VRDL_HW2 Report

0856054 莊紹平

Code

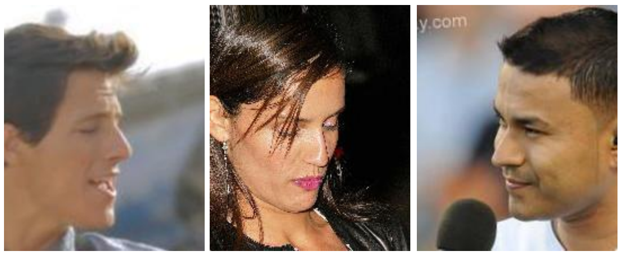
GitHub Link: https://github.com/Jhuangsp/VRDL_HW2 (https://github.com/Jhuangsp/VRDL_HW2)

- `main.py` : Main file contains training loop and log infomation.
- `dataloader.py` : Load data from TFRecord and form tf.data.Dataset.
- `dcgan.py` : Modified DCGAN model.
- `solver.py` : Define loss functions and optimizers.
- `inference.py` : Inference the model and make 500 samples.
- `helper.py` : Provided by TA.
- `pick_front.py` : Select front-face from dataset.
- `haarcascade_frontalface_default.xml` : Haarcascade frontalface model
- `front_face.txt` : Results select by `pick_front.py`
- `process_data.py` : Read original dataset and biuld TFRecord.

Brief introduction

In this assignment, we have to build a Generative Adversarial Network(GAN) to generate human faces by random number vectors.
In GAN model, it contains one generator and one discriminator. The generator digests the random number vector then generates a unique human face, while the discriminator takes the fake image generated by the generator then discriminates it with the real image provided by CelebA dataset.
The difficulty of this task is to keep the balance between generator and discriminator, we except that the generator and discriminator can improve simultaneously.

Methodology

- **Data pre-process**
  - Pick front-face
    In order to constrain the model stability, I only pick front-face samples as my training samples. In this way, I picked about 170,000 front-face images from original 200,000 images dataset.
    I used Haar cascade front face classifier to remove side face samples. i.e.
    
  - Digest large dataset
    170,000 images should be used in this training, but it is impossible that my computer can load such large data. So, I transform the dataset into TFRecord data format, although it might slow down the training speed a little bit.
- **Data augmentation**
  - Random horizontal flip
    In this task, I think that changing the image pose (shift, rotation) or image appearance (color, brightness) will lead to unreal results and hard to constrain. So, I only apply the random horizontal flip.
- **Model architecture**
  - Modify DCGAN
    In this task, I modify some parts of DCGAN:

- Add dropout layer (drop_rate=0.3) befor every Conv2DTranspose in Generator

```
model.add(layers.Dropout(0.3)) # like this
model.add(layers.Conv2DTranspose(
    128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
assert model.output_shape == (None, img_size//8, img_size//8, 128)
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU(0.2))
```

- Add one more Conv2DTranspose to Generator

```
...
model.add(layers.Dropout(0.3))
model.add(layers.Conv2DTranspose(
    64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
assert model.output_shape == (None, img_size//4, img_size//4, 64)
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU(0.2))

# ======================== Added ======================== #
model.add(layers.Dropout(0.3))
model.add(layers.Conv2DTranspose(
    32, (5, 5), strides=(2, 2), padding='same', use_bias=False))
assert model.output_shape == (None, img_size//2, img_size//2, 32)
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU(0.2))
# ======================================================= #

model.add(layers.Conv2DTranspose(
    3, (5, 5), strides=(2, 2),
    padding='same',
    use_bias=False, activation='tanh'))
...
```

- Add one more Conv2D to Discriminator

```
...
model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
model.add(layers.LeakyReLU(0.2))
model.add(layers.Dropout(0.3))

# ======================== Added ======================== #
model.add(layers.Conv2D(256, (5, 5), strides=(2, 2), padding='same'))
model.add(layers.LeakyReLU(0.2))
model.add(layers.Dropout(0.3))
# ======================================================= #

model.add(layers.Flatten())
model.add(layers.Dense(1))
...
```

- Smooth loss
  Add smooth rate 0.9 to Generator's cross_entropy
  Add smooth rate 0.9 to Discriminator's real_loss cross_entropy

```
# Generator loss
def generator_loss(self, fake_output):
    return self.cross_entropy(
        tf.ones_like(fake_output)*self.smooth, fake_output)

# Discriminator loss
def discriminator_loss(self, real_output, fake_output):
    real_loss = self.cross_entropy(
        tf.ones_like(real_output)*self.smooth, real_output)
    fake_loss = self.cross_entropy(
        tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

- Train by turns
  To keep the generator and discriminator improve simultaneously. I only let the loss larger one optimizes the weight in each step.

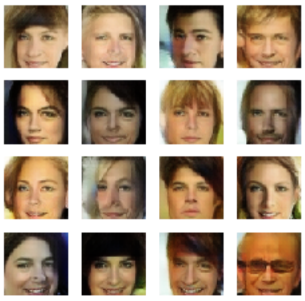
```
tf.cond(gen_loss > disc_loss,
        lambda: self.generator_optimizer.apply_gradients(
            zip(gradients_of_generator,
                self.generator.trainable_variables)),
        lambda: self.discriminator_optimizer.apply_gradients(
            zip(gradients_of_discriminator,
                self.discriminator.trainable_variables)))
```

- **Hyperparameters**
  - Epoch: 200 (but I stop at 126)
  - Batch size: 256
  - Optimizer: both Adam (beta1=0.5)
  - Learning rate: both 0.0002

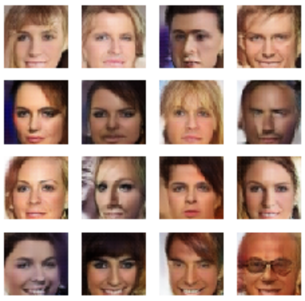
Summary

The valid data look better at epoch 77 than epoch 126, but at inference time epoch 126 performs better than epoch 77.

Valid epoch 77 vs 126:

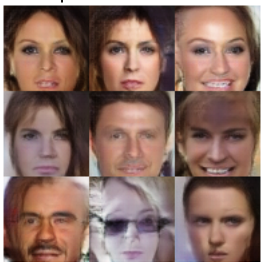


vs

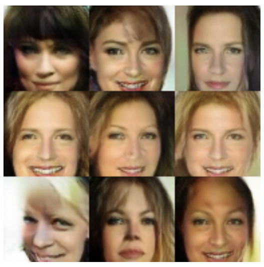


I think the 77's valid data is more smooth than 126's valid data. But for inference data 77 has faint effect due to its smoothness. On the contrary, 126 doesn't have this problem.

Test epoch 77 vs 126:



vs



Observation

According to DCGAN paper, using Adam with beta1=0.5 will help the model constrain.
High dropout rate will improve the diversity of faces but sometime distort the image.

Future work

According DCGAN paper, try relu on generator, leakrelu on discriminator.