ML Project#3 Report

DCGAN Real Face Generation

Author: 資工三 406410035 秦紫頤

Setup

Components

- celebA_data_preprocess.py: Doing data preprocessing (resize the image)
- pytorch_CelebA_DCGAN.py: DCGAN main program
- add_noise.py: add normal and uniform distribution noise
- README.md: README
- REPORT.pdf: this report file

Requirements

- Pytorch 0.4.1
- cuda80
- Python 2.7.6
- torchvision 0.1.8
- matplotlib 1.3.1
- imageio 2.2.0
- scipy 0.19.1

Execution

1. Create a conda environment that can fulfill the requirements above
2. Activate the environment
3. Put data into this project folder
4. First, change the path in the following files
    a. celebA_data_preprocess.py: change the path of `root` and `save_root`
    b. pytorch_CelebA_DCGAN.py: change the path of `data_dir`
5. Execute the preprocess program: `python celebA_data_preprocess.py` -> the resizing image will be saved in “resized_celebA/celebA”
6. Execute the DCGAN main program to get the generate result, weight and loss curve: `python pytorch_CelebA_DCGAN.py` -> after execution the following files will be generated in the folder “CelebA_DCGAN_results”
    a. Fixed_results: this folder will have every epoch (200) generated images
    b. discriminator_param.pkl: this file contains the weights of the trained discriminator
    c. generator_param.pkl: this file contains the weights of the trained generator

    d. generation_animation.gif: gif of all the generated images
7. Before executing add_noise.py first change the path of `MODEL_PATH` and `SAVEPATH` in the file
8. Execute add_noise.py: `python add_noise.py` -> the following files will be generated in the folder “CelebA_DCGAN_results”
    a. normal_0_1.png: generated image with normal distribution noise N(0,1)
    b. Normal_-10_1.png: generated image with normal distribution noise N(-10,1)
    c. Uniform_0_1.png: generated image with uniform distribution noise U(0,1)

Method Description

In this project, we are generating celebrity faces by using DCGAN. DCGAN is one of the 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. The figure below is the network design for the generator.
The detail procedure is as below:

1. Resize the data
2. Train with the resize image
3. For every epoch
    a. Fix generator update discriminator
    b. Fix discriminator update generator
    c. Generate image

Resize the data
The original data is too big the computation time may rise exponentially when training so we resize the image to a smaller size.

Fix generator and update discriminator
It compares the discriminator's predictions on real images (real loss), and the discriminator's predictions on fake (generated) images (fake loss). We will use 1 as real and 0 as fake so intuitively we are using BCELoss as my loss function. The total loss is real loss plus fake loss and updates the weights by Adam optimizer

Fix discriminator and update generator
The generator's loss quantifies how well it was able to trick the discriminator. Intuitively, if the generator is performing well, the discriminator will classify the fake images as real. Here, I will compare the discriminators’ decisions on the generated images. The same as updating discriminator, we use BCELoss as our loss function and Adam as our optimizer.

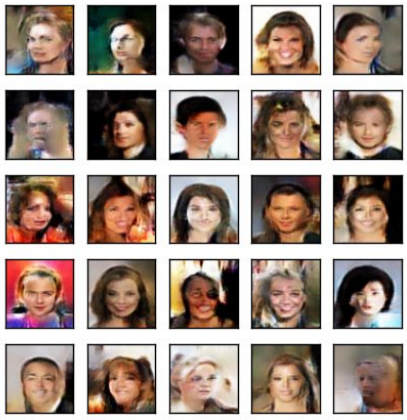
Generate Image
After every epoch, we will generate an image so we can see the difference of all the epochs at the end.

Results

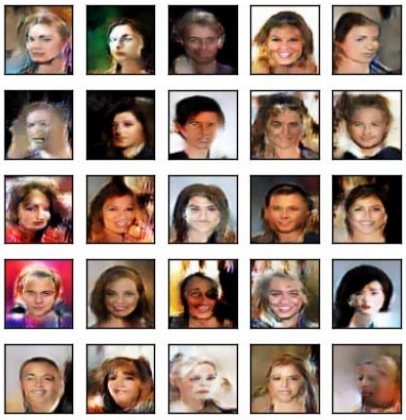
The generated image when epoch= 1, 50, 100, 200

Epoch 1

Epoch 50



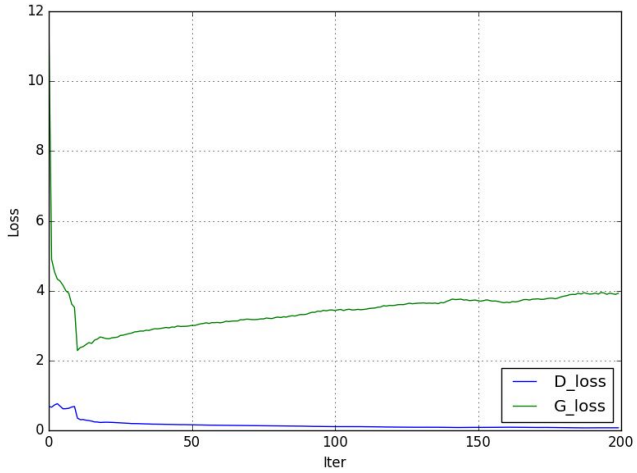
Epoch 100

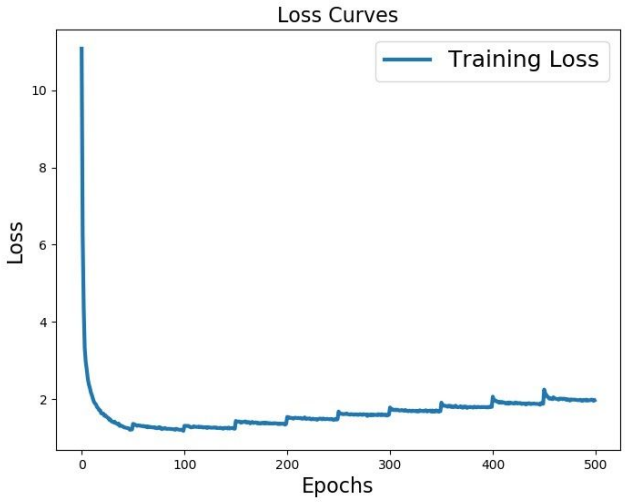


Epoch 200

From the four images above, I can only tell that the first epoch is a bit blurrier than the other 3 one, and the other 3 images just look slightly different. I think the reason is that the discriminator loss comes to 0 really fast and the generator loss didn't, so the discriminator think it is perfect already but generator and discriminator didn't balance eventually. The model didn't converge.

Compare GAN loss curve and normal CNN loss curve

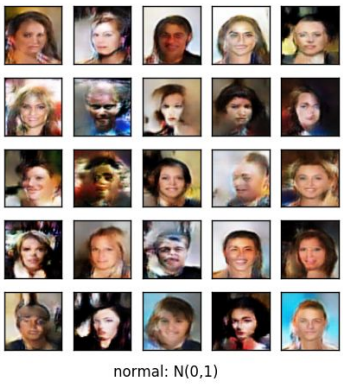


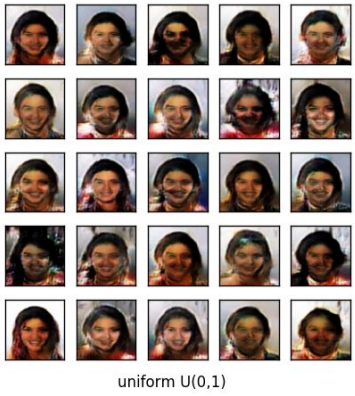


GAN Loss *CNN Loss*

GAN loss is from this project. And CNN loss is from my lane detection project. I think the obvious difference is that GAN loss always has generator loss and discriminator loss and CNN loss mostly only have one training loss. Also, GAN is difficult to train so the loss value can be too big or too small. CNN is easier to train and converge, and loss can decrease really fast. This is just my point of view.

Compare the generated results when adding noise of N(0,1), N(-10,1) and U(0,1)





N(0,1) *N(-10,1)* *U(0,1)*

The best result of the three images is the first one adding normal noise with mean 0 and standard deviation 1. When adding noise with mean -10 and standard deviation 1, I think this destroys all the images because of the strange mean value of the noise. If using uniform noise all the images style look similar in a good way but this is not what we want.

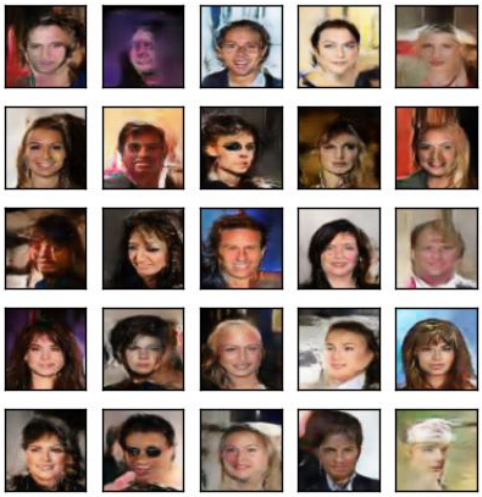
Will the quality change if we add more other datasets

I didn't try other datasets because I think using a dataset instead of celebrities won't be better because my target is to generate celebrity images. Instead, I still use the same dataset, but I use 200,000 images to train. It is 6

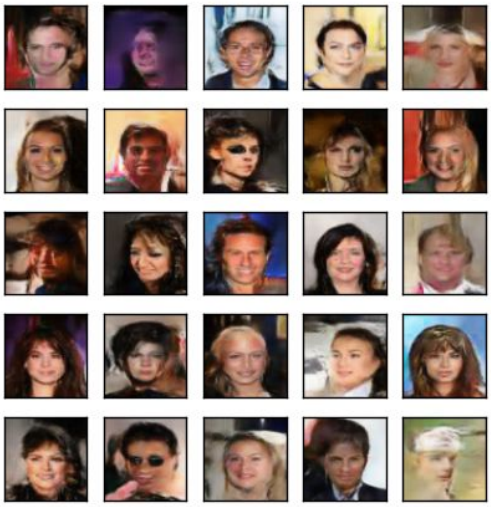
times more data than our original training size (30,000). Because of the large dataset size so I only train 30 epochs instead of 200 epochs. The resulting images are as below:

Epoch 1

Epoch 10

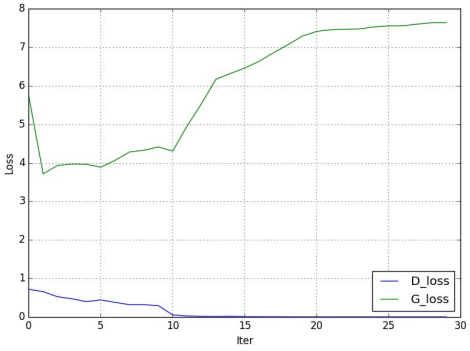


Epoch 20



Epoch 30

The resulting images look a bit better than using 30,000 images as training data. I think more data did improve the quality but the problem is still the same. The discriminator loss decrease too fast so it won't improve after some epochs. The loss curve is as below



Discussion

In the lecture, we have learned some ways to improve GAN. For my case discriminator loss becomes 0 after a few epochs but generator loss didn't converge. I think one of the ways to improve is to update generator loss every epoch but only update discriminator every 5 epoch may improve the generated result. Also, I may add some difficulty to the discriminator so that the discriminator may not be so confident by itself and its loss won't decrease that fast.