What You See is What You Get: Exploiting Visibility for 3D Object Detection
Supplementary Materials

Peiyun Hu[1], Jason Ziglar[2], David Held[1], Deva Ramanan[1,2]
[1] Robotics Institute, Carnegie Mellon University
[2] Argo AI
peiyunh@cs.cmu.edu, jziglar@argo.ai, dheld@andrew.cmu.edu, deva@cs.cmu.edu

A. Additional Results

More qualitative examples: Please find a result video at https://youtu.be/8bXkDxSgMsM. We provide a visual guide on how to interpret the visualization in Fig A.

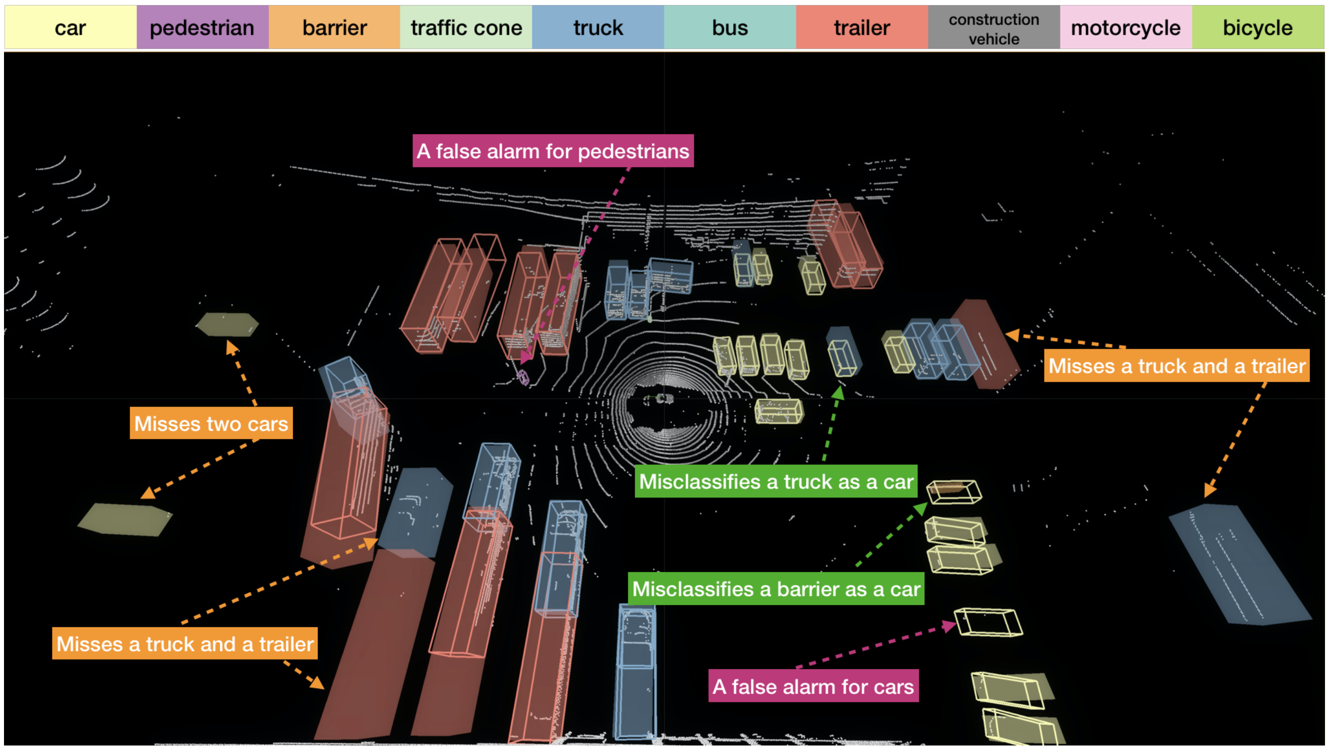
car
pedestrian
barrier
traffic cone
truck
bus
trailer
construction vehicle
motorcycle
bicycle
A false alarm for pedestrians
Misses a truck and a trailer
Misses two cars
Misclassifies a truck as a car
Misclassifies a barrier as a car
Misses a truck and a trailer
A false alarm for cars

Figure A: We use an example frame (same as Fig. 6-(a)) from the video to illustrate how we visualize. Solid transparent cuboids represent ground truth. Wireframe boxes represent predictions. Colors encode object classes (top). Inside this frame, our algorithm successfully detects most cars, trucks, and trailers. We also highlight all the mistakes made by our algorithm.

Performance over multiple runs: We re-train the same model for 5 more times, each time with a different random seed. The seed affects both network initialization and the order in which training data is sampled. We evaluate the detection accuracy of each run on nuScenes validation set and report the mean and standard deviation. As Table 1 shows, the standard deviations are much smaller than the reported improvements, suggesting the performance gain is not due to a lucky run.

Table 1: 3D detection mAP over multiple runs on NuScenes validation set.
†: reproduced based on an author-recommended third-party implementation.

	car	pedes.	barri.	traff.	truck	bus	trail.	const.	motor.	bicyc.	mAP
PointPillars†	76.9	62.6	29.2	20.4	32.6	49.6	27.9	3.8	11.7	0.0	31.5
Ours	80.0	66.9	34.5	**27.9**	35.8	54.1	28.5	7.5	**18.5**	0.0	35.4
Ours (run1)	80.2	67.3	34.5	27.0	36.1	**54.6**	28.9	6.4	17.5	**0.3**	35.3
Ours (run2)	80.5	67.1	35.2	27.0	36.1	52.9	30.5	6.9	16.2	0.0	35.2
Ours (run3)	80.7	66.9	35.4	24.9	35.0	54.0	**31.7**	6.8	16.6	0.1	35.2
Ours (run4)	**80.8**	67.2	34.5	**27.9**	35.8	52.9	29.6	**8.2**	14.8	0.0	35.2
Ours (run5)	80.4	**67.7**	**35.6**	26.8	**36.7**	52.3	31.1	7.9	17.8	0.0	**35.6**
Mean ± Std	80.4 ± 0.3	67.2 ± 0.3	35.0 ± 0.5	26.9 ± 1.1	35.9 ± 0.6	53.5 ± 0.9	30.1 ± 1.3	7.3 ± 0.7	16.9 ± 1.3	0.1 ± 0.1	35.3 ± 0.2

B. Additional Experimental Details

Here, we provide additional details about our method, including pre-processing, network structure, initialization, loss function, training etc. These details apply to both the baseline method (PointPillars) and our two-stream approach.

Pre-processing: We focus on points whose (x, y, z) satisfies $x \in [-50, 50], y \in [-50, 50], z \in [-5, 3]$ and ignore points outside the range when computing pillar features. We group points into vertical columns of size $0.25 \times 0.25 \times 8$. We call each vertical column a pillar. We resample to make sure each non-empty pillar contains 60 points. For raycasting, we do **not** ignore points outside the range and use a voxel size of $0.25 \times 0.25 \times 0.25$.

Network structure: We introduce (1) pillar feature network; (2) backbone network; (3) detection heads.

(1) Pillar feature network operates over each non-empty pillar. It takes points (x, y, z, t) within the pillar and produces a 64-d feature vector. To do so, it first compresses (x, y) to (r), where $r = \sqrt{x^2 + y^2}$. Then it augments each point with its offset to the pillar's arithmetic mean (x_c, y_c, z_c) and geometric mean (x_p, y_p). Please refer to Sec. 2.1 of PointPillars [2] for more details. Then, it processes augmented points $(r, z, t, x - x_c, y - y_c, z - z_c, x - x_p, y - y_p)$ with a 64-d linear layer, followed by BatchNorm, ReLU, and MaxPool, which results in a 64-d embedding for each non-empty pillar. Conceptually, this is equivalent to a mini one-layer PointNet. Finally, we fill empty pillars with all zeros. Based on how we discretize, pillar feature network produces a $400 \times 400 \times 64$ feature map.

(2) Backbone network is a convolutional network with an encoder-decoder structure. This is also sometimes referred to as Region Proposal Network. Please read VoxelNet [6], SECOND [5], and PointPillars [2] for more details. The network consists three blocks of fully convolutional layers. Each block consists of a convolutional stage and a deconvolutional stage. The first (de)convolution filter of the (de)convolutional stage changes the spatial resolution and the feature dimension. All (de)convolution is 3x3 and followed with BatchNorm and ReLU. For our two-stream early-fusion model, the backbone network takes an input of size $400 \times 400 \times 96$, where 64 channels are from pillar feature and 32 channels are from visibility. The first block contains 4 convolutional layers and 4 deconvolutional layers. The second and the third block each consists of 6 both of these layers. Within the first block, the feature dimension changes from $400 \times 400 \times 96$ to $200 \times 200 \times 96$ during the convolutional stage, and $200 \times 200 \times 96$ to $100 \times 100 \times 192$ during the deconvolutional stage. Within the second block, the feature dimension from $200 \times 200 \times 96$ to $100 \times 100 \times 192$. Within the third block, the feature map changes from $100 \times 100 \times 192$ to $50 \times 50 \times 384$ and back to $100 \times 100 \times 192$. At last, features from all three blocks are concatenated as the final output, which has a size of $100 \times 100 \times 576$.

(3) Detection heads include one for large object classes (i.e. car, truck, trailer, bus, and construction vehicles) and one for small object classes (i.e. pedestrian, barrier, traffic cone, motorcycle, and bicycle). The large head takes the concatenated feature map from backbone network as input ($100 \times 100 \times 576$) while the small head takes the feature from the backbone's first convolutional stage as input ($200 \times 200 \times 96$). Each head contains a linear predictor for anchor box classification and a linear prediction for bounding box regression. The classification predictor outputs a confidence score for each anchor box and the regression predictor outputs adjustment coefficients (i.e. x, y, z, w, l, h, θ).

Loss function: For classification, we adopt focal loss [3] and set $\alpha = 0.25$ and $\gamma = 2.0$. For regression output, we use smooth L1 loss (a.k.a. Huber loss) and set $\sigma = 3.0$, where σ controls where the transition between L1 and L2 happens. The final loss function is the classification loss multiplied by 2 plus the regression loss.

Training: We train all of our models for 20 epochs and optimize using Adam [1] as the optimizer. We follow a learning rate schedule known as "one-cycle" [4]. The schedule consists of 2 phases. The first phase includes the first 40% training steps, during which we increase the learning rate from $\frac{0.003}{10}$ to 0.003 while decreasing the momentum from 0.95 to 0.85 following cosine annealing. The second phase includes the rest 60% training steps, during which we decrease the learning rate from 0.003 to $\frac{0.003}{10000}$ while increasing the momentum from 0.85 to 0.95. We use a fixed weight decay of 0.01.

References

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[2] 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

[3] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *CVPR*, pages 2980–2988, 2017. 2

[4] Leslie N Smith. Cyclical learning rates for training neural networks. In *WACV*, pages 464–472. IEEE, 2017. 2

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

[6] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, pages 4490–4499, 2018. 2