PointPillars: Fast Encoders for Object Detection from Point Clouds

Alex H. Lang Sourabh Vora Holger Caesar Lubing Zhou Jiong Yang
Oscar Beijbom
nuTonomy: an APTIV company
{alex, sourabh, holger, lubing, jiong.yang, oscar}@nutonomy.com

Abstract

Object detection in point clouds is an important aspect of many robotics applications such as autonomous driving. In this paper, we consider the problem of encoding a point cloud into a format appropriate for a downstream detection pipeline. Recent literature suggests two types of encoders; fixed encoders tend to be fast but sacrifice accuracy, while encoders that are learned from data are more accurate, but slower. In this work, we propose PointPillars, a novel encoder which utilizes PointNets to learn a representation of point clouds organized in vertical columns (pillars). While the encoded features can be used with any standard 2D convolutional detection architecture, we further propose a lean downstream network. Extensive experimentation shows that PointPillars outperforms previous encoders with respect to both speed and accuracy by a large margin. Despite only using lidar, our full detection pipeline significantly outperforms the state of the art, even among fusion methods, with respect to both the 3D and bird's eye view KITTI benchmarks. This detection performance is achieved while running at 62 Hz: a 2 - 4 fold runtime improvement. A faster version of our method matches the state of the art at 105 Hz. These benchmarks suggest that PointPillars is an appropriate encoding for object detection in point clouds.

1. Introduction

Deploying autonomous vehicles (AVs) in urban environments poses a difficult technological challenge. Among other tasks, AVs need to detect and track moving objects such as vehicles, pedestrians, and cyclists in realtime. To achieve this, autonomous vehicles rely on several sensors out of which the lidar is arguably the most important. A lidar uses a laser scanner to measure the distance to the environment, thus generating a sparse point cloud representation. Traditionally, a lidar robotics pipeline interprets such point clouds as object detections through a bottom-up pipeline involving background subtraction, followed by spatiotemporal clustering and classification [12, 9].

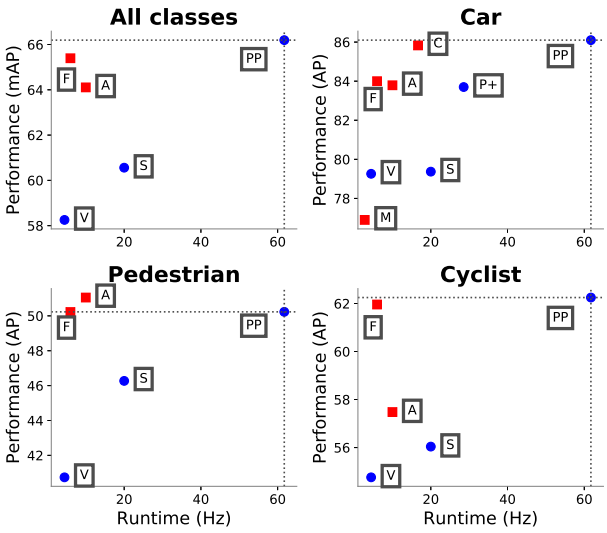

Figure 1. Bird's eye view performance vs speed for our proposed PointPillars, PP method on the KITTI [5] test set. Lidar-only methods drawn as blue circles; lidar & vision methods drawn as red squares. Also drawn are top methods from the KITTI leaderboard: M: MV3D [2], A AVOD [11], C: ContFuse [15], V: VoxelNet [33], F: Frustum PointNet [21], S: SECOND [30], P+ PIXOR++ [31]. PointPillars outperforms all other lidar-only methods in terms of both speed and accuracy by a large margin. It also outperforms all fusion based method except on pedestrians. Similar performance is achieved on the 3D metric (Table 2).

Following the tremendous advances in deep learning methods for computer vision, a large body of literature has investigated to what extent this technology could be applied towards object detection from lidar point clouds [33, 31, 32, 11, 2, 21, 15, 30, 26, 25]. While there are many similarities between the modalities, there are two key differences: 1) the point cloud is a sparse representation, while an image is dense and 2) the point cloud is 3D, while the image is 2D. As a result, object detection from point clouds does not trivially lend itself to standard image convolutional pipelines.

Some early works focus on either using 3D convolutions [3] or a projection of the point cloud into the image

这种俯视视角提供了几个优点：
首先，BEV 保留检测目标对象的尺度信息
其次，BEV 中的卷积保留局部语义信息

如果在图像视图中执行卷积，则会模糊深度信息，
然而，鸟瞰图往往非常稀疏，这使得卷积神经网络的直接应用不切实际且效率低下。

[14]. Recent methods tend to view the lidar point cloud from a bird's eye view (BEV) [2, 11, 33, 32]. This overhead perspective offers several advantages. First, the BEV preserves the object scales. Second, convolutions in BEV preserve the local range information. If one instead performs convolutions in the image view, one is blurring the depth information (Fig. 3 in [28]).

However, the bird's eye view tends to be extremely sparse which makes direct application of convolutional neural networks impractical and inefficient. A common workaround to this problem is to partition the ground plane into a regular grid, for example 10 x 10 cm, and then perform a hand-crafted feature encoding method on the points in each grid cell [2, 11, 26, 32]. However, such methods may be sub-optimal since the hard-coded feature extraction method may not generalize to new configurations without significant engineering efforts. To address these issues, and building on the PointNet design developed by Qi et al. [22], VoxelNet [33] was one of the first methods to truly do end-to-end learning in this domain. VoxelNet divides the space into voxels, applies a PointNet to each voxel, followed by a 3D convolutional middle layer to consolidate the vertical axis, after which a 2D convolutional detection architecture is applied. While the VoxelNet performance is strong, the inference time, at $4.4\ Hz$, is too slow to deploy in real time. Recently SECOND [30] improved the inference speed of VoxelNet but the 3D convolutions remain a bottleneck.

In this work, we propose PointPillars: a method for object detection in 3D that enables end-to-end learning with only 2D convolutional layers. PointPillars uses a novel encoder that learns features on pillars (vertical columns) of the point cloud to predict 3D oriented boxes for objects. There are several advantages of this approach. First, by learning features instead of relying on fixed encoders, PointPillars can leverage the full information represented by the point cloud. Further, by operating on pillars instead of voxels there is no need to tune the binning of the vertical direction by hand. Finally, pillars are fast because all key operations can be formulated as 2D convolutions which are extremely efficient to compute on a GPU. An additional benefit of learning features is that PointPillars requires no hand-tuning to use different point cloud configurations such as multiple lidar scans or even radar point clouds.

We evaluated our PointPillars network on the public KITTI detection challenges which require detection of cars, pedestrians, and cyclists in either BEV or 3D [5]. While our PointPillars network is trained using only lidar point clouds, it dominates the current state of the art including methods that use lidar *and* images, thus establishing new standards for performance on both BEV and 3D detection (Table 1 and Table 2). At the same time, PointPillars runs at 62 Hz, which is 2-4 times faster than previous state of the art (Figure 1). PointPillars further enables a trade off between speed and accuracy; in one setting we match state of the art performance at over 100 Hz (Figure 5). We have also released code[1] to reproduce our results.

1.1. Related Work

1.1.1 Object detection using CNNs

Starting with the seminal work of Girshick et al. [6], it was established that convolutional neural network (CNN) architectures are state of the art for detection in images. The series of papers that followed [24, 7] advocate a two-stage approach to this problem. In the first stage, a region proposal network (RPN) suggests candidate proposals, which are cropped and resized before being classified by a second stage network. Two-stage methods dominated the important vision benchmark datasets such as COCO [17] over single-stage architectures originally proposed by Liu et al. [18]. In a single-stage architecture, a dense set of anchor boxes is regressed and classified in one step into a set of predictions providing a fast and simple architecture. Recently, Lin et al. [16] convincingly argued that with their proposed focal loss function a single stage method is superior to two-stage methods, both in terms of accuracy *and* runtime. In this work, we use a single stage method.

1.1.2 Object detection in lidar point clouds

Object detection in point clouds is an intrinsically three dimensional problem. As such, it is natural to deploy a 3D convolutional network for detection, which is the paradigm of several early works [3, 13]. While providing a straightforward architecture, these methods are slow; e.g. Engelcke et al. [3] require $0.5s$ for inference on a single point cloud. Most recent methods improve the runtime by projecting the 3D point cloud either onto the ground plane [11, 2] or the image plane [14]. In the most common paradigm the point cloud is organized in voxels and the set of voxels in each vertical column is encoded into a fixed-length, hand-crafted, feature encoding to form a pseudo-image which can be processed by a standard image detection architecture. Some notable works include MV3D [2], AVOD [11], PIXOR [32] and Complex YOLO [26] which all use variations on the same fixed encoding paradigm as the first step of their architectures. The first two methods additionally fuse the lidar features with image features to create a multi-modal detector. The fusion step used in MV3D and AVOD forces them to use two-stage detection pipelines, while PIXOR and Complex YOLO use single stage pipelines.

In their seminal work Qi et al. [22, 23] proposed a simple architecture, PointNet, for learning from unordered point sets, which offered a path to full end-to-end learning. VoxelNet [33] is one of the first methods to deploy PointNets for object detection in lidar point clouds. In their method,

[1] https://github.com/nutonomy/second.pytorch

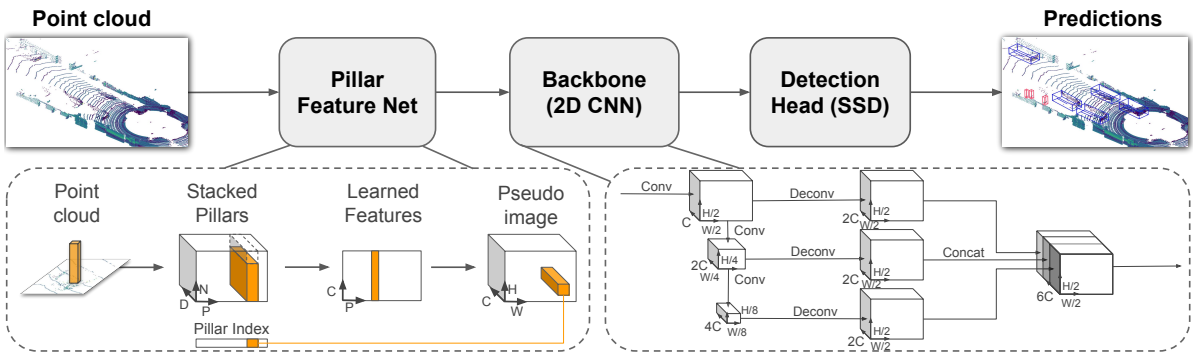

Figure 2. Network overview. The main components of the network are a Pillar Feature Network, Backbone, and SSD Detection Head (see Section 2 for details). The raw point cloud is converted to a stacked pillar tensor and pillar index tensor. The encoder uses the stacked pillars to learn a set of features that can be scattered back to a 2D pseudo-image for a convolutional neural network. The features from the backbone are used by the detection head to predict 3D bounding boxes for objects. Note: we show the car network's backbone dimensions.

PointNets are applied to voxels which are then processed by a set of 3D convolutional layers followed by a 2D backbone and a detection head. This enables end-to-end learning, but like the earlier work that relied on 3D convolutions, VoxelNet is slow, requiring 225ms inference time ($4.4\ Hz$) for a single point cloud. Another recent method, Frustum PointNet [21], uses PointNets to segment and classify the point cloud in a frustum generated from projecting a detection on an image into 3D. Frustum PointNet achieved high benchmark performance compared to other fusion methods, but its multi-stage design makes end-to-end learning impractical. Very recently SECOND [30] offered a series of improvements to VoxelNet resulting in stronger performance and a much improved speed of 20 Hz. However, they were unable to remove the expensive 3D convolutional layers.

1.2. Contributions

- We propose a novel point cloud encoder and network, PointPillars, that operates on the point cloud to enable end-to-end training of a 3D object detection network.
- We show how all computations on pillars can be posed as dense 2D convolutions which enables inference at 62 Hz; a factor of 2-4 times faster than other methods.
- We conduct experiments on the KITTI dataset and demonstrate state of the art results on cars, pedestrians, and cyclists on both BEV and 3D benchmarks.
- We conduct several ablation studies to examine the key factors that enable a strong detection performance.

2. PointPillars Network

PointPillars accepts point clouds as input and estimates oriented 3D boxes for cars, pedestrians and cyclists. It consists of three main stages (Figure 2): (1) A feature encoder network that converts a point cloud to a sparse pseudo-image; (2) a 2D convolutional backbone to process the pseudo-image into high-level representation; and (3) a detection head that detects and regresses 3D boxes.

2.1. Pointcloud to Pseudo-Image

To apply a 2D convolutional architecture, we first convert the point cloud to a pseudo-image.

We denote by l a point in a point cloud with coordinates x, y, and z. As a first step, the point cloud is discretized into an evenly spaced grid in the x-y plane, creating a set of pillars $\mathcal{P}$ with $|\mathcal{P}| = B$. Note that a pillar is a voxel with unlimited spatial extent in the z direction and hence there is no need for a hyper parameter to control the binning in the z dimension. The points in each pillar are then decorated (augmented) with r, x_c, y_c, z_c, x_p, y_p where r is reflectance, the c subscript denotes distance to the arithmetic mean of all points in the pillar, and the p subscript denotes the offset from the pillar x, y center (see Sec 7.3 for design details). The decorated lidar point $\hat{l}$ is now $D = 9$ dimensional. While we focus on lidar point clouds, other point clouds such as radar or RGB-D[27] could be used with PointPillars by changing the decorations for each point.

The set of pillars will be mostly empty due to sparsity of the point cloud, and the non-empty pillars will in general have few points in them. For example, at $0.16^2\ m^2$ bins the point cloud from an HDL-64E Velodyne lidar has 6k-9k non-empty pillars in the range typically used in KITTI for $\sim 97\%$ sparsity. This sparsity is exploited by imposing a limit both on the number of non-empty pillars per sample (P) and on the number of points per pillar (N) to create a dense tensor of size (D, P, N). If a sample or pillar holds too much data to fit in this tensor, the data is randomly sampled. Conversely, if a sample or pillar has too little data to populate the tensor, zero padding is applied.

Next, we use a simplified version of PointNet where, for each point, a linear layer is applied followed by Batch-Norm [10] and ReLU [19] to generate a (C, P, N) sized

Figure 3. Qualitative analysis on KITTI. We show a bird's eye view of the lidar point cloud (top), as well as the 3D bounding boxes projected into the image for clearer visualization. Note that our method *only* uses lidar. We show ground truth (gray) and predicted boxes for car (orange), cyclist (red) and pedestrian (blue). The box orientation is shown by a line from the bottom center to the front of the box.

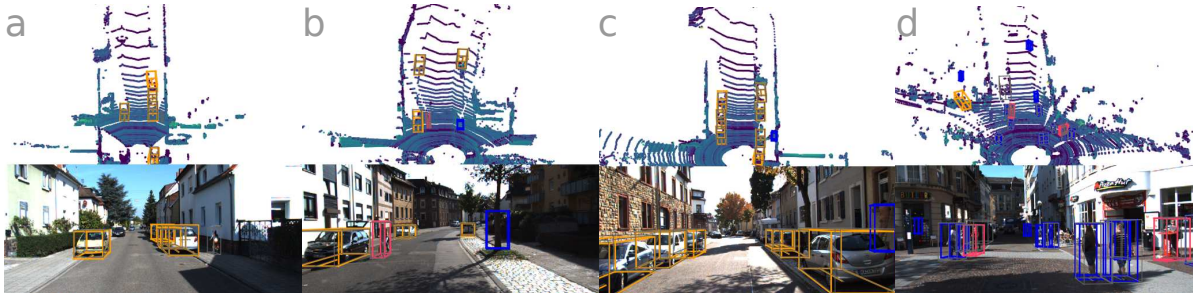

Figure 4. Failure cases on KITTI. Same visualize setup from Figure 3 but focusing on several common failure modes.

tensor. This is followed by a max operation over the channels to create an output tensor of size (C, P). Note that the linear layer can be formulated as a 1x1 convolution across the tensor resulting in very efficient computations.

Once encoded, the features are scattered back to the original pillar locations to create a pseudo-image of size (C, H, W) where H and W indicate the height and width of the canvas. Note that our choice of using pillars instead of voxels allows us to skip the expensive 3D convolutions in [33]'s Convolutional Middle Layer.

2.2. Backbone

We use a similar backbone as [33] and the structure is shown in Figure 2. The backbone has two sub-networks: one top-down network that produces features at increasingly small spatial resolution and a second network that performs upsampling and concatenation of the top-down features. The top-down backbone can be characterized by a series of blocks Block(S, L, F). Each block operates at stride S (measured relative to the original input pseudo-image). A block has L 3x3 2D conv-layers with F output channels, each followed by BatchNorm and a ReLU. The first convolution inside the layer has stride $\frac{S}{S_{in}}$ to ensure the block operates on stride S after receiving an input blob of stride S_{in}. All subsequent convolutions in a block have stride 1.

The final features from each top-down block are combined through upsampling and concatenation as follows. First, the features are upsampled, Up(S_{in}, S_{out}, F) from an initial stride S_{in} to a final stride S_{out} (both again measured wrt. the original pseudo-image) using a transposed 2D convolution with F final features. Next, BatchNorm and ReLU are applied to the upsampled features. The final output features are a concatenation of all features from different strides.

2.3. Detection Head

We use the Single Shot Detector (SSD) [18] setup to perform 3D object detection. If one is interested in a different task (e.g. segmentation), it would only require swapping out the detection head for a head specialized for the desired task. Similar to SSD, we match the priorboxes to the ground truth using 2D Intersection over Union (IoU) [4]. Bounding box height and elevation were not used for matching; instead given a 2D match, the height and elevation become additional regression targets.

3. Implementation Details

3.1. Network

Instead of pre-training our networks, all weights were initialized randomly using a uniform distribution as in [8].

The encoder network has $C = 64$ output features. The car and pedestrian/cyclist backbones are the same except for the stride of the first block ($S = 2$ for car, $S = 1$ for

pedestrian/cyclist). Both network consists of three blocks, Block1(S, 4, C), Block2($2S$, 6, 2C), and Block3($4S$, 6, 4C). Each block is upsampled by the following upsampling steps: Up1(S, S, 2C), Up2($2S$, S, 2C) and Up3($4S$, S, 2C). Then the features of Up1, Up2 and Up3 are concatenated together to create 6C features for the detection head.

3.2. Loss

We use the same loss functions introduced in SECOND [30]. Ground truth boxes and anchors are defined by $(x, y, z, w, l, h, \theta)$. The localization regression residuals between ground truth and anchors are defined by:

$$\Delta x = \frac{x^{gt} - x^a}{d^a}, \Delta y = \frac{y^{gt} - y^a}{d^a}, \Delta z = \frac{z^{gt} - z^a}{h^a}$$
$$\Delta w = \log \frac{w^{gt}}{w^a}, \Delta l = \log \frac{l^{gt}}{l^a}, \Delta h = \log \frac{h^{gt}}{h^a}$$
$$\Delta \theta = \sin\left(\theta^{gt} - \theta^a\right),$$

where x^{gt} and x^a are respectively the ground truth and anchor boxes and $d^a = \sqrt{(w^a)^2 + (l^a)^2}$. The total localization loss is:

$$\mathcal{L}_{loc} = \sum_{b \in (x,y,z,w,l,h,\theta)} \text{SmoothL1}\left(\Delta b\right)$$

Since the angle localization loss cannot distinguish flipped boxes, the heading is learned with a softmax classification loss, $\mathcal{L}_{dir}$, on the discretized directions [30].

The object classification loss uses focal loss [16]:

$$\mathcal{L}_{cls} = -\alpha_a \left(1 - p^a\right)^\gamma \log p^a,$$

where p^a is the class probability of an anchor. We use the original paper settings of $\alpha = 0.25$ and $\gamma = 2$. The total loss is therefore:

$$\mathcal{L} = \frac{1}{N_{pos}} \left(\beta_{loc}\mathcal{L}_{loc} + \beta_{cls}\mathcal{L}_{cls} + \beta_{dir}\mathcal{L}_{dir}\right),$$

where N_{pos} is the number of positive anchors and $\beta_{loc} = 2$, $\beta_{cls} = 1$, and $\beta_{dir} = 0.2$.

The loss function is optimized using Adam with an initial learning rate of $2 * 10^{-4}$ which decays by a factor of 0.8 every 15 epochs. The number of epochs is 160 and 320 with a batch size of 2 and 4 for val and test respectively.

4. Experimental setup

4.1. Dataset

All experiments use the KITTI object detection benchmark dataset [5], which consists of samples that have both lidar point clouds and images. We only train on lidar point clouds, but compare with fusion methods that use both lidar and images. The samples are originally divided into 7481 training and 7518 testing samples. For experimental studies we split the official training set into 3712 training samples and 3769 validation samples [1], while for our test submission we created a mini-val set of 784 samples from the validation set and trained on the remaining 6733 samples. The KITTI benchmark requires detections of cars, pedestrians, and cyclists. Since the ground truth objects were only annotated if they are visible in the image, we follow the standard convention [2, 33] of only using lidar points that project into the image. Following the standard literature practice on KITTI [11, 33, 30], we train one network for cars and one network for both pedestrians and cyclists.

4.2. Settings

Unless explicitly varied in an experimental study, we use an xy resolution: 0.16 m, max number of pillars (P): 12000, and max number of points per pillar (N): 100.

We use the same anchors and matching strategy as [33]. Each class anchor is described by a width, length, height, and z center, and is applied at two orientations: 0 and 90 degrees. Anchors are matched to ground truth using the 2D IoU with the following rules. A positive match is either the highest with a ground truth box, or above the positive match threshold, while a negative match is below the negative threshold. All other anchors are ignored in the loss.

At inference time we apply axis aligned non maximum suppression (NMS) with an overlap threshold of 0.5 IoU. This provides similar performance compared to rotational NMS, but is much faster.

Car. The x, y, z range is [(0, 70.4), (-40, 40), (-3, 1)] meters respectively. The car anchor has width, length, and height of (1.6, 3.9, 1.5) m with a z center of -1 m. Matching uses positive and negative thresholds of 0.6 and 0.45.

Pedestrian & Cyclist. The x, y, z range is [(0, 48), (-20, 20), (-2.5, 0.5)] meters respectively. The pedestrian anchor has width, length, and height of (0.6, 0.8, 1.73) meters with a z center of -0.6 meters, while the cyclist anchor has width, length, and height of (0.6, 1.76, 1.73) meters with a z center of -0.6 meters. Matching uses positive and negative thresholds of 0.5 and 0.35.

4.3. Data Augmentation

Data augmentation is critical for good performance on the KITTI benchmark [30, 32, 2].

First, following SECOND [30], we create a lookup table of the ground truth 3D boxes for all classes and the associated point clouds that falls inside these 3D boxes. Then for each sample, we randomly select $15, 0, 8$ ground truth samples for cars, pedestrians, and cyclists respectively and place them into the current point cloud. We found these settings to perform better than the proposed settings [30].

Next, all ground truth boxes are individually augmented. Each box is rotated (uniformly drawn from $[-\pi/20, \pi/20]$)

Method	Modality	Speed (Hz)	mAP	Car			Pedestrian			Cyclist		
			Mod.	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
MV3D [2]	Lidar & Img.	2.8	N/A	86.02	76.90	68.49	N/A	N/A	N/A	N/A	N/A	N/A
Cont-Fuse [15]	Lidar & Img.	16.7	N/A	88.81	85.83	77.33	N/A	N/A	N/A	N/A	N/A	N/A
Roarnet [25]	Lidar & Img.	10	N/A	88.20	79.41	70.02	N/A	N/A	N/A	N/A	N/A	N/A
AVOD-FPN [11]	Lidar & Img.	10	64.11	88.53	83.79	77.90	**58.75**	**51.05**	**47.54**	68.09	57.48	50.77
F-PointNet [21]	Lidar & Img.	5.9	65.39	88.70	84.00	75.33	58.09	50.22	47.20	75.38	61.96	54.68
HDNET [31]	Lidar & Map	20	N/A	89.14	**86.57**	78.32	N/A	N/A	N/A	N/A	N/A	N/A
PIXOR++ [31]	Lidar	35	N/A	**89.38**	83.70	77.97	N/A	N/A	N/A	N/A	N/A	N/A
VoxelNet [33]	Lidar	4.4	58.25	89.35	79.26	77.39	46.13	40.74	38.11	66.70	54.76	50.55
SECOND [30]	Lidar	20	60.56	88.07	79.37	77.95	55.10	46.27	44.76	73.67	56.04	48.78
PointPillars	Lidar	**62**	**66.19**	88.35	86.10	**79.83**	58.66	50.23	47.19	**79.14**	**62.25**	**56.00**

Table 1. Results on the KITTI test BEV detection benchmark.

Method	Modality	Speed (Hz)	mAP	Car			Pedestrian			Cyclist		
			Mod.	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
MV3D [2]	Lidar & Img.	2.8	N/A	71.09	62.35	55.12	N/A	N/A	N/A	N/A	N/A	N/A
Cont-Fuse [15]	Lidar & Img.	16.7	N/A	82.54	66.22	64.04	N/A	N/A	N/A	N/A	N/A	N/A
Roarnet [25]	Lidar & Img.	10	N/A	**83.71**	73.04	59.16	N/A	N/A	N/A	N/A	N/A	N/A
AVOD-FPN [11]	Lidar & Img.	10	55.62	81.94	71.88	66.38	50.80	42.81	40.88	64.00	52.18	46.61
F-PointNet [21]	Lidar & Img.	5.9	57.35	81.20	70.39	62.19	51.21	**44.89**	40.23	71.96	56.77	50.39
VoxelNet [33]	Lidar	4.4	49.05	77.47	65.11	57.73	39.48	33.69	31.5	61.22	48.36	44.37
SECOND [30]	Lidar	20	56.69	83.13	73.66	66.20	51.07	42.56	37.29	70.51	53.85	46.90
PointPillars	Lidar	**62**	**59.20**	79.05	**74.99**	**68.30**	**52.08**	43.53	**41.49**	**75.78**	**59.07**	**52.92**

Table 2. Results on the KITTI test 3D detection benchmark.

and translated (x, y, and z independently drawn from $\mathcal{N}(0, 0.25)$) to further enrich the training set.

Finally, we perform two sets of global augmentations that are jointly applied to the point cloud and all boxes. First, we apply random mirroring flip along the x axis [32], then a global rotation and scaling [33, 30]. Finally, we apply a global translation with x, y, z drawn from $\mathcal{N}(0, 0.2)$ to simulate localization noise.

5. Results

Quantitative Analysis. All detection results are measured using the official KITTI evaluation detection metrics which are: bird's eye view (BEV), 3D, 2D, and average orientation similarity (AOS). The 2D detection is done in the image plane and average orientation similarity assesses the average orientation (measured in BEV) similarity for 2D detections. The KITTI dataset is stratified into easy, moderate, and hard difficulties, and the official KITTI leaderboard is ranked by performance on moderate.

As shown in Table 1 and Table 2, PointPillars outperforms all published methods with respect to mean average precision (mAP)[2]. Compared to lidar-only methods, PointPillars achieves better results across all classes and difficulty strata except for the easy car stratum. It also outperforms fusion based methods on cars and cyclists.

While PointPillars predicts 3D oriented boxes, the BEV and 3D metrics do not take orientation into account. Orientation is evaluated using AOS [5], which requires projecting the 3D box into the image, performing 2D detection matching, and then assessing the orientation of these matches. The performance of PointPillars on AOS significantly exceeds in all strata as compared to the only two 3D detection methods [11, 30] that predict oriented boxes (Table 3). In general, image only methods perform best on 2D detection since the 3D projection of boxes into the image can result in loose boxes depending on the 3D pose. Despite this, PointPillars moderate cyclist AOS of 68.16 outperforms the best image based method [29].

Qualitative Analysis. We provide qualitative results in Figure 3 and 4. While we only train on lidar point clouds, for ease of interpretation we visualize the 3D bounding box predictions from the BEV and image perspective. Figure 3 shows our detection results, with tight oriented 3D bounding boxes. The predictions for cars are particularly accurate and common failure modes include false negatives on difficult samples (partially occluded or faraway objects) or false positives on similar classes (vans or trams). Detecting pedestrians and cyclists is more challenging and leads to some interesting failure modes. Pedestrians and cyclists are commonly misclassified as each other (see Figure 4a for a standard example and Figure 4d for the combination of pedestrian and table classified as a cyclist). Additionally, pedestrians are easily confused with narrow vertical features of the environment such as poles or tree trunks (see

[2]Val results were BEV AP of (87.7, 67.9, 66.8) and 3D AP of (77.4, 61.8, 64.9) on the moderate strata for cars, pedestrians, and cyclists.

Method	Modality	Speed (Hz)	mAOS	Car			Pedestrian			Cyclist		
			Mod.	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
SubCNN [29]	Img.	0.5	**72.71**	**90.61**	88.43	78.63	**78.33**	**66.28**	**61.37**	71.39	63.41	56.34
AVOD-FPN [11]	Lidar & Img.	10	63.19	89.95	87.13	79.74	53.36	44.92	43.77	67.61	57.53	54.16
SECOND [30]	Lidar	20	54.53	87.84	81.31	71.95	51.56	43.51	38.78	80.97	57.20	55.14
PointPillars	Lidar	**62**	68.86	90.19	**88.76**	**86.38**	58.05	49.66	47.88	**82.43**	**68.16**	**61.96**

Table 3. Results on the KITTI test average orientation similarity (AOS) detection benchmark. SubCNN is the best performing image only method, while AVOD-FPN, SECOND, and PointPillars are the only 3D object detectors that predict orientation.

Figure 4b). In some cases we correctly detect objects that are missing in the ground truth annotations (see Figure 4c).

6. Realtime Inference

As indicated by our results (Table 1, Figure 1, and Figure 5), PointPillars represent a significant improvement in terms of inference runtime. In this section, we break down our runtime and consider the different design choices that enabled this speedup. We focus on the car network, but the pedestrian and bicycle network runs at a similar speed since the smaller range cancels the effect of the backbone operating at lower strides. All runtimes are measured on a desktop with an Intel i7 CPU and a 1080ti GPU.

The main inference steps are as follows. First, the point cloud is loaded and filtered based on range and visibility in the images ($1.4\ ms$). Then, the points are organized in pillars and decorated ($2.7\ ms$). Next, the PointPillar tensor is uploaded to the GPU ($2.9\ ms$), encoded ($1.3\ ms$), scattered to the pseudo-image ($0.1\ ms$), and processed by the backbone and detection heads ($7.7\ ms$). Finally NMS is applied on the CPU ($0.1\ ms$) for a total runtime of $16.2\ ms$.

Encoding. The key design to enable this runtime is the PointPilar encoding. For example, at $1.3\ ms$ it is 2 orders of magnitude faster than the VoxelNet encoder ($190\ ms$) [33]. Recently, SECOND proposed a faster sparse version of the VoxelNet encoder for a total network runtime of $50\ ms$. They did not provide a runtime analysis, but since the rest of their architecture is similar to ours, it suggests that the encoder is still significantly slower; in their open source implementation[3] the encoder requires $48\ ms$.

Slimmer Design. We found that using fewer parameters did not affect detection performance. We reduced PyTorch runtime by $2.5\ ms$ by using a single PointNet in our encoder, instead of 2 sequential PointNets as in [33]. The first block dimension was lowered to 64 to match the encoder output size, which reduced the runtime by $4.5\ ms$. Finally, we saved another $3.9\ ms$ by cutting the output dimensions of the upsampled feature layers by half to 128.

TensorRT. While all our experiments were performed in PyTorch [20], the final GPU kernels for encoding, backbone and detection head were built using NVIDIA TensorRT, which is a library for optimized GPU inference. Switching to TensorRT gave a 45.5% speedup from the PyTorch pipeline which runs at $42.4\ Hz$.

The Need for Speed. As seen in Figure 5, PointPillars can achieve $105\ Hz$ with limited loss of accuracy. While it could be argued that such runtime is excessive since a lidar typically operates at $20\ Hz$, there are two key things to keep in mind. First, due to an artifact of KITTI ground truth annotations, only the $\sim 10\%$ of lidar points which project into the front image are utilized. However, an operational AV needs to view the full environment and process the complete point cloud, significantly increasing runtime. Second, timing measurements in the literature are typically done on a high-power desktop GPU. However, an operational AV may instead use embedded GPUs or embedded compute which will likely have lower throughput.

7. Ablation Studies

7.1. Spatial Resolution

Varying the size of the spatial binning provides a trade-off between speed and accuracy. Smaller pillars allow finer localization and lead to more features, while larger pillars are faster due to fewer non-empty pillars (speeding up the encoder) and a smaller pseudo-image (speeding up the CNN backbone). Figure 5 shows that the larger bin sizes lead to faster networks; at 0.28^2 we achieve $105\ Hz$ at similar performance to previous methods. The decrease in performance was mainly due to the pedestrian and cyclist classes, while car performance was stable across the bin sizes.

7.2. Per Box Data Augmentation

Both VoxelNet [33] and SECOND [30] recommend extensive per box augmentation. However, in our experiments, minimal box augmentation worked better. In particular, the detection performance for pedestrians degraded significantly with more box augmentation. Our hypothesis is that the introduction of ground truth sampling mitigates the need for extensive per box augmentation.

7.3. Point Decorations

The encoder takes the raw lidar returns: x, y, z, and reflectance, r, and adds deltas from pillar point cluster center

[3] https://github.com/traveller59/second.pytorch

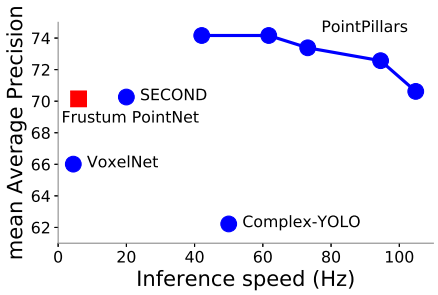

Figure 5. BEV detection performance (mAP) vs speed (Hz) on the KITTI [5] val set across pedestrians, bicycles and cars. Blue circles indicate lidar only methods, red squares indicate methods that use lidar & vision. Different operating points were achieved by using pillar grid sizes in $\{0.12^2, 0.16^2, 0.2^2, 0.24^2, 0.28^2\}\ m^2$ and max-pillars of $16000, 12000, 12000, 8000, 8000$ respectively.

$(\Delta x_c, \Delta y_c, \Delta z_c)$ (as was done in VoxelNet [33]) and the distance from the pillar center $(\Delta x_p, \Delta y_p)$ (our contribution). The pillar offsets $(\Delta x_p, \Delta y_p)$ encode the point location in the local coordinate system of each pillar. They are independent of the other points and thus standardize the local context of the points in a manner that is complementary to the 2D convolutions in x and y. We did not include the z pillar offset since this is a constant offset for all the points. While the cluster offsets $(\Delta x_c, \Delta y_c, \Delta z_c)$ provide another way to standardize the local context of the points, it requires calculating a summary statistic and hence creates a dependency between the points. Data augmentation and the subsampling of points in a pillar changes the cluster center, which leads to the higher variance when training only with the cluster offsets and not the pillar offsets. The strength of our decoration choice is shown in Table 4.

7.4. Encoding

To assess the impact of the proposed PointPillar encoding in isolation, we implemented several encoders in the official codebase of SECOND [30]. For details on each encoding, we refer to the original papers.

As shown in Table 5, learning the feature encoding is strictly superior to fixed encoders across all resolutions. This is expected as most successful deep learning architectures are trained end-to-end. Further, the differences increase with larger bin sizes where the lack of expressive power of the fixed encoders are accentuated due to a larger point cloud in each pillar. Among the learned encoders VoxelNet is stronger than PointPillars. However, when the comparison is made for a similar *inference time*, it is clear that PointPillars offers a better operating point (Figure 5).

There are a few curious aspects of Table 5. First, despite notes in the original papers that their encoder only works on cars, we found that the MV3D [2] and PIXOR [32] encoders can learn pedestrians and cyclists quite well. Second, our implementations beat the respective published results by a large margin ($1-10$ mAP). While this is not a direct comparison since we only used the respective *encoders* and not the full network architectures, the performance difference is noteworthy. We see several potential reasons. For VoxelNet and SECOND we suspect the boost in performance comes from improved data augmentation hyperparameters as discussed in Section 7.2. Among the fixed encoders, roughly half the performance increase can be explained by the introduction of ground truth database sampling [30], which we found to boost the mAP by around 3% mAP. The remaining differences are likely due to a combination of multiple hyperparameters including network design (number of layers, type of layers, whether to use a feature pyramid); anchor box design (or lack thereof [32]); localization loss with respect to 3D and angle; classification loss; optimizer choices (SGD vs Adam, batch size); and more. However, a more careful study is needed to isolate each cause and effect.

$\mathbf{x, y, z}$	$\mathbf{r}$	$\mathbf{x_c, y_c, z_c}$	$\mathbf{x_p, y_p}$	**BEV mAP**	**Δ mAP**
✓				66.6	-6.0
✓	✓			70.5	-2.1
✓	✓	✓		70.4	-2.2
✓	✓		✓	71.4	-1.2
✓	✓	✓	✓	**72.6**	0.0

Table 4. Ablation study for encoder point decorations. The lidar sensor outputs the spatial location, x, y, z, and reflectance r, of each lidar return. This can be supplemented with the cluster center offset $(\Delta x_c, \Delta y_c, \Delta z_c)$ or pillar center offset $(\Delta x_p, \Delta y_p)$. The best detection performance uses all this information.

Encoder	**Type**	$\mathbf{0.16^2}$	$\mathbf{0.20^2}$	$\mathbf{0.24^2}$	$\mathbf{0.28^2}$
MV3D [2]	Fixed	72.8	71.0	70.8	67.6
C. Yolo [26]	Fixed	72.0	72.0	70.6	66.9
PIXOR [32]	Fixed	72.9	71.3	69.9	65.6
VoxelNet [33]	Learn	**74.4**	**74.0**	**72.9**	71.9
PointPillars	Learn	73.7	72.6	**72.9**	**72.0**

Table 5. Encoder performance evaluation. To fairly compare encoders, the same network architecture and training procedure was used and only the encoder and xy resolution were changed between experiments. Performance is measured as BEV mAP on KITTI val. Learned encoders clearly beat fixed encoders, especially at larger resolutions.

8. Conclusion

In this paper, we introduce PointPillars, a novel deep network and encoder that can be trained end-to-end on lidar point clouds. We show that on the KITTI challenge, PointPillars dominates all existing methods by offering higher detection performance (BEV and 3D mAP) at a faster speed. Our results suggests that PointPillars offers the best architecture so far for 3D object detection from lidar.

References

[1] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun. 3d object proposals for accurate object class detection. In *NIPS*, 2015.

[2] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, 2017.

[3] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *ICRA*, 2017.

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 2010.

[5] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *CVPR*, 2012.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[7] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *ICCV*, 2017.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[9] M. Himmelsbach, A. Mueller, T. Lüttel, and H.-J. Wünsche. Lidar-based 3d object perception. In *Proceedings of 1st international workshop on cognition for technical systems*, 2008.

[10] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[11] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander. Joint 3d proposal generation and object detection from view aggregation. In *IROS*, 2018.

[12] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, et al. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 2008.

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

[14] B. Li, T. Zhang, and T. Xia. Vehicle detection from 3d lidar using fully convolutional network. In *RSS*, 2016.

[15] M. Liang, B. Yang, S. Wang, and R. Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *ECCV*, 2018.

[16] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *PAMI*, 2018.

[17] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. In *ECCV*, 2016.

[19] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[21] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. In *CVPR*, 2018.

[22] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017.

[23] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017.

[24] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[25] K. Shin, Y. Kwon, and M. Tomizuka. Roarnet: A robust 3d object detection based on region approximation refinement. *arXiv:1811.03818*, 2018.

[26] M. Simon, S. Milz, K. Amende, and H.-M. Gross. Complex-YOLO: Real-time 3d object detection on point clouds. *arXiv:1803.06199*, 2018.

[27] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015.

[28] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *CVPR*, 2019.

[29] Y. Xiang, W. Choi, Y. Lin, and S. Savarese. Subcategory-aware convolutional neural networks for object proposals and detection. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

[30] Y. Yan, Y. Mao, and B. Li. SECOND: Sparsely embedded convolutional detection. *Sensors*, 18(10), 2018.

[31] B. Yang, M. Liang, and R. Urtasun. HDNET: Exploiting HD maps for 3d object detection. In *CoRL*, 2018.

[32] B. Yang, W. Luo, and R. Urtasun. PIXOR: Real-time 3d object detection from point clouds. In *CVPR*, 2018.

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