Figure 5: Real-world grasping objects that range greatly in size and appearance. *Left*: about 1000 visually and physically diverse training objects used for joint finetuning. *Right*: the unseen test objects.

A. RCAN Architecture

The generator G is parameterized by weights of a convolutional neural network, summarized in Figure 7, and follows a U-Net style architecture [48] with downsampling performed via 3×3 convolutions with stride 2 for the first 2 layers, and average pooling with 3×3 convolution of stride 1 for the remaining layers. Upsampling was performed via bilinear upsampling, followed by a 3×3 convolutions of stride 1, and skip connections were fused back into the network via channel-wise concatenation, followed by a 1×1 convolution. All layers were followed by instance normalization [63] and ReLU non-linearities. The discriminator D is also parameterized by weights of a convolutional neural network with 2 layers of 32, 3×3 filters, followed by a layer of 64, 3×3 filters, and finally a layer of 128, 3×3 filters. The network follows a multi-scale patch-based design [3], where 3 scales of 472×472, 236×236, and 118×118, are used to produce domain estimates for all patches which are then combined to compute the joint discriminator loss.

B. QT-Opt Architecture

The action space of [27], which consists of gripper pose displacement and an open/close command, remains unchanged in our paper, and is defined as $\mathbf{a}_t = (\mathbf{t}_t, \mathbf{r}_t, g_{\text{close},t}, g_{\text{open},t}, e_t)$, containing Cartesian translation $\mathbf{t}_t \in \mathbb{R}^3$, sine-cosine rotation encoding $\mathbf{r}_t \in \mathbb{R}^2$, a one-hot vector gripper open/close command $[g_{\text{close},t}, g_{\text{open},t}] \in \{0, 1\}^2$, and a learned stopping criterion e_t. The reward function is sparse, consisting of a reward of 1 following a successful grasp, or 0 for an unsuccessful grasp, and -0.05

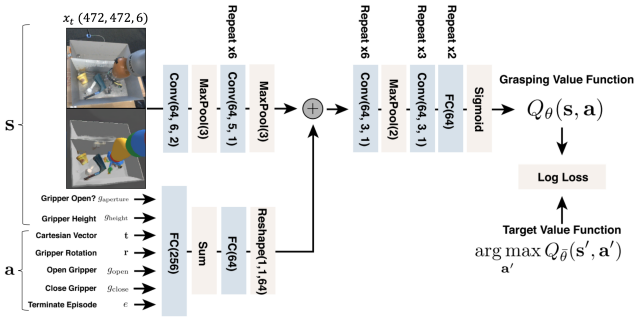


Figure 6: The Q-function of the grasping algorithm. The source image x (either from the randomized domain or real-world domain) and generated canonical image x_a are concatenated (channel-wise) and processed by a convolutional neural network (and fused with action and state variables) to produce a scalar representing the Q value $Q_\theta(s, a)$.

on all other transitions. Summarized in Figure 6, the Q-function follows the same architecture as [27] (originally inspired by [34]).

Rather than a single RGB image input, our network takes in a 6 channel image, consisting of channel-wise concatenation of the source image x (either from the randomized domain or real-world domain) and generated image x_a. Features are extracted from these images via 7 convolutional layers and then merged with a transformed action and state vector (which have passed through 2 fully-connected layers) via element-wise addition. The merged streams are then processed by a further 9 convolution layers and 2 fully-connected layers, resulting in a scalar output representing the Q value $Q_\theta(s, a)$. Each layer, excluding the final, uses batch normalization [22] and ReLU non-linearities. A summary of the architecture can be seen in Figure 6.

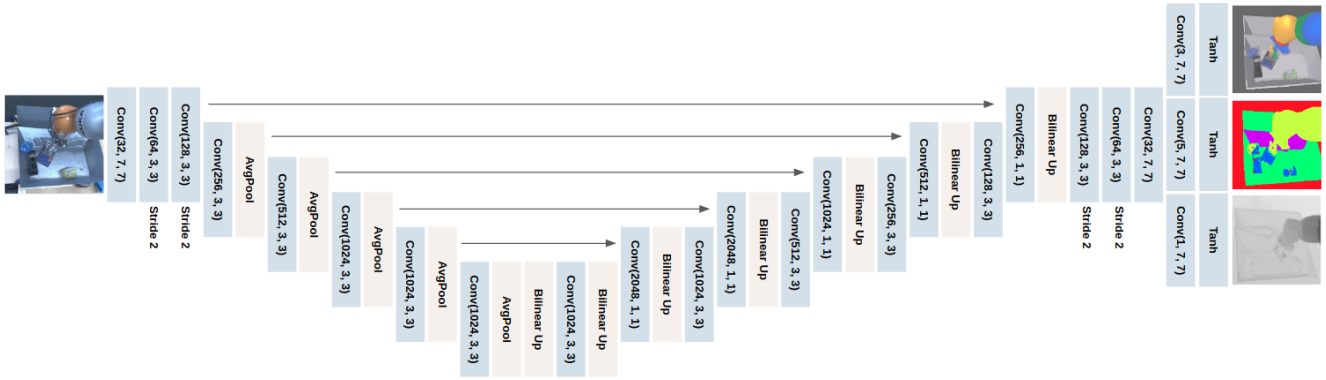
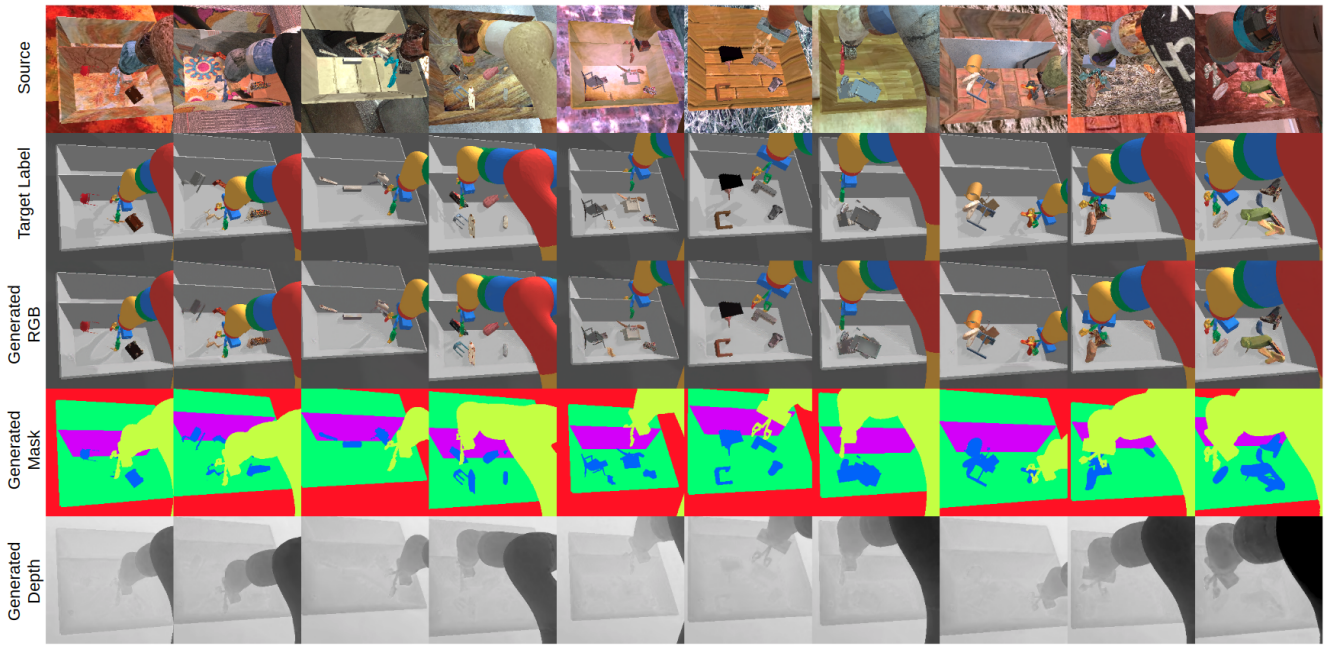
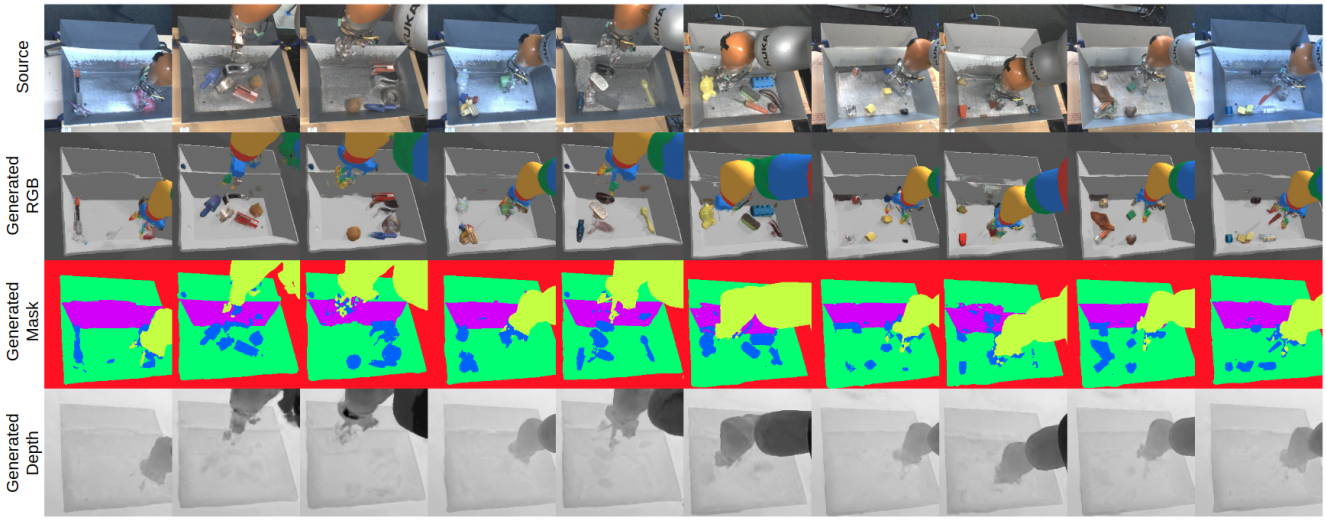


Figure 7: Network architecture of the generator function G. An RGB image from the source domain (either from the randomized domain or real-world domain) is processed via a U-Net style architecture [48] to produce a generated RGB image x_a, and auxiliaries that includes a segmentation mask m_a and depth image d_a. These auxiliaries forces the generator to extract semantic and depth information about the scene and encode them in the intermediate latent representation, which is then available during the generation of the output image.



(a) Randomized-to-canonical samples.



(b) Real-to-canonical samples.

Figure 8: Additional sample outputs of our trained generator G when given randomized sim images (8a) and real images (8b).