Generate Fake Faces using DCGANs

Yihui Wang

UFID# 8316-4355

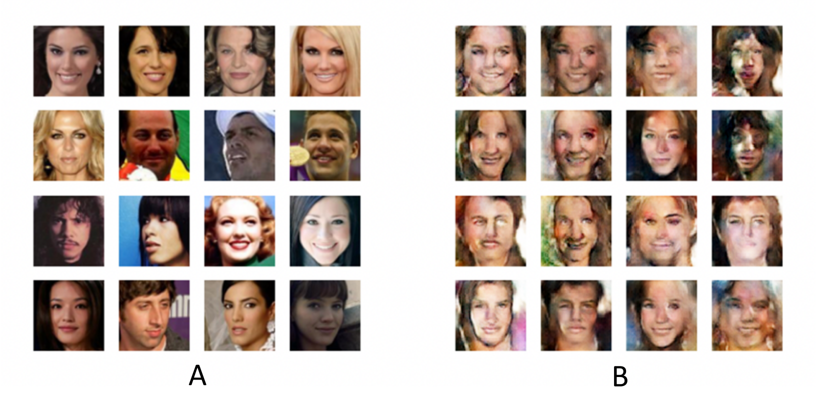


Figure 1: Fake faces generated by DCGANs with the CelebA dataset. A is the example of original faces from the CelebA dataset. B is the fake faces generated by the DCGANs after 300 epochs of training.

Abstract

In recent years, a huge number of researches about generative adversarial networks (GANs) appear. With the rapid development of this amazing technique, the applications of GANs are more and more common in our daily life, such as image inpainting, image transformation etc. In this paper, A Deep Convolutional Generative Adversarial Network (DCGAN) is implemented based on the paper of Radford, et al [1] and trained with the CelebrA datasets [2] to produce fake human faces, and I analyzed and evaluated the DCGANs, showing that by replacing the pooling in the GANs with the convolutional techniques, the DCGANs have great performances on unsupervised learning tasks.

1 Introduction

Generative Adversarial Networks (GANs) is designed by Ian Goodfellow, et al in 2014 [3]. With two neural networks compete with each other, the model can generate new images that have realistic characteristics as the given training dataset. Inspired by the part II of the course project, I was wondering whether I can combine the techniques of the Convolutional Neural Networks (CNNs) and GANs to improve the performance of GANs in certain unsupervised tasks. Then, I found the paper from Radford, et al [1], who designed the Deep Convolutional Generative Adversarial Networks (DCGANs).

In this paper, I implemented the DCGANs described in [1] with the frameworks of TensorFlow, and trained them with CelebrA datasets [2], which contains over 200k images of celebrities' faces. First, I built the basic networks based on the tutorials from the TensorFlow website [4] and tested it with the MINIST datasets [5]. Then, I replaced the datasets with the CelebrA datasets, modified and optimized the networks to make it work properly with the dataset. After that, I trained the model in HiperGator for 300 epochs, which cost more than 13 hours to finish. Finally, I analyzed and evaluated the performances of the DCGANs I implemented.

2 Related Work

Generative Adversarial Networks (GANs) The original Generative Adversarial Networks is first proposed by Ian Goodfellow, et al in 2014 [3]. The core idea of it is based on the "indirect" training through the discriminator, which itself is also being dynamically updated. [6] Based on the work of Ian Goodfellow, there are many different versions of GAN with different applications appear, such as CycleGAN [7] that realized the automatic training of image-to-image translation models without paired examples, StyleGAN [8] which not only generates photorealistic high-quality photos of faces, but also offers control over the style of the generated image at different levels of detail.

Convolutional Neural Networks (CNNs) The idea of convolutional networks is first appeared in Neocognitron [9] by Fukushima in 1980, which was inspired by the visual cortex of mammals. After that, Yann LeCun et al proposed the ConvNets [10] in 1989 that used the backpropagation to train the convolutional networks for zip code recognition. CNNs are commonly used many fields, such as recognition of images and videos, image analysis, natural language processing and so on.

Deep Convolutional Generative Adversarial Network (DCGANs) The DCGANs are proposed by Radford, et al [1]. In DCGANs, the original GANs are scaled up using CNNs architectures. All the pooling layers are replaced with stride convolutions in discriminator and fractional-strided convolutions in generator, and the fully connected hidden layers are removed for deeper architectures. More details of the DCGANs will be discussed in the following sections of this paper.

3 Model Architectures

The DCGAN is an extension of the GAN. It uses convolutional layers in the generator and discriminator, respectively. Figure 2 shows architecture of the generator specifically, and Figure 3 shows the architecture of the whole model. To scale up GANs using CNNs architectures, DCGANs applied the following four main techniques to make the GANs have capability of training deeper generative models with higher resolution.

3.1 Replacing the pooling with convolution

DCGANs replaced all the pooling layers with stride convolutions in discriminator and fractional-strided convolutions in generator. By applying this, the generator is able to learn its own spatial upsampling while the discriminator is able to learn its own downsampling.

3.2 Removing the fully connected layers

The fully connected layers are removed in DCGANs, because although the fully connected layers can make the model more stable, it will slow down the convergence speed of the model. With the fully connected layers removed, the model can have a deeper architecture with faster speed of convergence.

3.3 Batch normalization

Applying the batch normalization in the DCGANs normalizes the input to each unit to have unit variance and zero mean, which can make the learning process more stable. This technique helps deal with the problems caused by poor data initialization and enable the gradient to flow in deeper models. It turns out that it is important to prevent the generator from crashing all samples to a single point. However, applying batch normalization to all layers directly will cause sample oscillation and make the model instable. To avoid this, we can remove batch normalization from the output layer of generator and the input layer of discriminator.

3.4 ReLU in generator and LeakeyReLU in discriminator

The DCGANs use ReLU activation in the generator, except for the output layer which uses Tanh function. It turns out that using bounded activations can make the model learn faster to saturate and cover the color space of the training distribution. In the discriminator, LeakyReLU has great performance for higher resolution modeling, which is different from the original GAN with maxout activation function. Equation 1 is the function of LeakyReLU.

$$f(x) = max(a \times x, x) \tag{1}$$

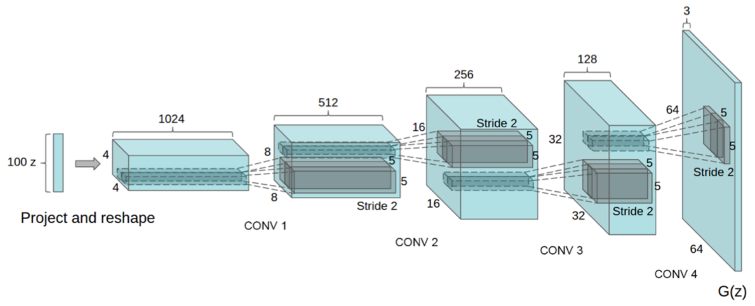


Figure 2 Example of DCGAN generator architecture from [1].

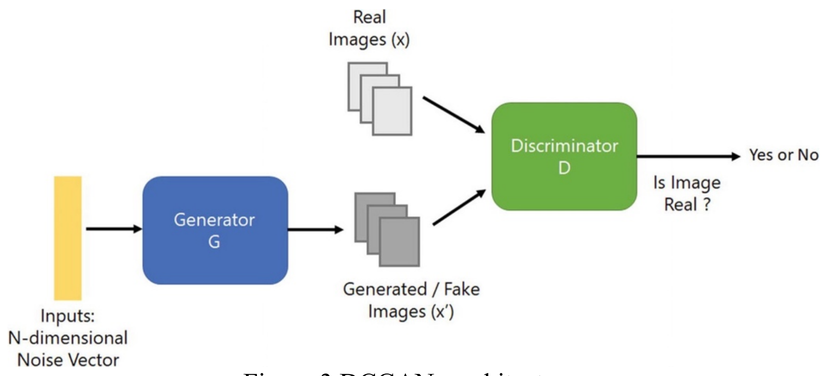


Figure 3 DCGANs architecture.

4 Training and Results

To train the DCGANs and evaluate its performance, I first trained the model with MNIST dataset, which is a simple handwriting digits dataset, to see how the DCGANs work with simple small dataset. After that, I modified the model and trained it again with the CelebA datasets, which is a large-scale human faces dataset with over 200k images of human faces.

4.1 MNIST

I first trained the model with MNIST dataset for 200 epochs, which costs about 9 minutes in HiperGator with one GPU requesting 4GB Memery. Figure 4 is the results of the training using MNIST dataset.

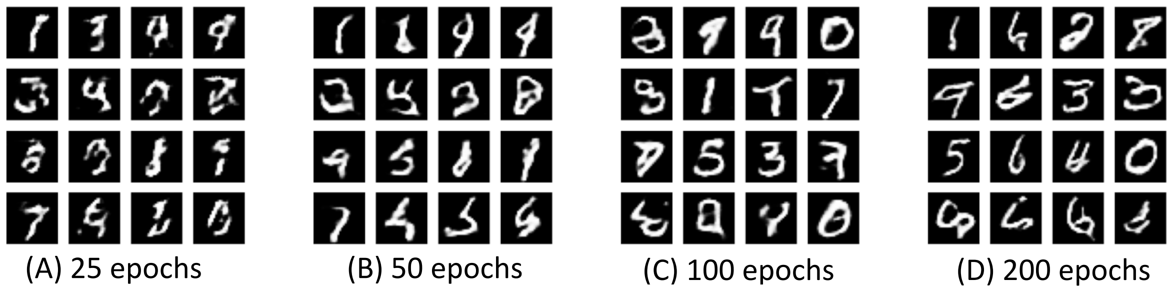


Figure 4 The result of training using MNIST dataset. A is the result after 25 epochs, B is the result after 50 epochs, C is the result after 100 epochs, and D is the result after 200 epochs.

As we can see, the DCGANs perform pretty well on this small simple dataset. The results after 100 epochs are already recognizable, and whole training process is completed very quickly.

4.2 CelebA

After testing the DCGANs with MNIST dataset, I modified and optimized the model to make it work properly with the CelebA dataset. I trained the model in HiperGator for 300 epochs, which cost more than 13 hours to finish. Figure 5 is the results of the training using CelebA dataset.

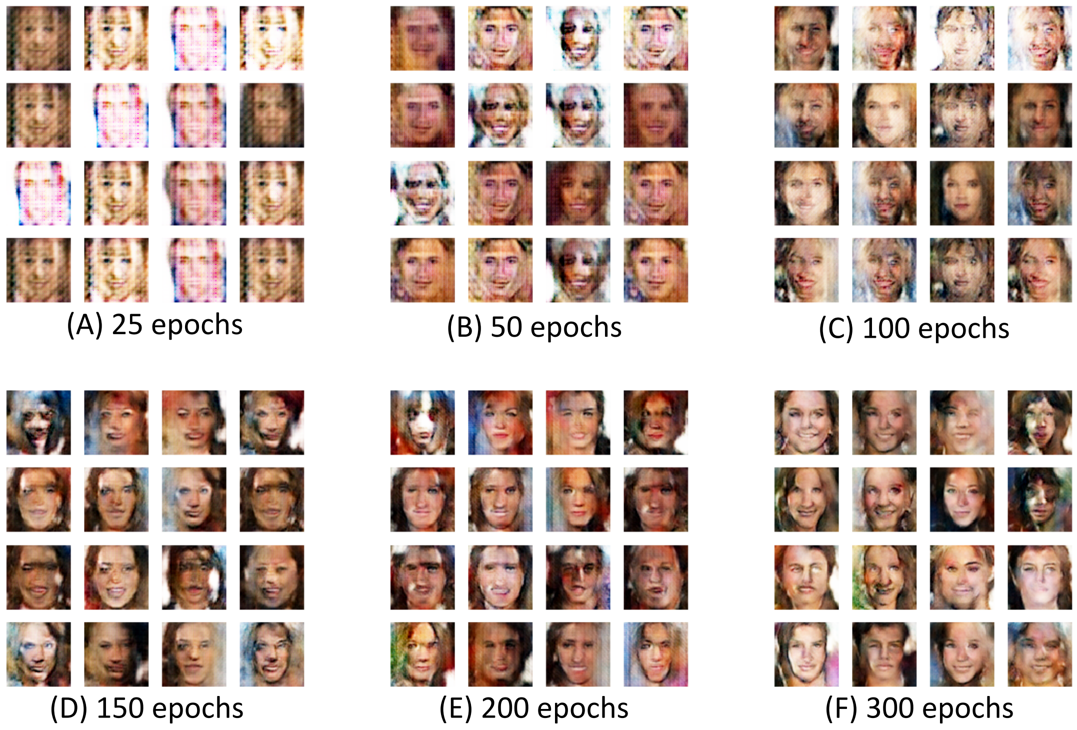


Figure 5 The result of training using CelebA dataset. A is the result after 25 epochs, B is the result after 50 epochs, C is the result after 100 epochs, D is the result after 150 epochs, E is the result after 200 epochs, and F is the result after 300 epochs.

For the results presented above, we can see that the shape of human faces appears after 50 epochs of training, and more details of face become recognizable after 200 epochs, such as teeth, hair and facial expressions.

5 Analysis and Evaluation

In this section, I will analyze and evaluate the model trained with CelebA dataset.
Figure 5 above is chart of the generator loss and the discriminator loss of the training.

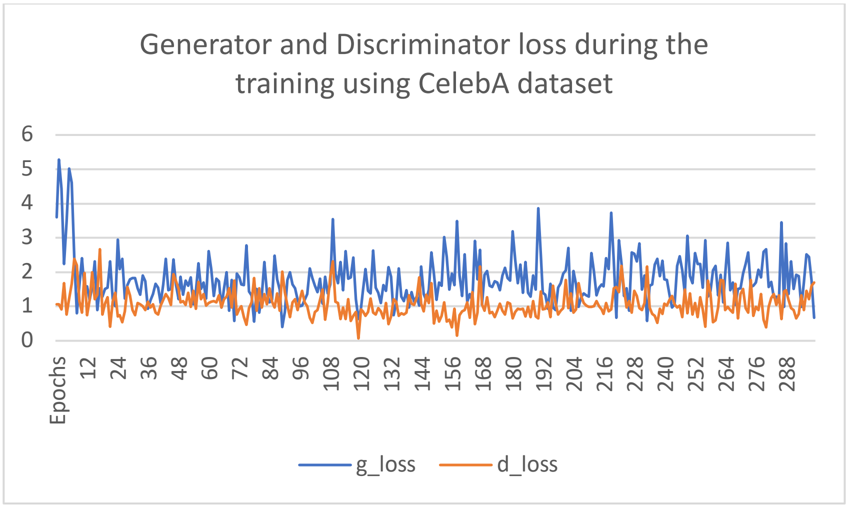
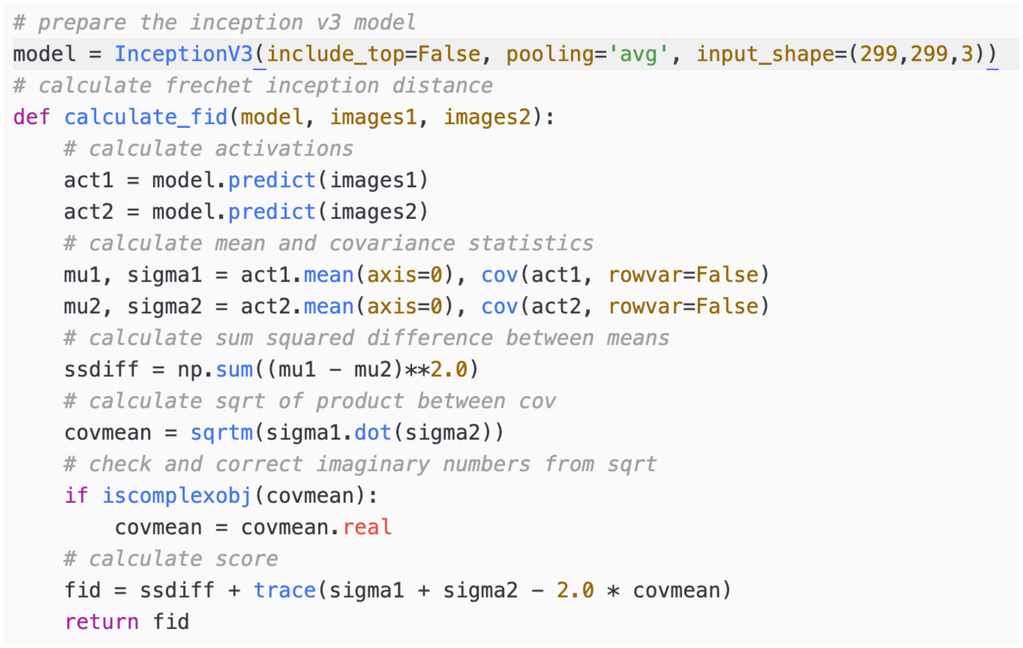


Figure 6 Generator and Discriminator loss during the training using CelebA dataset.

From the Figure 6, we can see that the loss of both the generator and discriminator shows a downward trend. I believe the trend will be more obvious if the model can be trained for more epochs such as 5000 epochs.

To evaluate the DCGANs, I used Frechet Inception Distance (FID) score which is a widely adopted metric for evaluating generated images. The FID score uses the pre-trained inception v3 model to capture the features of input images. These activations are calculated for a collection of real and generated images. After that, the activations are summarized as a multivariate Gaussian by calculating the mean and covariance. These data are then calculated for the activations across the collection of real and generated images. The distance between these two distributions is then calculated using the Frechet distance. A low FID indicates that generated images are of high quality, so the FID should be as low as possible.

To calculate the FID score, I followed the tutorial from Jason Brownlee [12] and implemented the inception v3 model. I used the trained DCGANs to generate 5000 random faces and randomly pick 5000 original faces from the CelebA dataset. And then, I used the following code in Figure 7 to calculate the FID score.



```
# prepare the inception v3 model
model = InceptionV3(include_top=False, pooling='avg', input_shape=(299,299,3))
# calculate frechet inception distance
def calculate_fid(model, images1, images2):
    # calculate activations
    act1 = model.predict(images1)
    act2 = model.predict(images2)
    # calculate mean and covariance statistics
    mu1, sigma1 = act1.mean(axis=0), cov(act1, rowvar=False)
    mu2, sigma2 = act2.mean(axis=0), cov(act2, rowvar=False)
    # calculate sum squared difference between means
    ssdiff = np.sum((mu1 - mu2)**2.0)
    # calculate sqrt of product between cov
    covmean = sqrtm(sigma1.dot(sigma2))
    # check and correct imaginary numbers from sqrt
    if iscomplexobj(covmean):
        covmean = covmean.real
    # calculate score
    fid = ssdiff + trace(sigma1 + sigma2 - 2.0 * covmean)
    return fid
```

Figure 7 Code for calculating the FID score

The table above is the FID score of the DCGANs.

Table 1 the FID score of the DCGANs

Model	Metric	Generated images	Real images	score
DCGANs	FID	5000	5000	228.433

As we can see, the FID score of the DCGANs is little bit higher than I expected, which indicates that the quality of the generated images of my DCGANs are not good enough. The FID score of the normal DCGANs should be around 50. The reason of my high score may be that the model still needs more times and data to be well-trained and my model still have a lot of room for improvement.

6 Conclusion

In this project, I implemented the DCGANs based on the paper of Radford, et al, and trained the model with MNIST and CelebA dataset. After training with the CelebA dataset of 300 epochs for almost 13 hours, my model can successfully generate random faces. At last, I evaluated the model with Frechet Inception Distance. The result of the FID scores indicates that although the model can generate random fake faces, the quality of the generated image still needs to be improved.

In the future, I will do more researches on GANs, and DCGANs specifically, trying to better optimize the current model I have, and explore more applications that GANs may have.

7 How to run the code

To run the code, you need to put the dataset [2] folder in the same path as the python files.
In the folder “face_new_training_checkpoints”, there is a saved model which have been trained for 300 epochs.

1. Open the terminal and go to the path of the python file.
2. To train the model, type “python3 dcgan_face_new.py”
3. To generate random faces, type “python3 randomface.py”
4. To calculate the FID score, type “python3 eval.py”

Reference

[1] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).
[2] CelebFaces Attributes (CelebA) Dataset: https://www.kaggle.com/jessicali9530/celeba-dataset
[3] Goodfellow, Ian, et al. "Generative adversarial networks." Communications of the ACM 63.11 (2020): 139-144.
[4] TensorFlow Tutorials: Deep Convolutional Generative Adversarial Network: https://www.tensorflow.org/tutorials/generative/dcgan
[5] The MNIST database of handwritten digits: http://yann.lecun.com/exdb/mnist/
[6] Luc, Pauline, et al. "Semantic segmentation using adversarial networks." arXiv preprint arXiv:1611.08408 (2016).
[7] Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017.
[8] Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2019.
[9] Fukushima, Kunihiko, and Sei Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition." Competition and cooperation in neural nets. Springer, Berlin, Heidelberg, 1982. 267-285.
[10] LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." Neural computation 1.4 (1989): 541-551.

[11] Salimans, Tim, et al. "Improved techniques for training gans." Advances in neural information processing systems 29 (2016): 2234-2242.
[12] How to Implement the Frechet Inception Distance (FID) for Evaluating GANs: https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/