1 Sampler GAN

1.1 Overview

- This GAN is a conditional, noisy generator coupled with a traditional discriminator. The generator learns to predict the depth map of single RGB patches while preserving stochasticity. The hope is that the generator learns to predict multiple samples for a single RGB patch, and that by taking the best sample for each patch, and multiple patches per image, we can then reconstruct a global depth map.

1.2 Dataset

- We use the NYUv2 dataset. We use the official train/test scene splits, then use the toolbox alignment tools to generate correlated RGB-Depth pairs. We select every 10th image from this set (under the idea that nearby frames likely contribute little new information) and use one pass of the toolbox fill code to fill in some of the missing depth values. We then center-crop each RGB-D pair to 561x427[1] and set aside 10% of the training data for validation purposes. This gives us a final set of 27,858 images for training, 3,095 for validation, and 20,343 for testing.

- From each image we then select a random 65x65 crop and throw away any that have invalid depth values (0 or uint16.max, representing pixels for which the Kinect camera did not receive back any reflected depth measurements). Currently this is not deterministic, but the resulting datasets tend to fluctuate around 95% of the original size. We then further center-crop the depth map to size 31x31.

- The final RGB-D pairs are then remapped to the range [-1, 1].

1.3 Architecture

1.3.1 Generator

- The input to the generator is a 65x65x3 RGB patch. To this we concatenate a 65x65x1 noisy uniform vector (over the same range as the input, or [-1, 1]).

- The output of the generator is a 31x31x1 estimated depth map.

- The generator is an autoencoder with skip layers. It takes the input, and, over 4 layers, reduces the dimensions to a latent vector of 1x1x256, then expands this back out to a final output of 31x31x1.

[1] This is a byproduct of using the dataset previously with mdepth/VGG, which uses this size as the input.

- Let `Ck.2` represent a 2D convolution of stride 2, VALID padding, filter size 5, and `k` output filters, followed by batch normalization and an activation LeakyReLU layer (leak=0.2).
- Let `Ck.1` represent a 2D convolution of stride 1, SAME padding, filter size 5, and `k` output filters, followed by batch normalization and an activation LeakyReLU layer (leak=0.2).
- Let `Dk.2` represent a 2D transpose convolution of stride 2, VALID padding, filter size 5, and k output filters, followed by batch normalization and an activation LeakyReLU (leak=0) layer. We then concatenate the output of the appropriate encoder layer (doubling the number of filters).
- Let `Dk.1` represent a 2D transpose convolution of stride 1, SAME padding, filter size 5, and k output filters, followed by batch normalization and an activation LeakyReLU (leak=0) layer.
- Then the generator's architecture can be described as

Unit	Layer	Input Size	Output Size
Encoder	`C64.2`	65x65x4	31x31x64
	`C64.1`	31x31x64	31x31x64
	`C64.1`	31x31x64	31x31x64
	`C128.2`	31x31x64	14x14x128
	`C128.1`	14x14x128	14x14x128
	`C128.1`	14x14x128	14x14x128
	`C256.2`	14x14x128	5x5x256
	`C256.1`	5x5x256	5x5x256
	`C256.1`	5x5x256	5x5x256
	`C256.2`	5x5x256	1x1x256
Decoder	`D256.2`	1x1x256	5x5x512
	`D512.1`	5x5x512	5x5x512
	`D128.2`	5x5x512	14x14x256
	`D128.1`	14x14x256	14x14x256
	`D64.2`	14x14x256	31x31x128
	`D64.1`	31x31x128	31x31x128
	`D1.2`	31x31x128	31x31x1

- All weights are initialized to a Gaussian with mean 0 and stddev 0.02.
- There is no batch norm on the first layer of the encoder.

1.3.2 Discriminator

- The discriminator has two inputs, the 65x65x3 RGB patch and the matching 33x33x1 center-cropped depth:
  - The RGB input is run through a 2D convolution (VALID padding, filter size 5, stride 2), followed by LeakyReLU (leak=0.2).

  - The Depth input is expanded through a 2D convolution (SAME padding, filter size 1, stride 1), followed by LeakyReLU (leak=0.2).

- These two inputs are then concatenated together and run through 3 more convolutions of increasing depth (128, 256, and 512 filters, respectively), all with stride 2, filter size 5, and LeakyReLU activation (leak=0.2).

- The final output is then run through sigmoid activation to generate a single output in range [0, 1].

1.4 Training

- The generator and discriminator are trained equally (one generator iteration followed by one discriminator) using SGD with the ADAM solver, learning rate 1e-5, and momentum 0.5.

- Training was conducted with a batch size of 64 over 200 epochs, with 217 batches per GPU per epoch, for 27,776 samples per epoch or about 5.5 million training images. Batches are selected from the dataset via fair random draw. The original random crops (one 65x65 RGB drop and one 33x33 depth crop) are generated only once, at the beginning of training.

- We use the standard GAN cross-entropy loss functions.

1.5 Results

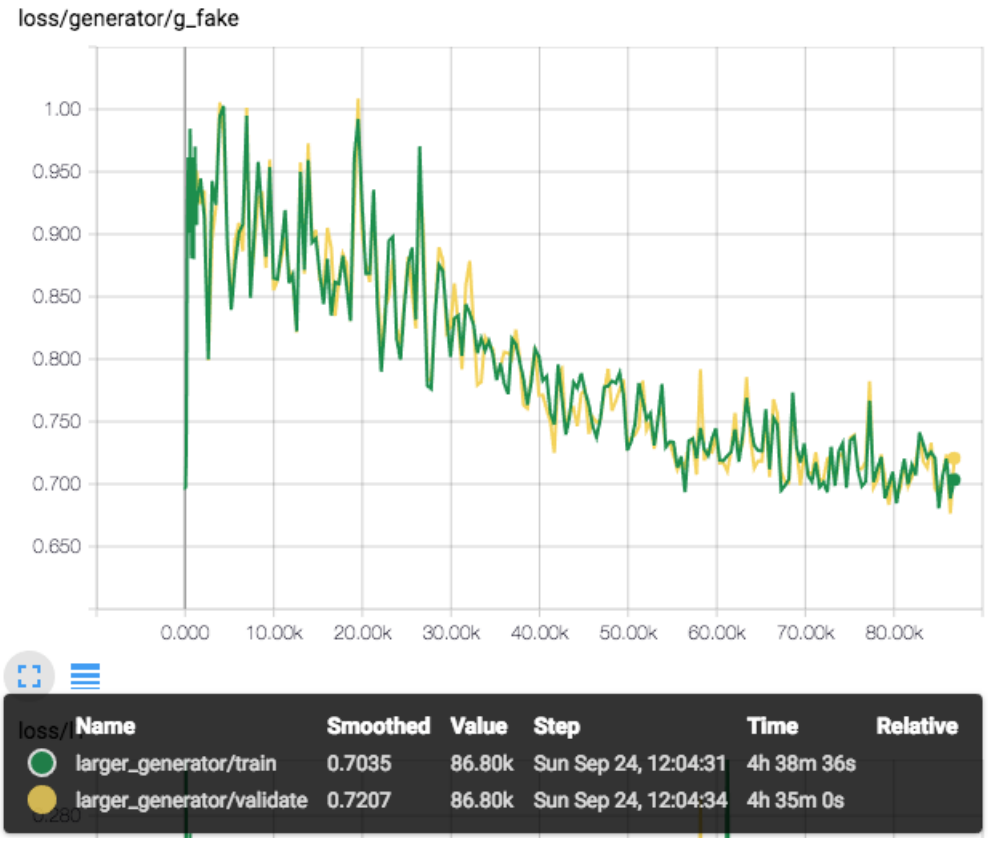

Figure 1: Graph of the generator's loss.

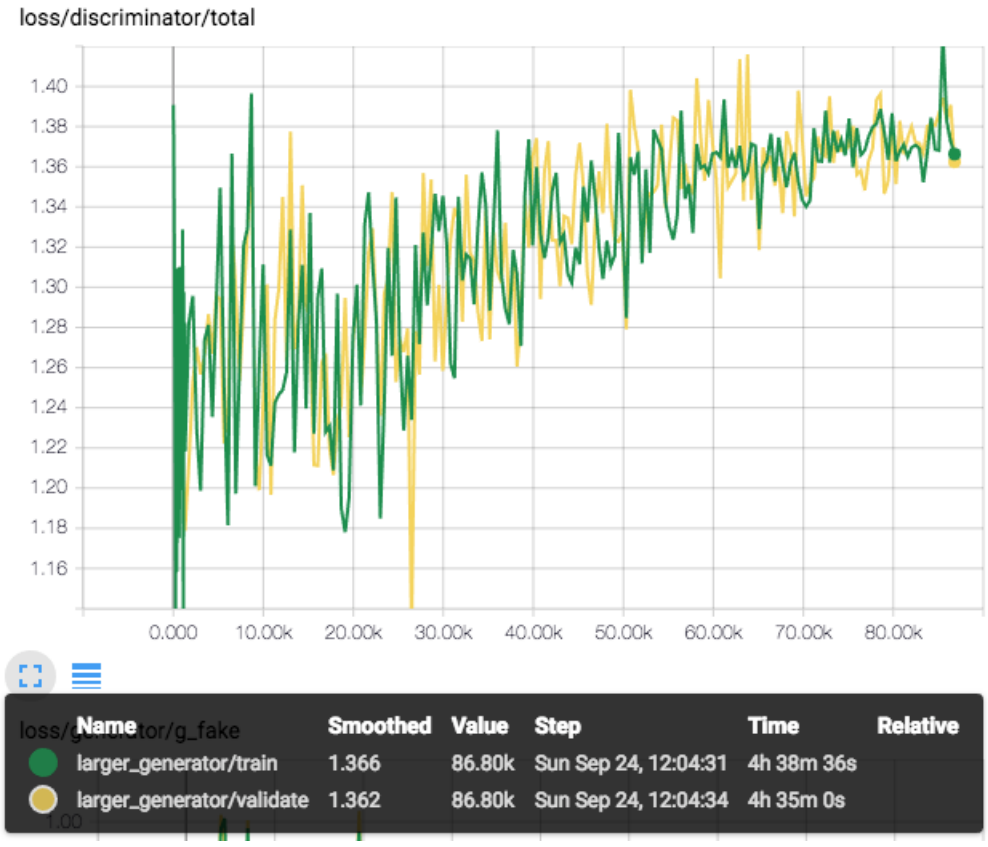

Figure 2: Graph of the discriminator's loss.

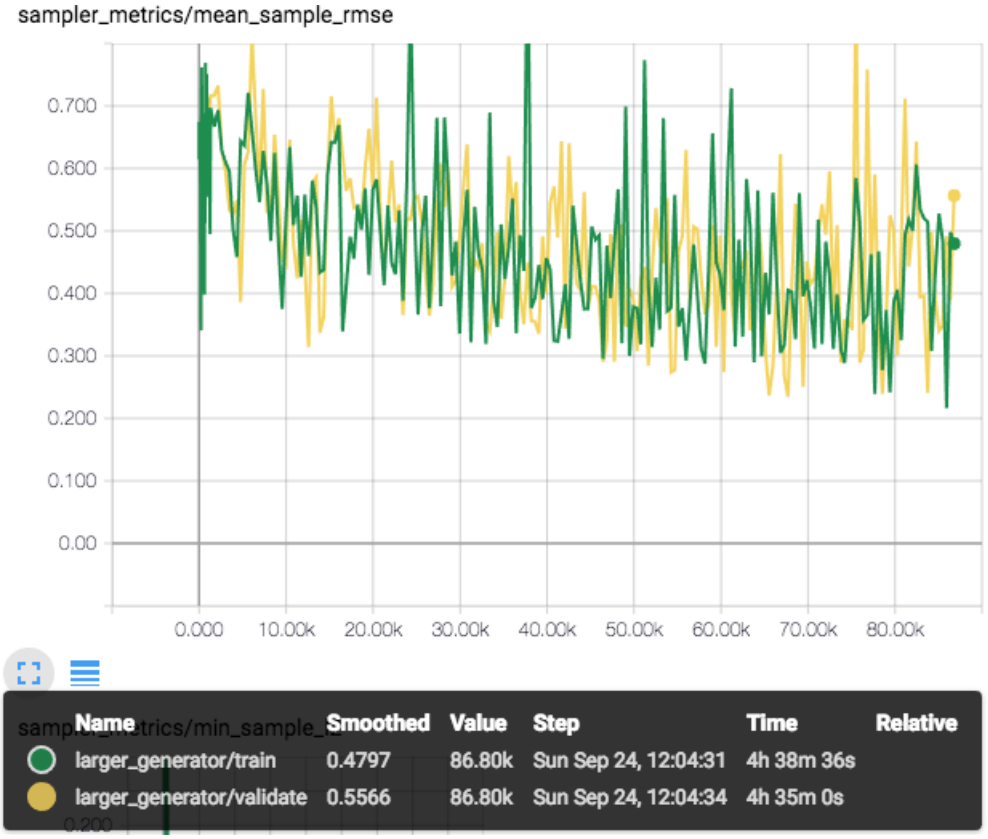

Figure 3: The mean RMSE of the sampler's output. The sampler is given 64 copies of the same RGB image and asked to generate predictions for each one. We calculate the RMSE of each one from ground-truth and display the mean of these here.

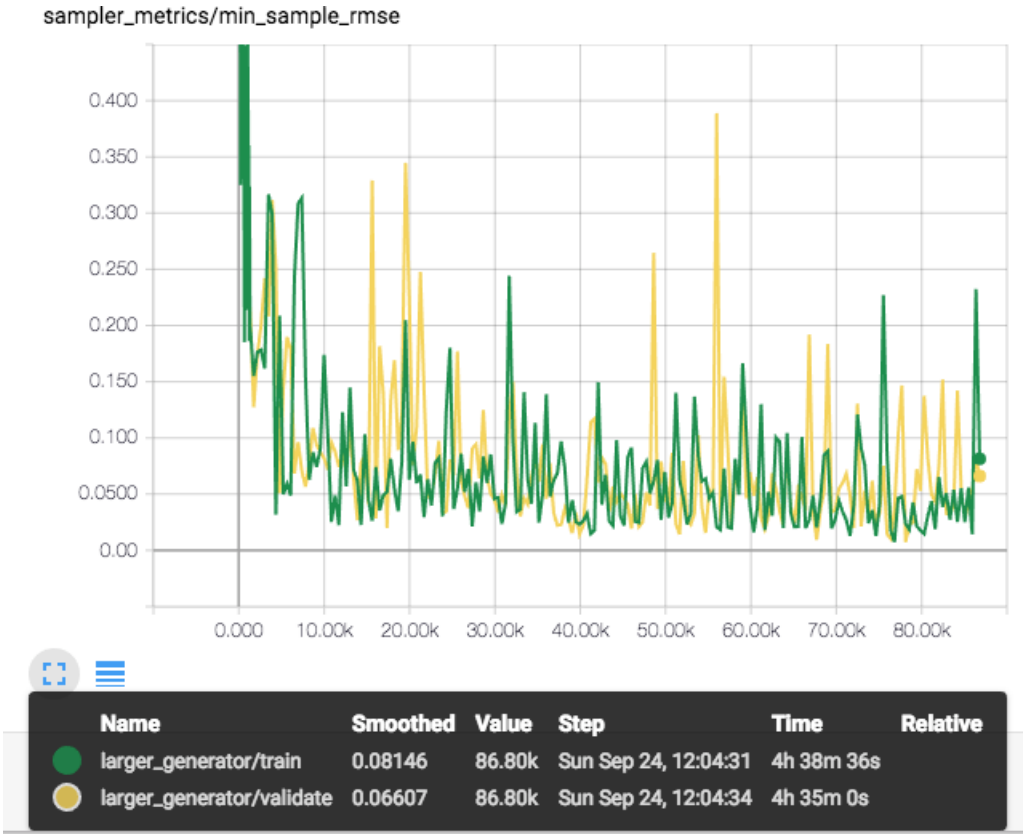

Figure 4: The min RMSE of the sampler's output. Likewise, we track the best of the sampler's outputs each time. Note that while the sampler's outputs are overall poor, there is a best one in each batch that is significantly better.

Figure 5: Example images.

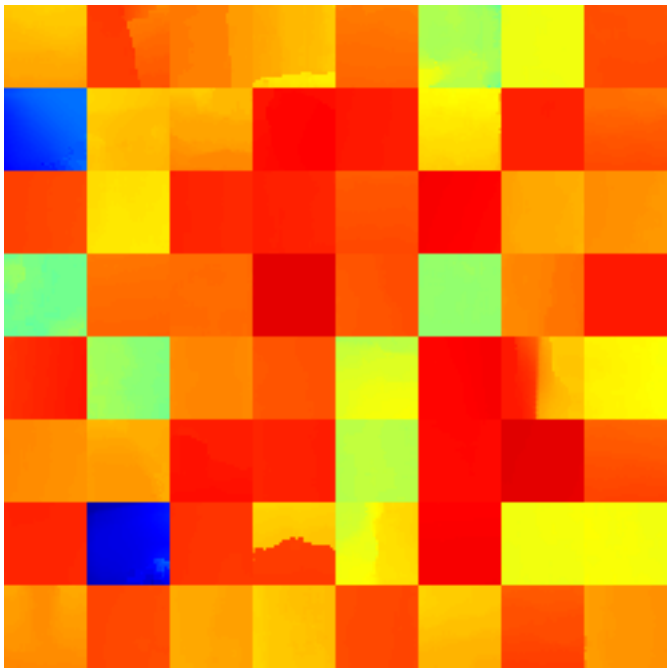

Figure 6: Example, ground-truth depths.

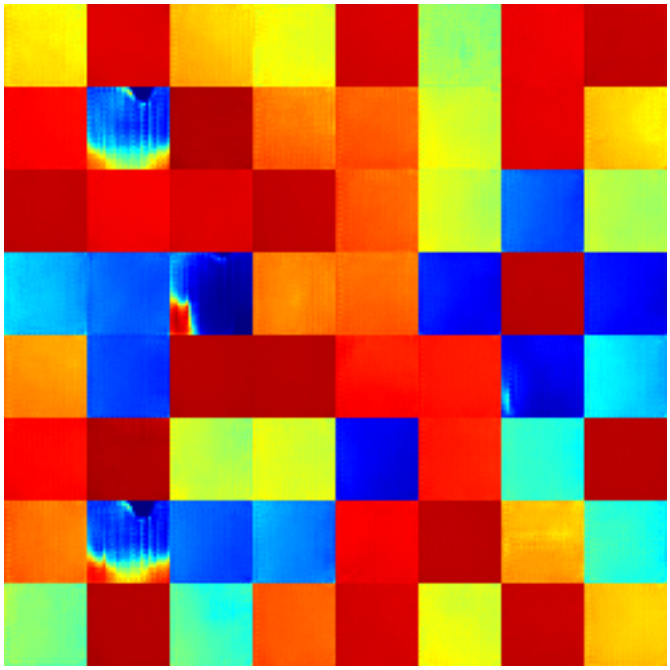

Figure 7: The corresponding generator output for the same images.

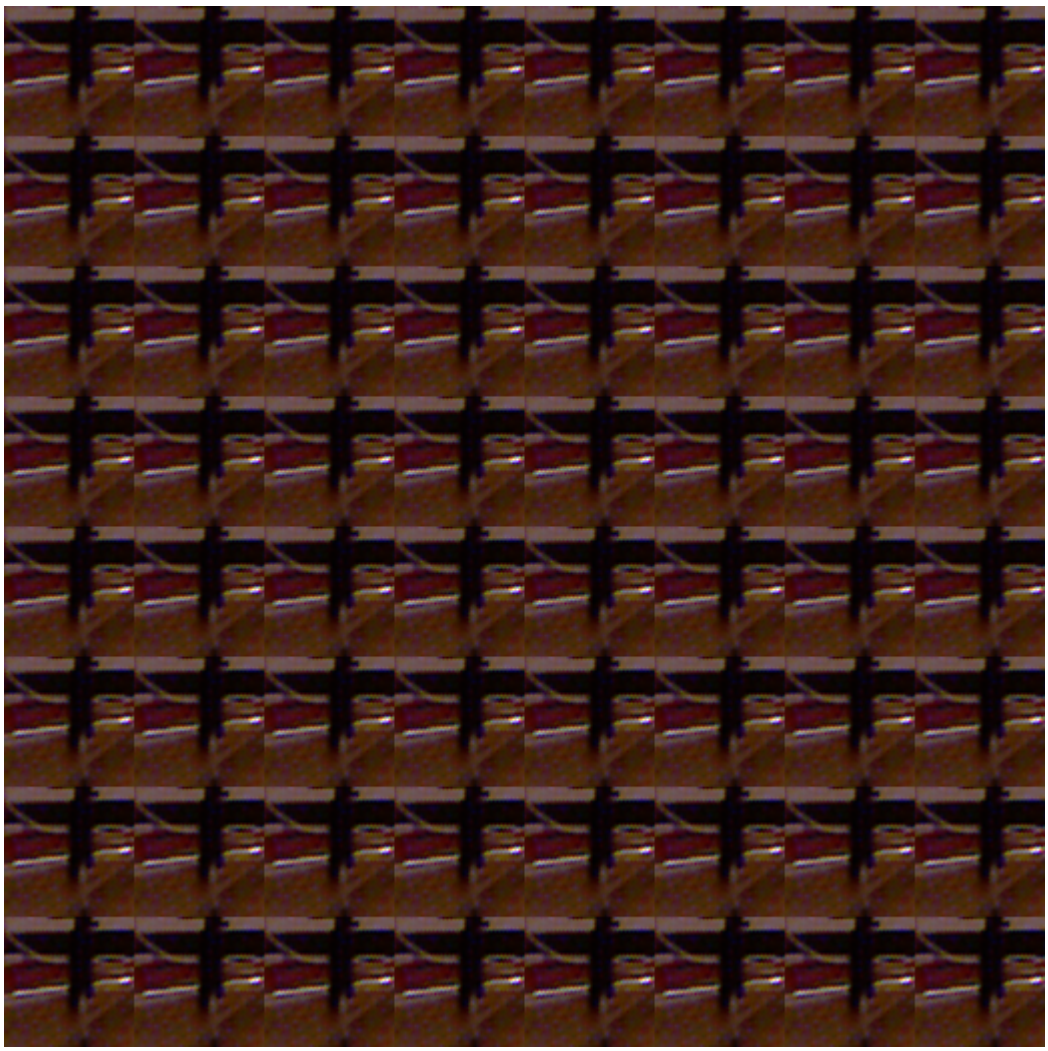

Figure 8: The sampler's real input images

Figure 9: The sampler's ground-truth depth.

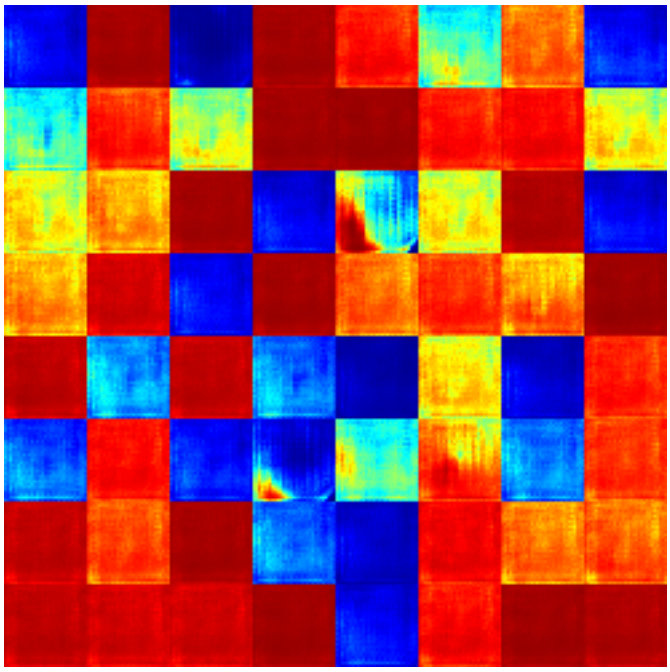

Figure 10: 64 depth predictions from the generator for the single sampler input image.