Programming Assignment 1: Classification

Joe Johnson CIS 472/572 Dr. Humphrey Shi

Foreword

I had many troubles with specific implementations, but found the tutorials provided to be quite helpful. I hope that what is found here will be sufficient.

1. Introduction

The following results are all trial runs of different methods, networks, and optimizations of the Canadian Institute for Advanced Research 10 Category (CIFAR-10) and National Institute of Standards and Technology (MNIST) dataset. All code is given in the included ipynb file, written in python, and making use of the PyTorch library. Testing done was using Google Colab Pro, so cuda implementation was tested on an Nvidia Tesla P100 (16GB version).

1.1. DCGAN

Thanks to the provided tutorial, I was able to implement a Direct Convolutional Generative Adversarial Networks (DCGAN) on the CIFAR-10 and MNIST datasets [1]. The base settings had the images at a size of 64 x 64 pixels.

1.2. CIFAR-10 Differing Image Sizes

For differing the image sizes, I used 32x32, 128x128, 256x256 and the base-case of 64x64. The 64x64 and 32x32 had similar results (figures 1 and 2 respectively) with the generator averaging around a loss of 4 with a very slight (insignificant) decrease over time and a discriminator loss hovering around .5 to 1 on loss. The 128x128 training case almost immediately saw the discriminator hit a loss of 100 and the generator bottoming out on loss (figure 3). For 256x256, there was interesting results in the generator loss going from 10 to around 40 by iteration 50 and staying steady until iteration 660; where it can be seen dipping to 30, spiking almost immediately to 80 and then bottoming out at 0 (figure 4). Meanwhile, the discriminator loss sees the same effects happen, but in reverse with the loss starting around 5, hitting 0 by iteration 50, staying right around 0 until iteration 660 where the loss jumped to just above 20, then quickly fluctuating between 0 and 100 loss for a few iterations before staying at 100 loss. The comparison between the images generated by these differing image sizes can be seen in comparison to the actual data set in figures 5-9.

1.3. MNIST Differing Image Sizes

The results here were similar to CIFAR-10, in that the larger images had issues in their prediction; however, the graphs saw much different trends. The 64x64 sees a sudden jump then plateau at iteration 450 (figure 10). Also, the 32x32 has much more volatile movements (figure 11). For 128x128, it was almost identical in its results to the CIFAR-10 128x128. And the 256 x 256 saw the same results as the 128x128 did in both datasets.

The generated vs real images can be seen in figures 15 – 19.

1.4. K-Means

I implemented a K-Means Clustering that tested the difference in loss and accuracy on CIFAR-10 and MNIST datasets [2]. This method takes images and groups them with similar images. The accuracy was then checked by the number of pictures within a category that are labeled correctly. The results of my implementation had an accuracy of only 23.79%.

1.5. PCA

I implemented Principal Component Analysis (PCA) on the CIFAR-10 dataset using Random Forest, K-Nearest Neighbors, Logistic Regression and Support Vector Machine classifiers [3]. Through this I found Random Forest and K-Nearest (as I implemented them at least) to have the lowest accuracy. Logistic Regression had a significantly higher accuracy but failed to reach the accuracy of SVM across multiple trials; however, the calculation time of SVM was significantly longer. In fact, the calculation times seem to correspond with the accuracy found. Random forest took training and predicting took roughly the same amount of time as KNN training. Then, KNN Predicting took roughly 3x as long, corresponding with the calculation time of Logistic Regression's training and predicting, respectively. The SVM took over 3x as long in both training and predicting when compared to Logistic Regression in every trial.

1.6. Linear comparison

This implementation was based on a tutorial provided by the University of Virginia and used a Softmax Classifier [4]. I ran out of time to compare the results produced to the PCA implementation

1.7. Conclusion

I found that a lack of background in machine learning had led to many footfalls along my journeys with this assignment. I tried to implement all aspects I could, yet I might have failed to reach certain requirements as time constraints were an issue. I found some results particularly interesting, however, such as when implementing DCGAN I expected larger photo sizes to see better prediction results (as there are more data being presented). I now understand that with more data present, more data are required to be predicted; thus, adding complication which ultimately led large image sizes to fail drastically.

I also had particular interest in the K-Means Clustering algorithm. When researching I found several examples that implemented graphs of random images and moved them into clusters among these x-y plots which I found visually quite intriguing. My implementation fails to display this in its results, but after seeing these visualizations I have developed a recognition of the beauty that goes on behind the scenes.

Ultimately, this was an interesting exploration into machine learning. One in which, although I struggled greatly, have developed new knowledge and appreciation for methods discussed in class.

References

[1] https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

[2] https://github.com/ZeyadZanaty/image-clustering

[3] https://github.com/wikiabhi/Cifar-10/blob/master/cifar10_pca.ipynb

[4] https://www.cs.virginia.edu/~vicente/recognition/notebooks/linear_classifier_sgd_lab.html

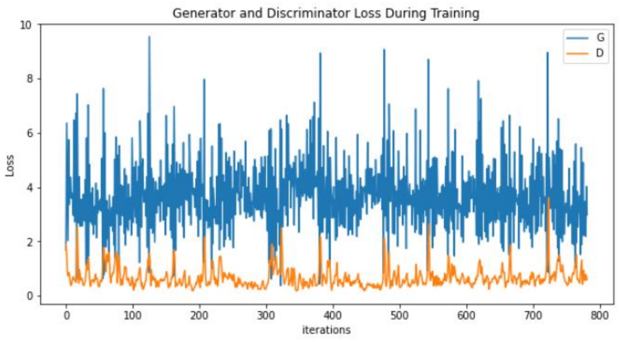

Figure 1: DCGAN Loss on 64x64

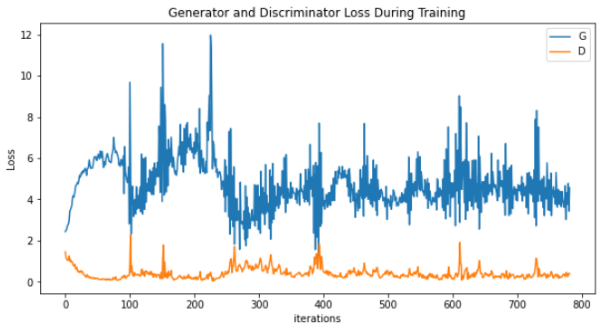

Figure 2: DCGAN Loss on 32x32

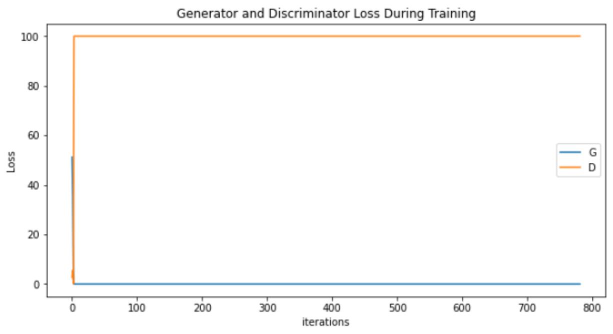

Figure 3: DCGAN Loss on 128x128

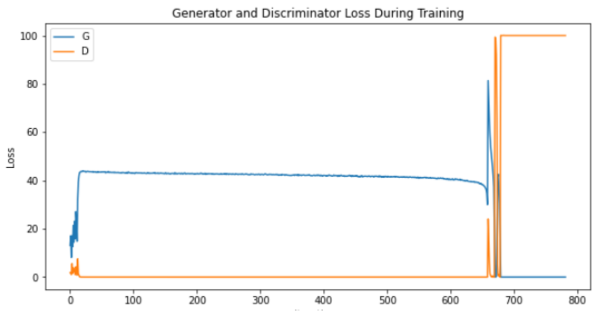

Figure 4: DCGAN Loss on 256x256

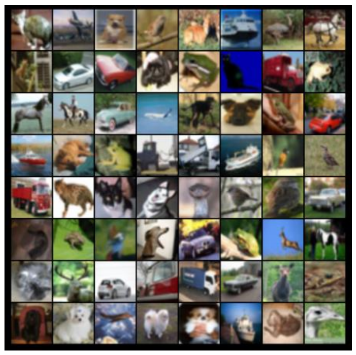

Figure 5: Actual Images of CIFAR-10

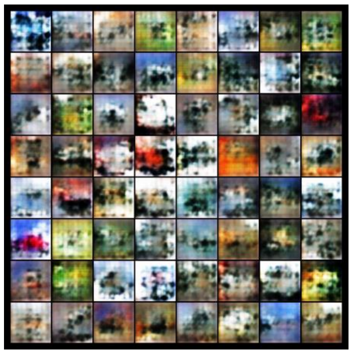

Figure 6: Generated Images from 64x64

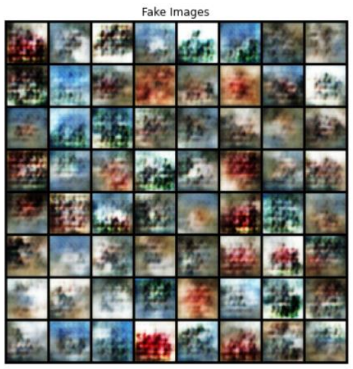

Figure 7: Generated Images from 32x32

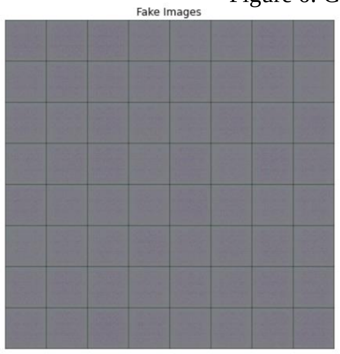

Figure 8: Generated Images from 128x128

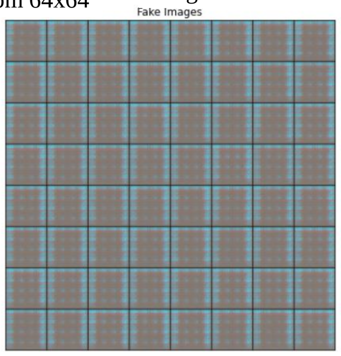

Figure 9: Generated Images from 256x256

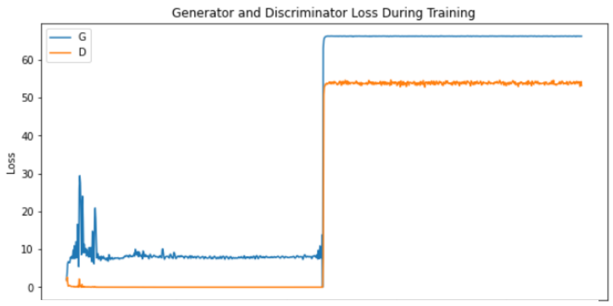

Figure 10: DCGAN Loss on 64x64

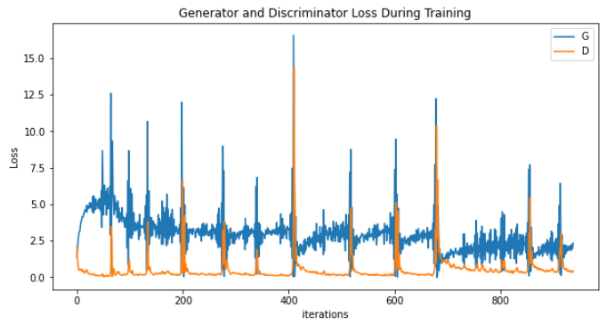

Figure 11: DCGAN Loss on 32x32

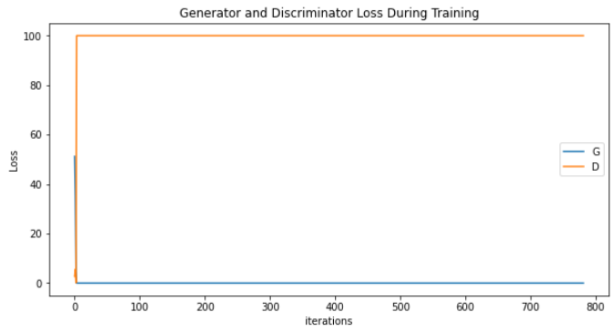

Figure 12: DCGAN Loss on 128x128

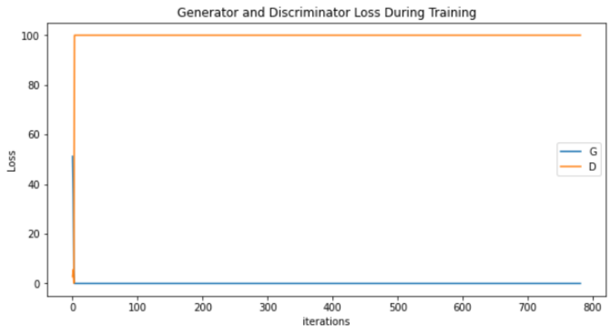

Figure 13: DCGAN Loss on 256x256

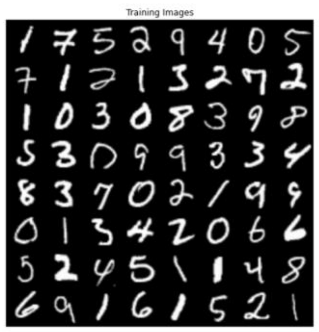

Figure 14: Actual Images of MNIST

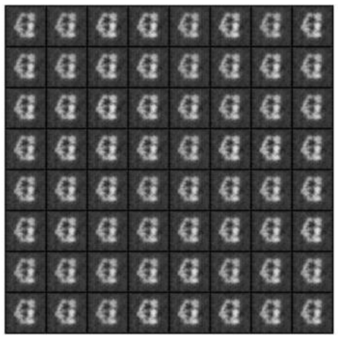

Figure 15: Generated Images from 64x64

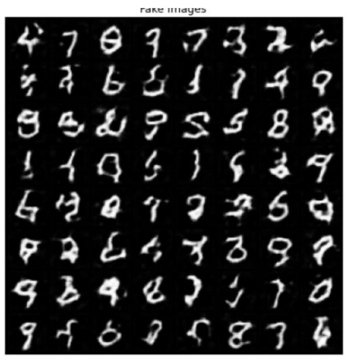

Figure 16: Generated Images from 32x32

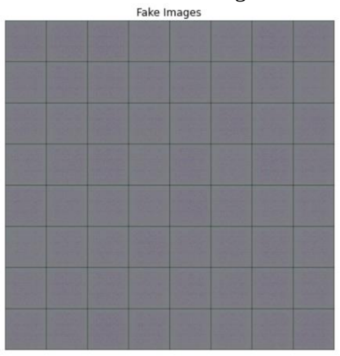

Figure 17: Generated Images from 128x128

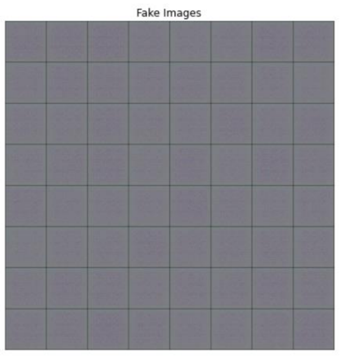

Figure 18: Generated Images from 256x256