we insert some extra 3D ground truth bounding boxes and their corresponding points into current training samples. Extra 3D bounding boxes are randomly picked from the database which includes all ground truth bounding boxes in the training dataset. Similarly, the collision test will be conducted after the increment.

5.2 Comparsions on the KITTI validation dataset

We design two kinds of 3D backbone network architectures and two different voxelization techniques that can also be employed to generate 3D voxel grid input. In order to find the performance of different combinations, we conduct comparison experiments on the KITTI validation dataset. The result can be found in Table 1 and shows that the combination of BV and

Figure 4: Visualization of 3D object detection on KITTI testing dataset. Top figures include 3D bounding boxes in RGB images and bottom ones are their corresponding point cloud.

3DBN-2 can achieve the best performance. However, the BV voxelization will lead to large increase in processing time. The main reason is that the high-resolution of 3D voxel grid makes sparse CNN inefficient to generate the rule book which is generated on CPU instead of GPU. On the other hand, the VFE is more efficient due to the fact that the most of PointNet operations, 1D CNN, are implemented on GPU. In addition, 3DBN-2 shows a better performance than 3DBN-1 with little extra computation time.

Voxelization	Backbone Network	mAp(%)	Time(s)
BV	3DBN-1	79.32	0.73
VFE	3DBN-1	79.04	0.18
BV	3DBN-2	80.74	0.78
VFE	3DBN-2	80.46	0.2

Table 1: Comparison of 3D detection model with the different combination of voxliezation and 3D backbone networks. The mAP(%) is the mean average prcesion of easy, moderate and hard categories for 3D car detection and time(s) is needed for processing one sample.

We also conduct various experiments to compare our 3D object detection methods. The combination of BV and 3DBN-2 and that of VEF and 3DBN-2 are selected to perform comparisons with other methods. Comparison results are shown in Table 2, from which we can see that our 3D backbone networks are more suitable for 3D object detection. Unlike the AVOD-FPN, F-PointNet and MV3D using both Lidar data and image, our method solely relies on point cloud.

Methods	Easy	Moderate	Hard	Time(s)
BV+3DBN-2	87.98	**77.89**	**76.35**	0.78
VEF+3DBN-2	**88.20**	77.59	75.58	0.2
SECOND[27]	87.43	76.48	69.10	**0.05**
AVOD-FPN[13]	84.41	74.44	68.65	0.1
VoxelNet[28]	81.97	65.46	62.85	0.23
F-PointNet[20]	83.76	70.92	63.65	0.36
MV3D[3]	71.29	62.68	56.56	0.17

Table 2: Comparsion of proposed methods with others for 3D car detection. The performance is depicted by mAP(%) of every category.

5.3 Performance on the KITTI testing dataset

The performance of our model consisting of BV and 3DBN-2 on KITTI testing dataset is shown in Table 3 and 4, and several samples are displayed in Fig. 4. Our method ranks top three among all known methods with released publications on the KITTI benchmark[3], which indicates that our method can reach top performance in comparisons with other 3D object detection methods.

Methods	Easy	Moderate	Hard
3DBN	**83.56**	**74.64**	**66.76**
SECOND[27]	83.13	73.66	66.20
AVOD-FPN[13]	81.94	71.88	66.38
VoxelNet[28]	77.47	65.11	57.73
F-PointNet[20]	81.20	70.39	62.19
MV3D[3]	71.09	62.35	55.12

Table 3: Comparison of proposed methods with others on KITTI testing dataset for 3D car detection.

Table 3 represents comparisons in terms of 3D object detection accuracy and shows that the proposed 3D backbone network is better at extracting 3D features from point cloud for 3D object detection tasks than other methods that compress 3D data into 2D one [3, 13] or employs a shallow 3D network as a supplement for 2D region proposal network [28, 27].

Methods	Easy	Moderate	Hard
3DBN	89.93	**87.95**	79.32
SECOND[27]	87.84	81.31	71.95
AVOD-FPN[13]	**89.95**	87.13	**79.74**

Table 4: 3D car orientation prediction performance of our proposed method in comparison with others on KITTI testing dataset.

The orientation prediction task is compared in Table 4 which only include AVOD-FPN[13] and

[3]Testing results can be found in the KITTI object detection leaderboard `http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d`. The date when our results are submit is January 23st 2019.

SECOND[27] due to the missing orientation results of other methods. Our simple orientation regression method can achieve competitive performance with AVOD-FPN which employs complex encoding mechanism.

6 Conclusion

We aim to build the 3D backbone networks by using sparse CNN and their corresponding detection network for 3D object detection. It is found that the proposed 3D backbone networks are more suitable for extracting 3D features than other approaches that convert 3D data into 2D format and apply 2D CNN to process data. However, the efficiency of sparse CNN still limits the depth of our 3D backbone network since its rule book generation is operated on CPU and cannot be computed in parallel. Further research work will be focused on how to improve the efficiency of sparse convolutional operation which is the key to sparse 3D data processing for point cloud.

References

[1] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Céline Teulière, and Thierry Chateau. Deep MANTA: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. *CoRR*, abs/1703.07570, 2017.

[2] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2147–2156, June 2016.

[3] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. *CoRR*, abs/1611.07759, 2016.

[4] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[5] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[6] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[7] Ben Graham. Sparse 3d convolutional neural networks. *CoRR*, abs/1505.02890, 2015.

[8] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[13] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Lake Waslander. Joint 3d proposal generation and object detection from view aggregation. *CoRR*, abs/1712.02294, 2017.

[14] Bo Li. 3d fully convolutional network for vehicle detection in point cloud. *CoRR*, abs/1611.08069, 2016.

[15] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3d lidar using fully convolutional network. *CoRR*, abs/1608.07916, 2016.

[16] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[17] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry. *CoRR*, abs/1612.00496, 2016.

[18] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[19] D. Prokhorov. A convolutional learning system for object classification in 3-d lidar data. *IEEE Transactions on Neural Networks*, 21(5):858–863, May 2010.

[20] Charles Ruizhongtai Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. *CoRR*, abs/1711.08488, 2017.

[21] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[25] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Data-driven 3d voxel patterns for object category recognition. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1903–1911, 2015.

[26] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[27] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10), 2018.

[28] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017.

Appendix

A Algorithm 1

Algorithm 1 Ruler book and output hash table generation algorithm

H_{in}: hash table of input date, the coordinates of all active points are saved in $key_i = (x_i, y_i, z_i)$
H_{out}: hash table of output data
$Cntr$: number of input and output pairs in ruler book
f: size of 3D CNN kernel
a_{in}: number of active point in input data
s: stride of 3D CNN
$GetOutputCoord$: getting coordinates of all output points in regards to one input point
$GetOffset$: calcuating offset of input point with regard to output point in 3D convolutional kernel
$H_{out} \leftarrow []$
$Rule \leftarrow []$
$Cntr \leftarrow []$
$a_{out} \leftarrow 0$
for $i_{in} \leftarrow 0$ to a_{in} **do**
  $P_{in} \leftarrow key_{i_{in}}$ in H_{in}
  $\boldsymbol{P}_{out} \leftarrow GetOutputCoord(P_{in}, k, s)$
  for $P \in \boldsymbol{P}_{out}$ **do**
    if $P \notin key$ in H_{out} **then**
      $key_{a_{out}} \leftarrow P$
      $value_{key_{a_{out}}} \leftarrow a_{out}$
      $a_{out} \leftarrow a_{out} + 1$
    end if
    $(i, j, k) \leftarrow GetOffset(P_{in}, P)$
    $\boldsymbol{Rule}^{(i,j,k)}(Cntr(i, j, k), 0) \leftarrow i_{in}$
    $key \leftarrow P$
    $\boldsymbol{Rule}^{(i,j,k)}(Cntr(i, j, k), 1) \leftarrow value_{key}$
    $Cntr(i, j, k) \leftarrow Cntr(i, j, k) + 1$
  end for
end for

B Algorithm 2

Algorithm 2 Output feature matrix generation algorithm

f: size of 3D CNN kernel
B: bias vector of 3D CNN with size $1 \times n_{out}$
$ActivationFunc$: activation function used by sparse 3D CNN
W: the 3D convolutional parameter matrix with the size, $n_{in} \times f^3 \times n_{out}$, $W^{(i,j,k)}$ is parameter weight for one offset or connection with size $n_{in} \times n_{out}$
$\boldsymbol{M}_{out}$: output feature matrix
$\boldsymbol{M}_{out} \leftarrow 0$
for $i \leftarrow 0$ to f **do**
  for $j \leftarrow 0$ to f **do**
    for $k \leftarrow 0$ to f **do**
      for $idx \leftarrow 0$ to $Cntr(i, j, k)$ **do**
        $i_{in} \leftarrow \boldsymbol{Rule}^{(i,j,k)}(idx, 0)$
        $i_{out} \leftarrow \boldsymbol{Rule}^{(i,j,k)}(idx, 1)$
        $\boldsymbol{M}_{out}(i_{out}, :) \leftarrow \boldsymbol{M}_{out}(i_{out}, :) +$
        $\boldsymbol{M}_{in}(i_{in}, :) \times W^{(i,j,k)}$
      end for
    end for
  end for
end for
$\boldsymbol{M}_{out} \leftarrow \boldsymbol{M}_{out} + B$
$\boldsymbol{M}_{out} \leftarrow ActivationFunc(\boldsymbol{M}_{out})$

C Residual vector computation

The 3D ground truth bounding box is parameterize as $[x_{gt}, y_{gt}, z_{gt}, l_{gt}, w_{gt}, h_{gt}, \theta_{gt}]$ and its corresponding anchor box are $[x_a, y_a, z_a, l_a, w_a, h_a, \theta_a]$, where $[x, y, z]$ denote the center location, $[l, w, h]$ is the length, width and height of the 3D bounding box, and θ is the yaw angle around Z-axis.

$\mathbf{r}^* = [r_x, r_y, r_z, r_l, r_w, r_h, r_\theta]$ can be computed as following:

$$
\begin{aligned}
& r_x = \frac{x_{gt} - x_a}{d_a}, r_y = \frac{y_{gt} - y_a}{d_a}, r_z = \frac{z_{gt} - z_a}{h_a} \\
& r_l = log(\frac{l_{gt}}{l_a}), r_w = log(\frac{w_{gt}}{w_a}), r_h = log(\frac{h_{gt}}{h_a}) \\
& u_\theta = \theta_{gt} - \theta_a, d_a = \sqrt{l_a^x + w_a^2}
\end{aligned}
\tag{5}
$$