FACE GENERATOR USING DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Columbian College of Arts & Sciences
The George Washington University, Washington D.C.

Students:
Renzo Castagnino
Aira Domingo
Professor: Amir Jafari
Date: April 2020

Keywords: Machine Learning, Convolutional Neural Networks (CNN), Deep Learning, Generative Adversarial Networks (GAN), Deep Convolutional Generative Adversarial Networks (DCGAN), fake images, face generator, PyTorch.

Abstract. *The ability to manipulate and recreate realistic images at scale has had a big impact on how modern societies perceive the truth about certain facts such as faking visuals, which is currently a trending topic. Artificial Intelligence and Machine Learning have allowed the creation of not only meaningful media, contributing to our society, but also fake images, news, propaganda, and other persuasive information. In this project we will review the theory behind the model that allows the creation of fake images, Deep Convolutional Generative Adversarial model (DCGAN). The main objective is to create a model in PyTorch, that can create fake faces that are unrecognizable as fake to the human eye.*

1. Table of Contents

2. Introduction

Nowadays, images and videos are one of the most common types of media shared by users on social networks such Facebook or Instagram. It is a very common practice to share content, but also consume it. That is the case for a large number of news and propaganda, either for commercial or political purposes, that is being shared in the media. Unfortunately, the manipulations of fake news and media and propaganda is also a trending topic, giving erroneous or persuasive information to the users. More specifically, in the case of images and videos, Machine Learning has played an important role, by developing more sophisticated models that allows the creation of very realistic images.

In the last years, we've seen an enormous improvement in how fake faces are being generated. Incredibly realistic fake faces made with modern Machine Learning techniques can easily trick any person, making them think as if it was a real person. In less than a decade, the techniques and models, such as Generative Adversarial Networks (GAN), are rapidly improving, creating detailed images in high resolution that are very hard to distinguish from a real person.

Ian Goodfellow developed the Generative Adversarial Network (GAN), a framework based on an adversarial process, where two neural networks compete against each other in a minimax game. The first player in a GAN is a generative model G that attempts to simulate the distribution of given data. The second is a discriminative model D which estimates the probability that a sample came from the given data rather than the generator G. The two models are trained simultaneously with each one getting better at their goal.

The generator takes in random noise taken from a Gaussian distribution as input that has been mapped to a latent space and creates a distribution. The discriminator then distinguishes which samples were taken from the distributions of the given data and which were the output of the generator. The objective of the generator is to fool the discriminator into identifying the generated distribution as a sample taken from the given data. Since both networks are differentiable, they are able to train using the same loss function. The discriminator is trained to maximize this loss function, while the generator is trained to minimize it.

The networks in GANs are defined by multilayer perceptrons, where each input uses one perceptron. Since the layers are fully connected, the number of parameters can get too high, especially when dealing with image data, leading to redundancy and inefficiency. This could then result in a model that has overfitted to the data while training and is ungeneralizable.

In 2014, Alec Radford introduced an extension to GANs by adding convolutional layers to both the generator and discriminator called Deep Convolutional Generative Adversarial Networks

(DCGANs). This was an improvement to the original GAN architecture as the strided convolutional layers replaced spatial pooling functions which allowed the generator to learn its own spatial upsampling and the discriminator its own spatial downsampling. They were also able to build a stable model by removing fully connected layers on top of convolutional layers. Overall, the constraints proposed by Radford for DCGAN allowed for a more stable training of the generator compared to GAN.

In this project, we will be focusing on how image data, specifically fake face images, can be generated for the purpose of fake media using the DCGAN architecture.

3. Main Problem

Fake media is a trending problem, and yet, very little work has been done about detecting invented images and video. Several studies are currently working on trying to identify how these fake images are created. While detecting fake images is not within the scope of this project, it is important to understand the theory behind the process of how these false images are being created. More specifically, the main problem to tackle in this project is how can we use Machine Learning to create fake images that are unrecognizable as fake by the human eye.

4. Deep Convolutional Generative Neural Network (DCGAN)

Deep Convolutional Generative Adversarial Networks (GANs) are a trending technique for both semi-supervised and unsupervised learning. Initially proposed in 2014, the main characteristic is having two networks (the generator and the discriminator) competing with each other. Basically, the discriminator network could be considered as a function that maps images to a probability of a real distribution: D: D(x) $\rightarrow$ (0, 1).

While having the generator G fixed, the discriminator performs gradient ascent and is trained to classify images as either being from the original data, or from the latent space Z. Once the discriminator has learned in the mini-batch process, we keep the discriminator frozen and continue training the generator with gradient descent. The graph bellow shows the algorithm proposed by Goodfellow in 2014:

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 1

Training a GAN network involves not only finding the parameters of the discriminator that maximizes the classification accuracy, but also finding the parameters of the generator which can optimally confuse the discriminator. The loss of the training is evaluated using a value function V (G, D) that depends on the discriminator and the generator. Basically, training the network means:

$$\max_{\mathcal{D}} \min_{\mathcal{G}} V(\mathcal{G}, \mathcal{D})$$

Where

$$V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{data}(\mathbf{x})} \log \mathcal{D}(\mathbf{x}) + \mathbb{E}_{p_g(\mathbf{x})} \log(1 - \mathcal{D}(\mathbf{x}))$$

During the training, the parameters of one model are updated, while the parameters of the other are fixed. According to Goodfellow, for a fixed generator, there is a unique optimal discriminator $D^*(X) = \frac{Pdata(X)}{Pdata(x)+Pg(x)}$. Furthermore, the generator G is optimal when $Pg(x) = Pdata(X)$ which is equivalent to the optimal discriminator predicting 0.5 or random predictions for all samples from X. In other words, the generator is optimal when the discriminator D is maximally confused and cannot distinguish real samples from fake ones. On an ideal process, the discriminator would be trained until it becomes optimal with respect to the current generator; then the generator is updated. However, in practice the discriminator might not be trained until optimal, but rather may only be trained for a certain number of iterations. Training a GAN model is challenging and often unstable, problems such as getting the pair of models to converge, the generative model collapsing to generate similar samples for different inputs, the discriminator loss converging quickly to zero,

giving no reliable path for the gradient to update the generator. Figure number 2 shows the overall diagram of a GAN architecture.

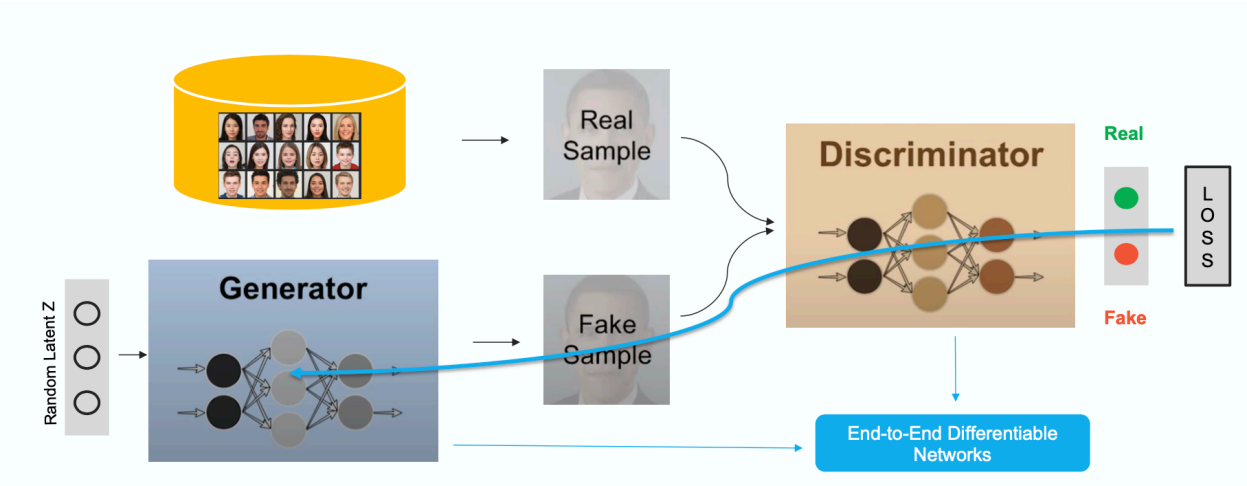

Figure 2

5. Dataset

The data that will be used for this project is the CelebA Dataset. This dataset is a free large-scale face attributes with more than 200,000 celebrity images, each with 40 attribute annotations. The images cover different pose variations and background colors. The pixels of the images are 178×218. The following is an example of some images obtained from the dataset:

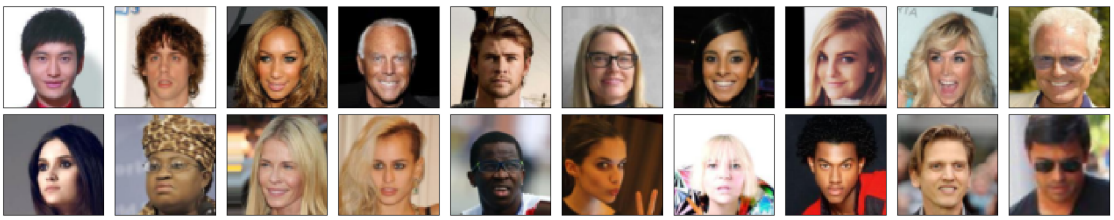

Figure 3

For the scope of this project, we will use all of the images in our training. Also, it is important to mention that all the images are not containing just the face, but some images contain an extra margin of background on the top. In this project, that problem was addressed by applying a simple CenterCrop transformation while loading the images in the Dataloader.

However, this issue could also be addressed by using a pre-trained model such as VGG and getting a more accurate crop of the face in the image.

Furthermore, in our model we have not filtered the images by their attributes. That means that, for instance, if you want to train the model just with faces, and if some of the images contain people with sunglasses, this could distort the model or create a bias. Another example would be if the model tries to create an image of a girl, but in the training batch there are pictures of men with beards. All of these variables should be considered in future models to improve the model.

6. Experimental setup

Using PyTorch, we implemented the DCGAN architecture to generate fake faces. The data was loaded from the Large-Scale CelebFaces Attributes (CelebA) Dataset Google Drive. The images in the dataset were already aligned by similarity transformation based on two eye locations and was also cropped to 178x218. We perform a center crop and resize the images to 32x32.

After loading the data, we wanted to make sure that the images were loaded correctly. Thus, we created a function to show the images with Pyplot. We then scaled the image to have pixel values in the range of -1 to 1 to match the output of the tanh activated generator.

The generator was designed to have 4 transposed convolutional layers. Guided by Radford's DCGAN model, each transposed convolutional layer included a batch normalization except for the last layer. This helps with the difficulties that result from poor initialization since it prevents the generator from collapsing all the samples to a single point. By not applying a batch normalization to the last layer, we are preventing sample oscillation and model instability. The hidden layers in the generator uses ReLU as the activation function because having a bounded activation allowed for faster learning to saturate the color space of the real data distribution. The output layer used a Tanh activation function. The generator took in random noise from a Gaussian distribution that has been mapped to a latent space as input and performed upsampling at each layer. Specifically, after the first transposed convolutional layer, the generator creates a 4x4 image, then 8x8, 16x16, and finally 32x32.

In the Discriminator, we created 3 convolutional layers, and one fully connected layer at the end which returns the logit (true or false) for the images that real or fake. Per common practice, the first convolutional layer doesn't include a batch normalization, but the other 2 layers include the batch normalization. By normalizing the input to have zero mean and a unit variance, this allowed us to create deeper layers. For the hidden units, we have used the Leaky ReLU as the activation function. Leaky ReLU allows the model to avoid saturation in the gradients because it has a lower bound slightly lower than zero. This means that we can still pass small gradient signals

for negative values. After each convolution layer, the height and width become half. For instance, after the first convolution 32X32 images will be resized into 16X16 and then to 8x8.

The discriminator D is trained to maximize the probability that it is correctly labeling the samples from the given data and the samples from generator G. The generator G is trained to minimize log(1-D(G(z))). Since only the second half of the function is dependent on G, only the second half is optimized to train generator G. Minimizing this part of the function is similar to maximizing D(G(z)), which means increasing the probability that discriminator D is incorrectly identifying the generated image as a real sample from the given data.

Thereafter, we defined our hyperparameters and initiated both the discriminator and the generator. We defined our vector latent space Z of size 100, with a gaussian distribution zero mean and unit variance. In case of our loss function, we defined it as the sum of the loss of the discriminator (loss for the real images), and the loss for the generator (fake images).

In a GAN architecture, we need to define 2 optimizers, one for the generator and one for the discriminator. This is because we need to keep updating them simultaneously to improve both networks. In both cases we have used Adam optimizer with a learning rate of 0.0002 as suggested in previous DCGAN papers.

During the training process, we will alternate between the discriminator and the generator. We trained the generator by alternating between the real and fake images with a mini batch process. We have also plotted the training losses for both the generator and discriminator. Figure number 4 shows the architecture of our model.

DCGAN MODEL

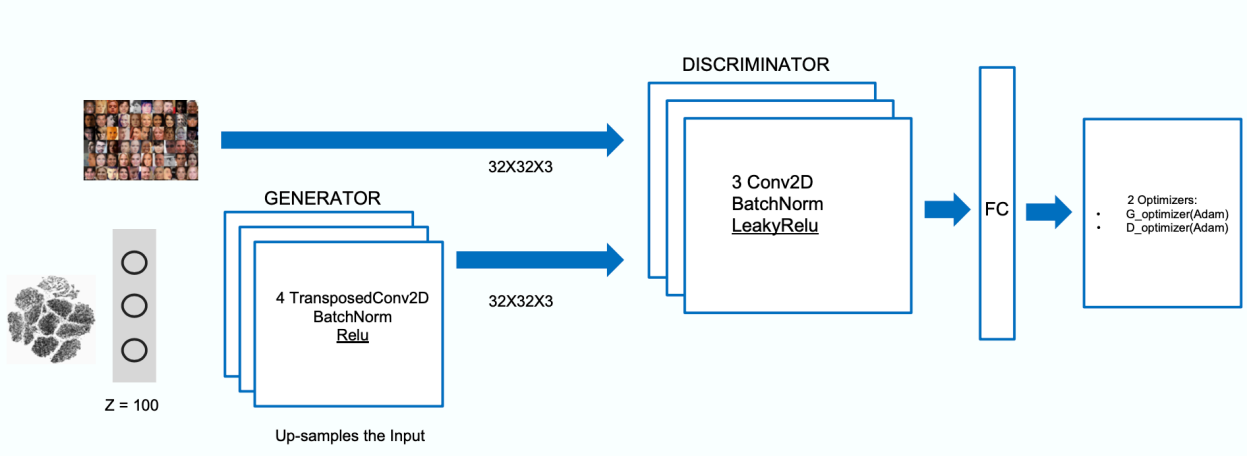

Figure 4

7. Results

Our initial results were not optimal as we expected, but considering the simplicity of the model, we can observe in figure 5 that the blurry faces are being created in very low resolution with some distortions. We need to consider that our model has as input a 32x32 pixel image size. Moreover, for this training, we used a batch size of 256 images, with 40 epochs. Furthermore, there is a bias in the dataset as all of the faces are from celebrities, and we have not filtered the images by their attributes.

Figure 5

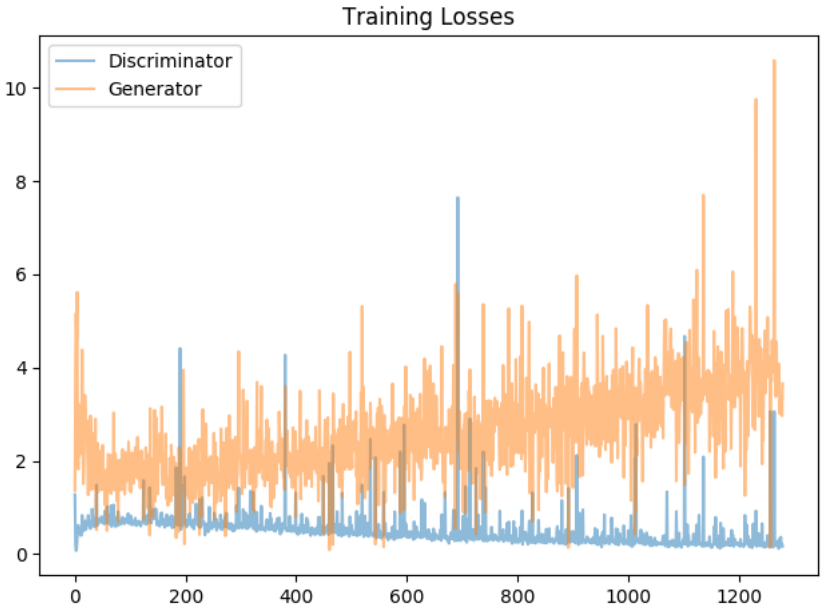

Figure 6

Our second model has a noticeable improvement compared to the first one, which can be seen in Figure 7. Basically, we changed the batch size to 128, and increased the learning rate to 0.0005, as well as the number of epochs to 30. By having a smaller batch size, the weights should be

updated more frequently. The learning rate was increased, and the number of epochs decreased to assist with the model's run time.

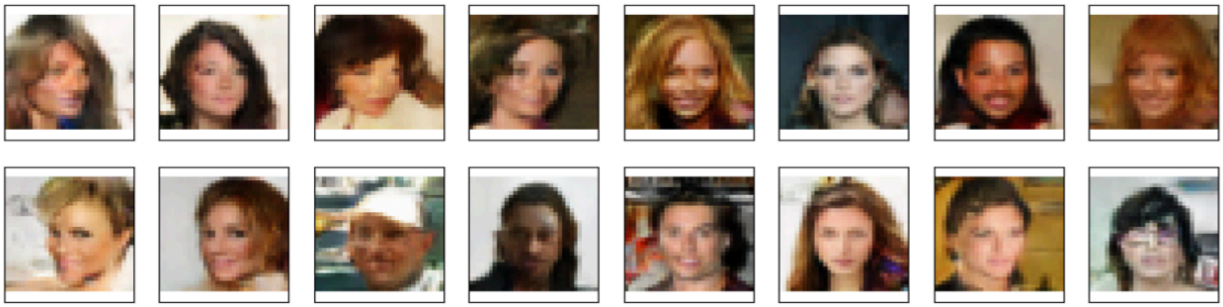

Figure 7

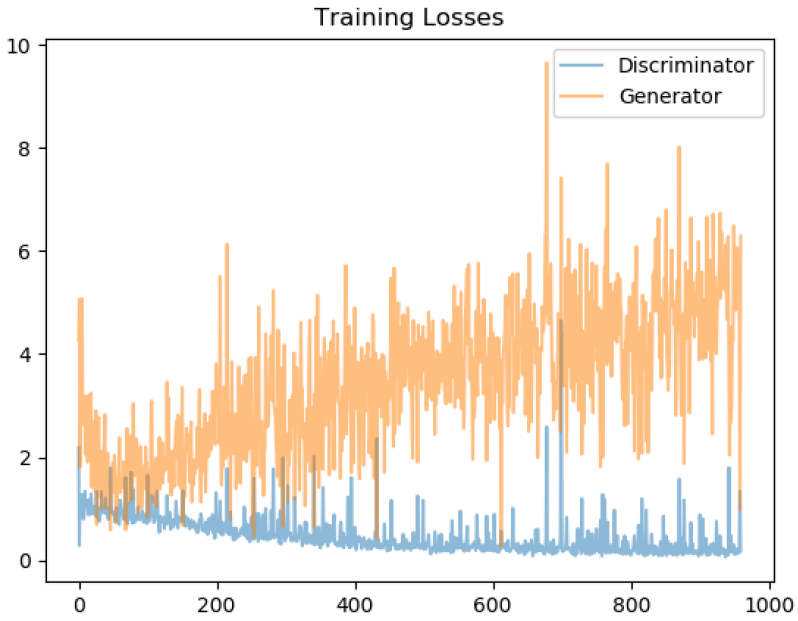

Figure 8

For both models, we observed that the training loss of the discriminator steadily declines, while the training loss of the generator increases, as depicted in Figure 6 and 8, which is inconsistent with the objectives of the network. Ideally, the generator loss should decrease while the discriminator loss increases. There is a lot of oscillation in the generator's loss, which could be due to its input of random noise to create the images that is fed to the discriminator. Since the generator's loss is increasing, meaning it is not training well, it makes sense that the discriminator's loss decreases as it is able to quickly learn how to differentiate the images coming from the real data and those that came from the generator. In these cases, the discriminator has the upper hand and the generator was not able to keep up to create images that would fool the discriminator.

On average, it took 3.5 hours to train our models. We also noticed that the images produced from a model that trained for 80 epochs did not have a drastic difference to one that was run for 30 epochs. Since training with 80 epochs took more than double the time to train 30 epochs with similar results, we decided to train with 30-40 epochs moving forward.

Moreover, in some of the other models we ran, we observed that the generator produced similar outputs for some of the images. This failure in GANs is called mode collapse. Sometimes, the generator creates a believable looking image that has fooled the discriminator and ends up learning to only produce that image since it will always want to create an output that is believable to the discriminator. If at any point in training the discriminator gets stuck at the local minimum, it is easy for the next iteration generator to output an image that is believable to the current discriminator. As the discriminator is being fooled, the weights for the generator will go unchanged. Since the hidden layers of the generator uses a ReLU activation function whose lower bound is 0, the gradient will approach 0 leading to a vanishing gradient and the generator will not learn.

8. Summary

The ability to create fake images using Machine Learning techniques such as GANs, is an emerging field that can have either a positive or negative impact in our society depending on how we use it. GANs is an algorithmic architecture that uses two neural networks competing with each other, that can generate new synthetic data that can pass for real data. It was proposed in 2014 by Ian Goodfellow and it one of the most interesting models in the last decade in Machine Learning. In fact, GANs can learn any distribution of data such as images, music, speeches, etc.

This project has reviewed the theory behind GAN and created a DCGAN network using PyTorch. Our model was composed by 3 convolutional layers for the discriminator and 4 transposed convolutional layers for the generator. For the discriminator, we used Leaky ReLU in the hidden layers and sigmoid in the output layer for the activation functions. As for the generator, we used ReLU in the hidden layers and Tanh in the output layer as the activation functions. For the dataset, the CelebA dataset was used containing pre-processed images of the faces of celebrities.

In terms of our results, after the training, our model was able to recreate fake faces. While the quality of the images is not very expressive, and the results are in very low resolution of 32x32 pixels, we can say that the main purpose of the project was achieved. It is important to mention that the quality of the original images was not in high resolution, constraining the capabilities of our model. Additionally, it is essential to note that the discriminator had the upper hand in the training of the model, instead of having the ideal zero-sum between the generator and discriminator.

9. Conclusions

In this project we have shown that with simple Machine Learning techniques, such as Generative Adversarial Networks, it is possible to create faces. To improve the model, further research can be done in state-of-the-art methods such as progressive growing, conditional GANs or BigGAN.

While the architecture of the DCGAN is important to create high-quality images, it is imperative to have good quality images as an input. CelebA dataset is a good start as the images are already preprocessed and is a mid-size dataset to manage. Other datasets such as VGGFace2 could also be used to create a better model, as it contains more than 3 million images which provides more samples for the training process.

We have seen that training a GAN can be very time-consuming. On a single GPU a GAN could take hours. In our case, training the model for 40 epochs took an average of 3.5 hours. However, in other cases, a realistic image of 1024x1024 could take up to 14 days. For future research, we would like to train our model using a higher quality dataset for a longer period of time, with a more complex architecture. Ideally, this would allow us to create more realistic images of faces. We could then extend the use of the model by applying vector arithmetic on certain attributes from a training dataset to control the features created on an image.

This project has shown us how feasible it is to create realistic faces just by receiving continuous feedback from a discriminator. With technology always evolving, it would seem almost impossible for the human eye to identify these kinds of generated images. Although this type of technology could have a positive impact if used in the right ethical way, this is not always the case, as it could also be used for the spread of fake media. A detector able to identify the most seemingly realistic fake media would then be needed to ensure that individuals are receiving truthful information. If manipulated data goes undetected, it is likely that many individuals will make ill-informed decisions.

10. References

1. AlShariah1, N. M. (n.d.). Detecting Fake Images on Social Media using Machine Learning. Retrieved from https://thesai.org/Downloads/Volume10No12/Paper_24-Detecting_Fake_Images_on_Social_Media.pdf

2. Brownlee, J. (2019, July 12). A Tour of Generative Adversarial Network Models. Retrieved from https://machinelearningmastery.com/tour-of-generative-adversarial-network-models/

3. Brownlee, J. (2019, November 21). How to Explore the GAN Latent Space When Generating Faces. Retrieved from https://machinelearningmastery.com/how-to-interpolate-and-perform-vector-arithmetic-with-faces-using-a-generative-adversarial-network/

4. Brownlee, J. (2019, September 11). Tips for Training Stable Generative Adversarial Networks. Retrieved from https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/

5. Common Problems | Generative Adversarial Networks | Google Developers. (n.d.). Retrieved from https://developers.google.com/machine-learning/gan/problems

6. Creswell§, A., Dumoulin, V., & Arulkumaran, K. (2017, October 19). Generative Adversarial Networks: An Overview. Retrieved from https://arxiv.org/pdf/1710.07035.pdf

7. Dinesh. (2019, November 28). CNN vs MLP for Image Classification. Retrieved from https://medium.com/analytics-vidhya/cnn-convolutional-neural-network-8d0a292b4498

8. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Bengio, Y. (2014). Generative Adversarial Nets. Retrieved from https://arxiv.org/pdf/1406.2661.pdf

9. Hui, J. (2019, October 29). GAN - Why it is so hard to train Generative Adversarial Networks! Retrieved from https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

10. Kingma, D., & Ba, J. L. (2017, January 30). ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. Retrieved from https://arxiv.org/pdf/1412.6980.pdf

11. Liu, Z., & Luo, P. (2016, August 7). Large-scale CelebFaces Attributes (CelebA) Dataset. Retrieved from http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

12. Radford, A., & Metz, L. (2016, January 7). UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS. Retrieved from https://arxiv.org/pdf/1511.06434.pdf

13. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016, June 10). Improved Techniques for Training GANs. Retrieved from https://arxiv.org/abs/1606.03498

14. Shen, K. (2018, June 19). Effect of batch size on training dynamics. Retrieved from https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

15. Shibuya, N. (2019, January 1). Up-sampling with Transposed Convolution. Retrieved from https://medium.com/activating-robotic-minds/up-sampling-with-transposed-convolution-9ae4f2df52d0

16. Silva, T. S. (2018, August 11). Advanced GANs - Exploring Normalization Techniques for GAN training: Self-Attention and Spectral Norm. Retrieved from https://sthalles.github.io/advanced_gans/

17. Sudhir, K. (2017, June 22). Generative Adversarial Networks- History and Overview. Retrieved from https://towardsdatascience.com/generative-adversarial-networks-history-and-overview-7effbb713545

18. Tiu, E. (2020, February 4). Understanding Latent Space in Machine Learning. Retrieved from https://towardsdatascience.com/understanding-latent-space-in-machine-learning-de5a7c687d8d

19. Uniqtech. (2019, June 13). Multilayer Perceptron (MLP) vs Convolutional Neural Network in Deep Learning. Retrieved from https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1

20. Vincent, J. (2018, December 17). These faces show how far AI image generation has advanced in just four years. Retrieved from https://www.theverge.com/2018/12/17/18144356/ai-image-generation-fake-faces-people-nvidia-generative-adversarial-networks-gans

21. VGGFACE2. (n.d.). Retrieved from http://www.robots.ox.ac.uk/~vgg/data/vgg_face2/data_infor.html

22. Vijay, R. (2019, November 12). Fake Face Generator Using DCGAN Model. Retrieved from https://towardsdatascience.com/fake-face-generator-using-dcgan-model-ae9322ccfd65

23. Xiang1, S., & Li1, H. (2017, December 4). On the Effects of Batch and Weight Normalization in Generative Adversarial Networks. Retrieved from https://arxiv.org/pdf/1704.03971.pdf