arXiv:1908.02990v2 [cs.CV] 16 Aug 2019

# Fast Point R-CNN

Yilun Chen[1] Shu Liu[2] Xiaoyong Shen[2] Jiaya Jia[1,2]
[1]The Chinese University of Hong Kong [2]Tencent YouTu Lab
{ylchen, leojia}@cse.cuhk.edu.hk, {shawnshuliu, dylanshen}@tencent.com

## Abstract

*We present a unified, efficient and effective framework for point-cloud based 3D object detection. Our two-stage approach utilizes both voxel representation and raw point cloud data to exploit respective advantages. The first stage network, with voxel representation as input, only consists of light convolutional operations, producing a small number of high-quality initial predictions. Coordinate and indexed convolutional feature of each point in initial prediction are effectively fused with the attention mechanism, preserving both accurate localization and context information. The second stage works on interior points with their fused feature for further refining the prediction. Our method is evaluated on KITTI dataset, in terms of both 3D and Bird's Eye View (BEV) detection, and achieves state-of-the-arts with a 15FPS detection rate.*

## 1. Introduction

One challenging task in 3D perception is 3D object detection, which serves as the basic component for perception in autonomous driving, robotics, *etc*. Deep convolutional neural networks (CNN) greatly improve performance of 3D object detection [5, 43, 25, 15, 40, 23]. Recent approaches of 3D object detection utilize different types of data, including monocular [3] images, stereo images [4] and RGB-D images [32, 33]. In autonomous driving, point clouds captured by LiDAR are the more general and informative data format to help make prediction [5, 25, 15, 23].

**Challenges** LiDAR point cloud is an essential type of geometry data for 3D detection. High sparseness and irregularity of point cloud, however, make it not easily tractable for CNN. One scheme is to transform the sparse point cloud to the volumetric representation in compact shape by discretization, which is called voxelization. This representation enables CNN to perform recognition.

However, volumetric representation is still computationally challenging. One line of solutions is to use a coarse grid [43, 40, 23, 2, 31, 1]; but coarse quantization prevents following CNN from utilizing fine-grained information. Several consecutive convolutional layers and subsampling operations in the CNN worsen the problem.

Another line [26, 28, 19, 36] is to process point cloud directly for 3D object recognition. Different from the volumetric representation, coordinates of point cloud and their structure are directly fed into the neural network to exploit precise localization information. We note applying these methods to large-scale point clouds for autonomous driving is still computationally very heavy.

**Our Contributions** In this paper, we propose a unified, fast and effective two-stage 3D object detection framework, making use of both voxel representation and raw point cloud data. The first stage of our network, named VoxelRPN, directly exploits the voxel representation of point clouds. Computationally economical convolutional layers are adopted for both high efficiency and surprisingly high-quality detection.

In the second stage, we apply a light-weight PointNet to further refine the predictions. With a small number of initial predictions, the second stage is also in a very fast speed. We design the module with attention mechanism to effectively fuse the coordinates of each interior point with the convolution feature from the first stage. It makes each point aware of its context information.

One characteristic of our approach is that it benefits from both representation of point clouds in volumetric representation and raw dense coordinates. The 3D volumetric representation provides a robust way to process point clouds. The light-weight PointNet in the second stage inspects coordinates of points again to capture more localization information with enlarged receptive fields, producing decent results. Since our method utilizes convolutional feature for each region on point clouds and is with high efficiency, we name it Fast Point R-CNN.

With this conceptually simple structure, we achieve high efficiency and meanwhile decent 3D detection accuracy, achieving state-of-the-art results. It is even more effective than prior methods that take both RGB and point clouds as input. The main contribution of this paper is threefold.

- We propose a quick and practical two-stage 3D object detection framework based on point clouds (without RGB images), exploiting both volumetric representation and raw dense input of point clouds.
- Our system consists of both 2D and 3D convolution to preserve information. We fuse convolutional features with point coordinate for box refinement.
- Our system runs at 15FPS and achieves state-of-the-art performance in terms of BEV and 3D detection, especially for high quality object detection.

## 2. Related Work

We briefly review recent work on 3D data representation of point clouds and 3D object detection.

**3D Data Representation** Representation of point clouds from 3D LiDAR scanners is fundamental for different tasks. Generally there are two main ways – voxelization [24, 37] or raw point clouds [26, 28, 19, 36]. For the first type, Maturana *et al*. [24] first applied 3D convolution for 3D object recognition. For the point-based approaches, PointNet [26] is the pioneer to directly learn feature representation based on raw points. It further aggregates global descriptors for classification. Recently, Rethage *et al*. [6] employed PointNet as the low-level feature descriptor in each 3D grid and applied 3D convolution. There are also other methods that do not process 3D data directly. For example, most view-based methods [34, 27, 35] care more about 2D color and gather information from different views of rendered images.

**3D Object Detection** Over past a few years, a series of 3D detectors [5, 25, 15, 43, 40, 23, 20, 29, 41, 16] achieved promising results on KITTI benchmark [8].

*Joint Image-LiDAR Detection*: Several approaches [5, 15, 20, 25] fused information from different sensors, such as RGB images and LiDAR. For example, MV3D [5] fused BEV and front view of LiDAR points as well as images, and designed a deep fusion scheme to combine region-wise features from multiple views. AVOD [15] fused BEV and images in full resolutions to improve prediction quality, especially for small objects. Accurate geometric information may be lost in the high-level layers with this scheme. Contfuse [20] compensated the geometric information via combining the convolution feature over LiDAR point cloud with the nearest image features and LiDAR point coordinates in the multi-scale scheme. In spite of geometric information encoded in each voxel, deeper layers have access mostly to coarse geometric feature. Based on a strong 2D detector on image, F-PointNet [25] and PointFusion [38] incorporated PointNet structures to estimate the amodal 3D box. But the 2D detector and PointNet are two separate stages and the final results heavily rely on the 2D detection results.

*LiDAR-based Detection*: Most LiDAR-based detection approaches process point clouds as voxel-input and apply either 2D convolution or 3D convolution to make prediction. Due to directly encoding coordinates of point clouds into voxel grid, deep layers may gradually lose this level of information. Several encoding techniques [32, 33, 17, 18] provide other representations to preserve more information. Chen *et al*. [5] encoded hand-crafted features for respective representation of BEV and front view. Instead of hand-crafted features, VoxelNet [43] applied VFE layers via a PointNet-like network to learn low-level geometric feature, by which it shows good performance. However the network structure is computationally heavy. Recently, SECOND [39] applied Sparse Convolution [10] to speed up VoxelNet and produce better results. PointPillars [16] applied acceleration techniques, including NVIDIA TensorRT, to achieve high speed. We note they may also accelerate our method. PointRCNN [29] and IPOD [41], concurrent with our work, generate point-wise proposals on Point Clouds, which consumes much computation on point-wise calculation in the similar region or background region.

## 3. Our Method

In this paper, we propose a simple and fast two-stage framework for 3D object detection with point cloud data, as shown in Figure 1. The first stage takes voxel representation as input and produces a set of initial predictions. To compensate the loss of precise localization information in the voxelization and consecutive convolution process, the second stage combines raw point cloud with context feature from the first stage to produce refinement results.

### 3.1. Motivation

Point cloud, captured by LiDAR, is a set of points with irregular structure and sparse distribution. It is not straightforward to make use of powerful CNN for training and inference on point cloud data. Discretizing points into voxelized input [43, 20] or projecting them to BEV with compact shape like RGB images [40, 23] forms a set of solutions, where abstract and rich feature representation can be produced. However, the discretization process inevitably introduces quantization artifacts with resolution decreasing to the number of bins in the voxel map. Moreover, consecutive convolution and downsampling operation may also weaken the precise localization signal that originally exists in point clouds.

Methods like PointNet [26] are specially designed for directly processing point cloud data. Directly applying these methods to entire point cloud, which is with a large scale in scenarios of autonomous driving, may produce more position-informative results. But they require a huge amount of GPU memory and computation, almost impossible to achieve a high detection speed. Other methods [25]

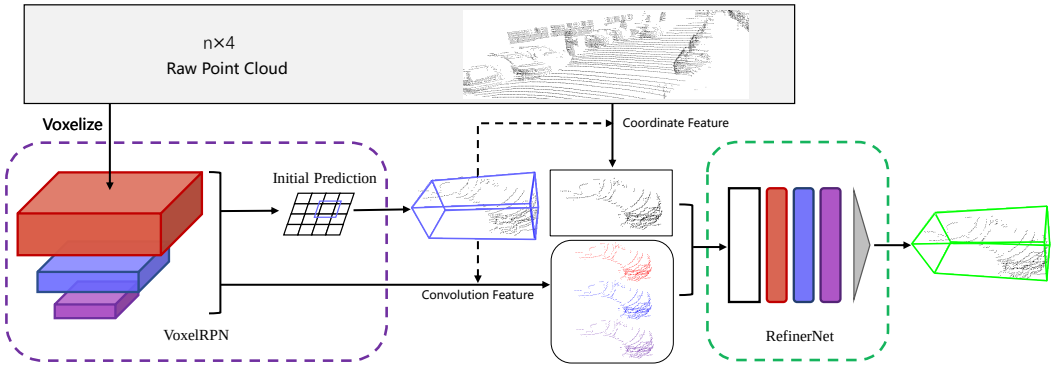


Figure 1. Overview of our two-stage framework. In the first stage, we voxelize point cloud and feed them to VoxelRPN to produce a small number of initial predictions. Then we generate the box feature for each prediction by fusing interior points' coordinates and context feature from VoxelRPN. Box features are fed to RefinerNet for further refinement.

rely on detection results from 2D detector followed by regression of the 3D amodal box for each object. This kind of pipeline heavily relies on 2D detection results, inheriting the weakness when detecting cluttered or distant objects in images. Clearly, directly working on point cloud data is a better choice if information can be properly made use of.

To this end, our method is new to exploit the hybrid of voxel and raw point cloud, without relying on RGB images. The two effective stages are voxel representation input to VoxelRPN to acquire a set of initial predictions in high speed, and RefinerNet to fuse raw point cloud and extracted context feature for better localization quality. These two components are elaborated on in the following.

### 3.2. VoxelRPN

VoxelRPN takes 3D voxel input and produces 3D detection results. It is a one-stage object detector.

**Input Representation** Input to VoxelRPN is the voxelized point cloud, which is actually a regular grid. Each voxel in the grid contains information of original points lying in the local region. Specifically, we divide the 3D space into spatially arranged voxels. Suppose the region of interest for the point cloud is a cuboid of size $(L, W, H)$ and each voxel is of size $(v_l, v_w, v_h)$, the 3D space can be divided into 3D voxel grid of size $(L/v_l, W/v_w, V/v_h)$.

There may be more than one points in a voxel. In VoxelNet [43], 35 points are kept and fed to the VFE layers to extract features. Our finding, however, is that simply using 6 points in each voxel followed a 8-channel MLP layer is already *adequate* to achieve reasonable performance empirically. With this representation in a compact shape, we easily exploit the great power of CNN for informative feature extraction.

**Network Structure** Aiming at 3D detection, our network needs to clearly filter information from $(X, Y, Z)$ dimensions. In [40, 23], the $Z$ dimension is simply transformed into the channels when generating the voxel representation. Then several 2D convolutions are applied. In this way, the information along $Z$ dimension vanishes quickly. As a result, detection only on BEV becomes achievable. Differently, VoxelNet [43] keeps three separate dimensions when producing voxels followed by three 3D convolutions. It is noticed that the efficiency is decreased.

Along a more appropriate direction, we find that a number of consecutive 3D convolutions are quite effective on preserving the 3D structure. Based on this observation, our backbone network is composed of 2D and 3D convolutions, achieving high efficiency as PIXOR [40] and even higher performance than VoxelNet [43].

We show details of our backbone network in Figure 2. The first part consists of six 3D convolutional layers, which only possess a small number of filters to keep time budget. Instead of aggressively downsampling features in the $Z$ dimension by filters with stride 2 and kernel size 3, we insert 3D convolution layers with kernel size 2 in the $Z$ dimension without padding, to better fuse and preserve information. What follows are three blocks of 2D convolutions for further abstraction and enlarging the receptive field.

Objects of the same category in 3D scene are generally with similar scales. Thus, different from the popular multi-scale object detector [21] in 2D images, which assigns object proposals to different layers according to their respective scales, we note that the HyperNet [14] structure is more appropriate.

Specifically, we upsample by deconvolution the feature maps from the last layers of the block 2, 3 and 4, as illustrated in Figure 2. Then we concatenate them to gather rich location information in lower layers and with stronger semantic information in higher layers. Pre-defined anchors [22] are used with specific scales and angles on this fused feature map. Then the classification and regression heads run on this feature map respectively to classify each anchor and regress the location of existing objects.

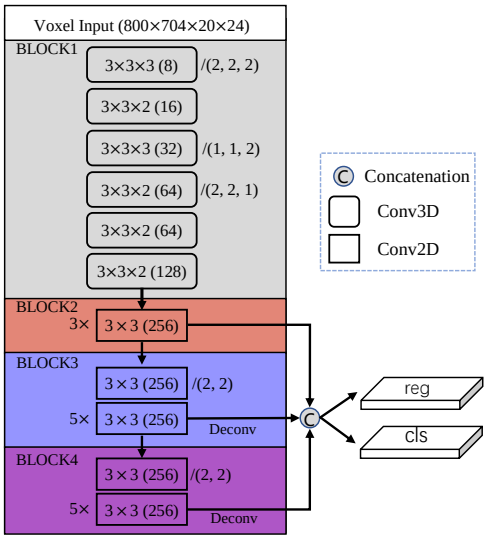


Figure 2. Network Structure of VoxelRPN. The format of layers used in the figure follows (kernel size)(channels) / (stride), *i.e.*, $(k_x, k_y, k_z)$(chn)/$(\mathrm{s}_x, \mathrm{s}_y, \mathrm{s}_z)$. The default stride is 1 unless otherwise specified.

### 3.3. RefinerNet

Although decent performance is achieved by VoxelRPN, We further improve the prediction quality through directly processing raw point cloud since the voxelization process and consecutively strided convolutions in the first block still lose an amount of localization information, which however can be supplemented by further feature enhancement in our RefinerNet.

RefinerNet makes use of the coordinates of point clouds. F-PointNet [25] is the pioneer work to utilize PointNet to regress 3D amodal bounding boxes from 2D detection results. Only interior points are used for inference without aware of context information. Our method, contrarily, also benefits from important context information.

**Box Feature** We use points in each bounding box prediction of VoxelRPN to generate box feature. Different from the two independent networks used in [25], we take not only coordinates but also features extracted from VoxelRPN as input. Convolutional feature maps from VoxelRPN capture local geometric structure of objects and gradually gather them in a hierarchical way, leading to a much larger receptive field to profit prediction. Then PointNet is applied to map each point to high-dimensional space and fuse point representation through max-pooling operation to gather information among all points with its context.

For each predicted bounding box from VoxelRPN, we first project it to BEV. Then all points around the region of BEV box are used as input, as illustrated in Figure 1.

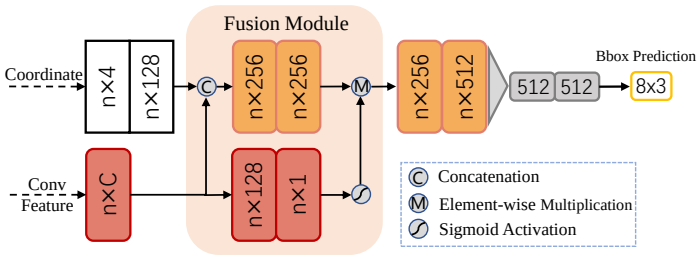


Figure 3. Network Structure of RefinerNet.

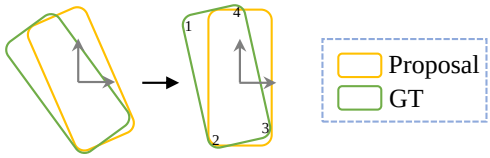


Figure 4. Canonization of a box. The number denotes the order of corner prediction in RefinerNet.

For each point $p$ with coordinate $(x_p, y_p)$ and feature map $F$ with size $(L_F, W_F, C_F)$, we define the corresponding feature as the feature vector with $C_F$ channels at location $(\lfloor \frac{x_p L_F}{L} \rfloor, \lfloor \frac{y_p W_F}{W} \rfloor)$. We grasp the final concatenation feature map from VoxelRPN with more comprehensive information.

Before feeding the coordinates of each point to the following network, we first canonize them for the purpose of guaranteeing the translation and rotation invariance. The coordinates of points within 0.3 meters around the proposal box are cropped and canonized by rotation and translation given the proposal box. As shown in Figure 3, we define the coordinate feature as the high-dimensional (128D) representation acquired via a MLP layer.

**Network Structure** With these two sources of features, we find a way to effectively fuse them. Instead of trivial concatenation, we design a new module with the attention mechanism for comprehensive feature generation. As illustrated in Figure 3, we first concatenate the high-dimensional coordinate feature with the convolutional feature. Then it is multiplied with the attention, generated by the convolutional features. What follows is a light-weight PointNet consisting of two MLP layers with max-pooling to aggregate all information in one box.

The final box refinement is achieved by two MLP layers to predict refined location of all box corner points based on proposals. As shown in Figure 4, when computing the regression target, the ground-truth box as well as point cloud are canonized by rotation and translation given the proposal box. This operation organizes ground-truth box corners in a specific order, which can reduce the uncertainty of the corner order caused by rotation. Our experiments manifest superiority of the canonized corner loss.

Without bells and whistles, this light-weight RefinerNet can already effectively improve the accuracy in box prediction, especially considering the $Z$ dimension and bounding boxes with higher IoUs in both 3D and BEV.

### 3.4. Network Training

Training our Fast Point R-CNN includes two steps. We first train VoxelRPN until convergence. Then the RefinerNet is trained based on the extracted features and inferred bounding boxes.

**VoxelRPN** In VoxelRPN, the anchors spread on each location of the global feature map. One anchor is considered as a positive sample if its IoU with ground-truth is higher than 0.6 in BEV. The regression target is the ground-truth bounding box with the highest IoU value. One anchor is considered as negative if its IoU value with all ground-truth boxes is lower than 0.45. We train VoxelRPN with a multi-task loss as

$$Loss = L_{cls} + L_{reg}, \tag{1}$$

where $L_{cls}$ is the classification binary cross entropy loss as

$$L_{cls} = \frac{1}{N_{pos}} \sum_i L_{cls}(p_i^{pos}, 1) + \frac{\gamma}{N_{neg}} \sum_i L_{cls}(p_i^{neg}, 0), \tag{2}$$

$$L_{cls}(p, t) = -(t \log(p) + (1-t) log(1-p)). \tag{3}$$

In our experiments, we use $\gamma = 10$. Due to the imbalanced distributions of positive and negative samples, we normalize their loss separately. OHEM [30] is applied to the negative term of the classification loss. Each anchor is parameterized as $(x_a, y_a, z_a, h_a, w_a, l_a, \theta_a)$ and the ground truth box is parameterized as $(x_g, y_g, z_g, h_g, w_g, l_g, \theta_g)$. For regression, we adopt parameterization following [43, 9] as

$$\begin{aligned} &\Delta_1 x = \frac{x_g - x_a}{d_a}, \Delta_1 y = \frac{y_g - y_a}{d_a}, \Delta_1 z = \frac{z_g - z_a}{h_a}, \\ &\Delta_1 h = \log(\frac{h_g}{h_a}), \Delta_1 w = \log(\frac{w_g}{w_a}), \Delta_1 l = \log(\frac{l_g}{l_a}), \\ &\Delta_1 \theta = \theta_g - \theta_a. \end{aligned} \tag{4}$$

The regression loss is defined as a smooth L1 loss of

$$L_{reg}(x) = \begin{cases} 0.5(\sigma x)^2, & \text{if } |x| < 1/\sigma^2 \\ |x| - 0.5/\sigma^2, & \text{otherwise} \end{cases} \tag{5}$$

where $\sigma$ is set to 3 in our experiments.

**RefinerNet** It is noticed that the recall of our VoxelRPN on 0.5 IoU thresh, in top 30 predicted boxes in Bird's Eve View (BEV), is over 95% for car. Our RefinerNet is for improving the quality of prediction boxes. We only train it on positive proposal boxes whose IoU with ground-truth is higher than 0.5 in BEV.

The regression target is defined as the offset from proposal center $(x_p, y_p, z_p)$ to 8 canonized corners $(x_{i,g}, y_{i,g}, z_{i,g}$ for $i = 1, ..., 8)$ of the target box as shown in Figure 4:

$$\Delta_2 x_i = x_{i,g} - x_p, \Delta_2 y_i = y_{i,g} - y_p, \Delta_2 z_i = z_{i,g} - z_p \tag{6}$$

This parameterization is a general and natural design for RefinerNet that processes directly on coordinates of points.

## 4. Experiments

We conduct experiments on the challenging KITTI [8] dataset in terms of 3D detection and BEV detection. Extensive ablation studies on our approach are conducted.

### 4.1. Experiment Setup

**Dataset and Evaluation Metric** The KITTI dataset provides 7,481 images and point clouds for training and 7,518 for testing. Note for evaluation on the test subset and comparison with other methods, we can only submit our result to the evaluation server. Following the protocol in [5, 43], we divide the training data into a training set (3,712 images and point clouds) with around 14,000 Car annotations and a validation set (3,769 images and point clouds). Ablation studies are conducted on this split. While for evaluation on test set, we train our model on the entire train set with 7k point clouds.

According to the occlusion/truncation level and the height of 2D boxes in images, evaluation on the KITTI dataset is split into three difficulty levels as "easy", "moderate" and "hard". The KITTI leaderboard ranks all methods according to $AP_{0.7}$ in "moderate" difficulty and takes it as the primary metric.

**Implementation Details** The point cloud is cropped to the range of $[0., 70.4] \times [-40., 40.] \times [-3., 1.]$ meters along $(X, Y, Z)$ axes respectively, following [5, 43]. The input to VoxelRPN is generated by voxelizing the point cloud into a 3D cuboid of size $800 \times 704 \times 20$, where each voxel is with size $0.1 \times 0.1 \times 0.2$ meter. As a result, the output convolutional feature map is with size $200 \times 176 \times 1$. 4 anchors are defined in each output location with different angles $(0°, 45°, 90°, 135°)$.

For the category of "car", we use the anchor size of $h_a = 1.73, w_a = 0.6, l_a = 0.8$ meters. NMS with IoU threshold 0.1 is applied to prediction from VoxelRPN to filter out duplicated predictions and help keep high efficiency of the RefinerNet. For the categories of *Pedestrian* and *Cyclist*, the network removes the downsampling in the fourth Conv3D layer since these two categories are much smaller than car category.

We use anchors of size $h_a = 1.73, w_a = 0.6, l_a = 0.8$ and $h_a = 1.73, w_a = 0.6, l_a = 1.76$ for *Pedestrian* and

*Cyclist* respectively. Like F-PointNet[25], multi-class prediction for RefinerNet is to concatenate predicted class label of VoxelRPN (one-hot encoding vector) with the feature after max-pooling operation and then refine box corners for all classes. We note that training on *Pedestrian* and *Cyclist* can improve their performance.

**Training Details** By default, models are trained on 8 NVIDIA P40 GPUs with batch-size 16 – that is, each GPU holds 2 point clouds. We apply ADAM [12] optimizer with an initial learning rate $0.01$ for training of VoxelRPN and RefinerNet. We train VoxelRPN for 70 epochs and the learning rate is decreased by 10 times at 50th and 65th epochs. Training of RefinerNet lasts for 70 epochs and the learning rate is decreased by 10 times at 40th, 55th and 65th epochs.

Batch Normalization is used following each parameter layer. A weight decay of $0.0001$ is used in both networks. Since the training of RefinerNet requires the convolutional feature from VoxelRPN, we train it for each frame instead of on objects, saving a large amount of computation.

**Data Augmentation** Multiple data augmentation strategies are applied during training in order to alleviate the overfitting problem considering the limited amount of training data. For each frame of the point cloud, we conduct left-right random flipping, random scaling with a uniformly sampled scale from $0.95 \sim 1.05$ and random rotation with a degree sampled from $-45^\circ \sim 45^\circ$ around the origin for entire scene of point clouds.

We also disturb each ground-truth bounding box and its corresponding interior points by random translation. Specifically, the shift is sampled from $\mathcal{N}(0, 1)$ for both $X$ and $Y$ axes and $\mathcal{N}(0, 0.3)$ for $Z$ axis. Random rotation around $Z$ axis is uniformly sampled from $-18^\circ \sim 18^\circ$. Note that there is a collision detection to prevent collision of different objects.

**MIXUP Augmentation** Similar to the spirit of [7, 42] in 2D object detection, we also augment input point clouds with cropped ground-truth from other point sets to greatly improve the convergence speed and quality. Instead of cropping solely interior points of each ground-truth box, we crop a larger region with extra 0.3 meters to better preserve the context information. With this regularization, cropped points and surrounding points are distributed more coherently with each other, making the network better capture the property of each object. In our setting, 20 objects are added in each frame of point clouds.

### 4.2. Main Results

As shown in Table 1, we compare Fast Point R-CNN with state-of-the-art approaches in 3D object detection and BEV object detection on KITTI test dataset. The official KITTI benchmark ranks different methods according to the performance on the moderate subset. Our model achieves state-of-the-art performance while accomplishing high efficiency (15FPS on NVIDIA Tesla P40 GPU). Note that SECOND [39] applies SparseConv [10] and PointPillars [16] used engineering techniques of NVIDIA TensorRT. These solutions are complementary to ours.

For better comparison, we reproduce VoxelNet [43] as a strong baseline network. It is noteworthy that our reproduction even yields much better results than those reported in [43]. As shown in Table 2, our proposed VoxelRPN outperforms VoxelNet in 3D object detection. Accompanied by RefinerNet, nearly twice as fast as VoxelNet, Fast Point R-CNN outperforms VoxelNet in both 3D object detection and BEV object detection. We show qualitative results in Figure 5. We can make good prediction at several challenging scenes.

## 5. Ablation Studies

We conduct extensive ablation study for each component based on the train/val. split.

### 5.1. VoxelRPN

To illustrate the effectiveness of VoxelRPN, we start with a fast and yet simple baseline and gradually add our proposed components. The baseline consists of only 2D convolutions and directly processes input voxel by encoding information along $Z$ axis into the channel dimension. The difference with VoxelRPN is that the first 6 Conv3D layers in the first block are replaced with 6 Conv2D layers. We keep the same kernel size in $X$ and $Y$ axes; the channels are 128 except the first layer with 64 channels. Two anchors with angles $0^\circ$ and $90^\circ$ are used. As shown in Table 3, the baseline achieves reasonable performance.

**More 3D Convolutions (Conv3D)** By replacing lower layers to 3D convolutions as illustrated in Figure 2 and processing the 3D voxels, we improve the baseline by nearly 1 point, manifesting the effectiveness of 3D convolutions on preserving the information, especially along $Z$ dimension. With this modification, the time cost only increases 5ms.

**Higher Resolution Input (HRI)** We also introduce the finer voxel, producing higher resolution grid input with size $800 \times 704 \times 20$, as described in Figure 2. Accordingly, we modify the stride of the first layer to 2 to effectively reduce the computation overhead. This technique can significantly improve the results without adding much computation.

**MIXUP Augmentation (MIXUP)** With MIXUP augmentation, we improve the performance with around 0.5 point. With MIXUP augmentation, we achieve comparable performance with only half of the original training epochs.

| Method | Input | Time (s) | 3D | | | BEV | | | GPU |
|---|---|---|---|---|---|---|---|---|---|
| | | | $AP_{easy}$ | $\mathbf{AP_{moderate}}$ | $AP_{hard}$ | $AP_{easy}$ | $\mathbf{AP_{moderate}}$ | $AP_{hard}$ | |
| MV3D [5] | L+I | 0.24 | 66.77 | 52.73 | 51.31 | 85.82 | 77.00 | 68.94 | TITAN X |
| AVOD-FPN [15] | L+I | 0.1 | 81.94 | 71.88 | 66.38 | 88.53 | 83.79 | **77.90** | TITAN XP |
| AVOD [15] | L+I | 0.1 | 73.59 | 65.78 | 58.38 | 86.80 | 85.44 | 77.73 | TITAN XP |
| F-PointNet [25] | L+I | 0.17 | 81.20 | 70.39 | 62.19 | 88.70 | 84.00 | 75.33 | GTX 1080 |
| ContFuse [20] | L+I | 0.06 | 82.54 | 66.22 | **64.04** | **88.81** | **85.83** | 77.33 | – |
| RoarNet [13] | L+I | 0.1 | **83.71** | **73.04** | 59.16 | 88.20 | 79.41 | 70.02 | TITAN X |
| IPOD [41] | L+I | 0.2 | 79.75 | 72.57 | 66.33 | 86.93 | 83.98 | 77.85 | Tesla P40 |
| VoxelNet [43] | L | 0.22 | 77.49 | 65.11 | 57.73 | **89.35** | 79.26 | 77.39 | TITAN X |
| PIXOR [40] | L | 0.1 | - | - | - | 84.44 | 80.04 | 74.31 | TITAN XP |
| SECOND [39] | L | 0.05 | 83.13 | 73.66 | 66.20 | 88.07 | 79.37 | 77.95 | GTX 1080Ti |
| PointPillars [16] | L | 0.016 | 79.05 | 74.99 | **68.30** | 88.35 | **86.10** | **79.83** | GTX 1080Ti |
| PointRCNN-deprecate [29] | L | 0.1 | 84.32 | 75.42 | 67.86 | 89.28 | 86.04 | 79.02 | TITAN XP |
| PointRCNN [29] | L | 0.1 | **85.94** | **75.76** | **68.32** | **89.47** | 85.68 | **79.10** | TITAN XP |
| Fast Point R-CNN | L | 0.065 | 84.28 | 75.73 | 67.39 | 88.03 | **86.10** | 78.17 | Tesla P40 |

Table 1. Comparison of main results on KITTI test set. Here 'L' denotes LiDAR input and 'I' denotes RGB image input.

| Method | Time (s) | 3D | | | BEV | | |
|---|---|---|---|---|---|---|---|
| | | $AP_{easy}$ | $\mathbf{AP_{moderate}}$ | $AP_{hard}$ | $AP_{easy}$ | $\mathbf{AP_{moderate}}$ | $AP_{hard}$ |
| VoxelNet (Paper) | 0.225 | 81.97 | 65.46 | 62.85 | 89.60 | 84.81 | 78.57 |
| VoxelNet (Reproduced) | 0.117 | 86.48 | 75.26 | 73.25 | 90.13 | 87.61 | 86.4 |
| VoxelRPN | 0.058 | 87.51 | 76.64 | 74.4 | 89.8 | 87.58 | 86.38 |
| Fast Point R-CNN | **0.065** | **89.12** | **79.00** | **77.48** | **90.12** | **88.10** | **86.24** |

Table 2. Comparison of main results on KITTI validation set.

| Conv3D | HRI | MIXUP | MA | 3D $AP_{0.7}$ (moderate) |
|---|---|---|---|---|
| - | - | - | - | 73.8 |
| ✓ | | | | 74.7 |
| ✓ | ✓ | | | 75.34 |
| ✓ | ✓ | ✓ | | 75.82 |
| ✓ | ✓ | ✓ | ✓ | 76.64 |

Table 3. Effectiveness of different techniques applied to VoxelRPN on KITTI val subset. The baseline network consists of only 2D convolutions. Conv3D denotes we replace the lower layers with 3D convolutions. HRI denotes high resolution input. MIXUP denotes the use of MIXUP augmentation. MA means anchors with 4 different angles are used instead of 2 of them.

| Fuse methods | 3D $AP_{0.7}$ (moderate) |
|---|---|
| Coordinate Feature | 77.82 |
| Convolution Feature | 76.90 |
| Concatenation | 78.38 |
| + Attention Module | 79.00 |

Table 4. Comparison of different fusion methods in RefinerNet.

**More Anchors (MA)** With $4$ anchors in angles $0°$, $45°$, $90°$ and $135°$ respectively, instead of using only 2 anchors, we further gain another $0.8$ point bonus. We find that the matching probability gain with ground-truth is significant with more anchors involved.

## 5.2. RefinerNet

**Input Features** We first investigate the importance of both coordinate and convolution features. As shown in Table 4, with only coordinate feature or convolution feature, the RefinerNet improves results over VoxelRPN. It is noticeable that the performance with coordinate feature as input is better than the one with convolution feature as input. This manifests that the accurate location information is lost in the quantization representation of point cloud and consecutive convolutional-and-downsampling operations.

**Feature Fusion** With the compensation of coordinate information, the performance boosts greatly. Much better performance is achieved with both coordinate and convolution features, since they provide semantically complementary information. We also compare our strategy of fusing these two sources of features with simple concatenation. Our fusion method with attention mechanism outperforms the alternative by 0.62 point, as shown in Table 4.

**Effect of Canonized Corner Loss** We compare parameterization of box prediction. The naive parameterization of 7 parameters as regression loss only achieves 78.45 in 3D $AP_{0.7}$. With canonized corner loss, it can further improve to 79.

**Comparison with RoI Align** One straightforward method for box refinement is to use RoI Align[11]. For comparison, we implement rotated RoI align that crops convolutional features from VoxelRPN given proposals. For the car class, we pool with size $8 \times 4$ along the

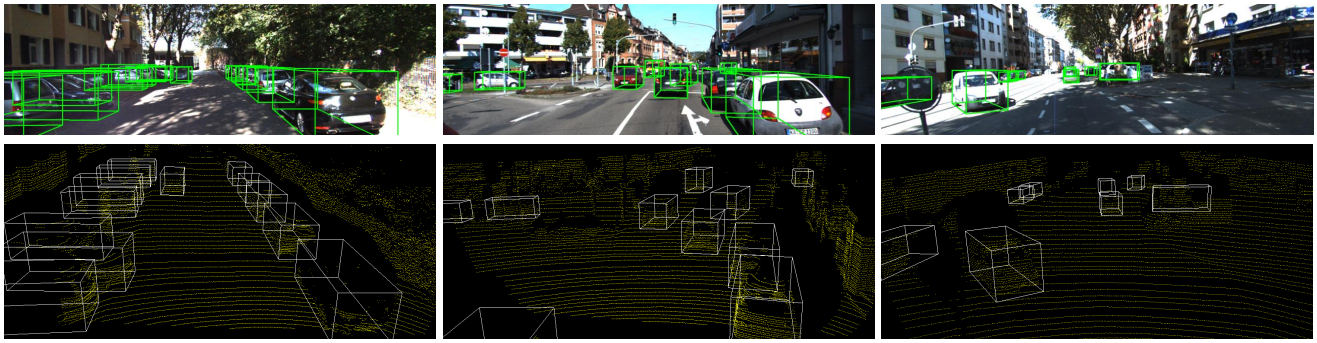

Figure 5. Visualization of our results.

| Method | Range (meters) | 3D (Moderate) | | BEV (Moderate) | |
|---|---|---|---|---|---|
| | | $AP_{0.7}$ | $AP_{0.8}$ | $AP_{0.7}$ | $AP_{0.8}$ |
| VoxelRPN | 0-30 | 88.39 | 58.81 | 90.22 | 83.32 |
| Fast Point R-CNN | 0-30 | 89.26 | 62.73 | 90.25 | 85.61 |
| VoxelRPN | 30-50 | 51.99 | 13.31 | 73.51 | 49.63 |
| Fast Point R-CNN | 30-50 | 58.41 | 15.39 | 73.9 | 50.05 |

Table 5. Comparison of nearby- and distant-object detection accuracy.

direction of car inside the rotated box region. Then two 4096D MLP layers are applied to perform classification and regression. Only the above operations are different – it achieves 77.39 with $AP_{0.7}$. Our RefinerNet performs better clearly. We conjecture that rotated RoI align still lacks precise localization information.

**Result Analysis** In the scenario of autonomous driving, faraway objects are with much less points due to the limited resolution of LiDAR and occlusion by nearby objects, making it more challenging to detect distant objects. As shown in Table 5, there is a large discrepancy between accuracy of nearby and faraway objects. It is noteworthy that RefinerNet significantly improves the performance of 3D detection accuracy of distant objects ranging from 30 to 50 meters, *i.e.*, from 51.99 to 58.41 with $AP_{0.7}$ metric. It is because distant objects generally possess only a small number of points. With only voxel representation, it is hard for VoxelRPN to fully capture the structure of objects. But with the profitable access to coordinate feature, RefinerNet can still infer the complete structure of objects and achieve better inference.

As shown in Tables 5 and 6, RefinerNet can further improve detection with higher quality, evaluated with $AP_{0.8}$, which demonstrates that RefinerNet better utilizes fine-grained localization information than VoxelRPN.

### 5.3. Experiments on Other Categories

KITTI benchmark provides limited annotations for *Pedestrian* and *Cyclist* categories. For reference, we provide results on these two classes. Following [43, 39], we train the network for these two categories. Our final results on *Pedestrian* and *Cyclist* are 63.05 and 64.32 respectively, with VoxelRPN results 60.78 and 62.41 on KITTI val dataset. We achieve comparable results on KITTI test data as listed in Table 7. We believe when more data is used, superiority of our two-stage network can be better demonstrated.

| Method | 3D (Moderate) | | | BEV (Moderate) | | |
|---|---|---|---|---|---|---|
| | $AP_{0.6}$ | $AP_{0.7}$ | $AP_{0.8}$ | $AP_{0.6}$ | $AP_{0.7}$ | $AP_{0.8}$ |
| VoxelRPN | 88.94 | 76.64 | 42.6 | 89.77 | 87.58 | 71.39 |
| Fast Point R-CNN | 89.14 | 79.0 | 52.95 | 89.86 | 88.10 | 74.58 |

Table 6. Detection results with different IoU thresholds.

| Method | $AP_{0.5}$ on *Pedestrian* | | $AP_{0.5}$ on *Cyclist* | |
|---|---|---|---|---|
| | 3D | BEV | 3D | BEV |
| PointPillars [16] | 43.53 | **50.23** | 59.07 | 62.25 |
| F-PointNet [25] | **44.89** | 50.22 | 56.77 | 61.96 |
| PointRCNN [29] | 41.78 | – | **59.60** | – |
| Fast Point R-CNN | 42.90 | 45.43 | 59.36 | **62.59** |

Table 7. Performance on *Pedestrian* and *Cyclist* on test set.

## 6. Conclusion

In this paper, we have proposed a generic, effective and fast two-stage framework for 3D object detection. Our method makes use of both voxel representation and raw point cloud to benefit from both of them. The first stage takes voxel representation as input and applies convolutional operations to acquire a set of initial predictions. Then the second stage further refines them based on raw point clouds and extracted convolution features.

With this conceptually simple but practically powerful design, our method is on par with existing solutions while maintaining higher detection speed. We believe our research shows a new way to properly utilize different dimensions of information for this challenging and yet practically fundamental task.

## References


[1] Waleed Ali, Sherif Abdelkarim, Mohamed Zahran, Mahmoud Zidan, and Ahmad El Sallab. Yolo3d: End-to-end real-time 3d oriented object bounding box detection from lidar point cloud. *arXiv:1808.02350*, 2018.

[2] Jorge Beltran, Carlos Guindel, Francisco Miguel Moreno, Daniel Cruzado, Fernando Garcia, and Arturo de la Escalera. Birdnet: a 3d object detection framework from lidar information. *arXiv:1805.01195*, 2018.

[3] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *CVPR*, 2016.

[4] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *NIPS*, 2015.

[5] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, 2017.

[6] Rethage Dario, Wald Johanna, Sturm Jrgen, Navab Nassir, and Tombari Federico. Fully-convolutional point networks for large-scale point clouds. In *ECCV*, 2018.

[7] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *ICCV*, 2017.

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

[9] Ross Girshick. Fast r-cnn. In *ICCV*, 2015.

[10] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. *CVPR*, 2018.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[13] Youngwook Paul Kwon Kiwoo Shin and Masayoshi Tomizuka. Roarnet: A robust 3d object detection based on region approximation refinement. *arXiv:1811.03818*, 2018.

[14] Tao Kong, Anbang Yao, Yurong Chen, and Fuchun Sun. Hypernet: Towards accurate region proposal generation and joint object detection. In *CVPR*, 2016.

[15] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. Joint 3d proposal generation and object detection from view aggregation. *IROS*, 2018.

[16] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *arXiv:1812.05784*, 2018.

[17] Bo Li. 3d fully convolutional network for vehicle detection in point cloud. In *IROS*, 2017.

[18] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3d lidar using fully convolutional network. *Robotics: Science and Systems*, 2016.

[19] Yangyan Li, Rui Bu, Mingchao Sun, and Baoquan Chen. Pointcnn. *NIPS*, 2018.

[20] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *ECCV*, 2018.

[21] Tsung-Yi Lin, Piotr Dollár, Ross B Girshick, Kaiming He, Bharath Hariharan, and Serge J Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[22] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[23] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *CVPR*, 2018.

[24] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS*, 2015.

[25] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *CVPR*, 2018.

[26] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CVPR*, 2017.

[27] Charles R. Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *CVPR*, 2016.

[28] Charles R. Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017.

[29] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. *arXiv:1812.04244*, 2018.

[30] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, 2016.

[31] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-yolo: Real-time 3d object detection on point clouds. *arXiv:1803.06199*, 2018.

[32] Shuran Song and Jianxiong Xiao. Sliding shapes for 3d object detection in depth images. In *ECCV*, 2014.

[33] Shuran Song and Jianxiong Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *CVPR*, 2016.

[34] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*, 2015.

[35] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Multi-view 3d models from single images with a convolutional network. In *ECCV*, 2016.

[36] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv:1801.07829*, 2018.

[37] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015.

[38] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. Pointfusion: Deep sensor fusion for 3d bounding box estimation. In *CVPR*, 2018.

[39] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 2018.

[40] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *CVPR*, 2018.

[41] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Ipod: Intensive point-based object detector for point cloud. *arXiv:1812.05276*, 2018.

[42] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2017.

[43] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, 2018.