# Object as Hotspots: An Anchor-Free 3D Object Detection Approach via Firing of Hotspots

Qi Chen *[1,2], Lin Sun †[1], Zhixin Wang[3], Kui Jia[3], and Alan Yuille[2]

[1]Samsung Inc, USA
[2]Johns Hopkins University
[3]South China University of Technology

## Abstract

*Accurate 3D object detection in LiDAR based point clouds suffers from the challenges of data sparsity and irregularities. Existing methods strive to organize the points regularly, e.g. voxelize, pass them through a designed 2D/3D neural network, and then define object-level anchors that predict offsets of 3D bounding boxes using collective evidence from all the points on the objects of interest. Converse to the state-of-the-art anchor-based methods, based on the very same nature of data sparsity and irregularities, we observe that even points on an isolated object part are informative about position and orientation of the object. We thus argue in this paper for an approach opposite to existing methods using object-level anchors. Technically, we propose to represent an object as a collection of point cliques; one can intuitively think of these point cliques as hotspots, giving rise to the representation of Object as Hotspots (OHS). Based on OHS, we propose a Hotspot Network (HotSpotNet) that performs 3D object detection via firing of hotspots without setting the predefined bounding boxes. A distinctive feature of HotSpotNet is that it makes predictions directly from individual hotspots, and final results are obtained by aggregating these hotspot predictions. Experiments on the KITTI benchmark show the efficacy of our proposed OHS representation. Our one-stage, anchor-free HotSpotNet beats all other one-stage detectors by at least 2% on cars , cyclists and pedestrian for all difficulty levels. Notably, our proposed method performs better on small and difficult objects and we rank the first among all the submitted methods on pedestrian of KITTI test set.*

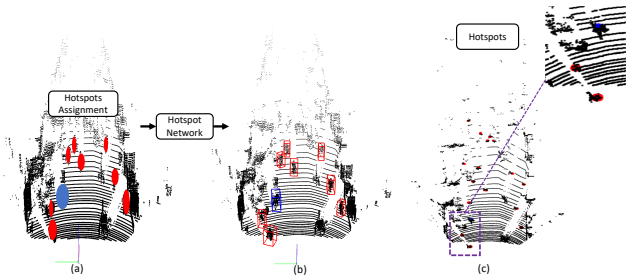



Figure 1: **Hotspots and 3D object detection in LiDAR point clouds using HotSpot Networks.** (a) and (b) presents our proposed HotSpotNet for LiDAR point clouds. (a) is the original LiDAR point clouds with assigned hotspots and (b) is the detection output. (c) illustrates the hotspots which are the firing for inferring instance 3D information. Pedestrians, cyclists and corresponding hotspots are marked in red and blue, respectively. Local region is zoomed in for better visualization. Better viewed in color and zoomed in for details.

## 1. Introduction

Self-driving vehicles, also known as an autonomous vehicle (AV) will definitely impact evolving transportation systems. As the "visual cortex" of the autonomous vehicles, 3D detection algorithms have gained attentions from both the research and industry communities. By representing spatial data as a collection of coordinates, i.e. geographic point clouds, we can construct the world that is analogous to the real world in three dimensions. LiDAR (Light Detection And Ranging) laser scanners are the most common instruments used to collect these geographic point clouds. Owing to self-occlusion, occlusion, reflection or undesirable weather conditions etc., LiDAR point clouds tend to be sparse and irregular. Due to the sparseness and disor-

*Work done during her internship in Samsung.
†corresponding author, lin1.sun@samsung.com

derly nature of this data, until now, no effective processing algorithms are proposed. Meanwhile, the cost of LiDAR are another barrier to its massive adoption on self-driving cars. It's desired for 3D detection algorithms to work with low beam density LiDAR point clouds which is a big challenge for current algorithms as well.

To tackle the existing challenges of LiDAR point clouds, most existing methods strive to organize points on individual objects together, and define object-level anchors that predict offsets of 3D bounding boxes using collective evidence from all the points on the objects of interest. VoxelNet [39] proposes a voxel feature encoder made of a MLP that aggregates sampled point features in a voxel and extracts voxel features with standard Full Convolution Network (FCN), which consists of 3D convolutions and Region Proposal Network (RPN). These voxel-based approaches such as VoxelNet [39], along with its succesors, SECOND [33] and PointPillar [14] require hyper-parameters, such as anchor ranges, anchor sizes and orientations to set anchors as well as the IOU matching to assign ground truths. However, predefined anchors requires the prior knowledge about the statistical size and orientation of objects etc. Additionally, in common practice, the number of anchors grows linearly with the number of object classes which introduces the extra burden. Furthermore, the IOU matching to assign the ground truths decreases the number of positive samples which aggravate the imbalance of negative and positive sample ratio.

Do we really need the pre-defined object-level anchors to infer the 3D location and orientation of the object? According to our observations, the dense 3D shape of objects is never captured by LiDAR point clouds. In some extreme cases, less than 10 points from one object are sensed. We thus argue in this paper for an approach opposite to existing methods using object-level anchors. Inspired by compositional part-based models [10, 40, 6, 4, 12], which have been shown to be robust when classifying partially occluded 2D objects and for detecting partially occluded object parts [37], we propose to detect objects in LiDAR point clouds by representing them as small cliques of points. We take these cliques of points as hotspots which will be described and discussed in Sec. 3.2. We adopt voxelization which partitioning points into cliques as our input. At the same time, we proposed a new object-as-hotspots (OHS) module to handle the voxels and generate 3D information. Based on OHS, we propose a Hotspot Network (HotspotNet) that performs 3D object detection via firing of hotspots without setting the predefined bounding boxes. More specifically, we first partition the 3D space as regular voxels, and aggregate features of individual points inside each voxel to form the hotspot representation; these hotspots are trained to directly predict objects' location and orientation; final results are obtained by applying non-maximum-suppression to the predictions of individual hotspots. The simplified concept of HotspotNet and visualization of hotspots in BEV are shown in Fig. 1.

The main contributions of our proposed method can be summarized as follows:

- We propose a novel representation of point clouds, termed Object as HotSpot (OHS) to effectively handle the data with sparsity and irregularities, e.g. LiDAR point clouds. At the same time, we propose two hotspot assignment methods and test the corresponding performance.
- Based on OHS, we architect the HotSpot Network (HotSpotNet) for LiDAR point clouds and this is the first one-stage and anchor-free 3D detection method which achieves the state-of-the-art performance.
- We propose a combined training loss to deal with the large variance of 3D bounding box regression without the constraint of anchors and train HotSpotNet accurately and effectively. Among them, quadrant classification loss is proposed to learn the inherent and invariant object-part relation which enables HotSpotNet to obtain the 3D information accurately and efficiently.
- Benchmark on KITTI shows the significance of our proposed method, particularly on pedestrian detection, our proposed method beats all the existing methods, ranking 1st on KITTI test dataset. Meanwhile, extensive experimental results on pesudo 32 beam LiDAR point clouds further verify our assumption on efficacy of handling sparse data using our proposed algorithm.

## 2. Related Work

Currently, there are four types of point clouds input for 3D detection algorithms. 1) ***Raw point clouds*** [27, 36] 2) ***Projection*** [34, 28, 1, 5, 21] Usually, they project the 3D LiDAR point clouds into Bird's Eye Views (BEV) or range views 3) ***Voxelization*** [33, 39, 14] Generally, they divide the raw point clouds into regular grids and feeding the regular grid to the algorithms to infer 3D information. 4) ***Mixture of representations*** [3, 19] These methods fuse raw point clouds and voxels on different stages of the networks. Different algorithms may consume different types of input, in this paper, we adopt voxelization representations.

Below, we will briefly review related works from one-stage and two-stage 3D object detection, and then we emphasize possibly related anchor-free 3D object detection.

**One-Stage 3D Detection** Similar to one-stage 2D detectors, one-stage 3D detectors process the input data once to obtain the 3D bounding boxes. One-stage methods always adopt sliding window mechanism, therefore, constructing a contiguous and regular feature representation is

important. VoxelNet [39] and SECOND [33] use the 3D voxel representation and 3D fully convolutional network as backbone network. PointPillar [14] carefully designs pillar shape to organize point cloud to 2D pseudo-image in order to avoid the use of 3D convolution. For these methods, FPN [17] is a commonly used network architecture because it contains rich semantics from all levels. [34, 16] concatenate the channels from a voxelized BEV and the height information of point clouds to form a new input representations based on which they try to obtain the 3D information. [28, 1, 5] try to extend the one stage 2D detector YOLO [25] to high dimensional detectors for point clouds.

**Two-Stage 3D Detection** Different from one-stage methods, two-stage methods have a proposal stage in the first to generate plausible candidate regions. MV3D [2] projects LIDAR point clouds to BEV, and then employs a Faster-RCNN [26] on this BEV to obtain the 3D object cue. AVOD [13] extends MV3D by aggregating multi-modal features to generate more reliable proposals. These methods always treat point clouds after projection in a similar fashion as RGB images and use traditional 2D detection pipelines under multi-modal setting. [27, 36] generate proposals for each foreground point in the entire space after semantic segmentation of point clouds. [23, 32] leverage some mature 2D image detectors to provide frustum proposals. [3] adopts one-stage VoxelRPN to provide initial prediction. Most of two-stage methods have relied heavily on design of pre-defined anchors.

Because of the complex design and low FPS (frame per second) of the two-stage methods, our algorithm is designed based on the one-stage 3D detection algorithm, but our design can be applicable to two-stage detection as well.

**Anchor-Free 3D Detection** To our knowledge, there are no existing anchor-free 3D detectors for LiDAR based point clouds. Some algorithms without anchor regression are proposed for indoors scenes. SGPN [31] segments instances by semantic segmentation and learning a similarity matrix to group points together. This method is not scalable since the size of similarity matrix grows quadratically with the number of points. 3D-BoNet [35] learns bounding boxes to provide a boundary for points from different instances. Unfortunately, both methods will fail when only partial point clouds have been observed, which is common in LiDAR point clouds. VoteNet [22] generates seed points from PointNet++ [24], and independently generates votes through shared voting module which is made of MLP and then refine the clustered points to obtain box proposals. Similar to us, they think each point in the point cloud contribute to the 3D geometry reconstruction. Differently, they take advantage of these information to generate the centroid, while ours are directly regressing the geometry information, i.e. 3D bounding boxes. Though with some anchor-free flavor, VoteNet is not strictly anchor-free because it uses anchor boxes for the size regression, similar to Point-RCNN [27].

# 3. Object as Hotspots

## 3.1. Interpretation of Hotspots

Part-based models are known to be robust to occlusion. When some parts are occluded, the rest are still able to provide hints for semantic information of objects. Likewise, we represents LiDAR scanned objects as a composition of multiple hotspots. Each hotspot individually predicts its likelihood of the object presence so that when an object is mostly occluded or partially scanned, some hotspots are still able to indicate the presence of the object and contribute to the 3D geometry information. Intuitively, hotspots are small cliques of LiDAR points captured on objects. For the voxelization representations, a voxel with several points enclosed is a sample of hotspot. A neuron on the CNN feature maps can represent a collection of voxels whose locations can be projected to the corresponding location of the neuron on the feature map. A neuron has more representation power than a voxel input. During training, neurons on the feature map are assigned as hotspots and non-hotspots for each object category are trained by a binary classifier. In inference, a neuron on a feature map is fired as a hotspot if it gives high confidence being part of the objects.

## 3.2. Hotspot Assignment

In 3D world, usually, objects are rigid which do not coincide with each other. Therefore, for each annotated ground-truth bounding box, it should contain the point clouds from only one object. We can take advantages of the annotated bounding box to determine hotspots which will lie on the objects. Given an object instance in the point clouds, the annotation defines a bounding box $b = [k, x, y, z, l, w, h, r]$ to indicate the object location, where $k$ is the category index, $(x, y, z)$ is the center of the box, $(l, w, h)$ is the dimensions of the box and $r$ is the rotation angle around z-axis in radius in LiDAR coordinates.

Due to the labeling error, boundary points could lie in a confusing area between object and background. These points definitely can not contribute to the final regression. In the consequence, we define an effective box of $[x, y, z, \epsilon_e \cdot l, \epsilon_e \cdot w, \epsilon_e \cdot h, r]$ so that points within the effective box are all high-confident hotspots. We also define an ignoring box $[x, y, z, \epsilon_i \cdot l, \epsilon_i \cdot w, \epsilon_i \cdot h, r]$ outside the effective boxes to be a soft boundary between object and background. $\epsilon_e$ and $\epsilon_i$ are the ratio to control the effective region and ignoring region and $\epsilon_i \geq \epsilon_e$. Points outside the effective box but inside the ignoring box will not do back-propagation during training. Points outside the ignoring box are all non-hotspots.

To decide whether a neuron on the feature map is representing the hotspot(s) or not, we have two alternative meth-

ods. The direct way is to project a collection of non-empty voxels within the effective bounding boxes, to the corresponding location of the neuron on the feature map. Alternatively, we can project all the voxels within the effective bounding boxes, including the empty and non-empty voxels from velodyne coordinates to the feature map. In this assignment, we assume empty voxels assist to acquire the 3D geometry information which can be trained as hotspots as well. We term the two variants of our network according to the hotspots assignment that either non-empty voxel projection or all voxels, including the non-empty and empty ones projection as HotSpotNet-Direct and HotSpotNet-Dense, respectively.

# 4. HotSpot Network

Hotspot Network (HotSpotNet) consists of a 3D feature extractor and Object-as-Hotspots head. OHS head has three subnets for hotspot classification, box regression and quadrant classification. Though we use voxel-based CNN backbone here, our OHS head can actually be stacked on top of any LiDAR point clouds based methods including raw point cloud based methods, e.g. PointNet++ [24].

The whole architecture of our proposed HotSpotNet is shown in Fig. 2. The input LiDAR point clouds are voxelized into regular grids. The regular grids pass through the 3D CNN to generate the feature maps. The three subnets will guide the supervision and generate the predicted 3D bounding boxes. Hotspot assignment happens at the last convolutional feature maps of the backbone. The details of network architecture and the three subnets for supervision will be described below.

## 4.1. Object-as-Hotspots Head

Our OHS head network consists of three sub-networks: 1) a hotspot classification sub-network that predicts the probability of each hotspot being part of an object category; 2) a box regression sub-network that regresses the center locations, dimensions and orientations of the 3D boxes. 3) a quadrant classification sub-network that predicts the quadrant each hotspot will fall into with regard to the local object coordinate originated around the object center.

## 4.2. Hotspot Losses for Supervision

**Hotspot Classification** The classification module is a convolutional layer with $K$ heatmaps and each heatmap corresponds to one category. The hotspots will be labeled as one. The points fall in the ignoring region will be ignored and do not contribute to back-propogation. The targets for all the non-hotspots will be zero. Binary classification will be applied for hotspots and non-hotspots. Focal loss [18] is applied at the end,

$$\mathcal{L}_{cls} = \sum_{k=1}^{K} \alpha(1-p_k)^{\gamma} log(p_k) \tag{1}$$

where,

$$p_k = \begin{cases} p & \text{, for points in effective region} \\ (1-p) & \text{, otherwise} \end{cases}$$

where $p$ is the output probability, $K$ is the number of categories. The total classification loss of one instance is the summation of the focal loss over all effective and negative regions, normalized by the total number of hotspot and non-hotspot units (excluding ones in the ignoring region).

**Box Regression** The bounding box regression only happens on the active hotspot features. For each hot spot feature, a eight-dimensional vector $[d_x, d_y, z, \log(l), \log(w), \log(h), \cos(r), \sin(r)]$ is regressed to represent the object instance in LiDAR point clouds. $d_x, d_y$ are the deviations of hotspot (colored in red) on the feature map to the instance centroid, illustrated in Fig. 3. The voxel centroid in velodyne coordinates in BEV can be obtained by:

$$\begin{aligned}[x_v, y_v] = (&\frac{j+0.5}{L}(x_{max}-x_{min})+x_{min}, \\ &\frac{i+0.5}{W}(y_{max}-y_{min})+y_{min}),\end{aligned} \tag{2}$$

where $i, j$ is the spatial index of a hotspot unit on the feature map, $[x_v, y_v]$ is the corresponding centriod in velodyne coordinates based on which we perform regression, and $[x_{min}, x_{max}]$, $[y_{min}, y_{max}]$ are the ranges for $x, y$ when we voxelize all the points.

Unlike anchor-based methods where there are predefined normalization factors, i.e., anchor box dimensions to regularize the training targets and stabilize training, our HotSpotNet will easily introduce training imbalance without pre-defined normalization factors, due to scale variances of the object dimension (both inter- or intra- object classes). 2D anchor-free detectors often utilizes FPN [17] to alleviate scale variances. Instead of introducing extra layers and computation overhead, e.g. adding FPN, to our network, we tackle scale variances by carefully designing the targets. We regress $\log(l)$, $\log(w)$, $\log(h)$ instead of their original values because $\log$ scales down the absolute values. We regress $\cos(r)$, $\sin(r)$ instead of $r$ directly because they can strictly constrain the rotation to obtain the unique solution. What's more, we use soft $argmin$ proposed in [11] to help to regress $d_x$, $d_y$ and $z$. To regress a point location in a segment ranging from $a$ to $b$ by soft $argmin$, we divide the segment into $N$ bins, each bin accounting for a length of $\frac{b-a}{N}$. The target location can be represented as $t = \Sigma_i^N (S_i C_i)$,

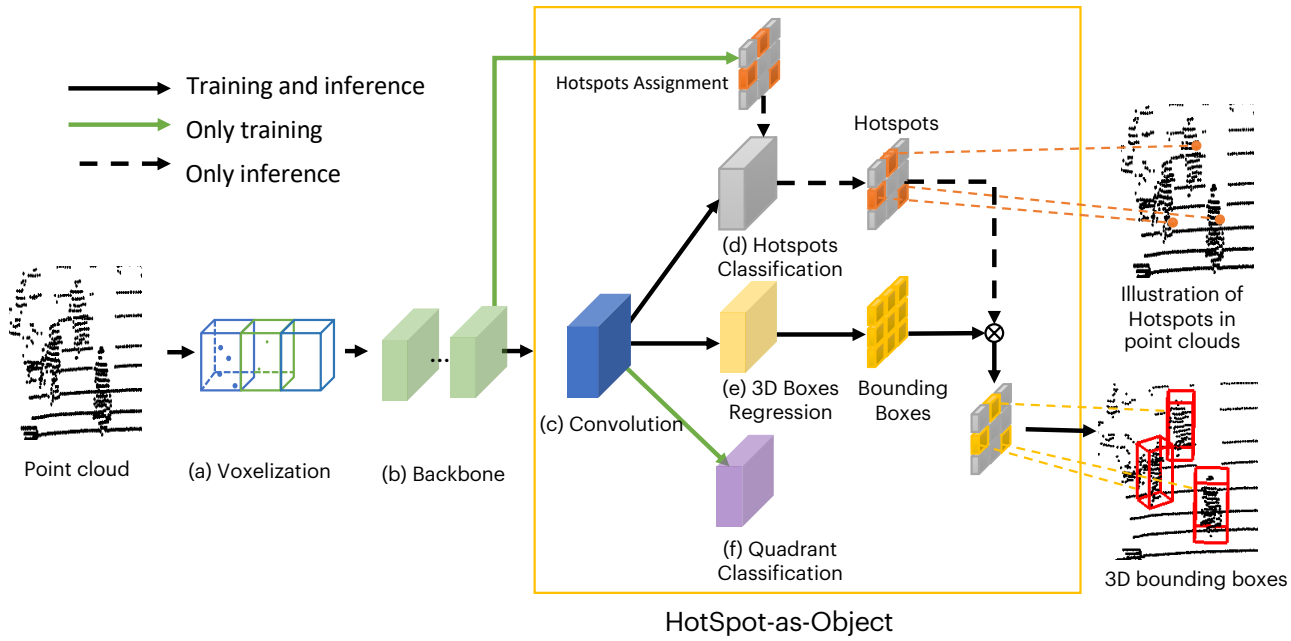



Figure 2: Outline of HotSpotNet. The point cloud is (a) voxelized, and passed through the (b) backbone network to produce 3D feature maps. These feature maps go through (c) a shared convolution layer, and pass into three modules to perform (d) Hotspot Classification and (e) 3D Bounding Box regression (f) Quadrant Classification to train the network. During the inference only (d) Hotspot Classification and (e) 3D Bounding Box Regression are performed to obtain hotspots and bounding boxes respectively. The bounding boxes generated from the hotspots will be projected back to original 3D space.

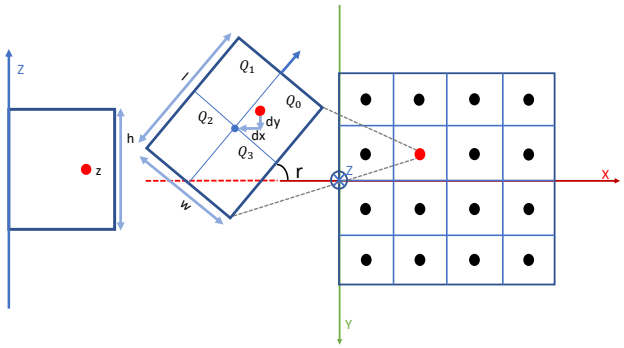



Figure 3: Illustration of regression. Eight parameters $[d_x, d_y, z, log(l), log(w), log(h), \cos r, \sin r]$ are regressed, where $d_x$ and $d_y$ are deviations from hotspot to the instance centroid. For quadrant classification, we partition the object into quadrants $(Q_0, Q_1, Q_2, Q_3)$ in bird eye view. The ordering of quadrants is fixed with regard to the object direction.

where $S_i$ represents the softmax score of the $i_{th}$ bin and $C_i$ is the center location of the $i_{th}$ bin. The soft $argmin$ turns the regression into classification problem and avoids regressing direct values. We find the choices of $a, b$ do not affect the performance of our approach as long as they cover the ranges of target values.

Smooth L1 loss [8] is adopted for regressing these bounding box targets.

$$\mathcal{L}_{loc}(x) = \begin{cases} 0.5x^2 & , |x| < 1 \\ |x| - 0.5 & , \text{otherwise} \end{cases} \quad (3)$$

We only compute the regression loss for locations over all effective box regions.

**Quadrant Classification** Our HotSpotNet predicts the axis-aligned deviations from hotspots to object centroids, i.e. $d_x, d_y$. This encoding does not show the inherent relation between hotspots and object centroids since the deviations will vary with object orientations. We want our model to learn the inherent and invariant object-part relation so we introduce another supervision signal for coarse estimation of the hotspots. For each hotspot within effective box regions, we categorizes the relative hotspot location to the object center (in BEV) into quadrants, as show in Fig. 3. We find quadrant classification helps our HotSpotNet converge faster. We train our quadrant classification subnetwork with binary cross-entropy loss and we compute the loss only for hotspots.

$$\mathcal{L}_q = \sum_{i=0}^{3} -[q_i \log(p_i) + (1 - q_i) \log(1 - p_i)] \quad (4)$$

where $i$ indexes the quadrant, $q_i$ is the target and $p_i$ the predicted likelihood falling into the specific quadrant.

### 4.3. Learning and Inference

The final loss for our proposed HotSpotNet is the weighted sum of losses from three branches:

$$\mathcal{L} = \delta \mathcal{L}_{cls} + \beta \mathcal{L}_{loc} + \zeta \mathcal{L}_q \quad (5)$$

Where, $\delta$, $\beta$ and $\zeta$ are the weights to balance the classification, box regression and quadrant classification loss.

During inference, if corresponding largest entry value of the $K$-dimensional vector of the classification heatmaps is above the threshold, we consider the location as hotspot firing for the corresponding object. For hotspots, it is straightforward to decode the associated predicted boxes from $b = [d_x, d_y, z, l, w, h, \cos(r), \sin(r)]$ to the canonical representation $[x, y, z, l, w, h, r]$. Since one instance might have multiple hotspots, we further use NMS with the Intersection Over Unit (IOU) threshold to pick the most confident hotspot for each object. The quadrant classification branch does not contribute to inference.

## 5. Experiments

In this section, we summarize the dataset in Sec 5.1 and present the implementing details of our proposed HotSpotNet in 5.2. In Sec 5.3 we evaluate our method on the challenging 3D detection Benchmark, KITTI [7]. We also compare our anchor-free approach with baseline on the psudo 32-beam KITTI dataset in Sec 5.4 and present ablation studies in Sec 5.5. In Sec 5.6, we illustrate visualization results.

### 5.1. Dataset and Evaluation

**Dataset** KITTI has 7,481 annotated LiDAR point clouds for training with 3D bounding boxes for object classes such as cars, pedestrians and cyclists. Additionally, KITTI provides 7,518 LiDAR point clouds without labeling for testing. In the rest of paper, without explicitly noting, all the experiments are running on the common train/val split, i.e. 3712 LiDAR point clouds for training and 3769 LiDAR point clouds for validation. The performance is reported on validation data. To further compare the results with other approaches on KITTI 3D detection benchmark, we randomly split the KITTI training data into $4:1$ for training and validation and report the performance on KITTI test dataset.

To further verify our proposed method can better tackle natural sparsity of point clouds, we generate pseudo 32 beam KITTI down-sampled from the original 64 beam KITTI. We observe that the point elevations are densely distributed within the range of $[-0.23, 0.07]$ and sparsely distributed within the range of $[-0.27, -0.23]$ and $[0.07, 0.13]$. In consequence, we uniformly divide $[-0.27, 0.23]$, $[-0.23, 0.07]$, $[0.07, 0.13]$ into 4, 56, 4 bins, respectively and sample the points from 32 bins with stride 2 to simulate the 32 beam point clouds, resulting in two sets of pseudo 32-beam KITTI datasets. The comparing visualization of 64 beam and pseudo 32 beam Lidar point clouds can be seen in Fig. 4. We report average performance on two pseudo 32-beam KITTI datasets.

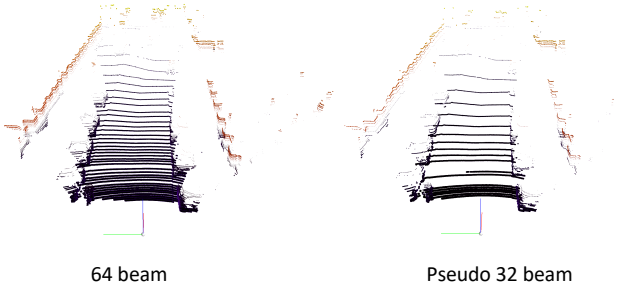



Figure 4: From left to right, real 64-beam LiDAR point clouds from KITTI, pseudo 32-beam LiDAR point clouds generated from our sampling.

**Metric** Same as others, average precision (AP) is used to evaluate our method. We follow the official KITTI evaluation protocol, i.e., the IoU threshold is 0.7 for car and 0.5 for pedestrian and cyclist. Precision and recall curves are computed using 40 points instead of 11 points.

### 5.2. Implementation Details

**Backbone Network** In all experiments, we adopt the similar backbone network architecture as the open source implementation in [33]. The details of our architecture of backbone network are show in Fig. 5. We use the simplified version of Voxel Feature Encoder, i.e. VFE [39], by taking the mean of fixed amount of points sampled in a voxel. Our backbone network has 3D and 2D parts. 3D part has in total 14 3D sparse convolution blocks and the 2D part has 5 2D convolution blocks. For the 3D part, we use the convolution proposed in [9], including sparse convolution and submanifold convolution. We down-sample in total four times except the last layer we only down-sample in the height dimension. For the transformation of 3D to 2D, we collapse the height dimension. For instance, if the dimension of height is 2 and channel dimension is 64, after collapsing, the channel dimension is 128.

**Object-as-Hotspots Head** Since the output feature map of the backbone network collapses to bird eye view, we thus

| Method | Input | Stage | 3D Detection (Car) | | | 3D Detection (Cyclist) | | | 3D Detection (Pedestrian) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Mod | Easy | Hard | Mod | Easy | Hard | Mod | Easy | Hard |
| PIXOR[34] | L | One | - | - | - | - | - | - | - | - | - |
| ComplexYOLO[28] | L | One | 47.34 | 55.93 | 42.60 | 18.53 | 24.27 | 17.31 | 13.96 | 17.60 | 12.70 |
| VoxelNet[39] | L | One | 65.11 | 77.47 | 57.73 | 48.36 | 61.22 | 44.37 | 39.48 | 33.69 | 31.51 |
| SECOND-V1.5[33] | L | One | 75.96 | 84.65 | 68.71 | - | - | - | - | - | - |
| HR-SECOND[33] | L | One | 75.32 | 84.78 | 68.70 | 60.82 | 75.83 | 53.67 | 35.52 | 45.31 | 33.14 |
| PointPillar[14] | L | One | 74.31 | 82.58 | 68.99 | 58.65 | 77.10 | 51.92 | 41.92 | 51.45 | 38.89 |
| 3D IoU Loss[38] | L | One | 76.50 | 86.16 | 71.39 | - | - | - | - | - | - |
| HRI-VoxelFPN[30] | L | One | 76.70 | 85.64 | 69.44 | - | - | - | - | - | - |
| ContFuse [16] | I + L | One | 68.78 | 83.68 | 61.67 | - | - | - | - | - | - |
| MV3D [2] | I + L | Two | 63.63 | 74.97 | 54.00 | - | - | - | - | - | - |
| AVOD-FPN [13] | I + L | Two | 71.76 | 83.07 | 65.73 | 50.55 | 63.76 | 44.93 | 42.27 | 50.46 | 39.04 |
| F-PointNet [23] | I + L | Two | 69.79 | 82.19 | 60.59 | 56.12 | 72.27 | 49.01 | 42.15 | 50.53 | 38.08 |
| MMF [15] | I + L | Two | 77.43 | **88.40** | 70.22 | - | - | - | - | - | - |
| PointRCNN [27] | L | Two | 75.64 | 86.96 | 70.70 | 58.82 | 74.96 | 52.53 | 39.37 | 47.98 | 36.01 |
| FastPointRCNN[3] | L | Two | 77.40 | 85.29 | 70.24 | - | - | - | - | - | - |
| HotSpotNet-Dense | L | One | **78.34** | 88.12 | **73.49** | 62.72 | **79.09** | 56.76 | 39.72 | 47.14 | 37.25 |
| HotSpotNet-Direct | L | One | 77.74 | 86.40 | 72.97 | **63.16** | 77.70 | **57.16** | **44.81** | **51.29** | **41.13** |

| Method | Input | Stage | BEV Detection (Car) | | | BEV Detection (Cyclist) | | | BEV Detection (Pedestrian) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Mod | Easy | Hard | Mod | Easy | Hard | Mod | Easy | Hard |
| PIXOR[34] | L | One | 80.01 | 83.97 | 74.31 | - | - | - | - | - | - |
| ComplexYOLO[28] | L | One | 68.96 | 77.24 | 64.95 | 25.43 | 32.00 | 22.88 | 18.26 | 21.42 | 17.06 |
| VoxelNet[39] | L | One | 79.26 | 89.35 | 77.39 | 54.76 | 66.70 | 50.55 | 46.13 | 40.74 | 38.11 |
| SECOND-V1.5[33] | L | One | 86.37 | 91.81 | 81.04 | - | - | - | - | - | - |
| HR-SECOND[33] | L | One | 86.40 | 91.68 | 81.40 | 64.21 | 78.79 | 57.82 | 40.06 | 50.05 | 36.47 |
| PointPillar[14] | L | One | 86.56 | 90.07 | 82.81 | 62.73 | 79.90 | 55.58 | 48.64 | 57.60 | 45.78 |
| 3D IoU Loss[38] | L | One | 86.22 | 91.36 | 81.20 | - | - | - | - | - | - |
| HRI-VoxelFPN[30] | L | One | 87.21 | 92.75 | 79.82 | - | - | - | - | - | - |
| ContFuse [16] | I + L | One | 85.35 | 94.07 | 75.88 | - | - | - | - | - | |
| MV3D [2] | I + L | Two | 78.93 | 86.62 | 69.80 | - | - | - | - | - | - |
| AVOD-FPN [13] | I + L | Two | 84.82 | 90.99 | 79.62 | 57.12 | 69.39 | 51.09 | **50.32** | **58.49** | **46.98** |
| F-PointNet [23] | I + L | Two | 84.67 | 91.17 | 74.77 | 61.37 | 77.26 | 53.78 | 49.57 | 57.13 | 45.48 |
| MMF [15] | I + L | Two | **88.21** | 93.67 | 81.99 | - | - | - | - | - | - |
| PointRCNN [27] | L | Two | 87.39 | 92.13 | 82.72 | **67.24** | **82.56** | 60.28 | 46.13 | 54.77 | 42.84 |
| FastPointRCNN[3] | L | Two | 87.84 | 90.87 | 80.52 | - | - | - | - | - | - |
| HotSpotNet-Dense | L | One | 88.11 | **93.73** | **84.98** | 66.86 | 82.13 | 60.86 | 44.59 | 50.87 | 42.14 |
| HotSpotNet-Direct | L | One | 87.95 | 93.59 | 83.21 | 67.20 | 79.66 | **61.04** | 49.48 | 55.90 | 45.79 |

Table 1: Performance of 3D object detection and BEV detection on KITTI test set for Car, Cyclist and Pedestrian categories. For fair comparison, we also submitted our implemented SECOND [33] with same setting as ours, represented by HR-SECOND in the table. "L" indicates the method uses LiDAR point clouds, "I" indicates the method uses RGB images and "L+I" indicates the method uses RGB images and LiDAR point clouds.

in this paper assign hotspots in bird eye view (though they can also be extended to 3D space). As shown in Fig. 2, our OHS head consists of a shared $3 \times 3$ convolution layer ((c) in the Fig. 4) with stride 1. We use a $1 \times 1$ convolution layer followed by a sigmoid to predict confidence for hotspots. For the regression, we apply several $1 \times 1$ convolution layers to different regressed values. For instance, two $1 \times 1$ convolution layers are stacked to predict soft argmin for $d_x$, $d_y$ and another $1 \times 1$ convolution layer is used to predict soft $argmin$ for $z$. Additional $1 \times 1$ convolution layer to predict the dimensions and another $1 \times 1$ convolution layer for rotation. We set the range $[-3, 3]$ with 16 bins for $d_x$, $d_y$, range $[-3, 1]$ with 16 bins for $z$. For quadrant classification, we use another a $1 \times 1$ convolution layer with softmax for cross-entropy classification.

We set $\gamma = 2.0$ and $\alpha = 0.25$ for focal loss. We set $\delta = 1$, $\beta = 1$ and $\zeta = 1$ in the weighted sum of losses.

We set $\epsilon_e = 0.9$, $\epsilon_i = 1.0$ for car and $\epsilon_e = 1.4$, $\epsilon_i = 1.4$ for pedestrian and cyclist. We set the effective region of pedestrian and cyclist larger to cover more hotspots.

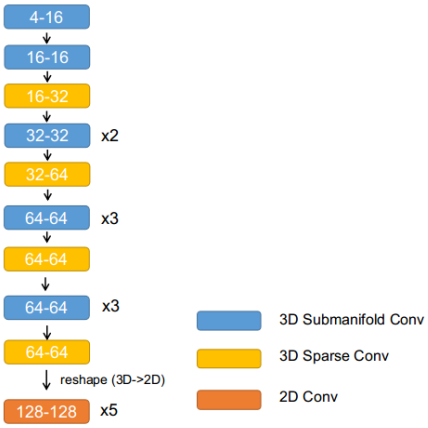



Figure 5: The architecture of backbone network. Each block is consisted of Convolution, BatchNorm, ReLU. The values in each block represent the number of input feature channels, output feature channels respectively. Unless otherwise noted, all the convolution has kernel size of 3. The stride of each 3D sparse convolution is (2, 2, 2), except the last Sparse Convolution, we only down-sample the height dimension.

**Training and Inference** We train the entire network end-to-end with adamW [20] optimizer and one-cycle policy [29] with LR max $2.25e^{-3}$, division factor 10, momentum ranges from 0.95 to 0.85, fixed weight decay 0.01. We train the network with batch size 8 for 150 epochs. During the testing, we keep 100 proposals after filtering the confidence lower than 0.3, and then apply the rotated NMS with IOU threshold 0.01 to remove redundant boxes.

**Data Augmentation** We augment data using random flipping along the $y$ dimension in velodyne coordinates, global scaling with scaling factor sampled from $[0.95, 1.05]$, global rotation around $z$ axis by an angle sampled from $[-\frac{\pi}{4}, \frac{\pi}{4}]$. We also apply ground truth database sampling [33] on each instance by translating with gaussian noise $\mathcal{N}(0, 1), \mathcal{N}(0, 1)$ and $\mathcal{N}(0, 0.5)$ for $x$, $y$, $z$ respectively and rotating around $z$ axis with a uniform noise from $[-\frac{\pi}{4}, \frac{\pi}{4}]$.

### 5.3. Experimental results on KITTI benchmark

As shown in Table 1, we evaluate our method on the 3D detection benchmark and the birds eye view detection benchmark of the KITTI test dataset. For the 3D object detection benchmark, by only using LiDAR point clouds, our proposed HotSpotNet-Direct outperforms all previous peer-reviewed LiDAR-based, one-stage detectors on cars, cyclists and pedestrian of all difficulty levels. In particular, HotSpotNet-Direct shows its advantages on small and difficult objects, for example, pedestrian and cyclists. When detecting relatively large objects, such as cars, the empty voxels, i.e. dense hotspots assignment, may provide help, consequently, HotSpotNet-Dense presents some superiority on cars. On the contrary, it introduces noise to the small and sparse LiDAR scanned objects, therefore, it performs worse on cyclist and pedestrian. In the rest of the paper, without further emphasizing, we will adopt HotSpotNet-Direct as our method for quantitative evaluations. Note that all other methods listed in Table 1 are anchor-based except that PIXOR [34] is an anchor-free 2D detector. The inspiring results show the success of representing objects as hotspots as well as potentials of anchor-free detectors in 3D detection. Our approach also beats some classic 3D two-stage detectors, even when they fuse LiDAR and RGB images information. Note that our method rank 1st on pedestrian, outperforming all the submitted results on KITTI test set.

### 5.4. Experimental results for pseudo 32-beam KITTI

To further verify our advantages on handling sparse LiDAR point clouds. We train and validate our proposed method on pesudo 32-beam KITTI, compared with SECOND [33]. As shown in Table 2, HotSpotNet significantly outperforms the baseline on all categories in all difficulty levels. Notably, our method achieves more obvious improvements on the hard level, where objects are usually far away, occluded and truncated, and thus most sparsely captured by LiDAR. This validates our motivation to tackle the sparse nature of LiDAR point clouds.

### 5.5. Ablation Studies

**Effect of quadrant classification** To prove the effectiveness of quadrant classification, we show the results of our HotSpotNet with and without quadrant classification on KITTI validation split for cars in Table 4. We can see that when our algorithm trained with the quadrant classification, the overall performance is boosted. Especially, the great improvement can be observed in hard level. To show that our HotSpotNet design can benefit from quadrant classification, we also add quadrant classification to the baseline SECOND [33]. Interestingly, we find quadrant classification impairs the baseline performance. One hypothesis is anchor-based methods 'treats an object as a whole', on the contrary, the quadrant classification is trying to encode part information, resulting in a conflict. Quadrant classification can particularly handle the ambiguities raised by anchor-free algorithm.

We proposed the quadrant classification as an additional loss to spatially localize the hotspots. This inherent and invariant object-part relation enables HotSpotNet to converge fast. We further investigate the effects of different hotspot-object spatial relation encoding methods. Besides the quadrant partition presented above, we present four more types of encodings as shown in Fig. 6. We supervise our network using different spatial encoding targets: 1) classify-

Table 2: Performance on pseudo 32-beam KITTI

| Method | 3D Detection on Car | | | 3D Detection on Cyclist | | | 3D Detection on Pedestrian | | |
|---|---|---|---|---|---|---|---|---|---|
| | Mod | Easy | Hard | Mod | Easy | Hard | Mod | Easy | Hard |
| [33] | 72.76 | 87.37 | 67.91 | 56.66 | 75.21 | 52.87 | 56.98 | 62.55 | 51.19 |
| Ours | **74.08** | **87.90** | **70.60** | **59.23** | **79.40** | **55.42** | **58.81** | **64.00** | **52.89** |
| Method | BEV Detection on Car | | | BEV Detection on Cyclist | | | BEV Detection on Pedestrian | | |
| | Mod | Easy | Hard | Mod | Easy | Hard | Mod | Easy | Hard |
| [33] | 82.98 | 92.55 | 78.09 | 59.82 | 78.84 | 56.00 | 63.36 | 69.25 | 57.37 |
| Ours | **84.90** | **93.10** | **81.65** | **62.36** | **82.12** | **58.60** | **64.48** | **70.34** | **58.74** |

Table 3: Effect of quadrant classification.

| Method | 3D Detection on Car | | | 3D Detection on Cyclist | | | 3D Detection on Pedestrian | | |
|---|---|---|---|---|---|---|---|---|---|
| | Mod | Easy | Hard | Mod | Easy | Hard | Mod | Easy | Hard |
| Ours w/o quadrant | 82.27 | 91.75 | 79.96 | 69.31 | **89.48** | 65.04 | 65.45 | **72.77** | 58.36 |
| Ours w quadrant | **82.75** | **91.87** | **80.22** | **72.55** | 88.22 | **68.08** | **65.9** | 72.23 | **60.06** |
| [33] w/o quadrant | 81.96 | 90.95 | 77.24 | 61.62 | 80.13 | 57.77 | 64.19 | 69.14 | 57.99 |
| [33] w/ quadrant | 78.1 | 89.49 | 73.11 | 47.75 | 64.37 | 44.31 | 49.02 | 55.19 | 43.23 |
| Method | BEV Detection on Car | | | BEV Detection on Cyclist | | | BEV Detection on Pedestrian | | |
| | Mod | Easy | Hard | Mod | Easy | Hard | Mod | Easy | Hard |
| Ours w/o quadrant | 89.29 | 95.75 | **88.86** | 71.63 | **90.63** | 67.25 | 68.86 | **76.38** | 62.93 |
| Ours w quadrant | **89.67** | **95.88** | 87.23 | **74.97** | 90.41 | **70.53** | **69.28** | 75.83 | **63.58** |
| [33] w/o quadrant | 89.29 | 93.15 | 86.53 | 64.78 | 83.29 | 61.08 | 67.9 | 72.66 | 61.68 |
| [33] w/ quadrant | 86.93 | 93.48 | 79.97 | 51.41 | 69.49 | 47.73 | 58.86 | 63.74 | 53.66 |

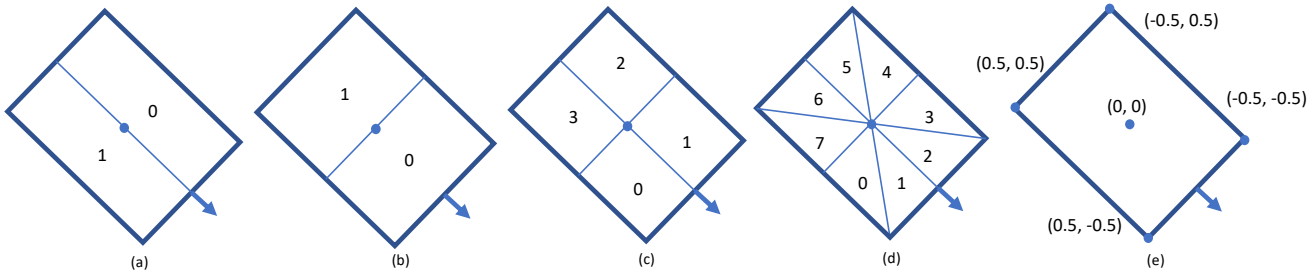



Figure 6: Different hotspot-object spatial relation encodings in local object coordinate system. (a) classifying the hotspot location into left or right part of object bounding box. (b) classifying the hotspot location into front or back of object bounding box. (c) classifying the hotspot location into quadrants of object bounding box. (d) classifying the hotspot location into 8 directions of object bounding box. (e) regressing the hotspot location relative to the object center. The object center is the origin. The farther away from the center, the higher the absolute degree of deviation $(0.5)$. The values are normalized by the sizes of the bounding box.

ing hotspot location into left or right part of the object; 2) classifying hotspot location into front or back of the object; 3) classifying hotspot location into quadrants of the objects; 4) classifying hotspot location into eight directions of the objects; 5) directly regressing deviation to the object center. Deviation is two decimals denoting the relative deviations from the center along the box width and length, and ranges within $[-0.5, 0.5]$ because we normalize the values by the box width and length. Thus, $(0, 0)$ is the center of the box and the four corners are $(-0.5, -0.5)$, $(-0.5, 0.5)$, $(0.5, -0.5)$ and $(0.5, 0.5)$. The performance of our approach without any spatial relation encoding is presented using 'Ours w/o directions'. The performances of integrating different encodings into our approach are listed

Table 4: Effects of different hotspot-object spatial relation encoding.

| Method | 3D Detection on Car | | | 3D Detection on Cyclist | | | 3D Detection on Pedestrian | | |
|---|---|---|---|---|---|---|---|---|---|
| | Mod | Easy | Hard | Mod | Easy | Hard | Mod | Easy | Hard |
| Ours w/o directions | 82.27 | 91.75 | 79.96 | 69.31 | 89.48 | 65.04 | 65.45 | 72.77 | 58.36 |
| Ours w/ left&right | 82.25 | 91.62 | 79.66 | 69.56 | **88.42** | 65.38 | 64.05 | 69.58 | 57.78 |
| Ours w/ front&back | 82.42 | 91.88 | 80.03 | 69.07 | 87.51 | 64.76 | 65.18 | 71.45 | 59.44 |
| Ours w quadrant | **82.75** | 91.87 | **80.22** | **72.55** | 88.22 | **68.08** | 65.9 | **72.23** | **60.06** |
| Ours w/ 8 directions | 82.66 | **92.04** | 80.21 | 69.99 | 86.29 | 64.94 | **66.26** | 71.11 | 58.92 |
| Ours w/ deviation regression | 82.03 | 91.92 | 79.66 | 70.25 | 87.29 | 65.29 | 65.13 | 71.42 | 58.77 |
| Method | BEV Detection on Car | | | BEV Detection on Cyclist | | | BEV Detection on Pedestrian | | |
| | Mod | Easy | Hard | Mod | Easy | Hard | Mod | Easy | Hard |
| Ours w/o directions | 89.29 | 95.75 | **88.86** | 71.63 | 90.63 | 67.25 | 68.86 | 76.38 | 62.93 |
| Ours w/ left&right | 89.08 | 95.42 | 88.8 | 72.04 | 90.07 | 67.91 | 67.96 | 73.71 | 62.45 |
| Ours w/ front&back | 89.06 | 95.6 | 88.62 | 71.95 | 89.28 | 67.72 | 70.18 | 76.42 | 64.53 |
| Ours w quadrant | **89.67** | **95.88** | 87.23 | **74.97** | **90.41** | **70.53** | 69.28 | 75.83 | 63.58 |
| Ours w/ 8 directions | 89.22 | 95.82 | 88.75 | 71.11 | 87.04 | 66.46 | **70.55** | **76.46** | **63.94** |
| Ours w/ deviation regression | 89.1 | 95.77 | 88.58 | 72.32 | 88.83 | 68.04 | 69.85 | 75.12 | 62.8 |

in Tab. 4. Generally, coarse, e.g. two partitions, left&right or front&back, or too sophisticated, e.g. eight directions, hotspot-object relations does not help the regression. The results show that only quadrant partition can significantly improve the performance.

Interestingly, adding additional deviation regression conversely degrades the performance. Further analyzing the reasons behind, we find deviation with the orientation together is sufficient for global center deviation regression, i.e. obtaining $d_x, d_y$. Denote deviation along box dimensions as $(c_x, c_y)$, and the relation between global center deviation $(d_x, d_y)$ and $(c_x, x_y)$ is represented by Eq. 6

$$(c_x \times l, c_y \times w) = -(d_x, d_y) \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \quad (6)$$

where $\theta$ is the rotation around $z$ axis and clockwise, and $l, w$ denote the length and width of the box.

Deviation regression introduces redundant target to our approach, and at the same time, forces the network to learn the complicated transformation (Eq. 6). When the network does not have enough representation power to learn this transformation, it gets overwhelmed by this redundant information and thus the performance drops. On the contrary, coarse hotspot-object spatial relations (front&back, left&right, quadrants and 8 directions) can be considered as relaxations for Eq. 6. They provide the inherent hotspot-object spatial relation. We find quadrant classification works best in our proposed method. Therefore, we adopt quadrant classification in the final experiments.

**Effect of soft argmin** We show the importance of soft **argmin** by removing it gradually. 'direct' in Table 5 means we directly regress the raw values of corresponding targets. We perceive improvements by using soft $argmin$ instead of the raw values. Particularly on cyclist and pedestrian, our soft **argmin** regression brings more improvements.

### 5.6. Visualization of Hotspots

Previously, we introduce the concept of hotspots and their assignment methods. Do we really learn the hotspots and what do they look like? We trace our detection bounding boxes results back to original fired cliques and visualize them in Fig. 7. Here we visualize some samples with cars from validation dataset, all the fired hotspots are marked green. (a) presents the original LiDAR point clouds in BEV and (b) shows all the point clouds from the detected cars. Interestingly, all the fired hotspots sit at the front corner of the car. It shows that the front corner may be the most distinctive 'part' for detecting/representing a car.

## 6. Qualitative Results

We visualize some representative results in Fig. 8 and Fig. 9. Our baseline, SECOND [33], the anchor-based method, typically fails to detect (misses) the objects when they are far away from the sensors, i.e., the objects appear with sparse LiDAR point clouds. Our approach however is robust to these circumstances, as shown in Fig. 8. Both baseline and our approach suffer from false positives, as presented in Fig. 9(b) and 9(c). The future step may be investigating ways to incorporate appearance cues from RGB images to prevent false positives, for instance, trees being recognized as cars or road signs being recognized as pedestrians.

Table 5: Performance of regression strategy on $(x, y, z)$ coordination.

| Method | 3D Detection on Car | | | 3D Detection on Cyclist | | | 3D Detection on Pedestrian | | |
|---|---|---|---|---|---|---|---|---|---|
| | Mod | Easy | Hard | Mod | Easy | Hard | Mod | Easy | Hard |
| $d_x, d_y, z$ direct | 82.31 | 91.53 | 79.88 | 68.65 | 88.11 | 64.36 | 63.7 | 67.62 | 57.15 |
| $d_x, d_y$ direct | 81.89 | 91.17 | 79.56 | 68.8 | 87.03 | 64.57 | 63.06 | 70.03 | 55.57 |
| $z$ direct | **82.87** | **92.03** | **80.46** | 70.8 | 88.19 | 66.54 | 61.74 | 68.96 | 54.35 |
| Ours | 82.75 | 91.87 | 80.22 | **72.55** | **88.22** | **68.08** | **65.9** | **72.23** | **60.06** |
| Method | BEV Detection on Car | | | BEV Detection on Cyclist | | | BEV Detection on Pedestrian | | |
| | Mod | Easy | Hard | Mod | Easy | Hard | Mod | Easy | Hard |
| $d_x, d_y, z$ direct | 89.3 | 95.83 | 86.97 | 72.64 | 90.32 | 68.36 | 66.96 | 71.02 | 60.54 |
| $d_x, d_y$ direct | 88.85 | 95.22 | **88.44** | 70.28 | 87.19 | 65.95 | 67.29 | 74.44 | 60.13 |
| $z$ direct | 89.48 | 95.62 | 87.2 | 72.64 | 90.32 | 68.36 | 67.17 | 74.18 | 59.59 |
| Ours | **89.67** | **95.88** | 87.23 | **74.97** | **90.41** | **70.53** | **69.28** | **75.83** | **63.58** |

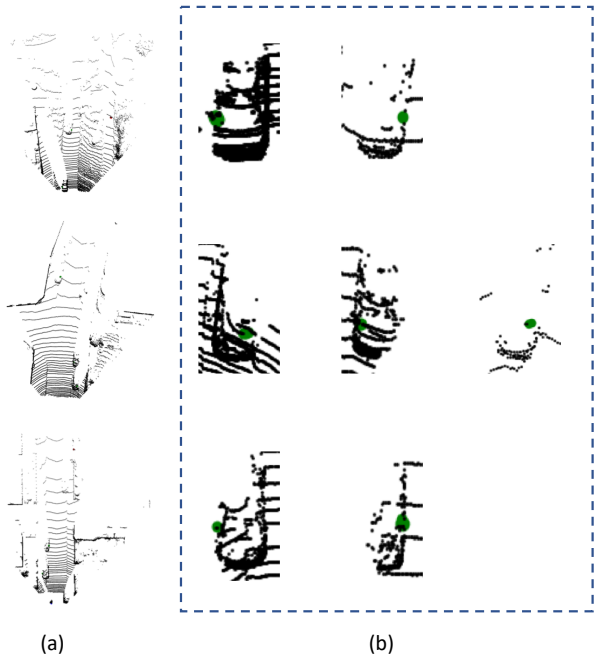



Figure 7: Hotspots visualization. (a) is the original LiDAR point clouds visualization in BEV. All the cars in (a) with active hotspots (colored in green) are visualized in (b). Better viewed in color and zoom in.

## 7. Conclusion

We propose a novel representation, object-as-hotspots and the first one-stage and anchor-free 3D object detector, HotspotNet, for 3D object detection in autonomous driving scenario. Our anchor-free detector outperforms all previous one-stage detectors on categories of KITTI dataset by a large margin. Extensive experiments show that our approach is robust and effective to sparse point clouds. We propose quadrant classification to encode the inherent relation between hotspots and objects and stabilize our network training. We believe our work sheds insights on rethinking 3D object representations and at the same time, shows the potential of anchor-free in 3D algorithm design.

## 8. Acknowledgement

We thank Ernest Cheung (Samsung), Gweltaz Lever (Samsung), Dr. Xingyu Zhang (Apple) and Chenxu Luo (Johns Hopkins University and Samsung) for useful discussions that greatly improved the manuscript.

## References

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

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

[3] Yilun Chen, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Fast point r-cnn. In *ICCV*, October 2019.

[4] Jifeng Dai, Yi Hong, Wenze Hu, Song-Chun Zhu, and Ying Nian Wu. Unsupervised learning of dictionaries of hierarchical compositional models. In *CVPR*, 2014.

[5] Ahmad El Sallab, Ibrahim Sobh, Mahmoud Zidan, Mohamed Zahran, and Sherif Abdelkarim. Yolo4d: A spatio-temporal approach for real-time multi-object detection and classification from lidar point clouds. 2018.

[6] Sanja Fidler, Marko Boben, and Ales Leonardis. Learning a hierarchical compositional shape vocabulary for multi-class object representation. *arXiv preprint arXiv:1408.5516*, 2014.

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

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

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

[10] Ya Jin and Stuart Geman. Context and hierarchy in a probabilistic image model. In *CVPR*, 2006.

[11] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *ICCV*, 2017.

[12] Adam Kortylewski, Aleksander Wieczorek, Mario Wieser, Clemens Blumer, Sonali Parbhoo, Andreas Morel-Forster, Volker Roth, and Thomas Vetter. Greedy structure learning of hierarchical compositional models. *arXiv preprint arXiv:1701.06171*, 2017.

[13] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L Waslander. Joint 3d proposal generation and object detection from view aggregation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[14] 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.

[15] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *CVPR*, 2019.

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

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

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

[19] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. *arXiv preprint arXiv:1907.03739*, 2019.

[20] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

[21] 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.

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

[23] 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.

[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 *Neural Information Processing Systems*, 2017.

[25] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

[26] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems*, 2015.

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

[28] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds. In *ECCV*, 2018.

[29] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 1100612. International Society for Optics and Photonics, 2019.

[30] Bei Wang, Jianping An, and Jiayan Cao. Voxel-fpn: multi-scale voxel feature aggregation in 3d object detection from point clouds. *arXiv preprint arXiv:1907.05286*, 2019.

[31] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. In *CVPR*, 2018.

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

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

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

[35] Bo Yang, Jianan Wang, Ronald Clark, Qingyong Hu, Sen Wang, Andrew Markham, and Niki Trigoni. Learning object bounding boxes for 3d instance segmentation on point clouds. *arXiv preprint arXiv:1906.01140*, 2019.

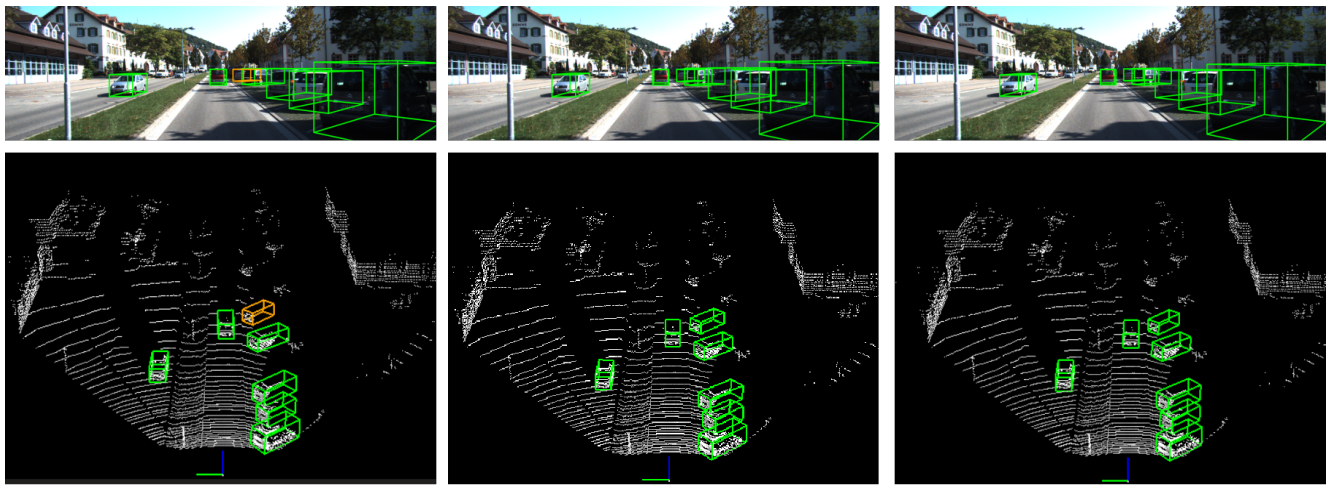
[36] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Std: Sparse-to-dense 3d object detector for point cloud. *arXiv preprint arXiv:1907.10471*, 2019.

[37] Zhishuai Zhang, Cihang Xie, Jianyu Wang, Lingxi Xie, and Alan L Yuille. Deepvoting: A robust and explainable deep network for semantic part detection under partial occlusion. In *CVPR*, 2018.

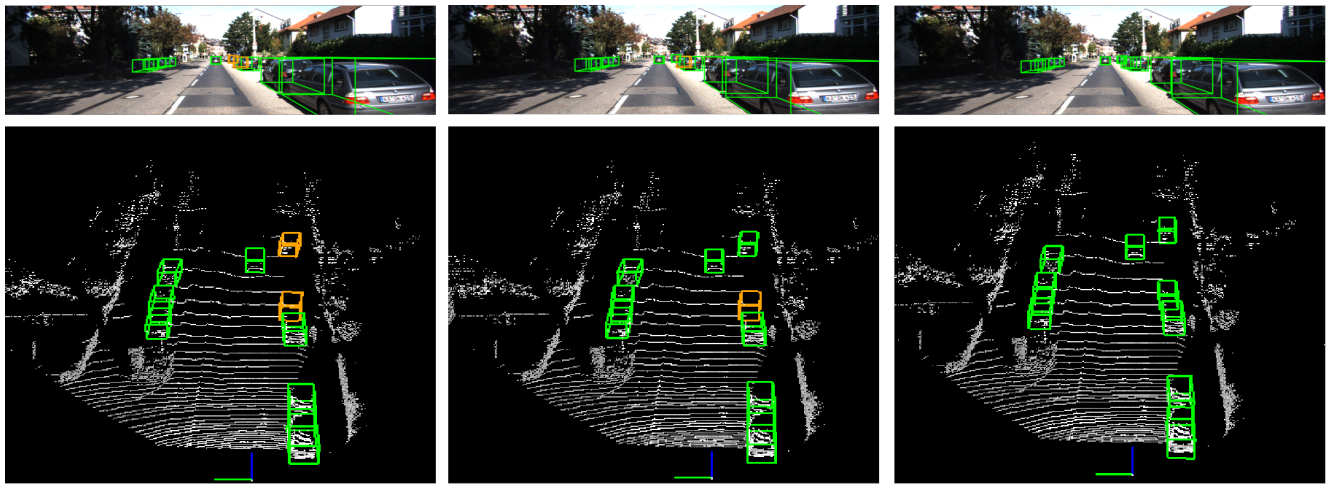
[38] Dingfu Zhou, Jin Fang, Xibin Song, Chenye Guan, Junbo Yin, Yuchao Dai, and Ruigang Yang. Iou loss for 2d/3d object detection. *arXiv preprint arXiv:1908.03851*, 2019.

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

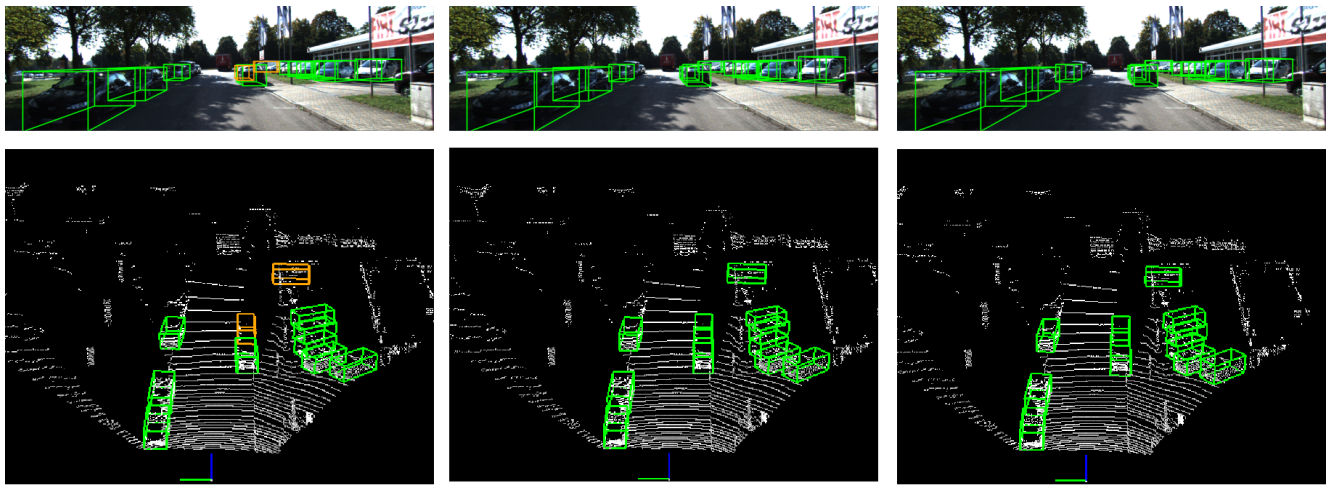
[40] Long Leo Zhu, Chenxi Lin, Haoda Huang, Yuanhao Chen, and Alan Yuille. Unsupervised structure learning: Hierarchical recursive composition, suspicious coincidence and competitive exclusion. In *ECCV*. 2008.



(a)

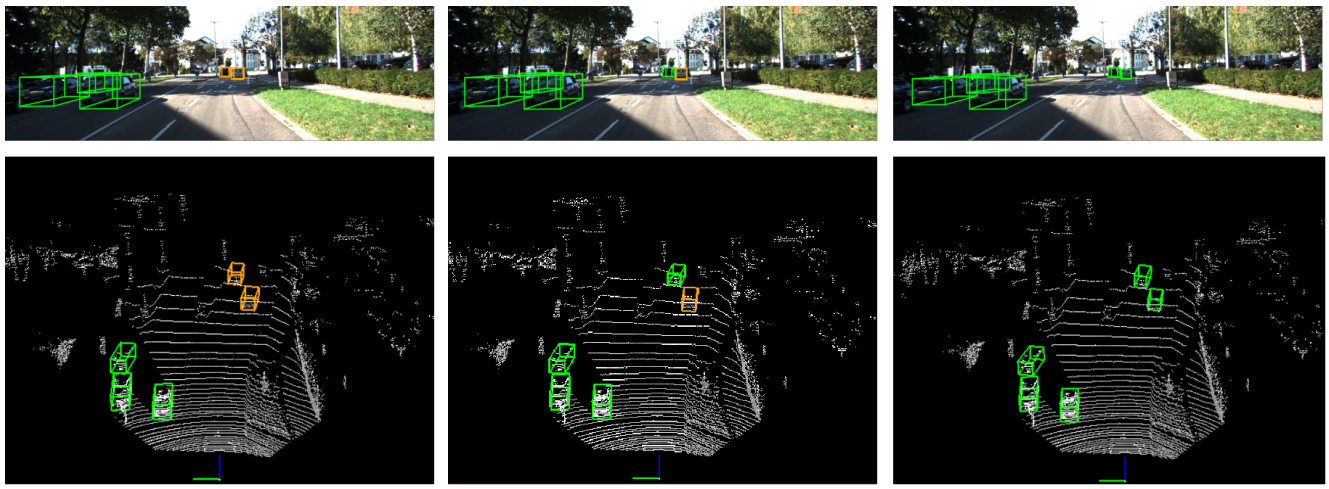


(b)

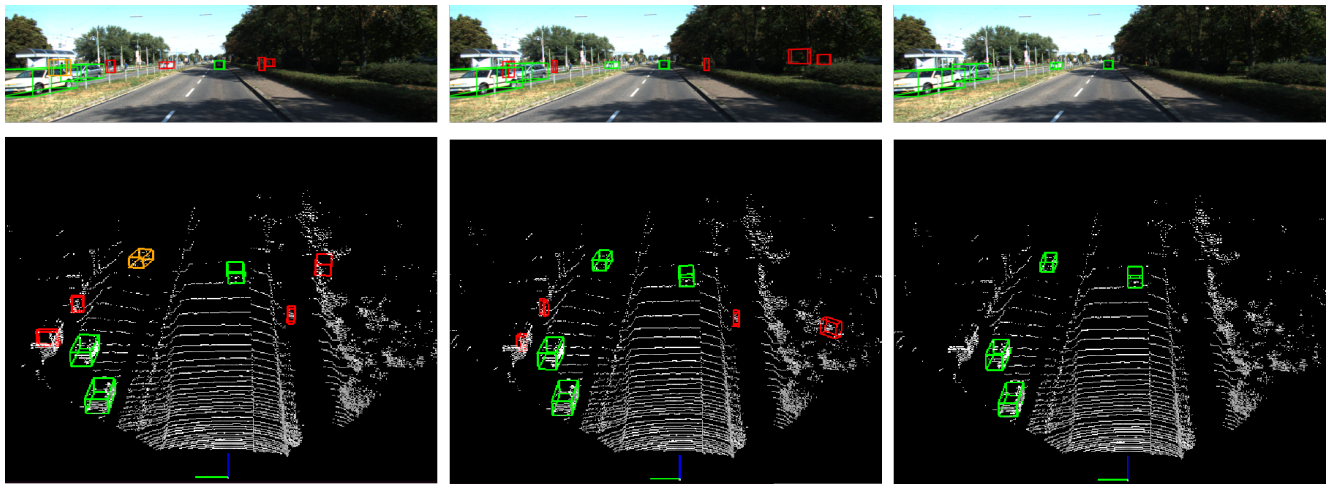


(c)

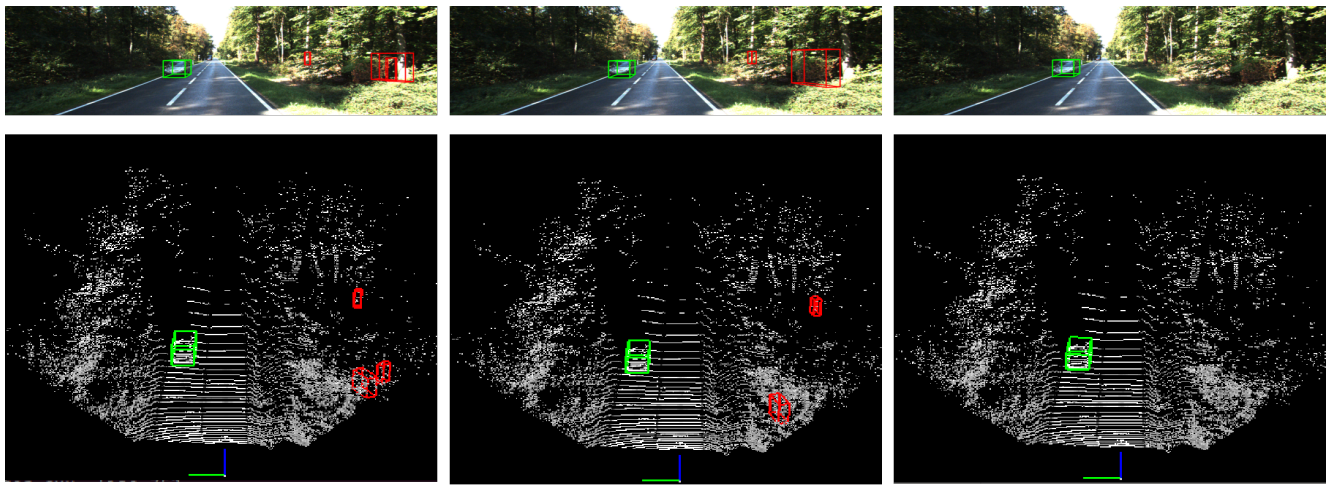
Figure 8: Qualitative comparison of 3D object detection on KITTI validation set. Left: baseline [33]. Middle: our approach. Right: ground truth. Green boxes: true positives (TP). Orange boxes: missed boxes (false negatives, FN). Red boxes: false positives (FP).



(a)



(b)



(c)

Figure 9: Qualitative comparison of 3D object detection on KITTI validation set. Left: baseline [33]. Middle: our approach. Right: ground truth. Green boxes: true positives (TP). Orange boxes: missed boxes (false negatives, FN). Red boxes: false positives (FP).