CS390-NIP - Fall 2020
Name: Michael Lee
Email: lee2965@purdue.edu
Date: November 3, 2020

Lab 4 Report

Basic Image Classification

Due Date: November 2, 2020

https://github.com/michaeldlee23/cs390-lab4

Resources Used

- Conv2DTranspose and UpSampling2D in Keras
  https://machinelearningmastery.com/upsampling-and-transpose-convolution-layers-for-generative-adversarial-networks/
- Calculating output size after Conv2DTranspose
  https://datascience.stackexchange.com/questions/26451/how-to-calculate-the-output-shape-of-conv2d-transpose
- Tensorboard with train_on_batch
  https://stackoverflow.com/questions/44861149/keras-use-tensorboard-with-train-on-batch

Completed Portions of Lab

1. [80] GANs
   - ✓[50] Complete the GAN to generate legible F-MNIST records. Generate from 3 classes.
   - ✓[10] Use a convnet for the GAN networks.
   - ✓[10] Implement the option to select the ratio of training between discriminator and generator.
   - ✓[10] Save a plot of loss over training steps for each network.
   - ✓? [+5] Generate very detailed F-MNIST records (there is no metric for this)
   - ✓? [+20] Generate legible CIFAR records from 3 classes
   - [+10] Create your own dataset of images and generate records of it. Images shoud be 32x32x1 or 32x32x3 and must be reasonably complex. Include the dataset in your submission with examples in your report.
2. ✓[20] Report

Questions

1. Describe the discriminator and generator.

   The discriminator and generator are two parts of the GAN that basically train and improve each other — the generator produces images, and the discriminator takes those generated images, along with real images, and tries to classify a given image as real or generated. The generator tries to fool the discriminator, and the discriminator tries to learn how to not be tricked. Thus, we create an adversarial relationship between the two networks which results in an effective image generator.

2. Why do we sometimes need to train the discriminator and generator different amounts?

   Due to the adversarial nature of the networks, we need to find the proper balance between the discriminator and generator in terms of their relative strength to each other. That is, if the discriminator is too good at its job, the generator won't gain enough information to improve. Likewise, if the generator is too good at its job, the discriminator will struggle to improve. One remedy to this is to adjust the training ratio between the two networks, allowing, for example, the generator to continue training while the discriminator stays constant until it can "catch up." We could achieve this by using a static constant that lets the generator get updates 10 times more frequently than the discriminator, for example, or we could set a threshold that only trains the discriminator when its loss reaches a certain level. Likewise, the same procedure could be applied to the generator.

Hyperparameters

Discriminator Model for MNIST data

```
Model: "MNIST_DISCRIMINATOR"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 14, 14, 64)        1664
_________________________________________________________________
leaky_re_lu_19 (LeakyReLU)   (None, 14, 14, 64)        0
_________________________________________________________________
dropout_15 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 7, 7, 128)         73856
_________________________________________________________________
leaky_re_lu_20 (LeakyReLU)   (None, 7, 7, 128)         0
_________________________________________________________________
dropout_16 (Dropout)         (None, 7, 7, 128)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_12 (Dense)             (None, 1024)              6423552
_________________________________________________________________
leaky_re_lu_21 (LeakyReLU)   (None, 1024)              0
_________________________________________________________________
dropout_17 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 1025
=================================================================
Total params: 6,500,097
Trainable params: 6,500,097
Non-trainable params: 0
_________________________________________________________________
```

Generator Model for MNIST data

```
Model: "MNIST_GENERATOR"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 6272)              633472
_________________________________________________________________
leaky_re_lu_22 (LeakyReLU)   (None, 6272)              0
_________________________________________________________________
batch_normalization_4 (Batch (None, 6272)              25088
_________________________________________________________________
reshape_2 (Reshape)          (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 14, 14, 64)        204864
_________________________________________________________________
leaky_re_lu_23 (LeakyReLU)   (None, 14, 14, 64)        0
_________________________________________________________________
batch_normalization_5 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 28, 28, 1)         1601
=================================================================
Total params: 865,281
Trainable params: 852,609
Non-trainable params: 12,672
_________________________________________________________________
```

Discriminator Model for CIFAR data

```
Model: "CIFAR_DISCRIMINATOR"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 16, 16, 64)        1792
_________________________________________________________________
leaky_re_lu_16 (LeakyReLU)   (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
leaky_re_lu_17 (LeakyReLU)   (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 4, 4, 128)         147584
_________________________________________________________________
leaky_re_lu_18 (LeakyReLU)   (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 2, 2, 256)         295168
_________________________________________________________________
leaky_re_lu_19 (LeakyReLU)   (None, 2, 2, 256)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 1025
=================================================================
Total params: 519,425
Trainable params: 519,425
Non-trainable params: 0
_________________________________________________________________
```

Generator Model for CIFAR data

```
Model: "CIFAR_GENERATOR"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 4096)              413696
_________________________________________________________________
leaky_re_lu_20 (LeakyReLU)   (None, 4096)              0
_________________________________________________________________
reshape_2 (Reshape)          (None, 4, 4, 256)         0
_________________________________________________________________
conv2d_transpose_6 (Conv2DTr (None, 8, 8, 128)         524416
_________________________________________________________________
leaky_re_lu_21 (LeakyReLU)   (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_transpose_7 (Conv2DTr (None, 16, 16, 128)       262272
_________________________________________________________________
leaky_re_lu_22 (LeakyReLU)   (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_transpose_8 (Conv2DTr (None, 32, 32, 128)       262272
_________________________________________________________________
leaky_re_lu_23 (LeakyReLU)   (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 32, 32, 3)         6147
=================================================================
Total params: 1,468,803
Trainable params: 1,468,803
Non-trainable params: 0
_________________________________________________________________
```

Relevant Plots - MNIST_F

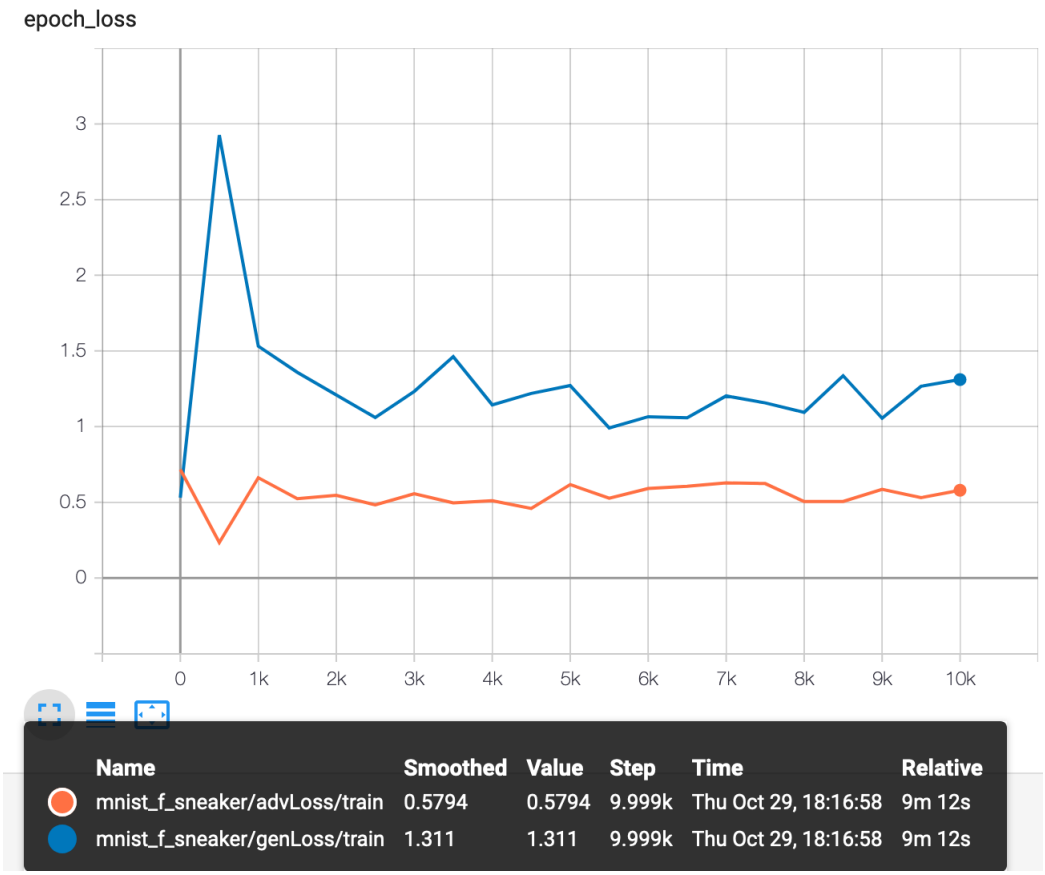

Figure 1: Loss over time for MNIST_F Sneaker GAN

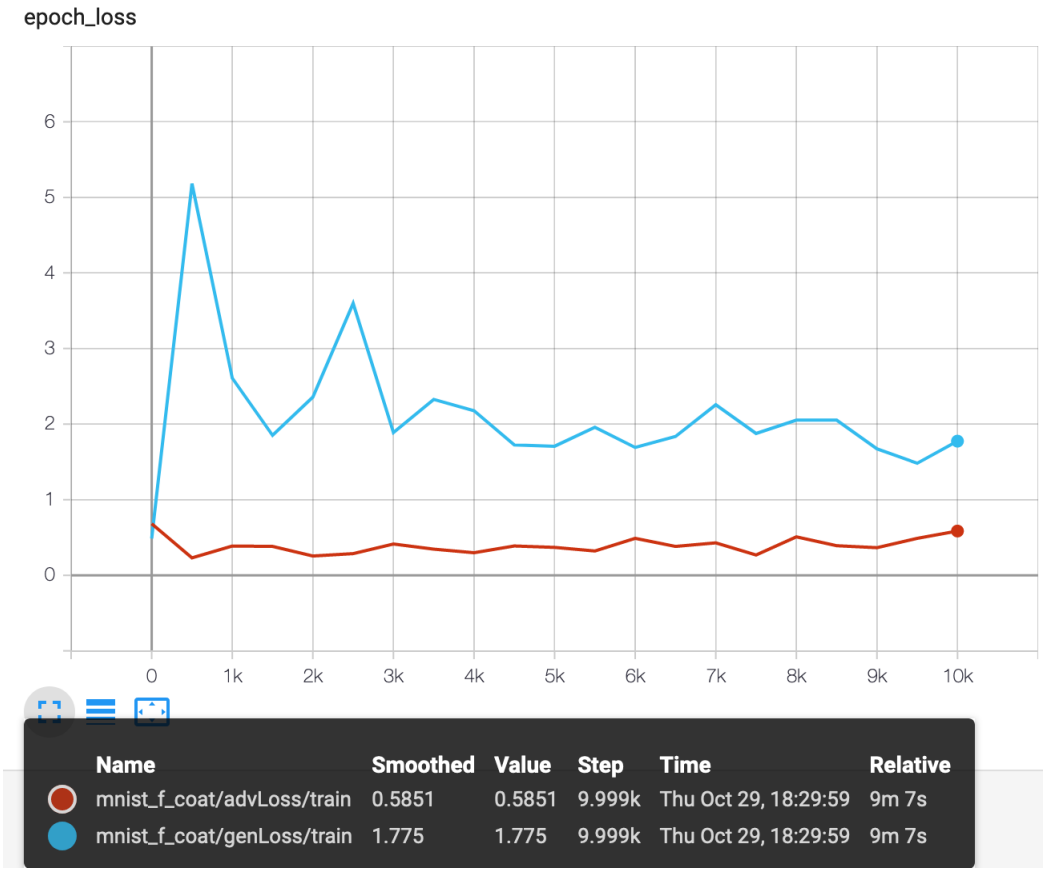

Figure 2: Loss over time for MNIST_F Coat GAN

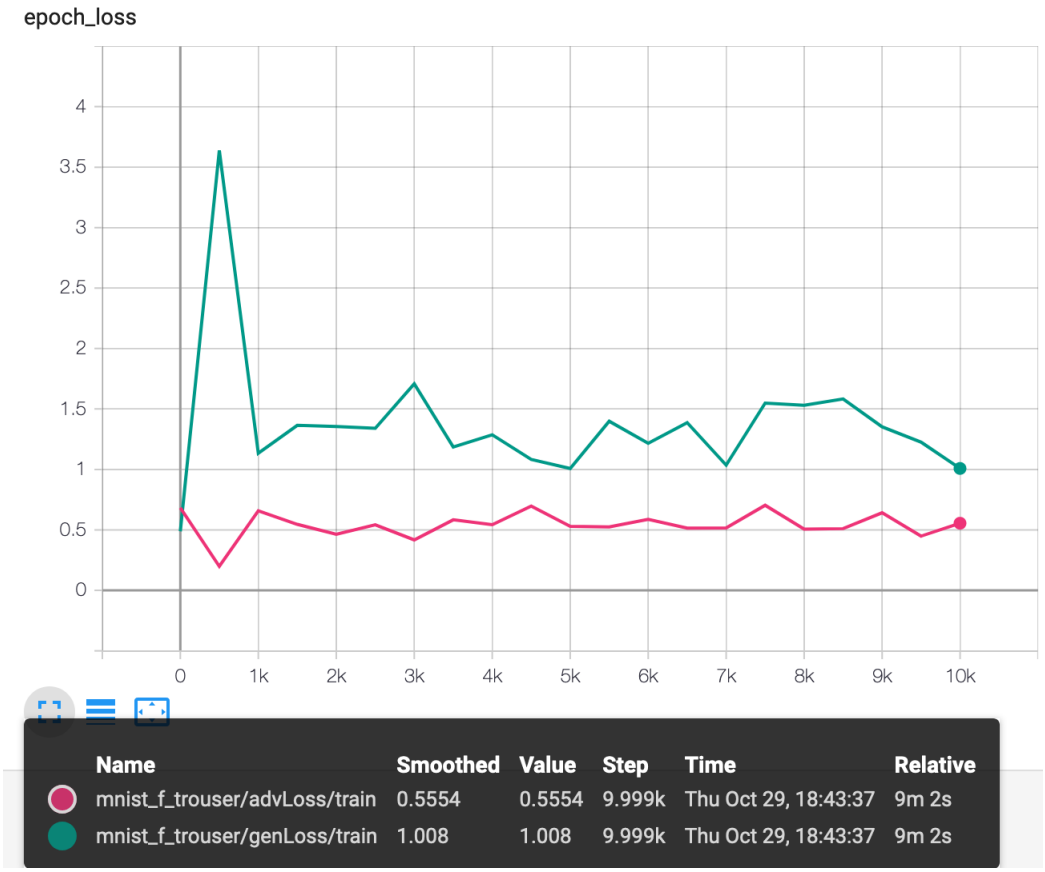

Figure 3: Loss over time for MNIST_F Trouser GAN

Generated Images - MNIST_F

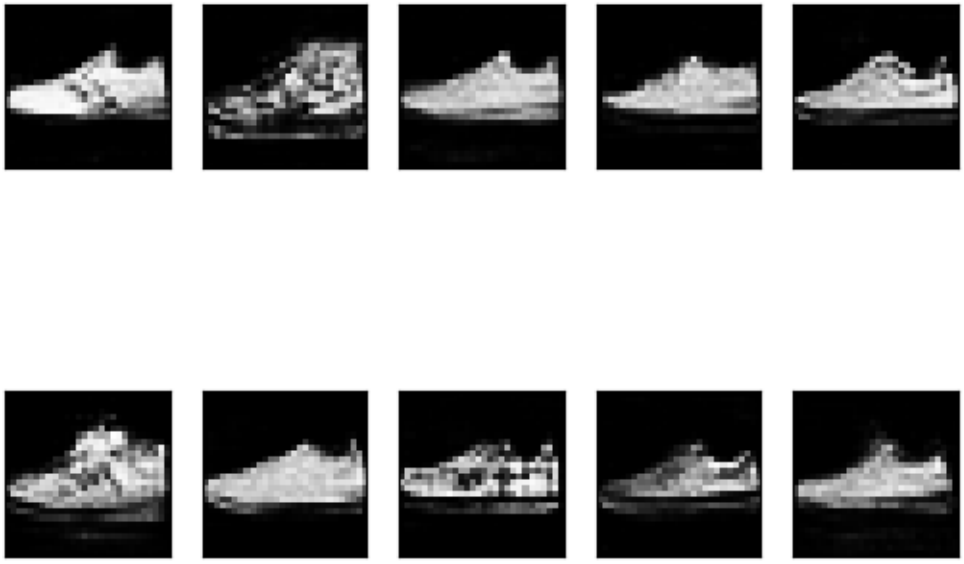

Figure 4: Generated Sneaker Images

Figure 5: Generated Coat Images

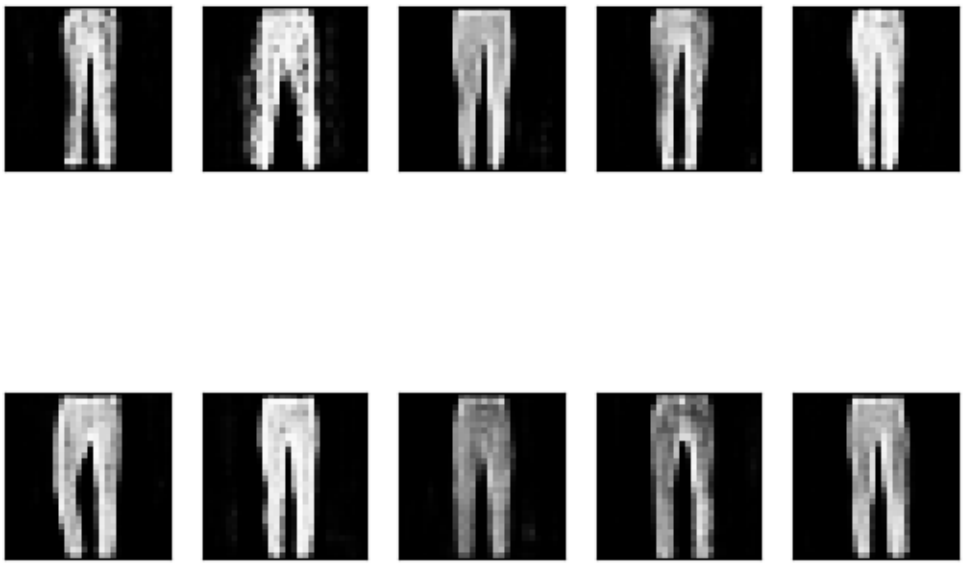

Figure 6: Generated Trouser Images

Relevant Plots - CIFAR_10

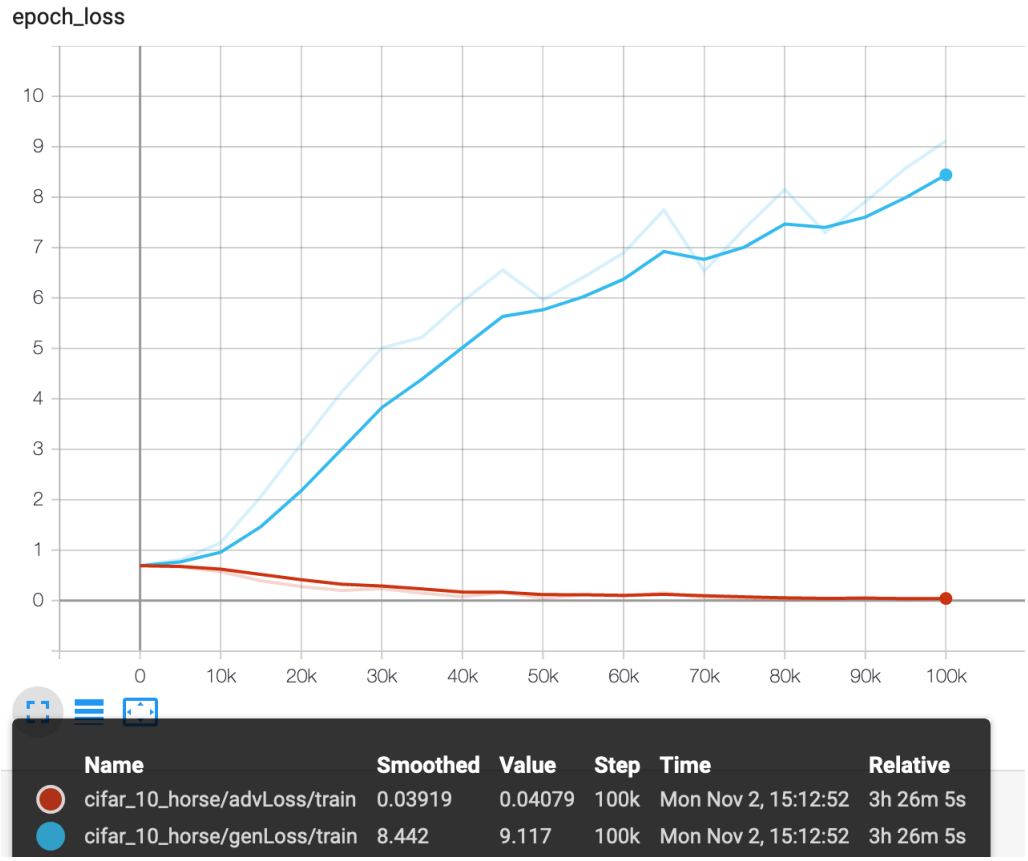

Figure 7: Loss over time for CIFAR_10 Horse GAN

Interestingly, while there is a clear divergence in loss which would suggest that the GAN is probably not performing well, we do nevertheless get a couple horse-like images. We have similar results for Bird and Dog.

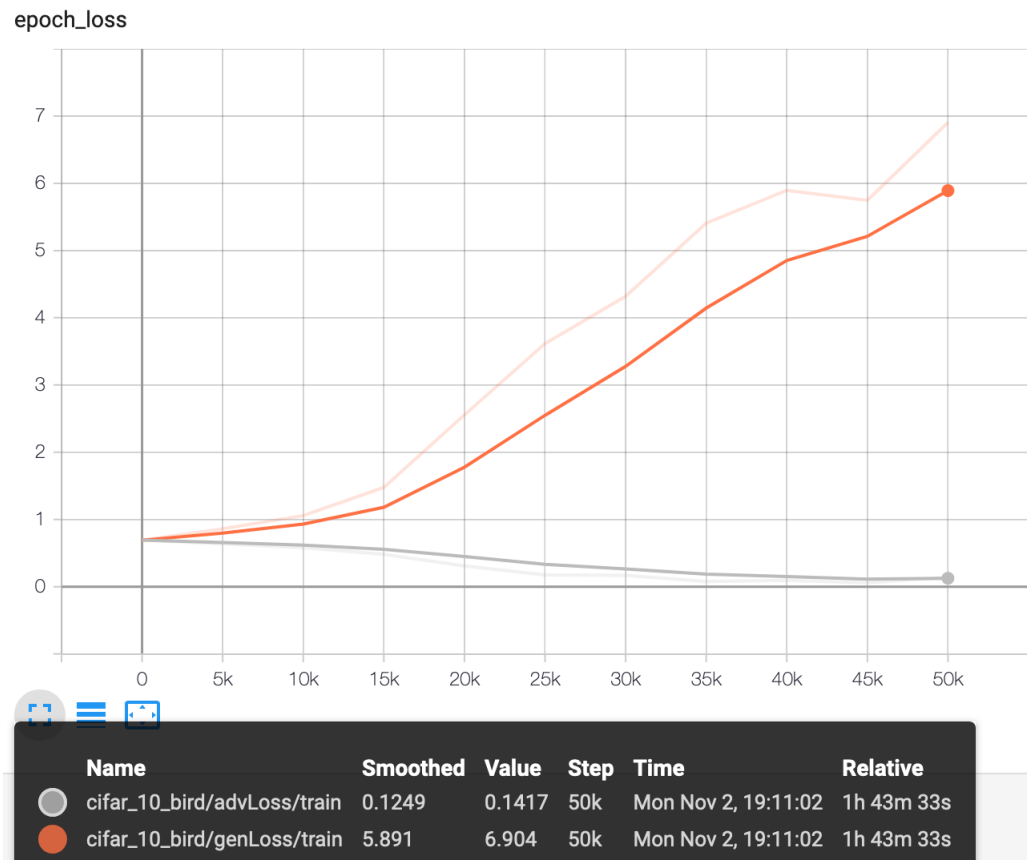

Figure 8: Loss over time for CIFAR_10 Bird GAN

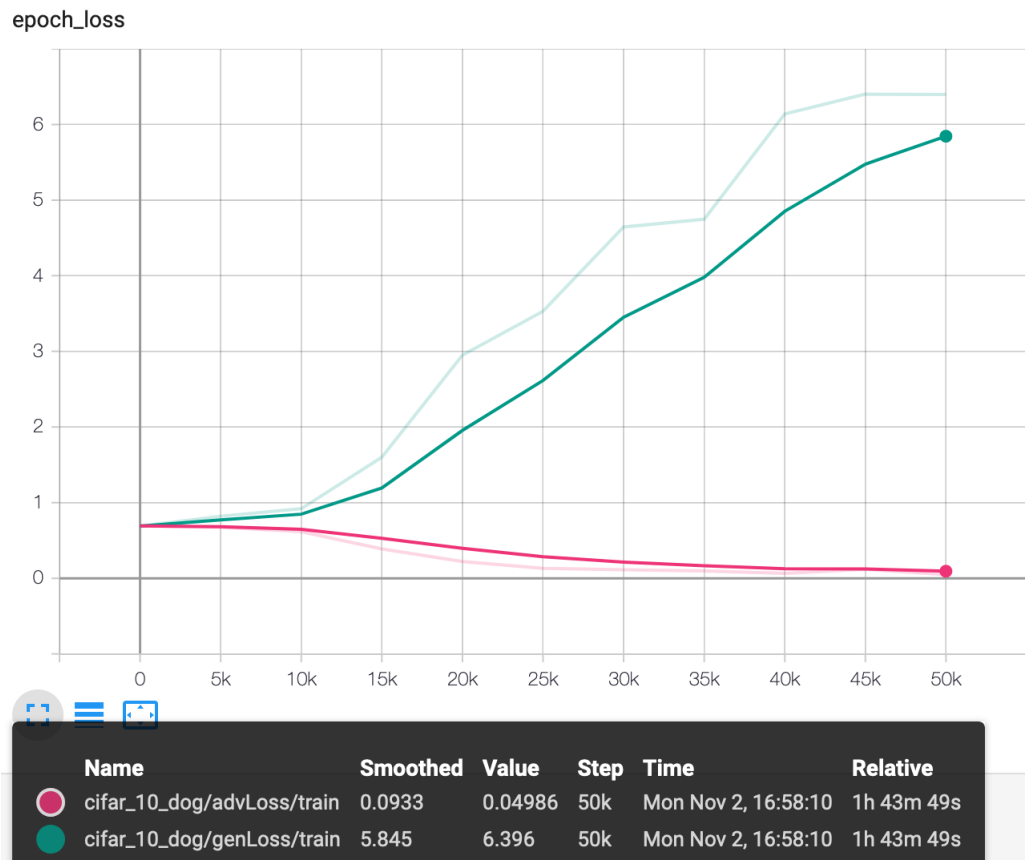

Figure 9: Loss over time for CIFAT_10 Dog GAN

Generated Images - CIFAR_10

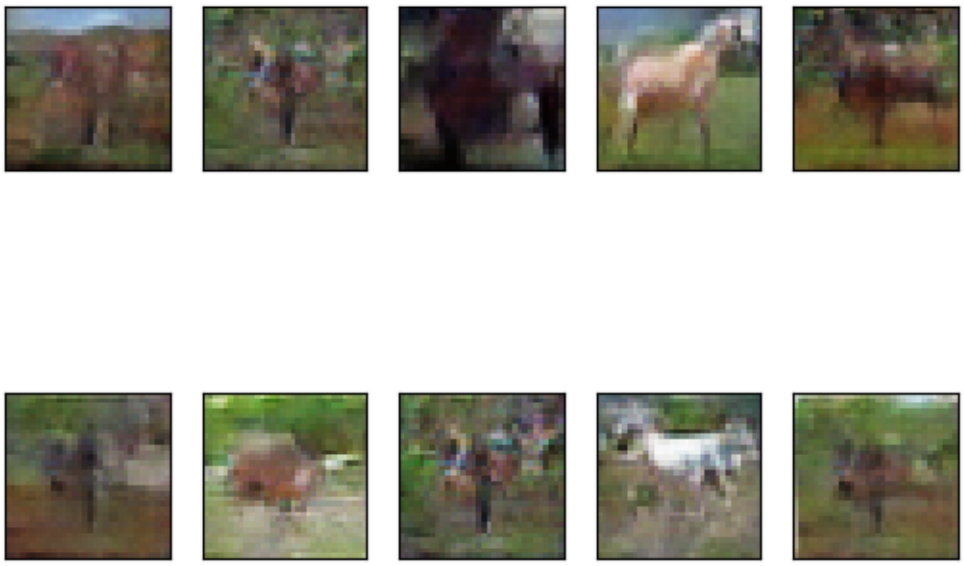

Figure 10: Generated Horse Images

While some of these pictures are pretty illegible, we still get a couple that definitely have a horse-like quality to them. Namely, the last two images in the first row.

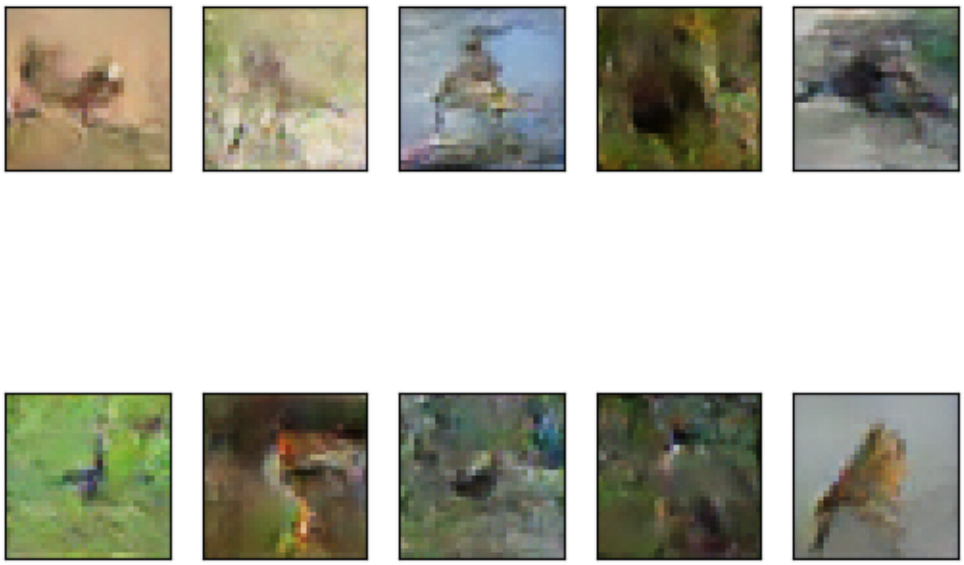

Figure 11: Generated Bird Images

The bird generator did not do that well, although the very last image does sort of look like a bird. Note that the bird generator may have produced better pictures, similar to the horse generator, if I trained it longer. The horse generator was trained twice as long as the bird and dog generator.

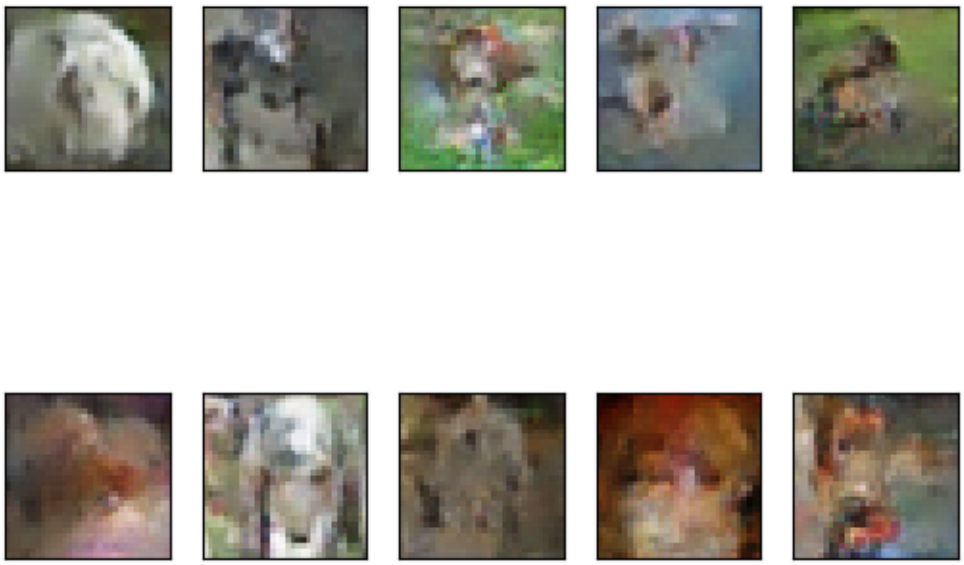

Figure 12: Generated Dog Images

The dog generator also did not perform very well, although we can kind of see the likeness of a dog in the very last picture. The right side of the image gets a little disfigured and what looks to be a tongue seems to be unnatural, but we we can definitely see an attempt at a dog.

Generated Images - CIFAR_10, Cherrypicked Horses

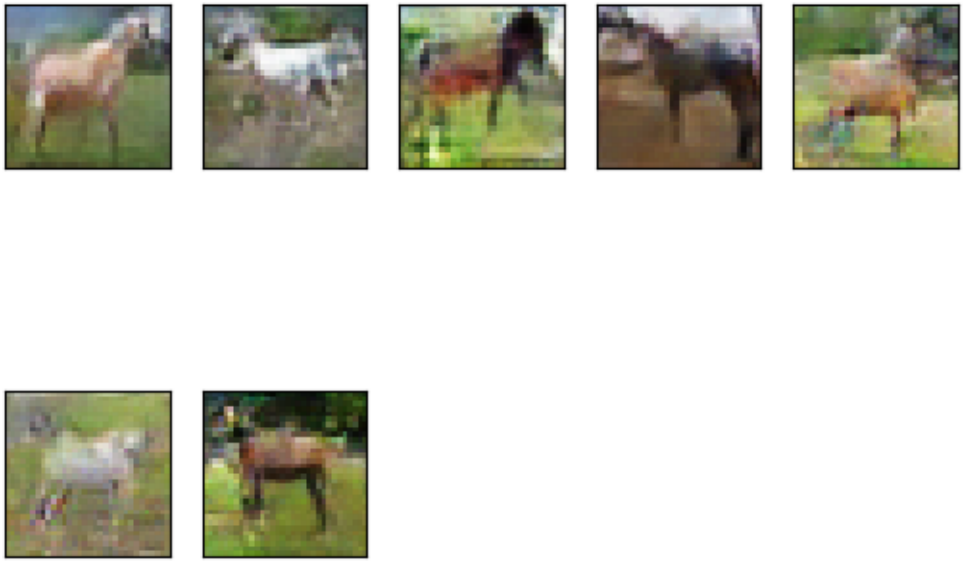

Figure 13: Cherrypicked Generated Horse Images

These were the best horse images after visually inspecting both the final and test outputs.