Meta-Sim: Learning to Generate Synthetic Datasets

http://nv-tlabs.github.io/meta-sim

Amlan Kar[1,2,3] Aayush Prakash[1] Ming-Yu Liu[1] Eric Cameracci[1] Justin Yuan[1]
Matt Rusiniak[1] David Acuna[1,2,3] Antonio Torralba[4] Sanja Fidler[1,2,3*]

[1]NVIDIA [2]University of Toronto [3]Vector Institute [4] MIT

1. Appendix

We provide additional details about Meta-Sim, as well as include more results.

1.1. Grammar and Attribute Sets

First, we describe the probabilistic grammars for each experiment in more detail, as well as attribute sets. We remind the reader that the probabilistic grammar defines the structure of the scenes, and is used to sample scene graphs that are input to Meta-Sim. The Distribution Transformer transforms the attributes on the nodes in order to optimize its objectives.

1.1.1 MNIST

Grammar. The sampling process for the probabilistic grammar used in the MNIST experiments is explained in the main paper, and we review it here. It samples a background texture and one digit texture (image) from the MNIST dataset [2] (which has an equal probability for any digit), and then samples a rotation and location for the digit, to place it on the scene.

Attribute Set. The attribute set s_A for each node consists of *[class, rotation, locationX, locationY, size]*. Here, class is a one-hot vector, spanning all possible node classes in the graph. This includes, *[scene, background, 0, ..., 9]*. Here, *scene* is the root node of the scene graph. The rotation, locations and size are floating point numbers between 0 to 1, and represent values in appropriate ranges *i.e.* rotation in 0 to 360 degrees, each location within the parent's boundaries etc.

1.1.2 Aerial Views (2D)

Grammar. The probabilistic grammar is explained in the main paper, and we summarize it again here. First, a background grass texture is sampled. Next, a (straight) road is sampled at a location and rotation on the background. Two cars are then sampled with independent locations (constrained to be in the road by parametrizing in the road's coordinate system), and rotations. In addition, a tree and a house are sampled and placed randomly on the scene. Each object in the scene also gets assigned a random texture from a repository of collected textures.

Attribute Set. The attribute set s_A for each node in the Aerial 2D experiment consists of *[class, rotation, locationX, locationY, size]*. Our choice of class includes road, car, tree, house and background, with class as a one-hot vector. Subclasses, such as type of tree, car, etc, are not included in the attribute set. They are sampled randomly in the grammar and are left unchanged by the Distribution Transformer. The values are normalized similarly to that in the MNIST experiment. We learn to transform the *rotation* and *location* for every object in the scene, but leave *size* unchanged, *i.e.* $size \notin s_{A,mut}$.

*Correspondence to amlan@cs.toronto.edu, sfidler@nvidia.com

1.1.3 Driving Scenes (3D)

Grammar. The grammar used in the driving experiment is adapted from [3] (Section III). Our adaptation uses some fixed global parameters (*i.e*. weather type, sky type) as compared to SDR. Our camera optimization experiment learns the height of the camera from the ground, while scene contrast, saturation and light direction are randomized. We also do not implement side-streets and parking lanes.

Attribute Set. The attribute set s_A for each node in the Driving Scene 3D experiment consists of *[class, rotation, distance, offset]*. The classes, subclasses and ranges are treated exactly like in the Aerial 2D experiment. *Distance* indicates how far along the parent spline the object is being placed and *offset* indicates how much across the parent spline the object is placed. For each experiment, a subset of nodes in the graph is kept mutable. We optimized the attributes of cars, context elements such as buildings, foliage and people. Camera height from the global parameters was optimized, which was injected into the sky node encoded in the distance attribute.

1.2. Training Details

Let a_{in} be the number of attributes in s_A for each dataset. Our Distribution Transformer use Graph Convolutional layers with different weight matrices for each direction of the edge following [5]. We use 2 layers for the encoder and 2 layers for the decoder. In the MNIST and Aerial2D experiments, it has the following structure: Encoder ($a_{in} - > 16 - > 10$), and Decoder ($10 - > 16 - > a_{in}$). For 3D driving scenes, we use 28 and 18 feature size in the intermediate layers.

After training the autoencoder step, the encoder is kept frozen and only the decoder is trained (*i.e*. for the distribution matching and task optimization steps). For the MMD computation, we always use the pool1 and pool2 feature layers of the InceptionV3 network [4]. For the MNIST and Aerial2D experiments, the images are not resized before passing them through the Inception network. For the Driving Scenes (3D) experiment, they are resized to 299 x 299 (standard inception input size) before computing the features. The task network training is iterative, our Distribution Transformer is trained for one epoch, followed by training of the task network for a few epochs.

1.2.1 MNIST

For the MNIST experiment, 500 samples coming from the probabilistic grammar ared said to constitute 1 epoch.

Distribution Transformer. We first train the autoencoder step of the Distribution Transformer for 8 epochs, with batch size of 8 and learning rate 0.001 using the adam optimizer [**?**]. Next, training is done with the joint loss, i.e. Distribution Loss and Task Loss, for 100 epochs. Note that when training with the task loss, there is only get 1 gradient step per epoch. We note that the MNIST tasks could be solved with the Distribution Loss alone, in around 3 epochs (since we can backpropagate the gradient for the distribution loss per minibatch when training only with the Distribution Loss).

Task Network. After getting each epoch of data from the Distribution Transformer, the Task Network is trained with this data for 2 epochs. The optimization was done using SGD with learning rate 0.01 and momentum 0.9. Note that the task network is not trained from scratch every time. Rather, training is resumed from the checkpoint obtained in the previous epoch.

1.2.2 Aerial Views (2D)

For the Aerial Views (2D) experiment, 100 samples coming from the probabilistic grammar are considered to be 1 epoch.

Distribution Transformer. Here, the autoencoder step is trained for 300 epochs using the adam optimizer with batch size 8 and learning rate 0.01, and then for an additional 100 epochs with learning rate 0.001. Finally, the model is trained jointly with the distribution and task loss for 100 epochs with the same batch size and a learning rate of 0.001.

Task Network. After getting each epoch of data from the Distribution Transformer, the Task Network is trained with this data for 8 epochs using SGD with a learning rate of 1e-8. Similar to the previous experiment, the task-network training is resumed instead of training from scratch.

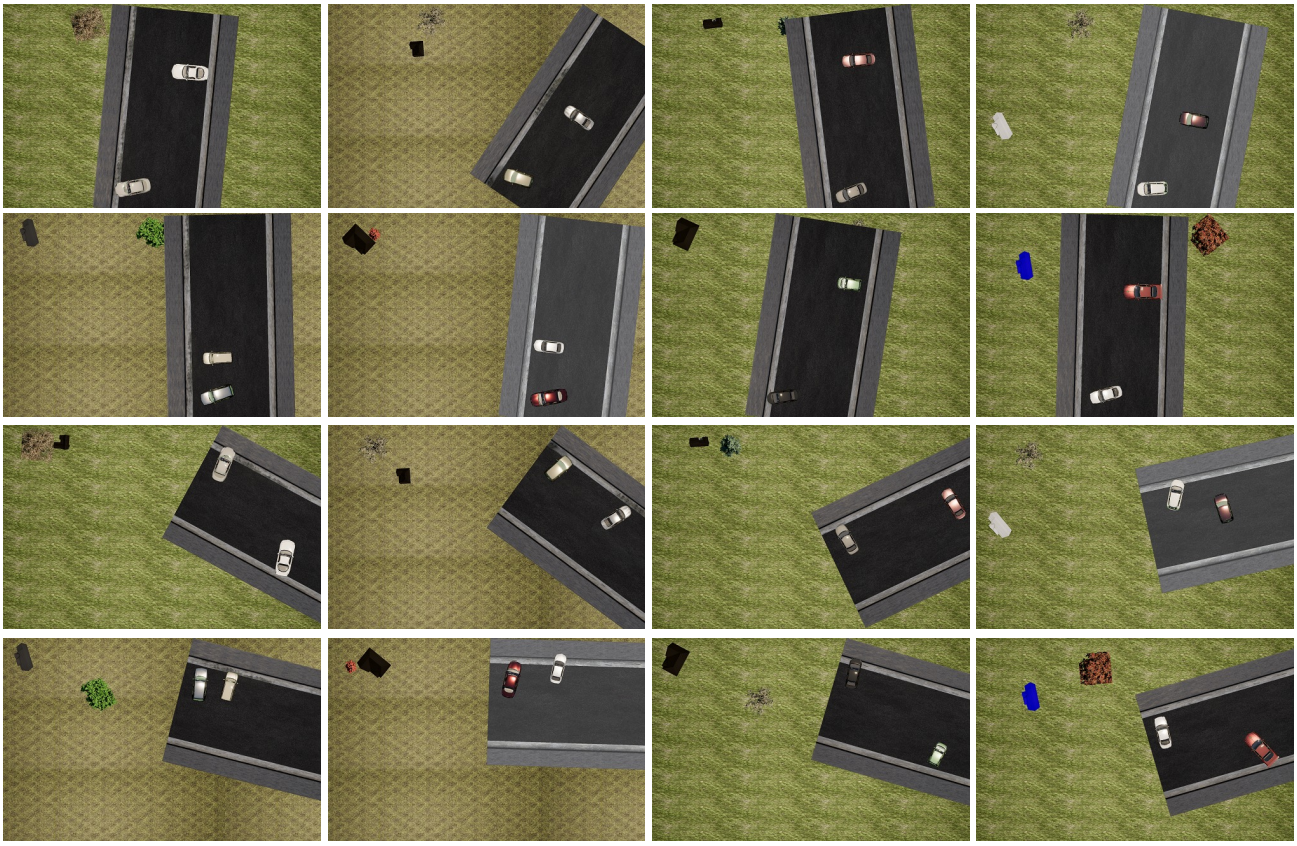

Figure 1. Meta-Sim generated samples **(top two rows)** and input scenes **(bottom two rows)**

1.2.3 Driving Scenes (3D)

For Driving Scenes (3D), 256 samples coming from the prob. grammar are considered to be 1 epoch.

Distribution Transformer. In this case, the autoencoder step is trained for 100 epochs using the adam optimizer with a batch size of 16 and learning rate 0.005. Next, the model is trained with the distribution loss for 40 epochs with the same batch size and learning rate 0.001, finally stopping at the best validation (in our case, performance on the 100 images taken from the training set) performance. Finally, the model is trained with the task loss for 8 epochs with the same learning rate.

Task Network. For task network, we use Mask-RCNN [1] with a FPN and Resnet-101 backbone. We pre-train the task network (with freely available data sampled from the probabilistic grammar), saving training time by not having to train the detector from scratch. We also do not train the task network from scratch every epoch. Rather, we resume training from the checkpoint from the previous epoch. We also heavily parallelize the rendering computation by working on multiple UE4 workers simultaneously and utilizing their batched computation features.

1.3. Additional Results

1.3.1 Aerial Views (2D)

Fig. 1 shows additional qualitative results for the Aerial 2D experiment, comparing the original samples to those obtained via Meta-Sim. The network learns to utilize the translation equivariance of convolutions and makes sufficient transformations to the objects for semantic segmentation to work on the validation data. Translating objects (such as cars) in the scene is still important, since our task-network for Aerial-2D has a maximum receptive field which is a quarter of the longest side of the image.

1.3.2 Driving Scenes (3D)

Fig 2 shows a few positive results for Meta-Sim. Our approach learns to rotate cars to look more like those in KITTI scenes and learns to place objects to avoid collisions between objects. The examples show that it can handle these even with a lot of objects present in the scene, while failing in some cases. It has also learnt to push cars out of the view of the camera when required. Sometimes, it positions two cars perfectly on top of each other, so that only one car is visible from the camera. This is in fact a viable solution since we solve for occlusion and truncation before generating ground truth bounding boxes. We also notice that the model modifies the camera height slightly, and moves context elements, usually densifying scenes.

Fig. 3 shows failure cases for Meta-Sim. Sometimes, the model does not resolve occlusions/intersections correctly, resulting in final scenes with cars intersection each other. Sometimes it places an extra car between two cars (top row), attempts a partial merge of two cars (second row), causes collision (third row) and fails to rotate cars in the driving lane (final row). In this experiment, we did not model the size of objects in the features used in the Distribution Transformer. This could be a major reason behind this, since different assets (cars, houses) have different sizes and therefore placing them at the same location is not enough to ensure perfect occlusion between them. Meta-Sim sometimes fails to deal with situations where the probabilistic grammar samples objects densely around a single location (Fig. 3). Another failure mode is when the model moves cars unrealistically close to the ego-car (camera) and when it overlaps buildings to create unrealistic composite buildings. The right column of fig. 4 show additional detection results on KITTI when the task-network is trained with images generated by Meta-Sim. The left column shows results when trained with samples from the probabilistic grammar. In general, we see see fewer false positives and negatives with Meta-sim. The results are also significant given that the task network has not seen any real KITTI images during training.

References

[1] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 3

[2] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*. 1

[3] A. Prakash, S. Boochoon, M. Brophy, D. Acuna, E. Cameracci, G. State, O. Shapira, and S. Birchfield. Structured domain randomization: Bridging the reality gap by context-aware synthetic data. In *arXiv:1810.10093*, 2018. 2

[4] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 2

[5] T. Yao, Y. Pan, Y. Li, and T. Mei. Exploring visual relationship for image captioning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 684–699, 2018. 2

Figure 2. **Successful cases**: **(left)** input scenes from the probabilistic grammar, **(right)** Meta-Sim's generated examples for the task of car detection on KITTI. Notice how meta-sim learns to align objects in the scene, slightly change the camera position and move context elements such as buildings and trees, usually densifying the scene.

Figure 3. **Failure cases**: **(left)** input scenes from the probabilistic grammar, **(right)** Meta-Sim's generated examples for the task of car detection on KITTI. Initially dense scenes are sometimes unresolved, leading to collisions in the final scenes. There are unrealistic colours on cars since they are sampled from the prior and not optimized in this work. In the last row, meta-sim moves a car very close to the ego-car (camera).

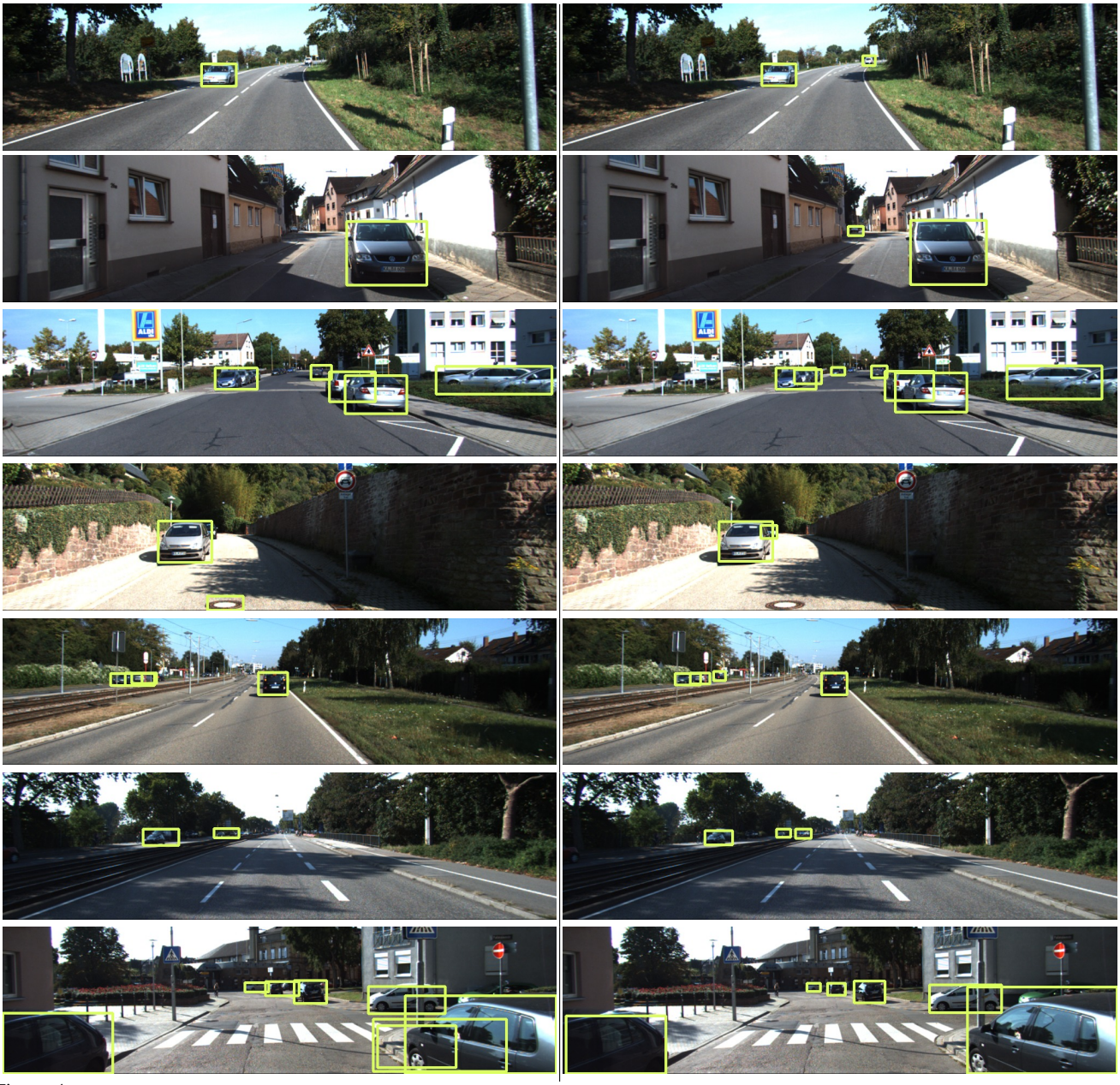

Figure 4. Car detection results with Mask-RCNN trained on **(right)** the dataset generated by Meta-Sim, and **(left)** the original dataset obtained by the probabilistic grammar. Notice how the models trained with meta-sim have lesser false positives and negatives.