Pillar-based Object Detection for Autonomous Driving

Yue Wang[1,2], Alireza Fathi[2], Abhijit Kundu[2], David A. Ross[2], Caroline Pantofaru[2], Tom Funkhouser[2], and Justin Solomon[1]

[1] MIT
{yuewangx,jsolomon}@mit.edu
[2] Google
{alirezafathi,abhijitkundu,dross,cpantofaru,tfunkhouser}@google.com

Abstract. We present a simple and flexible object detection framework optimized for autonomous driving. Building on the observation that point clouds in this application are extremely sparse, we propose a practical *pillar-based* approach to fix the imbalance issue caused by anchors. In particular, our algorithm incorporates a cylindrical projection into multi-view feature learning, predicts bounding box parameters per pillar rather than per point or per anchor, and includes an aligned pillar-to-point projection module to improve the final prediction. Our anchor-free approach avoids hyperparameter search associated with past methods, simplifying 3D object detection while significantly improving upon state-of-the-art.

1 Introduction

3D object detection is a central component of perception systems for autonomous driving, used to identify pedestrians, vehicles, obstacles, and other key features of the environment around a car. Ongoing development of object detection methods for vision, graphics, and other domain areas has led to steady improvement in the performance and reliability of these systems as they transition from research to production.

Most 3D object detection algorithms project points to a single prescribed view for feature learning. These views—e.g., the "bird's eye" view of the environment around the car—are not necessarily optimal for distinguishing objects of interest. After computing features, these methods typically make *anchor-based* predictions of the locations and poses of objects in the scene. Anchors provide useful position and pose priors, but they lead to learning algorithms with many hyperparameters and potentially unstable training.

Two popular architectures typifying this approach are PointPillars [16] and multi-view fusion (MVF) [51], which achieve top efficiency and performance on recent 3D object detection benchmarks. These methods use learning representations built from birds-eye view pillars above the ground plane. MVF also benefits from complementary information provided by a spherical view. These methods, however, predict parameters of a bounding box per anchor. Hyperparameters of anchors need to be tuned case-by-case for different tasks/datasets,

reducing practicality. Moreover, anchors are sparsely distributed in the scene, leading to a significant class imbalance. An anchor is assigned as positive when its intersection-over-union (IoU) with a ground-truth box reaches above prescribed threshold; the number of positive anchors is less than 0.1% of all anchors in a typical point cloud.

As an alternative, we introduce a fully pillar-based (anchor-free) object detection model for autonomous driving. In principle, our method is an intuitive extension of PointPillars [16] and MVF [51] that uses pillar representations in multi-view feature learning and in pose estimation. In contrast to past works, we find that predicting box parameters per anchor is neither necessary nor effective for 3D object detection in autonomous driving. A critical new component of our model is a per-pillar prediction network, removing the necessity of anchor assignment. For each birds-eye view pillar, the model directly predicts the position and pose of the best possible box. This component improves performance and is significantly simpler than current state-of-the-art 3D object detection pipelines.

In addition to introducing this pillar-based object detection approach, we also propose ways to address other problems with previous methods. For example, we find the spherical projection in MVF [51] causes unnecessary distortion of scenes and can actually degrade detection performance. So, we complement the conventional birds-eye view with a new cylindrical view, which does not suffer from perspective distortions. We also observe that current methods for pillar-to-point projection suffer from spatial aliasing, which we improve with bilinear interpolation.

To investigate the performance of our method, we train and test the model on the Waymo Open Dataset [39]. Compared to the top performers on this dataset [27, 51, 16], we show significant improvements by 6.87 3D mAP and 6.71 2D mAP for vehicle detection. We provide ablation studies to analyze the contribution of each proposed module in §4 and show that each outperforms its counterpart by a large margin.

Contributions. We summarize our key contributions as follows:

- We present a fully pillar-based model for high-quality 3D object detection. The model achieves state-of-the-art results on the most challenging autonomous driving dataset.
- We design an pillar-based box prediction paradigm for object detection, which is much simpler and stronger than its anchor-based and/or point-based counterpart.
- We analyze the multi-view feature learning module and find a cylindrical view is the best complementary view to a birds-eye view.
- We use bilinear interpolation in pillar-to-point projection to avoid quantization errors.
- We release our code to facilitate reproducibility and future research: https://github.com/WangYueFt/pillar-od.

2 Related Work

Methods for object detection are highly successful in 2D visual recognition [8, 7, 34, 11, 21, 33]. They generally involve two aspects: backbone networks and detection heads. The input image is passed through a backbone network to learn latent features, while the detection heads make predictions of bounding boxes based on the features. In 3D, due to the sparsity of the data, many special considerations are taken to improve both efficiency and performance. Below, we discuss related works on general object detection, as well as more general methods relevant to learning on point clouds.

2D object detection. RCNN [8] pioneers the modern two-stage approach to object detection; more recent models often follow a similar template. RCNN uses a simple selective search to find regions of interest (region proposals) and subsequently applies a convolutional neural network (CNN) to bottom-up region proposals to regress bounding box parameters.

RCNN can be inefficient because it applies a CNN to each region proposal, or image patch. Fast RCNN [7] addresses this problem by sharing features for region proposals from the same image: it passes the image in a one-shot fashion through the CNN, and then region features are cropped and resized from the shared feature map. Faster RCNN [34] further improves speed and performance by replacing the selective search with region proposal networks (RPN), whose features can be shared.

Mask RCNN [11] is built on top of Faster RCNN. In addition to box prediction, it adds another pathway for mask prediction, enabling enables object detection, semantic segmentation, and instance segmentation using a single pipeline. Rather than using ROIPool [7] to resize feature patch to a fixed size grid, Mask RCNN proposes using bilinear interpolation (ROIAlign) to avoid quantization error. Beyond significant structural changes in the general two-stage object detection models, extensions using machinery from image processing and shape registration include: exploiting multi-scale information using feature pyramids [19], iterative refinement of box prediction [2], and using deformable convolutions [6] to get an adaptive receptive field. Recent works [53, 41, 50] also show anchor-free methods achieve comparable results to existing two-stage object detection models in 2D.

In addition to two-stage object detection, many works aim to design real-time object detection models via one-stage algorithms. These methods densely place anchors that define position priors and size priors in the image and then associate each anchor with the ground-truth using an intersection-over-union (IoU) threshold. The networks classify each anchor and regress parameters of anchors; non-maximum suppression (NMS) removes redundant predictions. SSD [21] and YOLO [32, 33] are representative examples of this approach. RetinaNet [20] is built on the observation that the extreme foreground-background class imbalance encountered during training causes one-stage detectors trailed the accuracy of two-stage detectors. It proposes a focal loss to amplify a sparse set of hard examples and to prevent easy negatives from overwhelming the detector during training. Similar to image object detection, we also find the imbalance issue

causes instability in 3D object detection. In contrast to RetinaNet, however, we replace anchors with pillar-centric predictions to alleviate imbalance.

Learning on point clouds. Point clouds provide a natural representation of 3D shapes [3] and scenes. Due to irregularity and symmetry under reordering, however, defining convolution-like operations on point clouds is difficult.

PointNet [30] exemplifies a broad class of deep learning architectures that operate on raw point clouds. It uses a shared multi-layer perceptron (MLP) to lift points to high-demensional space and then aggregates features of points using symmetric set function. PointNet++ [31] exploits local context by building hierarchical abstraction of point clouds. DGCNN [44] uses graph neural networks (GCN) on the k-nearest neighbor graphs to learn geometric features. KPConv [40] defines a set of kernel points to perform deformable convolutions, providing more flexibility than fixed grid convolutions. PCNN [1] defines extension and restriction operations, mapping point cloud functions to volumetric functions and vice versa. SPLATNet [38] renders point clouds to lattice grid and perform lattice convolutions.

FlowNet3D [22] and MeteorNet [23] adopt these methods and learn pointwise flows on dynamical point clouds. In addition to high-level point cloud recognition, recent works [42, 43, 9, 35] tackle low-level registration problems using point cloud networks and show significant improvements over traditional optimization-based methods. These point-based approaches, however, are constrained by the number of points in the point clouds and cannot scale to large-scale settings such as autonomous driving. To that end, sparse 3D convolutions [10] have been proposed to apply 3D convolutions sparsely only on areas where points reside. Minkowski ConvNet [5] generalizes the definition of high-dimensional sparse convolution and improves 3D temporal perception.

3D object detection. The community has seen rising interest in 3D object detection thanks to the popularity of autonomous driving. VoxelNet [52] proposes a generic one-stage model for 3D object detection. It voxelizes the whole point cloud and uses dense 3D convolutions to perform feature learning. To address the efficiency issue, PIXOR [49] and PointPillars [16] both organize in vertical columns (pillars); a PointNet is used to transform features from points to pillars. MVF [51] learns to utilize the complementary information from both birds-eye view pillars and perspective view pillars. Complex-YOLO [37] extends YOLO to 3D scenarios and achieves real-time 3D perception; PointRCNN [36], on the other hand, adopts a RCNN-style detection pipeline. Rather than working in 3D, LaserNet [25] performs convolutions in raw range scans. Beyond point clouds only, recent works [15, 4, 46] combine point clouds with camera images to utilize additional information. Frustum-PointNet [29] leverages 2D object detectors to form a frustum crop of points and then uses a PointNet to aggregate features from points in frustum. [18] designs an end-to-end learnable architecture that exploits continuous convolutions to have better fused feature maps in every level. In addition to visual inputs, [48] shows that High-Definition (HD) maps provide strong priors that can boost the performance of 3D object detectors. [17] argues multi-tasking training can help the network to learn better representations

than single-tasking. Beyond supervised learning, [45] investigates how to learn a perception model for unknown classes.

3 Method

In this section, we detail our approach to object pillar-based detection. We establish preliminaries about the pillar-point projection, PointPillars, and MVF in §3.1 and summarize our model in §3.2. Next, we discuss three critical new components of our model: cylindrical view projection (§3.3), a pillar-based prediction paradigm (§3.4), and a pillar-to-point projection module with bilinear interpolation (§3.5). Finally, we introduce the loss function in §3.6. For ease of comparison to previous work, we use the same notation as MVF [51].

3.1 Preliminaries

We consider a three-dimensional point cloud with N points $P = \{p_i\}_{i=0}^{N-1} \subseteq \mathbb{R}^3$ with K-dimensional features $F = \{f_i\}_{i=0}^{N-1} \subseteq \mathbb{R}^K$. We define two functions $F_V(p_i)$ and $F_P(v_j)$, where $F_V(p_i)$ returns the index j of p_i's corresponding pillar v_j and $F_P(v_j)$ gives the set of points in v_j. When projecting features from points to pillars, multiple points can potentially fall into the same pillar. To aggregate features from points in a pillar, a PointNet [30] (denoted as PN) is used to aggregate features from points to get pillar-wise features, where

$$f_j^{\text{pillar}} = \text{PN}(\{f_i | \forall p_i \in F_P(v_j)\}). \tag{1}$$

Then, pillar-wise features are further transformed through an additional convolutional neural network (CNN), notated $\phi^{\text{pillar}} = \Phi(f^{\text{pillar}})$ where Φ denotes the CNN. To retrieve point-wise features from pillars, the pillar-to-point feature projection is given by

$$f_i^{\text{point}} = f_j^{\text{pillar}} \quad \text{and} \quad \phi_i^{\text{point}} = \phi_j^{\text{pillar}}, \quad \text{where} \quad j = F_V(p_i). \tag{2}$$

While PointPillars only considers birds-eye view pillars and makes predictions based on the birds-eye feature map, MVF also incorporates spherical pillars. Given a point $p_i = (x_i, y_i, z_i)$, its spherical coordinates $(\varphi_i, \theta_i, d_i)$ are defined via

$$\varphi_i = \arctan \frac{y_i}{x_i} \qquad \theta_i = \arccos \frac{z_i}{d_i} \qquad d_i = \sqrt{x_i^2 + y_i^2 + z_i^2}. \tag{3}$$

We can denote the established point-pillar transformations as $(F_V^{\text{bev}}(p_i), F_P^{\text{bev}}(v_j))$ and $(F_V^{\text{spv}}(p_i), F_P^{\text{spv}}(v_j))$ for the birds-eye view and the spherical view, respectively. In MVF, pillar-wise features are learned independently in two views; then the point-wise features are gathered from those views using Eq. 2. Next, the fused point-wise features are projected to birds-eye view again and embedded through a CNN as in PointPillars.

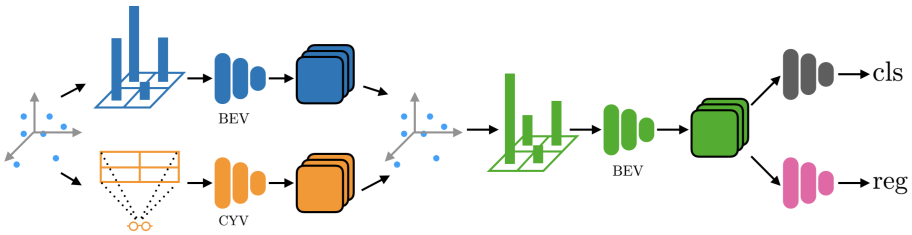


Fig. 1. Overall architecture of the proposed model: a point cloud is projected to BEV and CYV respectively; then, view-specific feature learning is done in each view; third, features from multiple views are aggregated; next, point-wise features are projected to BEV again for further embedding; finally, in BEV, a classification network and a regression network make predictions per pillar. BEV: birds-eye view; CYV: cylindrical view; cls: per pillar classification target; reg: per pillar regression target.

The final detection head for both PointPillars and MVF is an anchor-based module. Anchors, parameterized by $(x^a, y^a, z^a, l^a, w^a, h^a, \theta^a)$, are densely placed in each cell of the final feature map. During pre-processing, an anchor is marked as "positive" if its intersection-over-union (IoU) with a ground-truth box is above a prescribed positive threshold, and "negative" if its IoU is below a negative threshold; otherwise, the anchor is excluded in the final loss computation.

3.2 Overall architecture

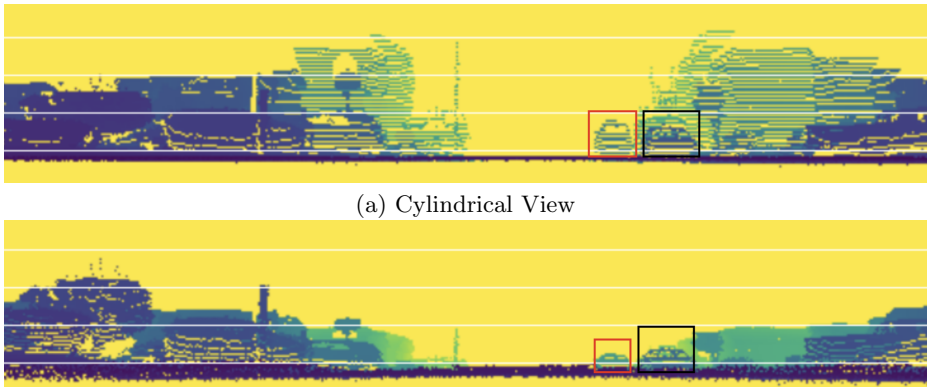
An overview of our proposed model is shown in Figure 1. The input point cloud is passed through the birds-eye view network and the cylindrical view network individually. Then, features from different views are aggregated in the same way with MVF. Next, features are projected back to birds-eye view and passed through additional convolutional layers. Finally, a classification network and a regression network make the final predictions per birds-eye view pillar. We *do not* use anchors in any stage. We describe each module in detail below.

3.3 Cylindrical view

In this section, we formulate the cylindrical view projection. The cylindrical coordinates (ρ_i, φ_i, z_i) of a point p_i is given by the following:

$$\rho_i = \sqrt{x_i^2 + y_i^2} \qquad \varphi_i = \arctan \frac{y_i}{x_i} \qquad z_i = z_i. \tag{4}$$

Cylindrical pillars are generated by grouping points that have the same φ and z coordinates. Although it is closely related to the spherical view, the cylindrical view does not introduce distortion in the Z-axis. We show an example in Figure 2, where cars are clearly visible in the cylindrical view but not distinguishable in the spherical view. In addition, objects in spherical view are no longer in their physical scales—e.g., distant cars become small.



(a) Cylindrical View

(b) Spherical View

Fig. 2. Comparison of (a) cylindrical view projection and (b) spherical view projection. We label two example cars in these views. Objects in spherical view are distorted (in Z-axis) and no longer in physical scale.

3.4 Pillar-based prediction

The pillar-based prediction module consists of two networks: a classification network and a regression network. They both take the final pillar features ϕ^{pillar} from birds-eye view. The prediction targets are given by

$$\mathrm{p} = f_{\text{cls}}(\phi^{\text{pillar}}) \quad \text{and} \quad (\Delta_x, \Delta_y, \Delta_z, \Delta_l, \Delta_w, \Delta_h, \theta^p) = f_{\text{reg}}(\phi^{\text{pillar}}), \tag{5}$$

where p denotes the probability of whether a pillar is a positive match to a ground-truth box and $(\Delta_x, \Delta_y, \Delta_z, \Delta_l, \Delta_w, \Delta_h, \theta^p)$ are the regression targets for position, size, and heading angle of the bounding box.

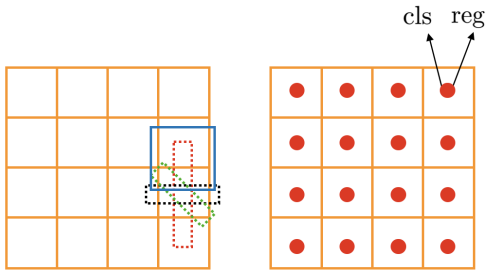
The differences between anchor-based method and pillar-based method are explained in Figure 3. Rather than associating a pillar with an anchor and predicting the targets with reference to the anchor, the model (on the right) directly makes a prediction per pillar.

3.5 Bilinear interpolation

The pillar-to-point projection used in PointPillars [16] and MVF [51] can be thought of as a version of nearest neighbor interpolation, however, which often introduces quantization errors. Rather than performing nearest neighbor interpolation, we propose using bilinear interpolation to learn spatially-consistent features. We describe the formulation of nearest neighbor interpolation and bilinear interpolation in the context of pillar-to-point projection below.

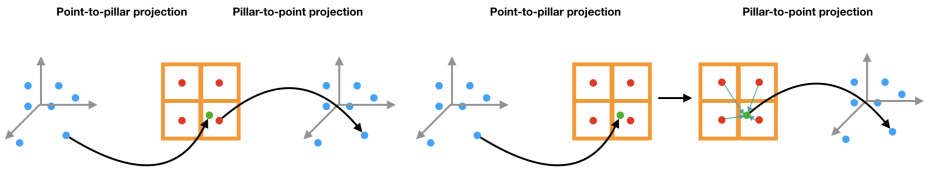
As shown in Figure 4 (a), we denote the center of a pillar v_j as c_j where c_j is defined by its 2D or 3D coordinates. Then, the point-to-pillar mapping function is given by

$$F_V(p_i) = j, \quad \text{where } \|p_i - c_j\| \leq \|p_i - c_k\| \quad \forall k \tag{6}$$



(a) Prediction per anchor (b) Prediction per pillar

Fig. 3. Differences between prediction per anchor and prediction per pillar. (a) Multiple anchors with different sizes and rotations are densely placed in each cell. Anchor-based models predict parameters of bounding box for the positive anchor. For ease of visualization, we only show three anchors. Grid (in orange): birds-eye view pillar; dashed box (in red): a positive match; dashed box (in black): a negative match; dashed box (in green): invalid anchors because their IoUs are above negative threshold and below positive threshold. (b) For each pillar (center), we predict whether it is within a box and the box parameters. Dots (in red): pillar center.



(a) Nearest neighbor interpolation (b) Bilinear interpolation

Fig. 4. Comparison between nearest neighbor interpolation and bilinear interpolation in pillar-to-point projection. Rectangles (in orange): birds-eye view pillars; dots (in blue): points in 3D Cartesian coordinates; dots (in green): points projected to pillar frame; dots (in red): centers of pillars.

and $\|\cdot\|$ denotes the $\mathcal{L}_2$ norm. When querying the features for a point p_i from a collection pillars, we determine the corresponding pillar v_j by checking F_V and copy the features of v_j to p_i—that is $\phi_i^{\text{point}} = \phi_j^{\text{pillar}}$.

This operation, though straightforward, leads to undesired spatial misalignment: if two points p_i and p_j with different spatial locations reside in the same pillar, their pillar-to-point features are the same. To address this issue, we propose using bilinear interpolation for the pillar-to-point projection. As shown in Figure 4 (b), the bilinear interpolation provides consistent spatial mapping between points and pillars.

3.6 Loss function

We use the same loss function as in SECOND [47], PointPillars [16], and MVF [51]. The loss function consists of two terms: a pillar classification loss and a pillar

regression loss. The ground-truth bounding box is parametrized as $(x^g, y^g, z^g, l^g, w^g, h^g, \theta^g)$; the center of pillar is (x^p, y^p, z^p); and the prediction targets for the bounding box are $(\Delta_x, \Delta_y, \Delta_z, \Delta_l, \Delta_w, \Delta_h, \theta^p)$ as in §3.4. Then, the regression loss is:

$$\begin{aligned} L_{\text{reg}} = \text{SmoothL1}(\theta^p - \theta^g) + \sum_{r \in \{x,y,z\}} \text{SmoothL1}(r^p - r^g - \Delta_r) \\ + \sum_{r \in \{l,w,h\}} \text{SmoothL1}(\log(r^g) - \Delta_r) \end{aligned} \tag{7}$$

where

$$\text{SmoothL1}(d) = \begin{cases} 0.5 \cdot d^2 \cdot \sigma^2, & \text{if} \quad |d| < \frac{1}{\sigma^2} \\ |d| - \frac{1}{2\sigma^2}, & \text{otherwise.} \end{cases} \tag{8}$$

We take $\sigma = 3.0$. For pillar classification, we adopt the focal loss [20]:

$$L_{\text{cls}} = -\alpha(1 - \text{p})^\gamma \log \text{p}. \tag{9}$$

We use $\alpha = 0.25$ and $\gamma = 2$, as recommended by [20].

4 Experiments

Our experiments are divided into four parts. First, we demonstrate performance of our model for vehicle and pedestrian detection on the Waymo Open Dataset [39] in §4.1. Then, we compare anchor-, point-, and pillar-based detection heads in §4.2. We compare different combinations of views in §4.3. Finally, we test the effects of bilinear interpolation in §4.4.

Dataset. The *Waymo Open Dataset* [39] is the largest publicly-available 3D object detection dataset for autonomous driving. The dataset provides 1000 sequences total; each sequence contains roughly 200 frames. The training set consists of 798 sequences with 158,361 frames, containing 4.81M vehicle and 2.22M pedestrian boxes. The validation set consists of 202 sequences with 40,077 frames, containing 1.25M vehicle and 539K pedestrian boxes. The detection range is set to $[-75.2, 75.2]$ meters (m) horizontally and $[-3, 3]$ m vertically.

Metrics. For our experiments, we adopt the official evaluation protocols from the Waymo Open Dataset. In particular, we employ the 3D and BEV mean average precision (mAP) metrics. The orientation-aware IoU threshold is 0.7 for vehicles and 0.5 for pedestrians. We also break down the metrics according to the distances between the origin and ground-truth boxes: 0m–30m, 30m–50m, and 50m–infinity (Inf). The dataset is split based on the number of points in each box: LEVEL_1 denotes boxes that have more than 5 points while LEVEL_2 denotes boxes that have 1-5 points. Following StarNet [27], MVF [51], and PointPillars [16] as reimplemented in [39], we evaluate our models on LEVEL_1 boxes.

Implementation details. Our model consists of three parts: a multi-view feature learning network; a birds-eye view PointPillar [16] backbone; and a per-pillar prediction network. In the multi-view feature learning network, we project point features to both birds-eye view pillars and cylindrical pillars. For each view, we apply three ResNet [12] layers with strides $[1, 2, 2]$, which gradually downsamples the input feature to $1/1$, $1/2$, and $1/4$ of the original feature map. Then, we project the pillar-wise features to points using bilinear interpolation and concatenate features from both views and from a parallel PointNet with one fully-connected layer. Then, we transform the per-point features to birds-eye pillars and use a PointPillars [16] backbone with three blocks to further improve the representations. The three blocks have $[4, 6, 6]$ convolutional layers, with dimensions $[128, 128, 256]$. Finally, for each pillar, the model predicts the categorical label using a classification head and 7 DoF parameters of its closest box using a regression head. The classification head and regression head both have four convolutional layers with 128 hidden dimensions. We use BatchNorm [13] and ReLU [26] after every convolutional layer.

Training. We use the Adam [14] optimizer to train the model. The learning rate is initially 3×10^{-4} and then linearly increased to 3×10^{-3} in the first 5 epochs. Finally, the learning rate is decreased to 3×10^{-6} using cosine scheduling [24]. We train the model for 75 epochs in 64 TPU cores.

Inference. The input point clouds pass through the whole model once to get the initial predictions. Then, we use non-maximum suppression (NMS) [7] to remove redundant bounding boxes. The oriented IoU threshold of NMS is 0.7 for vehicle and 0.2 for pedestrian. We keep the top 200 boxes for metric computation. The size of our model is on a par with MVF; the model runs at 15 frames per second (FPS) on a Tesla V100.

4.1 Results compared to state-of-the-art

We compare the proposed method to top-performing methods on the Waymo Open Dataset. StarNet [27] is a purely point-based method with a small receptive field, which performs well for small objects such as pedestrians but poorly for large objects such as vehicles. LaserNet [25] operates on range images, which is similar to our cylindrical view feature learning. Although PointPillars [16] does not evaluate on this dataset, MVF [51] and the Waymo Open Dataset [39] both re-implement the PointPillars. So we adopt the results from MVF [51] and [39]. The re-implementation from [39] uses larger feature map resolution in the first PointPillars block; therefore, it outperforms the re-implementation from MVF [51].

MVF [51] extends PointPillars [16] with the same backbone networks and multi-view feature learning. We use the same backbone networks with PointPillars [16] and MVF [51].

As shown in Table 1 and Table 2, we achieve significantly better results for both pedestrians and vehicles. Especially for distant vehicles (30m–Inf), the improvements are more significant.This is inline with our hypothesis: in distant areas, anchors are less possible to match to a ground-truth box; therefore, the

Method	BEV mAP (IoU=0.7)				3D mAP (IoU=0.7)			
	Overall	0 - 30m	30 - 50m	50m - Inf	Overall	0 - 30m	30 - 50m	50m - Inf
StarNet [27]	-	-	-	-	53.7	-	-	-
LaserNet [25]	71.57	92.94	74.92	48.87	55.1	84.9	53.11	23.92
PointPillars¶ [16]	80.4	92.0	77.6	62.7	62.2	81.8	55.7	31.2
PointPillars‡ [16]	70.59	86.63	69.34	49.3	54.25	76.31	48.08	24.21
PointPillars† [16]	75.57	92.1	74.06	55.47	56.62	81.01	51.75	27.94
MVF [51]	80.4	93.59	79.21	63.09	62.93	86.3	60.2	36.02
Ours	**87.11**	**95.78**	**84.74**	**72.12**	**69.8**	**88.53**	**66.5**	**42.93**
Improvements	**+6.71**	**+2.19**	**+5.53**	**+9.03**	**+6.87**	**+2.23**	**+6.3**	**+6.91**

Table 1. Results on vehicle. ¶: re-implemented by [39], the feature map in the first PointPillars block is two times as big as in others; ‡: our re-implementation; †: re-implemented by [51].

Method	BEV mAP (IoU=0.5)				3D mAP (IoU=0.5)			
	Overall	0 - 30m	30 - 50m	50m - Inf	Overall	0 - 30m	30 - 50m	50m - Inf
StarNet [27]	-	-	-	-	66.8	-	-	-
LaserNet [25]	70.01	78.24	69.47	52.68	63.4	73.47	61.55	42.69
PointPillars¶ [16]	68.7	75.0	66.6	58.7	60.0	68.9	57.6	46.0
PointPillars† [16]	68.57	75.02	67.11	53.86	59.25	67.99	57.01	41.29
MVF [51]	74.38	80.01	72.98	62.51	65.33	72.51	63.35	50.62
Ours	**78.53**	**83.56**	**78.7**	**65.86**	**72.51**	**79.34**	**72.14**	**56.77**
Improvements	**+4.15**	**+3.55**	**+5.72**	**+3.35**	**+5.71**	**+6.83**	**+8.77**	**+6.15**

Table 2. Results on pedestrian. ¶: re-implemented by [39]. †: re-implemented by [51].

imbalance problem is more serious. Also, to verify the improvements are *not* due to differences in training protocol, we re-implement PointPillars; using our training protocol, it achieves 54.25 3D mAP and 70.59 2d mAP, which are worse than the re-implementations in [51] and [39] . Therefore, we can conclude the improvements are due to the three new components added by our proposed model.

4.2 Comparing anchor-based, point-based, and pillar-based prediction

In this experiment, we compare to alternative means of making predictions: predicting box parameters per anchor or per point. For these three detection head choices, we use the same overall architecture with experiments in §4.1. We conduct this ablation study on vehicle detection.

Anchor-based model. We use the parameters and matching strategy from PointPillars [51] and MVF [51]. Each class anchor is described by a width, length, height, and center position and is applied at two orientations: 0° and 90°. Anchors are matched to ground-truth boxes using the 2D IoU with the fol-

Method	BEV mAP (IoU=0.7)				3D mAP (IoU=0.7)			
	Overall	0 - 30m	30 - 50m	50m - Inf	Overall	0 - 30m	30 - 50m	50m - Inf
Anchor-based	78.84	91.91	74.99	59.59	59.78	82.69	53.38	31.02
Point-based	79.77	92.35	76.58	60.00	60.6	83.66	55.48	30.95
Pillar-based	**87.11**	**95.78**	**84.74**	**72.12**	**69.8**	**88.53**	**66.5**	**42.93**

Table 3. Comparison of making prediction per anchor, per point, or per pillar.

lowing rules: a positive match is either the highest with a ground truth box, or above the positive match threshold (0.6); while a negative match is below the negative threshold (0.45). All other anchors are ignored in the box parameter prediction. The model is to predict whether a anchor is positive or negative, and width, length, height, heading angle, and center position of the bounding box.

Point-based model. The per-pillar features are projected to points using bilinear interpolation. Then, we assign each point to its surrounding box with the following rules: if a point is inside a bounding box, we assign it as a foreground point; otherwise it is a background point. The model is asked to predict the binary label whether a point is a foreground point or a background point. For positive points, the model also predicts the width, length, height, heading angle, and center offsets (with reference to point positions) of their associated bounding boxes. Conceptually, this point-based model is an instantiation of VoteNet [28] applied to this autonomous driving scenario. The key difference is: the VoteNet [28] uses a PointNet++ [31] backbone while we use a PointPillars [51] backbone.

Pillar-based model. Since we use the same architecture, we take the results from §4.1. As Table 3 shows, anchor-based prediction performs the worst while point-based prediction is slightly better. Our pillar-based prediction is top performing among these three choices. The pillar-based prediction model achieves the best balance between coarse prediction (per anchor) and fine-grained prediction (per point).

4.3 View combinations

In this section, we test different view projections in multi-view feature learning: birds-eye view (BEV), spherical view (SPV), XZ view, cylindrical view (CYV), and their combinations. First, we define the vehicle frame: the X-axis is positive forwards, the Y-axis is positive to the left, and the Z-axis is positive upwards. Then, we can write the coordinates of a point $p = (x, y, z)$ in different views; the range of each view is given in Table 4. The pillars in the corresponding view are generated by projecting points from 3D to 2D using the coordinate transformation. One exception is in XZ view, in which we use separate pillars for positive part and negative part for Y-axis to avoid undesired occlusions.

We show results of different view projections and their combinations in Table 5 for vehicle detection. When using a single view, the cylindrical view achieves

View	Coordinates	Range
3D Cartesian	(x, y, z)	(-75.2, 75.2)m, (-75.2, 75.2)m, (-3, 3)m
BEV	(x, y, z)	(-75.2, 75.2)m, (-75.2, 75.2)m, (-3, 3)m
SPV	$(\arctan(\frac{y}{x}), \arccos(\frac{z}{\sqrt{x^2+y^2+z^2}}), \sqrt{x^2+y^2+z^2})$	(0, 2π), (0.485π, 0.55π), (0, 107)m,
XZ view	(x, y, z)	(-75.2, 75.2)m , (-75.2, 75.2)m, (-3, 3)m
CYV	$(\sqrt{x^2+y^2}, \arctan(\frac{y}{x}), z)$	(0, 107)m, (0, 2π), (-3, 3)m

Table 4. View projection

Method	BEV mAP (IoU=0.7)				3D mAP (IoU=0.7)			
	Overall	0 - 30m	30 - 50m	50m - Inf	Overall	0 - 30m	30 - 50m	50m - Inf
BEV	81.58	92.69	78.64	63.52	61.86	83.61	56.91	33.53
SPV	81.58	93.7	78.43	63.2	62.08	83.31	56.59	34.05
XZ	81.49	94.03	78.04	62.32	61.67	84.64	55.01	32.06
CYV	83.43	95.21	81.49	66.77	64.77	87.09	60.91	37.99
BEV + SPV	85.09	95.19	82.01	69.13	66.31	86.56	61.15	39.36
BEV + XZ	82.45	94.1	79.19	63.91	62.76	85.08	56.8	33.36
BEV + CYV	**87.11**	**95.78**	**84.74**	**72.12**	**69.8**	**88.53**	**66.5**	**42.93**

Table 5. Ablation on view combinations.

significantly better results than the alternatives, especially in the long-range detection case (50m–Inf). When combining with the birds-eye view, the cylindrical view still outperforms others in all metrics. The spherical view, albeit similar to cylindrical view, introduces distortion in Z-axis, degrading performance relative to the cylindrical view. On the other hand, the XZ view does not distort the Z-axis, but occlusions in Y-axis prevent it from achieving as strong results as the cylindrical view. We also test with additional view combinations (such as using birds-eye view, spherical view, and cylindrical view) and do not observe any improvements over combining just the birds-eye view and the cylindrical view.

4.4 Bilinear interpolation or nearest neighbor interpolation?

In this section, we compare bilinear interpolation to nearest neighbor interpolation in pillar-to-point projection (for vechile detection). The architectures remain the same for both alternatives except the way we project multi-view features from pillars to points: In nearest neighbor interpolation, for each query point, we sample its closest pillar center and copy the pillar features to it, while in bilinear interpolation, we sample its four pillar neighbors and take a weighted average of the corresponding pillar features. Table 6 shows bilinear interpolation systematically outperforms its counterpart in all metrics. This observation is consistent with the comparison of ROIAlign [11] and ROIPool [34] in 2D.

Method	BEV mAP (IoU=0.7)				3D mAP (IoU=0.7)			
	Overall	0 - 30m	30 - 50m	50m - Inf	Overall	0 - 30m	30 - 50m	50m - Inf
Nearest neighbor	84.67	94.42	79.2	65.77	64.76	85.55	59.21	35.63
Bilinear	**87.11**	**95.78**	**84.74**	**72.12**	**69.8**	**88.53**	**66.5**	**42.93**

Table 6. Comparing bilinear interpolation and nearest neighbor projection.

5 Discussion

We present a pillar-based object detection pipeline for autonomous driving. Our model achieves state-of-the-art results on the largest publicly-available 3D object detection dataset. The success of our model suggests many designs from 2D object detection/visual recognition are *not* directly applicable to 3D scenarios. In addition, we find that learning features in correct views is import to the performance of the model.

Our experiments also suggest several avenues for future work. For example, rather than hand-designing a view projection as we do in §3.3, learning an optimal view transformation from data may provide further performance improvements. Learning features using 3D sparse convolutions rather than 2D convolutions could improve performance as well. Also, following two-stage object detection models designed for images, adding a refinement step might increase the performance for small objects.

Finally, we hope to find more applications of the proposed model beyond object detection. For example, we could incorporate instance segmentation, which may help with fine-grained 3D recognition and robotic manipulation.

6 Acknowledgements

Yue Wag, Justin Solomon, and the MIT Geometric Data Processing group acknowledge the generous support of Army Research Office grants W911NF1710068 and W911NF2010168, of Air Force Office of Scientific Research award FA9550-19-1-031, of National Science Foundation grant IIS-1838071, from the MIT–IBM Watson AI Laboratory, from the Toyota–CSAIL Joint Research Center, from gifts from Google and Adobe Systems, and from the Skoltech–MIT Next Generation Program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these organizations.

References

1. Atzmon, M., Maron, H., Lipman, Y.: Point convolutional neural networks by extension operators. ACM Transaction on Graphics (TOG) (2018)
2. Cai, Z., Vasconcelos, N.: Cascade R-CNN: Delving into high quality object detection. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
3. Chang, A.X., Funkhouser, T.A., Guibas, L.J., Hanrahan, P., Huang, Q.X., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An information-rich 3d model repository. CoRR (2015)
4. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3d object detection network for autonomous driving. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
5. Choy, C., Gwak, J., Savarese, S.: 4d spatio-temporal ConvNets: Minkowski convolutional neural networks. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
6. Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y.: Deformable convolutional networks. In: The International Conference on Computer Vision (ICCV) (2017)
7. Girshick, R.: Fast R-CNN. In: The International Conference on Computer Vision (ICCV) (2015)
8. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2014)
9. Goforth, H., Aoki, Y., Srivatsan, R.A., Lucey, S.: PointNetLK: Robust & efficient point cloud registration using PointNet. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
10. Graham, B., Engelcke, M., van der Maaten, L.: 3d semantic segmentation with submanifold sparse convolutional networks. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
11. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask R-CNN. In: The International Conference on Computer Vision (ICCV) (2017)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition Recognition (CVPR) (2016)
13. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: The International Conference on Machine Learning (ICML) (2015)
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: The International Conference on Learning Representations (ICLR) (2014)
15. Ku, J., Mozifian, M., Lee, J., Harakeh, A., Waslander, S.: Joint 3d proposal generation and object detection from view aggregation. In: The International Conference on Intelligent Robots and Systems (IROS) (2018)
16. Lang, A.H., Vora, S., Caesar, H., Zhou, L., Yang, J., Beijbom, O.: Pointpillars: Fast encoders for object detection from point clouds. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
17. Liang, M., Yang, B., Chen, Y., Hu, R., Urtasun, R.: Multi-task multi-sensor fusion for 3d object detection. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

18. Liang, M., Yang, B., Wang, S., Urtasun, R.: Deep continuous fusion for multi-sensor 3d object detection. In: The European Conference on Computer Vision (ECCV) (2018)
19. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
20. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal Loss for Dense Object Detection. In: The International Conference on Computer Vision (ICCV) (2017)
21. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single shot multibox detector. In: The European Conference on Computer Vision (ECCV) (2016)
22. Liu, X., Qi, C.R., Guibas, L.J.: FlowNet3D: Learning scene flow in 3d point clouds. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
23. Liu, X., Yan, M., Bohg, J.: MeteorNet: Deep learning on dynamic 3d point cloud sequences. In: The International Conference on Computer Vision (ICCV) (2019)
24. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. In: The International Conference on Learning Representations (ICLR) (2017)
25. Meyer, G.P., Laddha, A., Kee, E., Vallespi-Gonzalez, C., Wellington, C.K.: LaserNet: An efficient probabilistic 3d object detector for autonomous driving. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
26. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: The International Conference on Machine Learning (ICML) (2010)
27. Ngiam, J., Caine, B., Han, W., Yang, B., Chai, Y., Sun, P., Zhou, Y., Yi, X., Alsharif, O., Nguyen, P., Chen, Z., Shlens, J., Vasudevan, V.: StarNet: Targeted computation for object detection in point clouds. arXiv (2019)
28. Qi, C.R., Litany, O., He, K., Guibas, L.J.: Deep Hough voting for 3d object detection in point clouds. In: The International Conference on Computer Vision (2019)
29. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum PointNets for 3d object detection from RGB-D data. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
30. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: Deep learning on point sets for 3d classification and segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
31. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: Deep hierarchical feature learning on point sets in a metric space. In: Neural Information Processing Systems (NeurIPS) (2017)
32. Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
33. Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
34. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: Neural Information Processing Systems (NeurIPS) (2015)
35. Sarode, V., Li, X., Goforth, H., Aoki, Y., Dhagat, A., Srivatsan, R.A., Lucey, S., Choset, H.: One framework to register them all: PointNet encoding for point cloud alignment. arXiv (2019)
36. Shi, S., Wang, X., Li, H.: Pointrcnn: 3d object proposal generation and detection from point cloud. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

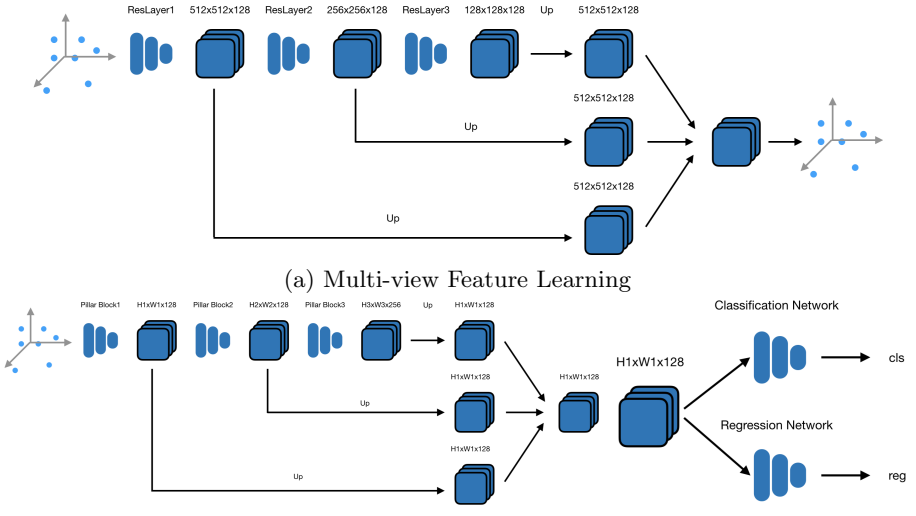
37. Simon, M., Milz, S., Amende, K., Groß, H.M.: Complex-YOLO: Real-time 3d object detection on point clouds. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
38. Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.H., Kautz, J.: Splatnet: Sparse lattice networks for point cloud processing. In: The Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2530–2539 (2018)
39. Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tak-wing Tsui, P., Guo, J.C.Y., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi, A., Zhang, Y., Shlens, J., Chen, Z.F., Anguelov, D.: Scalability in perception for autonomous driving: Waymo open dataset. arXiv (2019)
40. Thomas, H., Qi, C., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: Kpconv: Flexible and deformable convolution for point clouds. In: The International Conference on Computer Vision (ICCV) (2019)
41. Tian, Z., Shen, C., Chen, H., He, T.: Fcos: Fully convolutional one-stage object detection. In: The International Conference on Computer Vision (ICCV) (2019)
42. Wang, Y., Solomon, J.: Deep closest point: Learning representations for point cloud registration. In: The International Conference on Computer Vision (ICCV) (2019)
43. Wang, Y., Solomon, J.: PRNet: Self-supervised learning for partial-to-partial registration. In: Neural Information Processing Systems (NeurIPS) (2019)
44. Wang, Y., Sun, Y., Ziwei Liu, S.E.S., Bronstein, M.M., Solomon, J.M.: Dynamic graph CNN for learning on point clouds. ACM Transactions on Graphics (TOG) **38**, 146 (2019)
45. Wong, K., Wang, S., Ren, M., Liang, M., Urtasun, R.: Identifying unknown instances for autonomous driving. In: The Conference on Robot Learning (CORL) (2019)
46. Xu, D., Anguelov, D., Jain, A.: PointFusion: Deep sensor fusion for 3d bounding box estimation. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
47. Yan, Y., Mao, Y., Li, B.: Second: Sparsely embedded convolutional detection. In: Sensors (2018)
48. Yang, B., Liang, M., Urtasun, R.: HDNET: Exploiting hd maps for 3d object detection. In: The Conference on Robot Learning (CORL) (2018)
49. Yang, B., Luo, W., Urtasun, R.: Pixor: Real-time 3d object detection from point clouds. The Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
50. Zhou, X., Wang, D., Krähenbühl, P.: Objects as points. arXiv (2019)
51. Zhou, Y., Sun, P., Zhang, Y., Anguelov, D., Gao, J., Ouyang, T., Guo, J., Ngiam, J., Vasudevan, V.: End-to-end multi-view fusion for 3d object detection in LiDAR point clouds. In: The Conference on Robot Learning (CoRL) (2019)
52. Zhou, Y., Tuzel, O.: Voxelnet: End-to-end learning for point cloud based 3d object detection. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
53. Zhu, C., He, Y., Savvides, M.: Feature selective anchor-free module for single-shot object detection. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

A Supplementary Material

In this section, we provide details on the parameters of the model. The model consists of three parts: a multi-view feature learning network; a birds-eye view pillar backbone network; and a detection head. We show the pipeline in Figure 5 and the additional parameter specification in Table 7.

Stage	Vehicle Model		Pedestrian Model	
	Kernel	Output Size	Kernel	Output Size
Multi-view Feature Learning	3x3, 128, stride 1	512x512x128	3x3, 128, stride 1	512x512x128
	3x3, 128, stride 2	256x256x128	3x3, 128, stride 2	256x256x128
	3x3, 128, stride 2	128x128x128	3x3, 128, stride 2	128x128x128
Pillar Backbone Block1	3x3, 128, stride 2	256x256x128	3x3, 128, stride 1	512x512x128
	{3x3, 128, stride 1}x3	256x256x128	{3x3, 128, stride 1}x3	512x512x128
Pillar Backbone Block2	3x3, 128, stride 1	256x256x128	3x3, 128, stride 2	256x256x128
	{3x3, 128, stride 1}x5	256x256x128	{3x3, 128, stride 1}x5	256x256x128
Pillar Backbone Block3	3x3, 256, stride 2	128x128x256	3x3, 256, stride 2	128x128x256
	{3x3, 256, stride 1}x5	128x128x256	{3x3, 256, stride 1}x5	128x128x256
Detection Head	{3x3, 256, stride 1}x4	256x256x256	{3x3, 256, stride 1}x4	512x512x256

Table 7. Parameters of convolutional kernels and feature map sizes.



(a) Multi-view Feature Learning

(b) Pillar Backbone

(c) Detection Head

Fig. 5. Details of the proposed model: (a) the multi-view feature learning module, we show the network for one view; (b) Pillar backbone network; (c) the detection head, we show both the classification network and the regression network. For details on the parameters and the feature map sizes, refer to Table 7.