4D-Net for Learned Multi-Modal Alignment - Supplemental Material

AJ Piergiovanni
Google Research

Vincent Casser
Waymo LLC

Michael Ryoo
Robotics at Google

Anelia Angelova
Google Research

1. Implementation Details

The models were implemented in TensorFlow. We trained for 120 000 iterations using a batch size of 256, split across 8 devices. The learning rate was set to 0.0015 using a linear warmup for 6 000 steps followed by a cosine decay schedule.

The anchor boxes had size of $[4.7, 2.1, 1.7]$, and 2 rotations (0 and 45 degrees) used at each feature map location. For the PointPillar pseudo-image creation, we used a grid size of $(224, 224, 1)$, an x-range of $(-74.88, 74.88)$ and the same for y-range. The z-range was $(-5, 5)$. The max number of points per cell, $N = 128$. We used 10,000 pillars.

We applied data augmentation to the point clouds (random 3D rotations and flips). The camera matrices were also updated based on the augmentations so the projections would still apply. No augmentation was used on the RGB streams.

The PointPillars network used a feature dim of 64 for the input points. The created pseudo-image had 224×224 shape. This was followed by 3 convolutional blocks with 4, 6, and 6 repeats. Each block consisted of a convolution, batch norm and ReLU activation. This was followed by 3 deconvolutional layers which generate the predictions.

The RGB single frame network is a standard ResNet-18. The video network is based on TinyVideoNetworks. Specifically, we use a network that consists of 6 Residual blocks, the structure is outlined in Table 1. Note that the first two blocks apply both spatial and temporal convolutions to the data (in that order).

We trained on the Waymo Open Dataset, which consists of 1 950 segments that are each 20 seconds long (about 200 frames), a total of 390 000 frames. The LiDAR data is processed into point clouds grouped by timestamp and aligned with the RGB frames. In most experiments, we take sequences of 16 frames as input and predict 3D boxes for the last frame. We evaluate using the provided Waymo metrics library. Before NMS, we filter out boxes with probability less than 0.4 and boxes larger than 30m in length and 5m in width and boxes smaller than 0.5m in length and width.

Block	Conv Size	Channels	Repeat	Output Size
Input	-	-	-	$16{\times}224 \times 224$
Block 1	1x3x3 + 3x1x1 conv	32	1	$8{\times}112 \times 112$
Block 2	1x3x3 + 3x1x1 conv	64	1	$4{\times}56 \times 56$
Block 3	1x3x3 conv	128	4	$2{\times}28 \times 28$
Block 4	1x3x3 conv	256	4	$2{\times}14 \times 14$

Table 1. RGB Video Network structure used in 4D-Net. Note that the sizes are shown assuming 16 frames at 224×224 input size. For networks that used smaller inputs, the output sizes are each step would be smaller, following the same scaling. Average pooling was used after the convolution to reduce the spatial size. The first two blocks apply both spatial and temporal convolutions in that order.

Method	AP L1	AP L2	AP 30m	AP 30-50m	AP 50m+
Base PointPillars	55.7	52.8	65.0	51.3	35.4
RGB	56.7	53.6	66.2	52.5	37.5
Spatial Avg	58.9	57.6	69.7	56.1	39.6
Spatial Transformer	59.5	58.2	70.0	54.7	43.1
Projection	64.3	63.0	74.9	65.1	47.2

Table 2. Comparison of different spatial fusion methods.

2. Additional experimental results

Fusion Methods In the main paper, we focused on projection as the main method to fuse RGB and point cloud data. Here, we also compare to several other methods. The results are shown in Table 2.

Basic RGB. Here we flatten the RGB image feature output into a 1-D tensor, and concatenate it to the point cloud feature. This loses all spatial information and essentially puts the entire image into each point.

Spatial Avg Pooling. As another baseline, we apply average spatial pooling to the image-based features R_i, obtaining a F_i^R-dimensional feature vector. We then concatenate this to each feature in the PC-based features M_i, resulting in a $(X_i^M, Z_i^M, F_i^M + F_i^R)$ feature map. This is then passed through the remaining CNN for classification. This provides the point cloud stream with some RGB information, but it has no spatial information.

Spatial Transformer. We also tried using a spatial transformer [1] to crop regions around each projected point to append to the PC feature. However, despite small improvements, we found this to be extremely slow due to taking many spatial crops with the transformer.

These baselines are compared with the proposed *Projection* method which is in Table 2 of the main paper. The experiments are conducted in the same conditions as the Projection method in the main paper. The *Basic PointPillars* (also in Table 2 of the main paper) does not have an RGB input and is included for reference only.

3. Additional Visualizations

In Figure 1 we show more visualizations of the predictions of the 4D-Net on the Waymo Open Dataset.

References

[1] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *arXiv preprint arXiv:1506.02025*, 2015. 1

[2] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2446–2454, 2020. 3

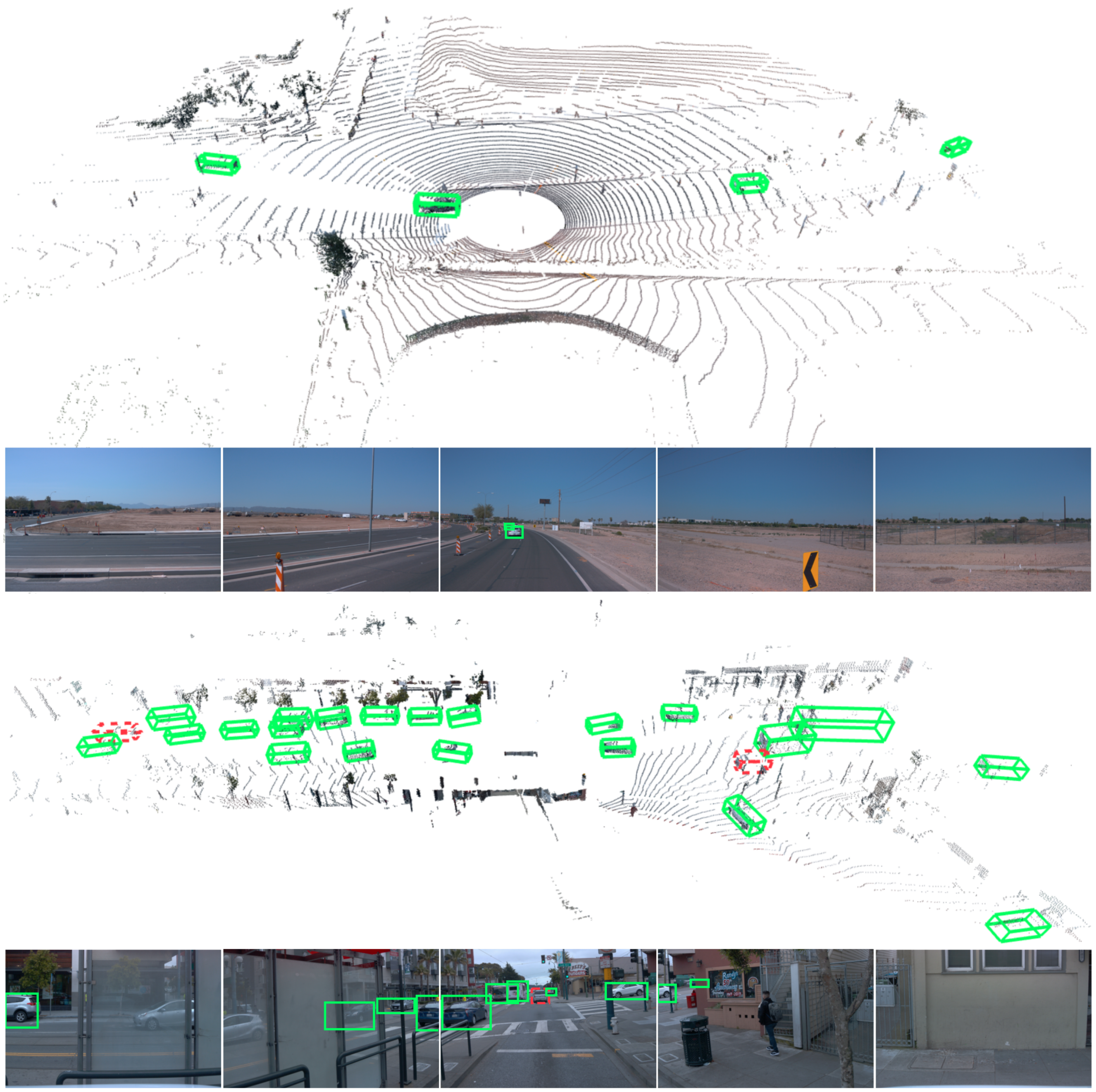
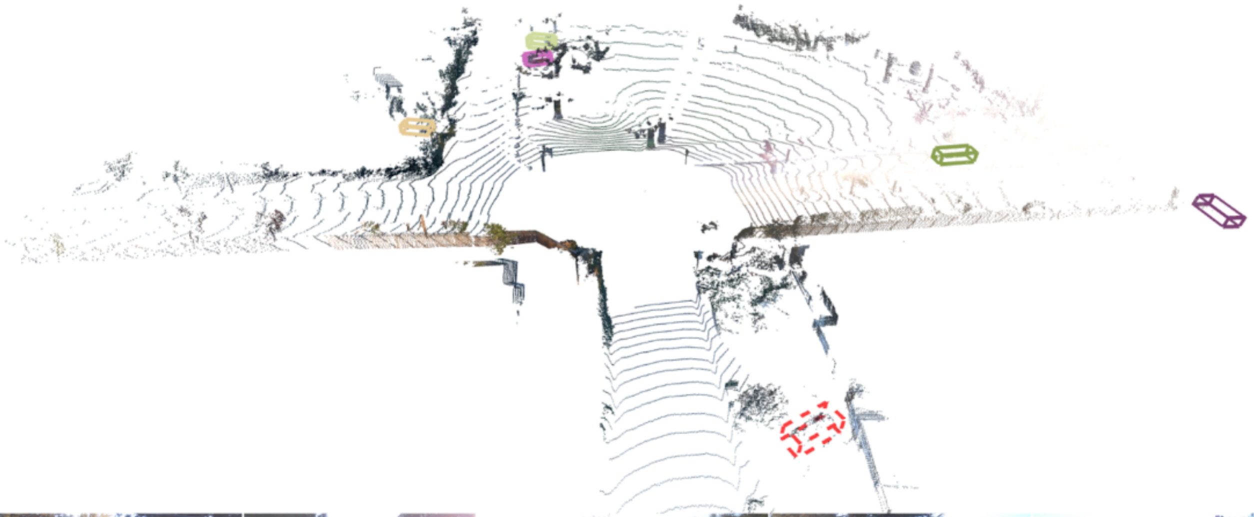
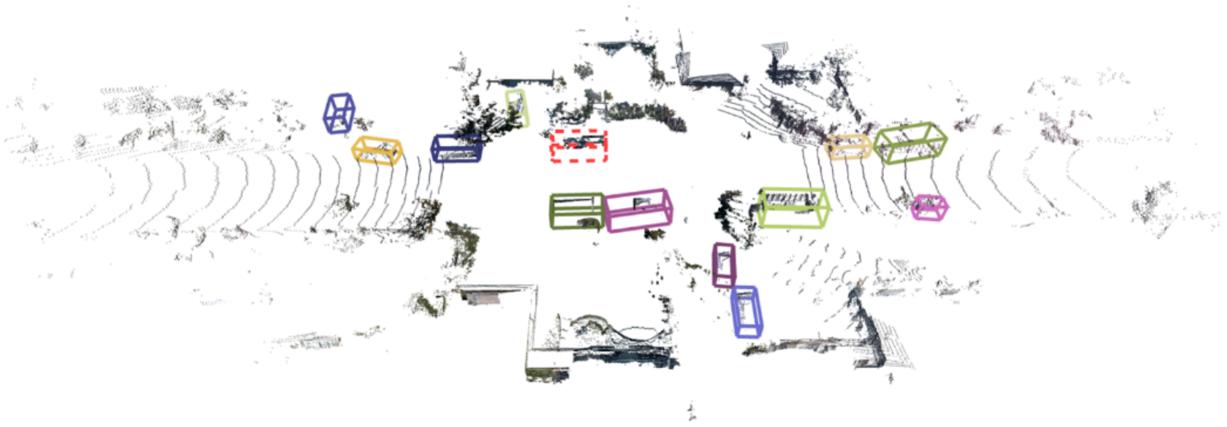


Figure 1. 4D-Net predictions on a scene in the Waymo Open Dataset [2]. Individual instances are shown in green (here) or in different colors in the figures below. Red boxes indicate errors (dashed lines: FN, solid lines: FP). The front camera (central image) is the only one used in our work presently, the others are included for visualization purposes. Note that any misalignments in the camera view are due to projection, not by inaccuracies in the predictions.





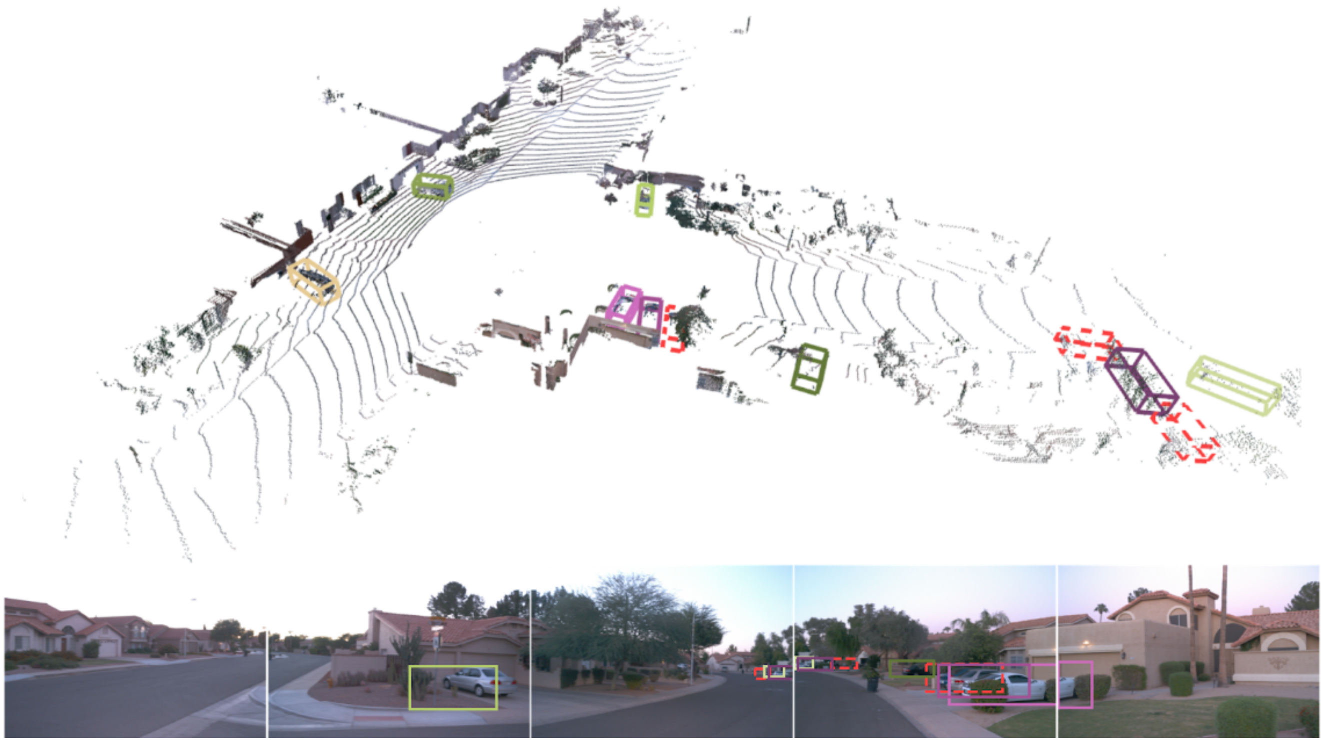
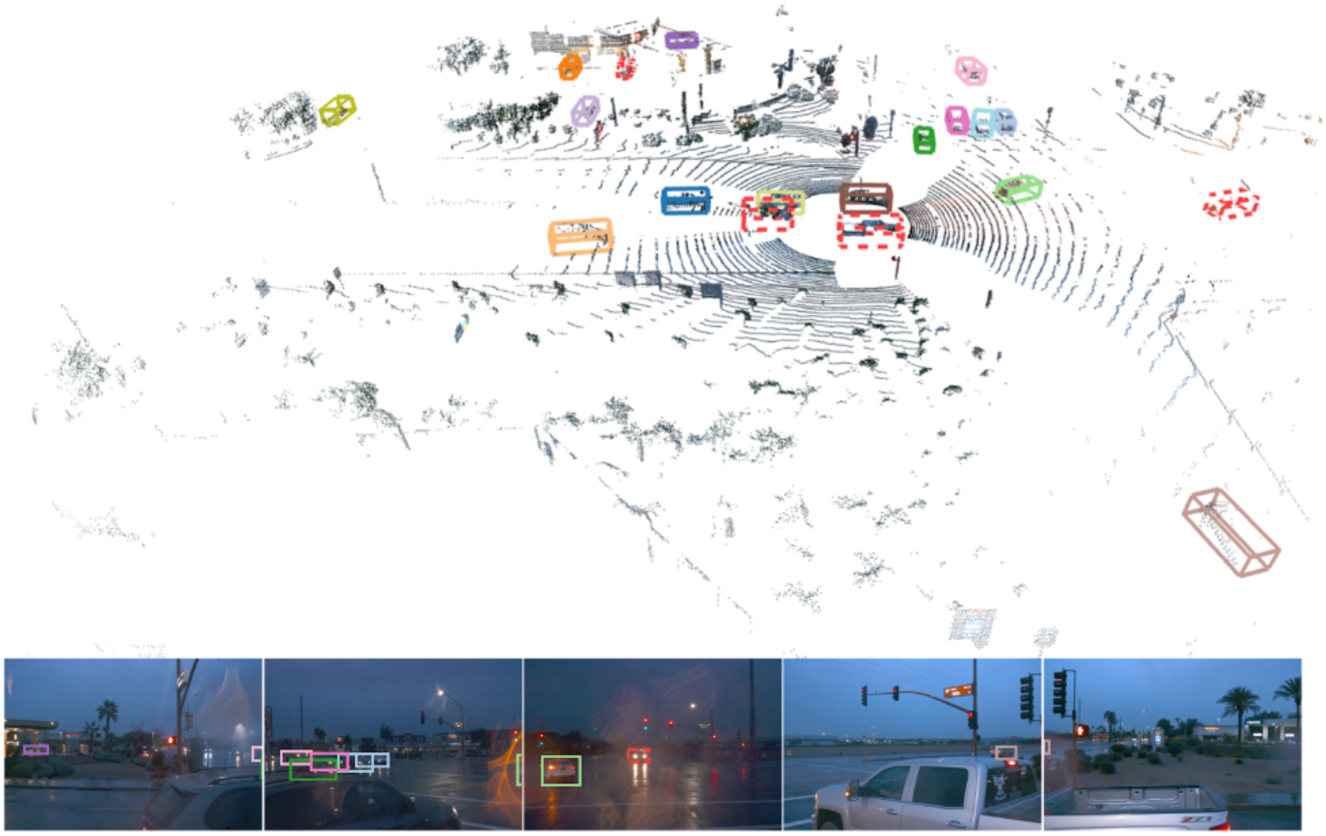



Figure 2. Failure cases examples: In these two challenging cases, the central image (stream) is not sufficient and a vehicle is missed.