Point Density-Aware Voxels for LiDAR 3D Object Detection

Jordan S. K. Hu   Tianshu Kuai   Steven L. Waslander
University of Toronto Robotics Institute
{jordan.hu, tianshu.kuai}@mail.utoronto.ca, steven.waslander@robotics.utias.utoronto.ca

Abstract

LiDAR has become one of the primary 3D object detection sensors in autonomous driving. However, LiDAR's diverging point pattern with increasing distance results in a non-uniform sampled point cloud ill-suited to discretized volumetric feature extraction. Current methods either rely on voxelized point clouds or use inefficient farthest point sampling to mitigate detrimental effects caused by density variation but largely ignore point density as a feature and its predictable relationship with distance from the LiDAR sensor. Our proposed solution, Point Density-Aware Voxel network (PDV), is an end-to-end two stage LiDAR 3D object detection architecture that is designed to account for these point density variations. PDV efficiently localizes voxel features from the 3D sparse convolution backbone through voxel point centroids. The spatially localized voxel features are then aggregated through a density-aware RoI grid pooling module using kernel density estimation (KDE) and self-attention with point density positional encoding. Finally, we exploit LiDAR's point density to distance relationship to refine our final bounding box confidences. PDV outperforms all state-of-the-art methods on the Waymo Open Dataset and achieves competitive results on the KITTI dataset.

1. Introduction

3D object detection is one of the key perception problems in the autonomous vehicle space as object pose estimation directly impacts the effectiveness of downstream tasks in the perception pipeline. Within the autonomous driving sensor stack, LiDAR has become one of the most popular sensors used for 3D object detection [23,25,37], because of the accurate 3D point cloud it produces through laser light.

However, the reliance on LiDAR data comes at the cost of point density variations across distance. Other factors such as occlusion play a role, but the primary reason is the natural divergence of points from the LiDAR with increasing distance due to the angular offsets between the LiDAR lasers. Thus, objects located at farther distances return fewer points than objects located closer to the LiDAR.

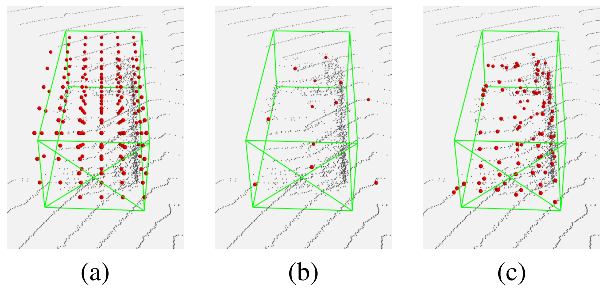
Figure 1. Voxel feature localization using (a) voxel centers, (b) farthest point sampling, and (c) voxel point centroids on a vehicle in the Waymo Open Dataset [29]. By using the raw point cloud to localize voxel features, voxel point centroids provide dense geometric information for second-stage proposal refinement.

Voxel-based methods [4,37,40,43] typically ignore point density, solely relying on the quantized representation of the point cloud. When a high voxel resolution is afforded, as is the case on the KITTI dataset [6], voxel-based methods [41] have outperformed point-based and point-voxel-based methods. However, on datasets with larger input spaces such as the Waymo Open Dataset [29], the voxel resolution is limited due to memory constraints. Fine object details are therefore lost due to spatial misalignment between the voxel features and the point cloud as shown in Figure 1 (a), resulting in a degradation in performance.

Other methods [23, 25] attempt to remedy point density variations through farthest point sampling (FPS) as seen in Figure 1 (b). Although effective at sampling locations on non-uniformly distributed point clouds, the computation scales poorly as a function of the number of points in the point cloud, increasing runtime and limiting the number of sampled points for second-stage proposal refinement.

Point density also affects detection of smaller objects such as pedestrians and cyclists. These objects have less surface area to intersect the LiDAR's laser beams, resulting in poorer object localization. Perhaps informatively, current state-of-the-art methods have largely ignored detection performance for pedestrians and cyclists, focusing solely on the car or vehicle class [4, 14, 15, 41]. As we move towards datasets with higher environment coverage, it is necessary for architectures to be scalable to larger input spaces and to serve as a multi-class solution for 3D object detection.

We therefore propose Point Density-Aware Voxel network (PDV) to resolve these identified issues by leveraging voxel point centroid localization and feature encodings that directly account for point density in multi-class 3D object detection. We summarize our approach with the following three contributions.

(1) Voxel Point Centroid Localization. PDV partitions the LiDAR points in each non-empty voxel to calculate a point centroid for each voxel feature, as shown in Figure 1 (c). By localizing the voxel features with point centroids for second-stage proposal refinement, PDV uses the point density distributions to retain fine-grained position information in the feature encodings without requiring an expensive point cloud sampling method such as FPS.

(2) Density-Aware RoI Grid Pooling. We augment region of interest (RoI) grid pooling [23] to encode local point density as an additional feature. First, we use kernel density estimation (KDE) [17, 21] to encode local voxel feature density at each grid point ball query, followed by self-attention [33] between grid points with a novel point density positional encoding. Density-aware RoI grid pooling captures localized point density information in the context of the whole region proposal for second-stage refinement.

(3) Density Confidence Prediction. We further refine our bounding box confidence predictions by using the final bounding box centroid location and the number of raw LiDAR points within the final bounding box as additional features. Thus, we use the inherent relationship between distance and point density established by LiDAR for more informed confidence predictions.

PDV outperforms all current state-of-the-art methods on the Waymo Open Dataset [29] with an increase of +0.65%/+1.25%, +0.53%/+0.46%, and +0.49%/+0.71% on the vehicle, pedestrian, and cyclist LEVEL_1/LEVEL_2 mAPH classes, respectively, and achieves competitive performance on the KITTI dataset [6].

2. Related Work

Point-based LiDAR 3D Object Detection. Point-based methods use the raw point cloud to extract point-level features for bounding box predictions. F-PointNet [18] applies PointNet [19, 20] on the point cloud, segmented via image-based 2D object detections. PointRCNN [25] directly generates RoIs at the point-level through a PointNet++ [20] backbone and uses point-level features for bounding box refinement. STD [39] proposes PointsPool for RoI feature extraction while 3DSSD [38] adopts a new sampling strategy on the raw point cloud to preserve enough interior points for objects in the downsampled point cloud. Point-GNN [27] constructs a graph using the raw point cloud and aggregates node-level features to generate predictions. Point-based methods utilize expensive point cloud sampling and grouping, which inevitably require long inference times.

Voxel-based LiDAR 3D Object Detection. Voxel-based methods divide the point cloud into a voxel grid to directly apply 3D and 2D convolutions for generating predictions [11, 37, 43]. CIA-SSD [40] adopts a light network on the bird's eye biew (BEV) grid to extract robust spatial-semantic features with a confidence rectification module for better post-processing. Voxel-RCNN [4] proposes Voxel RoI pooling to generate RoI features by aggregating voxel features. VoTr [15] proposes a transformer-based 3D backbone as an alternative to the standard sparse convolution layers [7, 37]. The performance of voxel-based methods is limited by the quantized point cloud as fine-grained point-level information is lost from the voxelization process.

Point-Voxel-based LiDAR 3D Object Detection. Point-voxel-based methods utilize both voxel and point representations of the point cloud. SA-SSD [8] uses an auxiliary network during training that interpolates point-level features from intermediate voxel layers. PV-RCNN [23] adopts RoI grid pooling to effectively aggregate FPS-sampled keypoint features on evenly spaced grids inside each bounding box proposal. PV-RCNN++ [24] proposes a modified version of FPS for faster point sampling and VectorPool aggregation for RoI grid pooling. CT3D [22] constructs a cylindrical-shaped RoI at each bounding box proposal. It adopts a transformer-based encoder-decoder architecture to extract RoI features directly from the nearby points without using intermediate voxel features. Pyramid-RCNN [14] extends the idea of RoI grid pooling over a single set of evenly spaced grids to multiple sets of grids at different scales with adaptive ball query radius at a significantly higher computation cost. Current point-voxel based methods do not explicitly account for variations in point cloud density within each RoI and typically require long inference times due to the dependence on point cloud sampling.

Point Density Estimation. KDE [17, 21] estimates a probability distribution function of a random variable using a finite set of samples and a chosen kernel function and bandwidth. A couple methods have used KDE for feature encoding in point clouds. MC Convolution [9] uses a Monte Carlo estimation of the convolution integral to handle non-uniform sampled point clouds and uses KDE to estimate the likelihood of a point within a local convolution. PointConv [36] also uses KDE, but estimates the likelihood of each sample with an additional feed-forward network (FFN). Rather than restricting the density estimate for reweighting, we use KDE as an additional feature within each grid point ball query in density-aware RoI grid pooling.

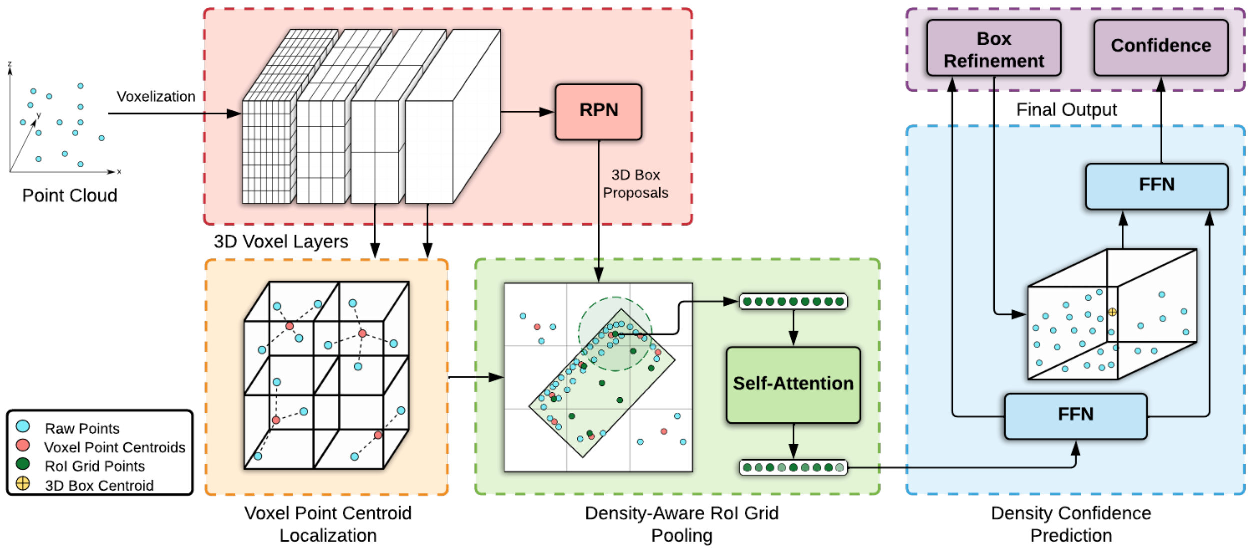

Figure 2. PDV architecture. The input point cloud is first voxelized and processed through 3D sparse convolutions and a RPN head to produce initial bounding box proposals. Voxel features in each layer $l = 1, \dots, L$ are then localized via voxel point centroids, which are then aggregated through density-aware RoI grid pooling. The RoI grid features are used to refine the bounding box proposals and their associated confidence, with additional adjustments from density confidence prediction.

3. Methodology

PDV uses a two-stage approach with a 3D sparse convolution backbone for initial bounding box proposals which are then refined in a second stage through voxel features in each voxel layer and the raw point cloud data. Figure 2 shows an overview of the PDV framework.

3.1. 3D Voxel Backbone

We use a similar voxel backbone for initial bounding box proposals to SECOND [37]. The input to PDV is a point cloud which is defined as a set of 3D points $\{\mathbf{p}_i = \{\mathbf{x}_{\mathbf{p}_i}, \mathbf{f}_{\mathbf{p}_i}\} \mid i = 1, \dots, N_\mathbf{p}\}$ where $\mathbf{x}_{\mathbf{p}_i} \in \mathbb{R}^3$ are the xyz spatial coordinates, $\mathbf{f}_{\mathbf{p}_i} \in \mathbb{R}^F$ are additional features such as the intensity or elongation of each point, and $N_\mathbf{p}$ is the number of points in the point cloud. First, the point cloud is voxelized and subsequently encoded using a series of 3D sparse convolutions [7, 37], followed by a region proposal network (RPN) for initial bounding box proposals. Each voxel layer has a different spatial resolution with 1x, 2x, 4x, and 8x downsampled resolutions based on the original voxel grid size. The voxel features in each layer are used for bounding box refinement in the second stage.

3.2. Voxel Point Centroid Localization

Inspired by grid-subsampling in KPConv [31, 32], the voxel point centroid localization module spatially locates non-empty voxel features for aggregation in density-aware RoI grid pooling.

Let $\mathcal{V}^l = \{\mathbf{V}_k^l = \{\mathbf{h}_{\mathbf{V}_k^l}, \mathbf{f}_{\mathbf{V}_k^l}\} \mid k = 1, \dots, N_l\}$ be the set of non-empty voxels in the l-th voxel layer where $\mathbf{h}_{\mathbf{V}_k^l}$ is the 3D voxel index, $\mathbf{f}_{\mathbf{V}_k^l}$ is the associated voxel feature vector, and N_l is the number of non-empty voxels for voxel layers $l = 1, \dots, L$. First, points that are within the same voxel are grouped together into a set $\mathcal{N}(\mathbf{V}_k^l)$ by calculating their voxel index $\mathbf{h}_{\mathbf{V}_k^l}$ from their spatial coordinates $\mathbf{x}_i$ and voxel grid dimensions. The point centroid of each voxel feature is then calculated as

$$\mathbf{c}_{\mathbf{V}_k^l} = \frac{1}{|\mathcal{N}(\mathbf{V}_k^l)|} \sum_{\mathbf{x}_{\mathbf{p}_i} \in \mathcal{N}(\mathbf{V}_k^l)} \mathbf{x}_{\mathbf{p}_i}. \tag{1}$$

Since the voxels in the convolution layers are in a sparse format, an intermediate hash table is used to efficiently map each calculated voxel point centroid to its corresponding feature vector. As shown in Figure 3, both voxel point centroids and sparse voxel features are associated with a shared voxel index. The intermediate hash table links the centroid $\mathbf{c}_{\mathbf{V}_k^l}$ with $\mathbf{V}_k^l$ using the matching voxel index $\mathbf{h}_{\mathbf{V}_k^l}$.

An advantage with using voxels is that we can use the previous voxel layer centroid calculations to efficiently compute the subsequent voxel point centroids based on the stride, padding, and kernel size of the convolution block. Let $\mathcal{C}_k^{l+1} = \{\mathbf{c}_{\mathbf{V}_j^l} \mid \mathcal{K}_{l+1}(\mathbf{h}_{\mathbf{V}_j^l}) = \mathbf{h}_{\mathbf{V}_k^{l+1}}\}$ be the set of voxel point centroids where $\mathcal{K}_{l+1}$ is the convolution block that maps voxel index $\mathbf{h}_{\mathbf{V}_j^l}$ to $\mathbf{h}_{\mathbf{V}_k^{l+1}}$. We can then perform

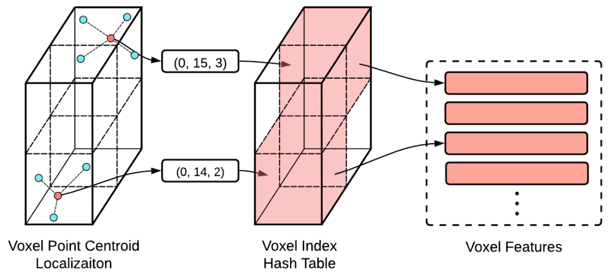

Figure 3. Voxel point centroids (red) are assigned to their respective voxel grid index. A 3D hash table maps the voxel index to the associated voxel feature in the sparse convolution layer.

a weighted average of the grouped voxel point centroids to calculate the centroids in the subsequent layer:

$$\mathbf{c}_{\mathbf{V}_k^{l+1}} = \frac{1}{\left|\mathcal{N}(\mathbf{V}_k^{l+1})\right|} \sum_{\mathbf{c}_{\mathbf{V}_j^l} \in \mathcal{C}_k^{l+1}} \left|\mathcal{N}(\mathbf{V}_j^l)\right| \mathbf{c}_{\mathbf{V}_j^l} \tag{2}$$

where $\left|\mathcal{N}(\mathbf{V}_k^{l+1})\right|$ is calculated from

$$\left|\mathcal{N}(\mathbf{V}_k^{l+1})\right| = \sum_{\mathbf{c}_{\mathbf{V}_j^l} \in \mathcal{C}_k^{l+1}} \left|\mathcal{N}(\mathbf{V}_j^l)\right|. \tag{3}$$

By avoiding recomputing centroids using the entire point cloud for each layer, voxel point centroid localization scales to larger point clouds more efficiently.

3.3. Density-aware RoI Grid Pooling

Density-aware RoI grid pooling builds upon RoI grid pooling [23] by augmenting the pooling method with a combination of KDE and self-attention to encode point density features into each proposal. First, $U \times U \times U$ uniform grid points $\mathcal{G}^{\mathbf{b}} = \{\mathbf{g}_1, \ldots, \mathbf{g}_{U^3}\}$ are sampled for each bounding box proposal $\mathbf{b}$.

Local Feature Density. We use KDE to estimate local feature density within each grid point ball query. Rather than limiting the estimated density as a feature reweighting like MC Convolution [9] and PointConv [36], density-aware RoI grid pooling encodes the estimated probability density as an additional feature in the ball query for a more implicit feature encoding. First, we aggregate neighbouring features near each grid point where $\mathcal{N}(\mathbf{g}_j)$ is the set of voxel point centroids in a sphere of radius r centered around $\mathbf{g}_j$:

$$\Psi_{\mathbf{g}_j}^l = \left\{ \begin{bmatrix} \mathbf{f}_{\mathbf{V}_k^l} \\ \mathbf{c}_{\mathbf{V}_k^l} - \mathbf{g}_j \\ p(\mathbf{c}_{\mathbf{V}_k^l} | \mathbf{g}_j) \end{bmatrix}^\top, \forall \mathbf{c}_{\mathbf{V}_k^l} \in \mathcal{N}(\mathbf{g}_j) \right\} \tag{4}$$

where the local offset $\mathbf{c}_{\mathbf{V}_k^l} - \mathbf{g}_j$ and likelihood $p(\mathbf{c}_{\mathbf{V}_k^l}|\mathbf{g}_j)$ are appended as additional features, as shown in Figure 4.

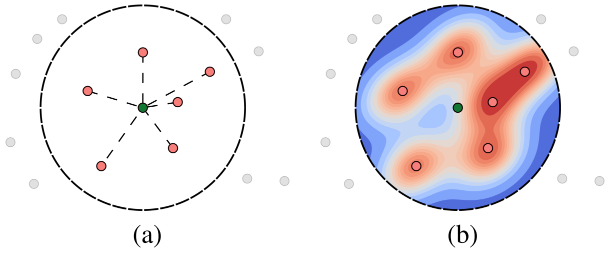

Figure 4. Two types of features are added to each grid point ball query: (a) the relative offset from the ball query center, and (b) the probability density of each point calculated through KDE. Red and blue indicate higher and lower probability density, respectively.

The likelihood is calculated for each grid point using KDE:

$$p(\mathbf{c}_{\mathbf{V}_k^l}|\mathbf{g}_j) \approx \frac{1}{|\mathcal{N}(\mathbf{g}_j)|\sigma^3} \sum_{\mathbf{c}_{\mathbf{V}_i^l} \in \mathcal{N}(\mathbf{g}_j)} \mathcal{W}(\mathbf{c}_{\mathbf{V}_k^l}, \mathbf{c}_{\mathbf{V}_i^l}) \tag{5}$$

where σ is the bandwidth and $\mathcal{W}$ is

$$\mathcal{W}(\mathbf{c}_{\mathbf{V}_k^l}, \mathbf{c}_{\mathbf{V}_i^l}) = \prod_{d=1}^{3} w\left(\frac{\mathbf{c}_{\mathbf{V}_k^l,d} - \mathbf{c}_{\mathbf{V}_i^l,d}}{\sigma}\right) \tag{6}$$

with an independent kernel w on each xyz dimension d. Once the features are appended, a PointNet multi-scale grouping (MSG) module [19, 20] is used to obtain a feature vector $\mathbf{f}_{\mathbf{g}_j}^l$ for each grid point $\mathbf{g}_j$:

$$\mathbf{f}_{\mathbf{g}_j}^l = \text{maxpool}(\text{FFN}(\Psi_{\mathbf{g}_j}^l)). \tag{7}$$

We use multiple radii r to capture feature density at different scales for each grid point and concatenate the output features together. Finally, features are appended from different voxel layers to obtain the final features for each grid point:

$$\mathbf{f}_{\mathbf{g}_j} = \left[\mathbf{f}_{\mathbf{g}_j}^1, \ldots, \mathbf{f}_{\mathbf{g}_j}^L\right]. \tag{8}$$

Grid Point Self-Attention. The features encoded at each RoI grid point are localized to the size of the ball query but lack interdependent relationships between different grid points. An easy solution is to use self-attention [33] to capture long-range dependencies between the grid points but simply adding an attention module lacks the geometric information of the LiDAR point cloud. Thus, we also introduce a novel type of positional encoding that takes into consideration the point density within the point cloud.

As shown in Figure 5, the self-attention module performs self-attention between the non-empty grid point features $\mathbf{f}_{\mathcal{G}^{\mathbf{b}}} = \left\{\mathbf{f}_{\mathbf{g}_i} \,\middle|\, |\mathcal{N}(\mathbf{g}_i)| > 0, \forall \mathbf{g}_i \in \mathcal{G}^{\mathbf{b}}\right\}$ using a standard transformer encoder layer [33] and a residual connection similar to a non-local neural network block [34]:

$$\tilde{\mathbf{f}}_{\mathbf{g}_i} = \mathcal{T}_{\mathbf{g}_i}(\mathbf{f}_{\mathcal{G}^{\mathbf{b}}}) + \mathbf{f}_{\mathbf{g}_i} \tag{9}$$

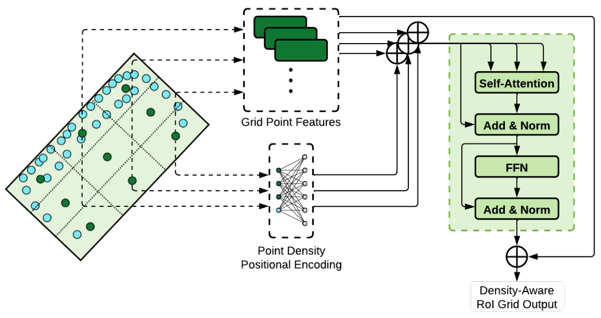
Grid Point Features
Point Density Positional Encoding
Self-Attention
Add & Norm
FFN
Add & Norm
Density-Aware RoI Grid Output

Figure 5. Density-aware RoI grid pooling performs self-attention on the grid point features (green) to capture long-range dependencies. Point density positional encoding uses the relative offset of the grid point and number of points (blue) in each grid voxel as inputs.

where $\mathcal{T}_{\mathbf{g}_i}$ is the transformer encoder layer output for $\mathbf{f}_{\mathbf{g}_i}$ and $\tilde{\mathbf{f}}_{\mathbf{g}_i}$ is the output grid feature. Empty grid point features $|\mathcal{N}(\mathbf{g}_i)| = 0$ are untouched by the self-attention module and are kept as their original feature encoding.

Point Density Positional Encoding. We add positional encoding to the self-attention module by using the local grid point positions and the number of points in the box proposal. The bounding box proposal is divided into voxels $\mathbf{V}_{\mathbf{g}_j}$ using the same $U \times U \times U$ grid resolution to establish voxels for each grid point. The positional encoding for each grid feature is then calculated as

$$\text{PE}(\mathbf{f}_{\mathbf{g}_j}) = \text{FFN}\left(\left[\boldsymbol{\delta}_{\mathbf{g}_j}, \log\left(\left|\mathcal{N}\left(\mathbf{V}_{\mathbf{g}_j}\right)\right| + \epsilon\right)\right]\right) \tag{10}$$

where $\boldsymbol{\delta}_{\mathbf{g}_j} = \mathbf{x}_{\mathbf{g}_j} - \mathbf{c}_{\mathbf{b}}$ is the relative position of $\mathbf{g}_j$ from the bounding box proposal centroid $\mathbf{c}_{\mathbf{b}}$, $|\mathcal{N}(\mathbf{V}_{\mathbf{g}_j})|$ is the number of points in each grid point voxel $\mathbf{V}_{\mathbf{g}_j}$, and ϵ is a constant offset. By leveraging the local offsets and number of points within each voxel, density-aware RoI grid pooling is able to capture point densities within each region proposal.

3.4. Density Confidence Prediction

PDV also uses the relationship between the distance and the number of LiDAR points on scanned objects to predict the confidence of the final bounding box predictions. A shared FFN first encodes the flattened features from the density-aware RoI grid pooling module. Then, two separate FFN branches encode the features for the box refinement and box confidence outputs. In the box confidence branch, we additionally append two features to predict the output confidence $p_{\tilde{\mathbf{b}}}$ for the final bounding box $\tilde{\mathbf{b}}$:

$$p_{\tilde{\mathbf{b}}} = \text{FFN}\Big(\Big[\mathbf{f}^s_{\tilde{\mathbf{b}}}, \mathbf{c}_{\tilde{\mathbf{b}}}, \log\big(|\mathcal{N}(\tilde{\mathbf{b}})|\big)\Big]\Big) \tag{11}$$

where $\mathbf{f}^s_{\tilde{\mathbf{b}}}$ is the output feature vector from the shared FFN, $\mathbf{c}_{\tilde{\mathbf{b}}}$ is the centroid of the final bounding box, and $|\mathcal{N}(\tilde{\mathbf{b}})|$ is the number of raw points in the final bounding box.

3.5. Training Losses

We use an end-to-end training strategy for PDV with a region proposal loss L_{RPN} and proposal refinement loss L_{RCNN} that are trained jointly. The L_{RPN} is calculated as

$$L_{\text{RPN}} = L_{\text{cls}}(\mathbf{y}_{\mathbf{b}}, \mathbf{y}^\star_{\mathbf{b}}) + \beta L_{\text{reg}}(\mathbf{r}_{\mathbf{b}}, \mathbf{r}^\star_{\mathbf{b}}) \tag{12}$$

where L_{cls} is the focal loss [12], L_{reg} is the smooth-L1 loss, $\mathbf{y}_{\mathbf{b}}$ is the predicted class vector, $\mathbf{y}^\star_{\mathbf{b}}$ is the ground truth class, $\mathbf{r}_{\mathbf{b}}$ is the predicted RoI anchor residual, $\mathbf{r}^\star_{\mathbf{b}}$ is the ground truth anchor residual, and β is a scaling factor. L_{RCNN} is composed of

$$L_{\text{IoU}} = -p^\star_{\tilde{\mathbf{b}}}\log(p_{\tilde{\mathbf{b}}}) - (1 - p^\star_{\tilde{\mathbf{b}}})\log(1 - p_{\tilde{\mathbf{b}}}) \tag{13}$$

where $p^\star_{\tilde{\mathbf{b}}}$ is the confidence training target scaled by the 3D RoI and their associated ground truth bounding box as done in PV-RCNN [23]. Thus L_{RCNN} is

$$L_{\text{RCNN}} = L_{\text{IoU}} + L_{\text{reg}}(\mathbf{r}_{\tilde{\mathbf{b}}}, \mathbf{r}^\star_{\tilde{\mathbf{b}}}) \tag{14}$$

where $\mathbf{r}_{\tilde{\mathbf{b}}}$ is the predicted bounding box residual and $\mathbf{r}^\star_{\tilde{\mathbf{b}}}$ is the ground truth residual. A smooth-L1 loss is used to regress the bounding box residuals. We use the same confidence and regression targets as PV-RCNN [23].

4. Experimental Results

Datasets. We evaluate PDV on the Waymo Open Dataset [29] and the KITTI 3D Object Detection benchmark [6]. The Waymo Open Dataset is one of the biggest and most diverse autonomous driving datasets available, containing 798 training sequences (approximately 158k point cloud samples) and 202 validation sequences (approximately 40k point cloud samples), with annotations for objects in full 360° field of view. The Waymo Open Dataset uses standard mean average precision (mAP) as well as mAPH, which factors in heading angle. The predictions are split into LEVEL_1, which only includes 3D labels with more than five LiDAR points, and LEVEL_2, which also includes 3D labels with at least one LiDAR point. The KITTI dataset contains 7,481 training samples and 7,518 testing samples and uses standard average precision (AP) on easy, moderate and hard difficulties. We adopt the standard *training* and *val* split sets from Chen et al. [3].

Input Parameters. For the Waymo Open Dataset [29], the detection range is [-75.2m, 75.2m] for the X and Y axes, and [-2m, 4m] for the Z axis. We divide the raw point cloud into voxels of size (0.1m, 0.1m, 0.15m). Since the KITTI dataset [6] only provides annotations in front camera's field of view, its detection range is set to be [0, 70.4m] for the X axis, [-40m, 40m] for the Y axis, and [-3m, 1m] for the Z axis. We set the voxel size to be (0.05m, 0.05m, 0.1m).

Network Architecture. PDV uses the last two voxel layers $l = 3, 4$ for voxel point centroid localization. Density-aware RoI grid pooling uses a grid size of $U = 6$. Each

Method	Veh. (LEVEL_1)		Veh. (LEVEL_2)		Ped. (LEVEL_1)		Ped. (LEVEL_2)		Cyc. (LEVEL_1)		Cyc. (LEVEL_2)	
	mAP	mAPH	mAP	mAPH	mAP	mAPH	mAP	mAPH	mAP	mAPH	mAP	mAPH
SECOND [37]	72.27	71.69	63.85	63.33	68.7	58.18	60.72	51.31	60.62	59.28	58.34	57.05
PointPillar [11]	56.62	-	-	-	59.25	-	-	-	-	-	-	-
MVF* [42]	62.93	-	-	-	65.33	-	-	-	-	-	-	-
Pillar-OD* [35]	69.8	-	-	-	72.51	-	-	-	-	-	-	-
AFDet [5]	63.69	-	-	-	-	-	-	-	-	-	-	-
Part-A2-Net† [26]	74.82	74.32	65.88	65.42	71.76	63.64	62.53	55.3	67.35	66.15	65.05	63.89
PV-RCNN† [23]	75.17	74.6	66.35	65.84	72.65	63.52	63.42	55.29	67.26	65.82	64.88	63.48
PV-RCNN++ [24]	76.14	75.62	68.05	67.56	73.97	65.43	65.64	57.82	68.38	67.06	65.92	64.65
Voxel-RCNN [4]	75.59	-	66.59	-	-	-	-	-	-	-	-	-
CT3D [22]	76.3	-	69.04	-	-	-	-	-	-	-	-	-
VoTr-TSD [15]	74.95	74.25	65.91	65.29	-	-	-	-	-	-	-	-
Pyramid-PV [14]	76.3	75.68	67.23	66.68	-	-	-	-	-	-	-	-
PDV (Ours)	**76.85**	**76.33**	**69.30**	**68.81**	**74.19**	**65.96**	**65.85**	**58.28**	**68.71**	**67.55**	**66.49**	**65.36**
Improvement	*+0.55*	*+0.65*	*+0.26*	*+1.25*	*+0.22*	*+0.53*	*+0.21*	*+0.46*	*+0.33*	*+0.49*	*+0.57*	*+0.71*

Table 1. Performance comparison on the Waymo Open Dataset with 202 validation sequences for 3D vehicle (IoU = 0.7), pedestrian (IoU = 0.5) and cyclist (IoU = 0.5) detection. $\star$: Results are on Waymo Open Dataset 1.0 version. $\dagger$: Results are from [24].

Method	LEVEL 1 mAP/mAPH		
	0-30m	30-50m	50m-Inf
PV-RCNN* [23]	91.92/91.34	69.21/68.53	42.17/41.31
Voxel-RCNN [4]	92.49/-	74.09/-	53.15/-
CT3D [22]	92.51/-	75.07/-	**55.36**/-
VoTr-TSD [15]	92.28/91.73	73.36/72.56	51.09/50.01
Pyramid-PV [14]	92.67/92.20	74.91/74.21	54.54/53.45
PDV (Ours)	**93.13/92.71**	**75.49/74.91**	54.75/**53.90**
Method	LEVEL 2 mAP/mAPH		
	0-30m	30-50m	50m-Inf
PV-RCNN* [23]	91.58/91.00	65.13/64.49	36.46/35.70
Voxel-RCNN [4]	91.74/-	67.89/-	40.80/-
CT3D [22]	91.76/-	68.93/-	**42.60**/-
PDV (Ours)	**92.41/91.99**	**69.36/68.81**	42.16/**41.48**

Table 2. Performance comparison on the Waymo Open Dataset with 202 validation sequences for 3D vehicle detection across distance. $\star$: Results are on Waymo Open Dataset 1.0 version.

Model	LEVEL_2 mAPH		
	Veh.	Ped.	Cyc.
PV-RCNN† [23]	68.41	57.61	63.98
PV-RCNN++ [24]	69.71	59.72	65.17
PDV (Ours)	**69.98**	**60.00**	**67.88**

Table 3. Performance comparison on the Waymo validation set for 3D multi-class with first and second LiDAR return. $\dagger$: Results are from [24].

ball query uses a set of radii $r = [[0.8, 1.2], [1.2, 2.4]]$ to aggregate voxel point centroid features at each layer with two 32-layer FFNs for $l = 3$ and two 64-layer FFNs for $l = 4$. KDE for each ball query is calculated using a Gaussian kernel, w, with bandwidth $\sigma = 0.25$. The self-attention module uses one transformer encoder layer with a single attention head. We follow Pointformer [16] for the FFN size for point density positional encoding with the added density feature from the number of points in each RoI grid voxel.

Training and Inference Details. PDV is trained end-to-end with the Adam optimizer [10]. We start with an initial learning rate of 0.01 and update it using one-cycle policy [28] and cosine annealing [13]. We train the model for 50 epochs on the Waymo Open Dataset [29] with a batch size of 24 on 6 NVIDIA Tesla V100 GPUs and 80 epochs on the KITTI Dataset [6] with a batch size of 4 on 2 NVIDIA Tesla P100 GPUs. We adopt commonly used data augmentation strategies for LiDAR 3D object detection, including random flipping about the X axis or Y axis, random global scaling with random scaling factor between 0.95 and 1.05, random global rotations about the Z axis between $-\frac{\pi}{4}$ and $\frac{\pi}{4}$, and ground truth data augmentation [37]. For post-processing, we use a non-maximum-suppresion (NMS) threshold of 0.1 for both Waymo and KITTI to remove redundant boxes.

4.1. Waymo Dataset Results

We report the multi-class Waymo Open Dataset results on the validation set in Table 1. PDV achieves state-of-the-art results on all classes on both LEVEL_1 and LEVEL_2 mAP/mAPH metrics. We outperform methods that use PV-RCNN as a base architecture [14, 15, 23] by at least +2.13% on vehicle LEVEL_2 mAPH. PDV performs well on the other classes as well, increasing performance by +2.99% and +1.88% on pedestrian and cyclist LEVEL_2 3D mAPH, respectively, compared to PV-RCNN. We also outperform PV-RCNN++ [24] by +1.25%, +0.46%, and +0.71% on vehicle, pedestrian, and cyclist LEVEL_2 3D mAPH, respectively. PDV effectively captures fine point details lost in the voxel backbone through point density for accurate bounding box refinement in the second stage.

Table 2 shows PDV's performance across distance on the

Method	Car 3D (IoU = 0.7)			Pedestrian 3D (IoU = 0.5)			Cyclist 3D (IoU = 0.5)		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
PV-RCNN* [23]	92.10	84.36	82.48	64.26	56.67	51.91	88.88	71.95	66.78
CT3D* [22]	92.34	84.97	82.91	61.05	55.57	51.10	89.01	71.88	67.91
PDV (Ours)	**92.56**	**85.29**	**83.05**	**66.90**	**60.80**	**55.85**	**92.72**	**74.23**	**69.60**
Improvement	*+0.22*	*+0.32*	*+0.14*	*+2.64*	*+4.13*	*+3.94*	*+3.71*	*+2.28*	*+1.69*

Table 4. 3D detection results on the KITTI *val* set for car, pedestrian, and cyclist classes using AP$|_{R_{40}}$. $\star$: Results are taken from publicly released models [22, 23, 30].

Method	Car 3D (IoU=0.7)			Cyclist 3D (IoU=0.5)		
	Easy	Mod.	Hard	Easy	Mod.	Hard
SECOND [37]	83.34	72.55	65.82	71.33	52.08	45.83
PointPillar [11]	82.58	74.31	68.99	77.10	58.65	51.92
STD [39]	87.95	79.71	75.09	78.69	61.59	55.30
3DSSD [38]	88.36	79.57	74.55	$\underline{82.48}$	64.10	56.90
SA-SSD [8]	88.75	79.79	74.16	-	-	-
SE-SSD [41]	**91.49**	**82.54**	77.15	-	-	-
PV-RCNN [23]	90.25	81.43	76.82	78.60	63.71	57.65
PV-RCNN++ [24]	90.14	81.88	77.15	82.22	67.33	60.04
Voxel-RCNN [4]	$\underline{90.90}$	81.62	77.06	-	-	-
DSA-PV [1]	88.25	81.46	76.96	82.19	**68.54**	**61.33**
CT3D [22]	87.83	81.77	77.16	-	-	-
VoTr-TSD [15]	89.90	$\underline{82.09}$	**79.14**	-	-	-
Pyramid-PV [14]	88.39	82.08	$\underline{77.49}$	-	-	-
PDV (Ours)	90.43	81.86	77.36	**83.04**	$\underline{67.81}$	$\underline{60.46}$

Table 5. 3D detection results on the KITTI *test* set for car and cyclist using AP$|_{R_{40}}$. Bolded and underlined values are best and second-best performance, respectively.

Waymo Open Dataset for the vehicle class. PDV outperforms all methods by +0.46%/+0.51% and +0.42%/+0.70% at 0-30m and 30-50m for LEVEL_1 mAP/mAPH, respectively, and +0.65%/+0.99% and +0.43%/+4.32% for LEVEL_2 mAP/mAPH, respectively, better utilizing Waymo's close-range LiDAR data for accurate detection. We also report the results on the Waymo validation set using first and second LiDAR return in Table 3, surpassing both PV-RCNN and PV-RCNN++. Detailed second LiDAR return results can be found in the supplementary materials.

4.2. KITTI Dataset Results

Table 4 shows the results on the KITTI *val* set. PDV achieves state-of-the-art multi-class results, improving 3D AP$|_{R_{40}}$ performance by +0.32%, +4.13%, and +2.28% on car, pedestrian, and cyclist classes, respectively, on the moderate difficulty. As shown in Table 5, PDV also obtains competitive results on the KITTI *test* set, showing improvements over PV-RCNN [23], Voxel-RCNN [4], and CT3D [22] with an increase of at least +0.09% on the moderate 3D AP$|_{R_{40}}$ car class, but falling short compared to other methods [14, 15, 24, 41]. We hypothesize that since detection architectures can use a higher voxel resolution for the truncated KITTI point cloud, PDV's modules provide smaller improvements compared to other methods.

4.3. Ablation Studies

We provide ablation studies to study the effects of each component in PDV. All models and variations are trained on 10% of the Waymo training dataset for 60 epochs.

Components. Table 6 shows the relative performance gain of each component on LEVEL_2 mAPH on the Waymo Open Dataset validation set. Experiment 1 uses voxel centers to localize voxel features. By localizing voxel features using voxel point centroids instead, Experiment 2 provides an improvement of +0.83%, +3.86%, and +3.67% for the vehicle, pedestrian, and cyclist classes, respectively. Voxel feature localization is extremely beneficial for smaller objects since voxel centers do not have proper alignment with the point cloud. By localizing the features closer to the object's scanned surface, voxel point centroids contain meaningful geometric shape information for proposal refinement.

Local feature density estimation improves scores by +0.07%, 4.45%, +1.78% as shown in Experiment 3. By capturing feature density relationships via KDE, local feature density estimation provides a large improvement for the deformable pedestrian class, where encoded features have variable spatial configurations. Experiment 4 shows attention improves results by +0.45%, +0.61%, +1.43% providing a large increase in the pedestrian and cyclist class by establishing long-range dependencies between RoI grid points. Finally, Experiment 5 shows the effect of density confidence prediction, improving performance on the pedestrian and cyclist classes by +0.23% and +0.74%, respectively, with a small drop on the vehicle class of -0.01%.

Point Density Positional Encoding. Table 7 shows the effects of point density positional encoding in density-aware RoI pooling. Experiment 1 shows the baseline performance without any positional encoding. Experiment 2 uses a sinusoidal encoding similar to DETR [2, 33] where each spatial dimension is encoded with independent sine and cosine functions at different frequencies, showing an improvement on the pedestrian and cyclist by +0.14% and +0.50%, but a reduction on vehicle by -0.25%. A FFN with spatial grid point coordinates as input provides an overall increase in performance of +0.17%, +0.18%, and +0.27% on vehicles, pedestrians, and cyclists, respectively, as shown in Experiment 3. In Experiment 4, the density feature, in the form of

Exp.	VC	LD	GA	DC	LEVEL_2 mAPH		
					Veh.	Ped.	Cyc.
1					64.01	43.15	56.41
2	✓				64.84	47.01	60.08
3	✓	✓			64.91	51.46	61.86
4	✓	✓	✓		**65.36**	52.07	63.29
5	✓	✓	✓	✓	65.35	**52.30**	**64.03**

Table 6. PDV ablation experiments trained on 10% of the Waymo training set. VC indicates voxel point centroids, LD indicates local feature density estimation, GA indicates grid point self-attention, and DC indicates density confidence prediction.

Exp.	PE	LEVEL_2 mAPH		
		Veh.	Ped.	Cyc.
1	None	65.20	51.23	62.21
2	Sinusoidal	64.95	51.37	62.71
3	FFN (XYZ)	65.37	51.41	62.48
4	FFN (D)	**65.42**	51.96	62.88
5	FFN (XYZD)	65.35	**52.30**	**64.03**

Table 7. Positional encoding (PE) ablation experiments trained on 10% of the Waymo training dataset. XYZ indicates spatial grid locations and D indicates number of points in each grid voxel.

number of points within each grid point voxel, shows similar improvements over the baseline with +0.22%, +0.73%, and +0.67%. When we combine both local spatial coordinates and density in Experiment 5, we obtain the best performance with increases of +0.15%, +1.07%, +1.82%.

4.4. Runtime Analysis

Table 8 shows the runtime comparisons of PDV. We train the models for 60 epochs on 10% of the training data for Waymo and 80 epochs on the *training* set for KITTI. Each model is evaluated with an Intel i7-6850K processor, a single Titan Xp GPU, and a batch size of 1. For a fair comparison, we use the same number of K proposals from the RPN ($K = 275$ for Waymo and $K = 100$ for KITTI) for all models. PDV outperforms PV-RCNN on inference speed and performance, reducing runtime by 14% and 5% for Waymo and KITTI, respectively, and improving mAPH by +2.42% and mAP by +1.51% on Waymo and KITTI, respectively.

Table 8 also shows the issue with voxel-based methods on larger input spaces. Although Voxel-RCNN is computationally efficient and performs well on the KITTI dataset, its performance degrades significantly, performing worse than PV-RCNN by -0.67% on Waymo. On the other hand, voxel point centroids provide an efficient alternative that retains spatial fidelity across the two datasets. Using voxel point centroids only (VC Only) results in an efficient architecture with a decrease in runtime by 61% and better performance on LEVEL_2 mAPH by +1.22% compared to PV-RCNN on Waymo. VC Only also scales to the large Waymo dataset, with only an increase of 64 ms in runtime from KITTI, showing voxel point centroid localization as an efficient alternative to FPS for spatially locating voxel features. PDV overcomes the limitations of voxel-based methods when scaling to larger datasets, while retaining computational efficiency.

Method	Waymo		KITTI	
	Speed (ms)	mAPH	Speed (ms)	3D mAP
PV-RCNN* [23]	396	58.14	142	71.93
Voxel-RCNN* [4]	**91**	57.47	**74**	72.97
VC Only (Ours)	154	59.36	90	72.82
PDV (Ours)	340	**60.56**	135	**73.44**

Table 8. Runtime comparison and performance on the Waymo and KITTI validation sets. We average the LEVEL_2 3D mAPH for Waymo and moderate difficulty 3D AP for KITTI across classes. $\star$: Models are trained using the publicly released code [30].

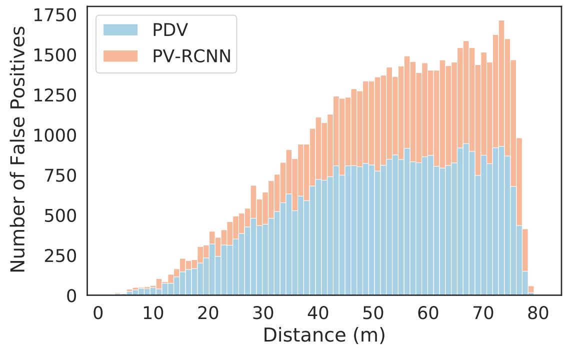

Figure 6. Number of false positive predictions on the vehicle class (IoU < 0.7) across distance for PDV and PV-RCNN evaluated on 10% of the Waymo validation set.

4.5. False Positives across Distance

Figure 6 shows the number of false positives (IoU < 0.7) predicted by PDV and PV-RCNN across distance for vehicles on the Waymo dataset. As the distance from the sensor increases, the gap between PDV and PV-RCNN increases. We attribute the increased differential to using point density to refine bounding box regression and confidence values, which is more beneficial at detecting challenging objects at farther ranges.

5. Conclusion

We present PDV, a novel LiDAR 3D object detection method that uses voxel features and raw point cloud data to account for point density variations in LiDAR point clouds. PDV is particularly useful on large input spaces where point cloud sampling is expensive and voxel resolutions are low, resulting in state-of-the-art performance on the Waymo dataset and competitive results for the KITTI dataset.

References

[1] Prarthana Bhattacharyya, Chengjie Huang, and Krzysztof Czarnecki. Sa-det3d: Self-attention based context-aware 3d object detection. *arXiv preprint*, 2021. 7

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *ECCV*, 2020. 7

[3] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. *NeurIPS*, 2015. 5

[4] Jiajun Deng, Shaoshuai Shi, Peiwei Li, Wengang Zhou, Yanyong Zhang, and Houqiang Li. Voxel r-cnn: Towards high performance voxel-based 3d object detection. *AAAI*, 2021. 1, 2, 6, 7, 8

[5] Runzhou Ge, Zhuangzhuang Ding, Yihan Hu, Yu Wang, Sijia Chen, Li Huang, and Yuan Li. Afdet: Anchor free one stage 3d object detection. *arXiv preprint*, 2020. 6

[6] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. *CVPR*, 2012. 1, 2, 5, 6

[7] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. *CVPR*, 2018. 2, 3

[8] Chenhang He, Hui Zeng, Jianqiang Huang, Xian-Sheng Hua, and Lei Zhang. Structure aware single-stage 3d object detection from point cloud. *CVPR*, 2020. 2, 7

[9] Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics (TOG)*, 2018. 2, 4

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint*, 2014. 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. *CVPR*, 2019. 2, 6, 7

[12] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CVPR*, 2017. 5

[13] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint*, 2016. 6

[14] Jiageng Mao, Minzhe Niu, Haoyue Bai, Xiaodan Liang, Hang Xu, and Chunjing Xu. Pyramid r-cnn: Towards better performance and adaptability for 3d object detection. *ICCV*, 2021. 1, 2, 6, 7

[15] Jiageng Mao, Yujing Xue, Minzhe Niu, Haoyue Bai, Jiashi Feng, Xiaodan Liang, Hang Xu, and Chunjing Xu. Voxel transformer for 3d object detection. *ICCV*, 2021. 1, 2, 6, 7

[16] Xuran Pan, Zhuofan Xia, Shiji Song, Li Erran Li, and Gao Huang. 3d object detection with pointformer. *CVPR*, 2021. 6

[17] Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 1962. 2

[18] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. *CVPR*, 2018. 2

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

[20] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *NeurIPS*, 2017. 2, 4

[21] Murray Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 1956. 2

[22] Hualian Sheng, Sijia Cai, Yuan Liu, Bing Deng, Jianqiang Huang, Xian-Sheng Hua, and Min-Jian Zhao. Improving 3d object detection with channel-wise transformer. *ICCV*, 2021. 2, 6, 7

[23] 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. *CVPR*, 2020. 1, 2, 4, 5, 6, 7, 8

[24] Shaoshuai Shi, Li Jiang, Jiajun Deng, Zhe Wang, Chaoxu Guo, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn++: Point-voxel feature set abstraction with local vector representation for 3d object detection. *arXiv preprint*, 2021. 2, 6, 7

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

[26] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *TPAMI*, 2020. 6

[27] Weijing Shi and Raj Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. *CVPR*, 2020. 2

[28] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint*, 2018. 6

[29] 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. *CVPR*, 2020. 1, 2, 5, 6

[30] OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. https://github.com/open-mmlab/OpenPCDet, 2020. 7, 8

[31] Hugues Thomas, François Goulette, Jean-Emmanuel Deschaud, Beatriz Marcotegui, and Yann LeGall. Semantic classification of 3d point clouds with multiscale spherical neighborhoods. *3DV*, 2018. 3

[32] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. *ICCV*, 2019. 3

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017. 2, 4, 7

[34] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. *CVPR*, 2018. 4

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

[36] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. *CVPR*, 2019. 2, 4

[37] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 2018. 1, 2, 3, 6, 7

[38] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3dssd: Point-based 3d single stage object detector. *CVPR*, 2020. 2, 7

[39] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Std: Sparse-to-dense 3d object detector for point cloud. *ICCV*, 2019. 2, 7

[40] Wu Zheng, Weiliang Tang, Sijin Chen, Li Jiang, and Chi-Wing Fu. Cia-ssd: Confident iou-aware single-stage object detector from point cloud. *AAAI*, 2021. 1, 2

[41] Wu Zheng, Weiliang Tang, Li Jiang, and Chi-Wing Fu. Se-ssd: Self-ensembling single-stage object detector from point cloud. *CVPR*, 2021. 1, 7

[42] 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. *CoRL*, 2020. 6

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