Appendix

A. Model Training Details

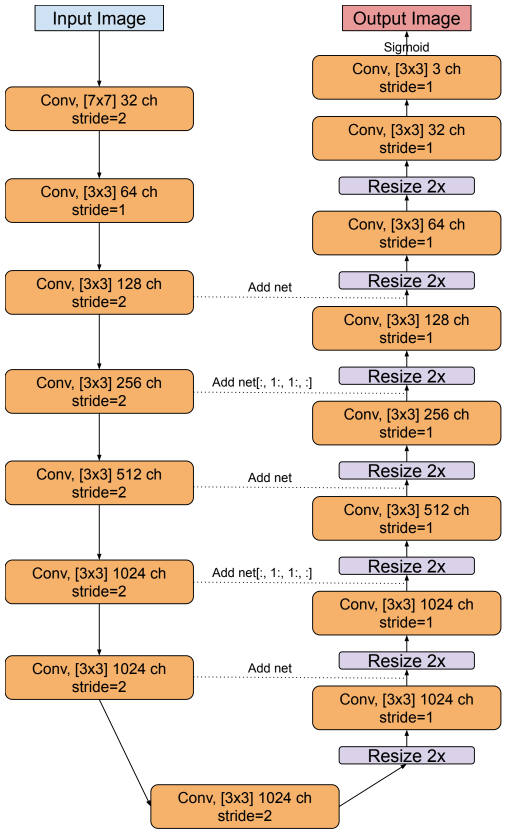

Figure 5. The U-net architecture used for G and F generator networks. Each convolutional block is shown with kernel size, number of filters and stride. Spectral normalization is applied to all convolutions. Images are resized to be twice as high and wide using nearest neighbor interpolation. Intermediate outputs from the down-convolutions (left) are added during the up-convolutions (right) as shown by the dotted lines, in two cases the first row and column are dropped during addition to match sizes. Instance normalization [32] is applied to all convolutions except the final output convolution.

We use data-sets consisting of grasping episodes from simulation and real robots. For Robot 1 and Robot 2 the data-sets are 580,000 and 80,000 real robot episodes respectively. Both data-sets are collected by starting with a human-designed scripted policy, which succeeds a small fraction of the time. Models are trained with this data, and periodically, those models are deployed to the robot to collect data from a better policy. When collecting data, random exploration noise is added to collect more diverse data. For this paper, we randomly subsample smaller datasets from these larger sets, to study the performance when using varying amounts of real episodes. For both setups several million simulated episodes are also generated during on-policy training.

Typical episodes contain 6-10 states represented by a 512 pixel high, 640 pixel wide RGB image. To increase data diversity images are randomly cropped to $472 \text{x} 472$ during training. At inference time, the center $472 \text{x} 472$ square from the image is used. The generator for the GAN is a convolutional neural network with a U-Net architecture [28] as shown in 5. The discriminator is smaller convolutional neural network that operates on three scales of the input image. Both networks are described in detail in [3]. The robotic grasping task is trained via QT-Opt with the Q-function represented as a convolutional neural network (see [20] for architecture). RL-CycleGAN jointly trains the Cycle-GAN along with the Q-function during QT-Opt. Models are trained on Google TPUv3 Pod as in [5] and required $bfloat16$ precision training to fit in memory. Each batch had 8 real images and 8 simulated images. We use Adam optimizer [21] with $\beta_1 = 0.1$ and $\beta_2 = 0.999$ and a constant learning rate of 0.0001. We employ Spectral Normalization [36] in the GAN generator networks and find that it improves stability.

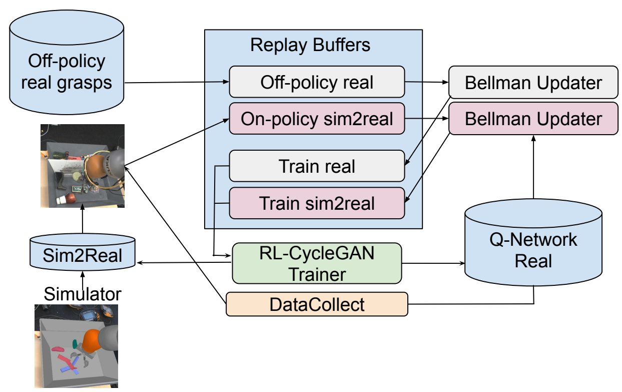

Figure 6. Training RL-CycleGAN via QT-Opt.

Figure 6 shows how we train RL-CycleGAN via QT-Opt. Images from a simulator are transformed by Sim2Real generator G and then passed to Q_{real} to generate an action. In this way, on-policy (w.r.t Q_{real}) episodes are generated in the simulation-to-real environment. Off-policy real grasps are read from disk. Separate replay buffers and bellman update instances are used for the off-policy real and off-policy simulation-to-real data. RL-CycleGAN is trained with batches with equal parts from real and simulation-to-real data. During training we evaluate the performance of both Q_{sim} and Q_{real} in the simulator, with simulation-to-real applied prior to evaluating Q_{real}. Training converges when certain conditions are met, the simulation-to-real images look realistic, the cycled images look reasonable with a reasonably low $\mathcal{L}_{cyc}$, both Q_{sim} and Q_{real} per-

Table 5. The various losses, their relative weights and the networks they affect for RL-CycleGAN .

Loss	Relative Weight (λ)	Networks Updated
$\mathcal{L}_{GAN}$	1	G, F, D_X, D_Y
$\mathcal{L}_{cycle}$	10	G, F
$\mathcal{L}_{RL}$	10	Q_{sim}, Q_{real}
$\mathcal{L}_{RL-scene}$	10	G, F

form well along with a reasonably low $\mathcal{L}_{RL-scene}$. A final Q_{real} is trained from scratch with the pre-trained and fixed Sim2Real generator G. This phase of training is as before, but with only the reinforcement learning loss.

Depending on the relative loss weights, λ, RL-CycleGAN might experience a particular mode of collapse, where Q outputs incorrect, uniform Q-values that give a spuriously low $\mathcal{L}_{RL-scene}$. This mode collapse can be caught by monitoring performance of Q during training and tuning λ appropriately. RL-CycleGAN involves multiple losses which are selectively applied to the various neural network components. The relative loss weights and the neural networks affected by the various losses is listed in Table 5.

We found that although adding a Q-network to CycleGAN improved performance, it was critical to maintain some separation between the two during optimization time. When $\mathcal{L}_{RL}$ and $\mathcal{L}_{RL-scene}$ were optimized entirely end-to-end, saliency analysis showed the Q-value for generated images was mostly dependent on generators G and F, rather than Q. We theorized the generators were computing the Q-value needed to minimize $\mathcal{L}_{RL-scene}$, embedding them within the generated image, and the Q-networks were simply decoding the embedded value. Such a Q-network generalizes poorly, does not understand the scene, and consequently does not provide a useful RL-scene consistency loss.

To fix this, the gradient for $\mathcal{L}_{RL}$ is only applied to Q, and the gradient for $\mathcal{L}_{RL-scene}$ is only applied to G and F. Note that in both cases, we still compute the full backward pass (there is no stop gradient), but we selectively choose which networks the gradient is applied to. Doing so makes it harder for the optimization to learn the poor encoding-decoding behavior mentioned above.

Since we train with batches of equal amounts of data from real data and from the simulator, we weight the loss from the real data depending on how many real episodes are available. While training the final Q_{real} a weighting term $\lambda_{RL-real}$ is applied,

$$\mathcal{L}_{RL} = \sum_{Sim2Real} \mathcal{L}_{RL} + \lambda_{RL-real} \sum_{Real} \mathcal{L}_{RL}$$

For all experiments we use $\lambda_{RL-real} = 0.1$ if using 10,000 real episodes or fewer, and $\lambda_{RL-real} = 2$ with 80,000 real episodes or more. Ablation results are shown in Table 6.

Table 6. The impact of using $\lambda_{RL-real}$. With only 3,000 real episodes a small real loss weight of 0.1 is optimal while with a large data-set of 80,000 real episodes a real loss weight of 2.0 was found to be best.

Off-policy episodes	$\lambda_{RL-real}$	Robot 2 Grasp Success
3,000	1.0	66%
3,000	0.1	72%
80,000	1.0	91%
80,000	2.0	95%

B. Robot Simulated Objects

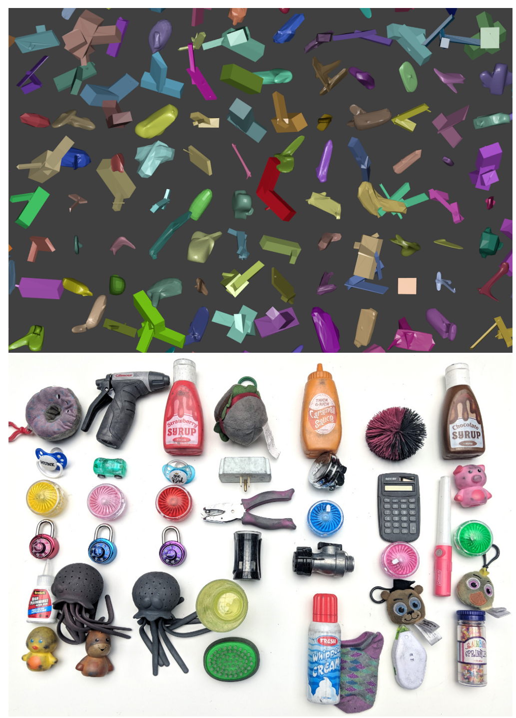

Figure 7. Robot 1: procedural objects generated in the simulator (top) and the unseen objects using during evaluation (bottom).

The goal with Robot 1 to be able to grasp unseen objects during evaluation. In simulation we procedurally generate objects with random shapes by attaching rectangular prisms at random locations and orientations. These procedural objects and the actual unseen objects used during evaluation are shown in Figure 7.

Since Robot 2 grasps trash-like objects we mimic the simulated objects more closely with the real world object. We create simulated versions of 51 common real world ob-

Figure 8. Robot 2: simulated versions of 51 common real world objects are used when training the RL-CycleGAN , including plastic bottles (8), coffee cups (18), plastic utensils (3), drink cans (6), mugs (15), and wine glass (1).

jects are created: plastic bottles (8), coffee cups (18), plastic utensils (3), drink cans (6), mugs (15), and wine glass (1), shown in Figure 8. These do not cover all the real world objects used evaluation.