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

Abstract

Training models to high-end performance requires availability of large labeled datasets, which are expensive to get. The goal of our work is to automatically synthesize *labeled datasets that are relevant for a downstream task. We propose Meta-Sim, which learns a generative model of synthetic scenes, and obtain images as well as its corresponding ground-truth via a graphics engine. We parametrize our dataset generator with a neural network, which learns to modify attributes of scene graphs obtained from probabilistic scene grammars, so as to minimize the distribution gap between its rendered outputs and target data. If the real dataset comes with a small labeled validation set, we additionally aim to optimize a meta-objective,* i.e. *downstream task performance. Experiments show that the proposed method can greatly improve content generation quality over a human-engineered probabilistic scene grammar, both qualitatively and quantitatively as measured by performance on a downstream task.*

1. Introduction

Data collection and labeling is a laborious, costly and time consuming venture, and represents a major bottleneck in most current machine learning pipelines. To this end, synthetic content generation [6, 36, 11, 34] has emerged as a promising solution since all ground-truth comes for free – via the graphics engine. It further enables us to train and test our models in virtual environments [38, 8, 48, 22, 41] before deploying to the real world, which is crucial for both scalability and safety. Unfortunately, an important performance issue arises due to the domain gap existing between the synthetic and real-world domains.

Addressing the domain gap issue has led to a plethora of work on synthetic-to-real domain adaptation [17, 27, 54, 10, 43, 34, 45]. These techniques aim to learn domain-invariant features and thus more transferrable models. One of the mainstream approaches is to learn to stylize synthetic images to look more like those captured in the real-world [17, 27, 51, 30, 18]. As such, these models address the *appearance gap* between the synthetic and real-world domains. They share the assumption that the domain gap is due to the differences that are fairly low level.

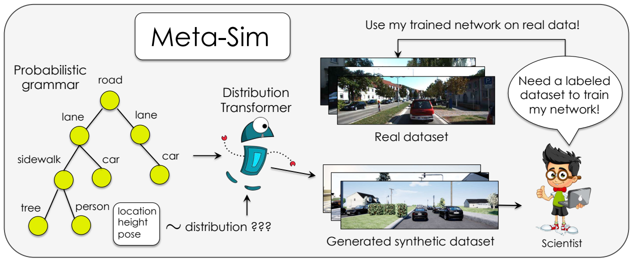

Figure 1. Meta-Sim is a method to generate synthetic datasets that bridge the *distribution gap* between real and synthetic data and are optimized for *downstream task performance*

Here, we argue that domain gap is also due to a *content gap*, arising from the fact that the synthetic content (*e.g.* layout and types of objects) mimics a limited set of scenes, not necessarily reflecting the diversity and distribution of objects of those captured in the real world. For example, the Virtual KITTI [11] dataset was created by a group of engineers and artists, to match object locations and poses in KITTI [13] which was recorded in Karlsruhe, Germany. But what if the target city changes to Tokyo, Japan, which has much heavier traffic and many more high-rise buildings? Moreover, what if the downstream task that we want to solve changes from object detection to lane estimation or rain drop removal? Creating synthetic worlds that ensure realism and diversity for any desired task requires significant effort by highly-qualified experts and does not scale to the fast demand of various commercial applications.

In this paper, we aim to learn a generative model of synthetic scenes that, by exploiting a graphics engine, produces *labeled* datasets with a content distribution matching that of imagery captured in the desired real-world datasets. Our *Meta-Sim* builds on top of probabilistic scene grammars which are commonly used in gaming and graphics to create diverse and valid virtual environments. In particular, we assume that the structure of the scenes sampled from the

*Correspondence to amlan@cs.toronto.edu, sfidler@nvidia.com grammar are correct (*e.g.* a driving scene has a road and cars), and learn to modify their attributes. By modifying locations, poses and other attributes of objects, Meta-Sim gains a powerful flexibility of adapting scene generation to better match real-world scene distributions. *Meta-Sim* also optimizes a meta objective of adapting the simulator to improve downstream real-world performance of a Task Network trained on the datasets synthesized by our model. Our learning framework optimizes several objectives using approximated gradients through a non-differentiable renderer.

We validate our approach on two toy simulators in controlled settings, where Meta-Sim is shown to excel at bridging the distribution gaps. We further showcase Meta-Sim on adapting a probabilistic grammar akin to SDR [34] to better match a real self-driving dataset, leading to improved content generation quality, as measured by sim-to-real performance. To the best of our knowledge, Meta-Sim is the first approach to enable dataset and task specific synthetic content generation, and we hope that our work opens the door to more adaptable simulation in the future.

2. Related Work

Synthetic Content Generation and Simulation. The community has been investing significant effort in creating high-quality synthetic content, ranging from driving scenes [38, 11, 36, 8, 34, 47, 2], indoor scenes [48, 50, 33], household robotics [35, 22], robotic control [44], game playing [5], optical flow estimation [6, 23], and quadcopter control and navigation [41]. While such environments are typically very realistic, they require qualified experts to spend a huge amount of time to create them. Domain Randomization (DR) is a cheaper alternative to such photo-realistic simulation environments [40, 43, 34]. The DR technique generates a large amount of diverse scenes by inserting objects in random locations and poses. As a result, the distribution of the synthetic scenes is very different to that of the real world scenes. We, on the other hand, aim to align the synthetic and real distributions through a direct optimization on the attributes and through a meta objective of optimizing for performance on a down-stream task.

Procedural modeling and probabilisic scene grammars are an alternative approach to content generation, which are able to produce worlds at the scale of full cities[1], and mimic diverse 3D scenes for self-driving[2]. However, the parameters for generating the distributions that control how a scene is generated need to be manually specified. This is not only tedious but also error-prone. There is no guarantee that the specified parameters can generate distributions that faithfully reflect real world distributions. [24, 32] use such probabilistic programs to invert the generative process and infer a program given an image, while we aim to learn the generative process itself from real data.

Domain Adaptation aims at addressing the gap between the distribution of data used to train and test or deploy the model. From synthetic to real, two kinds of domain gaps arise: the appearance (style) gap and the content (layout) gap. Most existing work [17, 27, 54, 10, 51, 30, 18] tackle the former by using image-to-image translation to transform the appearance distribution of the synthetic images to look more like that of the real images. Others [17, 27] add additional task based constraints to ensure that the layout of the stylized images remain the same. Other techniques use pseudo-label based learning [54] and student-teacher networks [10] for domain adaptation. Our work is an early attempt to tackle the latter *i.e.* the content gap. We note that the appearance gap is orthogonal to the content gap, and prior art could be directly plugged into our method.

Optimizing Simulators. [31] also attempt to optimize non-differentiable simulators using a variational upper-bound of a GAN-like objective to produce samples representative of a target distribution. We, on other hand, use the MMD [15] distance for comparing distributions and also optimize a meta objective to produce samples suitable for a downstream task. [7] learn to optimize simulator parameters for robotic control tasks, where trajectories between the real and simulated robot can be directly compared. [39] optimize high level exposed parameters by optimizing for downstream task performance using Reinforcement Learning. We, however, optimize low level scene parameters (at the level of every object) while also learning to match distributions and optimizing downstream task performance. [12] attempt to synthesize images by learning to generate even lower-level programs (at the level of brush strokes) that a graphics engine can interpret to generate realistic looking images, as measured by a trained discriminator. [46] model scene generation using a low dimensional space (imposing stronger restrictions) and a discriminator for estimating the likelihood of scenes. We, however, explicitly model the graphical structure in scenes, have fewer constraints on possible generated scenes and can theoretically optimize all parameters for all objects in one sampled scene as compared to only a few exposed low dimensional parameters, while also explicitly modeling a downstream task.

3. Meta-Sim

In this section, we introduce *Meta-Sim*. Given a dataset of real imagery X_R and a task T (*e.g.* object detection), our goal is to *synthesize* a training dataset $D_T = (X_T, Y_T)$ with X_T synthesized imagery that resembles the given real imagery, and Y_T the corresponding ground-truth for task T. To simplify notation, we omit subscript T from here on.

We parametrize data synthesis with a neural network, *i.e.* $D(\theta) = (X(\theta), Y(\theta))$. Our goal in this paper is to learn the parameters θ such that the distribution of $X(\theta)$ matches that

[1] https://www.esri.com/en-us/arcgis/products/esri-cityengine/overview
[2] https://www.paralleldomain.com/

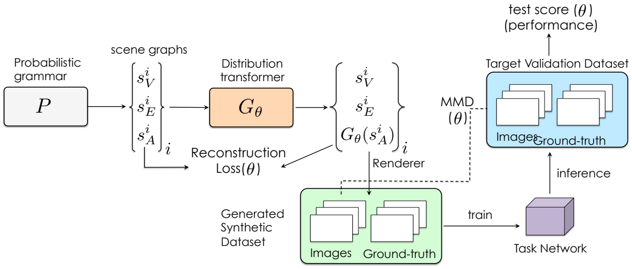

Figure 2. Overview of Meta-Sim: The goal is to learn to transform samples coming from a probabilistic grammar with a distribution transformer, aiming to minimize the *distribution gap* between simulated and real data and maximize *sim-to-real performance*

of X_R (real imagery). Optionally, if the real dataset comes with a small validation set V that is labeled for task T, we additionally aim to optimize a meta-objective, *i.e*. downstream task performance. The latter assumes we also have a trainable task solving module (*i.e*. another neural network), the performance of which we want to maximize by training it on our generated training data. We refer to this module as a Task Network, which will be treated as a black box in our work. Note that Meta-Sim has parallels to Neural Architecture Search [53], where our search is over the input datasets to a fixed neural network instead of a search over the neural network architecture given fixed data.

Image Synthesis vs Rendering. Generative models of pixels have only recently seen success in generating realistic high resolution images [4, 19]. Extracting task specific ground-truth (eg: segmentation) from them remains a challenge. Conditional generative models of pixels condition on input images and transform their appearance, producing compelling results. However, these methods assume ground truth labels remain unchanged, and thus are limited in their *content* (structural) variability. In Meta-Sim we aim to *learn* a **generative model of synthetic 3D content**, and obtain D via a graphics engine. Since the 3D assets come with semantic information (*i.e*., we know an asset is a *car*), compositing or modifying the synthetic scenes will still render perfect ground-truth. The main challenge is to learn the 3D scene composition by optimizing solely the distribution mismatch of rendered with real imagery. The following subsections layout Meta-Sim in detail and are structured as follows: Sec. 3.1 introduces the representation of parametrized synthetic worlds, while Sec. 3.2 describes our learning framework.

3.1. Parametrizing Synthetic Scenes

Scene Graphs are a common way to represent 3D worlds in gaming/graphics. A scene graph represent elements of a scene in a concise hierarchical structure, with each element having a set of attributes (eg. class, location, or even the id of a 3D asset from a library) (see Fig. 3). The hierarchy defines parent-child dependencies, where the attributes of the child elements are typically defined relative to the parent's, allowing for an efficient and natural way to create and modify scenes. The corresponding image and pixel-level annotations can be rendered easily by placing objects as described in the scene graph.

In order to generate *diverse* and *valid* 3D worlds, the typical approach is to specify the generative process of the graph by a *probabilistic scene grammar* [52]. For example, to generate a traffic scene, one might first lay out the centerline of the road, add parallel lanes, position aligned cars on each lane, etc. The structure of the scene is defined by the grammar, while the attributes are typically sampled from parametric distributions, which require careful tuning.

In our work, we assume access to a probabilistic grammar from which we can sample initial scene graphs. We assume the *structure* of each scene graph is correct, *i.e*. the driving scene has a road, sky, and a number of objects. This is a reasonable assumption, given that inferring structure (inverse graphics) is known to be a hard problem. Our goal is to modify the *attributes* of each scene graph, such that the transformed scenes, when rendered, will resemble the distribution of the real scenes. By modifying the attributes, we give the model a powerful flexibility to change objects' locations, poses, colors, asset ids, etc. This amounts to learning a conditional generative model, which, by conditioning on an input scene graph transforms its node attributes. In essence, we keep the *structure* generated by the probabilistic grammar, but transform the distribution of the *attributes*. Thus, our model acts as a *Distribution Transformer*.

Notation. Let P denote the probabilistic grammar from which we can sample scene graphs $s \sim P$. We denote a single scene graph s as a set of vertices s_V, edges s_E and attributes s_A. We have access to a renderer R, that can take in a scene graph s and generate the corresponding image and ground truth, $R(s) = (x, y)$. Let G_θ refer to our Distribution Transformer, which takes an input scene graph s and outputs a scene graph $G_\theta(s)$, with transformed attributes but the same structure, *i.e*. $G_\theta(s = [s_V, s_E, s_A])$ = $[s_V, s_E, G_\theta(s_A)]$. Note that by sampling many scene graphs, transforming their attributes, and rendering, we obtain a synthetic dataset $D(\theta)$.

Architecture of G_θ. Given the graphical structure of scene graphs, modeling G_θ via a Graph Neural Network is a natural choice. In particular, we use Graph Convolutional Networks (GCNs) [21]. We follow [49] and use a graph convolutional layer that utilizes two different weight matrices to capture top-down and bottom-up information flow separately. Our model makes per node predictions *i.e*. generates transformed attributes $G_\theta(s_A)$ for each node in s_V.

Mutable Attributes: We input to G_θ all attributes s_A, but we might want to only modify specific attributes and trust the probabilistic grammar P on the rest. For example, in Fig. 3 we may not want to change the heights of houses,

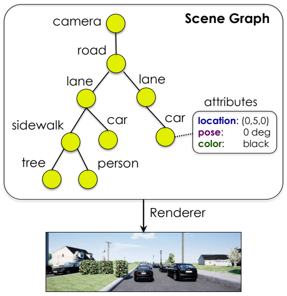

Figure 3. Simple scene graph example for a driving scene.

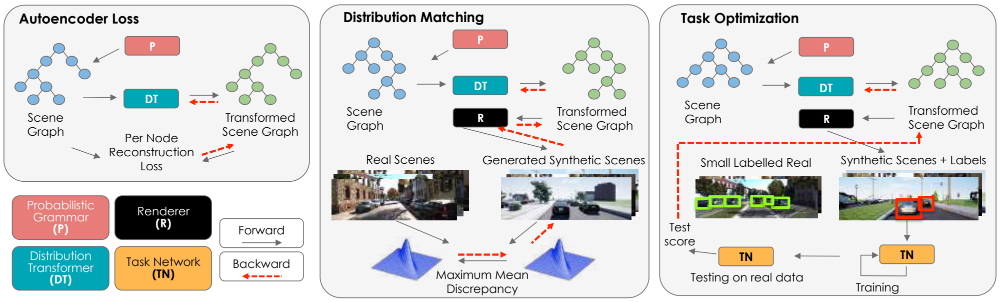

Figure 4. Illustration of different losses used in *Meta-Sim*, including forward and backward pass control flow for each step. We indicate transformed attributes of a scene graph by changing colors of the nodes.

or width of the sidewalks, if our final task is car detection. This reduces the number of exposed parameters our model is tasked to tune thus improving training time and complexity. Therefore, in the subsequent parts, we assume we have a subset of attributes per node $v \in s_V$ which are mutable (modifiable), denoted by $s_{A,mut}(v)$. From here onwards, it is assumed that only the mutable attributes in $s_{A,mut}(v) \forall v$ are changed by G_θ; others remain the same as in s.

3.2. Training Meta-Sim

We now introduce our learning framework. Since our learning problem is very hard and computationally intensive, we first pre-train our model using a simple autoencoder loss in Sec. 3.2.1. The distribution matching loss is presented in Sec 3.2.2, while meta-training is described in Sec 3.2.3. The overview of our model is given in Fig. 2, with the particular training objectives illustrated in Fig. 4.

3.2.1 Pre-training: Autoencoder Loss

A probabilistic scene grammar P represents a prior on how a scene should be generated. Learning this prior is a natural way to pre-train our *Distribution Transformer*. This amounts to training G_θ to perform the identity function *i.e.* $G_\theta(s) = s$. The input feature of each node is its attribute set (s_A), which is defined consistently across all nodes (see suppl.). Since s_A is composed of different categorical and continuous components, appropriate losses are used per feature component when training to reconstruct (*i.e.* cross-entropy loss for categorical attributes, and L1 loss for continuous attributes). We find pre-training to be crucial, and convergence during this stage strongly affects performance in the following training steps.

3.2.2 Distribution Matching

The first objective of training our model is to bring the distribution of the rendered images to be closer to the distribution of real imagery X_R. The Maximum Mean Discrepancy (MMD) [15] metric is a frequentist measure of the similarity of two distributions and has been used for training generative models [9, 29, 26] to match statistics of the generated distribution with the target distribution. An alternative, adversarial learning with discriminators, however, is known to suffer from mode collapse, and a general instability in training. Pixel-wise generative models with MMD have usually suffered from not being able to model high-frequency signals (resulting in blurry generations). Since our generative process goes through a renderer, we sidestep the issue altogether, and thus choose MMD for training stability.

We compute MMD in the feature space of an InceptionV3 [42] network (known as Kernel Inception Distance (KID) [3]) with a gaussian kernel $k(x_i, x_j)$. This feature extractor is denoted by the function ϕ. We refer the reader to [29] for more details. The *Distribution Matching* box in Fig. 4 depicts the training procedure. Specifically, given scene graphs $s_1, ..., s_N$ sampled from P and target real images X_R, the squared MMD distance can be computed as,

$$\begin{aligned}\mathcal{L}_{MMD^2} = & \frac{1}{N^2}\sum_{i=1}^{N}\sum_{i'=1}^{N} k(\phi(X_\theta(s_i)), \phi(X_\theta(s_{i'}))) \\ & + \frac{1}{M^2}\sum_{j=1}^{M}\sum_{j'=1}^{M} k(\phi(X_R^j), \phi(X_R^{j'})) \\ & - \frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M} k(\phi(X_\theta(s_i)), \phi(X_R^j)) \end{aligned} \tag{1}$$

where the image rendered from s is $X_\theta(s) = R(G_\theta(s)))$. Empirically, we found using lower layers of the Inception network helps ameliorate domain adaptation issues that arise in MMD computation with Inception features, due to one set of images being real while the other is rendered.

Backprop through a Renderer. We backpropagate through the non-differentiable rendering function R by approximating the gradient of $R(G_\theta(s))$ w.r.t. $G_\theta(s)$ using finite differences[3]. While this gives us noisy gradients, we found it sufficient to be able to train our models in practice, with the benefit of being able to use photorealistic rendering. We note that recent work on differentiable rendering [20, 28] could potentially benefit this work.

3.2.3 Optimizing Task Performance

The second objective of training the model G_θ is to generate data $R(G_\theta(S))$ given samples $S = \{s_1, ..., s_K\}$ from

[3] computed by perturbing each mutable attribute of each object in the predicted scene graph $G_\theta(s)$

Algorithm 1 Pseudocode for Meta-Sim's meta training phase

```
Given: $P, R, G_\theta$                    ▷ Probabilistic grammar, Renderer, GCN Model
Given: TaskNet, $X_R, V$                   ▷ Task Model, Real Images, Target Validation Data
Hyperparameters: $E_m, I_m, B_m$           ▷ Epochs, Iters, Batch size
while $e_m \leq E_m$ do                    ▷ Meta training
    $loss = 0$;
    $data$ = []; $samples$ = [];           ▷ Caching data & samples generated in epoch
    while $i_m \leq I_m$ do
        $S = G_\theta(\text{sample}(P, B_m))$;     ▷ Generate $B_m$ samples from $P$ and transform them
        $D = R(S)$;                        ▷ Render images, labels from $S$
        $data$ += $D$; $samples$ += $S$;
        $loss$ += $\mathcal{L}_{MMD^2}(D, X_R)$;   ▷ MMD between generated and target real images
    end while
    TaskNet = train(TaskNet, $data$);      ▷ Train TaskNet on $data$
    $score$ = test(TaskNet, $V$);          ▷ Test TaskNet on target val
    $loss$ += $-(score - moving\_avg(score)) \cdot \log p_{G_\theta}(samples)$   ▷ Eq. 3
    $G_\theta$ = optimize($G_\theta$, $loss$);     ▷ SGD step
end while
```

the probabilistic grammar P, such that a model trained on this data achieves best performance when tested on target data V. This can be interpreted as a meta-objective, where the input data must be optimized to improve accuracy on a validation set. We introduce a task network `TaskNet` to train using our data and to measure validation performance on. We train G_θ under the following objective,

$$\max_{\theta} \quad \mathbb{E}_{S' \sim G_\theta(S)} \left[\texttt{score}(S') \right] \tag{2}$$

where $\texttt{score}(S')$ is the performance metric achieved on validation data V after training `TaskNet` on data $R(G_\theta(S'))$. The task loss in Eq. 2 is not differentiable w.r.t the parameters θ, since the score is measured using validation data and not S'. We use the REINFORCE score function estimator (which is an unbiased estimator of the gradient) to compute the gradients of Eq. 2. Reformulating the objective as a loss and writing the gradient gives,

$$\begin{aligned} \mathcal{L}_{task} &= -\mathbb{E}_{S' \sim G_\theta(S)} \left[\texttt{score}(S') \right] \\ \nabla_\theta \mathcal{L}_{task} &= -\mathbb{E}_{S' \sim G_\theta(S)} \left[\texttt{score}(S') \times \nabla_\theta \log p_{G_\theta}(S') \right] \end{aligned} \tag{3}$$

To reduce the variance of the gradient from the estimator above, we keep track of an exponential moving average of previous scores and subtract it from the current score [14]. We approximate the expectation using one sample from $G_\theta(S)$. The *Task Optimization* box in Fig. 4 provides a pictorial overview of the task optimization.

Sampling from $\mathrm{G}_\theta(\mathrm{s})$**.** Eq. 3 requires us to be able to sample (and measure its likelihood) from our model. For continuous attributes, we interpret our model to be predicting the mean of a normal distribution per attribute, with a pre-defined variance. We use the reparametrization trick to sample from this normal distribution. For categorical attributes, it is possible to sample from a multinomial distribution from the predicted log probabilities per category. In this paper, we keep categorical attributes immutable.

Calculating $\log \mathrm{p}_{\mathrm{G}_\theta}(\mathrm{S}')$**.** Since we assume independence across scenes, attributes and objects in the scene, the likelihood in Eq 3 for the full scene is simply factorizable,

$$\log p_{G_\theta(S')} = \sum_{s' \in S'} \sum_{v \in s'_V} \sum_{a \in s'_{A,mut}(v)} \log p_{G_\theta}(s'(v,a)) \tag{4}$$

where $s'(v,a)$ represents the attribute a at node v in a single scene s' in batch S'. Note that the sum is only over mutable attributes per node $s_{A,mut}(v)$. The individual log probabilities come from the defined sampling procedure.

Training Algorithm. The algorithm for training with Distribution Matching and Task Optimization is presented in Algorithm 1.

4. Experiments

We evaluate Meta-Sim on three target datasets with three different tasks. The subsequent sections follow a general structure where we first outline the desired task, the target data and the task network[4]. Then, we describe the probabilistic grammar that the *Distribution Transformer* utilizes for its input, and the associated renderer that generates labeled synthetic data. Finally, we show quantitative and qualitative results after training the task network using synthetic data generated by Meta-Sim. We observe boosts in quantitative performance and noticeable qualitative improvements in content-generation quality.

The first two experiments presented are in a controlled setting, each with increasing complexity. The aim here is to probe Meta-Sim's capabilities when the shift between the target data distribution and the input distribution is known. The input distribution refers to the distribution of the scenes generated by samples from the probabilistic grammar that our *Distribution Transformer* takes as input. Target data for these tasks is created by carefully modifying the parameters of the probabilistic program, which represents a known distribution gap that the model must learn.

4.1. MNIST

We first evaluate our approach on digit **classification** on MNIST-like data. The probabilistic grammar samples a background texture, one digit texture (image) from the MNIST dataset [25] (which has an equal probability for any digit), and then samples a rotation and location for the digit. The renderer transforms the texture based on the sampled transformation and pastes it onto a canvas.

Task Network. Our task network is a small 2-layer CNN followed by 3 fully connected layers. We apply dropout in the fully connected layers (with 50, 100 and 10 features). We verify that this network can achieve greater than 99% accuracy on the regular MNIST classification task. We do not use data-augmentation while training (in all following

[4] Task Network training details in suppl. material

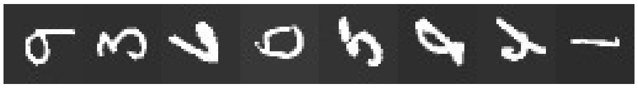

Figure 5. Examples from the rotated-MNIST dataset

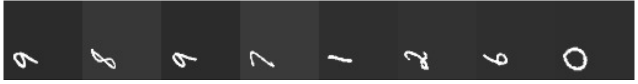

Figure 6. Examples from the rotated and translated MNIST

experiments as well), as it might interfere with our model's training by changing the configuration of the generated data, making the task optimization signal unreliable.

Rotating MNIST. In our first experiment, the probabilistic grammar generates input samples that are upright and centered, like regular MNIST digits (Fig 7 bottom). The target data V and X_R are images (at 32×32 resolution) where digits centered and always rotated by 90 degrees (Fig 5). Ideally, the model will learn this exact transformation, and rotate the digits in the input scene graph while keeping them in the same centered position.

Rotating and Translating MNIST. For the second experiment, we additionally add translation to the distribution gap, making the task harder for Meta-Sim. We generate V and X_R as 1000 images (at 64×64 resolution) where in addition to being rotated by 90 degrees, the digits are moved to the bottom left corner of the canvas (Fig 6). The input probabilistic grammar remains the same, *i.e.* one that generates centered and upright digits (Fig. 8 bottom).

Quantitative Results. Table 1 shows classification on the target datasets with the two distribution gaps described above. The target datasets are fresh samples from the target distribution (separate from V). Training directly on the input scenes (coming from the input probabilistic grammar *i.e.* generating upright and centered digits in this case) results in just above random performance. Our model recovers the transformation causing the distribution gap, and achieves greater than 99% classification accuracy.

Data	Rotation	Rotation + Translation
Prob. Grammar	14.8	13.1
Meta-Sim	**99.5**	**99.3**

Table 1. Classification performance on our MNIST with different distribution gaps in the data

Qualitative Results. Fig. 7 and Fig. 8 show generations from our model at the end of training, and compares with the input scenes. Clearly, the model has learnt to perfectly transform the input distribution to replicate the target distribution, corroborating our quantitative results.

4.2. Aerial Views (2D)

Next, we evaluate our approach on **semantic segmentation** of simulated aerial views of simple roadways. In the probabilistic grammar, we sample a background grass texture, followed by a (straight) road at some location and rotation on the background. Next, we sample two cars with independent locations (constrained to be in the road by parametrizing in the road's coordinate system), and rotations. In addition, we also sample a tree and a house randomly in the scene. Each object in the scene gets a random texture from a set of textures we collected for each object. We ended up with nearly 600 car, 40 tree, 20 house, 7 grass and 4 road textures. Overall, this grammar has more complexity than MNIST, due to the scene graphs having higher depth, more objects, and variability in appearance.

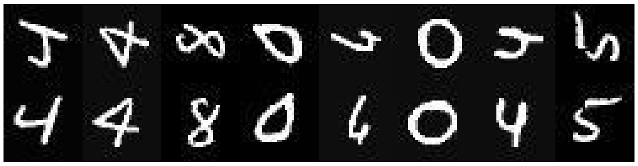

Figure 7. **(bottom)** Input scenes, **(top)** Meta-Sim's generated examples for MNIST with rotation gap

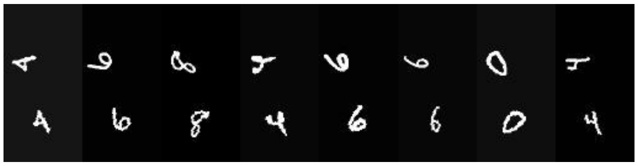

Figure 8. **(bottom)** Input scenes, **(top)** Meta-Sim's generated examples for MNIST with rotation and translation gap

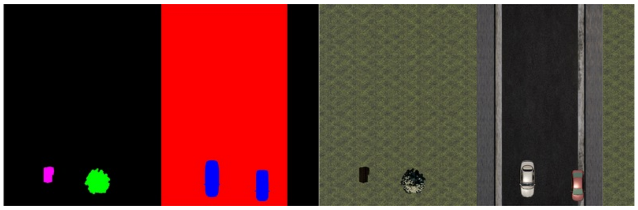

Figure 9. Example label and image from Aerial2D validation

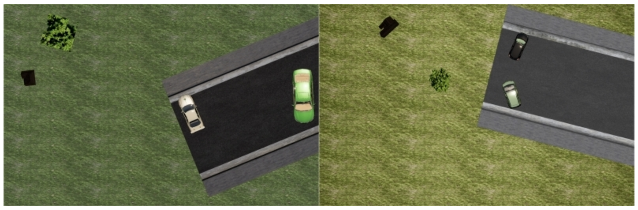

Figure 10. Example input scenes for Aerial2D

V and X_R are created by tuning the grammar parameters to generate a realistic aerial view. (Fig. 9). The input probabilistic grammar uses random parameters (Fig. 10) bottom.

Task Network. We use a small U-Net architecture [37] with a total of 7 convolutional layers (with 16 to 64 filters in the convolution layers) as our task-network.

Quantitative Results. Table 2 shows semantic segmentation results on the target set. The results show that Meta-Sim effectively transforms the outputs of the probabilistic grammar, even in this relatively more complex setup, and improves the mean IoU. Specifically, it learns to drastically reduce the gap in performance for cars and also improves

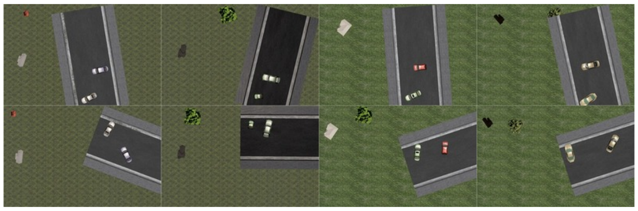

Figure 11. **(bottom)** input scenes, **(top)** Meta-Sim's generated examples for Aerial semantic segmentation

Data	Car	Road	House	Tree	Mean
Prob. Grammar	30.0	93.1	**98.3**	**99.7**	80.3
MetaSim	**86.7**	**99.6**	95.0	99.5	**95.2**

Table 2. Semantic segmentation results (IoU) on Aerial2D

performance on roads.

Qualitative Results. Qualitative results in Fig. 11 show that the model indeed learns to exploit the convolutional structure of the task network, by only learning to orient. This is sufficient to achieve its job since convolutions are translation equivariant, but not rotation equivariant.

4.3. Driving Scenes (3D)

After validating our approach on controlled experiments in a simulated setting, we now evaluate our approach for **object detection** on the challenging KITTI [13] dataset. KITTI was captured with a camera mounted on top of a car driving around the city of Karlsruhe in Germany. It consists of challenging traffic scenarios and scenes ranging from highways to urban to more rural neighborhoods. Contrary to the previous experiments, the distribution gap which we wish to reduce arises naturally here.

Current open-source self driving simulators [8, 41] do not offer the amount of low level control on object attributes that we require in our model. We thus turn to probabilistic grammars for road scenarios [34, 47]. Specifically, SDR [34] is a road scene grammar that has been shown to outperform existing synthetic datasets as measured by sim-to-real performance. We adopt a simpler version of SDR and implement portions of their grammar as our probabilistic grammar. Specifically, we remove support for intersections and side-roads for computational reasons. The exact parameters of the grammar used can be found in the supplementary material. We use the Unreal Engine 4 (UE4) [1] game engine for the 3D rendering from scene graphs. Fig. 12(left column) shows example renderings of scenes generated using our version of the SDR grammar. The grammar parameters were mildly tuned, since we aim to have our model do the heavy lifting in subsequent parts.

Task Network. We use Mask-RCNN [16] with a Resnet-50-FPN backbone (ImageNet initialized) detection head as our task network for object detection.

Experimental Setup. Following SDR [34], we use car detection as our task. Validation data V is formed by taking 100 random images (and their labels) from the KITTI train set. The rest of the training data (images only) forms X_R. We report results on the KITTI val set. Training and finer details can be found in the supplementary material.

Complexity. To reduce training complexity (coming from rendering and numerical gradients), we train Meta-Sim to optimize specific parts of the scene sequentially. We first train to optimize attributes of cars. Next, we optimize car and camera parameters, and finally add parameters of context elements (buildings, pedestrians, trees) together to the training. Similarly, we decouple distribution and task training. We first train the above with MMD, and finally optimize all parameters above with the meta task loss. The computation of the Jacobian through the renderer in our method is expensive (250 - 900 seconds for a batch of size 16) for large scene graphs in the 3D driving simulator, but we find that the abstraction into scene graphs exhibits decently fast convergence (4-5 hours on convergence of cars and 72 hours for all the training steps in Table 3) on one TITAN Xp GPU with the rendering also running on one TITAN Xp GPU.

Quantitative Results. Table 3 reports the average precision at 0.5 IoU of the task network trained using data generated from different methods, when tested on the KITTI val set. We see that training with Meta-Sim beats just using the data from the probabilistic grammar.

Data	Easy	Moderate	Hard
Prob. Grammar	63.7	63.7	62.2
MetaSim (Cars)	66.4	**66.5**	65.6
+ Camera	65.9	66.3	65.9
+ Context	65.9	66.3	66.0
+ Task Loss	**66.7**	66.3	**66.2**

Table 3. AP @ 0.5 IOU for car detection on the KITTI val dataset

Training the task network online with meta-sim and offline on final generated data results in similar final detection performance. This ensures the quality of the final generated data, since training while the transformation of data is being learned could be seen as data augmentation.

Bridging the appearance gap. We additionally add a state-of-the-art image-to-image translation network, MUNIT [18] after training our model to attempt to bridge the appearance gap between the generated synthetic images and real images. Table 4 shows training with image-to-image translation still leaves a performance gap between MetaSim and the baseline, confirming our *content gap* hypothesis.

Data	Easy	Moderate	Hard
Prob. Grammar	71.1	**75.5**	65.3
Meta-Sim	**77.5**	75.1	**68.2**

Table 4. Effect of adding image-to-image translation to bridge the appearance gap in generated images

Training on V. Since we have access to some labelled training data, a valid baseline is to train the models on V

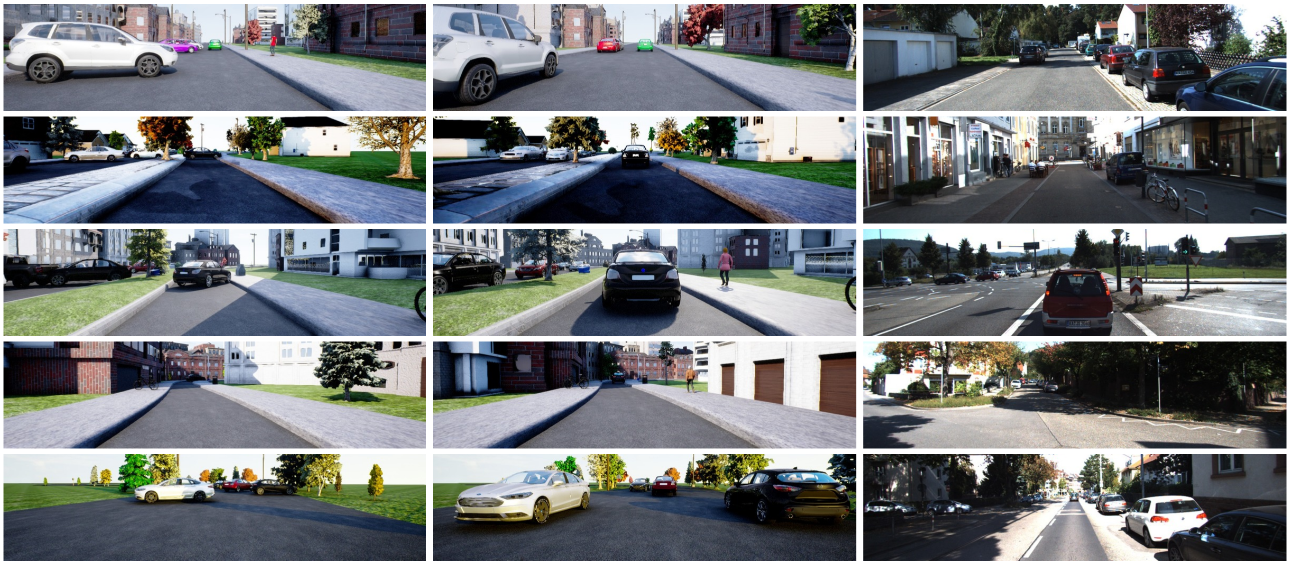

Figure 12. **(left)** samples from our prob. grammar, **(middle)** Meta-Sim's corresponding samples, **(right)** random samples from KITTI

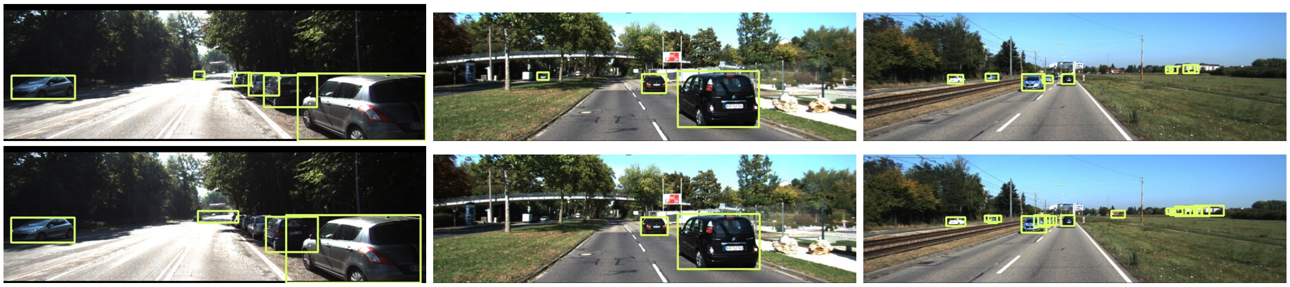

Figure 13. Car detection results **(top)** of task network trained with Meta-Sim vs **(bottom)** trained with our prob. grammar

(100 images from KITTI train split). In Table. 5 we show the effect of only training with V and finetuning using V.

TaskNet Initialization	Easy	Moderate	Hard
ImageNet	61.2	62.0	60.7
Prob. Grammar	71.3	72.7	72.7
Meta-Sim (Task Loss)	**72.4**	**73.9**	**73.9**

Table 5. Effect of finetuning on V

Qualitative Results. Fig. 12 shows a few outputs of Meta-Sim compared to the inputs sampled from the grammar, alongwith a few random samples from KITTI(train). There is a noticeable difference, as Meta-Sim's cars are well aligned with the road, and the distances between cars are meaningful. Also notice the small changes in camera, and the differences in the context elements, including houses, trees and pedestrians. The last row in Fig. 12 represents a failure case where Meta-Sim is unable to clear up a dense initial scene, resulting in collided cars. Interestingly, Meta-Sim perfectly overlaps two cars in the same image such that a single car is visible from the camera (first car in front of camera). This behavior is seen multiple times, indicating that the model learns to cheat its way to good data. Elements are moved to final configurations sequentially, following our training procedure. We remind the reader that these scene configurations are learned with only image/task level supervision. In Fig. 13, we show results of training the task network on our grammar vs. training with Meta-Sim. We observe fewer false positives and negatives than the baseline. Meta-Sim shows better recall and GT overlap. Both models lose in precision, arguably because of not training for similar classes like Bus/Truck which would be negative examples.

5. Conclusion

We proposed Meta-Sim, an approach that generates synthetic data to match real content distributions while optimizing performance on downstream (real) tasks. Our model learns to transform sampled scenes from a probabilistic grammar so as to satisfy these objectives. Experiments on two toy and one real task showcased that Meta-Sim generates quantitatively better and noticeably higher quality samples than the baseline. We hope this opens a new exciting direction for simulation in the computer vision community. Like any other method, it has its limitations. It relies on obtaining valid scene structures from a grammar, and hence is still limited in the kinds of scenes it can model. Inferring rules of the grammar from real images, learning to generate structure of scenes and introducing multimodality in the model are intriguing avenues for future work.

Acknowledgements: The authors would like to thank Shaad Boochoon, Felipe Alves, Gavriel State, Jean-Francois Lafleche, Kevin Newkirk, Lou Rohan, Johnny Costello, Dane Johnston and Rev Lebaredian for their help and support throughout this project.

References

[1] https://www.unrealengine.com/. 7

[2] Hassan Abu Alhaija, Siva Karthik Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer Vision*, 126(9):961–972, 2018. 2

[3] Mikołaj Bińkowski, Dougal J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. *ICLR*, 2018. 4

[4] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 3

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. In *arXiv:1606.01540*, 2016. 2

[6] Daniel J. Butler, Jonas Wulff, Garrett B. Stanley, and Michael J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *ECCV*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, 2012. 1, 2

[7] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *arXiv preprint arXiv:1810.05687*, 2018. 2

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *CORL*, pages 1–16, 2017. 1, 2, 7

[9] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. In *UAI*, 2015. 4

[10] Geoffrey French, Michal Mackiewicz, and Mark Fisher. Self-ensembling for visual domain adaptation. In *ICLR*, 2018. 1, 2

[11] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, 2016. 1, 2

[12] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*, 2018. 2

[13] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *CVPR*, 2012. 1, 7

[14] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *JMLR*, 5(Nov):1471–1530, 2004. 5

[15] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *JMLR*, 2012. 2, 4

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 7

[17] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isol, Kate Saenko Alexei A. Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *ICML*, 2018. 1, 2

[18] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *ECCV*, 2018. 1, 2, 7

[19] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2018. 3

[20] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *CVPR*, 2018. 4

[21] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 3

[22] Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. In *arXiv:1712.05474*, 2017. 1, 2

[23] Philipp Krähenbühl. Free supervision from video games. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2

[24] Tejas D Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum, and Vikash Mansinghka. Picture: A probabilistic programming language for scene perception. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 4390–4399, 2015. 2

[25] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*. 5

[26] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. In *NIPS*, 2017. 4

[27] Peilun Li, Xiaodan Liang, Daoyuan Jia, and Eric P. Xing. Semantic-aware grad-gan for virtual-to-real urban scene adaption. In *BMVC*, 2018. 1, 2

[28] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 2018. 4

[29] Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *ICML*, 2015. 4

[30] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In *NIPS*, 2017. 1, 2

[31] Gilles Louppe and Kyle Cranmer. Adversarial variational optimization of non-differentiable simulators. *arXiv preprint arXiv:1707.07113*, 2017. 2

[32] Vikash K Mansinghka, Tejas D Kulkarni, Yura N Perov, and Josh Tenenbaum. Approximate bayesian image interpretation using generative probabilistic graphics programs. In *Advances in Neural Information Processing Systems*, pages 1520–1528, 2013. 2

[33] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J Davison. Scenenet rgb-d: 5m photorealistic images of synthetic indoor trajectories with ground truth. *arXiv preprint arXiv:1612.05079*, 2016. 2

[34] Aayush Prakash, Shaad Boochoon, Mark Brophy, David Acuna, Eric Cameracci, Gavriel State, Omer Shapira, and Stan Birchfield. Structured domain randomization: Bridging the reality gap by context-aware synthetic data. In *arXiv:1810.10093*, 2018. 1, 2, 7

[35] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *CVPR*, 2018. 2

[36] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *ECCV*, 2016. 1, 2

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, pages 234–241. Springer, 2015. 6

[38] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016. 1, 2

[39] Nataniel Ruiz, Samuel Schulter, and Manmohan Chandraker. Learning to simulate. *arXiv preprint arXiv:1810.02513*, 2018. 2

[40] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016. 2

[41] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Aerial Informatics and Robotics platform. Technical Report MSR-TR-2017-9, Microsoft Research, 2017. 1, 2, 7

[42] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 4

[43] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, 2017. 1, 2

[44] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intl. Conf. on Intelligent Robots and Systems*, 2012. 2

[45] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. Learning to adapt structured output space for semantic segmentation. In *CVPR*, 2018. 1

[46] VSR Veeravasarapu, Constantin Rothkopf, and Ramesh Visvanathan. Adversarially tuned scene generation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2

[47] Magnus Wrenninge and Jonas Unger. Synscapes: A photorealistic synthetic dataset for street scene parsing. In *arXiv:1810.08705*, 2018. 2, 7

[48] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tiani. Building generalizable agents with a realistic and rich 3d environment. In *arXiv:1801.02209*, 2018. 1, 2

[49] Ting Yao, Yingwei Pan, Yehao Li, and Tao Mei. Exploring visual relationship for image captioning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 684–699, 2018. 3

[50] Yinda Zhang, Shuran Song, Ersin Yumer, Manolis Savva, Joon-Young Lee, Hailin Jin, and Thomas Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2

[51] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networkss. In *ICCV*, 2017. 1, 2

[52] Song-Chun Zhu, David Mumford, et al. A stochastic grammar of images. *Foundations and Trends® in Computer Graphics and Vision*, 2(4):259–362, 2007. 3

[53] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 3

[54] Yang Zou, Zhiding Yu, B. V. K. Vijaya Kumar, and Jinsong Wang. Domain adaptation for semantic segmentation via class-balanced self-training. In *ECCV*, 2018. 1, 2