Supplementary Materials for DSDNet: Deep Structured self-Driving Network

Wenyuan Zeng[1,2], Shenlong Wang[1,2], Renjie Liao[1,2], Yun Chen[1], Bin Yang[1,2], and Raquel Urtasun[1,2]

[1] Uber ATG
[2] University of Toronto
{wenyuan,slwang,rjliao,yun.chen,byang10,urtasun}@uber.com

1 Datasets

CARLA: This is a public available multi-agent trajectory prediction dataset, collected by [3] using CARLA simulator [1]. It contains over 60k training sequences, 7k testing sequences collected from Town01, and 17k testing sequences from Town02. Each sequence is composed of 2 seconds of history, and 4 seconds future information.

nuScenes: It contains 1000 driving snippets of length 20 seconds each. LiDAR point clouds are collected at 20Hz, and labels 3D bounding boxes are provided at 2Hz. To augment the labels, we generate bounding boxes for non-labeled frames using linear interpolation from 2 consecutive labeled frames. Since nuScenes dataset currently does not provide routing information for motion planning, we only conduct detection and prediction studies. We follow the official data split and compare against other methods on the "car" class.

ATG4D: We collected a challenging self-driving datasets over multiple cities across North America. It contains $\sim$ 5,000 snippets collected from 1000 different trips, with a 64-beam LiDAR running at 10 Hz and HD maps. We also labeled the data at 10 Hz, with maximum labeling range of 100 meters. We ignore parking areas far from the roads, as they will not interact with the SDV. We split out 500 snippets for testing and evaluate the full autonomy stack including motion planning, prediction as well as detection performance.

2 Network Architecture Details

In the following, we first describe our backbone network, the detection header, as well as the header for computing prediction (E_{traj}) as well as planning C_{traj}. Note we use the same architecture for nuScenes and ATG4D, but a slightly different one for CARLA as the setting there is different, which we will explain in section 4.

Backbone: Our backbone is adapted from the detection network of [4, 5], which has 5 blocks of layers in total. There are {2, 2, 3, 6, 5} Conv2D layers with {32, 64, 128, 256, 256} number of filters in those 5 blocks respectively. All Conv2D kernels are 3x3 and have stride 1. For the first three blocks, we use a max-pooling layer after each block to downsample the feature map by 2. After the 4-th block, we construct a multi-scale feature map by resizing the feature maps after each block to be of the same size (4 times smaller than the input) and then concatenate them together. This multi-scale feature map is then fed to the 5-th block. The final feature map computed by the 5-th block has a downsample rate of 4, and is shared for detection, prediction, and motion planning modules.

Detection Header: We use feature maps from the backbone and apply a single-shot detection header, similar to SSD [2], to predict the location, shape, orientation and velocity of each actor. More specifically, the detection header contains two Conv2D layers with 1×1 kernel, one for classification and the other one for regression. We apply the two Conv2D on the backbone feature map separately. To reduce the variance of regression outputs, we follow SSD [2] and use a set of predefined anchor boxes: each pixel at the backbone feature map is associated with 12 anchors, with different sizes and aspect ratios. We predict a classification score $p^k_{i,j}$ for each pixel (i, j) and anchor k on the feature map, which indicates how likely it is for an actor to be presented at this location. For the regression layer, the header outputs the offset values at each location. These offset values include position offset l_x, l_y, size s_w, s_h, heading angle a_{sin}, a_{cos}, and velocity v_x, v_y. Their corresponding ground-truth target values can be computed using the labeled bounding box, namely,

$$
\begin{aligned}
l_x = \frac{x^l - x^a}{w^a} \quad & l_y = \frac{y^l - y^a}{h^a}, \\
s_w = \log \frac{w^l}{w^a} \quad & s_h = \log \frac{h^l}{h^a}, \\
a_{sin} = \sin(\theta^l - \theta^a) \quad & a_{cos} = \cos(\theta^l - \theta^a), \\
v_x = l_x^{t=1} - l_x^{t=0} \quad & v_y = l_y^{t=1} - l_y^{t=0},
\end{aligned}
$$

where subscript l means label value, and a means anchor value. Finally, we combine these two outputs and apply an NMS operation to determine the bounding boxes for all actors and their initial speeds.

Prediction (E_{traj}) / Planning (C_{traj}) Headers: In addition to the detection header, our model has two headers: one outputs prediction energy E_{traj}, the other outputs motion planning costs C_{traj}. Note that these two headers have the same architecture, but different learnable parameters. After computing the backbone feature map, we apply four Conv2D layers with 128 filters. This increases the model capacity to better handle multiple tasks. To extract the actor features, we perform ROI pooling based on the actor's detection bounding box, which output a $16 \times 16 \times 128$ feature tensor for this actor. We then apply

another four Conv2D layers, each with a downsample rate of 2 and filter size $\{256, 512, 512, 512\}$ respectively. This gives us a 512 dimensional feature vector for each actor. Note that we parameterized trajectories with 7 waypoint (2Hz for 3 second, including the inital waypoint at 0 second). To extract trajectory features, we first index the feature on our header feature map at those waypoints with bilinear interpolation. This gives us a 7×128 feature. We then concatenate this feature with the $(x_t, y_t, \cos\theta_t, \sin\theta_t, \text{distance}_t)$ of those 7 waypoints, where (x_t, y_t) is the coordinate of that waypoint, $(\cos\theta_t, \sin\theta_t)$ is the direction, and distance_t is the traveled distance along the trajectory up to that waypoint. Finally, we feed the actor and trajectory features to a 5 layer MLP to compute the E_{traj} / C_{traj} value for this trajectory. The MLP has $(1024, 1024, 512, 256, 1)$ neurons for each layer.

3 Trajectory Sampler Details

Following NMP [5], we assume a bicycle dynamic model for vehicles, and we use a combination of straight line, circle arcs, and clothoid curves to sample possible trajectories. More specifically, to sample a trajectory for a given detected actor, we first estimate its initial position and speed as well as heading angle from our detection output. We then sample the mode of this trajectory, *e.g.*, straight, circle, clothoid proportional to $(0.3, 0.2, 0.5)$ probability. Next, we uniformly sample values of control parameters for the chosen mode, *e.g.*, radius for circle mode, radius and canonical heading angle of clothoid. These sample parameters determine the shape of this trajectory. We then sample an acceleration value, and compute the velocity values for the next 3 seconds based on this acceleration and the initial speed. Finally, we go along our sampled trajectory with our sampled velocity, to determine the waypoints along this trajectory for the next 3 seconds.

In our experiments, we notice that different numbers of samples used for inference will affect the final performance. For a metric only cares about precision, e.g., L2, which we used on nuScenes and ATG4D, more samples generally produces better performance. For instance, increasing number of trajectory samples (for inference) from 100 to 200 will decrease final timestep L2 error from 1.29m to 1.22m on ATG4D, but further increase number of samples only brings marginal improvements. Due to the consideration of memory and speed, we use 200 trajectory samples during inference and 100 trajectory samples for training. However, for a metric that considers both diversity and precision, e.g., minMSD (CARLA), there is a sweet point of number of samples. On CARLA, we found that sample 100 trajectories during inference performs worse than sampling 50 samples, which corresponds to 0.24 and 0.18 minMSD on validation set respectively, and sampling 1000 trajectories performs the worst, producing a minMSD of 0.30. This is because when presented with a set of dense samples, trajectories are spatially very close to each other. As a result, the highest-scored samples and its nearby samples will have very similar scores, and thus selected by the top-K evaluation procedure, which loses some diversity. We also found on CARLA, it's

helpful to regress a future trajectory from the backbone and add it to the trajectory samples set before our prediction module, but not on nuScenes or ATG4D. This is potentially due to the fact that our dynamic model and sampler fits well to real-world data, but has a gap with CARLA's dynamic model.

4 Implementation Details

nuScenes We follow the dataset range defined by the creators of nuScenes, and use an input region of $[-49.6, 49.6] \times [-49.6, 49.6] \times [-3, 5]$ meters centered at the ego car. We aggregate the current LiDAR sweep with past 9 sweeps (0.5s period), and voxelize the space with $0.2 \times 0.2 \times 0.25$ meter per voxel resolution. This gives us a $496 \times 496 \times 320$ input LiDAR tensor. We further rasterize the map information with the same resolution. The map information includes road mask, lane boundary and road boundary, which provide critical information for predicting the behavior of a vehicle.

We train our model for the car class, using Adam optimizer with initial learning rate of 0.001. We decay the learning rate by 10 at the 6-th and 7-th epoch respectively, and stop training at 8-th epoch. The training batch size is 80 and we use 16 GPUs in total. For detection, we treat anchors larger than 0.7 IoU with labels as positive examples, and smaller than 0.5 IoU as negative, and treat others as ignore. We adopt hard-mining to get good detection performance. For training the prediction, we treat a detection as positive if it has larger than 0.1 IoU with labels, and only apply prediction loss on those positive examples. Besides, we apply data augmentation [6] during training: randomly translating a frame (-1 to 1 meters in XY and -0.2 to 0.2 for Z), random rotating along the Z axis (-45 degree to 45 degree), randomly scaling the whole frame (0.95 to 1.05), and randomly flipping over the X and Y axes.

CARLA-PRECOG: We train the model with Adam optimizer, using a learning rate of 0.0001, and decay by 10 at 20 epochs and 30 epochs, and finish training at 40 epochs. We use batch size of 160 on 4 GPUs. No other data augmentation is applied on this dataset.

Since the dataset and experiment setup is different from nuScenes (Carla has no map and only 4 height channels for LiDAR; the prediction task is also provided with ground-truth actors' locations), we use a slightly different architecture. We first rasterize the LiDAR and the past positions of all actors similar to PRECOG [3], and feed them into a shallower backbone network with 5 blocks of Conv2D layers, each has {2,2,2,2,2} layers respectively. We then compute the feature of each actor by concatenating the actor features extracted from the backbone and a motion feature. Such a motion feature is computed by applying several Conv1D layers on top of the past locations of an actor, which is a $T \times 3$ tensor. Those 3 dimensions are x, y coordinates and timestamp indexes. Finally, we compute the E_{traj} and prediction results using the same architecture we have described in Section. 2.

5 More Qualitative Results

Additionally we provide more qualitative results showing our inference results on ATG4D. In Figure. 1 we show the inputs and outputs of our model, and in Figure. 2 we explain how we visualize the prediction uncertainty estimated by our model. We can clearly see that our approach produces multimodal estimates when an actor is approaching an intersection. We provide more cases showing prediction results in Figure. 3. Finally, we show our motion planner can handle various situations including lane following, turning and nudging around other vehicles to avoid collision in Figure. 4. Our detections and predictions are shown in cyan, and the ego-car as well as motion planning results are shown in red.

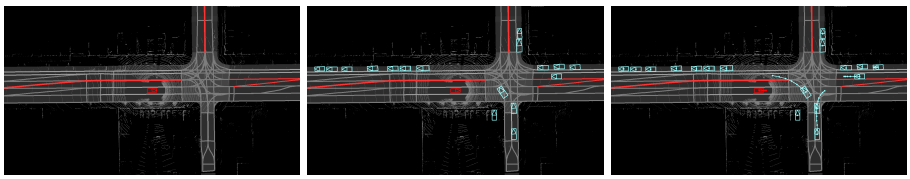

Fig. 1: Left: Input to our model. Middle: Detection outputs (shown in cyan). Right: Socially consistent prediction (shown in cyan) and safe motion planning (shown in red).

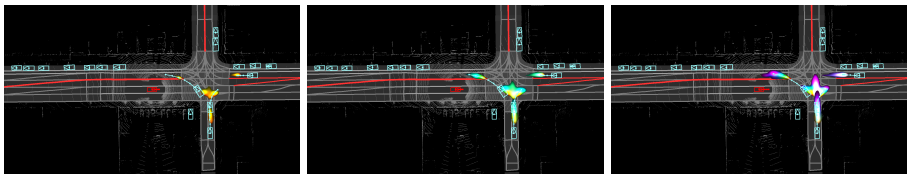

Fig. 2: We visualize the prediction uncertainty estimated by our model. We highlight the high-probability regions that actors might go to at different future timestamps using different colors, and overlay them together to have a better visualization. From left to right: high-probability region at 1 to 3 seconds into the future. We can observe clear multi-modality for the actor near an intersection (going straight / turning left / turning right).

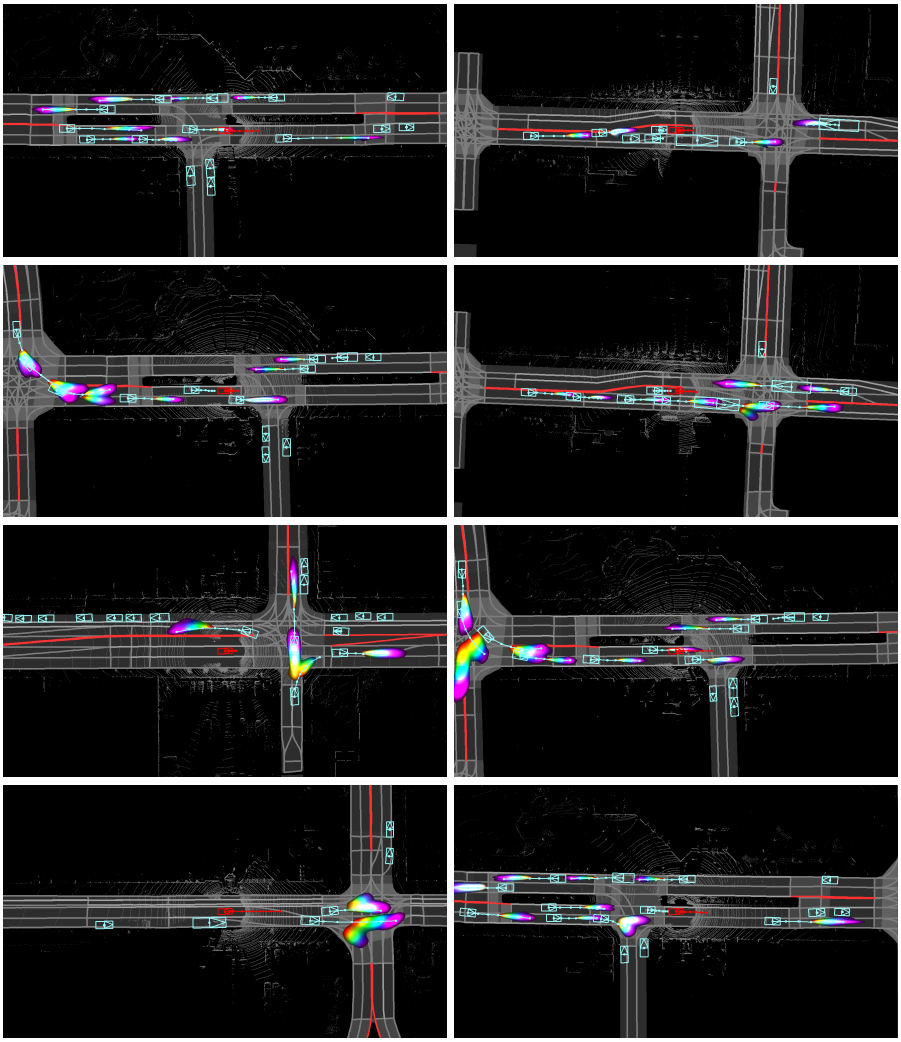

Fig. 3: More Prediction Visualizations. We show our predicted uncertainty: 1) aligns with lanes when an actor is driving on a straight road (meaning that we are certain about future direction but not certain about future speed in this case). 2) shows multi-modality when an actor approaches an intersection (either turning or going straight).

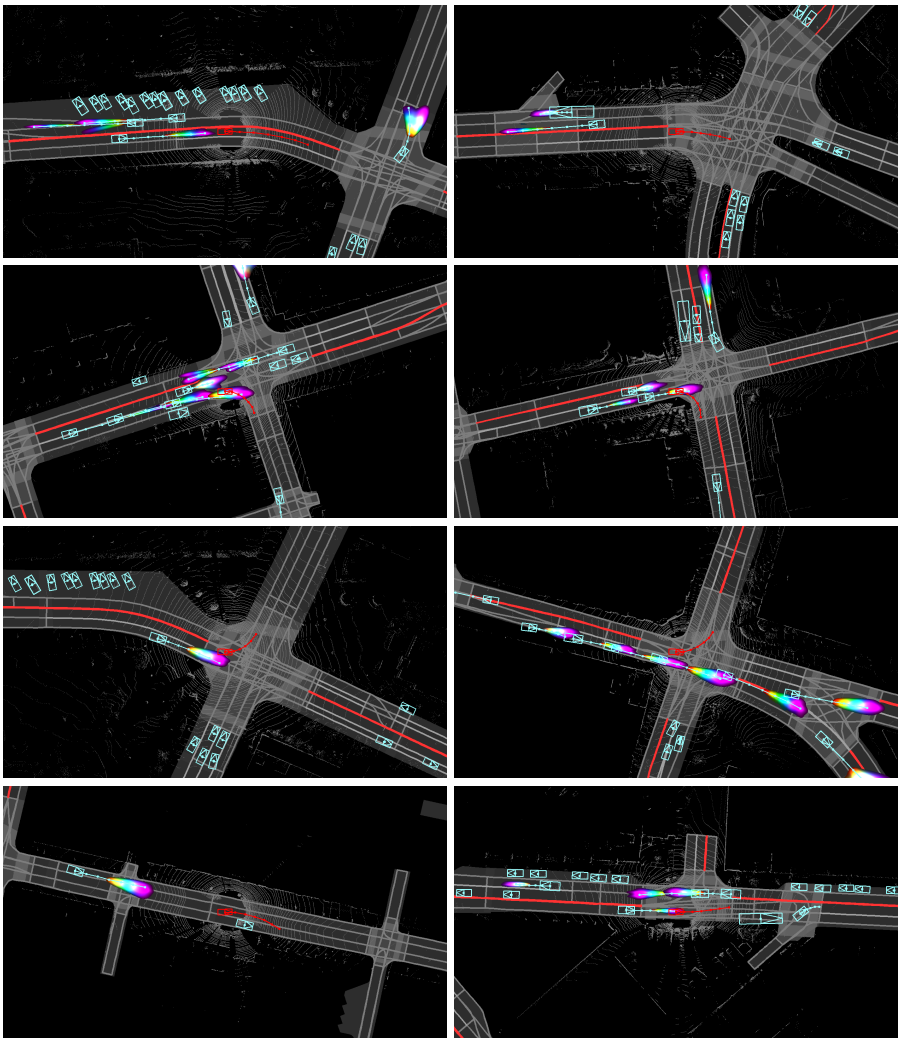

Fig. 4: More Planning Visualizations: We show our motion planner can nicely handle lane following (row 1), turning (row 2 & 3) and nudging to avoid collision (row 4).

References

1. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: Carla: An open urban driving simulator. arXiv preprint arXiv:1711.03938 (2017)
2. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: European conference on computer vision. pp. 21–37. Springer (2016)
3. Rhinehart, N., McAllister, R., Kitani, K., Levine, S.: Precog: Prediction conditioned on goals in visual multi-agent settings. arXiv preprint arXiv:1905.01296 (2019)
4. Yang, B., Luo, W., Urtasun, R.: Pixor: Real-time 3d object detection from point clouds
5. Zeng, W., Luo, W., Suo, S., Sadat, A., Yang, B., Casas, S., Urtasun, R.: End-to-end interpretable neural motion planner. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8660–8669 (2019)
6. Zhu, B., Jiang, Z., Zhou, X., Li, Z., Yu, G.: Class-balanced grouping and sampling for point cloud 3d object detection. arXiv preprint arXiv:1908.09492 (2019)