AFDet: Anchor Free One Stage 3D Object Detection

Runzhou Ge* Zhuangzhuang Ding* Yihan Hu*
Yu Wang Sijia Chen Li Huang Yuan Li
Horizon Robotics
{runzhou.ge, zhuangzhuang.ding, yihan01.hu, yu04.wang}@horizon.ai

Abstract

High-efficiency point cloud 3D object detection operated on embedded systems is important for many robotics applications including autonomous driving. Most previous works try to solve it using anchor-based detection methods which come with two drawbacks: post-processing is relatively complex and computationally expensive; tuning anchor parameters is tricky. We are the first to address these drawbacks with an anchor free and Non-Maximum Suppression free one stage detector called AFDet. The entire AFDet can be processed efficiently on a CNN accelerator or a GPU with the simplified post-processing. Without bells and whistles, our proposed AFDet performs competitively with other one stage anchor-based methods on KITTI validation set and Waymo Open Dataset validation set.

1. Introduction

Detecting 3D objects in the point cloud is one of the most important perception tasks for autonomous driving. To satisfy the power and efficiency constraints, most of the detection systems are operated on vehicle embedded systems. Developing embedded systems friendly point cloud 3D detection system is a critical step to make autonomous driving a reality.

Due to the sparse nature of the point cloud, it is inefficient to directly apply 3D or 2D Convolution Neural Networks (CNN) [9, 28] on the raw point cloud. On one hand, lots of point cloud encoders [38, 4, 33, 11, 37] are introduced to encode the raw point cloud to data formats that could be efficiently processed by 3D or 2D CNN. On the other hand, some work [22, 32, 25, 35, 20] directly extract features from raw point clouds for 3D detection which is inspired by PointNet [23, 24]. But for the detection part, most of them adopt anchor-based detection methods proven effective in image object detection tasks.

*These authors contributed equally to this work.

	Anchor-based	AFDet (Ours)
Anchor Free	✗	✓
NMS Free	✗	✓
Post-processing Friendly	✗	✓
Embedded Systems Friendly	✗	✓

Table 1. The comparison between anchor-based methods and our method. We use max pooling and AND operation to achieve a similar functionality with NMS but with a much higher speed. In our experiments, our max pooling and AND operation can achieve 2.5×10^{-5} s on one Nvidia 2080 Ti GPU which is approximately $1000\times$ faster than the CPU implemented NMS.

Anchor-based methods have two major disadvantages. First, Non-Maximum Suppression (NMS) is necessary for anchor-based methods to suppress the overlapped high confident detection bounding boxes. But it can introduce non-trivial computational cost especially for embedded systems. According to our experiments, it takes more than 20 ms to process one KITTI [6] point cloud frame even on a modern high-end desktop CPU with an efficient implementation, let alone CPUs typically deployed for embedded systems. Second, anchor-based methods requires anchor selection which is tricky and time-consuming, because critical parts of the tuning can be a manual trial and error process. For instance, every time a new detection class is added to the detection system, hyper parameters such as appropriate anchor number, anchor size, anchor angle and anchor density need to be selected.

Can we get rid of NMS and design an embedded system friendly anchor free point cloud 3D detection system with high efficiency? Recently, anchor free methods [12, 36, 31] in image detection have achieved remarkable performance. In this work, we propose an anchor free and NMS free one stage end-to-end point cloud 3D object detector (AFDet) with simple post-processing.

We use PointPillars [11] to encode the entire point cloud into pseudo images or image-like feature maps in Bird's Eye View (BEV) in our experiments. However, AFDet can be used with any point cloud encoder which generates

pseudo images or image-like 2D data. After encoding, a CNN with upsampling necks is applied to output the feature maps, which connect to five different heads to predict object centers in the BEV plane and to regress different attributes of the 3D bounding boxes. Finally, the outputs of the five heads are combined together to generate the detection results. A keypoint heat map prediction head is used to predict the object centers in the BEV plane. It will encode every object into a small area with a heat peak as its center. At the inference stage, every heat peak will be picked out by max pooling operation. After this, we no longer have multiple regressed anchors tiled into one location, therefore there is no need to use traditional NMS. This makes the entire detector runnable on a typical CNN accelerator or GPU, saving CPU resources for other critical tasks in autonomous driving.

Our contributions can be summarized as below:

(1) We are the first to propose an anchor free and NMS free detector for point cloud 3D object detection with simplified post-processing.

(2) AFDet is embedded system friendly and can achieve high processing speed with much less engineering effort.

(3) AFDet can achieve competitive accuracy compared with previous single-stage detectors on the KITTI validation set. A variant of our AFDet surpasses the state-of-the-art single-stage 3D detection methods on Waymo validation set.

In the following, we first discuss related work in Section 2. Then we show more details of our method in Section 3. Finally, we analyze and compare AFDet with other approaches in Section 4.

2. Related Work

Thanks to accurate 3D spatial information provided by LiDAR, LiDAR-based solutions prevail in 3D object detection task.

2.1. LiDAR-based 3D Object Detection

Due to non-fixed length and order, point clouds are in a sparse and irregular format which needs to be encoded before input into a neural network. Some works utilize mesh grid to voxelize point clouds. Features, such as density, intensity, height *etc*., are concatenated in different voxels as different channels. Voxelized point clouds are either projected to different views such as BEV, Range View (RV) *etc*., to be processed by 2D convolution [4, 10, 27, 34] or kept in 3D coordinates to be processed by sparse 3D Convolution [29]. PointNet [23] proposes an effective solution to use raw point cloud as input to conduct 3D detection and segmentation. PointNet wields Multilayer Perceptron (MLP) and max pooling operation to solve point cloud's disorder and non-uniformity and provides satisfactory performance. Successive 3D detection solutions based on the raw point cloud input provide promising performance such as PointNet++ [24], Frustum PointNet [22], PointRCNN [14] and STD [35]. VoxelNet [38] combines voxelization and PointNet to propose Voxel Feature Extractor (VFE) in which a PointNet style encoder is implemented inside each voxel. A similar idea is used in SECOND [33] despite that sparse 3D convolution is utilized to further extract and downsample information in z-axis following VFE. VFE improves the performance of the LiDAR-based detector dramatically, however, with encoders that are learned from data, the detection pipeline becomes slower. PointPillars [11] proposes to encode point cloud as pillars instead of voxels. As a result, the whole point cloud becomes a BEV pseudo image whose channels are equivalent to VFE's output channels instead of 3.

Anchor free. In anchor-based methods, pre-defined boxes are provided for bounding box encoding. However, using dense anchors lead to exhaustive numbers of potential target objects, which makes NMS an unavoidable issue. Some previous work [34, 18, 2, 25, 21] mention anchor free concepts. PointRCNN [25] proposes a 3D proposal generation sub-network without anchor boxes based on whole-scene point cloud segmentation. VoteNet [21] constructs 3D bounding boxes from voted interest points instead of predefined anchor boxes. But all of them are not NMS free, which makes them less efficient and is not friendly to the embedded systems. Besides, PIXOR [34] is a BEV detector rather than a 3D detector.

2.2. Camera-based 3D Object Detection

Camera-based solutions thrived in accordance with the willingness of reducing cost. With more sophisticated networks being designed, camera-based solutions are catching up rapidly with LiDAR-based solutions. MonoDIS [26] leverages a novel disentangling transformation for 2D and 3D detection losses and a novel self-supervised confidence score for 3D bounding boxes. It gets top ranking on nuScenes [1] 3D object detection challenge. CenterNet [36] predicts the location and class of an object from the center of its bounding box on a feature map. Though originally designed for 2D detection, CenterNet also has the potential to conduct 3D detection with a mono camera. TTFNet [16] proposes techniques to shorten training time and increase inference speed. RTM3D [13] predicts nine perspective keypoints of a 3D bounding box in image space and recover the 3D bounding box with geometric regulation.

3. Methods

In this section, we present the details of AFDet from three aspects: point cloud encoder, backbone and necks, and anchor free detector. The framework is shown in Figure 1.

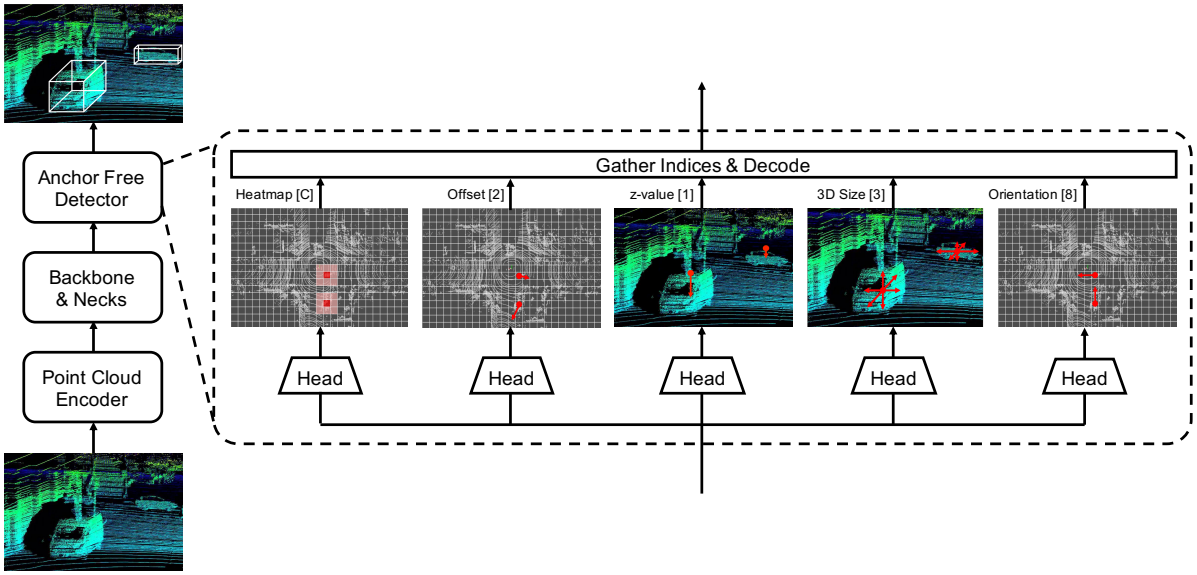

Figure 1. The framework of anchor free one stage 3D detection (AFDet) system and detailed structure of anchor free detector. The whole pipeline consists of the point cloud encoder, the backbone and necks, and the anchor free detector. The number in the square brackets indicate the number of last convolution layer's output channels. C is the number of categories used in the detection. Better viewed in color and zoomed in for details.

3.1. Point Cloud Encoder

To further tap the efficiency potential of our anchor free detector, we use PointPillars [11] as the point cloud encoder because of its fast speed. First, the detection range is discretized into pillars in the Bird's Eye View (BEV) plane which is also the x-y plane. Different points are assigned to different pillars based on their x-y values. Every point would also be augmented to $D = 9$ dimensional at this step. Second, the pre-defined P amount of pillars with enough number of points would be applied with a linear layer and a max operation to create an output tensor of size $F \times P$ where F is the number if output channels of the liner layer in PointNet [23]. Since P is the number of selected pillars, they are not one-to-one correspondent with the original pillars in the entire detection range. So the third step is to scatter the selected P pillars to their original location on the detection range. After that we can get a pseudo image $I \in \mathbb{R}^{W \times H \times F}$ where W and H indicate the width and height, separately.

Although we use PointPillars [11] as the point cloud encoder, our anchor free detector is compatible with any point cloud encoders which generate pseudo images or image-like 2D data.

3.2. Anchor Free Detector

Our anchor free detector consists of five heads. They are keypoint heatmap head, local offset head, z-axis location head, 3D object size head and orientation head. Figure 1 shows some details of the anchor free detector.

Object localization in BEV. For heatmap head and offset head, we predict a keypoint heatmap $\hat{M} \in \mathbb{R}^{W \times H \times C}$ and a local offset regression map $\hat{O} \in \mathbb{R}^{W \times H \times 2}$ where C is the number of keypoint types. The keypoint heatmap is used to find where the object center is in BEV. The offset regression map is to help the heatmap to find the more accurate object centers in BEV and also help to recover the discretization error caused by the pillarization process.

For a 3D object k with category c_k, we parameterize its 3D ground truth bounding box as $(x^{(k)}, y^{(k)}, z^{(k)}, w^{(k)}, l^{(k)}, h^{(k)}, \theta^{(k)})$ where $x^{(k)}$, $y^{(k)}$, $\theta^{(k)}$ represent the center location in LiDAR coordinate system, $w^{(k)}$, $l^{(k)}$, $h^{(k)}$ are the width, length and height of the bounding box, and $\theta^{(k)}$ is the yaw rotation around z-axis which is perpendicular to the ground. Let $[(back, front), (left, right)]$ denote the detection range in x-y plane. To be specific, $back$ and $front$ is along the x-axis and $left$ and $right$ is along the y-axis in the LiDAR coordinate system. In this work, the pillar in x-y plane is always a square. So let b denote the pillar side length. Following [12], for each object center we have the keypoint $p = \left(\frac{x^{(k)} - back}{b}, \frac{y^{(k)} - left}{b}\right) \in \mathbb{R}^2$ in BEV pseudo image coordinate. $\tilde{p} = \lfloor p \rfloor$ is its equivalent in the keypoint heatmap where $\lfloor \cdot \rfloor$ is the floor operation. The 2D bounding box in BEV could be expressed as $\left(\frac{x^{(k)} - back}{b}, \frac{y^{(k)} - left}{b}, \frac{w^{(k)}}{b}, \frac{l^{(k)}}{b}, \theta^{(k)}\right)$.

For each pixel (x, y) which are covered in the 2D bounding boxes in the pseudo image, we set its value in the

heatmap following

$$M_{x,y,c} = \begin{cases} 1, & \text{if } d = 0 \\ 0.8, & \text{if } d = 1 \\ \frac{1}{d}, & \text{else} \end{cases} \tag{1}$$

where d is the Euclidean distance calculated between the bounding box center and the corresponding pixel in the discretized pseudo image coordinates. A prediction $\hat{M}_{x,y,c} = 1$ represents the object center and $\hat{M}_{x,y,c} = 0$ indicates this pillar is background.

$\tilde{p}$, which represents the object centers in BEV, would be treated as positive samples while all other pillars would be treated as negative samples. Following [12, 36], we use the modified focal loss [15]

$$\mathcal{L}_{heat} = -\frac{1}{N} \sum_{x,y,c} \begin{cases} \left(1 - \hat{M}_{x,y,c}\right)^{\alpha} \\ \quad \log\left(\hat{M}_{x,y,c}\right), & \text{if } M_{x,y,c} = 1 \\ \left(1 - M_{x,y,c}\right)^{\beta} \left(\hat{M}_{x,y,c}\right)^{\alpha} \\ \quad \log\left(1 - \hat{M}_{x,y,c}\right), & \text{else} \end{cases} \tag{2}$$

to train the heatmap where N is the number of object in the detection range and α and β are the hyper parameters. We use $\alpha = 2$ and $\beta = 4$ in all our experiments.

For the offset regression head, there are two main functions. First, it is used to eliminate the error caused by the pillarization process in which we assign the float object centers to integer pillar locations in BEV as we mentioned above. Second, it plays an important role to refine the heatmap object centers' prediction especially when the heatmap predicts wrong centers. To be specific, once the heatmap predicts a wrong center which is several pixels away from the ground truth center, the offset head has the capability to mitigate and even eliminate several pixels' error to the ground truth object center.

We select a square area with the radius r around object center pixel in the offset regression map. The farther the distance to the object center is, the larger the offset value becomes. We train the offset using L_1 loss

$$\mathcal{L}_{off} = \frac{1}{N} \sum_{p} \sum_{\delta=-r}^{r} \sum_{\epsilon=-r}^{r} \left| \hat{O}_{\tilde{p}} - b\left(p - \tilde{p} + (\delta, \epsilon)\right) \right| \tag{3}$$

where the training is only for the square area with side length $2r + 1$ around the keypoint locations $\tilde{p}$. We will discuss more about the offset regression in Section 4.

***z*-axis location regression.** After the object localizations in BEV, we only have object x-y location. Thus we have the z-axis location head to regress the z-axis values. We directly regress z-value $\hat{Z} \in \mathbb{R}^{W \times H \times 1}$ using L_1 loss

$$\mathcal{L}_z = \frac{1}{N} \sum_{k=1}^{N} \left| \hat{Z}_{p^{(k)}} - z^{(k)} \right|. \tag{4}$$

Size regression. Additionally, we regress the object sizes $\hat{S} \in \mathbb{R}^{W \times H \times 3}$ directly. For each object, we have $s^{(k)} = (w^{(k)}, l^{(k)}, h^{(k)})$. The training loss for size regression is

$$\mathcal{L}_{size} = \frac{1}{N} \sum_{k=1}^{N} \left| \hat{S}_{p^{(k)}} - s^{(k)} \right| \tag{5}$$

which is also the L_1 loss.

Orientation prediction. Orientation $\theta^{(k)}$ for object k is the scalar angle rotated around z-axis which is perpendicular to the ground. We follow [19, 36] to encode it to an eight scalars with four scalars for each bin. Two scalars are for the softmax classification and the other two are for the angle regression. The angle ranges for two bins are $\Psi_1 = \left[-\frac{7\pi}{6}, \frac{\pi}{6}\right]$ and $\Psi_2 = \left[-\frac{\pi}{6}, \frac{7\pi}{6}\right]$ which overlap slightly. For each bin, we predict $\hat{\mu}_i^{(k)} \in \mathbb{R}^2$ which are used for softmax classification and $\hat{\nu}_i^{(k)} \in \mathbb{R}^2$ which are used for calculating sin and cos value of the offset to the bin center γ_i. The classification part $\hat{\mu}_i^{(k)}$ is trained with softmax while the offset part $\hat{\nu}_i^{(k)}$ is trained with L_1 loss. So the loss for the orientation training is

$$\mathcal{L}_{ori} = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{2} \left(\text{softmax}\left(\hat{\mu}_i^{(k)}, \eta_i^{(k)}\right) + \eta_i^{(k)} \left| \hat{\nu}_i^{(k)} - \nu_i^{(k)} \right| \right) \tag{6}$$

where $\eta_i^{(k)} = \mathbb{1}\left(\theta^{(k)} \in \Psi_i\right)$ in which $\mathbb{1}$ is the indicator function, and $\nu_i^{(k)} = \left(\sin\left(\theta^{(k)} - \gamma_i\right), \cos\left(\theta^{(k)} - \gamma_i\right)\right)$. We can decode the predicted orientation value using

$$\hat{\theta}^{(k)} = \arctan2\left(\hat{\nu}_{j,1}^{(k)}, \hat{\nu}_{j,2}^{(k)}\right) + \gamma_j \tag{7}$$

where j is the bin index with the larger classification score for object k.

Loss. We have described the losses for each head. The overall training objective is

$$\mathcal{L} = \mathcal{L}_{heat} + \lambda_{off}\mathcal{L}_{off} + \lambda_z\mathcal{L}_z + \lambda_{size}\mathcal{L}_{size} + \lambda_{ori}\mathcal{L}_{ori} \tag{8}$$

where λ represents the weight for each heads. For all regression heads including local offset, z-axis location, size, orientation regression, we only regress N objects which are in the detection range.

Gather indices and decode. At the training stage, we do not do back-propagation for the entire feature maps. Instead, we only back-propagate the indices that are the object centers for all regression heads. At the inference stage, we use max pooling and AND operation to find the peaks in the predicted heatmap following [36] which is much faster and more efficient than IoU-based NMS.

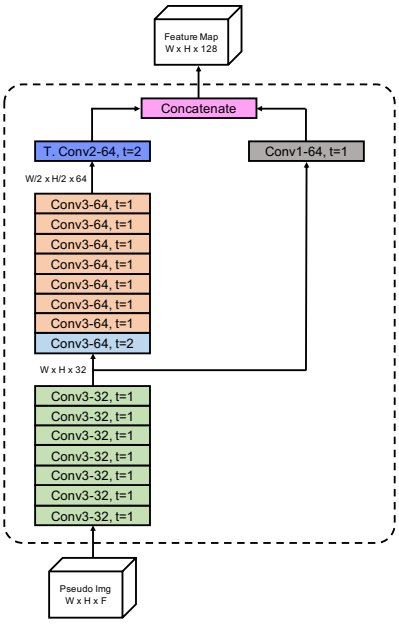

Figure 2. The backbone and necks we used in KITTI [6] detection. Different colors represent different operations with different parameters. The pseudo image is from point cloud encoder. t represents stride. F is the number of channels of the pseudo image. W and H are the width and height, separately. T. Conv is short for transposed convolution. Better viewed in color.

After the max pooling and AND operation, we can easily gather the indices of each center $(\hat{x}, \hat{y})$ from the keypoint heatmap. Let $\hat{\Phi}$ denote the set of detected BEV object centers. We have $\hat{\Phi} = \left\{\left(\hat{x}^{(k)}, \hat{y}^{(k)}\right)\right\}_{k=1}^{n}$ where n is the total number of detected objects. Then the final object center in BEV would be $\left(b\left(\hat{x}^{(k)} + 0.5\right) + \hat{o}_1^{(k)}, b\left(\hat{y}^{(k)} + 0.5\right) + \hat{o}_2^{(k)}\right)$ where $\left(\hat{o}_1^{(k)}, \hat{o}_2^{(k)}\right)$ are found in the $\hat{O} \in \mathbb{R}^{W \times H \times 2}$ using the index $\left(\hat{x}^{(k)}, \hat{y}^{(k)}\right)$. For all other prediction values, either they are directly from the regression results or we have mentioned the decoding process above. The predicted bounding box for object k is

$$\left(b\left(\hat{x}^{(k)} + 0.5\right) + \hat{o}_1^{(k)}, b\left(\hat{y}^{(k)} + 0.5\right) + \hat{o}_2^{(k)}, \hat{z}^{(k)}, \hat{w}^{(k)}, \hat{l}^{(k)}, \hat{h}^{(k)}, \hat{\theta}^{(k)}\right). \tag{9}$$

3.3. Backbone and Necks

In this work, we make several key modifications to the backbone used in [38, 33, 11] to support our anchor free detector. The network includes the backbone part and the necks part. The backbone part is similar to the network used in the classification tasks [28] which is used to extract features while downsampling the spatial size through different blocks. The necks part is used to upsample the features to make sure all outputs from different blocks of the backbone have the same spatial size so that we can concatenate them along one axis. Figure 2 shows details of the backbone and necks.

First, we reduce the backbone [38, 33, 11] from 3 blocks to 2 blocks. A block $\mathcal{B}(T, E, A)$ consists of E convolution layers with A output channels, each followed by a BatchNorm [8] and a ReLU. T is defined as the downsampling stride for this block. By reducing the blocks' number from 3 to 2, we remove the feature maps that are downsampled 4 times in [38, 33, 11]. We accordingly reduce the upsampling necks $\mathcal{V}(T, A)$ from 3 to 2. Each upsampling neck contains one transposed convolution with A output channels and T upsampling stride followed by BatchNorm and ReLU. Second, the first block we use is $\mathcal{B}(1, 8, 32)$ which does not downsample the output feature size compared with input size.

So the final backbone and necks consists of two blocks $\mathcal{B}(1, 7, 32)$ and $\mathcal{B}(2, 8, 64)$ followed by two upsampling necks $\mathcal{V}(1, 64)$, $\mathcal{V}(2, 64)$, separately. By doing this, the width and height of the input feature maps and the pseudo images are the same. In one word, in the process of generating feature maps we do not downsample, which is critical to maintaining a similar detection performance with [11] for KITTI [6] dataset. Reducing downsampling stride will only increase FLOPs, so we also reduce the number of filters in the backbone and necks. It turns out that we have fewer FLOPs in the backbone and necks than [38, 33, 11]. We will talk more about the backbone and necks in Section 4.

4. Experiments

In this section, we first introduce the two datasets. Then we describe the experiment settings and our data augmentation strategy. Finally, we show the performance on KITTI [6] validation set and some preliminary results on Waymo [30] validation set.

4.1. Datasets

KITTI object detection dataset [6] consists of $7,481$ training samples with both calibrations and annotations and $7,518$ test samples which only have calibrations. In our experiments, we split the official $7,481$ training samples into a training set comprising $3,712$ samples and a validation set with the rest $3,769$ samples following [3]. KITTI dataset provides both LiDAR point clouds and images, however, annotations are only labeled in the camera field of view (FOV). To accelerate the training process, we crop out points that are in camera FOV for training and evaluation [4, 38].

Waymo Open Dataset (Waymo OD) [30] is a newly released large dataset for autonomous driving. It consists of

798 training sequences with around $158,361$ samples and 202 validation sequences with around $40,077$ samples. Unlike KITTI where only the objects in camera FOV are labeled, the objects in Waymo are labeled in the full $360°$ field.

4.2. Experiments Settings

Unless we explicitly indicate, all parameters showing here are their default values. We use AdamW [17] optimizer with one-cycle policy [7]. We set learning rate max to 3×10^{-3}, division factor to 2, momentum ranges from 0.95 to 0.85, fixed weight decay to 0.01 to achieve convergence. The weight we use for different sub-losses are $\lambda_{off} = 1.0$, $\lambda_z = 1.5$, $\lambda_{size} = 0.3$ and $\lambda_{ori} = 1.0$. For the following part, we first introduce the parameters used in KITTI [6]. Then we introduce the Waymo OD parameters that are different from KITTI.

For KITTI car detection, we set detection range as $[(0, 70.4), (-40, 40), (-3, 1)]$ along x, y, z axes respectively. So the pseudo images are $I \in \mathbb{R}^{416 \times 480 \times 64}$. This range is the same as PointPillars [11] settings for a fair comparison. We use the max number of objects 50 which means at most we detect 50 objects for each class. For PointPillars encoder [11], we use pillar side length 0.16 m, max number of points per pillar 100 and max number of pillars $P = 12000$. We set the number of output channels of the linear layer in the encoder to 64. For the backbone, all the convolution layers are with kernel size 3. Their stride and number of output filters are shown in Figure 2. So the outputs of the backbone and necks are with shape $W \times H \times 128$ which have the same width and height with the pseudo images. For every head, we use two convolution layers: the first convolution layer is with kernel size 3 and channel number 32; the second convolution layer is with kernel size 1. Channel numbers are different for different heads which are shown in Figure 1. For offset regression head, we use $r = 2$ as default which means we will regress a square area with side length 5. We use max pooling with kernel size 3, stride 1 and apply AND operation between the feature map before and after the max pooling to get the peaks of the keypoint heatmaps at the inference stage. So we do not need NMS to suppress overlapped detections. The model is trained for 240 epochs. Due to the small size of the KITTI [6] dataset, we run every experiment 3 times and select the best one on the validation set.

For Waymo OD vehicle detection, we set detection range as $[(-76.8, 76.8), (-76.8, 76.8), (-3, 5)]$. The max number of objects is set to 200. The two convolution layers in every head are with channel number 64. For Waymo OD, we use the same backbone as [38, 33, 11].

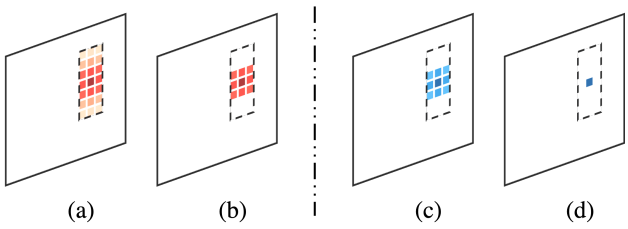

Figure 3. (a) is the car shape heatmap prediction method and (b) is the Gaussian kernel heatmap prediction method. (c) is our offset regression method and (d) is the regression method in [36]. The left two rectangles represent the heatmap outputs and the right two rectangles represent the offset regression outputs. The dashed line rectangles indicate the 2D bounding boxes.

Methods	3D AP IoU=0.7		
	Mod	Easy	Hard
Gaussian Kernel	72.50	82.57	68.91
Car Shape (Ours)	**75.57**	**85.68**	**69.31**
$r = 0$ [36]	74.51	84.45	69.03
$r = 1$	74.76	85.16	68.84
$r = 2$	**75.57**	**85.68**	**69.31**
$r = 3$	73.63	78.80	68.53

Table 2. The comparison between the two heatmap prediction methods and the comparison for different regression area radius.

4.3. Data Augmentation

First, we generate a database containing the labels of all ground truths and their associated point cloud data. For each sample, we randomly select 15 ground truth samples for car/vehicle and place them into the current point cloud. After this, we increase the number of ground truth in one point cloud. Second, each bounding box and the points inside it are rotated following the uniform distribution and translated following the normal distribution. The rotation follows $\mathcal{U}\left(-\frac{\pi}{20}, \frac{\pi}{20}\right)$ around z-axis. The translation follows $\mathcal{N}(0, 0.25)$ for all axes. Third, we also do randomly flip along z-axis [34], global rotation following $\mathcal{U}\left(-\frac{\pi}{4}, \frac{\pi}{4}\right)$ and global scaling [38, 33, 11].

4.4. Evaluation on KITTI Validation Set

We follow the official KITTI evaluation protocol to evaluate our detector, where the IoU threshold is 0.7 for the car class. We compare different methods or variants using average precision (AP) metric.

We first compare different heatmap prediction methods and different offset regression methods. Then we compare the different backbone for our detector. Finally, we compare our method with PointPillars.

Heatmap prediction. We compare our car shape heatmap prediction method with the Gaussian heatmap pre-

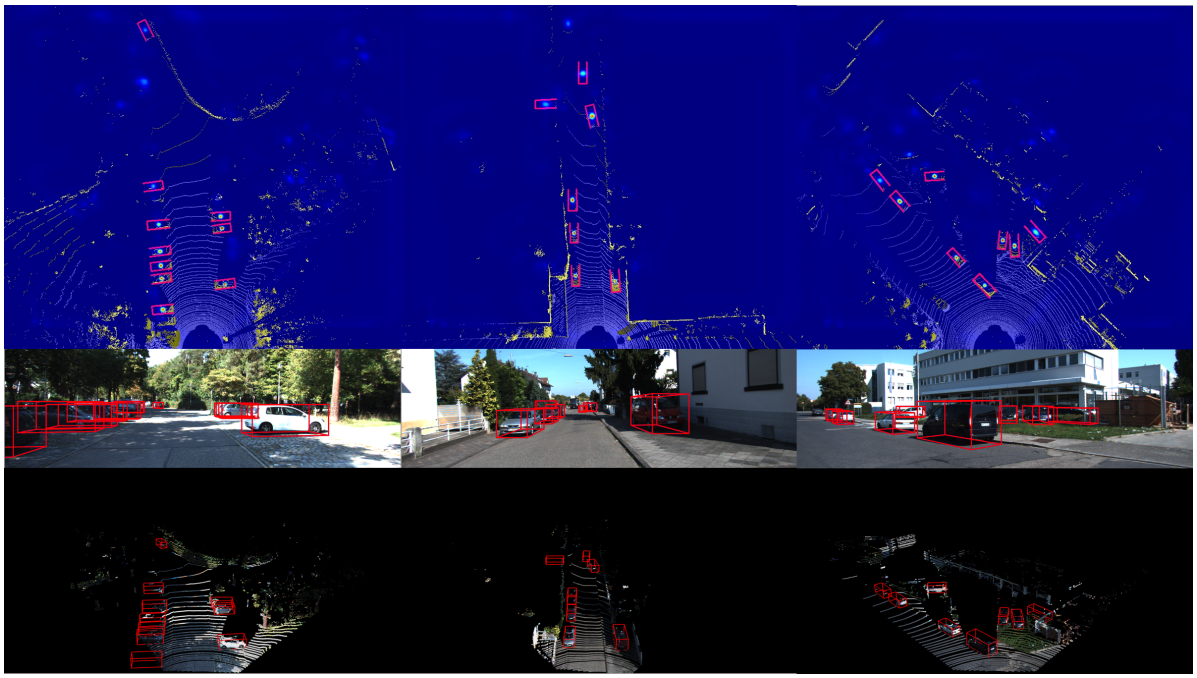

Figure 4. Results visualization on KITTI car detection with AFDet. Each one consists of heatmap, projection results in 2D RGB image and 3D point cloud results from top to bottom. Better viewed in color and zoomed in for details.

Methods	# Params	# MACs	Anchor Free	3D AP IoU=0.7			BEV AP IoU=0.7		
				Mod	Easy	Hard	Mod	Easy	Hard
PointPillars [11]	$4.81M$	$62.22G$	✗	76.04	83.73	69.12	86.34	89.68	84.38
$\mathcal{B}(2,4,64)+\mathcal{B}(2,6,128)+\mathcal{B}(2,6,256)$	$5.91M$	$125.37G$	✓	72.62	81.01	67.47	82.72	87.10	78.97
$\mathcal{B}(1,4,64)+\mathcal{B}(2,6,128)+\mathcal{B}(2,6,256)$	$5.91M$	$501.46G$	✓	75.33	85.18	69.18	84.69	88.91	79.83
$\mathcal{B}(1,7,32)+\mathcal{B}(2,8,64)$	$0.56M$	$76.53G$	✓	75.57	85.68	69.31	85.45	89.42	80.56

Table 3. The KITTI [6] validation set car detection performance comparison between different variants of AFDet and reimplemented PointPillars. The # parameters and # MACs are calculated on the entire network including backbone and necks and detector but except for the point cloud encoder. The # parameters and # MACs in the point cloud encoder are same for all listed methods above.

diction method [36]. For the car shape heatmap prediction, we have described in Section 3. For the Gaussian heatmap prediction, we splat all ground truth keypoints onto a heatmap $M \in \mathbb{R}^{W \times H \times C}$ using a Gaussian kernel $M_{x,y,c} = \exp\left(-\frac{(x-\tilde{p}_x)^2+(y-\tilde{p}_y)^2}{2\sigma_p^2}\right)$ where σ_p is the size adaptive standard deviation from [12]. The biggest difference between the two methods is the number of non-zero predictions in the heatmap. For the Gaussian kernel method, the non-zero predictions are only several pixels (*e.g.* 9 pixels) around the object center in the heatmap. While for the car shape method, all pixels in the 2D bounding box (car shape in BEV view) are non-zero. The illustration could be found in Figure 3 (a) and (b). From Table 2, we can see that predicting the entire car shape rather than the Gaussian kernel can improve about 2% on moderate difficulty.

Offset regression. To verify the effectiveness of our proposed offset regression method in which the training is for the square area with side length $2r+1$ around the object center $\tilde{p}$, we compare it with the offset regression method proposed in [36] in which the training is only for the object center $\tilde{p}$. Actually the latter regression method [36] is a special case of our method when r equals 0. The illustration of two methods is shown in 3 (c) and (d). We set r to 0, 1, 2 and 3. From Table 2, we can see that by setting r to 2. We can achieve 1 AP improvement over the regression method mentioned in [36].

Backbone and necks. We made some modifications in the backbone and necks for KITTI [6]. The baseline of our method is termed as $\mathcal{B}(2,4,64)+\mathcal{B}(2,6,128)+\mathcal{B}(2,6,256)$ in Table 3 which is same to [11].

First, the backbone used in [11] is downsampled $3\times$ with stride 2 for each block. After upsampling, the feature map size used in the detection head is downsampled by $2\times$ compared with the pseudo images. We remove the first down-

Methods	Anchor Free	# Epochs	LEVEL_1 3D AP IoU=0.7			
			Overall	0 - 30m	30 - 50m	50m - ∞
StarNet [20]	✗	75	53.70	-	-	-
PointPillars[1] [11]	✗	100	56.62	81.01	51.75	27.94
PPBA [5]+PointPillars	✗	-	62.44	-	-	-
MVF [37]	✗	100	62.93	86.30	60.02	**36.02**
AFDet+PointPillars-0.16 (Ours)	✓	16	58.77	84.99	55.76	24.78
AFDet+PointPillars-0.10 (Ours)	✓	16	**63.69**	**87.38**	**62.19**	29.27

Table 4. The vehicle detection performance comparison for single-stage 3D detection methods on Waymo OD validation set.

sampling stride and keep the following downsampling stride which is shown as $\mathcal{B}(1,4,64)+\mathcal{B}(2,6,128)+\mathcal{B}(2,6,256)$ in Table 3. The feature map sizes to the detector are the same as the pseudo images. We can see that the performance improves around 2% compared with the baseline. But the # MACs improve from $125.37G$ to $501.46G$ which is about $4\times$ of calculation of the baseline. This is mainly caused by doubling the feature maps' width and height.

Second, by modifying downsampling stride the performance improves. But we need to make sure that the performance improvement comes from enlarging the feature map size rather than from increasing computation. So we reduce the number of downsampling blocks from 3 to 2, in which we remove the last downsampling block. We also halve the number of output filters in the convolution layers. This computation reducing modification is shown as $\mathcal{B}(1,7,32)+\mathcal{B}(2,8,64)$ in Table 3. We can see that the performance has nearly no change by reducing the computation. From $\mathcal{B}(1,4,64)+\mathcal{B}(2,6,128)+\mathcal{B}(2,6,256)$ to $\mathcal{B}(1,7,32)+\mathcal{B}(2,8,64)$, we reduce about 84% # MACs and about 90% # parameters. So enlarging the feature map in our anchor free detector helps to improve the performance.

Comparison with PointPillars. We compare our method with PointPillars [11] on KITTI validation set. We use Det3D[2] [39] implementation to evaluate PointPillars [11]. All comparisons are under the same settings including but not limited to detection range and PointPillars size. As we can see, our AFDet with the modified backbone $\mathcal{B}(1,7,32)+\mathcal{B}(2,8,64)$ can achieve similar performance with PointPillars [11]. But our method does not have a complex post-processing process. We do not need the traditional NMS to filter out results. More importantly, the # parameters in AFDet is about $0.56M$, which is only about 11.6% of its equivalent in PointPillars [11].

Furthermore, using max pooling and AND operation rather than NMS would make it more friendly to deploy AFDet on the embedded systems. We can run nearly the entire algorithm on a CNN accelerator without the tedious post-processing on CPU. We could reserve more CPU computation resources for other tasks in autonomous driving cars. We also tried kernel sizes 5 and 7 in the max pooling. It does not show much difference with kernel size 3.

We show three qualitative results in Figure 4. As we can see, AFDet has the capability to detect the object centers in the heatmap. It can also regress other object attributes (*e.g.* object sizes, z-axis locations and others) well. We validate the effectiveness of the anchor free method on 3D point cloud detection.

4.5. Preliminary Results on Waymo Validation Set

We also include some preliminary evaluation results on Waymo OD [30] validation set. We use Waymo online system to evaluate our performance. We try our best to have the same settings and parameters for a fair comparison. But sometimes we do not know other methods' detailed parameters. On Waymo OD, we train our model with significantly less number of epochs compared with other methods. But we still show competitive or even better results.

We show two AFDet results with PoinPillars [11] encoders in Table 4. The number after the encoder name represents the voxel size in x-y plane. As we can see, our "AFDet+PointPillars-0.16" with voxel size 0.16 m beats "PointPillars" by 2% on LEVEL_1 vehicle detection. When we reduce the voxel size to 0.10 m, our "AFDet+PointPillars-0.10" outperforms the state-of-the-art single-stage methods on Waymo validation set. We only train our model for 16 epochs while others train their models for 75 or 100 epochs for better convergence.

5. Conclusion

In this paper, we tried to address the 3D point cloud detection problem. We presented a novel anchor free one stage 3D object detector (AFDet) to detect the 3D object in the point cloud. We are the first to use anchor free and NMS free method in 3D point cloud detection which has the advantage in the embedded systems. All experimental results proved the effectiveness of our proposed method.

[1] [37, 20, 5] report slightly different performance on the same method. Here we adopt the results reported in [37].

[2] https://github.com/poodarchu/Det3D

References

[1] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019. 2

[2] Qi Chen, Lin Sun, Zhixin Wang, Kui Jia, and Alan Yuille. Object as hotspots: An anchor-free 3d object detection approach via firing of hotspots. *arXiv preprint arXiv:1912.12791*, 2019. 2

[3] 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 *NeurIPS*, 2015. 5

[4] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, 2017. 1, 2, 5

[5] Shuyang Cheng, Zhaoqi Leng, Ekin Dogus Cubuk, Barret Zoph, Chunyan Bai, Jiquan Ngiam, Yang Song, Benjamin Caine, Vijay Vasudevan, Congcong Li, et al. Improving 3d object detection through progressive population based augmentation. *arXiv preprint arXiv:2004.00831*, 2020. 8

[6] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012. 1, 5, 6, 7

[7] Sylvain Gugger. The 1cycle policy. https://sgugger.github.io/the-1cycle-policy.html, 2018. 6

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 5

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 1

[10] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L Waslander. Joint 3d proposal generation and object detection from view aggregation. In *IROS*, 2018. 2

[11] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019. 1, 2, 3, 5, 6, 7, 8

[12] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *ECCV*, 2018. 1, 3, 4, 7

[13] Peixuan Li, Huaici Zhao, Pengfei Liu, and Feidao Cao. Rtm3d: Real-time monocular 3d detection from object keypoints for autonomous driving. *arXiv preprint arXiv:2001.03343*, 2020. 2

[14] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *NeurIPS*, 2018. 2

[15] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *CVPR*, 2017. 4

[16] Zili Liu, Tu Zheng, Guodong Xu, Zheng Yang, Haifeng Liu, and Deng Cai. Training-time-friendly network for real-time object detection. In *AAAI*, 2020. 2

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 6

[18] Gregory P. Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K. Wellington. Lasernet: An efficient probabilistic 3d object detector for autonomous driving. In *CVPR*, 2019. 2

[19] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry. In *CVPR*, 2017. 4

[20] Jiquan Ngiam, Benjamin Caine, Wei Han, Brandon Yang, Yuning Chai, Pei Sun, Yin Zhou, Xi Yi, Ouais Alsharif, Patrick Nguyen, et al. Starnet: Targeted computation for object detection in point clouds. *arXiv preprint arXiv:1908.11069*, 2019. 1, 8

[21] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *ICCV*, 2019. 2

[22] 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. 1, 2

[23] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 1, 2, 3

[24] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 1, 2

[25] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *CVPR*, 2019. 1, 2

[26] Andrea Simonelli, Samuel Rota Bulo, Lorenzo Porzi, Manuel López-Antequera, and Peter Kontschieder. Disentangling monocular 3d object detection. In *ICCV*, 2019. 2

[27] Martin Simony, Stefan Milzy, Karl Amendey, and Horst-Michael Gross. Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds. In *ECCV*, 2018. 2

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 5

[29] Shuran Song and Jianxiong Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *CVPR*, 2016. 2

[30] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. *arXiv preprint arXiv:1912.04838*, 2019. 5, 8

[31] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *ICCV*, 2019. 1

[32] Zhixin Wang and Kui Jia. Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection. In *IROS*, 2019. 1

[33] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 1, 2, 5, 6

[34] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *CVPR*, 2018. 2, 6

[35] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Std: Sparse-to-dense 3d object detector for point cloud. In *ICCV*, 2019. 1, 2

[36] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 1, 2, 4, 6, 7

[37] Yin Zhou, Pei Sun, Yu Zhang, Dragomir Anguelov, Jiyang Gao, Tom Ouyang, James Guo, Jiquan Ngiam, and Vijay Vasudevan. End-to-end multi-view fusion for 3d object detection in lidar point clouds. *arXiv preprint arXiv:1910.06528*, 2019. 1, 8

[38] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, 2018. 1, 2, 5, 6

[39] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *arXiv preprint arXiv:1908.09492*, 2019. 8