Project 3: MNIST GAN

Jannik Haas
WPI Data Science
Worcester Polytechnic University
100 Institute Road, Worcester MA, 01609
jbhaas@wpi.edu

# INTRODUCTION

Generative adversarial networks (GANs) have become increasingly popular and through the combination of a generator and discriminator network these deep neural networks are able to generate a vast amount of images and texts. In the scope of this project we will create a GAN that is able to generate all ten handwritten digits included in the MNIST dataset. We will also test the best performing model on the Fashion MNIST dataset as well as the keras CIFAR10 dataset.

# SET OF EXPERIMENTS PERFORMED

## Network Structure

### Discriminator

For the discriminator, the goal is a simple binary classification problem. Since we are looking at images of handwritten digits I decided to use a Convolutional Neural Network. After researching different complexities of models and considering the relative simplicity of the MNIST dataset I decided to use 2 convolutional layers with 32 and 64 filters respectively. I experimented with using a more complex network, however these simply created more computational costs and didn't increase the overall model's performance. I used a stride of two for downsampling the images and a fully connected output layer with a sigmoid activation due to the binary classification nature of the problem. I also used binary cross entropy loss since it's a binary classification problem.

### Generator

#### *Fully Connected Neural Network*

The first network structure I experimented with was a simple FC linear network that simply reshaped the final layer into 64 28x28 filters or feature maps. These maps were then fed through a convolutional layer to result in one final image. From research I discovered that the tanh activation function is the most popular one used for the generator's final output and ReLU and Leaky ReLU tend to perform the best within both the generator and discriminator. For all the experiments I performed I used LeakyReLU activation between each layer. A network with four linear layers was able to generate images fairly well, however depending on the initialization it was only able to recreate a single image as shown in Figure 1. Depending on the epoch it switched between different digits as well.

Figure 1: Generated images from a simple Fully Connected Neural Network

### *Conv2D Transpose*

To increase the complexity of the model I wanted to also use convolutional layers in the generator. Since the generator and discriminator are opposing forces, the convolutional layers should work well with the generator as well. Using Conv2DTranspose, I started with a fully connected layer that output 7*7*feature map size (ranging from 32 to 512). Similarly to the fully connected network, we aim to generate a large number of feature maps so each filter map learns a specific part of the image. I experimented with 32 up to 512 feature maps and discover that 128 resulted in the best tradeoff of computational costs and performance. With a convolutional 2d transpose layer and a stride of 2, each layer up samples the image to result in twice the size of the original image. Since we need a 28x28 final image, I used two Conv2DTranspose layers, both with strides of 2. This model performed well and was able to create all ten of the handwritten digits, however using the sample plotting some of the images were still very blurry or not recognizable.

### *UpSampling2D*

The Keras UpSampling2D layer similarly to the Conv2DTranspose layer upsamples an each image to double its size, however unlike its convolutional counterpart, there are no convolutions learned and it simply up samples by replicating the same pixel around itself. Surprisingly this model was actually the best performing model for the MNIST dataset, even though I was expecting the complexity of the transpose layers to outperform the simple up sampling layers.

## Hyperparameters

I used a learning rate of 0.0002 which has been proven to work well with GANs. I experimented with a wide variety of batch sizes and found that between the smaller batch sizes the

performance didn't vary greatly, however large batch sizes were unable to create a good performing model and most of the time although the final digits were mostly recognizable, it was a very gray background unlike the traditional black MNIST background. For the majority of the experiments I used a batch size of either 64 or 128 meaning that each batch contained either 32 or 64 of each fake and generated samples respectively.
For the activation functions I used LeakyReLU for the hidden layers with the default alpha of 0.3. For the generator final output layer I used the tanh activation function and for the discriminator a sigmoid activation. For the optimizer I used the Adam optimizer with the learning rate of 0.0002 and beta1 of 0.5. I also used 100 epochs however most of the good generators were able to generate recognizable images within 20 to 30  epochs of training.

### Special Skills

Using some tips from GanHacks [1] I was able to experiment and create a good performing generator. I used LeakyReLU activations for the hidden layers, the tanh function for the final generator output layer, and noisy labels to train the discriminator. The noisy labels provided some improvement of performance, however not a great amount.
The most interesting find I discovered during the experimenting had to do with normalizing the inputs. Since a large part of the training depends on the original MNIST dataset, I originally normalized the data to between 0 and 1. However I wanted to also try normalizing it between -1 to 1 since it was mentioned in GanHacks [1]. Interestingly, the UpSampling2D models only performed well when being trained with normalized data between -1 to 1 and the Conv2DTranspose model only performed well when trained with data that was normalized between 0 to 1.

## Visualizations

Figure 2: Best Generator Output

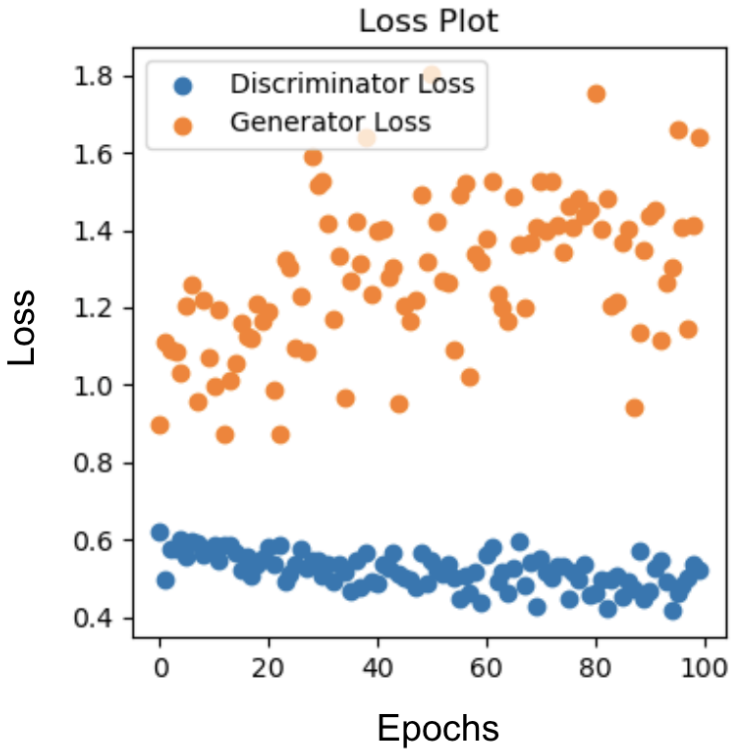



Figure 3: Loss Plot of Best Generator

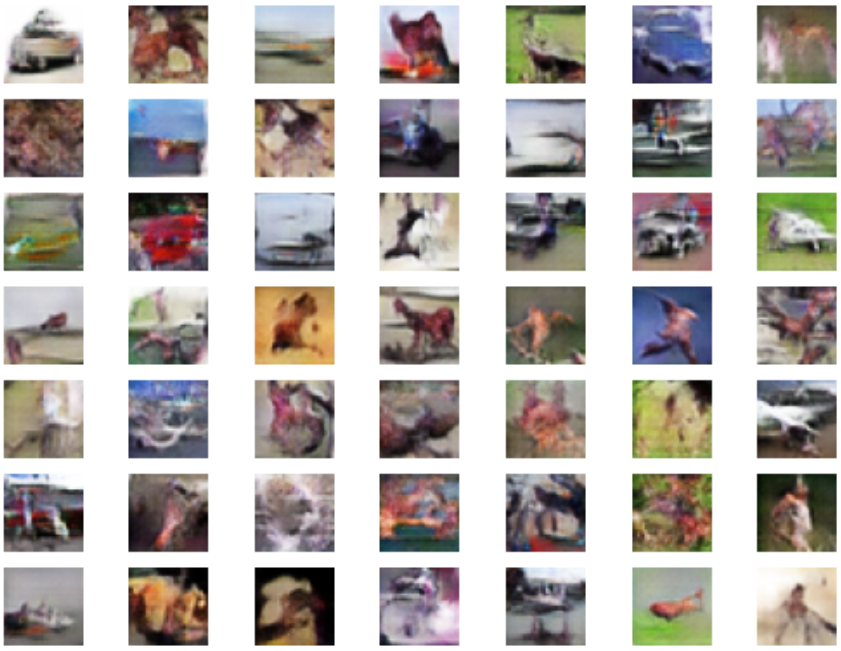
## OTHER DATASETS

### Fashion MNIST

Due to the model's good performance on the original MNIST dataset I wanted to test the model on other datasets. The Fashion MNIST dataset is a dataset consisting of ten different clothing items. It uses the same size of gray scale images so I was able to use the exact same model and plotting code to generate the images. The model required no tweaking to create a good generator with the following output:

## CIFAR10 Dataset

To increase the complexity I also wanted to test a more complex image dataset. The CIFAR10 dataset from Keras consists of 50000, 32 by 32 training images. These images are of ten different classes including airplanes and frogs. For the increased complexity of the CIFAR dataset and differing dimensions I used three Conv2DTranspose layers to upsample the images from 4 to 8 to 16 to 32. I also added one more convolutional layer to the discriminator due to the increased complexity. As you can see it was able to generate some recognizable images, however most of the images were unrecognizable.



# REFERENCES

[1] https://github.com/soumith/ganhacks
[2] https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-an-mnist-handwritten-digits-from-scratch-in-keras/
[3] https://keras.io/api/