

1. Model description (2%) :

Generator:

主結構 : (all kernel_initializer='lecun_normal')

Concatenate(noise_input, text_input)

Dense(65536)

Activation(ReLu)

Reshape(16 * 16 * 256)

Conv2DTranspose(filter=64, [3, 3], stride=[2, 2], padding='same')

Activation(ReLu)

Conv2DTranspose(filter=32, [5, 5], stride=[2, 2], padding='same')

Activation(ReLu)

Conv2DTranspose(filter=16, [5, 5], stride=[1, 1], padding='same')

Activation(ReLu)

Conv2DTranspose(filter= 3, [1, 1], stride=[1, 1], padding='same')

Activation(Sigmoid)

輸入 :

noise_input: dim = 40

text_input: dim = 24

輸出 :

dim = 64 * 64 * 3

Discriminator:

主結構 : (all kernel_initializer='lecun_normal')

Conv2D(filter=64, [5, 5], stride=[2, 2], padding='same')

Activation(ReLu) + AlphaDropout(0.2)

Conv2D(filter=128, [5, 5], stride=[2, 2], padding='same')

Activation(ReLu) + AlphaDropout(0.2)

Conv2D(filter=256, [3, 3], stride=[2, 2], padding='same')

Activation(ReLu) + AlphaDropout(0.2)

Conv2D(filter=256, [3, 3], stride=[2, 2], padding='same')

Activation(ReLu) + AlphaDropout(0.2)

Concatenate repeated text_input at the last dimension of the 4*4*256 net.

Conv2D(filter=512, [4, 4], stride=[1, 1], padding='valid')

Activation(ReLu) + AlphaDropout(0.2) + Flatten

Dense(1, activation='linear')

輸入 :

img_input: dim = 64 * 64 * 3

text_input: dim = 24

輸出 :

dim = 1

Objective function:

```
loss_true_case = mean(D([true_img, true_txt]))
loss_fake_case = mean(D([fake_img, true_txt]))
loss_wrong_txt = mean(D([true_img, wrong_txt]))
mixed_img = mix_factor * true_img + (1. - mix_factor) * fake_img
mixed_grad = gradients(D([mixed_img, true_txt]), w.r.t: [mixed_img, true_txt])
norm_mixed_grad = sqrt(sum(square(mixed_grad[0]), axis=[1, 2, 3]) +
                        sum(square(mixed_grad[1]), axis=[1]))
grad_penalty = mean(square(norm_mixed_grad - 1.))
obj_d = loss_true_case - 0.5 * loss_fake_case - 0.5 * loss_wrong_txt -
penalty_factor * grad_penalty
optimizers_d: Adam(lr=2e-4, beta_1=0., beta_2=0.9, loss= - obj_d)
```

obj_g = loss_fake_case

optimizers_d: Adam(lr=2e-4, beta_1=0., beta_2=0.9, loss= - obj_g)

2. How do you improve your performance (2%)

- wgan_GP: 相較於basic gan更快學習到臉型特徵與髮色、眼睛色等特徵，且少輸出壞掉的機率。
- kernel_initializer='lecun_normal': 使用正確分佈的初始化值可以加速模型進入狀況、減少發散的機會。
- 使用Dense layer作為generator的初始結構，可以減少輸出圖片有方塊狀的格紋，讓整個構圖較連續自然。
- 圖片預處理：將color channel除以255以將數據分佈縮至[0, 1]。
。
- text預處理：將12種髮色與11種眼睛色分別以onehot encode至24維vector，再將同時符合的vector “OR” 在一起。

3. Experiment settings and observation (2%)

使用Dense layer作為generator的初始結構，可以減少輸出圖片有方塊狀的格紋，讓整個構圖較平順自然。

a. 使用Conv2D(padding='valid')作為初始結構：



Epoch 30

可見許多格狀干擾紋，較不自然，但較早學習到部份顏色分類。

b. 使Dense作為初始結構：



Epoch 30

可見一開始圖形就相當完整連續且多樣化，但晚一些學到顏色類。

4. style-transfer (2%)

實作Cycle GAN

(Result @ 47 epoch)

dataset_A = Anime Dataset



dataset_A = celebA Dataset



A->B



B->A



A->B->A



B->A->B



Loss plot、架構：請參閱cycle_gan_2_celebA資料夾。