Celebrity Faces Generation and Optimization Using DCGAN

Lingfeng Zhao & Zhixia Zhang
Tandon School of Enginerring
New York University
{lz1973,zz2445}@nyu.edu

Abstract

This project aims to build a GAN network and to optimize the training process to obtain a better result. This project refers to DCGAN that use deep convolutional GAN and apply different learning rates to generator and discriminator, and we got a better result than the use the original DCGAN.

1 Introduction of DCGAN

Deep Convolutional Generative Adversarial Networks (DCGAN)[2], is one of the most popular and successful network design for GAN. It mainly composes of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the downsampling and the upsampling. It uses batchnorm in both the generator and the discriminator. It uses ReLU activation in generator for all layers except for the output, which uses Tanh. It uses LeakyReLU activation in the discriminator for all layers.

1.1 Structure of DCGAN

The generator of DCGAN is shown in Figure 1. All weights were initialized from a zero-centered Normal distribution with standard deviation 0.02. In the LeakyReLU, the slope of the leak was set to 0.2 in all models.

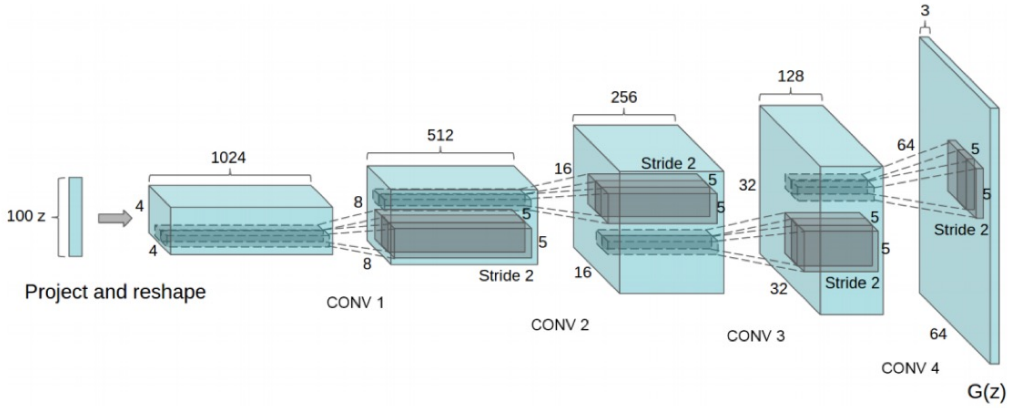

Figure 1: DCGAN generator.

The discriminator uses only convolution layer and replaces deterministic spatial pooling functions (such as maxpooling) with strided convolutions, allowing the network to learn its own spatial downsampling.

2 Dataset

We use Large-scale Celebrity Faces Attributes (CelebA)[3] as our dataset. CelebA is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations. The images in this dataset cover large pose variations and background clutter. CelebA has large diversities, large quantities, and rich annotations. In this project, we choose CelebA as the training set to train the discriminator. Some samples from the dataset is shown in Figure 2.

Figure 2: Sample images from CelebA

3 Our work

3.1 Structure

Our GAN includes two parts, generator and discriminator, as described in [2], we build our network as follows:

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (8): ReLU(inplace=True)
```

```
    (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (13): Tanh()
  )
)

Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1))
    (12): Sigmoid()
  )
)
```

We crop the training images to 64×64 and set the batch size to 128 to fit the discriminator.

3.2 Optimization

Compared to use same learning rate on G and D, we changed our strategy to use different learning rates on G and D. Trying different parameters, we find out the best learning rates for G and D is 0.0002 and 0.00005 respectively. Using different helps the discriminator converge more precise and avoid potential over fitting.

In practice, we run the program for 10 epochs, and finally get the result. We generated some faces, and the final result is shown in Figure 3.

The training losses is shown in Figure 4. And to compare the original approach with fixed learning rate, the result is shown in Figure 5

4 Conclusion

To complete this project, we read serval papers about GAN, and find it can get a better result by using DCGAN. And we trying to apply different approach to improve the result. We tried to change the structure and to use different learning rates.

Shown in the result, DCGAN archives pretty good result in generating new images. And by applying different learning rates, we denotes that our approach improves the quality of generated images.

References

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

[2] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

[3] Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep Learning Face Attributes in the Wild. In Proceedings of International Conference on Computer Vision (ICCV).

Figure 3: Real images and generated images

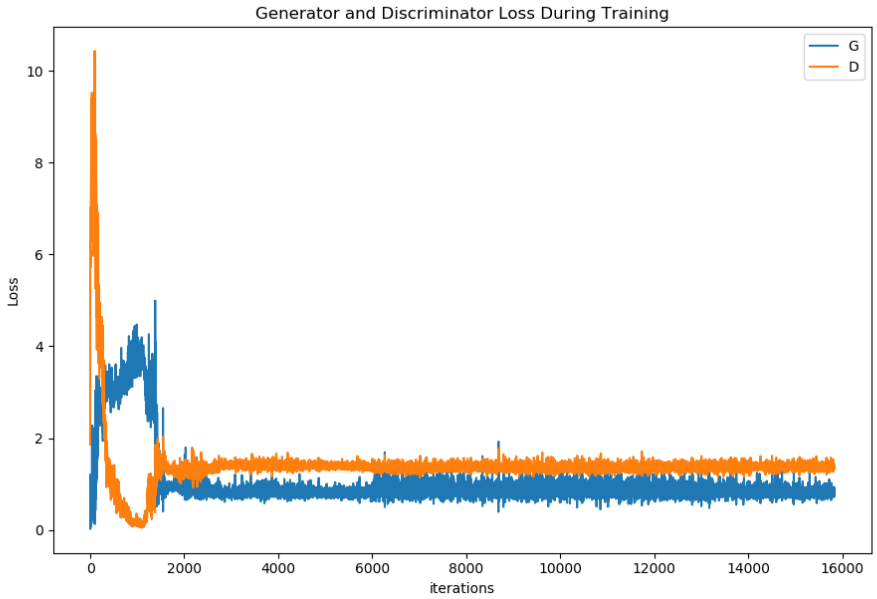

Figure 4: Loss in training process

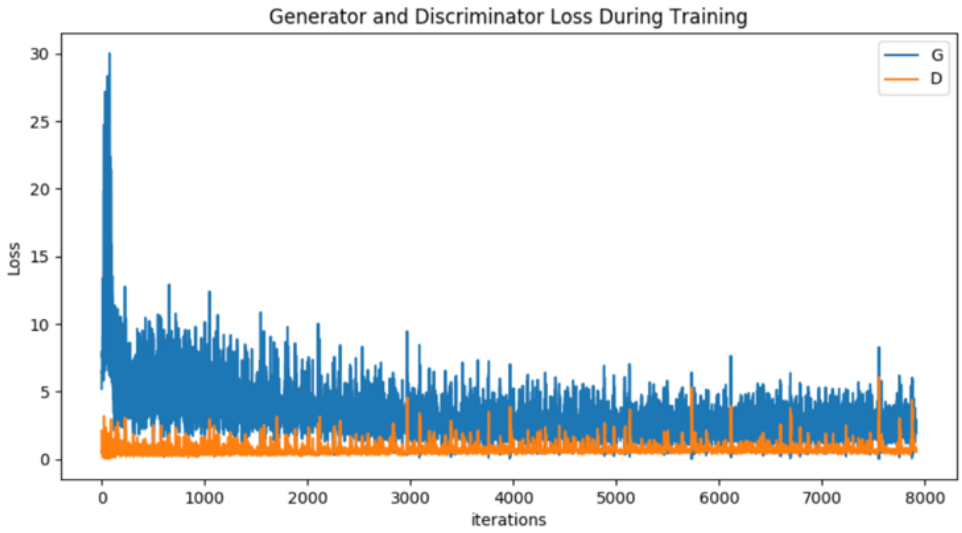

Figure 5: Comparison on different learning rates