

## MLDS HW3 Report Question

- **Model Description**

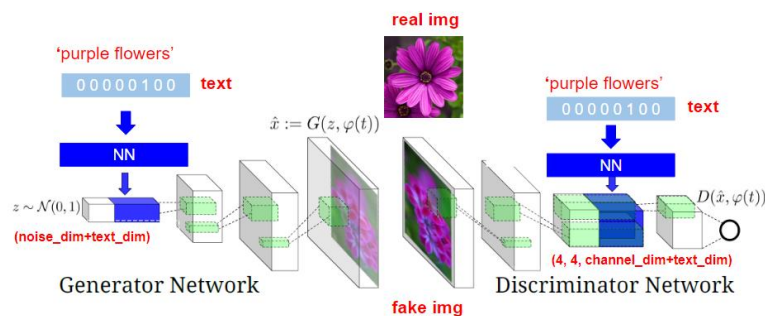
(Describe the models you use to, including the model architecture, objective function for G and D)

- ♦ **Image Generation. (2%)**

在作業 3-1 我參考的模型為 DCGAN，在 generator 的部分 input 為 100 個高斯分布的 noise，整個網路架構為 5 層，從原本 4x4 的大小一路升維到 64x64 的圖形，每一層使用 transpose\_conv2d 其中 strides 為 1,2,2,1 並在每一層使用 batchnorm 和 relu 的激活函數，最後一層輸出層通過 tanh 將數字標準化到 -1~1 之間，使用的 loss function 為  $-\log(\text{sigmoid}(D(G(x))))$ 。而 discriminator 的部分則是捨棄使用 maxpooling 或是 average pooling，全部使用 conv2d 來進行降維，而在這邊激活函數是使用 leaky\_relu，使用的 loss function 為  $\log(D(x)) + \log(1 - D(G(z)))$ 。使用的 Adam optimizer 將學習率特別調為 0.0002 而 beta1 調為 0.5。

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

這邊使用的 model 架構基本上跟上一題是類似的，使用 DCGAN 的作法，generator 為五層，discriminator 為六層，有改變的地方只有在 generator 最前面加入 one-hot encoding 的資訊，也就是頭髮的顏色和眼睛的顏色的資訊，而 discriminator 則是在最後面一層加入 one-hot encoding 的資訊，以下圖片顯示的就是 model 的架構(摘自作業 3-2)，



而 one-hot encoding 的 dimension 為 120(也就是 hair 跟 eye 的排列組合，blue hair blue eyes 為一種、blue hair green eyes 為另外一種)。之後訓練 2 萬個 iterations，也是使用 Adam 的 optimizer。

- **Experiment settings and observation.(Show generated images)**

- ♦ **Image Generation. (1%)**

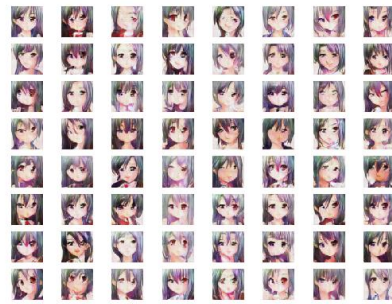
以下為訓練 1000、5000、10000、15000 和 20000 iterations 的結果。



到 1000 iterations 的時候其實就已經可以畫出有不同色彩及不同型態的人像，只不過並不是很清楚，我覺得要一直到 10000 個 iterations 後才變得比較精緻，但其實可以發現有些臉是嚴重的變形的(像是第二排從右邊數來第六個)，不過隨著 iterations 的增加 model 並沒有改善這張臉的輸出，也許代表 model 已經 train 到一個比較收斂的點了，除此之外也可以看到 20000 iteration 的圖並不一定比 15000 的好，像是倒數第二排從右邊數來第四個在 20000 iteration 甚至連眼睛都不見了，也許需要改善 model 的架構來增進圖片的精緻度。

#### ♦ Text-to-image Generation. (1%)

以下為訓練 1000、5000、10000、15000 和 20000 iterations 的結果。



和 DCGAN 一樣的架構但增加 condition 會讓圖像變的比較難以訓練，不過可以看出 model 似乎有抓到規則，因為在這邊我都讓他輸出 aqua hair aqua eyes 的圖像，雖然頭髮有點偏灰白色，但顏色都較為一致，且很多眼睛也都相似 aqua 的顏色，代表 Conditional gan 是有訓練起來的。以下圖片是用不同 condition 所生成的動漫圖像，大致上是有成功生成符合條件的圖像，不過有時候臉會有些扭曲，且眼睛會出現兩顆顏色不一樣的情形出現。



1,blue hair blue eyes	13,blue hair red eyes
2,blue hair blue eyes	14,blue hair red eyes
3,blue hair blue eyes	15,blue hair red eyes
4,blue hair blue eyes	16,green hair blue eyes
5,blue hair blue eyes	17,green hair blue eyes
6,blue hair green eyes	18,green hair blue eyes
7,blue hair green eyes	19,green hair blue eyes
8,blue hair green eyes	20,green hair blue eyes
9,blue hair green eyes	21,green hair red eyes
10,blue hair green eyes	22,green hair red eyes
11,blue hair red eyes	23,green hair red eyes
12,blue hair red eyes	24,green hair red eyes
	25,green hair red eyes



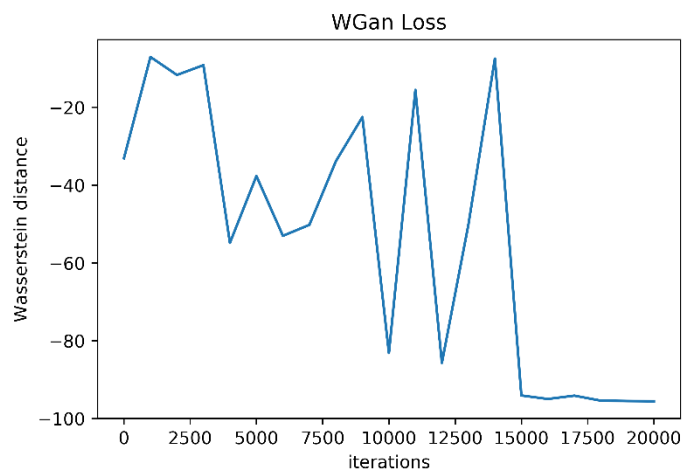
- **Compare your model with WGAN, WGAN-GP, LSGAN(choose 1)**

- ♦ **Model Description of the choosed model. (1%)**

我選擇的是比較 WGAN 與我原本的 model 的差別。原本 model 是依照 DCGAN 去建構的，而要比較的 model 是依照我原本的 model 再加上 WGAN 的技巧建成，其中 WGAN 中所使用的 trick 為 1. Discriminator 最後一層拿掉 sigmoid 函數、2. 生成器和判別器中的 loss 不取 log、3. 每次更新 discriminator 的參數後把它們的絕對值 clip 到不超過某個參數(這邊設的是-0.02~0.02)、4. 將 Adam optimizer 改為 RMSProp 的 optimizer。其他部分像是 model 的層數、每層的神經元數等等都不做更動，比較好比使用一般 GAN 的算法和使用 Wasserstein 距離之間的差別。

- ♦ **Result of the model. (1%)**

因為使用 WGAN 時可以計算 Wasserstein distance 代表 training 時的 loss，因此在 training WGAN 時紀錄了 discriminator 的 loss，每 iteration 一千筆時紀錄一次 Wasserstein distance。從圖中可以看到在 5000 epochs 以前，loss 是比較高的，然後在 10000~15000 之間 loss 震動幅度很大，不過在 15000 之後 loss 就變得比較穩定，同時 loss 也比較低，不過對照下面第 10000 iteration 的結果和 20000 iteration 的結果發現其實雖然在第 20000 iteration 時 loss 比較低不過和第 10000 次產生出來的圖像其實差不多，代表雖然 loss 有降，但不代表在視覺上人們所看到的圖像會變好很多，如教授在課堂上所講的。



我總共使用的訓練集為 1.5 萬筆圖片資料，總共 iteration 2 萬次。以下分別秀 iteration 1000 次、5000 次、10000 次、15000 次及 20000 次的結果。在 100 次的時候已經可以很清楚看出動畫圖像的臉，但是畫質、頭髮顏色及眼睛還不是很清楚，但基本上到 5000 次以後，只剩眼睛還畫得不是很好



#### ◆ Comparison Analysis. (1%)

以下圖片左邊為 DCGAN 的結果；右邊為加入 WGAN 技巧的 DCGAN 的結果。



從最原始的 DCGAN 到使用 WGAN 技巧的 DCGAN 結果上來看我會覺得最原本的 DCGAN 會比較好，因為大部分眼睛畫的是比 WGAN 好的，不過可以看出來 WGAN 畫出來的臉沒有整個毀容的，每個都是可以看到臉和

眼睛的，不像 DCGAN 會出現整個壞掉的，不過這項評斷標準很主觀，假設使用助教提供的人臉識別 model 來辨別人像的話，會發現 WGAN train 出來的頭像能成功從 64 個人臉中識別出 52 個臉，而使用 DCGAN 只從 64 個頭像中分辨出 48 個臉被識別出來，不過這個方法也不夠客觀，也許需要使用更客觀的方法來看要如何 evaluation 結果。

- Training tips for improvement. (6%)  
原始 model:

Generator

Layer (type)	Output Shape	Param #
Linear-1	[-1, 128]	12,928
ReLU-2	[-1, 128]	0
Linear-3	[-1, 256]	33,024
ReLU-4	[-1, 256]	0
Linear-5	[-1, 256]	65,792
ReLU-6	[-1, 256]	0
Linear-7	[-1, 256]	65,792
ReLU-8	[-1, 256]	0
Linear-9	[-1, 12288]	3,158,016
Tanh-10	[-1, 12288]	0
Total params: 3,335,552		
Trainable params: 3,335,552		
Non-trainable params: 0		

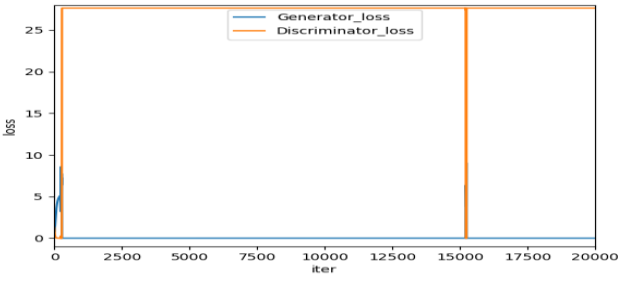
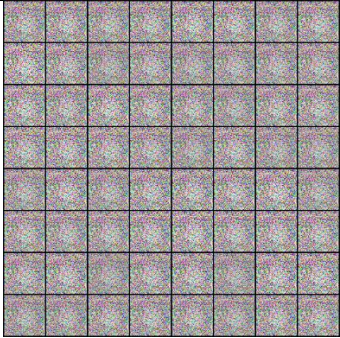
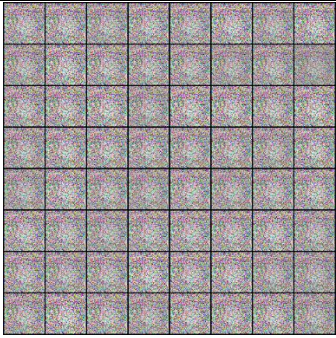
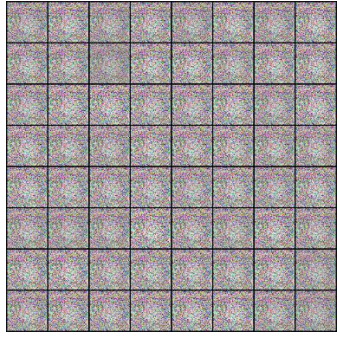
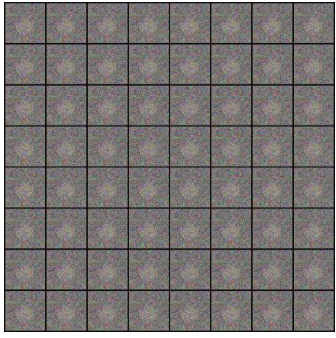
```
G_o(
  (main): Sequential(
    (0): Linear(in_features=100, out_features=128, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): ReLU(inplace)
    (4): Linear(in_features=256, out_features=256, bias=True)
    (5): ReLU(inplace)
    (6): Linear(in_features=256, out_features=256, bias=True)
    (7): ReLU(inplace)
    (8): Linear(in_features=256, out_features=12288, bias=True)
    (9): Tanh()
  )
)
```

Discriminator

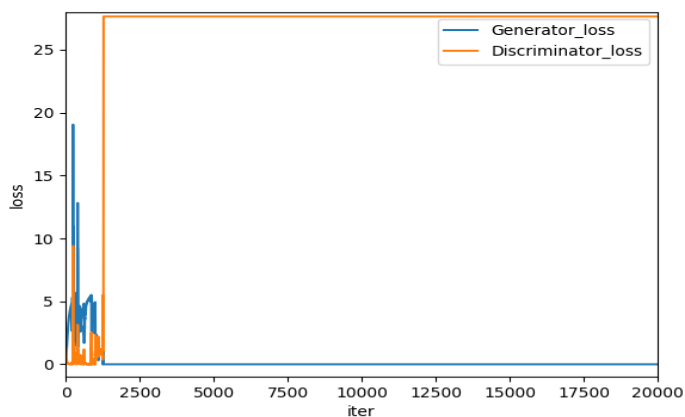
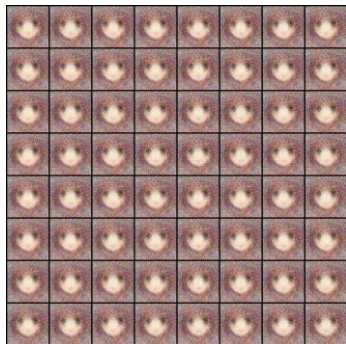
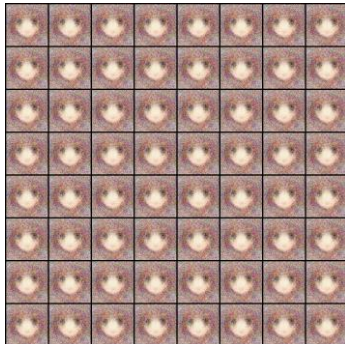
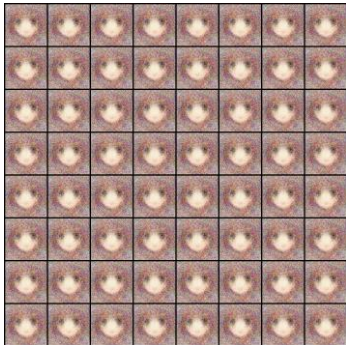
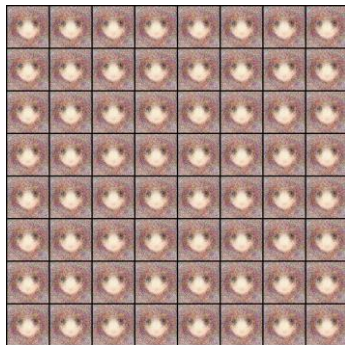
Layer (type)	Output Shape	Param #
Linear-1	[-1, 128]	1,572,992
ReLU-2	[-1, 128]	0
Linear-3	[-1, 64]	8,256
ReLU-4	[-1, 64]	0
Linear-5	[-1, 32]	2,080
ReLU-6	[-1, 32]	0
Linear-7	[-1, 16]	528
ReLU-8	[-1, 16]	0
Linear-9	[-1, 1]	17
Sigmoid-10	[-1, 1]	0
Total params: 1,583,873		
Trainable params: 1,583,873		
Non-trainable params: 0		

```
D_o(
  (main): Sequential(
    (0): Linear(in_features=12288, out_features=128, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU(inplace)
    (4): Linear(in_features=64, out_features=32, bias=True)
    (5): ReLU(inplace)
    (6): Linear(in_features=32, out_features=16, bias=True)
    (7): ReLU(inplace)
    (8): Linear(in_features=16, out_features=1, bias=True)
    (9): Sigmoid()
  )
)
```

Tips

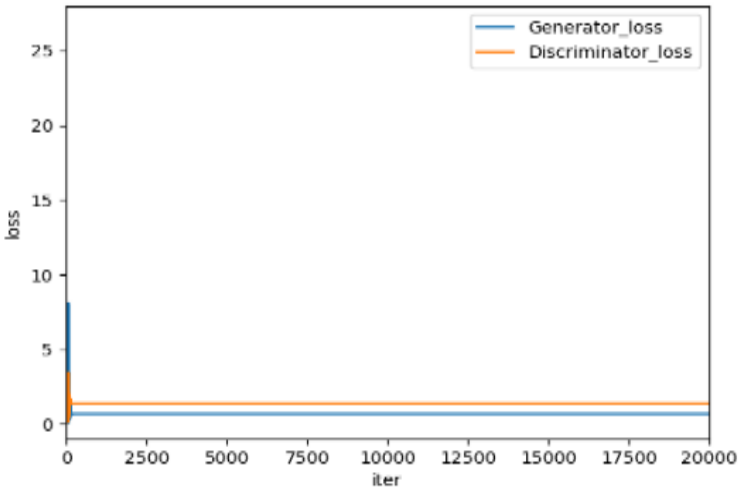
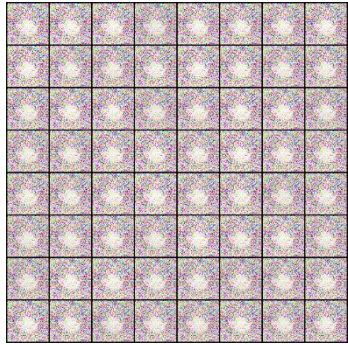
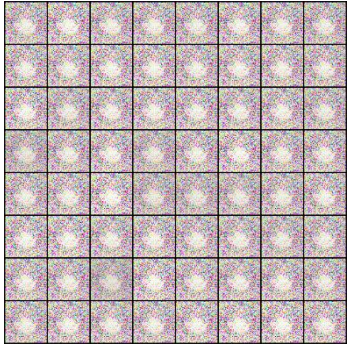
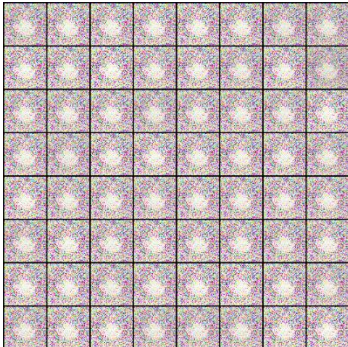
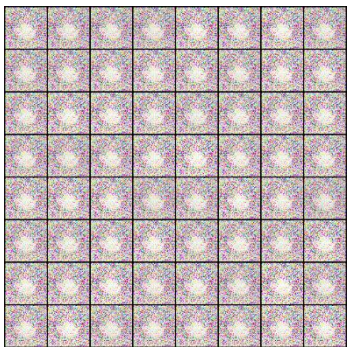
1. No use any tips (optimizer SGD with fully connected layer)			
Loss			
Results		1000iter	
		5000iter	
		10000iter	
		20000iter	



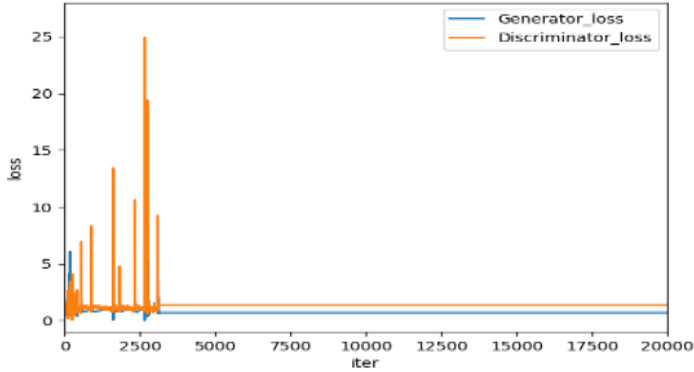
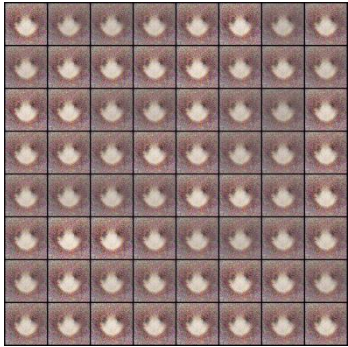
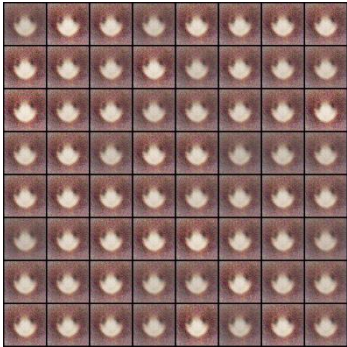
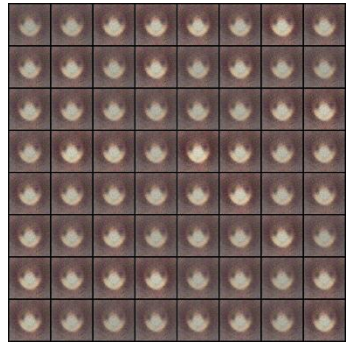
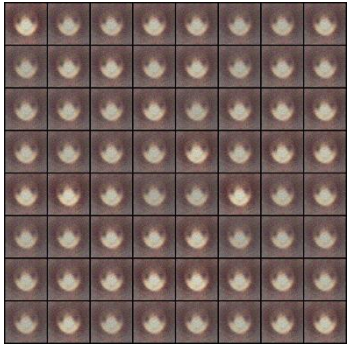
2. Tip1-Normalize the inputs (same as 1. But normalize inputs)	
Loss	
Results	 <p>1000iter</p>
	 <p>5000iter</p>
	 <p>10000iter</p>
	 <p>20000iter</p>

這邊稍微比較 1 與 2 的結果，可以發現有把 inputs normalize 可以讓 loss 不至於太早爆掉，但即使有做 normalize loss 還是在大約 1500 個 iteration 就會爆掉沒辦法在 train 下去了(這是有調整過 learning rate 的最好結果)，但 normalize 之後的結果至少看得出眼睛頭髮嘴巴在哪裡，雖然每一張臉都長的一樣，但我們可以說機器稍微學到了一些東西，而沒有做 normalize 的則是像電視壞掉的畫面一樣糟糕，如果沒有告訴我那是臉我是怎麼樣都看不出來那是臉的，同時幾乎是無法 train 起來的，所以我們可以說 normalize inputs 實在是，行！

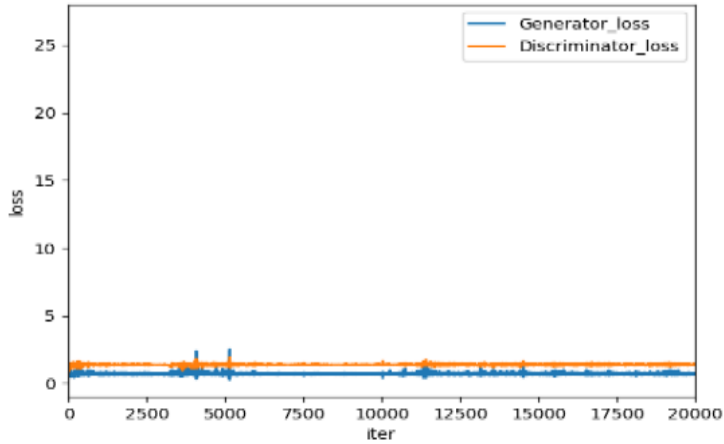
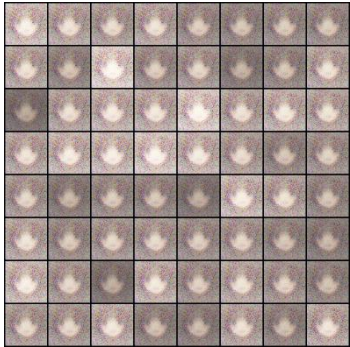
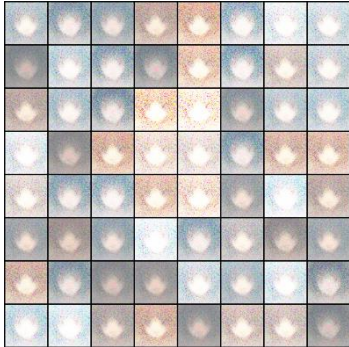
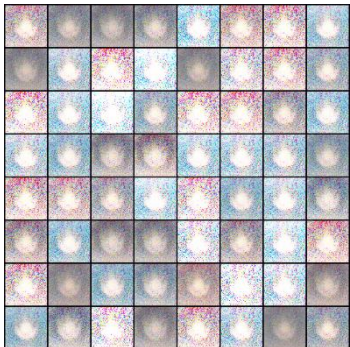
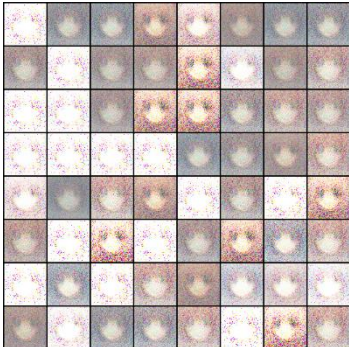


Tip9-Adam(on generator) (but no normaize inputs)		
Loss		
Results	 <b>1000iter</b>	 <b>5000iter</b>
	 <b>10000iter</b>	 <b>20000iter</b>

由於 Tip-9 裡面寫說 Use SGD for discriminator and ADAM for generator，就稍微測試了一下只把 generator 的 optimizer 換成 adam，而在沒有 normalize inputs 的情況下，loss 還是很不爭氣的一下就爆掉了，而生成出來的圖像也是壞掉的電視畫面。

Tip9-Adam(on generator) + Tip1-Normalize inputs	
Loss	
Results	<div data-bbox="459 692 807 1037">  </div> <div data-bbox="571 1061 694 1097">1000iter</div>
	<div data-bbox="916 692 1264 1037">  </div> <div data-bbox="1027 1061 1149 1097">5000iter</div>
	<div data-bbox="459 1126 807 1469">  </div> <div data-bbox="564 1494 702 1529">10000iter</div>
	<div data-bbox="916 1126 1264 1469">  </div> <div data-bbox="1019 1494 1157 1529">20000iter</div>

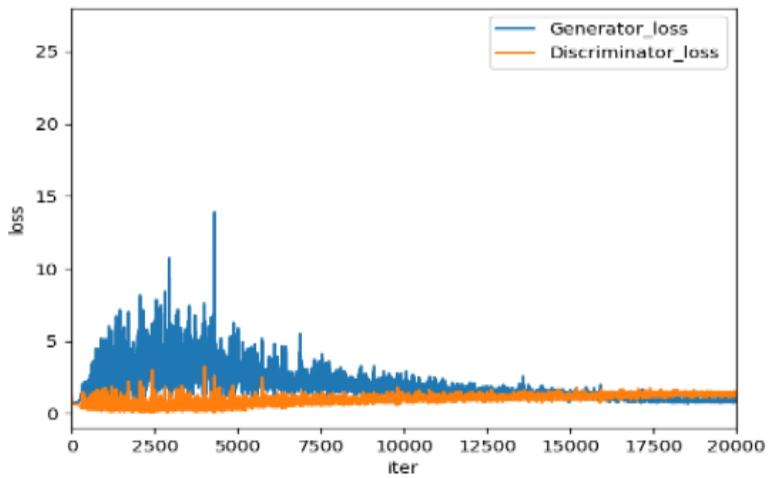
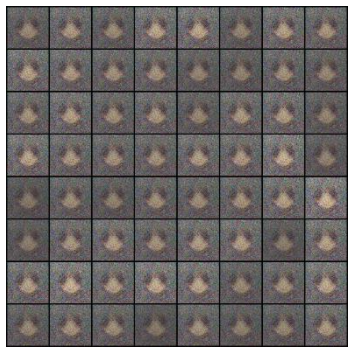
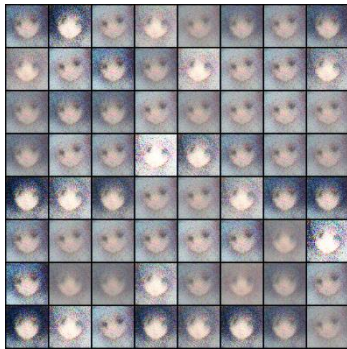


同時使用 adam(只有 generator 用 adam)和 normalize inputs 的結果稍比只有 normalize inputs 的結果好一些，圖片的解析度看起來比較高，並且 generator 似乎想要製造顏色深淺的變化，但並不是很明顯，而從 loss 來看，終於撐超過 2500 個 iteration 才爆掉，可謂之 training 的大躍進阿，所以說 adam 實在是，行！

Tip9-Adam(on G and D but no normalize)	
Loss	
Results	 <p>1000iter</p>
	 <p>5000iter</p>
	 <p>10000iter</p>
	 <p>20000iter</p>

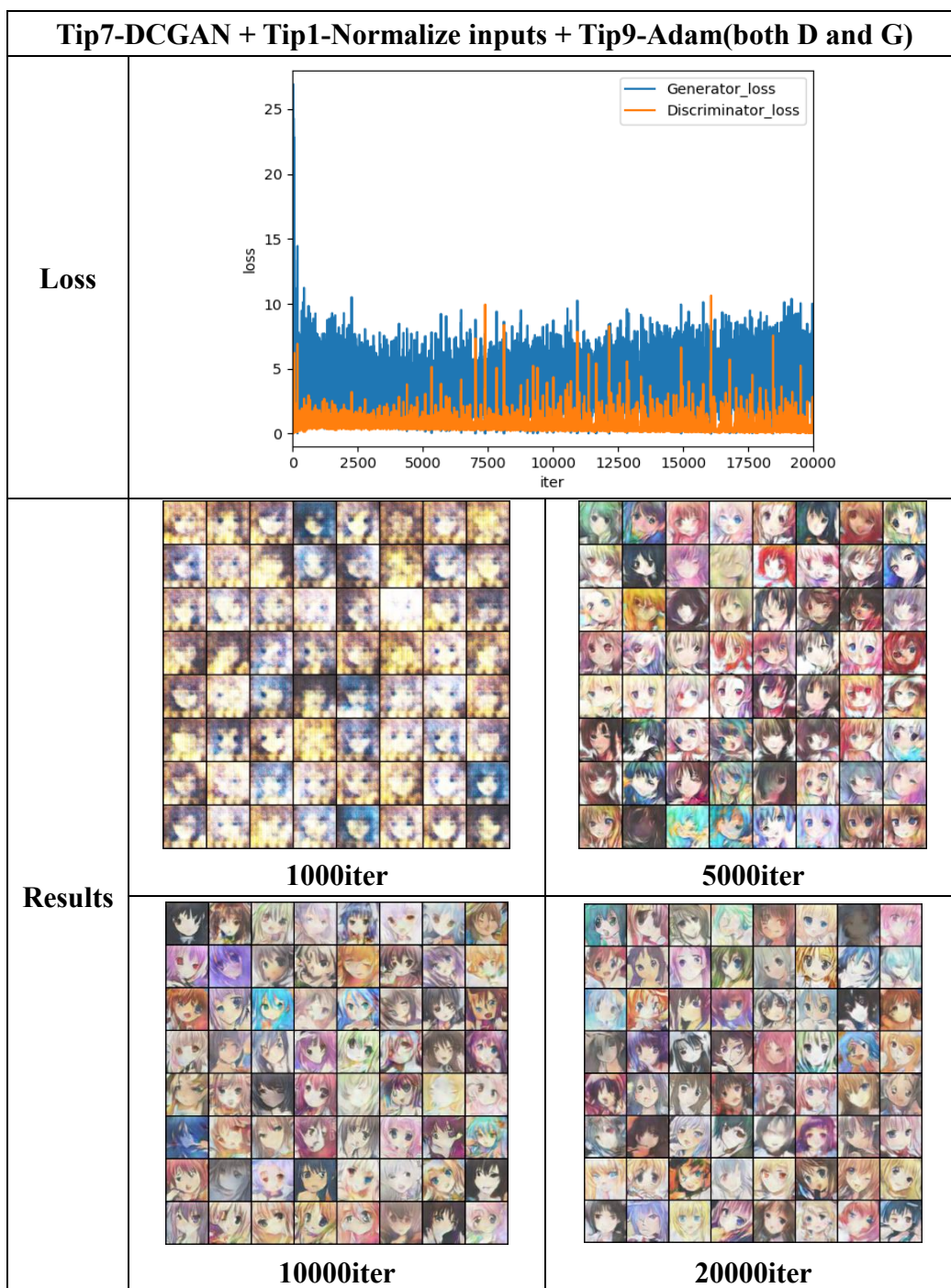
原本在 training 之前想說，兩個都用 Adam 應該會讓結果變差吧？

但沒有想到不僅 loss 看起來沒有明顯的爆掉的傾向、結果變異度變大，還讓在沒有 normalize inputs 的情況下出現能辨識的臉，所以我們偉大的馬前總統說的對，一個 adam 沒有用...你有沒有用兩個？



Tip9-Adam(on both G and D) + Tip1-Normalize inputs	
Loss	
Results	<div data-bbox="459 786 807 1133" data-label="Image">  </div> <p>1000iter</p> <div data-bbox="914 786 1262 1133" data-label="Image">  </div> <p>5000iter</p>
	<div data-bbox="459 1220 807 1565" data-label="Image">  </div> <p>10000iter</p> <div data-bbox="914 1220 1262 1565" data-label="Image">  </div> <p>20000iter</p>

用了兩個 adam 之後再加上 normalize inputs 顯然讓結果變好很多，而在 loss 上面也可以看到一些特出的變化，可以看到 generator 的 loss 隨著 training 的過程慢慢穩定了下來，並且有個下降的趨勢；而 discriminator 也是逐漸穩定，但有個上升的趨勢，不知道是否有特殊的解釋，可能隨著繼續 training 可以看到他的 loss 互相交叉很多次，或許是兩個在互相對抗，但可以看到結果是非常不錯的。



由於如果要在一個一個比較，會使得結果比較過於冗長，相信助教大概也覺得簡短就好，所以最後一個就直接開大招，全部一起用。

首先可以看到 D 跟 G 的 loss 跳動幅度都非常的大，或許是兩個在很有效率的對抗才會造成她跳動的比之前都大(也有可能是 cnn 的性質?)，而在結果上面可以看到大概 1000 個 iteration 之後就可以說產生比之前全部都好的結果了，DCGAN 實在厲害，最後終於可以產生出騙過媽媽說這是我畫的動漫人物的結果，但其實他會嘴我我根本不會畫畫 QAQ。

# Style Transfer

模型使用 StarGAN : <https://github.com/yunjey/StarGAN>

## 1. Result + 2. Analysis

原始結果:



原圖 黑髮 金髮 棕髮 男性化 變老



套用在同學上的結果:

(1) 不調參數直接套用:



原圖

黑髮

金髮

棕髮

變老

女性化

可以看到大概就像拿筆刷隨便亂畫一樣，整張照片可以說是壞掉了，但還是保留了原本的性質，例如黑髮就會在照片上塗很多黑色、加強對比，但是連衣服上面或是背景上都會被塗黑；而變老看起來是最不錯的，每個可辨識的臉上都被畫滿了皺紋；女性化似乎會讓下巴變尖、皮膚變平滑。

而在多人的照片或是臉部不好辨識的照片顯然會直接壞掉。

(2) 微調參數:



調整參數過後的結果讓照片看起來不會崩壞了，但是性質變得不太明顯，其中最能看出改變的大概是黑髮(第二列)把整個頭髮都塗黑，但要有染髮的朋友才會比較能看出差別；及變老(倒數第二列)，大家都變成宇智波鼬，而棕髮(第四列)似乎只是把對比調到接近紅色而已，其他大概看不出在幹嘛。

分工表:

3-1, 3-2 code 部分及 report: 羅章碩

3-1 tip, 3-3 style transfer 及 report: 鄒適文