

Model Description: Describe the models you use to, including the model architecture, objective function for G and D.

1. Image Generation (2%)

a. 模型架構 (DCGAN)

i. Discriminator

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

按：圖片經過Deep Convolution過程逐漸增加channel數，並減少維度，其中使用BN及LeakyRelu，經過(11)後的維度將只會剩下(batchx1x1x1)，最後再通過Sigmoid activation function (寫在forward裡，所以沒顯示) 給出評分。

ii. Generator

```
Generator(  
  (main): Sequential(  
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)  
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)  
    (2): ReLU(inplace)  
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)  
    (5): ReLU(inplace)  
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)  
    (8): ReLU(inplace)  
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)  
    (11): ReLU(inplace)  
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (13): Tanh()  
  )  
)
```

按：noise為100維，在通過Deep Convolution的過程中做Upsampling，並且逐漸減少channel數，其中使用了BN及ReLu，通過(13)後的維度為(batchx3x64x64)，最後再通過Tanh得到image。

b. 目標函數

i. Criterion : Binary Cross Entropy(BCELoss)

$$l_n = -w_n [y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

給定True label時，Loss簡化為 $\log(x_n)$ 。給定False label時，Loss簡化為 $\log(1-x_n)$ 。

ii. Optimizer : D跟G皆為Adam(lr = 2e-5, betas = (0.5, 0.999))。

iii. D與G訓練的比例為1:1。

iv. 實作法：

tlabel = 1 , flabel = 0。

| | |
|---------|---------|
| D train | G train |
|---------|---------|

```

netD.zero_grad()
# Train w/ (true img, true label)
output = netD(real_input)
errD_real = criterion(output, tlabel)
errD_real.backward()
D_x = output.mean()
# Train w/ (fake img, fake label)
fake = netG(noise)
output = netD(fake.detach())
errD_fake = criterion(output, flabel)
errD_fake.backward()
D_G_z1 = output.mean()
errD = errD_real + errD_fake
optimizerD.step()

```

```

netG.zero_grad()
output = netD(fake)
errG = criterion(output, tlabel)
errG.backward()
D_G_z2 = output.mean()
optimizerG.step()

```

2. Text-to-image Generation (2%)

a. 前處理：Condition labeling

```

def _one_hot_labeling(self):
    i = 0
    label_dict = {}
    for label in self.label_list[:]:
        # print(label)
        hair_color = label.strip().split(' ')[0]
        eye_color = label.strip().split(' ')[2]
        if (hair_color, eye_color) not in label_dict:
            label_dict[(hair_color, eye_color)] = i
            i += 1
    n_comb = len(label_dict)
    with open('tag2onehot.pkl', 'wb') as f:
        pickle.dump(label_dict, f, pickle.HIGHEST_PROTOCOL)
    self.one_hot_label = []
    for label in self.label_list[:]:
        hair_color = label.strip().split(' ')[0]
        eye_color = label.strip().split(' ')[2]
        index = label_dict[(hair_color, eye_color)]
        self.one_hot_label.append([0] * (index) + [1] + [0] * (n_comb-index-1))
    self.one_hot_label = torch.FloatTensor(self.one_hot_label)

```

按：使用的是extra_faces，只要找到不同的顏色組合就會列入。除了將dictionary儲存供inference使用，也將各個condition label做one hot encoding。

b. 模型架構(Conditional DCGAN)

i. Discriminator

```

Discriminator(
  (embed): Linear(in_features=119, out_features=119, bias=True)
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(0.2, inplace)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (4): LeakyReLU(0.2, inplace)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (7): LeakyReLU(0.2, inplace)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (10): LeakyReLU(0.2, inplace)
  )
  (c_cv): ConvTranspose2d(119, 119, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (main2): Sequential(
    (0): Conv2d(631, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (2): LeakyReLU(0.2, inplace)
  )
  (dense): Linear(in_features=512, out_features=1, bias=True)
)

```

Forward流程：

1. Condition：使119維的one-hot encoded vector通過"embed"後，經過leaky_relu，unsqueeze成(batchx119x1x1)，再通過"c_cv"後upsampling成(batchx119x4x4)，最後再經過leaky_relu。
2. Image：將image通過"main"（其中用到leaky_relu及BN）後成(batchx512x4x4)。

3. Concatenate : 將(1)及(2)在第二個維度疊起來後(512+119) , 通過"main2" , 最後再通過"dense"在第二維把維度降至1做輸出。

ii. Generator

```
Generator(
  (embed): Linear(in_features=119, out_features=119, bias=True)
  (main): Sequential(
    (0): ConvTranspose2d(219, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (5): ReLU(inplace)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (8): ReLU(inplace)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (11): ReLU(inplace)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Forward流程 :

1. Condition : 使119維的one-hot encoded vector通過"embed"後 , 經過relu , unsqueeze成(batchx119x1x1)。
2. Concatenate : 將100維的noise及(1)在第二個維度疊起來(100+119) , 通過"main" (其中使用relu及BN) 做輸出。

c. 目標函數

- i. Criterion : Binary Cross Entropy(BCELoss)

$$l_n = -w_n [y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

- ii. Optimizer : D跟G皆為Adam(lr = 2e-5, betas = (0.5, 0.999))。

- iii. D與G訓練的比例為1:1。

- iv. 實作法

tlabel = 1 , flabel = 0 , rand_c為隨機選擇之condition之one hot label。

| D train | G train |
|---|---|
| <pre>netD.zero_grad() # Train w/ (true img, true label) output = netD(real_input, real_c) errD_real = criterion(output, tlabel) errD_real.backward() D_x = output.mean() # Train w/ (fake img, fake label) fake = netG(noise, rand_c) output = netD(fake.detach(), rand_c) errD_fake = criterion(output, flabel) errD_fake.backward() D_G_z1 = output.mean() errD = errD_real + errD_fake optimizerD.step()</pre> | <pre>netG.zero_grad() output = netD(fake, rand_c) errG = criterion(output, tlabel) errG.backward() D_G_z2 = output.mean() optimizerG.step()</pre> |

Experiment settings and observation: Show generated images

1. Image Generation (1%)

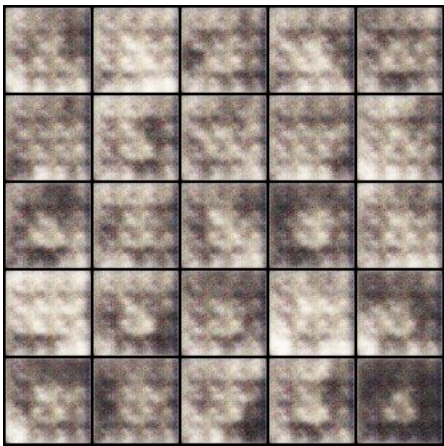

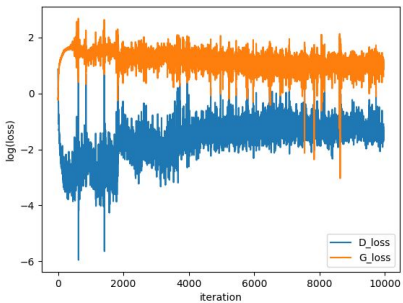
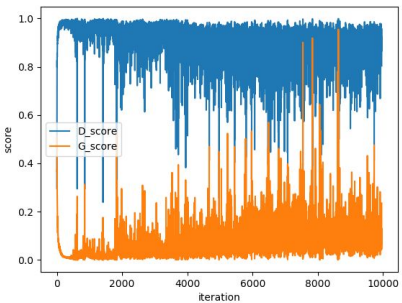
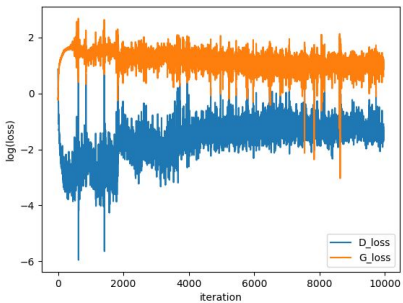
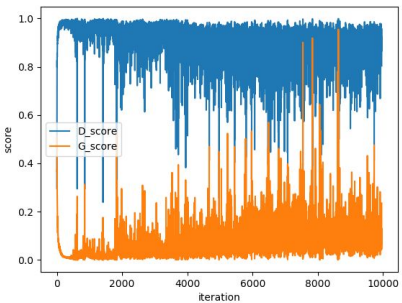
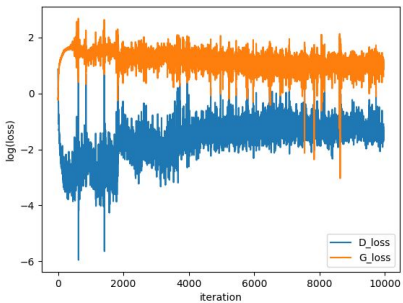
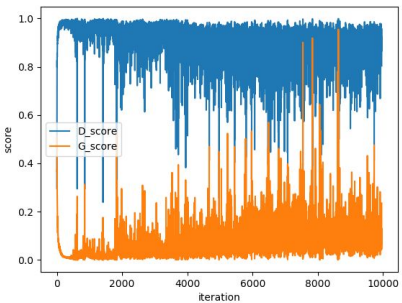
a. 環境

- i. Optimizer: D跟G皆為Adam。
- ii. D與G訓練的比例為1:1。
- iii. 使用extra_faces資料集 , batch size = 64 , 訓練19 epoch , 約10000 iteration。

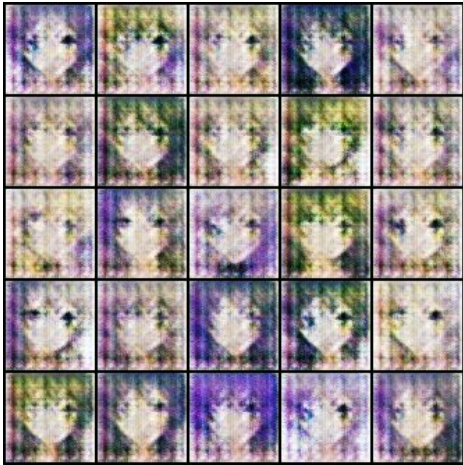

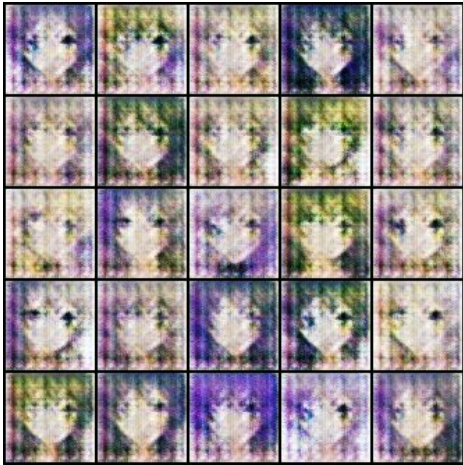

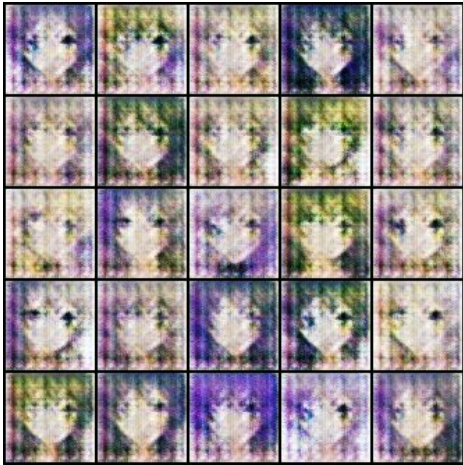

b. 結果

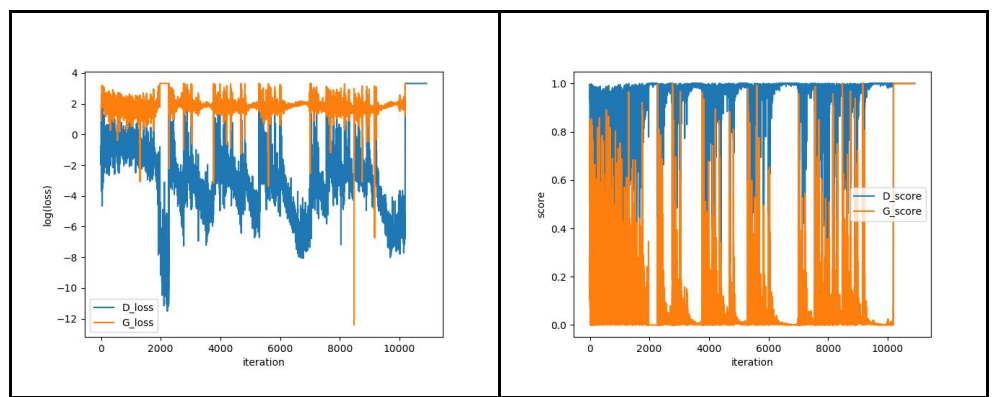
- i. DCGAN(lr = 2e-5)

| 生成圖 | | |
|-----|-----------|------------|
| | Epoch = 1 | Epoch = 19 |

| |   | | | | |
|--|---|------|-------|--|--|
| Data | <table> <tr> <th>Loss</th><th>Score</th></tr> <tr> <td>  </td><td>  </td></tr> </table> | Loss | Score |  |  |
| Loss | Score | | | | |
|  |  | | | | |

ii. DCGAN($\text{lr} = 2\text{e-}5$)

| 生成圖 | <table> <tr> <th>Epoch = 1</th><th>Epoch = 19 (crashed)</th></tr> <tr> <td>  </td><td>  </td></tr> </table> | Epoch = 1 | Epoch = 19 (crashed) |  |  |
|---|---|-----------|----------------------|---|---|
| Epoch = 1 | Epoch = 19 (crashed) | | | | |
|  |  | | | | |
| Data | <table> <tr> <th>Loss</th><th>Score</th></tr> <tr> <td></td><td></td></tr> </table> | Loss | Score | | |
| Loss | Score | | | | |
| | | | | | |



iii. Comment

1. 查到的資料都顯示當optimizer的learning rate很重要，不可以過高，因此就試做了這個實驗。生成圖顯示當adam_lr = 2e-5時可以達到較好的效果，因此在做後續的實驗時都是用這個learning rate。
2. 觀察曲線，可見在lr較大時，數值變化較為極端：D的loss可降到極低，而且會有幾段時間G的loss看似沒有任何變化，導致生成圖一直沒辦法很穩定的變好。

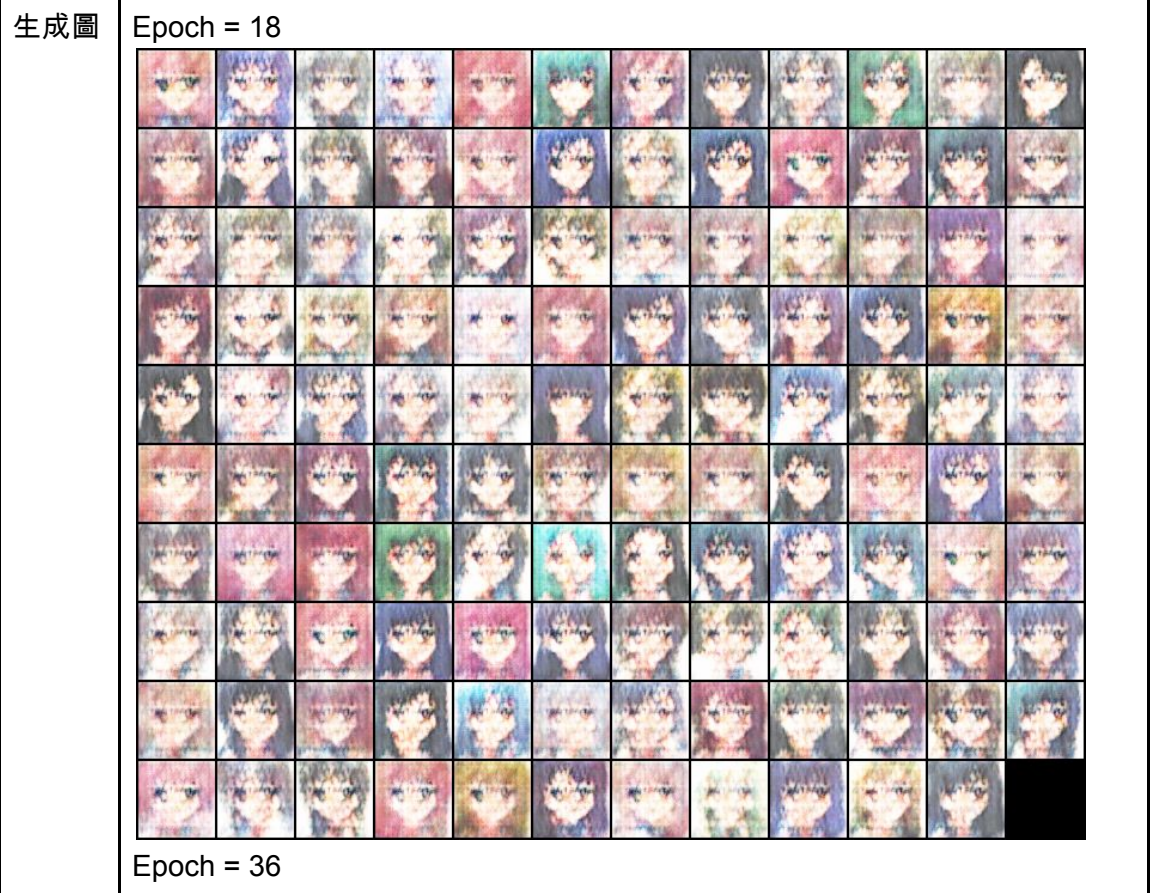
2. Text-to-image Generation (1%)

a. 環境

- i. Optimizer: D跟G皆為Adamprop。
- ii. D與G訓練的比例為1:1。
- iii. 使用extra_faces資料集，batch size = 64，訓練了約20000iteration。

b. 結果

- i. Conditional DCGAN(lr = 2e-5)



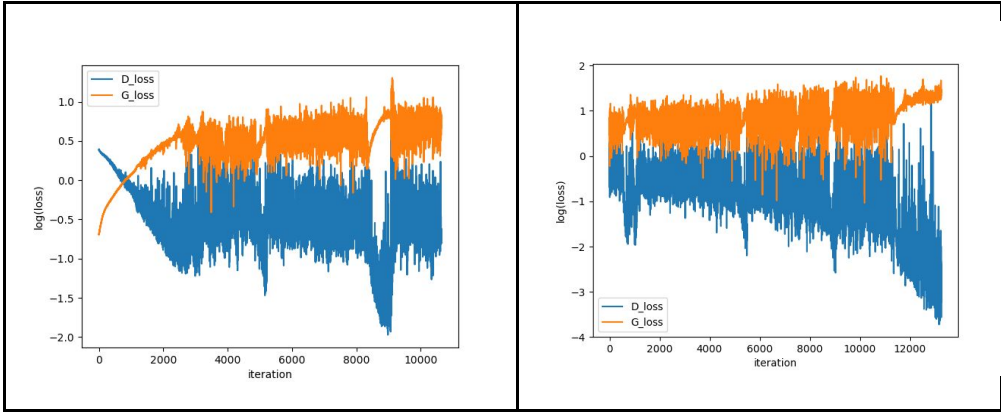
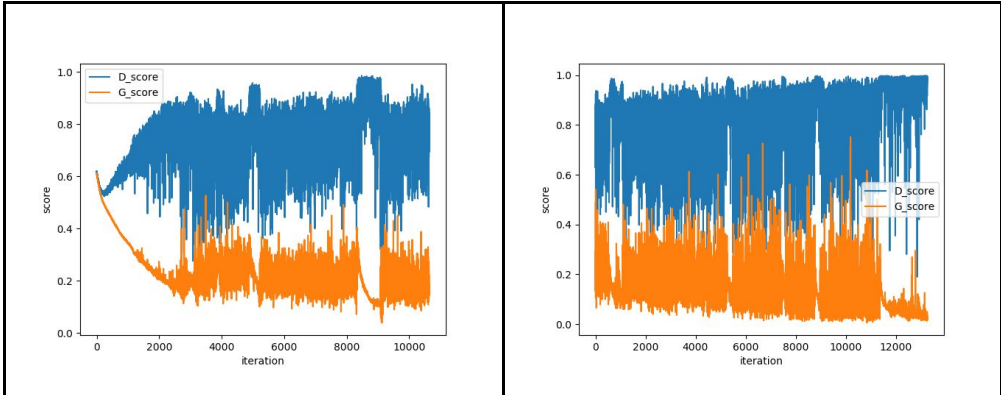


Tags(上圖的排列順序是由左而右，從上而下排列)

| | | | |
|----------------------|----------------------|----------------------|-----------------------|
| 0 : (aqua,aqua) | 36 : (orange,pink) | 72 : (blonde,brown) | 108 : (purple,aqua) |
| 1 : (aqua,black) | 37 : (orange,purple) | 73 : (blonde,green) | 109 : (purple,blue) |
| 2 : (aqua,blue) | 38 : (orange,red) | 74 : (blonde,orange) | 110 : (purple,brown) |
| 3 : (aqua,brown) | 39 : (orange,yellow) | 75 : (blonde,pink) | 111 : (purple,green) |
| 4 : (aqua,green) | 40 : (red,aqua) | 76 : (blonde,purple) | 112 : (purple,orange) |
| 5 : (aqua,orange) | 41 : (red,black) | 77 : (blonde,red) | 113 : (purple,pink) |
| 6 : (aqua,pink) | 42 : (red,blue) | 78 : (blonde,yellow) | 114 : (purple,red) |
| 7 : (aqua,purple) | 43 : (red,brown) | 79 : (blue,aqua) | 115 : (purple,yellow) |
| 8 : (aqua,red) | 44 : (red,green) | 80 : (blue,blue) | 116 : (black,black) |
| 9 : (aqua,yellow) | 45 : (red,orange) | 81 : (blue,brown) | 117 : (blue,black) |
| 10 : (gray,aqua) | 46 : (red,pink) | 82 : (blue,green) | 118 : (purple,purple) |
| 11 : (gray,black) | 47 : (red,purple) | 83 : (blue,orange) | |
| 12 : (gray,blue) | 48 : (red,red) | 84 : (blue,pink) | |
| 13 : (gray,brown) | 49 : (red,yellow) | 85 : (blue,purple) | |
| 14 : (gray,green) | 50 : (white,aqua) | 86 : (blue,red) | |
| 15 : (gray,orange) | 51 : (white,black) | 87 : (blue,yellow) | |
| 16 : (gray,pink) | 52 : (white,blue) | 88 : (brown,aqua) | |
| 17 : (gray,purple) | 53 : (white,brown) | 89 : (brown,black) | |
| 18 : (gray,red) | 54 : (white,green) | 90 : (brown,blue) | |
| 19 : (gray,yellow) | 55 : (white,orange) | 91 : (brown,brown) | |
| 20 : (green,aqua) | 56 : (white,pink) | 92 : (brown,green) | |
| 21 : (green,black) | 57 : (white,purple) | 93 : (brown,orange) | |
| 22 : (green,blue) | 58 : (white,red) | 94 : (brown,pink) | |
| 23 : (green,brown) | 59 : (white,yellow) | 95 : (brown,purple) | |
| 24 : (green,green) | 60 : (black,aqua) | 96 : (brown,red) | |
| 25 : (green,orange) | 61 : (black,blue) | 97 : (brown,yellow) | |
| 26 : (green,pink) | 62 : (black,brown) | 98 : (pink,aqua) | |
| 27 : (green,purple) | 63 : (black,green) | 99 : (pink,black) | |
| 28 : (green,red) | 64 : (black,orange) | 100 : (pink,blue) | |
| 29 : (green,yellow) | 65 : (black,pink) | 101 : (pink,brown) | |
| 30 : (orange,aqua) | 66 : (black,purple) | 102 : (pink,green) | |
| 31 : (orange,black) | 67 : (black,red) | 103 : (pink,orange) | |
| 32 : (orange,blue) | 68 : (black,yellow) | 104 : (pink,pink) | |
| 33 : (orange,brown) | 69 : (blonde,aqua) | 105 : (pink,purple) | |
| 34 : (orange,green) | 70 : (blonde,black) | 106 : (pink,red) | |
| 35 : (orange,orange) | 71 : (blonde,blue) | 107 : (pink,yellow) | |

Loss

左圖：前21個epoch，右圖：後22個epoch。

| | |
|-------|--|
| |  |
| Score | <p>左圖：前21個epoch，右圖：後22個epoch。</p>  |

ii. Comment

1. 由於Conditional GAN比較難train，在此將目前較好的架構寫進報告。
2. 從生成圖可觀察到在訓練初期（epoch = 18）時臉仍然模糊，髮色也容易有錯誤的情況，顯示比起原本的not-conditional DCGAN，模型需要較多的時間去學習出好的結果。在後期結果（epoch = 36）中，則可以在髮色上看到明顯的區別，眼睛的部分則還是由於模糊的關係不一定會成功。
3. 觀察Loss與Score曲線，常可觀察到有時候會發生D的loss單調下降、Score單調上升，G則恰好相反的情況，此時生成的圖就會崩壞掉，當這段期間結束後，生成圖又會漸漸恢復成原本的样子，但有時就會產生mode collapse的現象。

Compare your model with WGAN, WGAN-GP, LSGAN (choose 1) (Image Generation Only)

1. Model Description of the chosen model (1%)

a. 模型架構 (WGAN-GP)

- i. Discriminator：與第一題DCGAN的架構相同，惟最後改為不通過Sigmoid。
- ii. Generator：與第一題DCGAN的架構相同。

b. 目標函數

- i. Criterion：無。
- ii. Optimizer：RMSprop(lr = 2e-4)。
- iii. D與G訓練的比例為1:1。
- iv. 實作法：

| | |
|---------|--|
| D train | <pre> netD.zero_grad() output1 = netD(real_input) fake = netG(noise) output2 = netD(fake.detach()) gradient_penalty = calc_gradient_penalty(netD, real_input.data, fake.data) errD = output2.mean() - output1.mean() + gradient_penalty errD.backward() optimizerD.step() </pre> |
|---------|--|

| | |
|---------|--|
| G train | <pre>netG.zero_grad() output = netD(fake) errG = -output.mean() errG.backward() D_G_z2 = output.mean() optimizerG.step()</pre> |
|---------|--|

c. Gradient Penalty計算方式

```
def calc_gradient_penalty(netD, real_data, fake_data):
    BATCH_SIZE = real_data.shape[0]
    alpha = torch.rand(BATCH_SIZE, 1)
    alpha = alpha.expand(real_data.size())
    alpha = alpha.cuda()



    interpolates = alpha * real_data + ((1 - alpha) * fake_data)
    interpolates = interpolates.cuda()
    interpolates = Variable(interpolates, requires_grad=True)


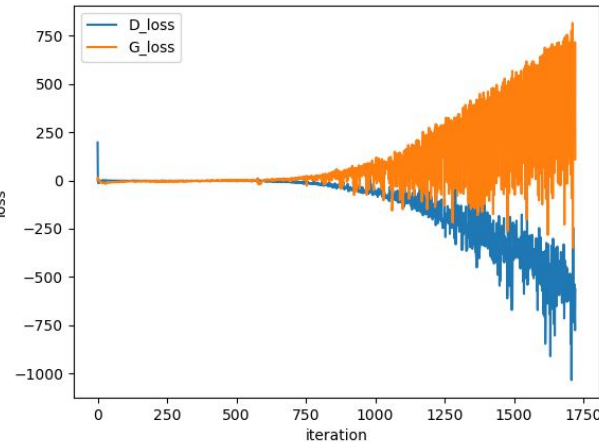
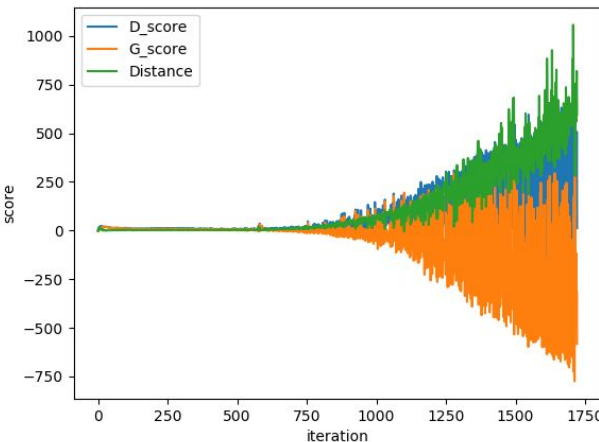
    disc_interpolates = netD(interpolates)
    gradients = grad(outputs=disc_interpolates, inputs=interpolates,
                      grad_outputs=torch.ones(disc_interpolates.size()).cuda(),
                      create_graph=True, retain_graph=True, only_inputs=True)[0]
    gradients = gradients.view(BATCH_SIZE, -1)
    gradients_norm = torch.sqrt(torch.sum(gradients ** 2, dim=1) + 1e-12)
    return 10 * ((gradients_norm - 1) ** 2).mean()
```

- 透過隨機的alpha取樣出real image與fake image在高維空間中連線間的位置，稱作interpolates。
- 將interpolates交給D，得值disc_interpolates，藉由torch.autograd.grad計算梯度。
- 同下式方法，計算出gradient_norm後，以此做運算，最後乘上lambda作為GP的weight（此處為10）。

$$\lambda E[(\|\nabla D(\alpha x + (1 - \alpha)G(z))\|_2 - 1)^2]$$

2. Result of the model (1%)

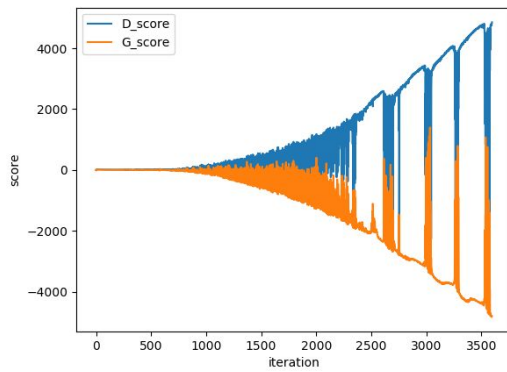
| | | | | |
|-----|---|--|--|--|
| 生成圖 | Epoch = 1 | | Epoch = 2 | |
| |  | |  | |
| | Epoch = 3 | | | |

| | |
|----------|---|
| |  |
| Loss |  |
| Distance |  |

3. Comparison Analysis (1%)

- WGAN在生成圖形的速度比DCGAN快很多，僅僅通過1個epoch就可得到不錯的圖，但是越train下去會越容易有崩潰的情況發生。
- 觀察loss與score，起初750個iteration內的數值都算小，但後面的D和G的變動幅度都很大。Distance的差距越訓練下去也會差越多，感覺沒有收斂的希望。如下圖，有時會出現數值下降的

時候，此時生成圖又會稍微變好。



Training tips for improvement (Image generation Only) (6%): Which tip & implement details (1%)

Result (image or loss...etc.) and Analysis (1%)

1. 控制組實驗環境：
 - a. 使用extra dataset，共36740張圖片。
 - b. 架構為DCGAN。
 - c. Batch size= 64，Epoch= 19。
 - d. Adam optimizer，Lr= 2e-5，Betas= (0.5, 0.999)

2. 讓Discriminator訓練較多次

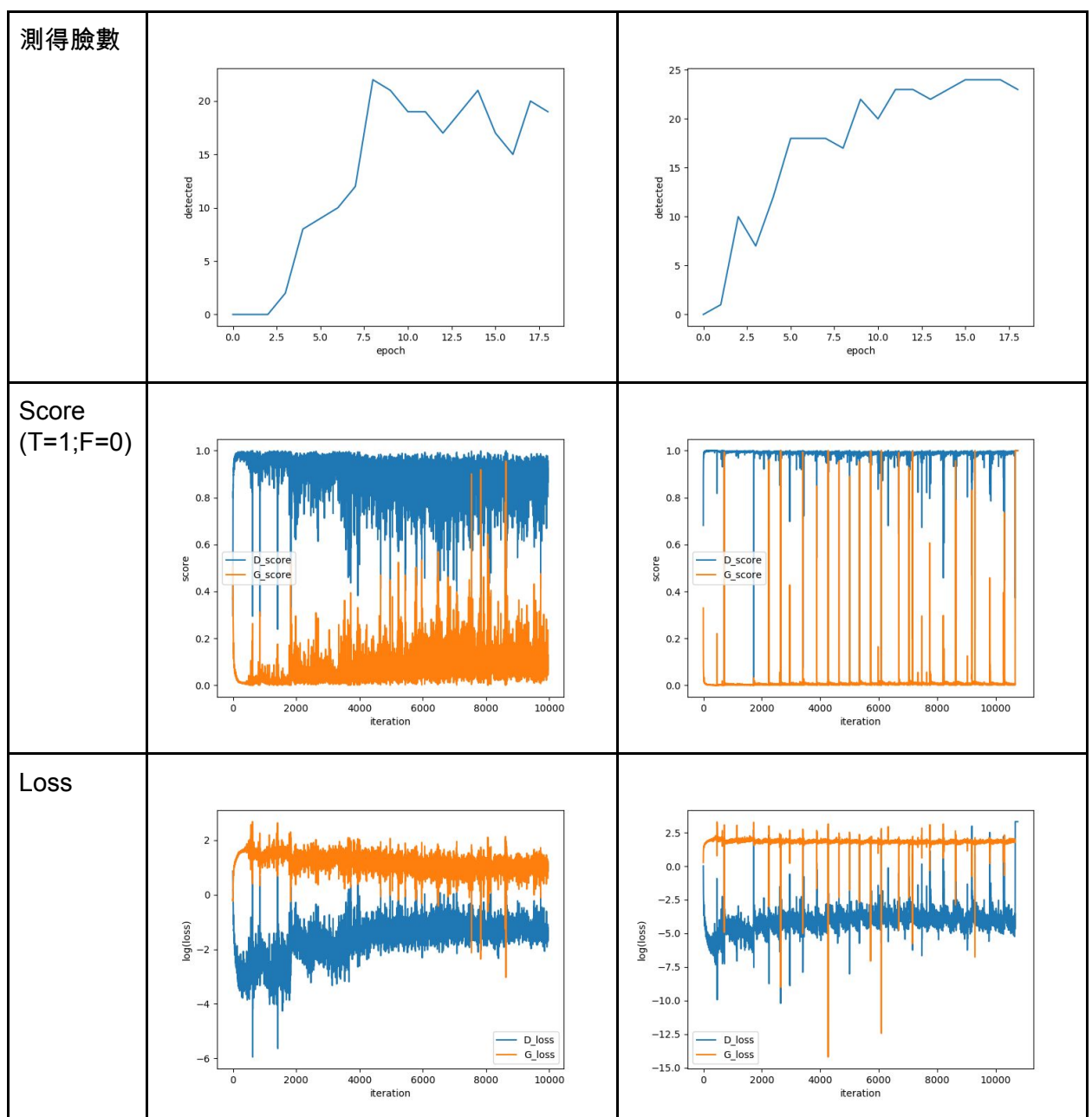
a. 實作

```
for epoch in range(opt.epoch):
    for i, data in enumerate(data_loader, 0):
        total_iter += 1
        #####
        # (1) Train Discriminator
        #   a. Train w/ (true img, true label)
        #   b. Train w/ (fake img, fake label)
        #   c. Backprop w/ optimizer on Discriminator
        #####
        if (total_iter-1) % opt.dtrain == 0:
            #####
            # (2) Train Generator
            #   a. Train w/ (faketrue img, true label)
            #   b. Backprop w/ optimizer on Generator
            #####
```

藉調整opt.dtrain值，決定在內層的iteration是否要對G做訓練的動作。

b. 結果分析

| | 控制組(opt.dtrain = 1) | 實驗組(opt.dtrain = 3) |
|-----|---------------------|---------------------|
| 生成圖 | | |



c. 討論

- 從baseline.py的結果觀察，Discriminator多train幾次有助於產生好的performance，且不容易crash的非常嚴重。
- 在Score及Loss的表現上，控制組的表現看似D跟G對抗的很激烈，變動幅度很大。實驗組的數值則較為穩定，呈現一種D很強的狀態（Loss來到一個很低的狀態），也許是這樣就讓G能更快產生出理想的圖案。

3. 在Generator加入一些Dropout layer

a. 實作

```

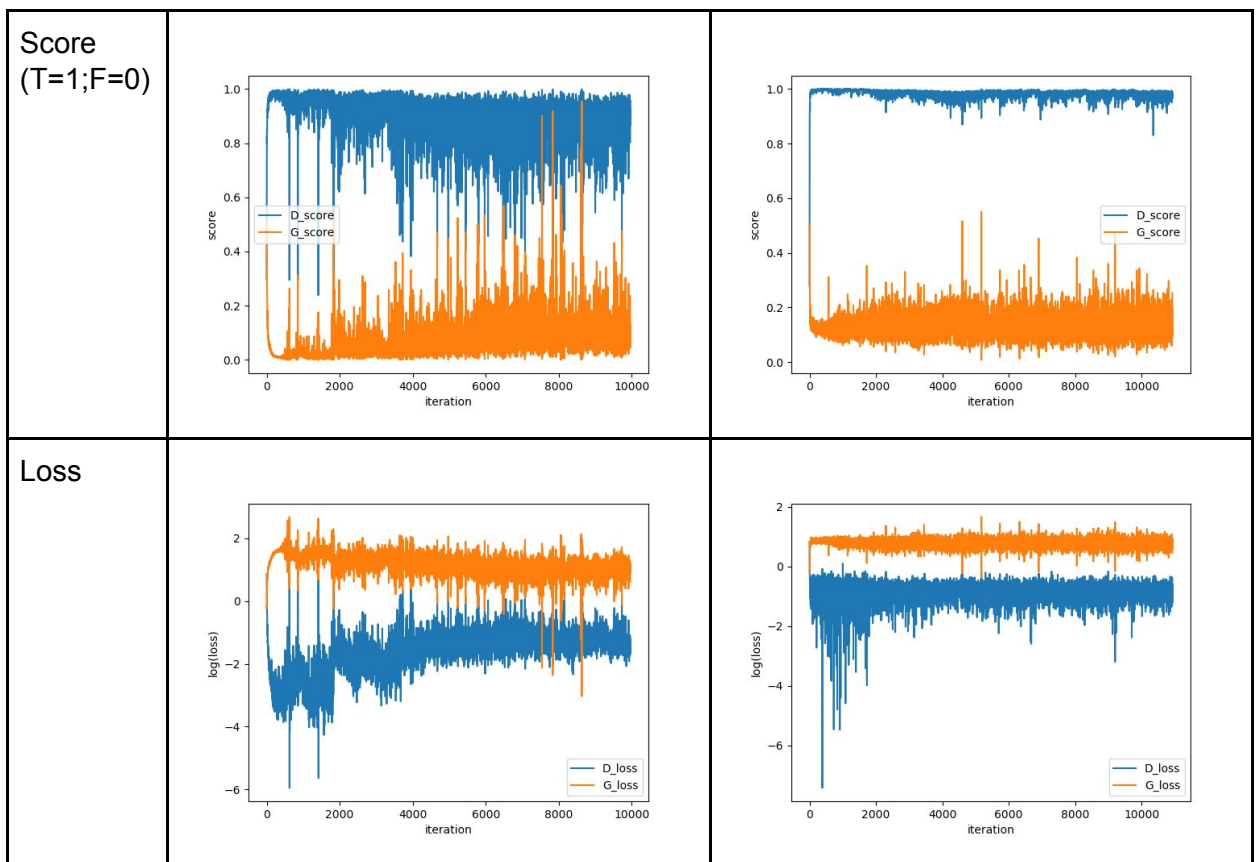
self.main = nn.Sequential(
    # (nz) x 1
    nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=False),
    nn.BatchNorm2d(ngf * 8),
    nn.Dropout(0.5),
    nn.ReLU(True),
    # (ngf*8) x 4 x 4
    nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ngf * 4),
    nn.Dropout(0.5),
    nn.ReLU(True),
    # (ngf*4) x 8 x 8
    nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ngf * 2),
    nn.Dropout(0.5),
    nn.ReLU(True),
    # (ngf*2) x 16 x 16
    nn.ConvTranspose2d(ngf * 2, ngf, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ngf),
    nn.Dropout(0.5),
    nn.ReLU(True),
    # (ngf) x 32 x 32
    nn.ConvTranspose2d(ngf, nc, 4, 2, 1, bias=False),
    nn.Tanh()
    # (nc) x 64 x 64
)

```

在Generator架構中，通過Upsampling後還須額外通過Dropout layer，dropout rate= 0.5，共有4層。（控制組則完全不做Dropout）

b. 結果分析

| | 控制組(opt.dtrain = 1) | 實驗組 |
|------|---------------------|-----|
| 生成圖 | | |
| 測得臉數 | | |



c. 討論

- 從生成圖觀察，可見實驗組的圖仍存在許多雜訊，沒辦法呈現像控制組所產生較為清晰、細節較多的圖案。此外，baseline.py的結果也顯示實驗組的學習情形較差。
- 在Score的表現上，呈現一種D的分數始終很高的狀況，雖然類似於前一項tip的表現，但這次的G score並沒有顯得特別低。Loss的表現則是沒有出現兩者趨勢對抗的情形（如控制組D loss會逐漸因G的表現更好而上升）。
- 在這個實驗中D和G訓練比例為1:1，表現之所以會差可能也是因為model本身就較差的緣故，這時再dropoutG的參數就容易讓performance壞掉，但相信dropout在train更久的情形下還是能起到避免overfit或collapse的作用拉。

4. 使用Noisy label



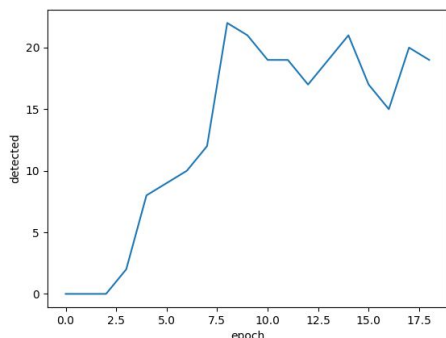
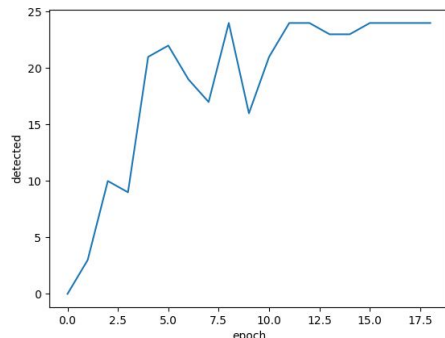
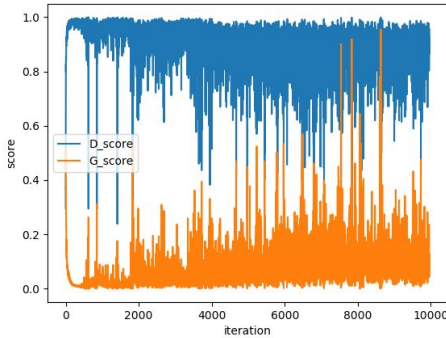
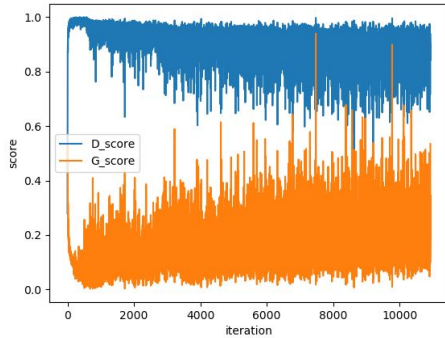
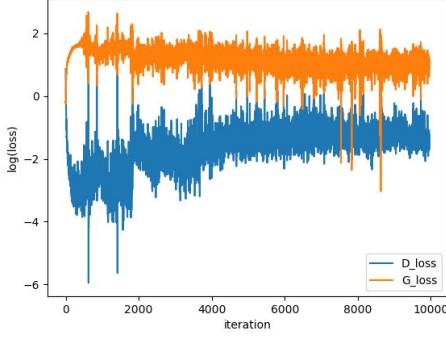
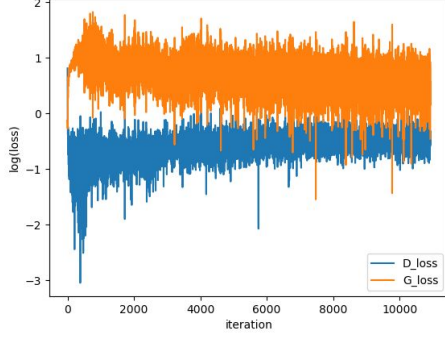
a. 實作

```
if opt.noiselabel:
    tlabel = Variable(torch.FloatTensor(torch.ones(batch_size, 1)+(torch.rand(batch_size, 1)-0.5)*0.5))
    flabel = Variable(torch.FloatTensor(torch.zeros(batch_size, 1)+(torch.rand(batch_size, 1)*0.25)))
```

對label的數值加上uniform distribution的雜訊，其值再乘上權重以調整label變化的幅度。控制組的True label = 1; False label = 0。實驗組的True label介於0.75-1.25; False label介於0-0.25。

b. 結果分析

| | 控制組 | 實驗組 |
|--|-----|-----|
|--|-----|-----|

| | | |
|--------------------|---|---|
| 生成圖 |  |  |
| 測得臉數 |  |  |
| Score (T=1;F=0) |  |  |
| Loss |  |  |

c. 討論

- i. 觀察生成圖與baseline.py的分數，可見實驗組的圖崩壞程度較少，學習速度較快，performance也較好。
- ii. Score與Loss的表現上，兩組的趨勢很類似，惟由於noisy labeling的關係，實驗組的震盪情形較為嚴重。但也是由於加了noise的關係使得model可更robust。

Style Transfer (2%)

在3-3這個實驗中我們研究cycleGAN使用LSGAN與否對訓練的情況有何影響。資料使用[Berkely](#)所提供的“maps” dataset來訓練，這個dataset提供google map和衛星圖兩種domain的影像。而模型的部份，Convolution Layer都有使用ResNet的架構，因此可以疊得較深。

1. 模型架構 (cycleGAN)

在Discriminator中，我們可以在最後一層加入sigmoid函數讓output限縮在[0, 1]之間，這是naive GAN的作法，而如果將之去掉直接輸出Conv2d()的結果，則是LSGAN的架構。預設是使用LSGAN的，而我們將在此加入sigmoid函數比較差別。

a. Generator

| number of block | block |
|---------------------|--|
| 1 | nn.ReflectionPad2d() nn.Conv2d() nn.BatchNorm2d() nn.ReLU() |
| 2 | nn.Conv2d() nn.BatchNorm2d() nn.ReLU() |
| 6 (resNet Block) | nn.Conv2d() nn.BatchNorm2d() nn.ReLU() |
| 2 | nn.ConvTranspose2d() nn.BatchNorm2d() nn.ReLU() |
| 1 | nn.ReflectionPad2d() nn.Conv2d() nn.Tanh() |

b. Discriminator

| number of block | block |
|-----------------|--|
| 1 | nn.Conv2d() nn.LeakyReLU(0.2) |
| 3 | nn.ConvTranspose2d() nn.BatchNorm2d() nn.LeakyReLU(0.2) |
| 1 | nn.ConvTranspose2d() nn.BatchNorm2d() nn.LeakyReLU(0.2) nn.Conv2d() nn.Sigmoid() |

2. 目標函數

a. GAN: Mean Squared Error(MSE)/Binary Crossentropy Error(BCE)

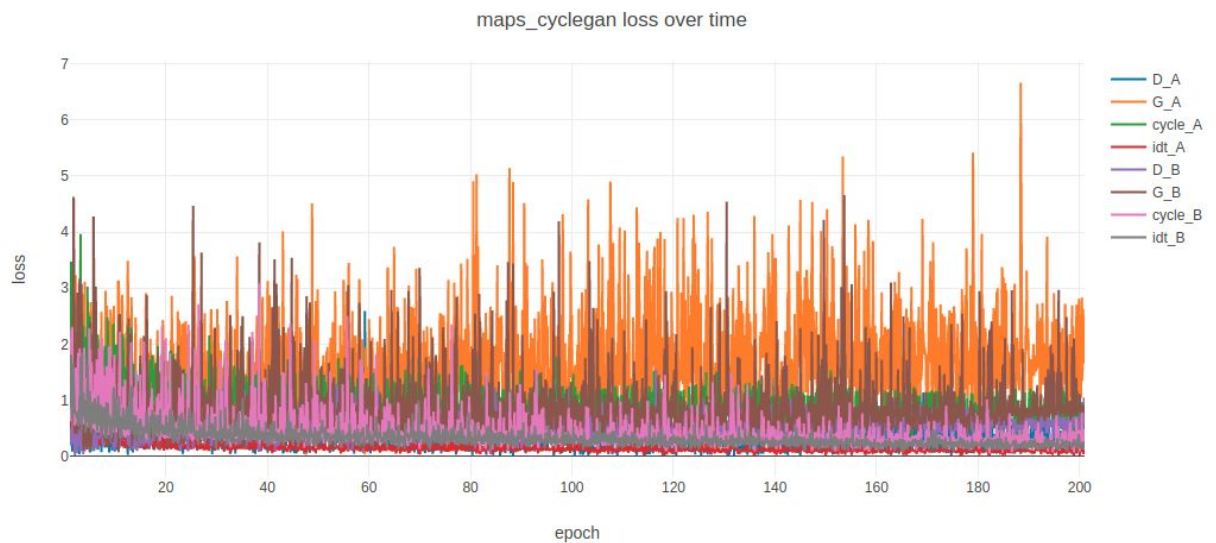
- b. Cycle: L1loss
- c. 若是LSGAN架構，Discriminator的loss為MSE，而若為naive GAN則為BSE，與3-1相同。而input在通過兩個Generator轉回到原本domain這個步驟，我們希望輸出和輸入盡量相同，這個部份使用的是L1loss，在”Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”這篇論文中，作者經過實驗認為更換為其他criterion並沒有表現上的進步，因此就繼續沿用。

3. 訓練細節

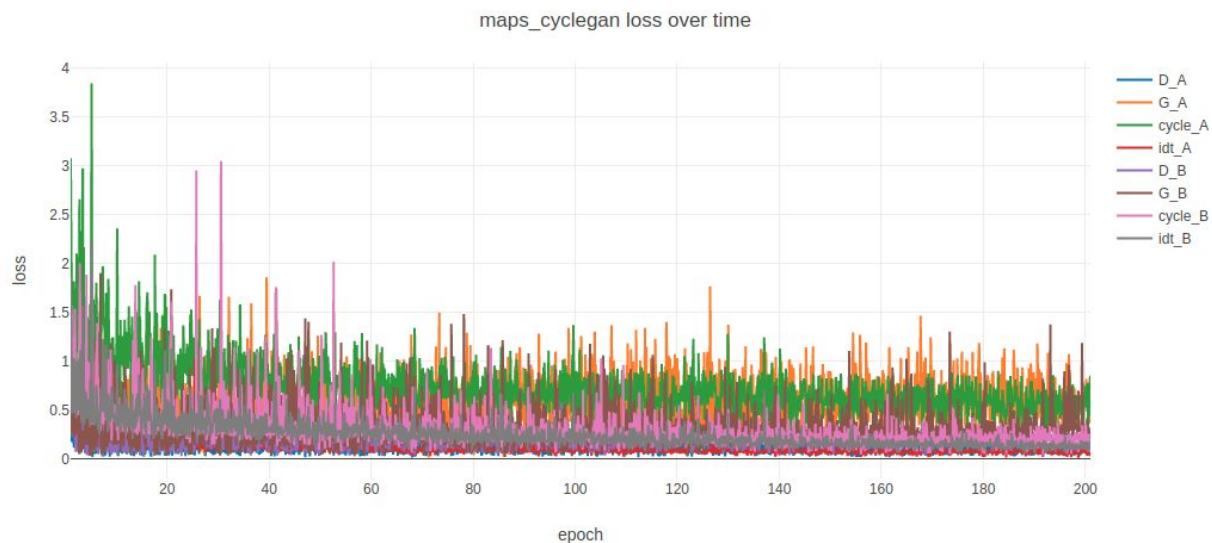
- a. epoch: 200
- b. learning rate: $2e-4$
- c. betas=(0.5, 0.999)
- d. Adam optimizer

Show your result (1%)

- without LSGAN



- with LSGAN



| | | | |
|--------|--------|-------|-------|
| real_A | fake_B | rec_A | idt_A |
| real_B | fake_A | rec_B | idt_B |

Analysis (1%)

在不使用LSGAN架構的結果中，可以看到Generator的loss比使用LSGAN架構的結果高出許多，而data在兩個domain間的轉換出現雜訊的比例也較高，像下圖在google map domain的圖中就出現了粉紅色的模糊影像。LSGAN的設計相較於naive GAN似乎更能讓Generator產生符合真實data的distribution，原因在於naive GAN使用sigmoid讓Discriminator做二元分類，當值接近1或0時，Generator很難再訓練得更好，因為梯度已經接近於0，所以產生出來的分佈沒辦法逼近真實，而使用LSGAN架構較沒有這個問題，我們推測這是造成結果更差的原因。



- StarGan

另外也有嘗試使用Celeb A的dataset訓練StarGAN，訓練 2500 epoch。對 input 同時增加下列特徵
Black hair, Male, Young, Mustache。



分工表

| | |
|-----|-------|
| 孫盟強 | HW3-3 |
| 塗是澂 | HW3-2 |
| 葛竑志 | HW3-1 |