Selected Topics in Visual Recognition using Deep Learning
HW2 Celebrity face images generation Report

0856049 吳毓軒

GitHub repository link

Link: https://github.com/yuhsuan1203/VRDL/tree/master/HW2

HW2
- |------ DCGAN.ipynb: all the code of this homework
- |------ DCGAN_with_training_image.ipynb: similar to the above one, except with the generated images in each epoch during training process
- |------ celebA_face.gif: an animated gif using the images saved during training
- |------ haarcascade_frontalface_default.xml: used to detect frontal face
- |------ my_helper.py: include some useful function, such as crop the image to the desired size or save the generated image at the end
- |------ G.png: model architecture of the generator
- |------ D.png: model architecture of the discriminator

Introduction

In this homework, we need to train a deep convolutional generative adversarial network (DCGAN) to generate a celebrity face as real as possible. In this network, there are two main models, which are the generator and the discriminator. A generator will take a random noise as input and output an image. A discriminator will take the real image and the fake image produced by the generator as input and try to classify the generated images as real or fake. Due to the different functions provided by these two models, there are also two different loss functions and optimizers for each model. Each model updates its weights respectively. As training progresses, the generator and the discriminator will get better.

Methodology

- Data Preprocess

  The data preprocessing consists of two parts, which are **Discard the profile face** and **Normalize the data**.

  Discard the profile face

  Due to some profile faces in the real image dataset, as the two figures shown below, which will influence the realness and quality of the generated faces, I decided to discard them by using the detecting technique provided by opencv. About 60,000 images are discarded after finishing this process.


If we don't apply this preprocessing technique, the result image will be more abnormal, such as the top middle image in the above rightmost image.

However, there might be a tradeoff between the diversity of the training set after applying this work and the completeness of the face produced by the generator. So the threshold used in discarding images is also important when doing this technique.

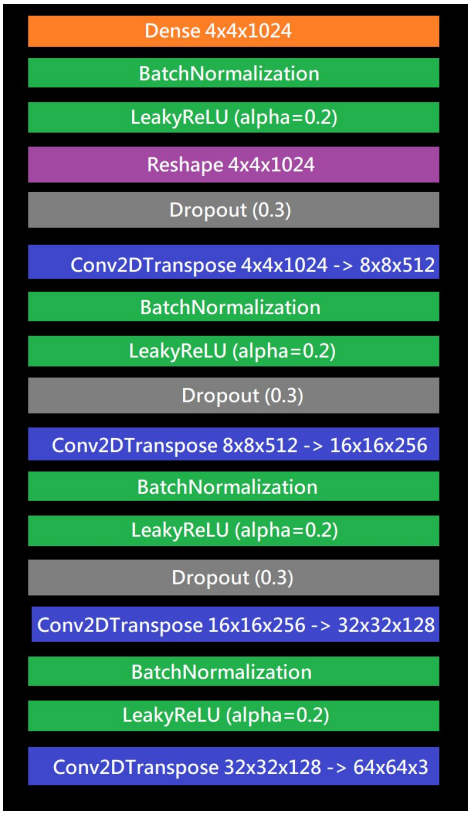
Normalize the data

In this preprocessing stage, all the training images will be normalized to the value between -1 and 1.

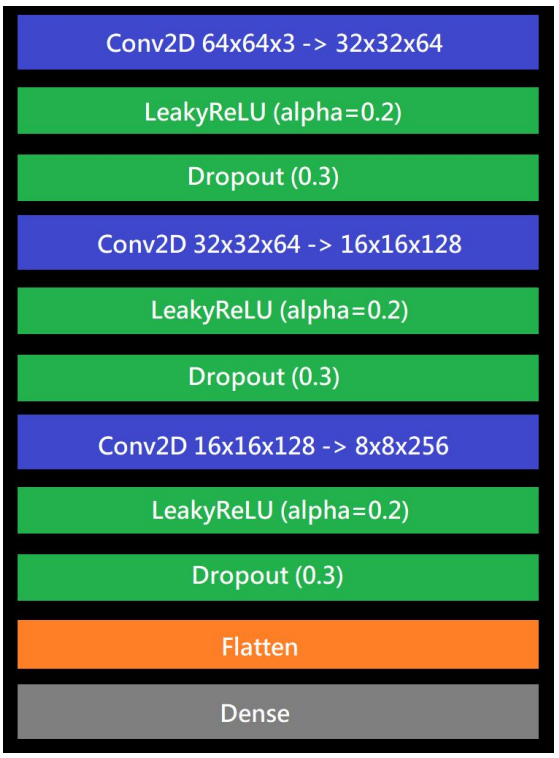
- Model Architecture

The model is based on the DCGAN model provided by tensorflow tutorial in this website. I have also applied some guidelines mentioned in the DCGAN paper. Below are the details of architecture of these two models.

Generator model architecture



Discriminator model architecture



BinaryCrossentropy function is used to calculate loss for each model with label smoothing, and the weights of the generator and discriminator are updated by using the gradients. The optimizer applied in this work is Adam optimizer with learning rate and beta_1 set to the appropriate value.

- Hyperparameters

  There are a set of parameters to be tuned in this work, such as epochs, batch size, dropout rate in generator or discriminator, and so on. After doing a lot of experiments, I decided to use the following value for each parameter.

  ★ epochs: 100
  ★ batch size: 256
  ★ dropout rate: Generator: 0.3, Discriminator: 0.3
  ★ Optimizer: Adam with learning rate=0.0002, beta_1=0.5

Summary & Findings

It is quite difficult to train a stable generative adversarial network. Many hyperparameters, for example, the dropout rate in the generator or discriminator, need to be fine-tuned in order to get a better result. Instead of using the recommended value 0.5 in the ganhacks document, I set the dropout rate to 0.3 in the generator. Without dropout layers in the generator, the generated images will look like the paintings as the figure shown below.

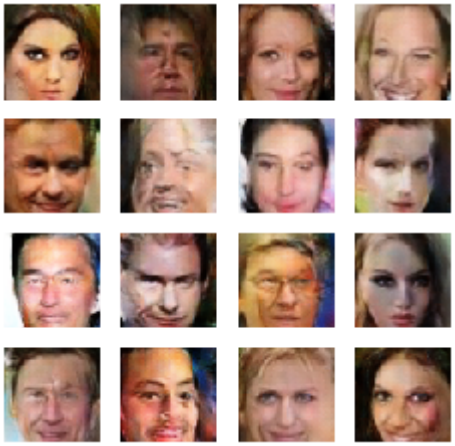


Fig. the image saved in epoch 60 during training with no dropout layer in generator

The DCGAN paper mentioned that they used ReLU activation in each layer except the output layer of the generator. But in my work, LeakyReLU activation is applied in both models after evaluating the performance of these two methods.

I have tried adding the dropout layers between BN and LeakyReLU or after LeakyReLU, but the result is bad. Only adding the layers before Conv2DTranspose layer can give us a better result. In my model, all weights were initialized with mean 0.0 and standard deviation 0.02. Besides, the BatchNormalization is called with epsilon set to 0.00005. I have not tried removing these two parameters in this work.

As the result images still look not so good, I think that there is a lot to improve in my work. Hyperparameters such as dropout rate in the generator or the batch size should be tuned properly in order to generate a more real result.