Anderson, Christopher
Bronco ID: 014105585

Lab 4 Report

Part 1: Training the Discrimator

Chosen digit: 7

Architecture:

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator,self).__init__()
        #Define inputs, outputs
        self.linear1 = nn.Linear(784,512)
        self.linear2 = nn.Linear(512,256)
        self.linear3 = nn.Linear(256,128)
        self.linear4 = nn.Linear(128,1)
        self.relu1 = nn.LeakyReLU()
        self.relu2 = nn.LeakyReLU()
        self.relu3 = nn.LeakyReLU()
        self.sig1 = nn.Sigmoid()
        self.dropout = nn.Dropout(0.6)

    def forward(self, x):
        h = self.linear1(x)
        h = self.relu1(h)
        h = self.dropout(h)
        h = self.linear2(h)
        h = self.relu2(h)
        h = self.dropout(h)
        h = self.linear3(h)
        h = self.relu3(h)
        h = self.dropout(h)
        h = self.linear4(h)
        return self.sig1(h)
```

The network consisted of 4 linear layers that went from 784 -> 512 -> 256 -> 128 -> 1.

The activation function LeakyReLu was used after every linear layer followed by a dropout of .6 to prevent overfitting of the network- this was also implemented mainly for part 2 as part one wasn't having any problems with learning rates or mode collapsing. The final output was a sigmoid activation to produce values 0-1.

Anderson, Christopher
Bronco ID: 014105585

Results:

```
 0   1   2   3   4   5   6   7   8   9
[3, 4, 15, 27, 28, 32, 32, 94, 35, 44]
TP  934 FN  94 FP  44 TN  8928
934 94 44 8928
Accuracy:  0.9862
Precision:  0.9550102249488752
Recall:  0.9085603112840467
```

Here is the results of running the classifier, the first array ranging from 0-9 is how many of that digit was misclassified as a 7(or in 7s case not classified properly). The number I thought would be the worst was a 2, since the bottom could be short and be very similar to a 7.

The image confused most with my number(that wasn't my number) ended up being a 9 which actually isn't that surprising since a slimmed top loop on a 9 could look like a 7, the least misclassified number was a 0 followed by a 1. 0 wasn't that surprising but I thought 1 would be higher due to the similar vertical line.

Worst predictions:

Here are the worst predicted mistakes according to the classifiers output.

This was calculated by using the following:

For the worst 7 miscalculated(the worst FN) the index of the lowest predicted value was taken using argmin on y_predictions. Then using the imagelist and applying that index to find and save the worst one.

```
imageList = x_validation[labels_validation == i]
```

```
worst=torch.argmin(y_pred.float()).item()
show_image(imageList[worst], "FN%d.png"%(i), scale = SCALE_01)
```

For the worst numbers calculated as my number(FP) the code was only slightly changed to argmax.

```
worst=torch.argmax(y_pred.float()).item()
show_image(imageList[worst], "FP%d.png"%(i), scale = SCALE_01)
```

Overall, most of these make sense such as the light handed left side of the 0, but some like 4,5,6 I cant really see what happened. Also the 7 makes some sense since it was stylized with a horizontal line.

Part 2: The Gan

My generator network was created similar to my classifier but in reverse order.

```
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1 = nn.Linear(100, 128)
        self.linear2 = nn.Linear(128, 256)
        self.linear3 = nn.Linear(256, 512)
        self.linear4 = nn.Linear(512, 784)
        self.dropout = nn.Dropout(0.6)
        self.relu1 = nn.LeakyReLU()
        self.relu2 = nn.LeakyReLU()
        self.relu3 = nn.LeakyReLU()
        self.sig = nn.Sigmoid()
        self.tanh = nn.Tanh()
    def forward(self, x):
        h = self.linear1(x)
        h = self.relu1(h)
        h = self.dropout(h)
        h = self.linear2(h)
        h = self.relu2(h)
        h = self.dropout(h)
        h = self.linear3(h)
        h = self.relu3(h)
        h = self.dropout(h)
        h = self.linear4(h)

        return self.sig(h)
```

It was made up of linear layers going from 100 -> 128 -> 256 -> 512 -> 784. LeakyReLU followed by a dropout of 0.6 was applied after every linear layer change. The output applied a Sigmoid function to get values between 0 and 1.

Anderson, Christopher
Bronco ID: 014105585

Training Structure:

```
n = 20
#Disc Train
n1 = 50
#gen Train
n2 = 50
```

```
gen_optimizer = torch.optim.Adam(G.parameters(), lr = 0.0001)
disc_optimizer = torch.optim.Adam(D.parameters(), lr = 0.0001)
```

Here is the training structure applied to my program. Due to the massive dropout and low learning rate of 0.0001, I had to raise the loops to achieve a loss of near or less than 0.1 per round and these were the number that allowed for that. I really liked the high n count with the very low learning rate as it gave a decent evolution of images, it also prevented any loss value from spiking- lower numbers with higher learning rate cause my false detection to spike to 100.0 fairly quickly.

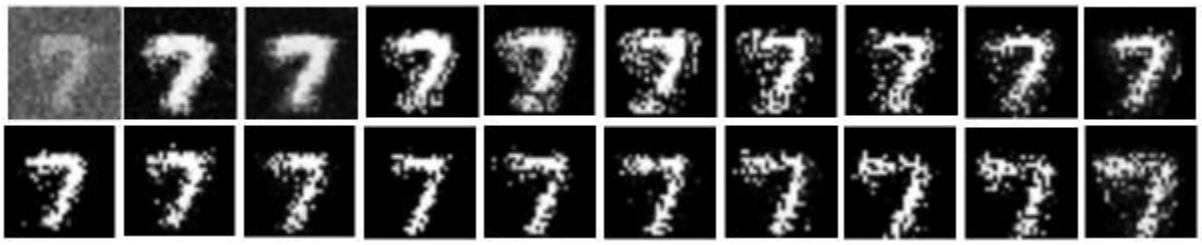
Example of loop 20 training:

```
 25 X_True:  tensor(0.0847, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0859, grad_fn=<BinaryCrossEntropyBackward>)
7] 26 X_True:  tensor(0.0845, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0815, grad_fn=<BinaryCrossEntropyBackward>)
   27 X_True:  tensor(0.0823, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0895, grad_fn=<BinaryCrossEntropyBackward>)
   28 X_True:  tensor(0.0798, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0688, grad_fn=<BinaryCrossEntropyBackward>)
   29 X_True:  tensor(0.0784, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0785, grad_fn=<BinaryCrossEntropyBackward>)
   30 X_True:  tensor(0.0797, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0598, grad_fn=<BinaryCrossEntropyBackward>)
   31 X_True:  tensor(0.0798, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0561, grad_fn=<BinaryCrossEntropyBackward>)
   32 X_True:  tensor(0.0788, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0415, grad_fn=<BinaryCrossEntropyBackward>)
   33 X_True:  tensor(0.0789, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0400, grad_fn=<BinaryCrossEntropyBackward>)
   34 X_True:  tensor(0.0792, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0335, grad_fn=<BinaryCrossEntropyBackward>)
   35 X_True:  tensor(0.0814, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0351, grad_fn=<BinaryCrossEntropyBackward>)
   36 X_True:  tensor(0.0811, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0358, grad_fn=<BinaryCrossEntropyBackward>)
   37 X_True:  tensor(0.0808, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0448, grad_fn=<BinaryCrossEntropyBackward>)
   38 X_True:  tensor(0.0799, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0396, grad_fn=<BinaryCrossEntropyBackward>)
   39 X_True:  tensor(0.0837, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0248, grad_fn=<BinaryCrossEntropyBackward>)
   40 X_True:  tensor(0.0809, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0309, grad_fn=<BinaryCrossEntropyBackward>)
   41 X_True:  tensor(0.0816, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0390, grad_fn=<BinaryCrossEntropyBackward>)
   42 X_True:  tensor(0.0819, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0215, grad_fn=<BinaryCrossEntropyBackward>)
   43 X_True:  tensor(0.0800, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0137, grad_fn=<BinaryCrossEntropyBackward>)
   44 X_True:  tensor(0.0802, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0175, grad_fn=<BinaryCrossEntropyBackward>)
   45 X_True:  tensor(0.0807, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0161, grad_fn=<BinaryCrossEntropyBackward>)
   46 X_True:  tensor(0.0784, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0193, grad_fn=<BinaryCrossEntropyBackward>)
   47 X_True:  tensor(0.0780, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0159, grad_fn=<BinaryCrossEntropyBackward>)
   48 X_True:  tensor(0.0758, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0203, grad_fn=<BinaryCrossEntropyBackward>)
   49 X_True:  tensor(0.0752, grad_fn=<BinaryCrossEntropyBackward>) X_false:  tensor(0.0120, grad_fn=<BinaryCrossEntropyBackward>)
```

Anderson, Christopher
Bronco ID: 014105585

```
13 tensor(1.1022, grad_fn=<BinaryCrossEntropyBackward>)
14 tensor(1.1522, grad_fn=<BinaryCrossEntropyBackward>)
15 tensor(0.9945, grad_fn=<BinaryCrossEntropyBackward>)
16 tensor(0.9237, grad_fn=<BinaryCrossEntropyBackward>)
17 tensor(1.0743, grad_fn=<BinaryCrossEntropyBackward>)
18 tensor(1.0586, grad_fn=<BinaryCrossEntropyBackward>)
19 tensor(1.0270, grad_fn=<BinaryCrossEntropyBackward>)
20 tensor(1.0287, grad_fn=<BinaryCrossEntropyBackward>)
21 tensor(0.9602, grad_fn=<BinaryCrossEntropyBackward>)
22 tensor(0.9473, grad_fn=<BinaryCrossEntropyBackward>)
23 tensor(0.8535, grad_fn=<BinaryCrossEntropyBackward>)
24 tensor(0.7235, grad_fn=<BinaryCrossEntropyBackward>)
25 tensor(0.6158, grad_fn=<BinaryCrossEntropyBackward>)
26 tensor(0.5593, grad_fn=<BinaryCrossEntropyBackward>)
27 tensor(0.4962, grad_fn=<BinaryCrossEntropyBackward>)
28 tensor(0.4320, grad_fn=<BinaryCrossEntropyBackward>)
29 tensor(0.3625, grad_fn=<BinaryCrossEntropyBackward>)
30 tensor(0.3357, grad_fn=<BinaryCrossEntropyBackward>)
31 tensor(0.3227, grad_fn=<BinaryCrossEntropyBackward>)
32 tensor(0.2871, grad_fn=<BinaryCrossEntropyBackward>)
33 tensor(0.2847, grad_fn=<BinaryCrossEntropyBackward>)
34 tensor(0.2812, grad_fn=<BinaryCrossEntropyBackward>)
35 tensor(0.2490, grad_fn=<BinaryCrossEntropyBackward>)
36 tensor(0.2482, grad_fn=<BinaryCrossEntropyBackward>)
37 tensor(0.2402, grad_fn=<BinaryCrossEntropyBackward>)
38 tensor(0.2573, grad_fn=<BinaryCrossEntropyBackward>)
39 tensor(0.2490, grad_fn=<BinaryCrossEntropyBackward>)
40 tensor(0.2320, grad_fn=<BinaryCrossEntropyBackward>)
41 tensor(0.2434, grad_fn=<BinaryCrossEntropyBackward>)
42 tensor(0.2276, grad_fn=<BinaryCrossEntropyBackward>)
43 tensor(0.2586, grad_fn=<BinaryCrossEntropyBackward>)
44 tensor(0.2595, grad_fn=<BinaryCrossEntropyBackward>)
45 tensor(0.2292, grad_fn=<BinaryCrossEntropyBackward>)
46 tensor(0.2266, grad_fn=<BinaryCrossEntropyBackward>)
47 tensor(0.2081, grad_fn=<BinaryCrossEntropyBackward>)
48 tensor(0.1970, grad_fn=<BinaryCrossEntropyBackward>)
49 tensor(0.2281, grad_fn=<BinaryCrossEntropyBackward>)
```

Input sample: 

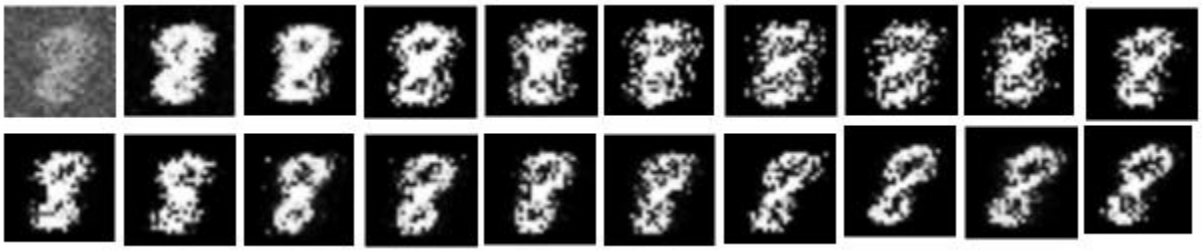


Above is the result of 20 training loops and the evolution of the generated images, there is one image sample from each loop.

I think there are some that look decent such as the image numbers 2,3,4, 11-16. So roughly 9/20, the rest tend to become weird blotchy shapes or very pixelated.

Anderson, Christopher
Bronco ID: 014105585

Second Test on 8s:



Overall the 8s looked much worse than the 7s most likely due to their complexity, the last 3 are the only ones that I think look good giving a 3/20. But there are a few more on the bottom row that could be called decent.

Results:

Overall most the images all looked different from each other, although they did have a pattern of becoming very fat and pixelated and thinning in both the 7s and the 8s.