

MLDS HW 3 Generative Adversarial Network (GAN)

A. Model description

Data Preprocess

a. Image Generation

在 faces 的 folder 共有 33431 張圖片，大小為 96 pixels x 96 pixels；將無標註眼睛、頭髮顏色，或同時具有多個頭髮、眼睛顏色 tags 的圖片去除；再者，bicolored eyes 的圖片可能會對 network 造成混淆，在此也予以捨棄。對於剩下的每張 image 先 resize 成 64 pixels x 64 pixels，再將其作水平翻轉與順逆時針 5 degrees 的旋轉以藉由 data augmentation 擴增 training data 的資料量，最後產生的 training data 共有 45628 張圖片。

將 training minibatch size 設為 50，每個 iteration 從 training data 中 randomly sample 50 張 images；對每張 image 都先做 normalization $((\text{pixel value} - 127.5) / 127.5)$ ，形成數值介於 -1 至 1，shape 為 (64,64,3) 的 vector；將這 50 個 vector 形成一個 batch 以作為 discriminator 的 input。

Generator input 為 50(batch size) 個由 normal distribution 隨機產生的、shape 為 (1,1,100) 的 random noise vectors

b. Text-to-image Generation

在 faces 的 folder 共有 33430 張圖片，大小為 96 pixels x 96 pixels；將無標註眼睛、頭髮顏色，或同時具有多個頭髮、眼睛顏色 tags 的圖片去除；再者，bicolored eyes 的圖片可能會對 network 造成混淆，在此也予以捨棄。對於剩下的每張 image 作水平翻轉與順逆時針 5 degrees 的旋轉以藉由 data augmentation 擴增 training data 的資料量，最後產生的 training data 共有 45628 張圖片。

將 training minibatch size 設為 64，generator input 為將頭髮、眼睛顏色的 onehot vector concatenate 而成的 23(12+11) 維的 vector；而 discriminator input 則為一個大小為 64 的 batch，batch 是由 64 個 shape 為 (96,96,3) 的 vector 所組成；而每個 vector 是由對每張 image 做 normalization $((\text{pixel value} - 127.5) / 127.5)$ 而來。

每個 iteration 從 training data 中 randomly sample 50 張 images；對每張 image 都先做 normalization $((\text{pixel value} - 127.5) / 127.5)$ ，形成數值介於 -1 至 1，shape 為 (64,64,3) 的 vector；將這 50 個 vector 形成一個 batch 以作為 discriminator 的 input。

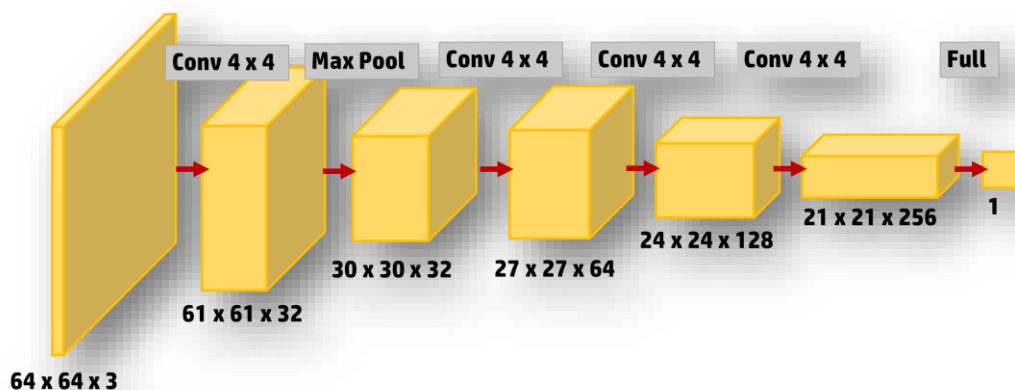
Model Architecture

a. Image Generation

Discriminator 和 generator 是採取 DCGAN 的架構，詳細架構如下：

Discriminator

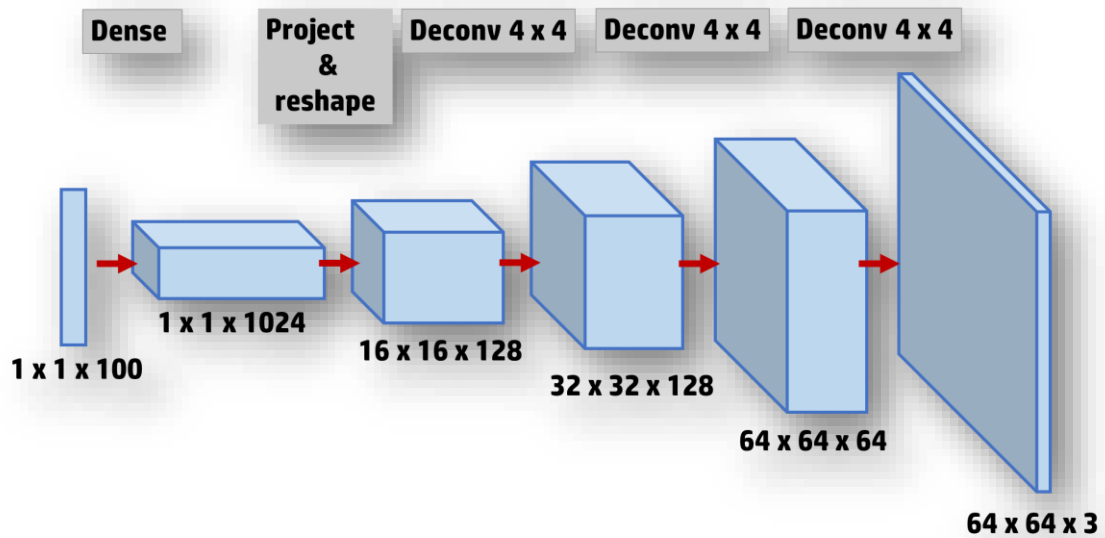
- Input: images(50,64,64,3)
- Output: scores(50,)
- Objective function: $L_D = \min -E[D(x)] + E[D(G(z))]$ z: noise



None		
Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 61, 61, 32)	1568
batch_normalization_1 (Batch Normalization)	(None, 61, 61, 32)	128
leaky_re_lu_1 (LeakyReLU)	(None, 61, 61, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_2 (Conv2D)	(None, 27, 27, 64)	32832
batch_normalization_2 (Batch Normalization)	(None, 27, 27, 64)	256
leaky_re_lu_2 (LeakyReLU)	(None, 27, 27, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	131200
batch_normalization_3 (Batch Normalization)	(None, 24, 24, 128)	512
leaky_re_lu_3 (LeakyReLU)	(None, 24, 24, 128)	0
conv2d_4 (Conv2D)	(None, 21, 21, 256)	524544
batch_normalization_4 (Batch Normalization)	(None, 21, 21, 256)	1024
leaky_re_lu_4 (LeakyReLU)	(None, 21, 21, 256)	0
flatten_1 (Flatten)	(None, 112896)	0
dense_1 (Dense)	(None, 1)	112897
activation_1 (Activation)	(None, 1)	0
=====		
Total params: 804,961		
Trainable params: 0		
Non-trainable params: 804,961		

Generator

- Input: noises (50,1,1,100)
- Output: images (50,64,64,3)
- Objective function: $L_G = \min -E[D(G(z))]$ z: noise



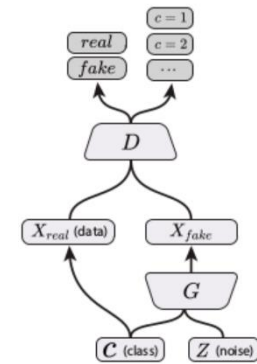
Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 1, 1, 1024)	103424
activation_2 (Activation)	(None, 1, 1, 1024)	0
dense_3 (Dense)	(None, 1, 1, 32768)	33587200
batch_normalization_5 (Batch Normalization)	(None, 1, 1, 32768)	131072
activation_3 (Activation)	(None, 1, 1, 32768)	0
dropout_1 (Dropout)	(None, 1, 1, 32768)	0
reshape_1 (Reshape)	(None, 16, 16, 128)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 32, 32, 128)	262272
batch_normalization_6 (Batch Normalization)	(None, 32, 32, 128)	512
activation_4 (Activation)	(None, 32, 32, 128)	0
dropout_2 (Dropout)	(None, 32, 32, 128)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 64, 64, 64)	131136
batch_normalization_7 (Batch Normalization)	(None, 64, 64, 64)	256
activation_5 (Activation)	(None, 64, 64, 64)	0
dropout_3 (Dropout)	(None, 64, 64, 64)	0
conv2d_5 (Conv2D)	(None, 64, 64, 3)	3075
batch_normalization_8 (Batch Normalization)	(None, 64, 64, 3)	12
activation_6 (Activation)	(None, 64, 64, 3)	0
Total params: 34,218,959		
Trainable params: 34,153,033		
Non-trainable params: 65,926		

b. Text-to-image Generation

Reference:

<https://arxiv.org/pdf/1610.09585.pdf>

https://makegirlsmoe.github.io/assets/pdf/technical_report.pdf



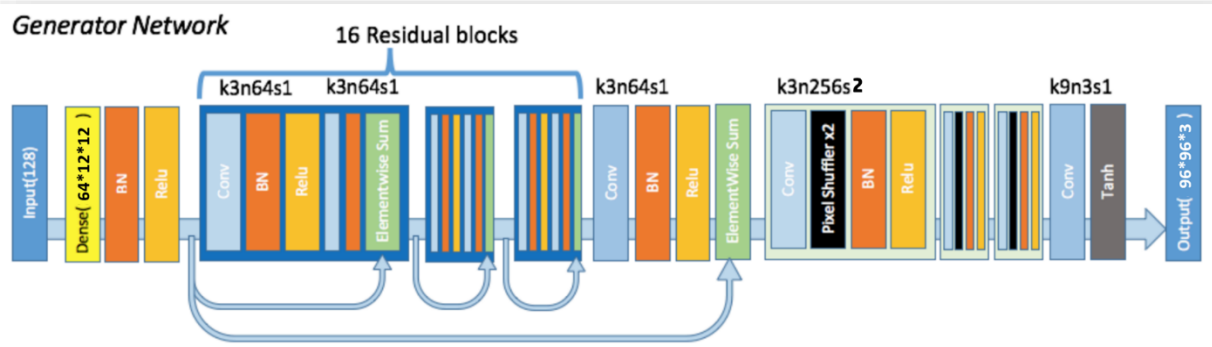
ACGAN

在這個 task 中，Model 架構為 ACGAN 架構，如右上示意圖所示。在 ACGAN 中，Generator 使用 noise z 以及一個對應的 class label $c \sim P_c$ 生成 fake image $X_{fake} = G(z, c)$ ；而 Discriminator 則會根據 $X_{real}(data)$ 及 X_{fake} 給出 input image 的 real/fake 概率分布以及 class label 上的概率分布 $P(S | X), P(C | X) = D(X)$ 。

Generator

Generator 的 network 架構採取 SRResnet 的架構，input 為 noise z + 眼睛頭髮顏色的 tags 所組成的維度為 $(100(z) + 23(tag))$ 的 vector。

1. 先經過 fully connected function (dense) 放大為維度 $64 \times 12 \times 12$ 的 tensor，再經過 BN (Batch Normalization) 與 relu function
2. 通過 16 個 residual blocks (channel = 64)；每個 residual block 內為兩層(conv layer (k3n64s1)+BN+relu)
3. 再經過三個 block，每個 block 是由 conv layer(k3n256s2)+兩層 pixelShuffler(used to do upscaling)+BN+relu 所組成
4. 最後經過 conv layer (k9n3s1)，通過 activation function tanh 形成 $96 \times 96 \times 3$ 的 output image X_{fake}



5. minimize

$$\mathcal{L}(G) = \lambda_{adv} \mathcal{L}_{adv}(G) + \mathcal{L}_{cls}(G)$$

$$\mathcal{L}_{adv}(G) = \mathbb{E}_{x \sim P_{noise}, c \sim P_{cond}} [\log(D(G(z, c)))]$$

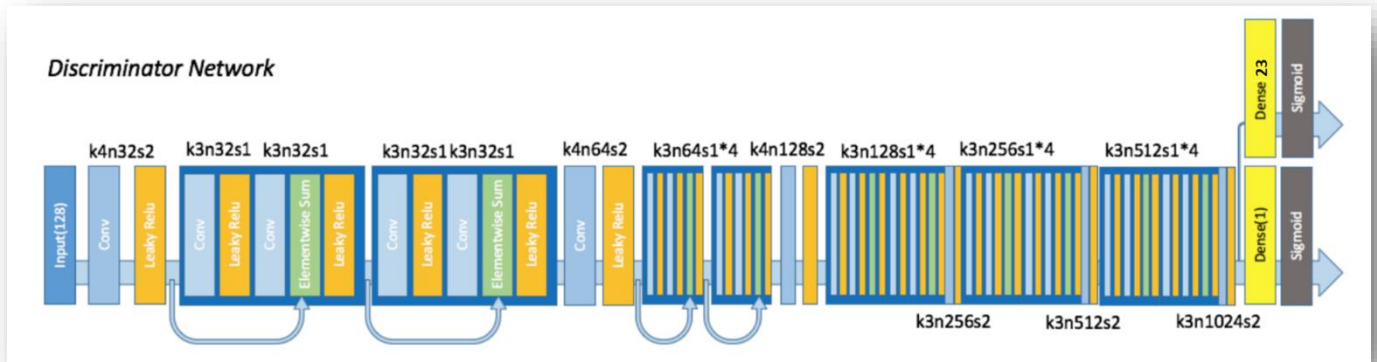
$$\mathcal{L}_{cls}(G) = \mathbb{E}_{x \sim P_{noise}, c \sim P_{cond}} [\log(P_D[c|G(z, c)])]$$

P.S. P_{cond} indicates the prior distribution of assigned tags. λ_{adv} is the balance factor for the adversarial loss.

Discriminator

Discriminator 的 input 為 $96 \times 96 \times 3$ 的 input image，依序經過

1. conv layer(k4n32s2), 兩個 residual block(2*conv layer(k3n32s1)), conv layer(k4n64s3)
2. 四個 residual block (2*conv layer(k3n64s1)), conv layer(k4n128s2)
3. 四個 residual block (2*conv layer(k3n128s1)), conv layer(k3n256s2)
4. 四個 residual block (2*conv layer(k3n256s1)), conv layer(k3n512s2)
5. 四個 residual block (2*conv layer(k3n512s1)), conv layer(k3n1024s2)
6. 最後再經過 fully connected 輸出一 scalar，同時也拉出一 branch 以輸出 23 維的 vector 來做 multilabel classification.



7. minimize

$$\mathcal{L}(D) = \mathcal{L}_{cls}(D) + \lambda_{adv} \mathcal{L}_{adv}(D)$$

$$\mathcal{L}_{adv}(D) = -\mathbb{E}_{x \sim P_{data}} [\log D(x)] - \mathbb{E}_{x \sim P_{noise}, c \sim P_{cond}} [\log(1 - D(G(z, c)))]$$

$$\mathcal{L}_{cls}(D) = \mathbb{E}_{x \sim P_{data}} [\log P_D[label_x|x]]$$

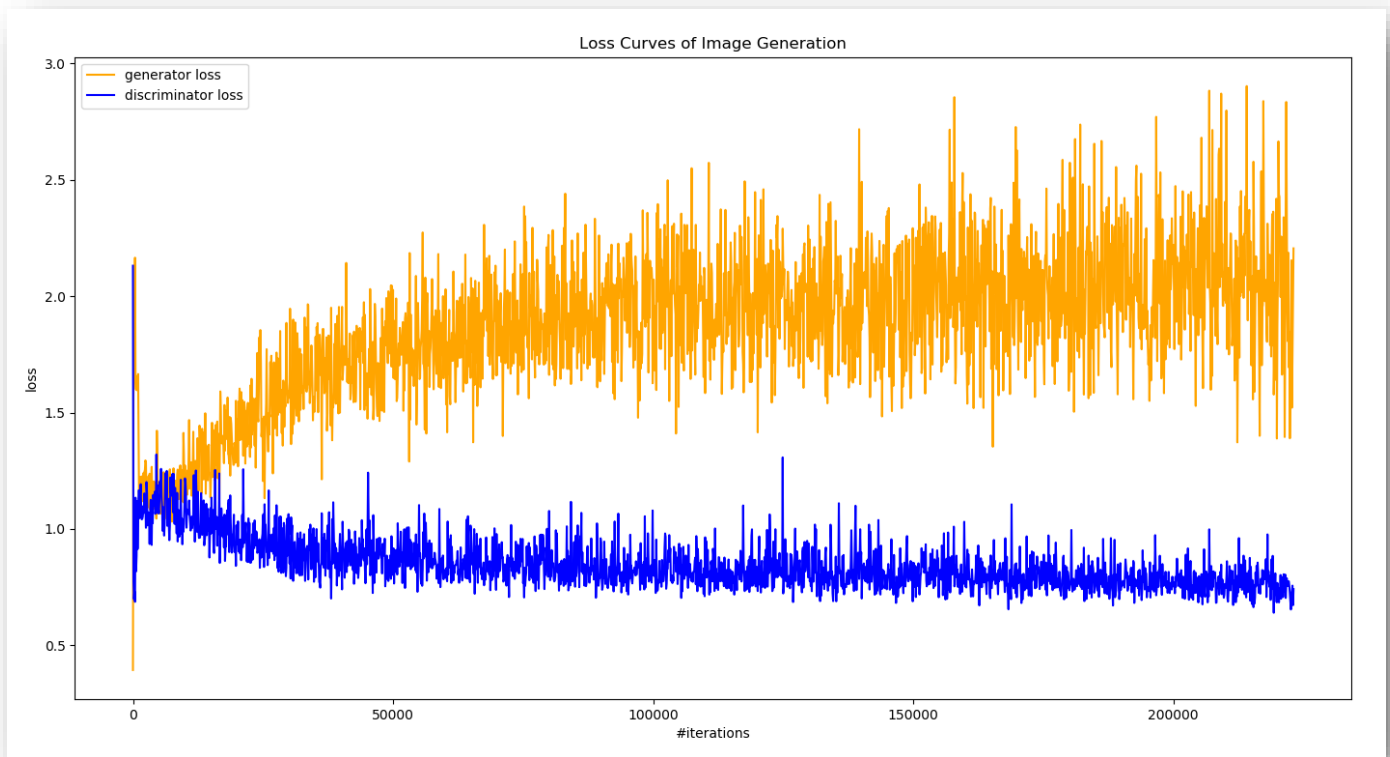
P.S. P_{cond} indicates the prior distribution of assigned tags. λ_{adv} is the balance factor for the adversarial loss.

B. Experiment settings and observation

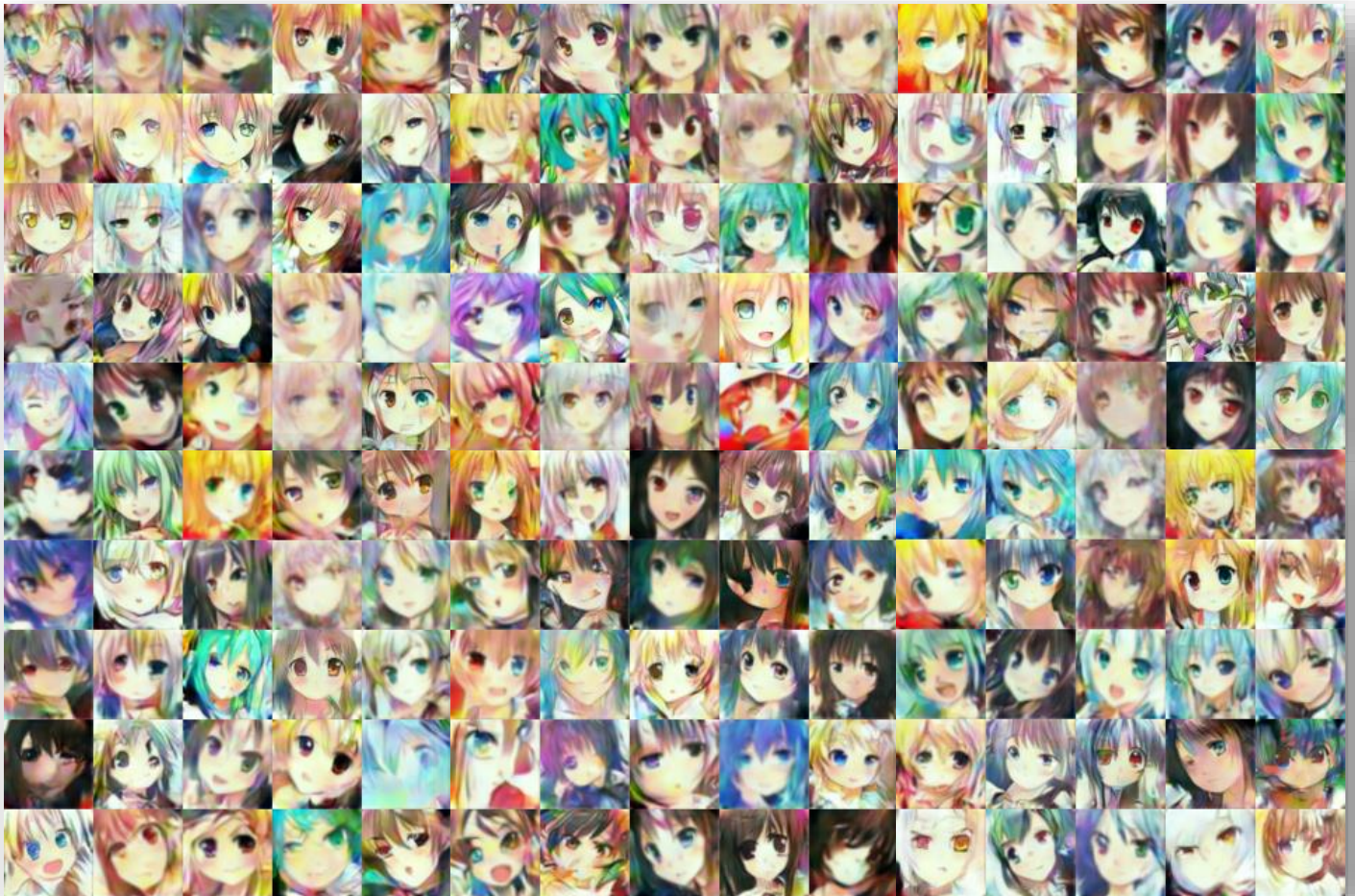
a. Image Generation

- Batch size = 50
- Update times of Generator and Discriminator in one epoch = 1: 1
- Optimizer
Discriminator (Adam lr=0.00020, beta_1 = 0.5)
Generator (Adam lr =0.00015, beta_1 = 0.5)
- BatchNormalization
Discriminator / Generator (0.5)
- Noise:
Normal distribution
Dim = 100

- Loss records of training process

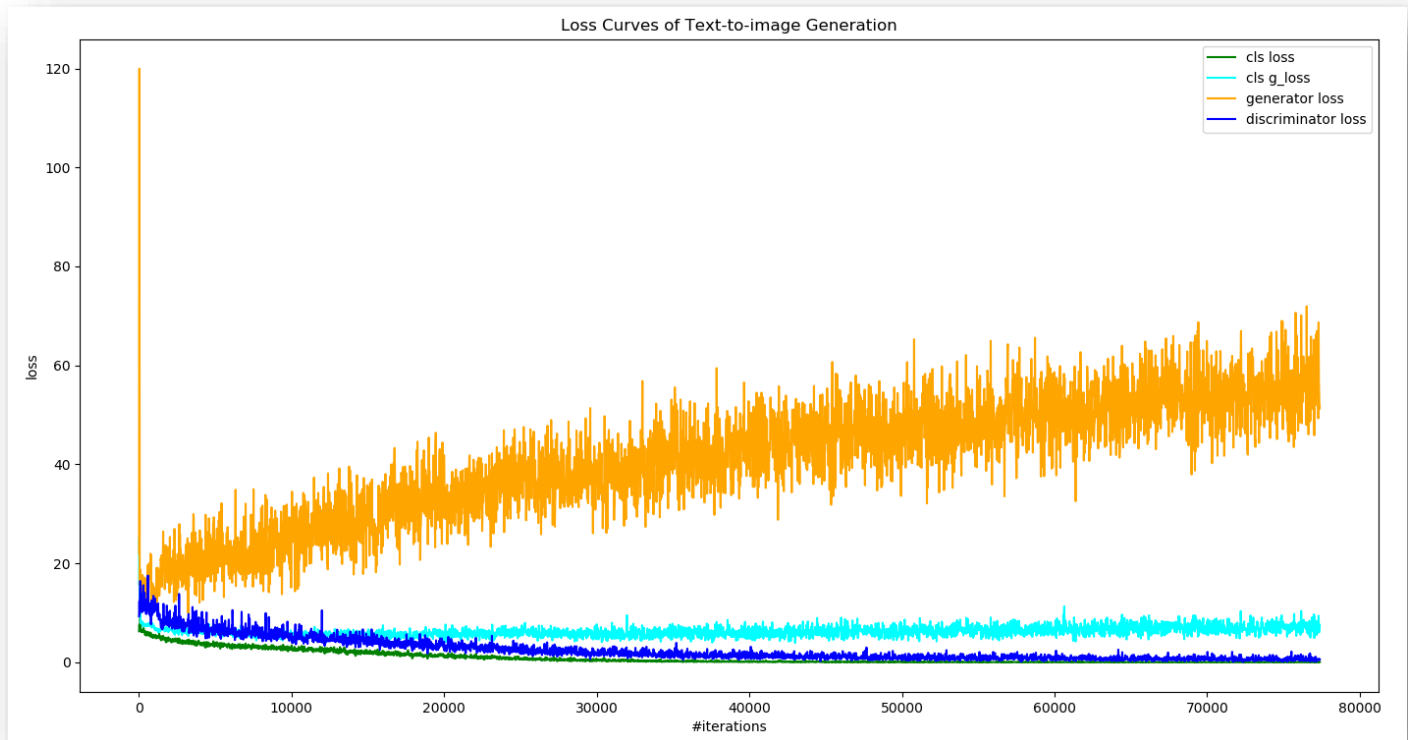


- Generated images



b. Text-to-image Generation

- Batch size = 64
- Update times of Generator and Discriminator in one epoch = 1:1
- Optimizer
Discriminator (Adam lr=0.00020, beta_1 = 0.5, beta_2 =0.9)
Generator (Adam lr =0.00020, beta_1 = 0.5, beta_2 = 0.9)
- BatchNormalization
Discriminator / Generator (0.5)
- Noise:
Normal distribution
Dim = 100
- 除了 discriminator 的 objective function 外，為穩定 training，在 discriminator loss 加上對 real image class 的 prediction error；並加上 gradient penalty techniques.
- Loss records of training process



- Generated images



C. Compare your model with WGAN/ WGAN-GP/ LSGAN

a. Model Description of WGAN

由於原始的 GAN 會因為 JS Divergence 衡量分布距離時突變的性質，在訓練 Generator 時會出現 Gradient Vanish 的問題；於是，WGAN 改用 Wasserstein Distance ($K \cdot \text{mean}(y_{\text{true}} * y_{\text{predicted}})$) 作為 loss function 以避免 mode collapses (generating similar images even when fully connected layer GANS are used or when batch normalization is not used).

在本次實作的 WGAN model 中，Discriminator 和 Generator 的架構都與 DCGAN 大致相同；但 objective function 及 Discriminator output 需做修改，實際 implementation details 如下：

- 將 Discriminator 最後一層 sigmoid function 拿掉
- 對 Discriminator 做 weight clipping，限制在 $[-c, c]$ 之間， c 取 0.01
- 選擇 RMSProp (lr = 0.00005) 作為 Optimizer
- Update times of Generator and Discriminator in one epoch:
 - The critic is trained for 100 times for each generator train iteration in the first 25 generator train updates. After this, it is trained for 5 iterations for each generator iteration (every 500 generator iterations the critic is trained again for 100 times).
- Loss function

$$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$$
$$L_G^{WGAN} = E[D(G(z))]$$
$$W_D \leftarrow \text{clip_by_value}(W_D, -0.01, 0.01)$$

b. Result of the model

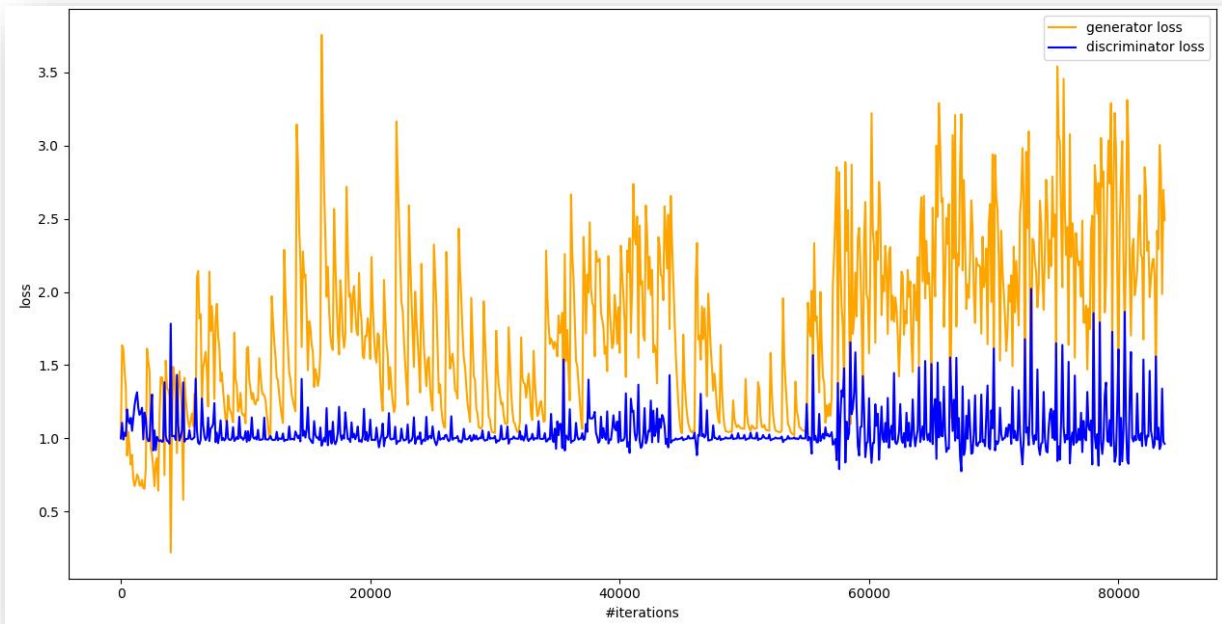


c. Comparison Analysis

理論上在 WGAN 中由於在同一 epoch 下會對 discriminator update 較多次，應可有效減緩 mode collapse 的情形；然而實際比較原先 model 與 WGAN model 所產生的 Image，可以發現 WGAN model 所產生的 image 解析度較低、face features 不明顯、mode collapse 情形改善不明顯，整體而言 WGAN 的 generated images 沒有原 model 優秀。推測其因可能與 Generator 和 Discriminator 的更新比例有關，但礙於時間，並沒有再針對兩者的更新比例做進一步的分析比較。

D. Training tips for improvement

Original loss curves (without below tips)



Original generated images (without below tips)



- **Analysis**

在尚未加入以下 3 個 tips 前，所生成的圖片解析度較低，face features 較為模糊不明顯；此外也可從 training process 中觀察到 mode collapse 的現象。

- a. Use Soft and Noisy Labels**

- **Implement details**

- 將原先 fake label 由 0 改成 0.0~0.2 間一隨機數，並將原先 real label 由 1 改成 0.8~1.2 間一隨機數，以使 labels 對 Discriminator 來說更加 noisy

- **Result**

- Generated images

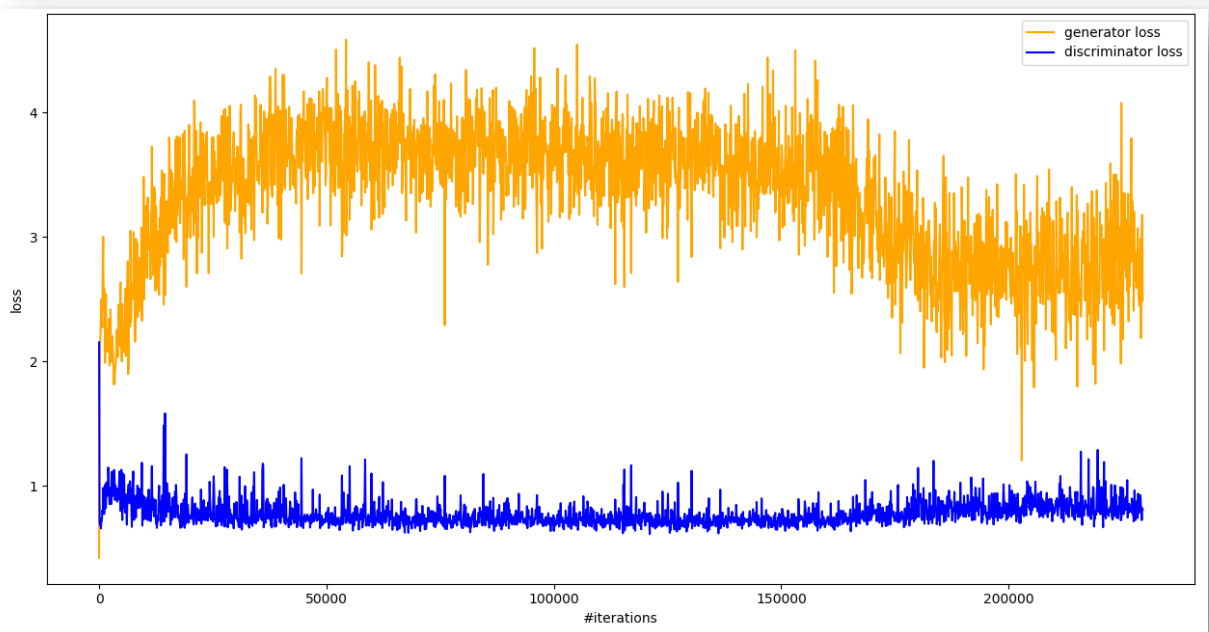


- Analysis

由上圖觀察可知，在使用 **noisy label** 後能有效改善 **model mode collapse** 的問題，使 **generated images** 不過分相似、**distribution** 過於集中。

b. Use Dropouts in G in both train and test phase

- Implement details
 - Provide noise in the form of dropout (50%)
 - Apply on several layers of our generator at both training and test time
- Result
 - Loss curves



- Generated images



- Analysis

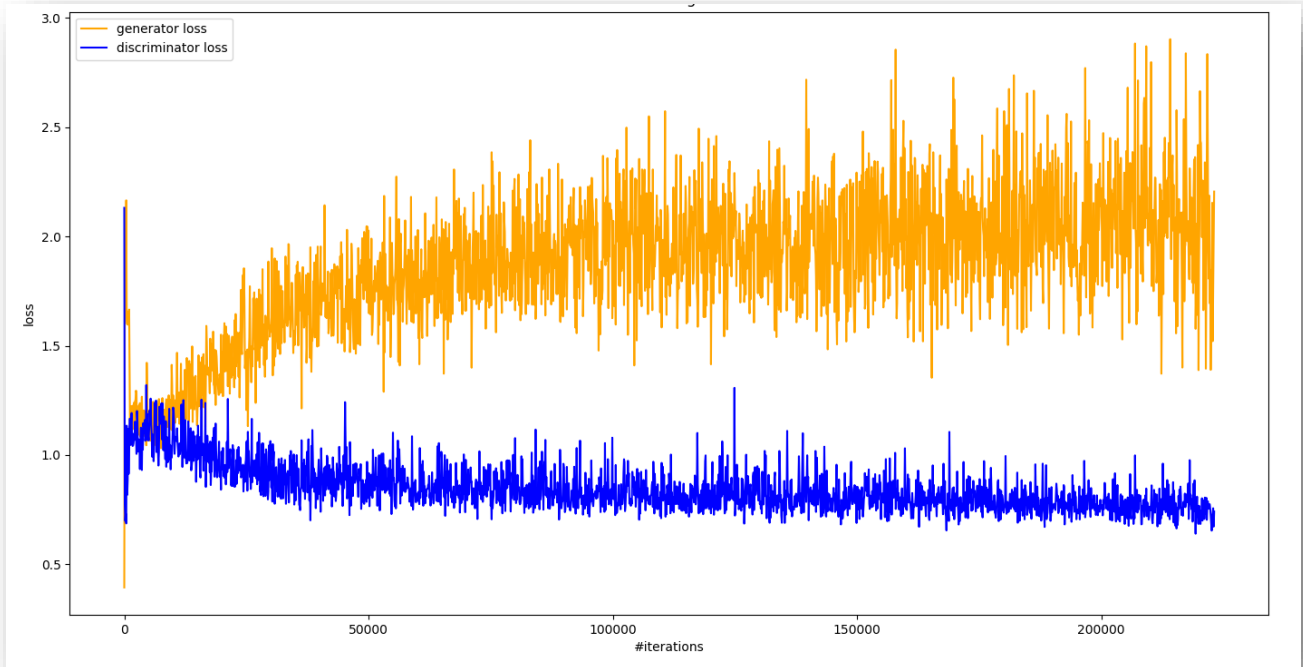
觀察以上 loss curves 及 generated images 可發現；

- Generator 的 loss curve 在 training 初期呈上升，中段大致維持定值，末段則逐漸下降。
- 在 generator 中加入數層的 dropout function(train & test)，可使相同的 input noise 在 generator 的作用下 generated 出差異較大的 anime faces；並同時可以防止 overfitting 的現象。

c. Avoid Sparse Gradients: ReLU, MaxPool

- Implement details
 - 將 Discriminator 及 Generator 中的 activation functions 除了最後一層外，由 ReLU 改為 LeakyReLU
 - 在 Generator 中用 ConvTranspose2d + stride 替代 UpSampling
 - 在 Discriminator 中做 downsampling 時使用 Conv2d + stride

- Result
 - Loss curves



- Generated images









- Analysis

觀察以上 **loss curves** 及 **generated images** 可發現；在使用此 **tip** 後可有效使 **discriminator** 的 **loss** 呈下降趨勢；並使生成圖像的品質大幅進步，不但影像解析度較高、較少 **noise** 干擾，且五官較明顯。





E. Style Transfer

a. Results

1. Photographs to artistic styled drawings

	Input	Style	Output
Result 1			
Result 2			

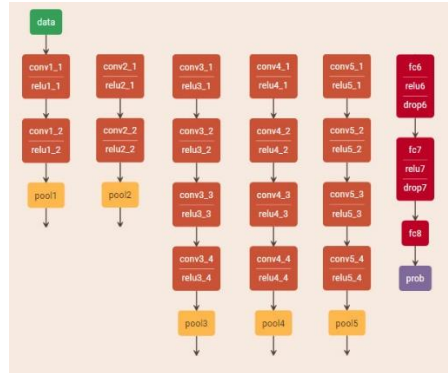
2. Edges to Photos

	Input	Output
edges2cats		
edges2handbags		

b. Analysis

1. Photographs to artistic styled drawings

這裡所使用的 model 是來自 A Neural Algorithm of Artistic Style 這篇 paper ; model 主要是使用 VGG19 的分類網絡(如下圖)，並將 max pooling 換成 average pooling



而此篇 paper 的核心思想就是 Image 的 content 和 style 是可以分離的；則可將 style transfer 轉化為以下目標: 使 output image 的 content 與 input image 盡可能相似；同時使 output image 的 style 與 style source image 相似。

根據上述的目標，可將 loss functions 分為 content loss 及 style loss:

- Content loss function

在 convolution neural network 中，不同層會形成對應 filter 數量的 feature map，而在 content 這個維度上，model 的目標是希望 generated image 的每一個 feature 與 input image 盡可能接近，故定義 content loss function 為

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 .$$

P.S. p: input image (content)

x: generated image (initial input: white noise image)

F, P 分別是兩張圖片在 l 層上 i, j 位置的特徵值

- Style loss function

在這篇 paper 中，使用 gram matrix 來表示 image 的風格，gram matrix 的

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l .$$

定義是，此 matrix 可看作體現了不同 filter 特徵的相互關係，並同時忽略了 content information；可將 l 層的 style loss function 定義

為
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$
 (G, A: generated image/ style source image's gram matrix)

整體 style loss function 為每一層 style loss 的加權:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^L w_l E_l$$

P.S. a: style source image

x: generated image

w: layer weight

- Total loss function

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

利用 back propogation 調整 generated image x 的值，直到產生所需的圖像

由以上 2 個 result 觀察得知，此 model 在 style transfer 的表現上相當好，generated image 不僅有保留細節部分，且除了 style 外其餘部分都和 Input image 類似，無太大失真。

2. Edges to Photos

這裡主要是使用 Image-to-Image Translation with Conditional Adversarial Networks 裡的 model。在這篇 paper 中所使用的 model 最主要是基於 cGAN 的架構，Generator 為 $G: x, z \rightarrow G(x, z) \rightarrow y$ (x : input image, z : input noise image, y : real image)，而 Discriminator 則是藉由區分 $G(x, z)$ 及 y 調整 model 的參數，使 model 能產生更接近 y 的 output image。

而這篇 paper 對於 objective function 做出以下修改；在原先 cGAN model 的目標函數基礎上，加入 L1 norm 項以使 generated image 和原 image 更接近，將 model 的性質改得更像 auto encoder

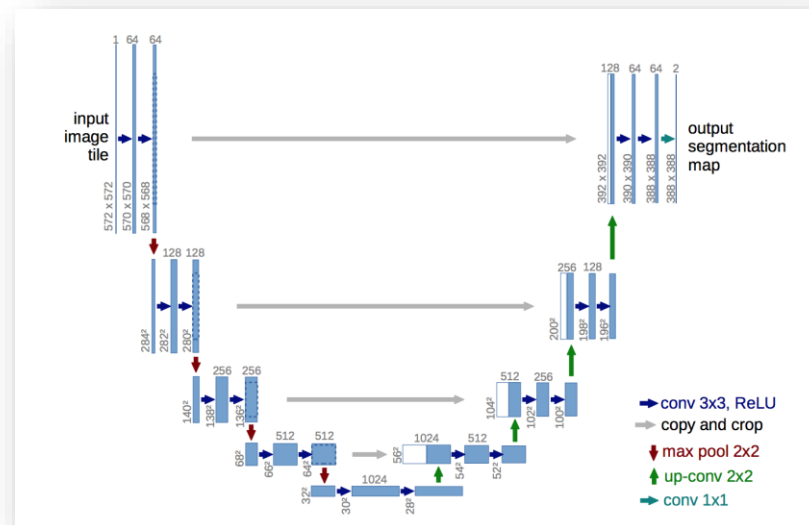
$$L_{GAN}(G, D) = E_{y \sim p_{data}(y)} [\log D(y)] + E_{x \sim p_{data}(x), z \sim p_{data}(z)} [\log(1 - D(G(x, z)))]$$

$$L_{L1}(G) = E_{x, y \sim p_{data}(x, y), z \sim p_{data}(z)} [||y - G(x, z)||]$$

$$G^* = \arg \min_G \max_D L_{GAN}(G, D) + L_{L1}(G)$$

<http://blog.csdn.net/u010620946>

Paper 認為 Input image x 和 targeted image y 存在相同的底部結構，差異只在 surface appearance，故在 Generator 的實作上採用 U-Net(如下):



此外，一般 GAN 的 discriminator 在評斷 generated image 和 real image 時都是做整體(整張 image)的判斷；但在這篇 paper 中則提出分區塊判斷法，將 generated image 分成多區塊給 discriminator 判斷，使 generator 能產生更精緻(保留更多 details)的 output image。

根據以上結果可發現，在 input image 提供足夠細節的情形下，model 能產生不錯的 generated image；但若提供的 input image edge 太少、太簡潔，則容易產生較差的 image(cat 較嚴重)。

F. 分工表

All: b03901156