End-to-End Pseudo-LiDAR for Image-Based 3D Object Detection

Rui Qian[*1,2] Divyansh Garg[*1] Yan Wang[*1] Yurong You[*1]
Serge Belongie[1,2] Bharath Hariharan[1] Mark Campbell[1] Kilian Q. Weinberger[1] Wei-Lun Chao[3]
[1] Cornell Univeristy [2] Cornell Tech [3] The Ohio State University
{rq49, dg595, yw763, yy785, sjb344, bh497, mc288, kqw4}@cornell.edu chao.209@osu.edu

Abstract

Reliable and accurate 3D object detection is a necessity for safe autonomous driving. Although LiDAR sensors can provide accurate 3D point cloud estimates of the environment, they are also prohibitively expensive for many settings. Recently, the introduction of pseudo-LiDAR *(PL) has led to a drastic reduction in the accuracy gap between methods based on LiDAR sensors and those based on cheap stereo cameras. PL combines state-of-the-art deep neural networks for 3D depth estimation with those for 3D object detection by converting 2D depth map outputs to 3D point cloud inputs. However, so far these two networks have to be trained separately. In this paper, we introduce a new framework based on differentiable Change of Representation (CoR) modules that allow the entire PL pipeline to be trained* end-to-end*. The resulting framework is compatible with most state-of-the-art networks for both tasks and in combination with PointRCNN improves over PL consistently across all benchmarks — yielding the highest entry on the KITTI image-based 3D object detection leaderboard at the time of submission. Our code will be made available at* https://github.com/mileyan/pseudo-LiDAR_e2e.

1. Introduction

One of the most critical components in autonomous driving is 3D object detection: a self-driving car must accurately detect and localize objects such as cars and pedestrians in order to plan the path safely and avoid collisions. To this end, existing algorithms primarily rely on LiDAR (Light Detection and Ranging) as the input signal, which provides precise 3D point clouds of the surrounding environment. LiDAR, however, is very expensive. A 64-beam model can easily cost more than the car alone, making self-driving cars prohibitively expensive for the general public.

One solution is to explore alternative sensors like commodity (stereo) cameras. Although there is still a noticeable gap to LiDAR, it is an area with exceptional progress in the past year [14, 20, 29, 36, 46, 45]. For example, pseudo-LiDAR (PL) [36, 45] converts a depth map estimated from stereo images into a 3D point cloud, followed by applying (any) existing LiDAR-based detectors. Taking advantage of the state-of-the-art algorithms from both ends [2, 15, 30, 34, 45], pseudo-LiDAR achieves the highest image-based 3D detection accuracy (34.1% and 42.4% at the moderate case) on the KITTI leaderboard [10, 11].

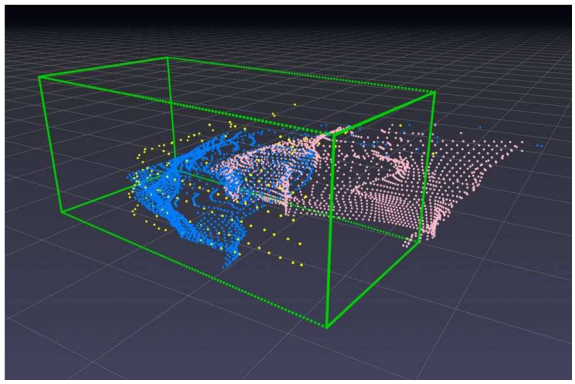

Figure 1: **An illustration of the effectiveness of our end-to-end pipeline.** The green bounding box is the ground truth detection of a car. The yellow points are points from LiDAR. The pink point cloud is generated from an independently trained depth estimator, which is inaccurate and lies out of the green box. By making depth estimation and 3D object detection end-to-end, we obtain a better blue point cloud. Upon this, the object detector could yield the state-of-the-art performance.

While the modularity of pseudo-LiDAR is conceptual appealing, the combination of two independently trained components can yield an undesired performance hit. In particular, pseudo-LiDAR requires two systems: a depth estimator, typically trained on a generic depth estimation (stereo) image corpus, and an object detector trained on the point cloud data converted from the resulting depth estimates. It is unlikely that the two training objectives are optimally aligned for the ultimate goal, to maximize final detection accuracy. For example, depth estimators are typi-

[*] Equal contributions

cally trained with a loss that penalizes errors across all pixels equally, instead of focusing on objects of interest. Consequently, it may over-emphasize nearby or non-object pixels as they are over-represented in the data. Further, if the depth network is trained to estimate disparity, its intrinsic error will be exacerbated for far-away objects [45].

To address these issues, we propose to design a 3D object detection framework that is trained *end-to-end*, while preserving the modularity and compatibility of pseudo-LiDAR with newly developed depth estimation and object detection algorithms. To enable back-propagation based end-to-end training on the final loss, the *change of representation (CoR)* between the depth estimator and the object detector must be differentiable with respect to the estimated depth. We focus on two types of CoR modules — subsampling and quantization — which are compatible with different LiDAR-based object detector types. We study in detail on how to enable effective back-propagation with each module. Specifically, for quantization, we introduce a novel differentiable soft quantization CoR module to overcome its inherent non-differentiability. The resulting framework is readily compatible with most existing (and hopefully future) LiDAR-based detectors and 3D depth estimators.

We validate our proposed end-to-end pseudo-LiDAR (E2E-PL) approach with two representative object detectors — PIXOR [43] (quantized input) and PointRCNN [34] (subsampled point input) — on the widely-used KITTI object detection dataset [10, 11]. Our results are promising: we improve over the baseline pseudo-LiDAR pipeline and the improved PL++ pipeline [45] in all the evaluation settings and significantly outperform other image-based 3D object detectors. At the time of submission our E2E-PL with PointRCNN holds the best results on the KITTI image-based 3D object detection leaderboard. Our qualitative results further confirm that end-to-end training can effectively guide the depth estimator to refine its estimates around object boundaries, which are crucial for accurately localizing objects (see Figure 1 for an illustration).

2. Related Work

3D Object Detection. Most works on 3D object detection are based on 3D LiDAR point clouds [7, 8, 9, 16, 17, 19, 26, 34, 35, 41, 42, 44]. Among these, there are two streams in terms of point cloud processing: 1) directly operating on the unordered point clouds in 3D [16, 30, 34, 47], mostly by applying PointNet [31, 32] or/and applying 3D convolution over neighbors; 2) operating on quantized 3D/4D tensor data, which are generated from discretizing the locations of point clouds into some fixed grids [6, 15, 22, 43]. Images can be included in both types of approaches, but primarily to supplement LiDAR signal [6, 8, 15, 21, 22, 25, 30, 40].

Besides LiDAR-based models, there are solely image-based models, which are mostly developed from the 2D frontal-view detection pipeline [12, 23, 33], but most of them are no longer competitive with the state of the art in localizing objects in 3D [1, 4, 5, 3, 27, 18, 28, 37, 38, 39].

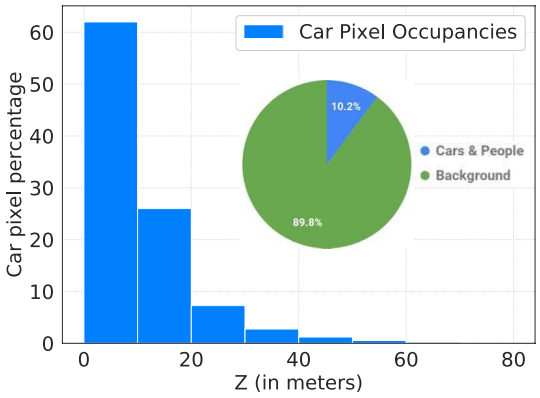

Figure 2: **Pixel distribution:** 90% of all pixels correspond to background. The 10% pixels associated with cars and people (< 1% people) are primarily within a depth of 20m.

Pseudo-LiDAR. This gap has been greatly reduced by the recently proposed pseudo-LiDAR framework [36, 45]. Different from previous image-based 3D object detection models, pseudo-LiDAR first utilizes an image-based depth estimation model to obtain predicted depth $Z(u, v)$ of each image pixel (u, v). The resulting depth $Z(u, v)$ is then projected to a "pseudo-LiDAR" point (x, y, z) in 3D by

$$z = Z(u, v), \quad x = \frac{(u - c_U) \cdot z}{f_U}, \quad y = \frac{(v - c_V) \cdot z}{f_V}, \quad (1)$$

where (c_U, c_V) is the camera center and f_U and f_V are the horizontal and vertical focal lengths. The "pseudo-LiDAR" points are then treated as if they were LiDAR signals, over which any LiDAR-based 3D object detector can be applied. By making use of the separately trained state-of-the-art algorithms from both ends [2, 15, 30, 34, 45], pseudo-LiDAR achieved the highest image-based performance on KITTI benchmark [10, 11]. Our work builds upon this framework.

3. End-to-End Pseudo-LiDAR

One key advantage of the pseudo-LiDAR pipeline [36, 45] is its *plug-and-play* modularity, which allows it to incorporate any advances in 3D depth estimation or LiDAR-based 3D object detection. However, it also lacks the notion of end-to-end training of both components to ultimately maximize the detection accuracy. In particular, the pseudo-LiDAR pipeline is trained in two steps, with different objectives. First, a depth estimator is learned to estimate generic depths for all pixels in a stereo image; then a LiDAR-based detector is trained to predict object bounding boxes from depth estimates, generated by the frozen depth network.

As mentioned in section 1, learning pseudo-LiDAR in this fashion does not align the two components well. On one

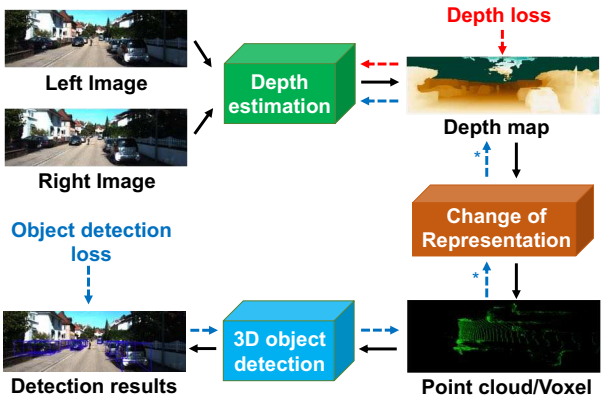

Figure 3: **End-to-end image-based 3D object detection:** We introduce a *change of representation (CoR)* layer to connect the output of the depth estimation network as the input to the 3D object detection network. The result is an end-to-end pipeline that yields object bounding boxes directly from stereo images and allows back-propagation throughout all layers. Black solid arrows represent the forward pass; Blue and red dashed arrows represent the backward pass for the object detection loss and depth loss, respectively. The * denotes that our *CoR* layer is able to back propogate the gradients between different representations.

end, a LiDAR-based object detector heavily relies on accurate 3D points on or in the proximity of the object surfaces to detect and localize objects. Especially, for far-away objects that are rendered by relatively few points. On the other end, a depth estimator learned to predict all the pixel depths may place over-emphasis on the background and nearby objects since they occupy most of the pixels in an image. For example, in the KITTI dataset [13] only about 10% of all pixels correspond to cars and pedestrians/cyclists (Figure 2). Such a misalignment is aggravated with fixing the depth estimator in training the object detector: the object detector is unaware of the intrinsic depth error in the input and thus can hardly detect the far-away objects correctly.

Figure 3 illustrates our proposed end-to-end pipeline to resolve these shortcomings. Here, the error signal from misdetecting or mislocalizing an object can "softly attend" to pixels which affect the prediction most (likely those on or around objects in 2D), instructing the depth estimator where to improve for the subsequent detector. To enable back-propagating the error signal from the final detection loss, the *change of representation (CoR)* between the depth estimator and the object detector must be differentiable with respect to the estimated depth. In the following, we identify two major types of CoR — subsampling and quantization — in incorporating existing LiDAR-based detectors into the pseudo-LiDAR pipeline.

3.1. Quantization

Several LiDAR-based object detectors take voxelized 3D or 4D tensors as inputs [6, 15, 22, 43]. The 3D point locations are discretized into a fixed grid, and only the occupation (*i.e.*, $\{0, 1\}$) or densities (i.e., $[0, 1]$) are recorded in the resulting tensor[1]. The advantage of this kind of approaches is that 2D and 3D convolutions can be directly applied to extract features from the tensor. Such a discretization process, however, makes the back-propagation difficult.

Let us consider an example where we are given a point cloud $\boldsymbol{P} = \{\boldsymbol{p}_1, \dots, \boldsymbol{p}_N\}$ with the goal to generate a 3D occupation tensor $\boldsymbol{T}$ of M bins, where each bin $m \in \{1, \cdots, M\}$ is associated with a fixed center location $\hat{\boldsymbol{p}}_m$. The resulting tensor T is defined as follows,

$$\boldsymbol{T}(m) = \begin{cases} 1, & \text{if } \exists\, \boldsymbol{p} \in \boldsymbol{P} \text{ s.t. } m = \underset{m'}{\operatorname{argmin}} \|\boldsymbol{p} - \hat{\boldsymbol{p}}_{m'}\|_2 \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

In other words, if a point $\boldsymbol{p} \in \boldsymbol{P}$ falls into bin m, then $\boldsymbol{T}(m) = 1$; otherwise, 0. The forward pass of generating $\boldsymbol{T}$ is as straightforward. The backward pass to obtain the gradient signal of the detection loss $\mathcal{L}_{\text{det}}$ with respect to $\boldsymbol{p} \in \boldsymbol{P}$ or the depth map Z (Equation 1), however, is non-trivial.

Concretely, we can obtain $\nabla_{\boldsymbol{T}} \mathcal{L}_{\text{det}}$ by taking the gradients of $\mathcal{L}_{\text{det}}$ with respect to $\boldsymbol{T}$. Intuitively, if $\frac{\partial \mathcal{L}_{\text{det}}}{\partial \boldsymbol{T}(m)} < 0$, it means that $\boldsymbol{T}(m)$ should increase; *i.e.*, there should be points falling into bin m. In contrast, if $\frac{\partial \mathcal{L}_{\text{det}}}{\partial \boldsymbol{T}(m)} > 0$, it means that $\boldsymbol{T}(m)$ should decrease by pushing points out from bin m. But how can we pass these messages back to the input point cloud $\boldsymbol{P}$? More specifically, how can we translate the single digit $\frac{\partial \mathcal{L}_{\text{det}}}{\partial \boldsymbol{T}(m)}$ of each bin to be useful information in 3D in order to adjust the point cloud $\boldsymbol{P}$?

As a remedy, we propose to modify the forward pass by introducing a differentiable soft quantization module (see Figure 4). We introduce a radial basis function (RBF) around the center $\hat{\boldsymbol{p}}_m$ of a given bin m. Instead of binary occupancy counters[2], we keep a "soft" count of the points inside the bin, weighted by the RBF. Further, we allow any given bin m to be influenced by a local neighborhood $\mathcal{N}_m$ of close bins. We then modify the definition of $\boldsymbol{T}$ accordingly. Let $\boldsymbol{P}_m$ denote the set of points that fall into bin m,

$$\boldsymbol{P}_m = \{\boldsymbol{p} \in \boldsymbol{P}, \text{s.t. } m = \underset{m'}{\operatorname{argmin}} \|\boldsymbol{p} - \hat{\boldsymbol{p}}_{m'}\|_2\}.$$

We define $\boldsymbol{T}(m, m')$ to denote the average RBF weight of points in bin m' w.r.t. bin m (more specifically, $\hat{\boldsymbol{p}}_m$),

$$\boldsymbol{T}(m, m') = \begin{cases} 0 & \text{if } |P_{m'}| = 0; \\ \frac{1}{|\boldsymbol{P}_{m'}|} \sum\limits_{\boldsymbol{p} \in \boldsymbol{P}_{m'}} e^{-\frac{\|\boldsymbol{p} - \hat{\boldsymbol{p}}_m\|^2}{\sigma^2}} & \text{if } |P_{m'}| > 0. \end{cases} \tag{3}$$

The final value of the tensor $\boldsymbol{T}$ at bin m is the combination

[1] For LiDAR data the reflection intensity is often also recorded.

[2] We note that the issue of back-propagation cannot be resolved simply by computing real-value densities in Equation 2.

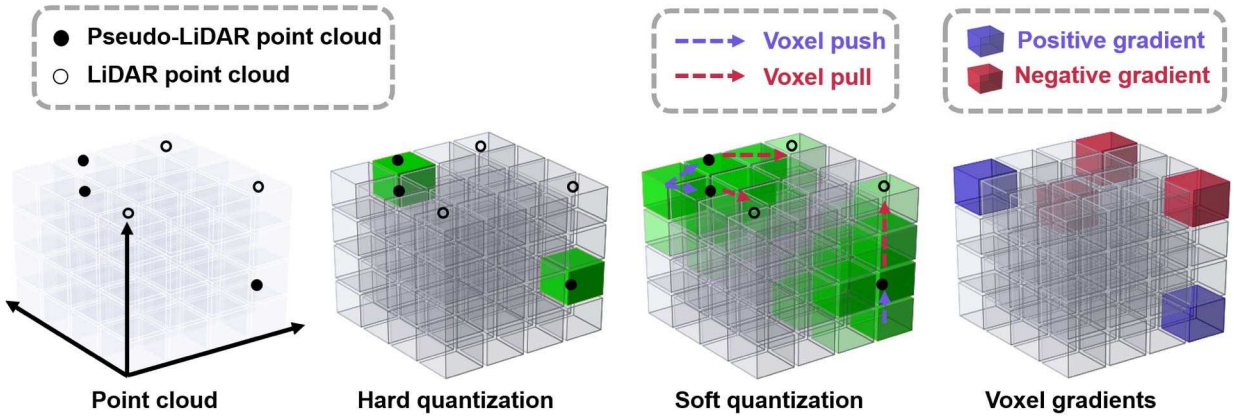

Figure 4: **Quantization:** We voxelize an input pseudo-LiDAR (PL) point cloud using soft or hard quantization. Green voxels are those influenced by the PL points. A blue voxel having a **positive** gradient of the detection loss $\mathcal{L}_{\text{det}}$ exerts a force to push points away from its center to other voxels, whereas a red voxel having a negative **gradient** exerts a force to pull points of other voxels to its center. These forces at the red and blue voxles can only affect PL points if PL points influence those voxels. Soft quantization increases the area of influence of the PL points and therefore the forces, allowing points from other voxels to be pushed away or pulled towards. The updated PL points thus can become closer to the ground truth LiDAR point cloud.

of soft occupation from its own and neighboring bins,

$$\boldsymbol{T}(m) = \boldsymbol{T}(m,m) + \frac{1}{|\mathcal{N}_m|} \sum_{m' \in \mathcal{N}_m} \boldsymbol{T}(m,m'). \quad (4)$$

We note that, when $\sigma^2 \gg 0$ and $\mathcal{N}_m = \varnothing$, Equation 4 recovers Equation 2. Throughout this paper, we set the neighborhood $\mathcal{N}_m$ to the 26 neighboring bins (considering a 3x3x3 cube centered on the bin) and $\sigma^2 = 0.01$. Following [43], we set the total number of bins to $M = 700 \times 800 \times 35$.

Our soft quantization module is fully differentiable. The partial derivative $\frac{\partial \mathcal{L}_{\text{det}}}{\partial \boldsymbol{T}(m)}$ directly affects the points in bin m (*i.e.*, $\boldsymbol{P}_m$) and its neighboring bins and enables end-to-end training. For example, to pass the partial derivative to a point $\boldsymbol{p}$ in bin m', we compute $\frac{\partial \mathcal{L}_{\text{det}}}{\partial \boldsymbol{T}(m)} \times \frac{\partial \boldsymbol{T}(m)}{\partial \boldsymbol{T}(m,m')} \times \nabla_{\boldsymbol{p}} \boldsymbol{T}(m,m')$. More importantly, even when bin m mistakenly contains no point, $\frac{\partial \mathcal{L}_{\text{det}}}{\partial \boldsymbol{T}(m)} > 0$ allows it to drag points from other bins, say bin m', to be closer to $\hat{\boldsymbol{p}}_m$, enabling corrections of the depth error more effectively.

3.2. Subsampling

As an alternative to voxelization, some LiDAR-based object detectors take the raw 3D points as input (either as a whole [34] or by grouping them according to metric locations [16, 41, 47] or potential object locations [30]). For these, we can directly use the 3D point clouds obtained by Equation 1; however, some subsampling is required. Different from voxelization, subsampling is far more amenable to end-to-end training: the points that are filtered out can simply be ignored during the backwards pass; the points that are kept are left untouched.

First, we remove all 3D points higher than the normal heights that LiDAR signals can cover, such as pixels of the sky. Further, we may sparsify the remaining points by subsampling. This second step is optional but suggested in [45] due to the significantly larger amount of points generated from depth maps than LiDAR: on average 300,000 points are in the pseudo-LiDAR signal but 18,000 points are in the LiDAR signal (in the frontal view of the car). Although denser representations can be advantageous in terms of accuracy, they do slow down the object detection network. We apply an angular-based sparsifying method. We define multiple bins in 3D by discretizing the spherical coordinates (r, θ, ϕ). Specifically, we discretize θ (polar angle) and ϕ (azimuthal angle) to mimic the LiDAR beams. We then keep a single 3D point (x, y, z) from those points whose spherical coordinates fall into the same bin. The resulting point cloud therefore mimics true LiDAR points.

In terms of back-propagation, since these 3D object detectors directly process the 3D coordinates (x, y, z) of a point, we can obtain the gradients of the final detection loss $\mathcal{L}_{\text{det}}$ with respect to the coordinates; i.e., $(\frac{\partial \mathcal{L}_{\text{det}}}{\partial x}, \frac{\partial \mathcal{L}_{\text{det}}}{\partial y}, \frac{\partial \mathcal{L}_{\text{det}}}{\partial z})$. As long as we properly record which points are subsampled in the forward pass or how they are grouped, back-propagating the gradients from the object detector to the depth estimates Z (at sparse pixel locations) can be straightforward. Here, we leverage the fact that Equation 1 is differentiable with respect to z. However, due to the high sparsity of gradient information in $\nabla_Z \mathcal{L}_{\text{det}}$, we found that the initial depth loss used to train a conventional depth estimator is required to jointly optimize the depth estimator.

This subsection, together with subsection 3.1, presents a general end-to-end framework applicable to various object detectors. We do not claim this subsection as a technical contribution, but it provides details that makes end-to-end training for point-cloud-based detectors successful.

3.3. Loss

To learn the pseudo-LiDAR framework end-to-end, we replace Equation 2 by Equation 4 for object detectors that take 3D or 4D tensors as input. For object detectors that take raw points as input, no specific modification is needed.

We learn the object detector and the depth estimator jointly with the following loss,

$$\mathcal{L} = \lambda_{\text{det}}\mathcal{L}_{\text{det}} + \lambda_{\text{depth}}\mathcal{L}_{\text{depth}},$$

where $\mathcal{L}_{\text{det}}$ is the loss from 3D object detection and $\mathcal{L}_{\text{depth}}$ is the loss of depth estimation. λ_{depth} and λ_{det} are the corresponding coefficients. The detection loss $\mathcal{L}_{\text{det}}$ is the combination of classification loss and regression loss,

$$\mathcal{L}_{\text{det}} = \lambda_{\text{cls}}\mathcal{L}_{\text{cls}} + \lambda_{\text{reg}}\mathcal{L}_{\text{reg}},$$

in which the classification loss aims to assign correct class (e.g., car) to the detected bounding box; the regression loss aims to refine the size, center, and rotation of the box.

Let Z be the predicted depth and Z^* be the ground truth, we apply the following depth estimation loss

$$\mathcal{L}_{\text{depth}} = \frac{1}{|\boldsymbol{A}|}\sum_{(u,v)\in\boldsymbol{A}} \ell(Z(u,v) - Z^*(u,v)),$$

where $\boldsymbol{A}$ is the set of pixels that have ground truth depth. $\ell(x)$ is the smooth L1 loss defined as

$$\ell(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1; \\ |x| - 0.5, & \text{otherwise.} \end{cases} \tag{5}$$

We find that the depth loss is important as the loss from object detection may only influence parts of the pixels (due to quantization or subsampling). After all, our hope is to make the depth estimates around (far-away) objects more accurately, but not to sacrifice the accuracy of depths on the background and nearby objects[3].

4. Experiments

4.1. Setup

Dataset. We evaluate our end-to-end (E2E-PL) approach on the KITTI object detection benchmark [10, 11], which contains 3,712, 3,769 and 7,518 images for training, validation, and testing. KITTI provides for each image the corresponding 64-beam Velodyne LiDAR point cloud, right image for stereo, and camera calibration matrices.

[3]The depth loss can be seen as a regularizer to keep the output of the depth estimator physically meaningful. We note that, 3D object detectors are designed with an inductive bias: the input is an accurate 3D point cloud. However, with the large capacity of neural networks, training the depth estimator and object detector end-to-end with the detection loss alone can lead to arbitrary representations between them that break the inductive bias but achieve a lower training loss. The resulting model thus will have a much worse testing loss than the one trained together with the depth loss.

Table 1: Statistics of the gradients of different losses on the predicted depth map. Ratio: the percentage of pixels with gradients.

	Depth Loss	P-RCNN Loss	PIXOR Loss
Ratio	3%	4%	70%
Mean	10^{-5}	10^{-3}	10^{-5}
Sum	0.1	10	1

Metric. We focus on 3D and bird's-eye-view (BEV) object detection and report the results on the *validation set*. We focus on the "car" category, following [6, 36, 40]. We report the average precision (AP) with the IoU thresholds at 0.5 and 0.7. We denote AP for the 3D and BEV tasks by AP_{3D} and AP_{BEV}. KITTI defines the easy, moderate, and hard settings, in which objects with 2D box heights smaller than or occlusion/truncation levels larger than certain thresholds are disregarded. The hard (moderate) setting contains all the objects in the moderate and easy (easy) settings.

Baselines. We compare to seven stereo-based 3D object detectors: PSEUDO-LIDAR (PL) [36], PSEUDO-LIDAR ++ (PL++) [45], 3DOP [4], S-RCNN [20], RT3DSTEREO [14], OC-STEREO [29], and MLF-STEREO [39]. *For* PSEUDO-LIDAR ++*, we only compare to its image-only method.*

4.2. Details of our approach

Our end-to-end pipeline has two parts: stereo depth estimation and 3D object detection. In training, we first learn only the stereo depth estimation network to get a depth estimation prior, and then we fix the depth network and use its output to train the 3D object detector from scratch. In the end, we joint train the two parts with balanced loss weights.

Depth estimation. We apply SDN [45] as the backbone to estimate a dense depth map Z. We follow [45] to pre-train SDN on the synthetic Scene Flow dataset [24] and fine-tune it on the 3,712 training images of KITTI. We obtain the depth ground truth Z^* by projecting the corresponding LiDAR points onto images.

Object detection. We apply two LiDAR-based algorithms: PIXOR [43] (voxel-based, with quantization) and PointRCNN (P-RCNN) [34] (point-cloud-based). We use the released code of P-RCNN. We obtain the code of PIXOR from the authors of [45], which has slight modification to include visual information (denoted as PIXOR*).

Joint training. We set the depth estimation and object detection networks trainable, and allow the gradients of the detection loss to back-propagate to the depth network. We study the gradients of the detection and depth losses w.r.t. the predicted depth map Z to determine the hyper-parameters λ_{depth} and λ_{det}. For each loss, we calculate the percentage of pixels on the entire depth map that have gra-

Table 2: **3D object detection results on the KITTI validation set.** We report AP_{BEV} / AP_{3D} (in %) of the **car** category, corresponding to average precision of the bird's-eye view and 3D object detection. We arrange methods according to the input signals: S for stereo images, L for 64-beam LiDAR, M for monocular images. PL stands for PSEUDO-LIDAR. *Results of our end-to-end* PSEUDO-LIDAR *are in blue.* Methods with 64-beam LiDAR are in gray. Best viewed in color.

Detection algo	Input	IoU = 0.5			IoU = 0.7		
		Easy	Moderate	Hard	Easy	Moderate	Hard
3DOP [4]	S	55.0 / 46.0	41.3 / 34.6	34.6 / 30.1	12.6 / 6.6	9.5 / 5.1	7.6 / 4.1
MLF-STEREO [39]	S	-	53.7 / 47.4	-	-	19.5 / 9.8	-
S-RCNN [20]	S	87.1 / 85.8	74.1 / 66.3	58.9 / 57.2	68.5 / 54.1	48.3 / 36.7	41.5 / 31.1
OC-STEREO [29]	S	90.0 / 89.7	80.6 / 80.0	71.1 / 70.3	77.7 / 64.1	66.0 / 48.3	51.2 / 40.4
PL: P-RCNN [36]	S	88.4 / 88.0	76.6 / 73.7	69.0 / 67.8	73.4 / 62.3	56.0 / 44.9	52.7 / 41.6
PL++: P-RCNN [45]	S	89.8 / 89.7	83.8 / 78.6	77.5 / 75.1	82.0 / 67.9	64.0 / 50.1	57.3 / 45.3
E2E-PL: P-RCNN	S	**90.5 / 90.4**	**84.4 / 79.2**	**78.4 / 75.9**	**82.7 / 71.1**	**65.7 / 51.7**	**58.4 / 46.7**
PL: PIXOR⋆ [36]	S	89.0 / -	75.2 / -	67.3 / -	73.9 / -	54.0 / -	46.9 / -
PL++: PIXOR⋆ [45]	S	89.9 / -	78.4 / -	74.7 / -	79.7 / -	61.1 / -	54.5 / -
E2E-PL: PIXOR⋆	S	**94.6 / -**	**84.8 / -**	**77.1/ -**	**80.4 / -**	**64.3 / -**	**56.7 / -**
P-RCNN [34]	L	97.3 / 97.3	89.9 / 89.8	89.4 / 89.3	90.2 / 89.2	87.9 / 78.9	85.5 / 77.9
PIXOR⋆ [43]	L + M	94.2 / -	86.7 / -	86.1 / -	85.2 / -	81.2 / -	76.1 / -

dients. We further collect the mean and sum of the gradients on the depth map during training, as shown in Table 1. The depth loss only influences 3% of the depth map because the ground truth obtained from LiDAR is sparse. The P-RCNN loss, due to subsampling on the dense PL point cloud, can only influence 4% of the depth map. For the PIXOR loss, our soft quantization module could back-propagate the gradients to 70% of the pixels of the depth map. In our experiments, we find that balancing the sums of gradients between the detection and depth losses is crucial in making joint training stable. We carefully set λ_{depth} and λ_{det} to make sure that the sums are on the same scale in the beginning of training. For P-RCNN, we set $\lambda_{\text{depth}} = 1$ and $\lambda_{\text{det}} = 0.01$; for PIXOR, we $\lambda_{\text{depth}} = 1$ and $\lambda_{\text{det}} = 0.1$.

4.3. Results

On KITTI validation set. The main results on KITTI validation set are summarized in Table 2. It can be seen that **1)** the proposed E2E-PL framework consistently improves the object detection performance on both the model using subsampled point inputs (P-RCNN) and that using quantized inputs (PIXOR⋆). **2)** While the quantization-based model (PIXOR⋆) performs worse than the point-cloud-based model (P-RCNN) when they are trained in a non end-to-end manner, end-to-end training greatly reduces the performance gap between these two types of models, especially for IoU at 0.5: the gap between these two models on AP_{BEV} in moderate cases is reduced from 5.4% to -0.4%. As shown in Table 1, on depth maps, the gradients flowing from the loss of the PIXOR⋆ detector are much denser than those from the loss of the P-RCNN detector, suggesting that more gradient information is beneficial. **3)** For IoU at 0.5 under easy and moderate cases, E2E-PL: PIXOR⋆ performs on a par with PIXOR⋆ using LiDAR.

Table 3: **3D object (car) detection results on the KITTI test set.** We compare E2E-PL (blue) with existing results retrieved from the KITTI leaderboard, and report AP_{BEV} / AP_{3D} at IoU=0.7.

Method	Easy	Moderate	Hard
S-RCNN [20]	61.9 / 47.6	41.3 / 30.2	33.4 / 23.7
RT3DSTEREO [14]	58.8 / 29.9	46.8 / 23.3	38.4 / 19.0
OC-STEREO [29]	68.9 / 55.2	51.5 / 37.6	43.0 / 30.3
PL [36]	67.3 / 54.5	45.0 / 34.1	38.4 / 28.3
PL++: P-RCNN [45]	78.3 / 61.1	58.0 / 42.4	51.3 / 37.0
E2E-PL: P-RCNN	**79.6 / 64.8**	**58.8 / 43.9**	**52.1 / 38.1**
PL++:PIXOR⋆ [45]	70.7 / -	48.3 / -	41.0 / -
E2E-PL: PIXOR⋆	**71.9 / -**	**51.7 / -**	**43.3 / -**

On KITTI test set. Table 3 shows the results on KITTI *test* set. We observe the same consistent performance boost by applying our E2E-PL framework on each detector type. At the time of submission, E2E-PL: P-RCNN achieves the state-of-the-art results over image-based models.

4.4. Ablation studies

We conduct ablation studies on the the point-cloud-based pipeline with P-RCNN in Table 4. We divide the pipeline into three sub networks: depth estimation network (Depth), region proposal network (RPN) and Regional-CNN (RCNN). We try various combinations of the sub networks (and their corresponding losses) by setting them to trainable in our final joint training stage. The first row serves as the baseline. In rows two to four, the results indicate that simply training each sub network independently with more iterations does not improve the accuracy. In row five, joint training RPN and RCNN (*i.e.*, P-RCNN) does not have significant improvement, because the point cloud from Depth is not updated and remains noisy. In row six

Table 4: **Ablation studies on the point-cloud-based pipeline with P-RCNN.** We report AP_{BEV} / AP_{3D} (in %) of the **car** category, corresponding to average precision of the bird's-eye view and 3D detection. We divide our pipeline with P-RCNN into three sub networks: Depth, RPN and RCNN. $\surd$ means that we set the sub network trainable and use its corresponding loss in joint training. We note that the gradients of the later sub network would also back-propagate to the previous sub network. For example, if we choose Depth and RPN, the gradients of RPN would also be back-propogated to the Depth network. The best result per column is in blue. Best viewed in color.

			IoU = 0.5			IoU = 0.7		
Depth	RPN	RCNN	Easy	Moderate	Hard	Easy	Moderate	Hard
			89.8/89.7	83.8/78.6	77.5/75.1	82.0/67.9	64.0/50.1	57.3/45.3
$\surd$			89.7/89.5	83.6/78.5	77.4/74.9	82.2/67.8	64.5/50.5	57.4/45.4
	$\surd$		89.3/89.0	83.7/78.3	77.5/75.0	81.1/66.5	63.9/50.0	57.1/45.2
		$\surd$	89.6/89.4	83.9/78.2	77.6/75.2	81.7/68.2	63.4/50.4	57.2/45.9
	$\surd$	$\surd$	90.2/90.1	84.2/78.8	78.0/75.7	81.9/69.1	64.0/51.2	57.7/46.1
$\surd$	$\surd$		89.3/89.1	83.9/78.5	77.7/75.2	81.3/69.4	64.7/50.7	57.7/45.7
$\surd$		$\surd$	89.8/89.7	84.2/79.1	78.2/76.5	84.2/69.9	65.5/51.0	58.1/46.2
$\surd$	$\surd$	$\surd$	90.5/90.4	84.4/79.2	78.4/75.9	82.7/71.1	65.7/51.7	58.4/46.7

we jointly train Depth with RPN, but the result does not improve much either. We suspect that the loss on RPN is insufficient to guide the refinement of depth estimation. By combining the three sub networks together and using the RCNN, RPN, and Depth losses to refine the three sub networks, we get the best results (except for two cases).

For the quantization-based pipeline with soft quantization, we also conduct similar ablation studies, as shown in Table 5. Since PIXOR* is a one-stage detector, we divide the pipeline into two components: Depth and Detector. Similar to the point-cloud-based pipeline (Table 4), simply training each component independently with more iterations does not improve. However, when we jointly train both components, we see a significant improvement (last row). This demonstrates the effectiveness of our soft quantization module which can back-propagate the Detector loss to influence 70% pixels on the predicted depth map.

More interestingly, applying the *soft quantization* module alone without jointly training (rows one and two) does not improve over or is even outperformed by PL++: PIXOR* with *hard quantization* (whose result is 89.9/79.7|78.4/61.1|74.7/54.5, from Table 2). But with joint end-to-end training enabled by soft quantization, our E2E-PL:PIXOR* consistently outperforms the separately trained PL++: PIXOR*.

4.5. Qualitative results

We show the qualitative results of the point-cloud-based and quantization-based pipelines.

Depth visualization. We visualize the predicted depth maps and the corresponding point clouds converted from the depth maps by the quantization-based pipeline in Figure 5. For the original depth network shown in the first row, since the ground truth is very sparse, the depth prediction is not accurate and we observe clear misestimation (artifact) on the top of the car: there are massive misestimated 3D points on the top of the car. By applying end-to-end joint training, the detection loss on cars enforces the depth network to reduce the misestimation and guides it to generate more accurate point clouds. As shown in the second row, both the depth prediction quality and the point cloud quality are greatly improved. The last row is the input image

Table 5: **Ablation studies on the quantization-based pipeline with PIXOR*.** We report AP_{BEV} at IoU $=$ 0.5 / 0.7 (in %) of the **car** category. We divide our pipeline into two sub networks: Depth and Detector. $\surd$ means we set the sub network trainable and use its corresponding loss in join training. The best result per column is in blue. Best viewed in color.

Depth	Detector	Easy	Moderate	Hard
		89.8 / 77.0	78.3 / 57.7	69.5 / 53.8
$\surd$		89.9 / 76.9	78.7 / 58.0	69.7 / 53.9
	$\surd$	90.2 / 78.1	79.2 / 58.9	69.6 / 54.2
$\surd$	$\surd$	94.6 / 80.4	84.8 / 64.3	77.1 / 56.7

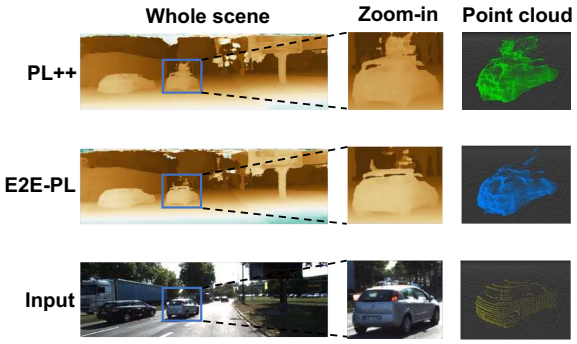

Figure 5: **Qualitative results on depth estimation.** PL++ (image-only) has many misestimated pixels on the top of the car. By applying end-to-end training, the depth estimation around the cars is improved and the corresponding pseudo-LiDAR point cloud has much better quality. (Please zoom-in for the better view.)

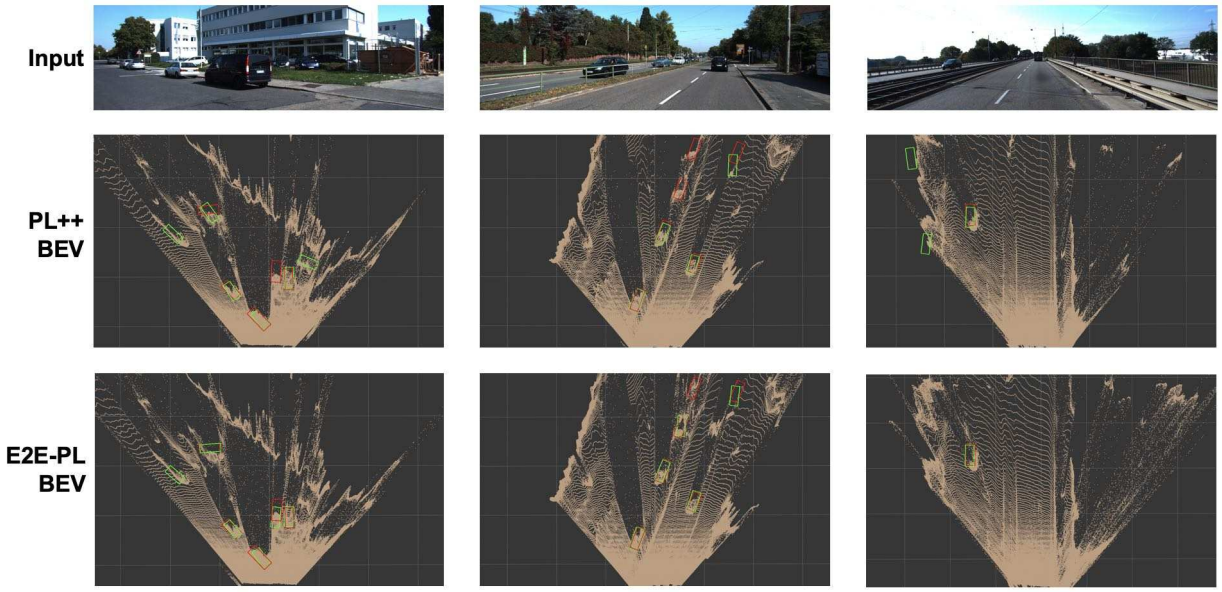

Figure 6: **Qualitative results from the bird's-eye view.** The red bounding boxes are the ground truth and the green bounding boxes are the detection results. PL++ (image-only) misses many far-away cars and has poor bounding box localization. By applying end-to-end training, we get much accurate predictions (first and second columns) and reduce the false positive predictions (the third column).

to the depth network, the zoomed-in patch of a specific car, and its corresponding LiDAR ground truth point cloud.

Detection visualization. We also show the qualitative comparisons of the detection results, as illustrated in Figure 6. From the BEV, the ground truth bounding boxes are marked in red and the predictions are marked in green. In the first example (column), pseudo-LiDAR++ (PL++) misses one car in the middle, and gives poor localization of the far-away cars. Our E2E-PL could detect all the cars, and gives accurate predictions on the farthest car. For the second example, the result is consistent for the far-away cars where E2E-PL detects more cars and localize them more accurately. Even for the nearby cars, E2E-PL gives better results. The third example indicates a case where there is only one ground truth car. Our E2E-PL does not have any false positive predictions.

4.6. Other results

Speed. The inference time of our method is similar to pseudo-LiDAR and is determined by the stereo and detection networks. The soft quantization module (with 26 neighboring bins) only computes the RBF weights between a point and 27 bins (roughly $N = 300,000$ points per scene), followed by grouping the weights of points into bins. The complexity is $O(N)$, and both steps can be parallelized. Using a single GPU with PyTorch implementation, E2E-PL: P-RCNN takes 0.49s/frame and E2E-PL: PIXOR* takes 0.55s/frame, within which SDN (stereo network) takes 0.39s/frame and deserves further study to speed it up (e.g., by code optimization, network pruning, etc).

Others. See the Supplementary Material for more details and results.

5. Conclusion and Discussion

In this paper, we introduced an end-to-end training framework for pseudo-LiDAR [36, 45]. Our proposed framework can work for 3D object detectors taking either the direct point cloud inputs or quantized structured inputs. The resulting models set a new state of the art in image based 3D object detection and further narrow the remaining accuracy gap between stereo and LiDAR based sensors. Although it will probably always be beneficial to include active sensors like LiDARs in addition to passive cameras [45], it seems possible that the benefit may soon be too small to justify large expenses. Considering the KITTI benchmark, it is worth noting that the stereo pictures are of relatively low resolution and only few images contain (labeled) far away objects. It is quite plausible that higher resolution images with a higher ratio of far away cars would result in further detection improvements, especially in the hard (far away and heavily occluded) category.

Acknowledgments

This research is supported by grants from the National Science Foundation NSF (III-1618134, III-1526012, IIS-1149882, IIS-1724282, and TRIPODS-1740822), the Office of Naval Research DOD (N00014-17-1-2175), the Bill and Melinda Gates Foundation, and the Cornell Center for Materials Research with funding from the NSF MRSEC program (DMR-1719875). We are thankful for generous support by Zillow and SAP America Inc.

References

[1] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Céline Teulière, and Thierry Chateau. Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. In *CVPR*, 2017. 2

[2] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *CVPR*, 2018. 1, 2

[3] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *CVPR*, 2016. 2

[4] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *NIPS*, 2015. 2, 5, 6

[5] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals using stereo imagery for accurate object class detection. *TPAMI*. 2

[6] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, 2017. 2, 3, 5

[7] Yilun Chen, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Fast point r-cnn. In *ICCV*, 2019. 2

[8] Xinxin Du, Marcelo H Ang Jr, Sertac Karaman, and Daniela Rus. A general pipeline for 3d detection of vehicles. In *ICRA*, 2018. 2

[9] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *ICRA*, 2017. 2

[10] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 1, 2, 5

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

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. 2

[13] Jinyong Jeong, Younggun Cho, Young-Sik Shin, Hyunchul Roh, and Ayoung Kim. Complex urban dataset with multi-level sensors from highly diverse urban environments. *The International Journal of Robotics Research*, 38(6):642–657, 2019. 3

[14] Hendrik Königshof, Niels Ole Salscheider, and Christoph Stiller. Realtime 3d object detection for automated driving using stereo vision and semantic information. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019. 1, 5, 6

[15] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. Joint 3d proposal generation and object detection from view aggregation. In *IROS*, 2018. 1, 2, 3

[16] 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. 2, 4

[17] Bo Li. 3d fully convolutional network for vehicle detection in point cloud. In *IROS*, 2017. 2

[18] Buyu Li, Wanli Ouyang, Lu Sheng, Xingyu Zeng, and Xiaogang Wang. Gs3d: An efficient 3d object detection framework for autonomous driving. In *CVPR*, 2019. 2

[19] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3d lidar using fully convolutional network. In *Robotics: Science and Systems*, 2016. 2

[20] Peiliang Li, Xiaozhi Chen, and Shaojie Shen. Stereo r-cnn based 3d object detection for autonomous driving. In *CVPR*, 2019. 1, 5, 6

[21] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *CVPR*, 2019. 2

[22] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *ECCV*, 2018. 2, 3

[23] Tsung-Yi Lin, Piotr Dollár, Ross B Girshick, Kaiming He, Bharath Hariharan, and Serge J Belongie. Feature pyramid networks for object detection. In *CVPR*, volume 1, page 4, 2017. 2

[24] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *CVPR*, 2016. 5

[25] Gregory P Meyer, Jake Charland, Darshan Hegde, Ankit Laddha, and Carlos Vallespi-Gonzalez. Sensor fusion for joint 3d object detection and semantic segmentation. In *CVPRW*, 2019. 2

[26] 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. 2

[27] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Košecká. 3d bounding box estimation using deep learning and geometry. In *CVPR*, 2017. 2

[28] Cuong Cao Pham and Jae Wook Jeon. Robust object proposals re-ranking for object detection in autonomous driving using convolutional neural networks. *Signal Processing: Image Communication*, 53:110–122, 2017. 2

[29] Alex D Pon, Jason Ku, Chengyao Li, and Steven L Waslander. Object-centric stereo matching for 3d object detection. In *ICRA*, 2020. 1, 5, 6

[30] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *CVPR*, 2018. 1, 2, 4

[31] 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. 2

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

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 2

[34] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *CVPR*, 2019. 1, 2, 4, 5, 6

[35] 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. 2

[36] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *CVPR*, 2019. 1, 2, 5, 6, 8

[37] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Data-driven 3d voxel patterns for object category recognition. In *CVPR*, 2015. 2

[38] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Subcategory-aware convolutional neural networks for object proposals and detection. In *WACV*, 2017. 2

[39] Bin Xu and Zhenzhong Chen. Multi-level fusion based 3d object detection from monocular images. In *CVPR*, 2018. 2, 5, 6

[40] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. Pointfusion: Deep sensor fusion for 3d bounding box estimation. In *CVPR*, 2018. 2, 5

[41] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 2, 4

[42] Bin Yang, Ming Liang, and Raquel Urtasun. Hdnet: Exploiting hd maps for 3d object detection. In *CoRL*, 2018. 2

[43] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *CVPR*, 2018. 2, 3, 4, 5, 6

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

[45] Yurong You, Yan Wang, Wei-Lun Chao, Divyansh Garg, Geoff Pleiss, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving. In *ICLR*, 2020. 1, 2, 4, 5, 6, 8

[46] X. Ye X. Tan W. Yang S. Wen E. Ding A. Meng L. Huang Z. Xu, W. Zhang. Zoomnet: Part-aware adaptive zooming neural network for 3d object detection. In *AAAI*, 2020. 1

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