Vignesh Ravindranath
vgr325

Generative Adversarial Networks (GANs) on the MNIST Dataset

1 Overview

The purpose of this assignment is to build a generative adversarial network to generate images similar to the MNIST handwritten images. Generative adversarial networks or GANs are a combination of two networks – generator and discriminator. The discriminator network is trained on the MNIST dataset and is trying to identify whether images are fake or real. The goal of the generate network is to generate fake images and 'trick' the discriminator network to thinking they are real images. Over time, the generator will become better at creating fake images (see GIF above). Application of GANs include creating photorealistic images and for fraud detection.

2 Method

2.1 Libraries

- tensorflow (version 2.2.0) – keras.layers, keras.utils
- matplotlib – pyplot
- seaborn
- sklearn – metrics, confusion_matrix
- numpy
- PIL
- glob

- time
- imageio
- git+https://github.com/tensorflow/docs

Libraries PIL, glob, imagio, and tensorflow docs were necessary to create a GIF of how the GANs generated images evolved over time.

2.2 Loading and Preparing MNIST Dataset

The MNIST dataset has 60,000 training images and 10,000 testing images. All training images were used to train the feed forward deep neural network (baseline) and fine-tune the discriminator network. Similarly, all testing images were used to evaluate the networks.

2.3 Feed Forward Deep Neural Network (Baseline)

The baseline network consisted of two 2D convolutional with `LeakyReLU` activation and one dense layer with `softmax` activation. The input and output sizes of this CNN was `(28,28,1)` and `(10)` respectively to classify the MNIST dataset. It was optimized using `Adam` and loss was calculated using `categorical crossentropy`. See summary below.

```
Model: "sequential_37"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 14, 14, 64)        1664
_________________________________________________________________
leaky_re_lu_29 (LeakyReLU)   (None, 14, 14, 64)        0
_________________________________________________________________
dropout_20 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 7, 7, 128)         204928
_________________________________________________________________
leaky_re_lu_30 (LeakyReLU)   (None, 7, 7, 128)         0
_________________________________________________________________
dropout_21 (Dropout)         (None, 7, 7, 128)         0
_________________________________________________________________
flatten_11 (Flatten)         (None, 6272)              0
_________________________________________________________________
dense_31 (Dense)             (None, 10)                62730
=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```

2.4 Generator and Discriminator Networks

The generator network consisted of four layers: a dense layer and three 2D convolutional layers. The generator took in noise input with size `(100,)` and converted it into an output image with size `(28,28,1)`, similar to the input of the discriminator. See summary below.

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 12544)             1254400
_________________________________________________________________
batch_normalization_6 (Batch (None, 12544)             50176
_________________________________________________________________
leaky_re_lu_16 (LeakyReLU)   (None, 12544)             0
_________________________________________________________________
reshape_2 (Reshape)          (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_transpose_6 (Conv2DTr (None, 7, 7, 128)         819200
_________________________________________________________________
batch_normalization_7 (Batch (None, 7, 7, 128)         512
_________________________________________________________________
leaky_re_lu_17 (LeakyReLU)   (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose_7 (Conv2DTr (None, 14, 14, 64)        204800
_________________________________________________________________
batch_normalization_8 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
leaky_re_lu_18 (LeakyReLU)   (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_transpose_8 (Conv2DTr (None, 28, 28, 1)         1600
=================================================================
Total params: 2,330,944
Trainable params: 2,305,472
Non-trainable params: 25,472
_________________________________________________________________
```

The discriminator network is similar to the baseline network, except the last dense layer has a size of `(1)` instead of `(10)` to be able to classify the generated image as real or fake. See summary below.

```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 14, 14, 64)        1664
_________________________________________________________________
leaky_re_lu_27 (LeakyReLU)   (None, 14, 14, 64)        0
_________________________________________________________________
dropout_18 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 7, 7, 128)         204928
_________________________________________________________________
leaky_re_lu_28 (LeakyReLU)   (None, 7, 7, 128)         0
_________________________________________________________________
dropout_19 (Dropout)         (None, 7, 7, 128)         0
_________________________________________________________________
flatten_10 (Flatten)         (None, 6272)              0
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 6273
=================================================================
Total params: 212,865
Trainable params: 212,865
Non-trainable params: 0
_________________________________________________________________
```

Both networks were optimized using `Adam`, and loss was measured using `binary crossentropy`.

The GAN was trained over 100 epochs. At each epoch, the generator would generate a fake image (`tf.random.normal`) and would be used to train the discriminator and generator networks. Both models would optimize their parameters by determining the loss between a true and fake image.

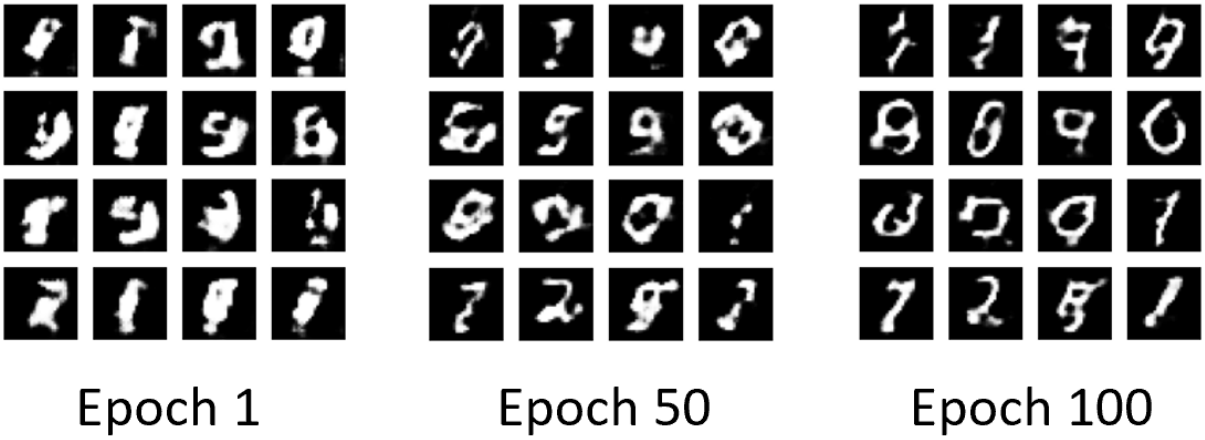
3 Results and Discussion

3.1 Feed Forward Deep Neural Network

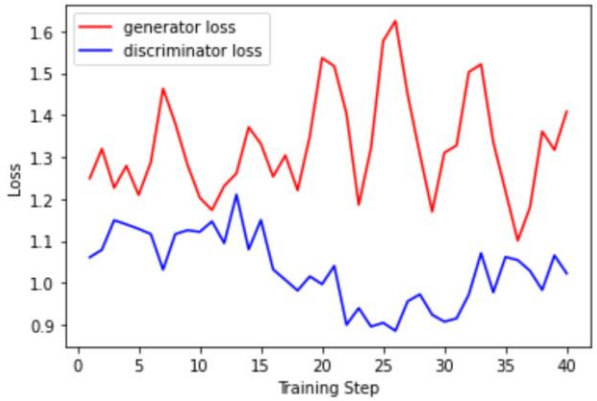
The baseline network was trained on all 60,000 training images provided by the MNIST dataset over 10 epochs. The training loss and accuracy were found to be `0.0498` and `0.9840`, respectively. The testing loss and accuracy was found to be `0.0461` and `0.9848`, respectively.

3.2 Loss Evaluation of Generator and Discriminator Networks

Over time, the GAN began producing images that resembled handwritten images! Below are images generated by the generator network at 1, 50, and 100 epochs. The evolution of the generated images seemed to produce numbers that resemble 0, 1, 2, 7, 8, and 9, as well as some indistinguishable numbers. Overall, the GAN was very impressive!



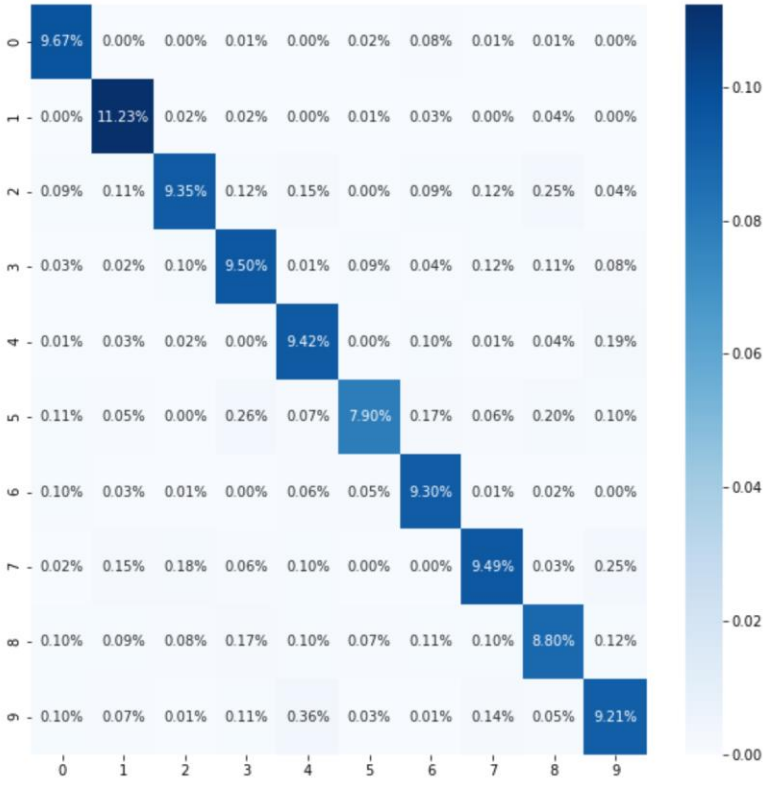
After 100 epochs of training, the loss of the generator and discriminator models were around `1.3` and `1.0`, respectively. The loss of the discriminator was found to always be less than the loss of the generator. See plot below.



3.3 Fine-tune Discriminator Network

Lastly, the discriminator network was fine-tuned on the MNIST dataset. This was done by removing the last dense layer of the discriminator network and adding a new dense layer with an output of `(10)`. The parameters for the previous two 2D convolutional layers were frozen (`trainable = False`) so the last layer could be fine-tuned only. The network had a training loss and accuracy of `0.2637` and `0.9235`, respectively. Similarly, it had a test loss and accuracy of `0.2185` and `0.9387`, respectively. Overall, it performed worst than the baseline network. This makes sense, because it was trained on the fake images produced by the generator, which did not always produce a distinguishable image, even after 100 epochs of training.

The confusion matrix below shows where the discriminator struggled to classify numbers. Specifically, it seems as though the discriminator misclassified: 1's as 2's and 7's, 2's as 7's, 3's as 2's and 8's, 4's as 9's, 6's as 5's, 7's as 9's, 8's as 3's, and 9's as 4's and 7's.



4 References

[1] https://www.tensorflow.org/tutorials/generative/dcgan

[2] https://towardsdatascience.com/mnist-cnn-python-c61a5bce7a19