RAD: Realtime and Accurate 3D Object Detection on Embedded Systems

Hamed H. Aghdam[1], Elnaz J. Heravi[2], Selameab S. Demilew[1], Robert Laganiere[1,2]
[1] University of Ottawa, [2] Sensorcortek
Ottawa, Canada
h.aghdam@uottawa.ca, elena@sensorcortek.ai, sdemi032@uottawa.ca, laganiere@uottawa.ca

Abstract

To our knowledge, the fastest 3D object detector on LiDAR data works at 42.03 *point clouds-per-second on high-end machines and* 6.15 *point clouds-per-second on embedded boards. We propose a deep 3D object detector with higher detection accuracy running at* 84.46 *point clouds-per-second on high-end machines and* 10.91 *point clouds-per-second on computing boards that is* 2 *and* 1.77 *times faster compared to fastest published network. We achieve considerably higher processing rate without reducing the complexity of the network but by designing a more efficient decoder. Our extensive and practical experiments reveal that the detection accuracy of our proposed network is comparable to the best-performed method using practical metrics but it is* 3.36 *times faster. Besides, we carefully analyze the model and indicate that negligible error in two regression outputs contributes to the reduction in the average precision. Overall, considering the accuracy and speed, our proposed network is highly practical to be executed on embedded boards for ADAS applications.*

1. Introduction

There is still a long road ahead to make driverless cars affordable and safely usable worldwide. Regardless, the number of active vehicles in the world grows constantly every day. Hence, improving the safety of modern cars is crucial to reduce the number of car accidents and injuries. With the advent of cost-effective LiDAR sensors, there is a growing demand for integrating LiDARs on cars (*eg.* on bumpers) to perform various tasks such as detecting vulnerable road users and cars. In contrast to autonomous vehicles, regular cars are usually equipped with low-end embedded systems with limited computational power. To make this technology affordable for a wide range of users, processing LiDAR point clouds has to be done on low-end embedded systems. This requires methods that are *accurate* and *computationally highly efficient*.

Raw LiDAR point clouds do not form a grid, and points

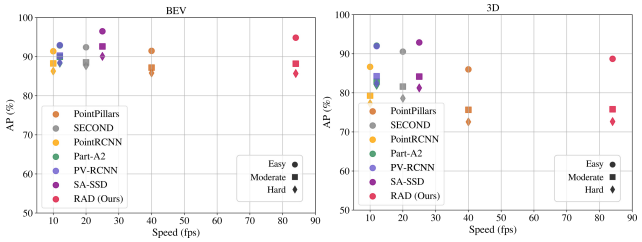

Figure 1. Speed *vs.* average precision for different methods.

are not uniformly distributed in the cloud. For this reason, most state-of-the-art 3D detection networks apply quantization techniques and convert raw point clouds to 3D grids. For instance, VoxelNet [23], SA-SSD [5] and SECOND [20] create a 3D grid of voxels where each voxel may contain zero or more points. Likewise, PointPillars [7] creates a 3D grid of voxels in which each voxel covers the entire vertical axis and contains up to 100 3D points. These kinds of voxel are called pillars.

Even though SA-SSD has outperformed other methods on the KITTI dataset [4], it has major practical drawbacks. Specifically, first layers in the network apply 3D *sparse* convolutions that makes its execution time dependant on the density of the point cloud. If a car uses multiple LiDAR sensors or a LiDAR sensor with 128 channels, the time-to-completion of this method will increase several times due to processing denser point clouds. Likewise, the pillar representation in PointPillars is limited by the number of pillars and the number points in each pillar. Moreover, pillars are only generated from the points that are within the field-of-view (FOV) of the camera. Although 12K pillars [7] suffice to cover the data inside FOV adequately on KITTI, it will become less accurate on larger FOVs and denser point clouds. PointPillars is the fastest published 3D object detector where it can process 40 point clouds-per-second (pps) on a high-end GPU. Despite that, both of the above methods will potentially fail to detect objects using embedded systems in real-time on denser and larger point clouds.

We argue that the processing time of 3D detection networks must be independent of the density of the point cloud

to increase its scalability. PIXOR [22] and PIXOR++[21] have previously used 3D occupancy grids that are faster to compute compared to voxels and pillars. The main advantage of these methods is that they are independent of the density of the point cloud, but compared to PointPillars and SA-SSD, their performances are significantly lower.

Contribution: In this paper, we propose a fast 3D detection network that utilizes 3D occupancy grids to encode point clouds, but not only it is accurate, it is also the fastest published 3D detection network. More importantly, we do not improve the execution time by reducing the expressive power of the neural network. Instead, we utilize an input encoding that is multiple times faster to compute and transfer, and propose a more computationally efficient decoder for the detection network. Our main gain in the execution time is primarily due to these three factors and not by designing a smaller network. Furthermore, we show that the performance of our network is comparable to more computational approaches when we evaluate them using a *practical* threshold for IoU. As we will explain in our experiments, the differences of predictions whose IoU are more than 0.7, and the ones whose IoU are in $[0.5, 0.7]$ are negligible in practical applications. Last but not least, we assess our model extensively from different perspectives and carefully analyze the error and demonstrate that the central source of the error in most models boils down to two factors, and improving these two factors will improve the results.

We start by explaining the state-of-the-art in Section 2. The proposed network and fast input representation will be explained in Section 3, and we will analyze the performance of our method in Section 4. Finally, Section 5 concludes the paper and provides a guideline for future work.

2. Related Work

From point clouds: MV3D [2] and AVOD [6] generate range-view (RV) and bird's eye view (BEV) projections that encode point cloud density, height and intensity. PIXOR [22] voxelizes the input using a 3D binary occupancy grid where a cell is one if it contains at least one point. VoxelNet [23] and SECOND [20] divide the space into a grid of voxels in which each voxel contains the same number of points and apply a mini PointNet [14] called Voxel Feature Encoding layer to each voxel and map them into a higher dimension. Next, they apply a set of 2D and 3D convolutions to classify and regress 3D bounding boxes. PointPillars [7] improves the runtime of [20] by removing the 3D convolutions, using a smaller PointNet encoder and a single bin along the height axis in its voxel representation. Similarly, Patch Refinement [8] stacks two networks where the second network refines predictions by fusing the information from the first network with higher resolution voxels. As it turns out, this method is not computationally efficient since it requires encoding and processing two different voxel-based representations. SA-SSD [5] surpassed other methods by utilizing a smaller grid size and 3D sparse convolutions.

Point-based 3D object detectors work on points rather than 3D girds. Point-RCNN [15] classifies all the points as foreground or background. Next, it crops foreground area along with corresponding features to classify and refine 3D bounding boxes. Fast Point R-CNN[3] replaces the proposal generation stage with a faster VoxelRPN that resembles [23] and [12] adopts a sampling strategy for proposing candidate regions. In general, two-stage object detection networks may have better performance, while single-stage detectors offer higher frame rates. Currently, the single-stage network PointPillars [7] reports the shortest runtime at $16ms$ on a high-end machine.

Multimodal 3D object detection: Several attempts have been made to fuse point cloud and images. [10, 19, 9] run two concurrent streams for each modality and perform a late-fusion at the feature level. [13, 18] use PointNet to regress boxes on point cloud crops generated from 2D image proposals. PointPainting [17] also uses a sequential approach of decorating points with a class vector that is computed using an image-only semantic segmentation network.

Discussion: Two-stage detection networks are not computationally efficient and cannot detect 3D objects in real-time on embedded devices. Besides, any method that depends on the density of the point cloud will not scale well with newer LiDAR sensors. Last but not least, networks that rely on 3D convolutions are unlikely to satisfy *both* our requirements that are *computational efficiency* and *accuracy*.

3. Proposed Method

We achieve the highest processing rate for 3D object detection reported to date by using a highly efficient input representation and flexible but fast network.

Input encoding: Denoting the i^{th} 3D point in a point cloud using $p_i = (x_i, y_i, z_i)$, our goal is to transform the point cloud $\mathbf{P} = \{p_0, \ldots, p_i, p_{N-1}\}$ into a regular 3D grid representation. Assuming that $\mathbf{P}$ is bounded within (x_{min}, x_{max}), (y_{min}, y_{max}) and (z_{min}, z_{max}) along x, y and z axes respectively, we compute the *sparse occupancy matrix* $\mathbf{M}$ using the quantization factors $(\delta x, \delta y, \delta z)$. All points belonging to the same cuboid in the 3D occupancy grid must have one key in the sparse matrix. To achieve this goal, we utilize dictionary of keys (DOK) representation to create $\mathbf{M}$. Concretely, the p_i updates $\mathbf{M}$ using:

$$\mathbf{M}\Big[\lfloor\frac{x_i}{\delta x}\rfloor, \lfloor\frac{y_i}{\delta y}\rfloor, \lfloor\frac{z_i}{\delta z}\rfloor\Big] = 1 \qquad i = 0 \ldots N-1 \qquad (1)$$

where $\lfloor\rfloor$ is the floor operation and $\Big[\lfloor\frac{x_i}{\delta x}\rfloor, \lfloor\frac{y_i}{\delta y}\rfloor, \lfloor\frac{z_i}{\delta z}\rfloor\Big]$ is the 3D bin index of p_i after quantization and is used as the key for DOK. Instead of using a dictionary, we use a 3D flag tensor to track indices that are already added to $\mathbf{M}$. To optimize the runtime of this encoding further, we share the flag

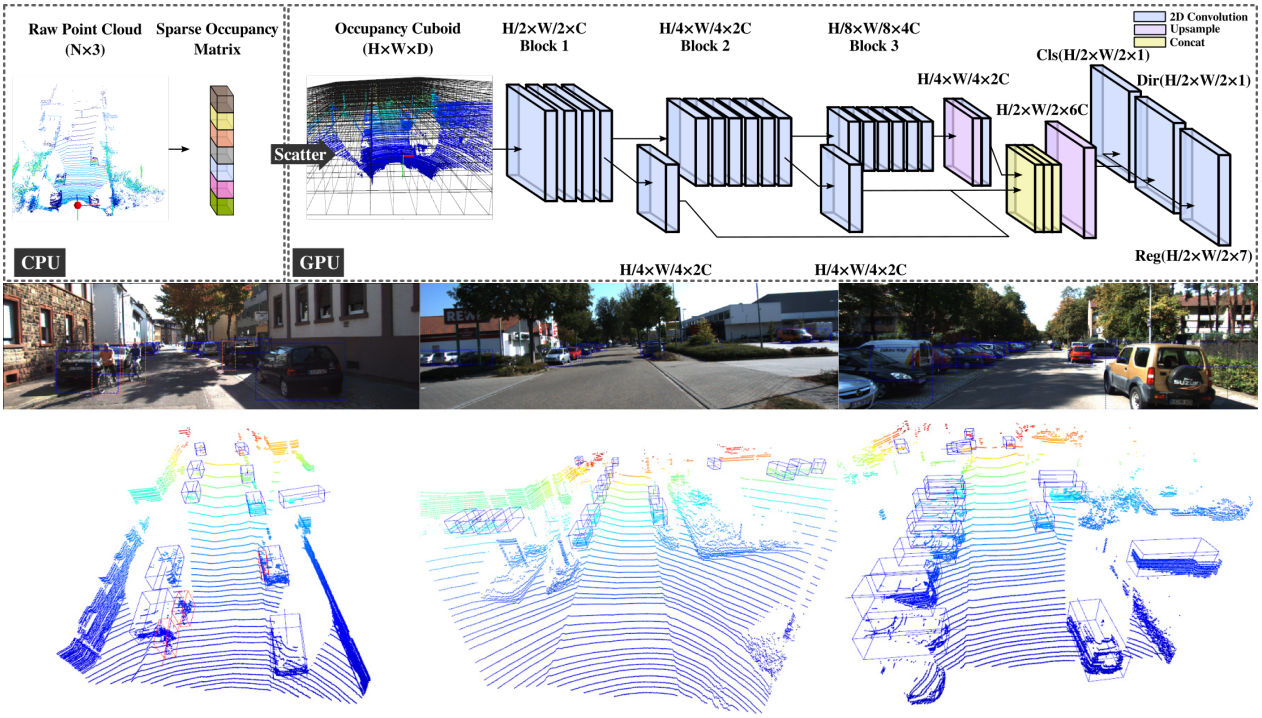

Figure 2. Overview of the proposed 3D detection network. The stride of the first convolution layer in each block is 2. Best viewed in color.

tensor across various calls to avoid the memory allocation, value initialization, and memory deallocation overhead.

Then, the sparse matrix $\mathbf{M}$ is used as the input of our network. It should be noted that 99.88% of cells are empty (*ie.* zero) in the dense representation of $\mathbf{M}$. In other words, the size of $\mathbf{M}$ is 0.12% of the dense representation. Thus, our sparse representation significantly reduces the amount of data that is transferred from the CPU to the GPU. The sparse matrix $\mathbf{M}$ is scattered into the dense 3D occupancy grid of size $H \times W \times D$ on the GPU. In addition to the computational efficiency of the above representation, it is relatively invariant to LiDAR scanning errors. Assuming $\pm 3cm$ scanning error for a point in the LiDAR point cloud, given sufficiently large quantization factors $(\delta x, \delta y, \delta z)$, the 3D point will potentially fall into the same cell in $\mathbf{M}$ if the scanning is repeated several times which in the sequel will generate identical $\mathbf{M}$ regardless of the scanning error.

Figure 2 shows the architecture of our network to detect 3D objects using the dense 3D occupancy grid. The encoder of the network is composed of three blocks where each block contains 3×3 convolution layers. The stride of the first convolution layer in each block is two, and the stride of the remaining convolutions in the same block is one. We utilize ReLU as the activation function and place batch normalization between each convolution and ReLU activation (*a.k.a* conv-bn-relu pattern). The decoder starts with concatenating feature maps obtained at the end of each block to form the final feature map. To this end, it first applies a 3×3 convolution with stride 2 to the output of Block 1. Next, it applies a 3×3 convolution with stride 1 to the output of Block 2. Then, it upsamples the output of Block 3 using nearest-neighbor interpolation followed by a 3×3 convolution with stride 1. Finally, these three feature maps are concatenated to form the input of the prediction heads.

During training, we upsample the final feature map using *bilinear upsampling* and use the upsampled feature map as the input to the prediction heads. After training, the bilinear upsampling stage can be discarded, and the prediction heads can be directly applied to concatenated feature maps. We will show in the next section that even using the last upsampling in the inference our proposed method is significantly faster than the runner-up network (*ie.* PointPillars).

Similar to [7, 20], our network utilizes one *classification* head for computing objectness score, one head for *regressing* bounding boxes and one *direction* head for predicting if the car is heading forward or backward. Whereas the classification and direction head are binary classifiers, the regression head computes the offsets $(\Delta x, \Delta y, \Delta z)$, the bounding box sizes (l, w, h) and the bounding box orientation θ at location (x, z) on the regression map where:

$$\Delta x = \frac{(x - x_g)}{\sqrt{{l_a}^2 + {w_a}^2}} \quad \Delta y = \frac{(1 - y_g)}{h_a} \quad \Delta z = \frac{(z - z_g)}{\sqrt{{l_a}^2 + {w_a}^2}}$$

$$h = \log \frac{h_g}{h_a} \quad w = \log \frac{w_g}{w_a} \quad l = \log \frac{l_g}{l_a}$$

$$\theta = \theta_g$$

Here, h_a, w_a and l_a denote the average size of the object, x_g, y_g, z_g, h_g, w_g, l_g and θ_g show the values of the ground-truth box. We use weighted cross-entropy $\mathcal{L}_{cls}$ for training the classification head, the cross-entropy $\mathcal{L}_{dir}$ for training the direction head, and the weighted smooth L1 loss $\mathcal{L}_{reg}$ for training the regression head. The total loss is equal to the weighted sum of these losses.

$$\mathcal{L}_{cls} = \frac{1}{N}\sum_{i=1}^{N} CE(c'_i, c_i, w_i), \quad \mathcal{L}_{dir} = \frac{1}{N}\sum_{i=1}^{N} CE(d'_i, d_i)$$

$$\mathcal{L}_{reg} = \frac{1}{N_{pos}} \sum_{p \in pos} \sum SmoothL1(\mathbf{r}'(p) - \mathbf{r}(p))$$

$$\mathcal{L} = \mathcal{L}_{cls} + \alpha\mathcal{L}_{dir} + \beta\mathcal{L}_{reg} + \lambda||\mathbf{W}||_2^2$$

where $\mathbf{r}(p) = (\Delta x, \Delta y, \Delta z, l, w, h, \theta)$, N_{pos} denotes the number of positive samples, $\mathbf{W}$ shows trainable parameters. We set $\alpha = 0.2$, $\beta = 2$ in all our experiments. Also, $w_i = 1$ for background and $w_i = 2$ for object samples.

4. Experiments

We evaluate our proposed method on KITTI[4] dataset. The KITTI dataset contains $7,481$ training samples and $7,518$ testing samples, where labels for test samples are not provided. Out of the $7,481$ entries, we follow the commonly used training-validation split [1, 11] containing $3,712$ training samples and $3,769$ validation samples. Following the discussion in [16] that was later adapted by the official KITTI benchmark, we use 40 samples for computing the average-precision in all our experiments.

3D Occupancy grid: The point cloud is bounded to $(-40, 40)$, $(-1, 3)$ and $(0, 70.4)$ along x (left-right), y (up-down) and z (forward) axes, respectively. Also, the quantization factors $(\delta x, \delta y, \delta z)$ are $(0.16, 0.1, 0.16)$ in all our experiments. The matrix is transposed such that the y axis denotes the channels of the matrix. This will generate a dense occupancy grid of size $500 \times 440 \times 40$. Finally, (h_a, w_a, l_a) in the regression head is fixed to $(1.6, 3.9, 1.56)$ and $(0.6, 1.76, 1.73)$ for cars and cyclists, respectively.

Training: The network is trained using the Adam optimizer for 200 epochs with batch size 2, the initial learning rate $2e{-}4$ that is multiplied with 0.8 every 15 epochs and the regularization coefficient $\lambda = 1e{-}4$.

Augmentation: To mitigate the overfitting problem, we augment the training data using random sampling, global rotation and translation, per box rotation and translation, scaling and flipping [20]. Our random sampling approach is slightly different as it transforms (rotate+translate) each newly added sample with a probability. Also, for each point cloud in the training set, we define 15 different patterns for adding random samples instead of generating a random pattern at each augmentation step.

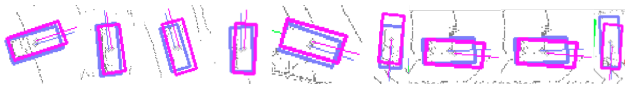

Figure 3. A few samples where the IoU between the predicted sample (magenta) and the ground-truth sample (blue) is in $[0.5, 0.7)$

4.1. Detection Accuracy

Table 1 compares our network with state-of-the-art on the validation set using the average precision computed at *IoU threshold* 0.7. According to the results, our method is slightly more accurate and faster than PointPillars. As we will show, augmentation methods have a significant impact on the results since KITTI is a small dataset. In particular, the y value of newly added random samples by the augmentation method must be corrected according to the ground plane. Without an accurate y value in training samples, the network will not predict y accurately on validation samples leading to a drop on 3D evaluation. We hypothesize that the larger difference in 3D evaluation between our method and SA-SSD is due to this issue. Despite that, our method already overfits on the training set with an average precision close to 100%. That said, it does not suffer from a high bias and provided with better augmentation methods and more data, the results will be potentially improved.

Note that Table 1 uses the IoU threshold 0.7 to compute the average precision. Figure 3 illustrates a few samples where the IoU between the predicted bounding box and the ground-truth box is in the interval $[0.5, 0.7)$. These samples are considered as false-negatives since their IoUs are less than 0.7. As it turns out, the predicted and ground-truth boxes are aligned well and these kinds of errors are negligible in ADAS applications. To be more specific, our last experiments will show that the $\pm 20cm$ prediction error accounts for 4% and 9% reduction in average-precision of BEV and 3D metrics. Note that $\pm 20cm$ error is of low importance for ADAS applications compared to autonomous vehicles since ADAS are not meant to plan precise motions or perform complex maneuvers. Therefore, it is more practical to evaluate the network using the IoU threshold 0.5. Table 2 compares the same methods using this IoU threshold and shows that the gap between our method and SA-SSD reduces substantially on all three metrics using the IoU threshold 0.5 while our method is 336% faster.

Even though the average precision is a commonly used metric for evaluating object detection methods, it might not be the vital metric for evaluating object detectors in practical applications. Practical applications use a fixed confidence threshold for making predictions. Consequently, a more practical metric for comparing two methods is to find their best performed confidence threshold and compare them using only one pair of precision-recall or F1 score at this threshold. The KITTI evaluation kit computes average

Method	FPS	**2D** (car)			**BEV** (car)			**3D** (car)		
		easy	moderate	hard	easy	moderate	hard	easy	moderate	hard
PointPillars [7]	42	94.46	90.30	87.94	91.49	87.21	85.83	86.00	75.66	72.57
SECOND	20	95.63	94.17	91.77	92.41	88.55	87.64	90.54	81.60	78.59
PointRCNN [3]	10	98.72	92.52	90.0	91.39	88.28	86.31	86.65	79.30	77.26
Part-A2	12.5	97.88	93.76	91.76	92.87	89.98	88.34	92.07	82.86	81.93
PV-RCNN	12.5	98.23	94.40	92.25	92.95	90.28	88.50	91.94	84.25	82.41
SA-SSD [5]	25	99.16	95.98	93.45	96.50	92.62	90.08	92.89	84.16	81.26
RAD (Ours)	**84**	96.24	91.94	89.34	94.86	88.21	85.70	88.70	75.80	72.65

Table 1. Average precision (@IoU 0.7) of our network compared to other methods on the validation set.

Method	**2D** (car)			**BEV** (car)			**3D** (car)		
	easy	moderate	hard	easy	moderate	hard	easy	moderate	hard
PointPillars	94.94	94.01	93.40	94.81	93.78	93.12	94.79	93.40	92.33
SECOND	95.79	95.14	94.65	95.68	94.84	94.24	95.66	94.74	94.05
PointRCNN	99.01	95.16	92.72	95.91	94.95	92.80	95.89	94.83	92.68
Part-A2	98.07	95.81	94.41	97.80	93.95	93.90	97.78	93.87	93.74
PV-RCNN	98.45	96.57	96.63	98.22	94.56	94.41	98.21	94.50	94.30
SA-SSD	99.37	96.51	93.97	99.24	96.25	93.70	99.23	96.19	93.64
RAD (Ours)	96.56	94.93	92.57	96.38	94.49	93.86	96.35	94.25	91.95

Table 2. Average precision (@IoU 0.5) of our network compared to other methods on the validation set.

precision for three different categories including easy, moderate and hard. Denoting the F1 score at threshold t for category $c \in \{easy(e), moderate(m), hard(h)\}$ with $F1_t^c$, we compute the mean F1 score $F1_t = \frac{1}{3}(F1_t^e + F1_t^m + F1_t^h)$ and the threshold with the maximum $F1_t$ is selected as the best-performed threshold. We computed the maximum $F1_t$ for each metric at the IoU thresholds 0.7 and 0.5. According to Table 3, the detection accuracy of highly computational methods such as SA-SSD is close to considerably faster methods such as PointPillars and our network. The results in this table suggest that there is not a huge difference between our realtime method and other highly computational methods in terms of best performed F1 score.

Detecting vulnerable road users (VRUs) such as cyclists is important to increase the safety of all road users. We trained our network on cyclists in the KITTI dataset. To this end, the network is trained after fixing the stride of the first convolution layer in the first block to 1. Table 4 shows the results on the validation set. It is worth mentioning that other methods have not reported results on Cyclists. Concretely, our method outperforms PointPillars in detecting cyclists. Nonetheless, the training set for VRUs is small in the KITTI dataset, and the network overfits on the dataset quickly. For this reason, the model generalizes less on VRUs compared to cars in the KITTI dataset.

4.2. Latency on Embedded Boards

PointPillars is the fastest reported network until now. We computed the time-to-completion (TTC) of our model and PointPillars on a high-end machine and an embedded computing board. The high-end machine is equipped with a Core-i7 CPU and a GTX 2080Ti graphics card while the embedded board is a Jetson AGX Xavier with an 8-core ARM processor and 512-core Volta GPU. It is worth mentioning that we have **not** optimized our model using TensorRT to make a fair comparison with PointPillars. The inference procedure is broken down into encoding, forward pass, and post-processing, and the latency of each stage is measured. Table 5 shows the results on the high-end workstation and the Jetson with various power models. Whereas PointPillars processes 6.15 point clouds per second (pps) on Jetson Xavier, our method processes 10.91 pps that is 177% improvement in runtime on embedded boards. Furthermore, our method processes 84.46 pps on the high-end machine that is 200% increase compared to PointPillars with the processing speed of 42.03 pps. Compared to SA-SSD with the processing speed of 25 pps on a similar machine (Table 1 in [5]), our network processes 336% point clouds per second with practically comparable accuracy. Note that the TTCs in this table reflect the average latency of each method. In practice, the latency of NMS varies depending on the number of objects in the scene. As the sample video in supplementary material indicates, the speed of our network increases to more than 90 pps when there are less than 5 objects and it reduces to 80 pps when there are more than 10 objects in the scene.

The dramatic increase in processing speed is due to the efficient input encoding and decoder. Specifically, we upsample the encoder by a factor of two in the decoder instead of four that makes the output stride of our network

Method	**IoU 0.7** (car)			**IoU 0.5** (car)		
	2D	BEV	3D	2D	BEV	3D
SA-SSD	92.72	90.34	85.38	94.11	93.59	93.37
Point-RCNN	90.80	88.13	82.99	92.23	91.74	91.58
PointPillars	89.30	87.24	79.38	91.86	91.37	91.07
Our	89.66	87.32	78.79	92.26	91.88	91.38

Table 3. Maximum $F1_t$ computed for each metric.

Method	**2D** (cyclist)			**BEV** (cyclist)			**3D** (cyclist)		
	easy	moderate	hard	easy	moderate	hard	easy	moderate	hard
PointPillars	87.11	67.58	63.62	85.51	65.22	61.10	83.72	61.57	57.57
Our	90.20	71.27	68.23	86.39	65.53	61.79	83.95	62.03	58.28

Table 4. Average precision (@IoU 0.5) of our network compared to PointPillars on the validation set for cyclists

Device	**Point Pillars (ms)**				**Our method (ms)**			
	Enc.	**Net.**	**Post**	**pps**	**Enc.**	**Net.**	**Post**	**pps**
Jetson (Max-N)	3.71	139.73	19.05	6.15	0.62	83.20	7.85	10.91
Jeston (15W)	6.45	248.41	29.37	3.51	1.31	162.01	10.74	5.75
Jetson (30W-All)	5.44	195.06	27.60	4.38	1.32	122.83	10.34	7.43
Corei7+RTX 2080Ti	3.08	15.69	5.02	42.03	0.48	9.79	1.57	84.46

Table 5. Comparing the latency of our propose method with PointPillars on Jetson Xavier using different power settings and a RTX-2080Ti machine. The *pps* stands for point clouds per second.

four. This reduces the spatial size of feature maps and the latency of the convolution operations. Positive labels in the final feature map occupy a small region and downsampling them four times, reduces the number of positive samples significantly and make the training difficult. To alleviate this problem, the last feature map is upsampled right before the prediction head by factor of two to increase the number of positive examples and facilitates training. During inference, upsampling before prediction heads is not essential and can be removed from the network. We do not reduce the width or depth of the network, meaning that the complexity of the network remains the same. Instead, upsampling layers are rearranged to generate spatially smaller feature maps and reduce computational complexity. To the best of our knowledge, the proposed method is the fastest 3D detection network reported on the KITTI dataset.

4.3. Ablation Study

Quantization factor: Quantization factor of the occupancy grid plays an important role in latency and accuracy of the network. Table 6 reports the results of our network using different quntization factors. The results indicate that AP on all metrics reduces slightly using cuboid of $12cm$. Additionally, it reduces the processing speed to 53 pps since it generates spatially larger input grids. In contrast, increasing the quantization factor to $20cm$ increases the processing speed to 103 pps and reduces AP notably on hard cases.

Augmentation: Table 7 shows how augmentation affects the performance. Specifically, we divided augmentation methods to two categories. The First category includes the method for adding random samples to point clouds and the second category contains all other augmentation techniques such as rotation, translation, flipping and scaling. The results show that AP on all metrics reduces substantially without any augmentation. Moreover, augmenting the data by only adding random samples has a higher impact on the results compared to only applying augmentations from the second category. This is partially due to the fact that the KITTI is a small dataset and this augmentation techniques help the network to generalize better on unseen samples.

4.4. Error Analysis

We showed that evaluating the network using the IoU threshold 0.5 increases the average precision notably compared to the IoU threshold 0.7. This suggests that the main source of the accuracy drop at higher IoUs is related to prediction errors in the regression head. To study this more carefully, we evaluated our network at different IoU thresholds. Table 8 presents the results. The average precision on moderate and hard classes increases substantially in BEV and 3D metrics by setting the IoU threshold to 0.65. According to IoU thresholds 0.65 and 0.60, while the increase in the average precision is negligible on the easy class, the average precision increase in hard and moderate classes is still remarkable. In general, the point density in hard and moderate classes is lower than easy class. This means the regression head may not generate highly tight bounding boxes for sparse regions in point clouds (more in supplementary materials). Figure 4 shows a few examples where the IoU of the predicted box with the ground-truth (blue) is

QF	PPS	**2D** (car)			**BEV** (car)			**3D** (car)		
		easy	moderate	hard	easy	moderate	hard	easy	moderate	hard
16cm	84	96.24	91.94	89.34	94.86	88.21	85.70	88.70	75.80	72.65
12cm	53	95.86	90.97	87.97	91.97	87.24	85.10	86.35	74.88	72.30
20cm	103	95.56	88.94	88.23	93.78	86.14	84.43	84.74	73.54	69.54

Table 6. The effect of quantization factor on the performance.

Method	**2D** (car)			**BEV** (car)			**3D** (car)		
	easy	moderate	hard	easy	moderate	hard	easy	moderate	hard
w augmentation	96.24	91.94	89.34	94.86	88.21	85.70	88.70	75.80	72.65
w/o augmentation	91.53	77.92	75.84	85.54	75.11	71.64	69.84	57.46	54.93
w/o random samples	93.88	86.85	85.02	92.34	83.04	81.53	77.22	65.73	64.51
w/o transformations	95.45	86.24	83.07	91.47	82.09	79.20	83.01	69.36	64.71

Table 7. The effect of augmentation on the performance.

IoU Threshold	**2D** (car)			**BEV** (car)			**3D** (car)		
	easy	moderate	hard	easy	moderate	hard	easy	moderate	hard
IoU 0.70	96.24	91.94	89.34	94.86	88.21	85.70	88.70	75.80	72.65
IoU 0.65	96.40	92.55	91.94	95.86	91.66	90.89	94.03	84.88	82.13
IoU 0.60	96.45	94.40	92.30	96.22	94.03	91.85	95.83	90.74	88.45
IoU 0.55	96.52	94.83	92.48	96.37	94.41	92.10	96.23	92.14	91.40
IoU 0.50	96.56	94.93	92.57	96.38	94.49	93.86	96.35	94.25	91.95

Table 8. Average precision of the network at different IoU thresholds.

either in $[0.7, 1]$ (green) or in $[0.65, 0.7)$ (magenta).

We selected all the 3D boxes whose IoUs with the ground-truth box is in $[0.65, 0.7)$ and computed the signed difference between regression outputs and the ground-truth. The above procedure was repeated for the range $[0.5, 0.7)$. Then, the histograms of differences for each output were calculated. Figure 5 illustrates stacked histogram for each of the intervals. We did not include the histogram of θ differences because of the page limit (available in the supplementary material). In general, θ differences are mainly distributed around zero and slightly around π and $-\pi$, which shows that the yaw regression is accurate. According to these figures, the offset errors for x and y values are mainly bounded between $-25cm$ and $25cm$ while the offset error for z has wider support and it is bounded between $-70cm$ and $70cm$. Likewise, whereas width and height errors are bounded between $-20cm$ and $20cm$, the length difference is distributed between $-1.5m$ and $1.0m$.

There are 522 instances of car in the validation set whose IoUs are in $[0.65, 0.7)$. To study which outputs have the highest contribution to the average precision reduction, we replaced the predicted values of 522 3D bounding boxes with the ground truth values and computed the average precision. Table 9 illustrates the results. The complete table is available in the supplementary material. According to 3D metric, x and z (8^{th} row) have the most significant contribution to the error since small error in these two quantities will reduce the IoU between predicted and ground-truth boxes. Specifically, replacing predicted x ($2^{nd} row$) values with actual values increase AP on 3D hard class by more than 4%, showing that the $\pm 25cm$ error (according to the histogram) in x values will affect the average precision. The same argument holds for z (4^{th} row) values as replacing them with actual values improves AP about 4% on 3D hard cases, proving that $\pm 70cm$ error in z values contributes to the average precision reduction considerably. Finally, replacing both l and z values (9^{th} row) with the ground-truth will have a profound impact on average precision proving that training a network with more accurate and stable l and z heads will increase AP substantially.

5. Conclusion

In this paper, we proposed a neural network for detecting 3D objects from LiDAR point clouds. Our method improves the processing speed of the fastest published network (to date) by 200% and 177% on high-end machines and embedded computing boards, respectively. It is also 336% faster than the method with the highest detection accuracy reported on the KITTI dataset. More importantly, the remarkable speed gain was achieved while keeping the network complexity high. Specifically, our network utilizes an effective input encoding technique and uses a computationally efficient but highly flexible decoder. Moreover, we assessed our model using *practical metrics* and indicated that the results of the proposed network are comparable to the most accurate published model. Besides, our extensive

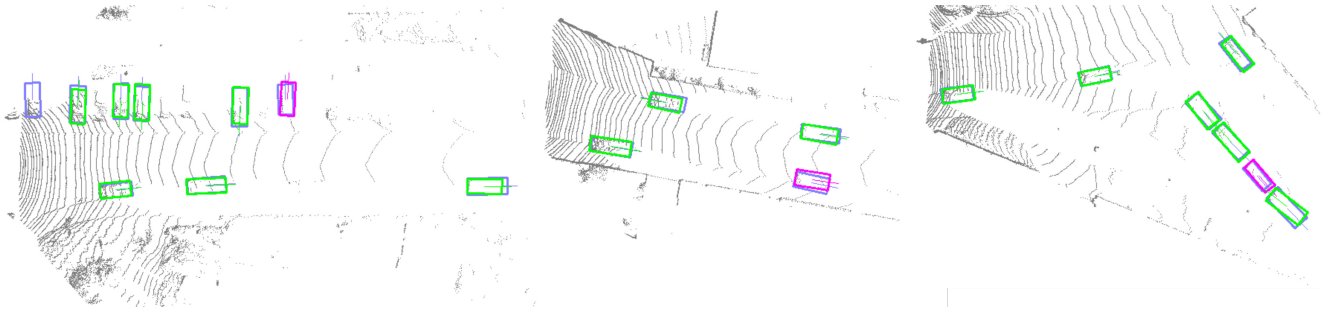

Figure 4. A few samples where the IoU of the predicted box is in $[0.65, 0.7)$ (magenta), $[0.7, 0.1]$ (green).

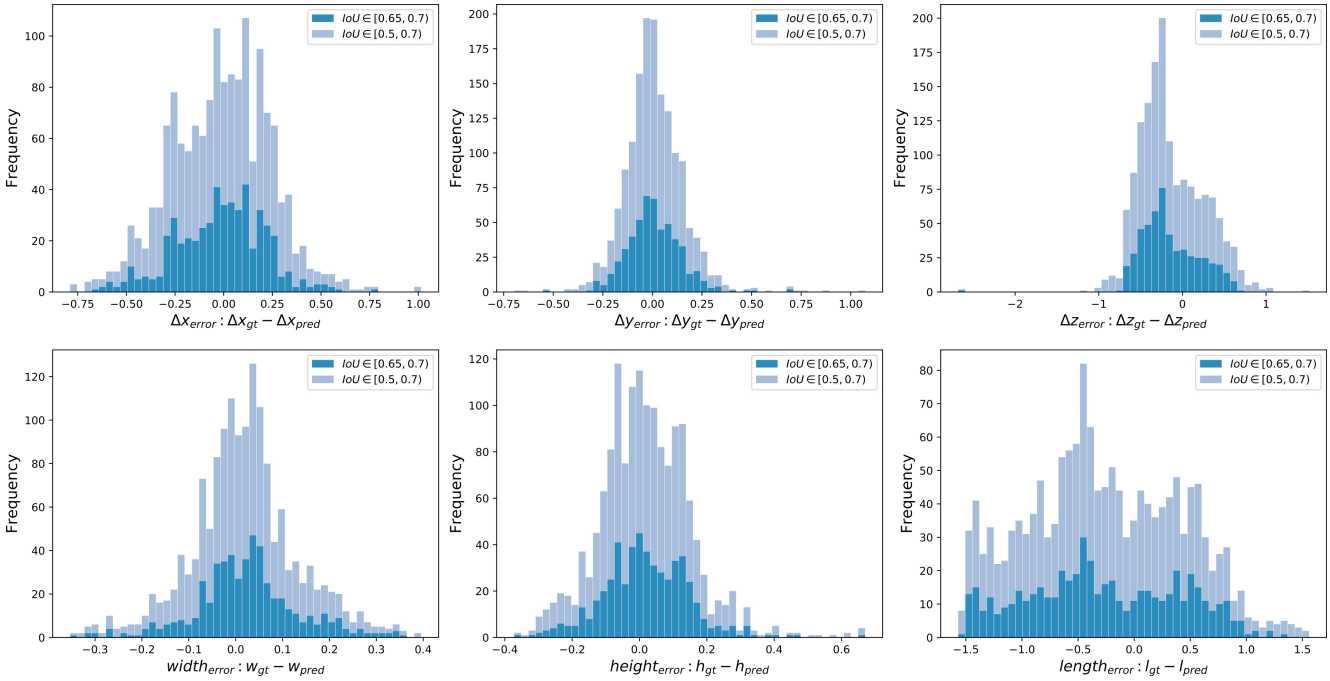

Figure 5. Histograms of differences for each output computed using all the 3D boxes whose IoU with the ground-truth box is in $[0.65, 0.7)$.

Replaced quantity	**2D** (car)			**BEV** (car)			**3D** (car)		
	easy	moderate	hard	easy	moderate	hard	easy	moderate	hard
original	96.24	91.94	89.34	94.86	88.21	85.70	88.70	75.80	72.65
x	96.27	92.25	91.32	95.57	90.76	88.40	90.77	81.26	76.70
y	96.38	94.42	92.06	94.89	88.25	85.74	91.27	82.20	79.62
z	96.16	91.93	89.35	95.40	90.46	88.43	90.10	79.09	76.50
w	96.28	92.03	89.42	95.09	88.40	85.91	89.71	77.97	73.51
h	98.15	91.85	90.83	96.95	90.36	87.83	87.94	76.78	72.45
l	96.24	91.91	89.37	95.04	88.65	87.86	90.08	78.58	74.06
xz	96.18	92.26	91.47	96.13	91.90	91.39	94.13	85.26	82.56
zl	96.26	91.96	89.43	96.21	92.23	91.64	95.05	88.10	85.15
xzy	98.37	94.69	94.01	96.15	93.44	91.41	95.36	90.67	88.27
xzl	96.33	92.39	91.79	96.31	94.44	93.82	95.77	91.58	89.02

Table 9. Contribution of each regression output to the average precision reduction.

analysis showed that $\pm 25cm$ error in Δx values and $\pm 1m$ in Δl values contribute to the reduction in the average precision notably. Overall, considering the accuracy and speed, our proposed network is highly practical to be utilized in ADAS applications on embedded computing boards.

References

[1] 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 C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 424–432. Curran Associates, Inc., 2015.

[2] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-View 3D Object Detection Network for Autonomous Driving.

[3] Yilun Chen, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Fast point R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob:9774–9783, 2019.

[4] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research. *The International Journal of Robotics Research*, (October):1–6, 2013.

[5] Chenhang He, Hui Zeng, Jianqiang Huang, Xian-sheng Hua, and Lei Zhang. Structure Aware Single-stage 3D Object Detection from Point Cloud. *IEEE Conference on Computer Vision and Pattern Recognition*, 1, 2020.

[6] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L. Waslander. Joint 3D Proposal Generation and Object Detection from View Aggregation. *IEEE International Conference on Intelligent Robots and Systems*, pages 5750–5757, 2018.

[7] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:12689–12697, 2019.

[8] Johannes Lehner, Andreas Mitterecker, Thomas Adler, Markus Hofmarcher, Bernhard Nessler, and Sepp Hochreiter. Patch Refinement – Localized 3D Object Detection. (NeurIPS 2019), 2019.

[9] Ming Liang, Yang Bin, Yun CHen, Rui Hu, and Raquel Urtaun. Multi-Task Multi-Sensor Fusion for 3D Object Detection. 2019.

[10] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep Continuous Fusion for Multi-sensor 3D Object Detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11220 LNCS:663–678, 2018.

[11] Arsalan Mousavian, Dragomir Anguelov, Jana Košecká, and John Flynn. 3D bounding box estimation using deep learning and geometry. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:5632–5640, 2017.

[12] Jiquan Ngiam, Benjamin Caine, Wei Han, Brandon Yang, Yuning Chai, Pei Sun, Yin Zhou, Xi Yi, Ouais Alsharif, Patrick Nguyen, Zhifeng Chen, Jonathon Shlens, and Vijay Vasudevan. StarNet: Targeted Computation for Object Detection in Point Clouds. 2019.

[13] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

[14] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:77–85, 2017.

[15] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D object proposal generation and detection from point cloud. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:770–779, 2019.

[16] Andrea Simonelli, Samuel Rota Bulo, Lorenzo Porzi, Manuel Lopez-Antequera, and Peter Kontschieder. Disentangling monocular 3D object detection. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob:1991–1999, 2019.

[17] Sourabh Vora, Alex H. Lang, Bassam Helou, and Oscar Beijbom. PointPainting: Sequential Fusion for 3D Object Detection. 2019.

[18] Zhixin Wang and Kui Jia. Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal. *IEEE International Conference on Intelligent Robots and Systems*, pages 1742–1749, 2019.

[19] Zining Wang, Wei Zhan, and Masayoshi Tomizuka. Fusing Bird's Eye View LIDAR Point Cloud and Front View Camera Image for 3D Object Detection. *IEEE Intelligent Vehicles Symposium, Proceedings*, 2018-June:834–839, 2018.

[20] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors (Switzerland)*, 18(10):1–17, 2018.

[21] Bin Yang, Ming Liang, Raquel Urtasun, Uber Advanced, and Technologies Group. HDNET : Exploiting HD Maps for 3D Object Detection. (CoRL):1–10, 2018.

[22] Bin Yang, Wenjie Luo, and Raquel Urtasun. PIXOR: Real-time 3D Object Detection from Point Clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[23] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.