To the Point: Efficient 3D Object Detection in the Range Image with Graph Convolution Kernels

Yuning Chai[1]   Pei Sun[1]   Jiquan Ngiam[2]   Weiyue Wang[1]
Benjamin Caine[2]   Vijay Vasudevan[2]   Xiao Zhang[1]   Dragomir Anguelov[1]

[1]Waymo LLC, [2]Google Brain
{chaiy}@waymo.com

Abstract

3D object detection is vital for many robotics applications. For tasks where a 2D perspective range image exists, we propose to learn a 3D representation directly from this range image view. To this end, we designed a 2D convolutional network architecture that carries the 3D spherical coordinates of each pixel throughout the network. Its layers can consume any arbitrary convolution kernel in place of the default inner product kernel and exploit the underlying local geometry around each pixel. We outline four such kernels: a dense kernel according to the bag-of-words paradigm, and three graph kernels inspired by recent graph neural network advances: the Transformer, the PointNet, and the Edge Convolution. We also explore cross-modality fusion with the camera image, facilitated by operating in the perspective range image view. Our method performs competitively on the Waymo Open Dataset and improves the state-of-the-art AP for pedestrian detection from 69.7% to 75.5%. It is also efficient in that our smallest model, which still outperforms the popular PointPillars in quality, requires 180 times fewer FLOPS and model parameters.

1. Introduction

Deep-learning-based point cloud understanding has increased in popularity in recent years. Numerous architectures [9, 11, 14, 19, 17, 22, 21, 28, 30, 33] have been proposed to handle the sparse nature of point clouds, with successful applications ranging from 3D object recognition [4, 25, 29], to indoor scene understanding [6, 23] and autonomous driving [2, 8, 24].

Point clouds may have different properties based on the way they are acquired. For example, point clouds for 3D object recognition are often generated by taking one or many depth images from multiple views around a single object. In other applications such as robotics and autonomous driving,

a device such as a LiDAR continuously scans its surroundings in a rotating pattern, producing a 2D scan pattern called the *range image*. Each pixel in this image contains a range value and other features, such as each laser return's intensity.

The operating range of these sensors has significantly improved over the past few years. As a result, state-of-the-art methods [11, 21, 30, 33] that require projecting points into a dense 3D grid have become less efficient as their complexity scales quadratically with the range. In this work, we propose a new point cloud representation that directly operates on the perspective 2D range image without ever projecting the pixels to the 3D world coordinates. Therefore, it does not suffer from the efficiency scaling problem as mentioned earlier. We coin this new representation *perspective point cloud*, or PPC for short. We are not the first to attempt to do so. [12, 14] have proposed a similar idea by applying a convolutional neural network to the range image. However, they showed that these models, despite being more efficient, are not as powerful as their 3D counterparts, i.e. 3D grid methods [9, 11, 21, 30, 33] and 3D graph methods [19, 22]. We believe that this quality difference traces its root to the traditional 2D convolution layers that cannot easily exploit the range image's underlying 3D structure. To counter this deficiency, we propose four alternative kernels (Fig. 1: c, d) that can replace the scalar product kernel at the heart of the 2D convolution. These kernels inject much needed 3D information to the perspective model, and are inspired by recent advances in graph operations, including transformers [26], PointNet [18] and Edge Convolutions [28].

We summarize the contributions of this paper as follows: 1) We propose a perspective range-image-based 3D model which allows the core of the 2D convolution operation to harness the underlying 3D structure; 2) We validate our model on the 3D detection problem and show that the resulting model sets a new state-of-the-art for pedestrians on the Waymo Open Dataset, while also matching the SOTA on vehicles; 3) We provide a detailed complexity/model-size-
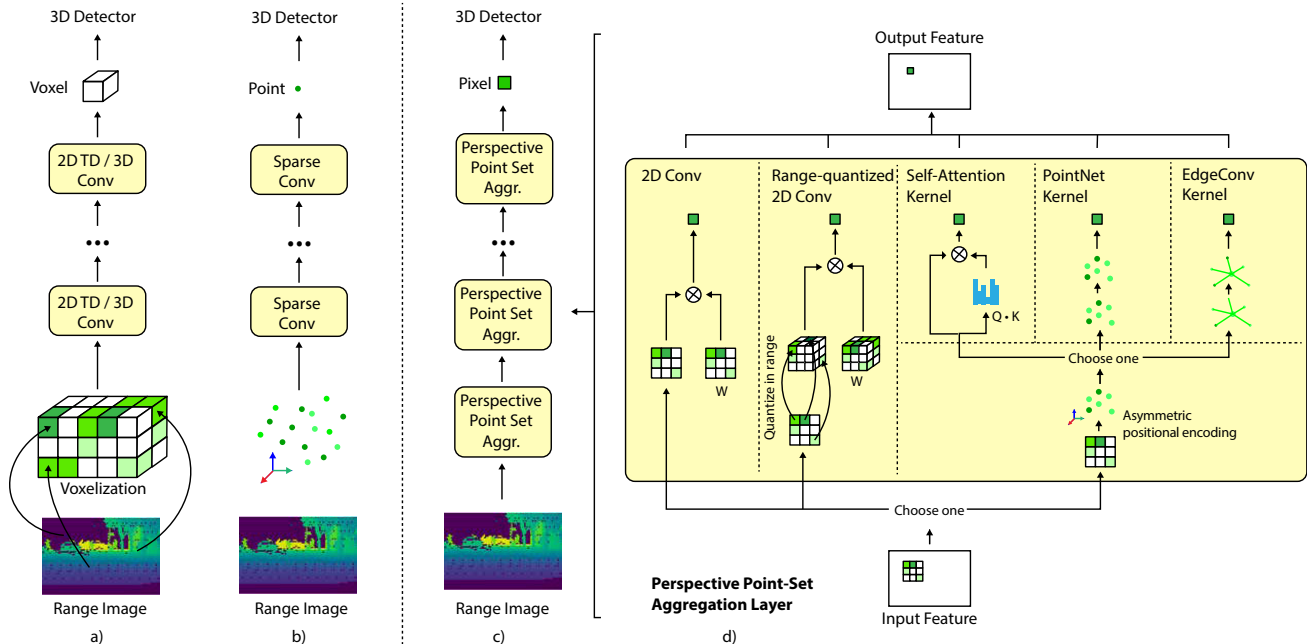
Figure 1: Overview of existing 3D detectors and our proposed perspective point cloud representation. **a)** 3D grid-based methods [9, 11, 21, 33] first voxelizes the 3D space, feeds the 3D dense structure to a 3D convolution network or a 2D top-down network, and make the final prediction based on 3D voxels. **b)** 3D graph models [19, 22] builds a graph neural network on top of the sparse point cloud and makes predictions based on points. **c)** Our method, PPC, operates directly on the perspective range image view and predicts from pixels. **d)** It utilizes a set of specialized 2D convolution layers in the perspective 2D view. We propose four improved kernels in addition to the traditional inner product kernel (2D conv).

vs.-accuracy analysis, and show that we can maintain the efficiency benefits from operating on the 2D range image. Our smallest model with only 24k parameters has higher accuracy than the popular PointPillars [11] model with over 4M parameters.

2. Related Work

We focus on 3D object detection tasks where a perspective range image view is available, such as a LiDAR scan for autonomous driving. We group most of existing works in this field into 3 categories (see Fig. 1 a,b,c):

3D Grid. The key component for these methods is the voxelization stage, where the projected sparse point cloud in 3D is voxelized into a 3D dense *grid* structure that is friendly to dense convolution operations in either 3D or 2D top-down. Popular works in this category include [11, 30, 33], all of which apply a PointNet-style [18] encoding for each voxel in the 3D grid. 3D grid methods have been performing the best in recent years and appear in some of the top entries on several academic and industrial leaderboards [2, 8, 24], thanks to its strong generalization and high efficiency due to the use of dense convolutions. There are three major drawbacks to 3D grid methods. 1) Needing a full dense 3D grid poses a limitation to handle long-range, since both the complexity

and the memory consumption scale quadratically with the range. 2) The voxel representation has a limited resolution due to the scalability issue mentioned above. Therefore, the detection of thin objects such as the pedestrian or signs can be inaccurate. 3) There is no special handling in the model for treating occluded areas than true empty areas.

3D Graph. This line of methods differs from the voxelized grid counterparts in that there is no voxelization stage after the 3D point cloud projection. Without voxelization, dense convolutions can no longer apply. Therefore, these methods resort to building a graph neural network (GNN) that preserves the points' spatial relationship. Popular methods include [16, 19, 28, 22]. Although these methods can scale better with range, they lag behind the quality of voxelized grid methods. Moreover, the method requires a nearest neighbor search step to create the input graph for the GNN. Finally, like in the 3D grid case, these methods also cannot model occlusion either.

Perspective 2D Grid. There has been minimal prior work that tries to solve the 3D point cloud representation problem with a 2D perspective range image alone. [12, 14] applied a traditional 2D convolution network to the range image. Operating in 2D is more efficient than in 3D because compute is not wasted on empty cells as in the 3D grid case, nor do

we need to perform a nearest-neighbor search for 3D points as in the 3D graph case. Additionally, occlusion is implicitly encoded in the range image, where for each pixel, the ray to its 3D position is indeed empty, and the area behind it is occluded. Unfortunately, perspective 2D grid methods often cannot match the quality of 3D methods. Our proposed method also belongs to this category, and the goal of this paper is to improve the perspective 2D models to match the accuracy of 3D methods.

Finally, a few wildcard methods do not categorize into any of the three groups above. F-PointNet [17] generates proposals via the camera image and validates the proposals using a point-level rather than scene-level PointNet encoding [18]. StarNet [15] shares a similar mechanism for proposal validation, but the proposals generation use farthest-point-sampling instead of relying on the camera.

3. Perspective Point Cloud Model

In this section, we look at the proposed perspective point cloud (PPC) model. The heart of the model is a set of perspective 2D layers that can exploit the underlying 3D structure of the range-image pixels (Sec. 3.1). Because the range image can have missing returns, we need to handle down- and up-sampling differently than in a traditional CNN (Sec. 3.2). Finally, we outline the backbone network, a cross-modality fusion mechanism with the camera, and the detector head in Sec. 3.3, Sec. 3.4 and Sec. 3.5.

3.1. Perspective Point-Set Aggregation Layers

As shown in Fig. 1, we propose a generalization of the 2D convolution network that operates on a 2D LiDAR range or RGB-D image. Each layer takes inputs in the form of a feature map $\mathbf{F}_i$ of shape [H, W, D], a per-pixel spherical polar coordinates map $\mathbf{X}_i$ of shape [H, W, 3], and a binary mask $\mathbf{M}_i$ of shape [H, W] that indicates the validity of each pixel, since returns may be missing. The three dimensions in the spherical polar coordinates $\{\theta, \phi, r\}$ describe the azimuth, the inclination, and the depth of each pixel from the sensor's view. The layer outputs a new feature map $\mathbf{F}_o$ of shape [H, W, D'].

Each pixel in the output feature map $F_o[m, n]$ is a function of the corresponding input feature and its neighborhood $F_i[m', n']$ where $m' \in [m - k_H/2, m + k_H/2]$ and $n' \in [n - k_W/2, n + k_W/2]$. k_H and k_W are neighborhood/kernel sizes along the height and width dimensions:

$$F_o[m, n] = f(\{\mathbf{F}_i, \mathbf{X}_i, \mathbf{M}_i\}[m', n'], \forall m', n') \quad (1)$$

where $f(.)$ is the **Point-Set Aggregation kernel** that reduces information from multiple pixels to a single one. A layer equivalent to the conventional 2D convolution can be constructed by applying the **2D convolution kernel** f_{2D}:

$$f_{2D} := \sum_{m', n'} \mathbf{W}[m' - m, n' - n] \cdot \mathbf{F}_i[m', n'] \quad (2)$$

where $\mathbf{W}$ are a set of trainable weights. Please note that we omit the depth dimension D and D' in the kernel definitions for writing simplicity.

f_{2D} does not depend on the 3D coordinates $\mathbf{X}_i$. Therefore, it cannot reason about the underlying geometric pattern of the neighborhood. Next, we will present four kernels that can leverage this geometric pattern.

Range-quantized (RQ) 2D convolution kernel. Inspired by the linearization idea in the bag-of-words approach, one of the simplest ways of adding the range information to the layer is to apply different sets of weights to the input feature depending on the relative depth difference of each neighboring pixel to the center pixel:

$$f_{2D+} := \sum_{m', n'} \mathbf{W}_r[m' - m, n' - n] \cdot \mathbf{F}_i[m', n'] \cdot \delta \quad (3)$$

$$\mathbf{W}_r = \sum_{k \in K} \mathbb{1}[\alpha_k \le \Delta r < \beta_k] \cdot \mathbf{W}_k$$

$$\Delta r = \mathbf{R}_i[m', n'] - \mathbf{R}_i[m, n]$$

where we define K sets of weights $\mathbf{W}_k$, each with a predefined scalar range $[\alpha_k, \beta_k]$. These ranges differ from layer to layer and are computed using histograms over many input samples. Different weights are applied depending on the range difference Δr. $\mathbf{R}_i$ denotes the range channel and is part of $\mathbf{X}_i$. $\mathbb{1}$ is the indicator function and has the value 1 if the expression is true and 0 otherwise. δ is an indicator function based on the validity of each the participating pixels, defined as:

$$\delta = \mathbf{M}_i[m', n'] \cdot \mathbf{M}_i[m, n] \quad (4)$$

δ also appears in subsequent kernels.

While f_{2D+} takes the range information into account, it is very inefficient in that the number of parameters increases by K-fold, which can be significant and cause overfitting. Moreover, the amount of computation also increases by K-fold.

Self-attention kernel. Given the sparse nature of the range image data in the 3D space, graph operation are a more natural choice than projecting to a higher-dimensional space. The transformer [26] is one of the most popular graph operators. It has found success in both NLP [7] and computer vision [3]. In its core, the transformer generates weights depending on the input features and spatial locations of the features, and therefore does not require a set of weights in a dense form. A transformer-inspired kernel looks like follows:

$$f_{SA} := \sum_{m', n'} \text{softmax}(\mathbf{F}_i[m, n]^T \cdot \mathbf{W}_q^T \cdot (\mathbf{W}_k \cdot \mathbf{F}_i[m', n'] + r))$$

$$\cdot \mathbf{W}_v \cdot \mathbf{F}_i[m', n'] \cdot \delta \quad (5)$$

$$r = \mathbf{W}_r \gamma(\mathbf{X}_i[m,n], \mathbf{X}_i[m',n'])$$

where $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$ and $\mathbf{W}_r$ are four sets of trainable weights. $\gamma(.,.)$ is an **asymmetric positional encoding** between two points. It is defined as:

$$\gamma(\mathbf{x}, \mathbf{x}') := \{ r' \cdot \cos(\Delta\theta) \cdot \cos(\Delta\phi) - r,$$
$$r' \cdot \cos(\Delta\theta) \cdot \sin(\Delta\phi),$$
$$r' \cdot \sin(\Delta\theta) \} \tag{6}$$

$$\Delta\theta = \theta' - \theta, \quad \Delta\phi = \phi' - \phi$$

where $\mathbf{x} = \{\theta, \phi, r\}$, $\mathbf{x}' = \{\theta', \phi', r'\}$ are the azimuth, inclination and depth of the points. $\gamma(.,.)$ is also used in subsequent kernels.

This asymmetric positional encoding has a geometric meaning. Namely, it is in an oblique Cartesian frame viewed from the sensor's location. For each pixel, after rotating the sphere by $-\theta$ and $-\phi$, $\mathbf{x}$ has the spherical polar coordinates $\{0, 0, r\}$, while $\mathbf{x}'$ is at $\{\Delta\theta, \Delta\phi, r'\}$. We project them to Cartesian, which yields $\{r, 0, 0\}$ for $\mathbf{x}$ and $\{\cos(\Delta\theta) \cdot \cos(\Delta\phi) \cdot r', \cos(\Delta\theta) \cdot \sin(\Delta\phi) \cdot r', \sin(\Delta\theta) \cdot r'\}$ for $\mathbf{x}'$. The encoding is then their element-wise difference. Note that the oblique Cartesian frame is different from pixel to pixel, but does not depend on the weights of each layer, and therefore can be pre-computed once for all layers per sample. This positional encoding is also used for the subsequent kernels.

PointNet kernel. While the transformer has seen great success in NLP and computer vision, PointNet [18] on the other hand has laid the groundwork to the majority of works for 3D point cloud understanding in the past years. It is widely used in robotics, thanks to VoxelNet [33], PointPillars [11] and PointRCNN [22]. The PointNet formulation is yet quite simple. It learns a multi-layer perceptron (MLP) that encodes the neighboring features and their relative coordinates to the center, and pools the encodings via max-pooling. Our PointNet-inspired kernel looks as follows:

$$f_{PN} := \max_{m',n'} \mathrm{MLP}($$
$$[\mathbf{F}_i[m',n'], \gamma(\mathbf{X}_i[m,n], \mathbf{X}_i[m',n'])], \Theta) \cdot \delta \tag{7}$$

where Θ are trainable weights for the MLP.

EdgeConv kernel. The edge convolution proposed by [28] is very similar to PointNet. In PointNet, the input to the MLP is the feature itself and a relative positional encoding. The edge convolution adds one more feature that is the center feature to the input:

$$f_{EC} := \max_{m',n'} \mathrm{MLP}($$
$$[\mathbf{F}_i[m',n'], \mathbf{F}_i[m,n], \gamma(\mathbf{X}_i[m,n], \mathbf{X}_i[m',n'])], \Theta) \cdot \delta \tag{8}$$

Although the last three, the Transformer, the PointNet, and EdgeConv kernels, are inspired by the 3D graph literature discussed in Sec. 2, they do not result in the inability to model occlusion and the inefficiency due to the need of the nearest neighbor search for each point. The perspective point-set aggregation layer can model occlusion just like any 2D range image-based method. Moreover, it does not require the nearest neighbor search, as it selects neighbors based on the distances in the 2D range image rather than in 3D. Finding neighbors in a dense 2D grid is trivial.

3.2. Smart Down-Sampling

Unlike RGB images, the LiDAR range or RGBD images can have a noticeable amount of invalid range pixels in the image. It can be due to light-absorbing or less reflective surfaces. In the case of LiDAR, quantization and calibration artifacts can even result in missing returns that form a regular pattern, where the down-sampling with a fixed stride can inadvertently further emphasize the missing returns. Therefore, we define a smart down-sampling strategy to avoid missing returns as we sample: When we down-sample with a stride of, for example, 2×2, we select 1 pixel from 4 neighboring pixels. But instead of always selecting the first or the last pixel, we select a *valid* pixel, if available, which is the closest to the centroid of all valid pixels among the four.

We define the down-sampling layer as follows (for brevity, we depict the math for the 1D space here):

$$\mathbf{F}_o[m] = \mathbf{F}_i[\hat{m}], \ \mathbf{X}_o[m] = \mathbf{X}_i[\hat{m}], \ \mathbf{M}_o[m] = \mathbf{M}_i[\hat{m}] \tag{9}$$

$$\hat{m} = \arg\min_{m' \in \mathcal{S}} \|\mathbf{R}_i[m'] - \mu\|^2, \ \mu = \frac{\sum_{m' \in \mathcal{S}} \mathbf{R}_i[m']}{\|\mathcal{S}\|_0}$$

where $\mathcal{S}$ contains valid $s' \in m \cdot \lambda, \ldots, (m+1) \cdot \lambda - 1$ according to the mask $\mathbf{M}_i$, and λ is the intended stride. $\mathbf{R}_i$ is the range part of the spherical polar coordinates $\mathbf{X}_i$.

During up-sampling, technically, we would need to generate new points $\mathbf{X}_o$ from the input 3D coordinates $\mathbf{X}_i$, which is difficult to do. Luckily, an up-sampling usually mirrors a previous down-sampling and never exceeds the original input resolution. Therefore, we can remember the coordinates and the mask from the input of a corresponding down-sampling layer and reuse them for after up-sampling. We use the zeros vector as features for the new pixels generated from up-sampling. The up-sampling layer is the reverse operation of the down-sampling layer:

$$\mathbf{X}_o = \mathbf{X}_{i'}, \ \mathbf{M}_o = \mathbf{M}_{i'}$$

$$\mathbf{F}_o[m] = \begin{cases} \mathbf{F}_i[\bar{m}], \text{ if } m = \hat{m} \text{ and } m \in \mathcal{S} \\ 0, \text{ else} \end{cases} \tag{10}$$

$$\hat{m} = \arg\min_{m' \in \mathcal{S}} \|\mathbf{R}_o[m'] - \mu\|^2, \ \mu = \frac{\sum_{m' \in \mathcal{S}} \mathbf{R}_o[m']}{\|\mathcal{S}\|_0}$$

where $\mathcal{S}$ contains all valid $s' \in \bar{m} \cdot \lambda, \ldots, (\bar{m}+1) \cdot \lambda - 1$ according to the mask $\mathbf{M}_{i'}$ where λ is the up-sampling stride. $\mathbf{X}_{i'}$ and $\mathbf{M}_{i'}$ are values taken from a previous layer i' whose down-sampling with stride S yields the input layer i. $\mathbf{R}_o$ is the range part of the spherical polar coordinates $\mathbf{X}_o$.

3.3. Backbone Architecture

Now that we have both the perspective point-set aggregation layers and the sampling layers defined, we look at the backbone architecture to chain the layers together into a network. We performed a low-effort manual architecture search using the 2D convolution kernel and kept using the best architecture for the remaining experiments. Our network builds on top of the building blocks proposed in [14]: the **feature extractor (FE)** that extracts features with an optional down-sampling, and the **feature aggregator (FA)** that merges most features to lower-level features to create skip connections. Our pedestrian network consists of 4 FE and 1 FA blocks, and predictions are made on half of the input resolution. Since the vehicles appear wider in the range image, we extend the network to 8 FE and 5 FA blocks. Please find an illustration of the architecture in Supp. Sec. A.

3.4. Point-Cloud-Camera Sensor Fusion

The perspective range image representation provides a natural way to fuse camera features to point cloud features since each location in the range image can be projected into the camera space. For each camera image, we first compute dense features using a modern convolution U-network [20] (please see Supp. Sec. B for details). We project it to a location in its corresponding camera image for each point in the range image. We then collect the feature vector computed at that pixel in the camera image and concatenate the feature vector to the range image features. A zeros feature vector is appended, should an area be not covered by any camera. This approach can apply to any layer since there is always a point associated with each location. We train our networks end-to-end, with the camera convolution networks randomly initialized.

3.5. CenterNet Detector

We validate our point cloud representation via 3D object detection. We extend the CenterNet [31] to 3D: For each pixel in the backbone network's output feature map, we predict both a classification distribution and a regression vector. The classification distribution contains $C+1$ targets, where C is the number of classes plus a background class.

During training, the classification target is controlled by a Gaussian ball around the center of each box (see Supp. Fig. 5): $s_{i,j} = \mathcal{N}(||\mathbf{x}_i - \mathbf{b}_j||_2, \sigma)$, where $\mathbf{x}$ are points and $\mathbf{b}$ are boxes. σ is the Gaussian standard deviation, set to 0.25 meters for pedestrians and 0.5 meters for vehicles. For 2D detection in images, where CenterNet was originally pro-posed, the box center is always a valid pixel in the 2D image. In 3D, however, points are sparse, and the closest point to the center might be far away. Therefore, we normalize the target score by the highest score within each box to ensure that there is at least one point with a score 1.0 per box. We then take the maximum over all boxes and get the final training target score per point: $\mathbf{y}_i^{cls} = \max_j s_{i,j} / \max_{i \in B_j} s_{i,j}$.

The regression target $\mathbf{y}_i^{reg}$ for 3D detection contains 8 targets for 7 degrees-of-freedom boxes: a three-dimensional relative displacement vector from the point's 3D location to the center of the predicted box; another three dimensions that contain the absolute length, width, and height; and a single angle split into its sine and cosine forms in order to avoid discontinuity around 2π.

We used the penalty-reduced focal loss for the classification, as proposed by [31], and the ℓ^1-loss for the regression. We then train with a batch size of 256 over 300 epochs with an Adam optimizer. The initial learning rate is set to 0.001, and it decays exponentially over the 300 epochs.

All of our experiments applies the CenterNet detector head to the final feature map of the backbone. We have observed no significant difference in quality between CenterNet and other single-shot detectors such as the SSD [13]. Two-stage methods, such as [22], usually outperform single-stage methods on vehicles by a significant margin. However, the impact on pedestrians is less prevalent.

4. Experiments

4.1. Waymo Open Dataset

We conducted experiments on the pedestrians and vehicles of the Waymo Open Dataset [24]. The dataset contains 1000 sequences, split into 780 training, 120 validation, and 100 test. Each sequence contains 200 frames, where each frame captures the full 360 degrees around the ego-vehicle that results in a range image of a dimension 64×2650 pixels. The LiDAR has a maximum range of around 100 meters.

Metrics. We use metrics defined by the Waymo Open Dataset. **AP**: Average precision at 0.5 IOU for vehicles and 0.7 IOU for pedestrians. **APH**: Same as AP, but also takes the heading into account when matching boxes. **3D** vs. **BEV**: Whether IOU is measured on rotated 3D boxes or projected top-down rotated 2D boxes. **L1** vs. **L2**: Level of difficulty, L2 include more difficult boxes.

Results for pedestrians and vehicles detections are highlighted in Tab. 1 and Tab. 2. Fig. 3 shows a few example results. Our method significantly outperforms all recent works on the pedestrian category, including the 3D grid or graph representations methods. This boost is not surprising for two reasons: a) pedestrians are tall so that the perspective view captures its full shape, b) they are also thin so that the voxels in the voxel-based methods end up too large and therefore cannot accurately make predictions. We perform very

Method	3D		BEV		3D APH$_{L2}$ by distance		
	AP$_{L1}$	**APH$_{L2}$**	AP$_{L1}$	APH$_{L2}$	<30m	30-50m	>50m
LaserNet CVPR'19 [14]*	62.9	45.4	69.7	50.4	62.6	39.2	17.4
PointPillars CVPR'19 [11]*	61.6	43.0	70.4	49.5	54.2	40.7	25.7
MultiView CORL'19 [32]	65.3	-	74.4	-	-	-	-
StarNet Arxiv'19 [15]	68.3	52.8	73.8	57.3	63.1	52.1	35.5
PPBA-StarNet ECCV'20 [5]	69.7	53.9	74.9	58.4	64.4	53.2	36.6
Pilar-based ECCV'20 [27]	72.5	-	78.5	-	-	-	-
PPC + Conv2D (Ours)	63.4	47.0	71.3	53.1	63.4	42.0	19.0
PPC + RQ-Conv2D (Ours)	68.4	54.3	76.9	61.6	67.1	52.6	32.2
PPC + Self-Attention (Ours)	57.9	43.7	65.3	49.5	60.3	39.8	17.1
PPC + PointNet (Ours)	72.4	57.9	79.3	63.9	70.3	56.3	35.5
PPC + EdgeConv (Ours)	73.9	59.6	80.6	65.6	71.5	58.4	38.1
PPC + EdgeConv + Camera (Ours)	**75.5**	**61.5**	**82.2**	**67.6**	**72.3**	**61.3**	**41.3**

Table 1: **Pedestrians** on the Waymo Open Dataset *validation* set. **3D APH$_{L2}$** is the primary metric for the dataset. Results denoted with * are based on our reimplementation. Others are taken from papers or via email communication with paper authors. Our method significantly improves over recent works.

Method	3D		BEV		3D APH$_{L2}$ by distance		
	AP$_{L1}$	**APH$_{L2}$**	AP$_{L1}$	APH$_{L2}$	<30m	30-50m	>50m
LaserNet CVPR'19 [14]*	56.1	48.4	73.1	63.9	75.1	45.6	21.7
PointPillars CVPR'19 [11]*	56.1	48.2	77.2	67.6	-	-	-
MultiView CORL'19 [32]	62.9	-	80.4	-	-	-	-
StarNet Arxiv'19 [15]	55.1	48.3	67.7	60.0	79.1	43.1	20.2
PV-RCNN CVPR'20 [21]†	**70.3**	**64.8**	**83.0**	**67.6**	**91.0**	**64.5**	**35.7**
PPBA-PointPillars ECCV'20 [5]	61.8	53.4	81.4	72.2	-	-	-
LSTM ECCV'20 [10]	63.4	-	-	-	-	-	-
Pillar-based ECCV'20 [27]†	67.7	-	86.1	-	-	-	-
RCN CORL'20 [1]	69.5	-	83.4	-	-	-	-
PPC + Conv2D (Ours)	60.3	52.2	78.1	68.9	79.7	49.0	24.3
PPC + RQ-Conv2D (Ours)	56.8	49.2	76.2	67.2	75.7	46.1	22.8
PPC + PointNet (Ours)	64.5	56.2	80.5	71.6	80.7	54.6	31.2
PPC + EdgeConv (Ours)	65.2	56.7	80.8	71.8	81.4	55.1	31.2

Table 2: **Vehicles** on the Waymo Open Dataset *validation* set. **3D APH$_{L2}$** is the primary metric for the dataset. Results denoted with * are based on our reimplementation. Others are taken from papers or via email communication with paper authors. Top two results per column are marked bold. Our method performs better than most published methods except for PV-RCNN [21]. † PV-RCNN and RCN relies on a two-stage detection pipeline, and is therefore superior quality but less efficient than the other models in this table.

competitively on vehicles, outperforming recently published methods, including several published this year.
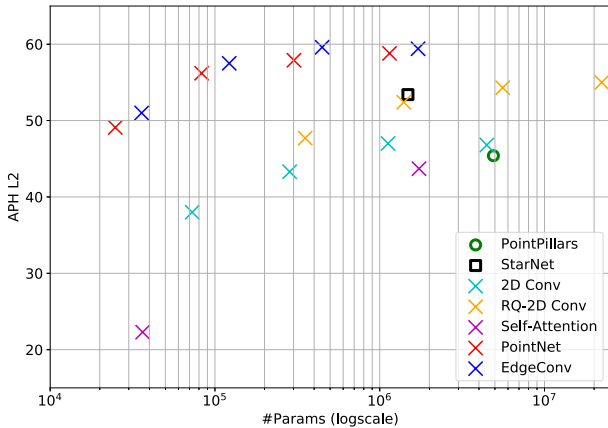
4.2. Detailed Kernel Analysis

We take a closer look to compare the five kernels introduced in Sec. 3.1. Since different kernels have different computational complexity, it is unfair to compare them by their quality alone. Fig. 2a shows a complexity-vs.-accuracy analysis, while Fig. 2b shows the model-size-vs.-accuracy analysis. For each kernel, we train multiple models with each with a different depth multiplier, ranging from 0.25 to

EdgeConv	FE$_1$	FE$_2$	FE$_3$	FA	FE$_4$	Conv2D
59.6	56.8	54.9	51.7	53.2	55.2	47.0

Table 3: Ablation to determine where the EdgeConv kernels are most effective. We start with a network of only 2D Conv kernels, and replace any one of the backbone blocks with EdgeConv. Experiments were done on the pedestrians of Waymo Open Dataset *validation* set. The APH$_{L2}$ metric is reported.
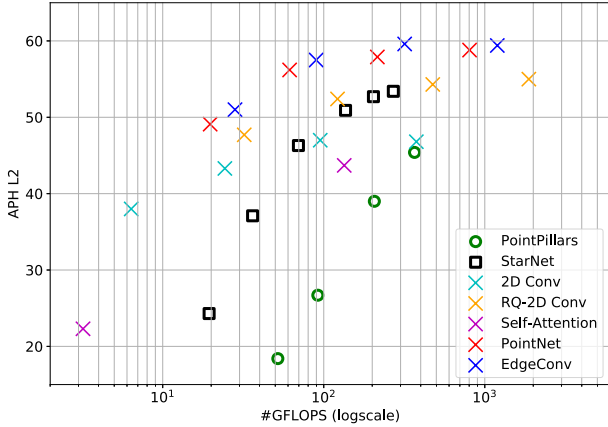
(a) Complexity vs. accuracy

(b) #Parameters vs. accuracy

Figure 2: Detailed analysis of the different Point-Set Aggregation Kernels. For each kernel, we train up to 4 models with a depth multiplier ranging from 0.25 to 2.0. PointPillars [11] and Star-Net [15] are also plotted for reference. The operating points for PointPillars and StarNet are from one single model each, but with different input dimensions. a) Complexity vs. accuracy. b) Model size vs. accuracy. Experiments done on the Waymo Open Dataset pedestrians *validation* set. The kernels with the best accuracy are PointNet and EdgeConv, while the most efficient kernels are the baseline 2D convolution and the transformer. We shed some light to the sub-par quality of the transformer in Sec. 4.2. Note that the most efficient PointNet kernel outperforms PointPillars in quality, while needing *180 times* fewer FLOPS and model parameters.

2.0. A depth multiplier is a factor that is applied to the number of channels in each layer in a network and is a simple way to yield multiple models at different complexity and accuracy.

Complexity vs. accuracy. The **2D** kernel baseline is one of the least expensive methods, as expected. The **Range-Quantized 2D** kernel with quantization of 4 buckets adds approximately four times to the complexity and improves

EdgeConv baseline	59.6
W/o smart down-sampling	58.2
Cartesian instead of polar	54.2

Table 4: Ablation on the polar vs. Cartesian parametrization and the smart down-sampling strategy. Experiments done on the pedestrians of the Waymo Open Dataset *validation* set. The APH_{L2} metric is reported.

significantly over the baseline. **Self-attention** is fairly cheap in computation. However, it does not perform as well as any other kernel. We believe that the reason might be that we kept the kernel size to be 3×3 throughout the network, and the transformer works better with larger kernel sizes. **PointNet** and **EdgeConv** have a relatively small difference in quality at around 1-2% at 20-30% difference in computation.

Model size vs. accuracy. PPC has a significant advantage over existing models in terms of model size. Our smallest model, a PointNet kernel model with a depth multiplier of 0.25, achieves higher accuracy than the baseline PointPillars, while only having **24k** parameters compared to close to 5M for PointPillars.

4.3. Mix-and-Match Kernels

In the previous section, we noticed that a network consisting of all EdgeConv kernels delivers the strongest results. However, we also observed that the EdgeConv kernel with the same depth multiplier is not as efficient as the 2D kernel. In this ablation, we study if we can keep most of the 2D kernels while only applying the EdgeConv in a few layers. Tab. 3 shows the accuracy numbers. **Conv2D** and **EdgeConv** are networks of only 2D or EdgeConv kernels and serve as a pair of pseudo upper and lower bounds. We then replace each of the blocks of the backbone from 2D to Edge-Conv. Interestingly, replacing either the first or the last block generates the most benefits. Since our backbone resembles a U-Net [20], it means that the EdgeConv kernel has the most impact with larger resolutions.

4.4. Additional Ablation

Point-Cloud-Camera Sensor Fusion. Each frame in the Waymo Open Dataset comes with five calibrated camera images capturing views from the front and sides of the car. We downsize each camera image to 400×600 pixels and use a convolutional neural network to extract a 192-dimensional feature at each location. These features are concatenated to one of the layers in the perspective point cloud network. We experimented with fusing the features at different layers of the network and found that the model performed better regardless of the layer we chose to fuse. Our best result in Tab. 1 is obtained when the camera features fuses at the input to the first extractor layer.
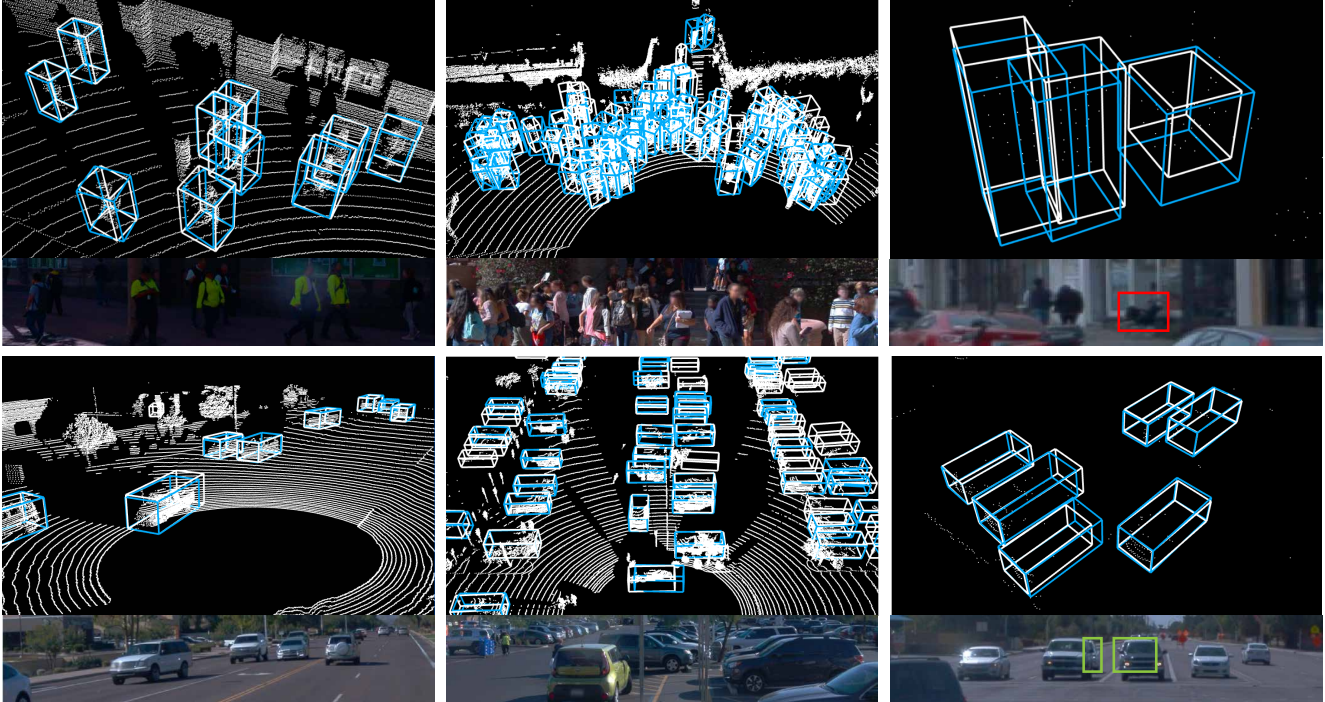
Figure 3: (Best viewed in color) Example pedestrian and vehicle detection results of PPC + EdgeConv on the Waymo Open Dataset. White boxes are groundtruth and blue boxes are our results. **Left**: Our method performs well when objects are close and mostly visible. **Center**: It can also handle large crowds with severe occlusion. Many of the false-negatives in the center bottom image have no points in the groundtruth boxes. **Right**: It can also detect objects in the long range where points become sparse. Note in the top right image, the pedestrian on the right (highlighted in a red box on the image) is sitting in a chair. And in the bottom right example, the there is severe occlusion (green boxes) for the two cars behind the front two.

Sensor Polar vs. World Cartesian. In (7), the asymmetric positional encoding is defined in the spherical polar coordinate system around the sensor. It is possible to take the displacement vector between two points in the projected world Cartesian frame instead. If so, the method's overall concept becomes very similar to PointNet++ [19], with the main difference being that the neighbors of a point are taken from the perspective range image grid rather than through nearest neighbor search. Operating in the polar coordinate system is natural in the range image. However, the Cartesian system has a strong prior in the heading since most objects move perpendicular to the ego-vehicle. In Tab. 4 we show the ablation of polar vs. Cartesian for the PPC + EdgeConv model on pedestrians. It appears that the benefits of operating in the polar coordinate system significantly outweigh the drawbacks at 59.6% vs. 54.2% APH_{L2}.

Smart Down-Sampling. The smart down- and up-sampling strategy outlined in Sec. 3.2 allows the down-sampling to avoid missing returns at the cost that the down-sampling no longer follows a regular pattern. As shown in Tab. 4, the smart sampling technique yields 1.4% benefit in APH_{L2}.

5. Conclusion and Limitations

This paper presents a new 3D representation based on the range image, which leverages recent advances in graph convolutions. It is efficient and yet powerful, as demonstrated on pedestrians and vehicles on the Waymo Open Dataset.

It is not without limitations. Most 3D detection tasks use a 7 degrees-of-freedom that only has a yaw rotation around the Z-axis in the world coordinate system. Suppose the sensor has a significant pitch or roll wrt. the world coordinate system, the boxes no longer appear only yaw-rotated in the range image. It is an issue for indoor scene datasets but less of a problem for autonomous driving configurations, where the rotating LiDAR usually sits upright to the world coordinate system.

Another challenge is data augmentation. [5] in Tab. 1 and Tab. 2 shows a significant improvement by applying data augmentation to PointPillars [11] and StarNet [15]. In 3D, data augmentation can be diverse and effective. When points are in the dense range image form, we can no longer apply most of them without disturbing the dense structure. We also observed that the EdgeConv kernel network is not sensitive to strategies that are still reasonable in the range image, e.g., random flip and random points drop.

References

[1] A. Bewley, P. Sun, Thomas Mensink, Dragomir Anguelov, and C. Sminchisescu. Range conditioned dilated convolutions for scale invariant 3d object detection. In *CoRL*, 2020. 6

[2] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020. 1, 2

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *arXiv preprint arXiv:2005.12872*, 2020. 3

[4] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 1

[5] Shuyang Cheng, Zhaoqi Leng, Ekin Dogus Cubuk, Barret Zoph, Chunyan Bai, Jiquan Ngiam, Yang Song, Benjamin Caine, Vijay Vasudevan, Congcong Li, et al. Improving 3d object detection through progressive population based augmentation. In *ECCV*, 2020. 6, 8

[6] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017. 1

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 3

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012. 1, 2

[9] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017. 1, 2

[10] Rui Huang, Wanyue Zhang, Tom Funkhouser, Abhijit Kundu, David Ross, Caroline Pantofaru, and Alireza Fathi. An lstm approach to temporal 3d object detection in lidar point clouds. In *ECCV*, 2020. 6

[11] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019. 1, 2, 4, 6, 7, 8

[12] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3d lidar using fully convolutional network. *arXiv preprint arXiv:1608.07916*, 2016. 1, 2

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. 5

[14] Gregory P Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K Wellington. Lasernet: An efficient probabilistic 3d object detector for autonomous driving. In *CVPR*, 2019. 1, 2, 5, 6, 11

[15] Jiquan Ngiam, Benjamin Caine, Wei Han, Brandon Yang, Yuning Chai, Pei Sun, Yin Zhou, Xi Yi, Ouais Alsharif, Patrick Nguyen, et al. Starnet: Targeted computation for object detection in point clouds. *arXiv preprint arXiv:1908.11069*, 2019. 3, 6, 7, 8

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

[17] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 918–927, 2018. 1, 3

[18] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 1, 2, 3, 4

[19] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 1, 2, 8

[20] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 5, 7, 11

[21] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *CVPR*, 2020. 1, 2, 6

[22] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointr-cnn: 3d object proposal generation and detection from point cloud. In *CVPR*, 2019. 1, 2, 4, 5

[23] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *CVPR*, 2015. 1

[24] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*, 2020. 1, 2, 5

[25] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *ICCV*, 2019. 1

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1, 3

[27] Yue Wang, Abhijit Kundu, Alireza Fathi, Caroline Pantofaru, David Ross, Justin Solomon, and Tom Funkhouser. Pillar-based object detection for autonomous driving. In *ECCV*, 2020. 6

[28] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics*, 2019. 1, 2, 4

[29] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 1

[30] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 2018. 1, 2

[31] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 5

[32] Yin Zhou, Pei Sun, Yu Zhang, Dragomir Anguelov, Jiyang Gao, Tom Ouyang, James Guo, Jiquan Ngiam, and Vijay Vasudevan. End-to-end multi-view fusion for 3d object detection in lidar point clouds. In *CORL*, 2019. 6

[33] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, 2018. 1, 2, 4, 11