



Deep Learning (Homework 3)



Due date : 6/16/2019

1. Cartoon Character Generation

In this exercise, you will construct a **Variational Autoencoder (VAE)** for image reconstruction by the provided animation faces dataset.

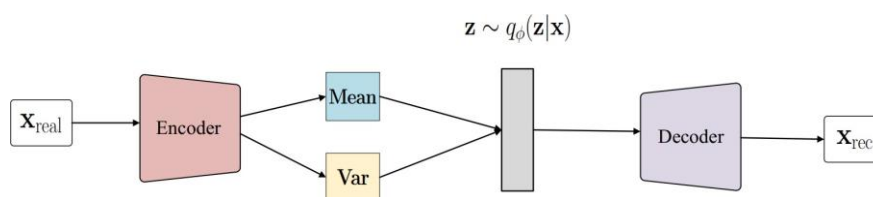


Figure 1: Structure of VAE

- Describe** in details how to **preprocess images** (such as resize). Implement a VAE for image reconstruction by using convolution layers or fully connection layers. You need to **design** the network architecture and show it in the report. Finally, plot the **learning curve** in terms of loss function or negative evidence lower bound.

前處理：

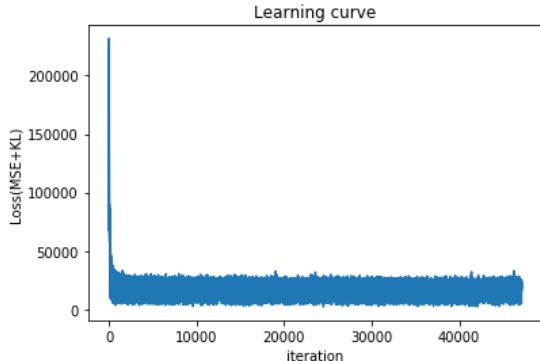
前次作業(作業二)在資料前處理的時候使用ImageFolder，但是因為此套件的設定是需要在多個資料夾下才可以讀取檔案的(此套件讀取方式：將讀取進來時的資料夾作為該圖片label)，因此這次嘗試自己寫一個**專屬dataset**。

首先寫一個小function將資料夾內所有檔案位置記錄成一份名為cartoon的txt，後續製作CartoonDataset的時候，依據將圖片位置讀取後以loader讀取圖片。讀取的格式是將圖片Resize成64*64，做Normalize，將圖片以Batch_size=64讀取。

網路架構：

由於是圖片處理，因此選用Conv2D來建構網路，因應使用Conv2D所以使用ReLU作為 Activation function，Encoder的最後一層做Flatten，而Decoder則反向設計，Decoder的最後一層是sigmoid來判斷是否還原得像，因為最後一層是sigmoid所以寫了兩種Loss，一種是Binary cross entropy 另外一種是將整張圖片每個值去計算MSE，無論是哪一種最後都會再加上KL的Loss。

本實驗網路參數，Batch size = 256，latent code dimension : (64, 512)，Learning rate = 1e-3，epoch = 300。

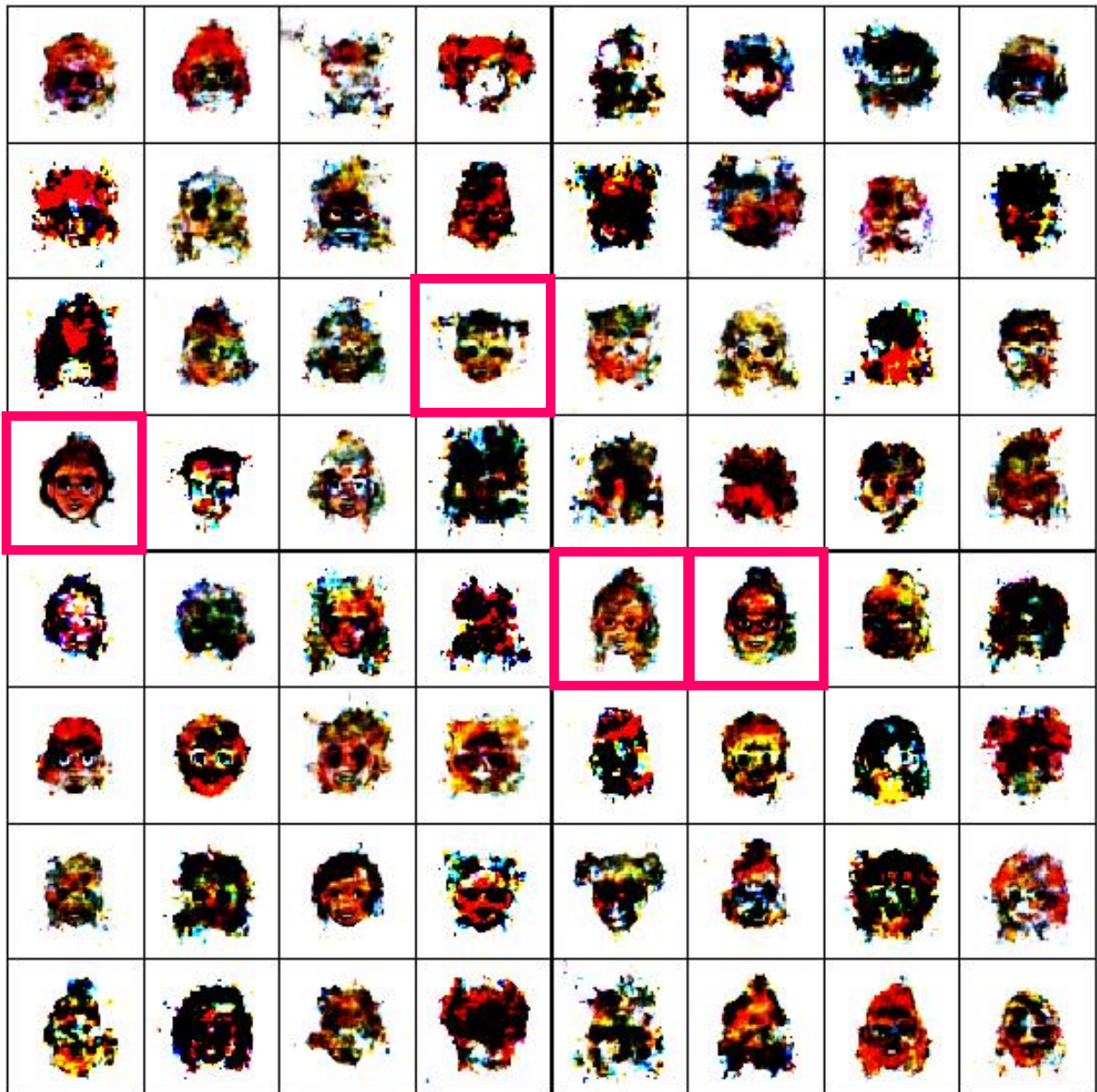
網路架構	<pre> VAE((encoder): Sequential((0): Conv2d(3, 16, kernel_size=(4, 4), stride=(2, 2)) (1): ReLU() (2): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2)) (3): ReLU() (4): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2)) (5): ReLU() (6): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2)) (7): ReLU() (8): Flatten()) (fc1): Linear(in_features=512, out_features=64, bias=True) (fc2): Linear(in_features=512, out_features=64, bias=True) (fc3): Linear(in_features=64, out_features=512, bias=True) (decoder): Sequential((0): UnFlatten() (1): ConvTranspose2d(512, 64, kernel_size=(5, 5), stride=(2, 2)) (2): ReLU() (3): ConvTranspose2d(64, 32, kernel_size=(5, 5), stride=(2, 2)) (4): ReLU() (5): ConvTranspose2d(32, 16, kernel_size=(6, 6), stride=(2, 2)) (6): ReLU() (7): ConvTranspose2d(16, 3, kernel_size=(6, 6), stride=(2, 2)) (8): Sigmoid())) </pre>
Learning curve	 <p>The figure is a line plot titled "Learning curve". The y-axis is labeled "Loss(MSE+KL)" and ranges from 0 to 200,000 in increments of 50,000. The x-axis is labeled "iteration" and ranges from 0 to 40,000 in increments of 10,000. The plot shows a blue line representing the loss over time. The loss starts at a high value of approximately 200,000 at iteration 0. It drops very rapidly, reaching a value of about 20,000 by iteration 10,000. From iteration 10,000 to 45,000, the loss remains relatively stable, fluctuating between approximately 15,000 and 25,000, indicating that the model has converged.</p>

2. Show some examples reconstructed by your model.



- i. Sample the prior $p(\mathbf{z})$ to generate some examples when your model is well-trained with convergence.

把Train出來比較好的結果用紅色框圈起來。



3. Style Transfer

In this exercise, you will implement a **cycleGAN** to transfer the American cartoon character into the Japanese animation style, and vice versa. There are 10000 images in both folder “cartoon” and folder “animation”. CycleGAN paper can be downloaded [here](#). The example pytorch codes are provided as `cyclegan train.py` and `cyclegan test.py`. You only need to fill up **TODO** parts we marked in the code.

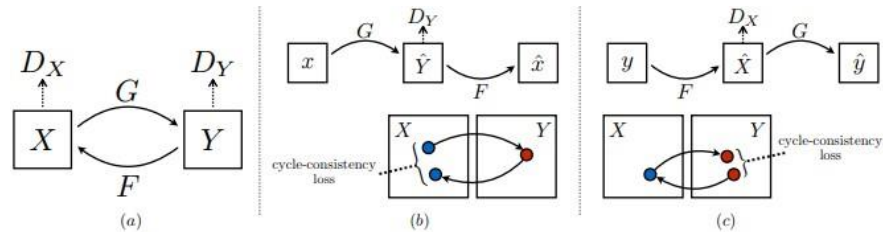
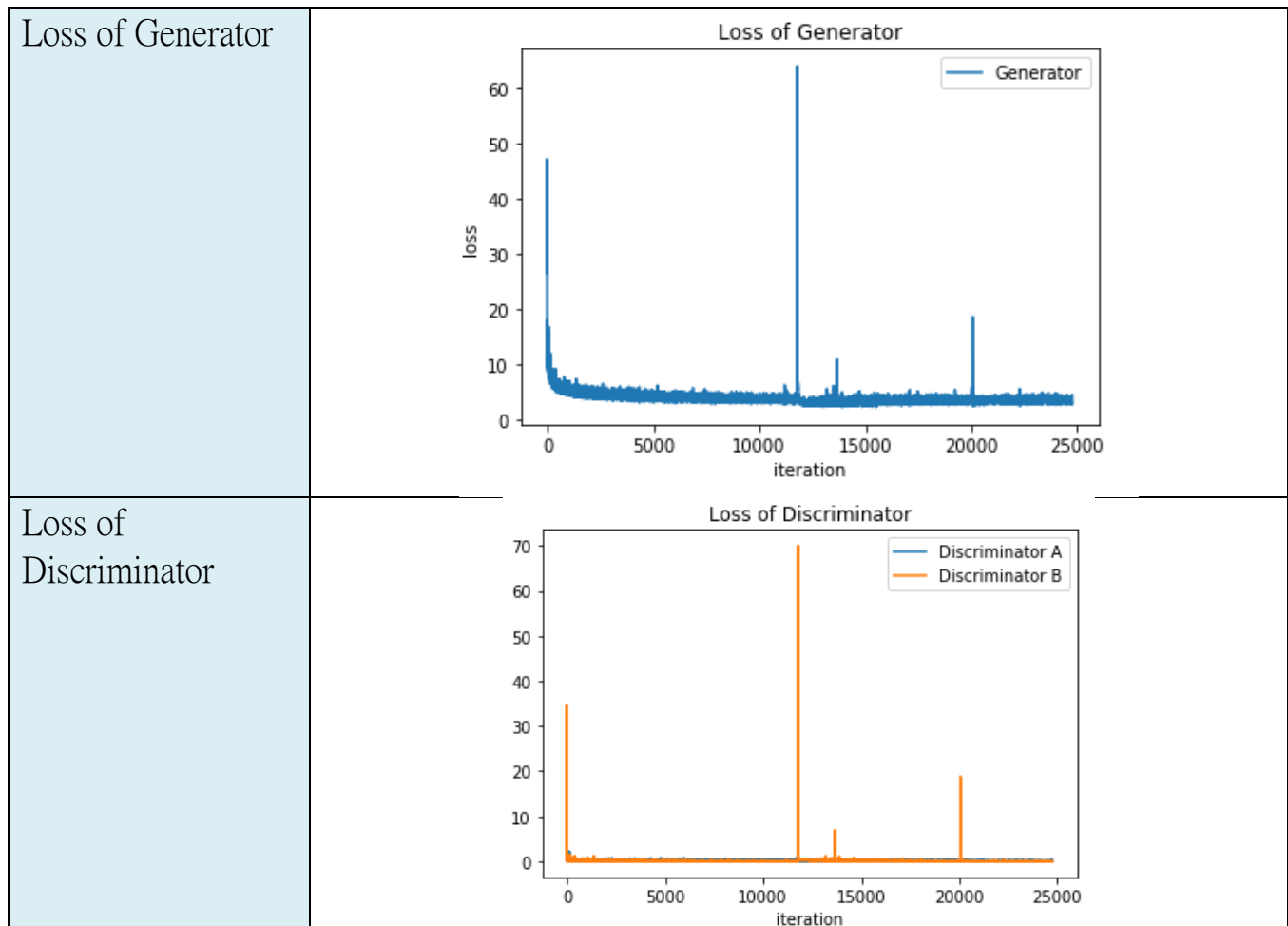
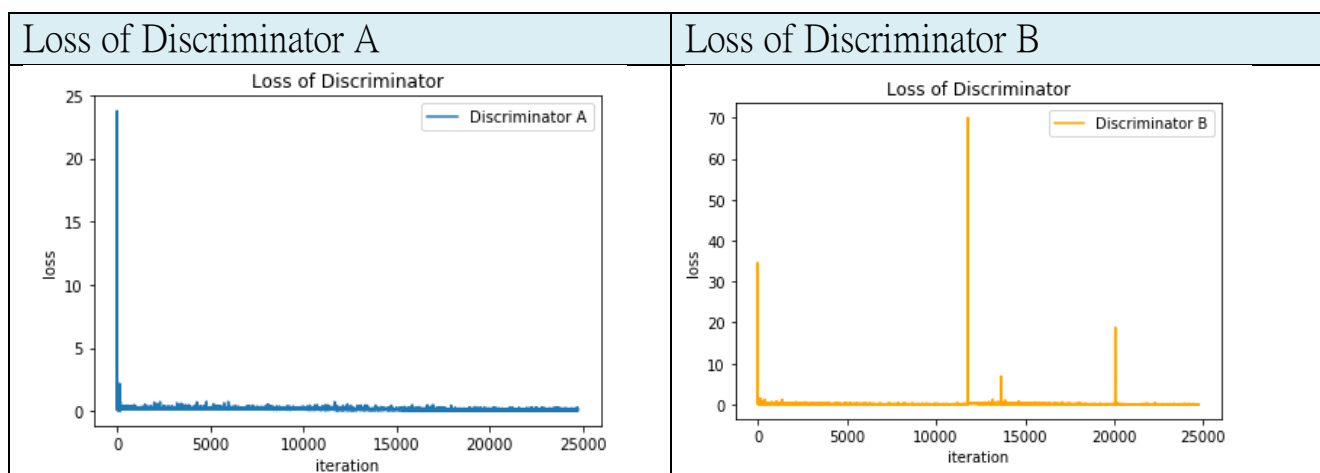


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X . D_Y encourages G to translate X into outputs indistinguishable from domain Y , and vice versa for D_X and F . To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

Figure 5: Structure of cycleGAN

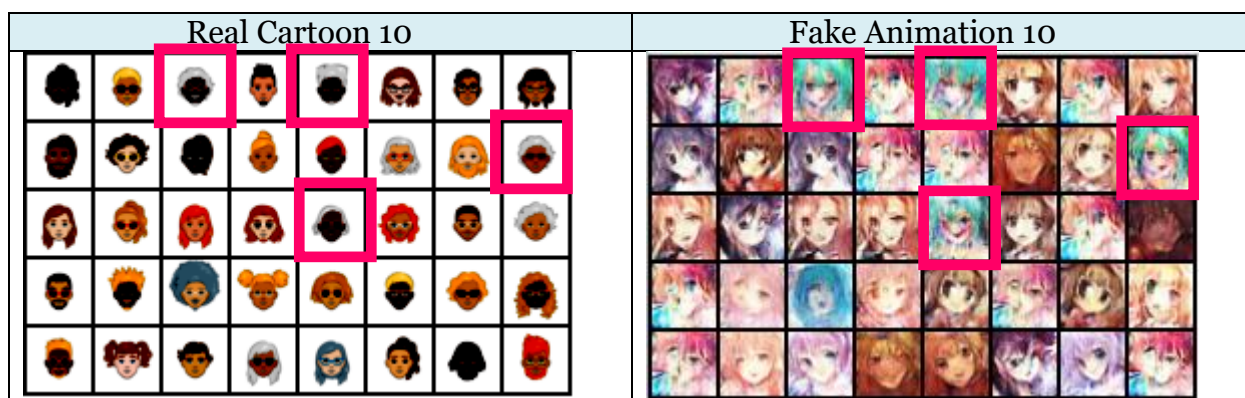
- Construct a cycleGAN with the loss function below. Plot the learning curve of both generators and discriminators. You can sum up the loss of two generators and plot in one curve.





- ii. Please sample some cartoon images in animation style and animation images in cartoon style. Show your results and make some discussion in the report.

將兩Domain間風格互換的結果表格，放置於本文件的最後兩頁，這裡只展示出欲討論的圖片。下方表示由Cartoon Domain轉換到 Animation Domain的結果，如果依照頭髮的顏色來看，Model 好像沒有完全依照餵入的資訊做出相對應的Animation圖，例如第一張圖片，黑臉黑頭髮，但在右邊生成出來卻是白臉紫頭髮，不過可以看到Real Cartoon 10 與 Fake Animation 10 的組合中，在 **cartoon domain** 中白色頭髮且黑色臉的到 Animation Domain 中都會變成藍色頭髮白色臉的模樣(以桃紅色框框，框起來的部分)，可以看出Model是有學到兩個Domain之間的畫圖技巧跟某些轉換方式的，只是可能因為training 次數不夠多，所以細節轉換的效果有限。



想特別比較一下Real Cartoon 5到 Fake Animation 5的轉換。由下圖兩條當中可以看到藍框的部分，有將左邊原始圖片的髮色參考至Animation Domain去做生成，只是可能因為圖片的size開很小，能夠學到的原圖上的特徵不夠精細。



iii. Briefly describe what is mode collapse. According to (ii), is mode collapse issue serious in this task? Why?

mode collapse 是指當Generator發現生成某個樣本點很容易騙過 Discriminator 的時候，Generator 會生成很多類似該樣本點的分佈，導致生成出來的結果都長很像，但 loss 卻已經下降。通常解決 mode collapse 的方法就是做 batch normalization 讓 sample 的點不要太偏。在本次實驗當中的確有某些臉很常被生成，感覺像是某幾個固定

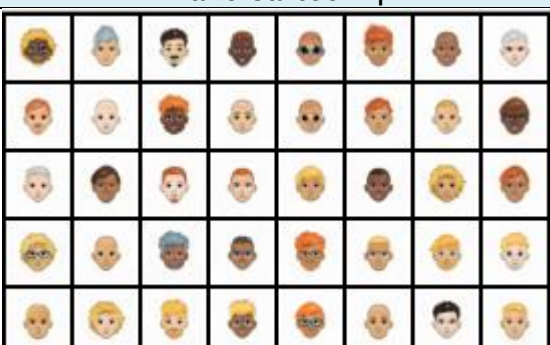
班底一直出現，例如：，這五張圖片出現次數非常高。但可能因為Batch size比較大，一次生成40張圖片，所以一眼望去比較沒有mode collapse的問題(有看到其他同學把batch size設為10的時候mode collapse蠻嚴重的)。


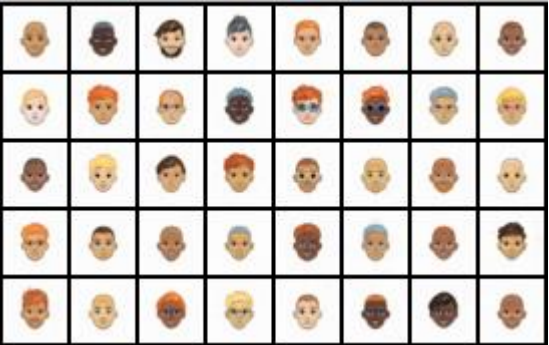

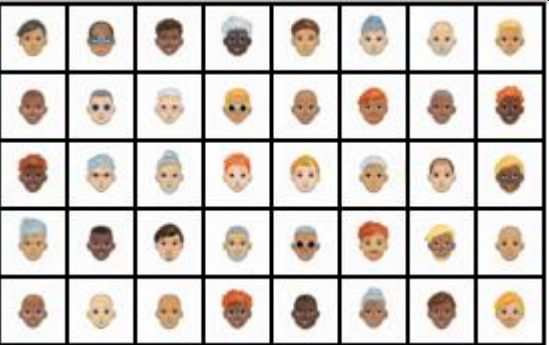

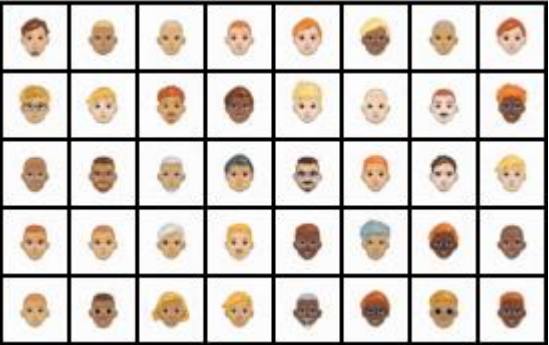

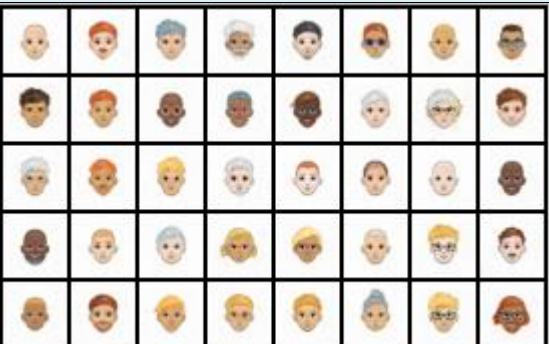

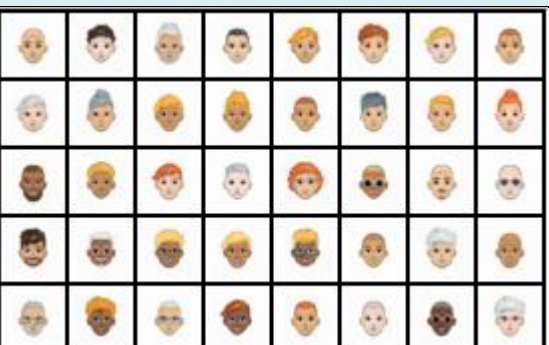
另外，學生認為本實驗比較關鍵沒有 train 好的原因是 model 似乎沒有在乎 Input 的資訊是什麼，以至於雖然生成很像另一個 domain 的圖片，卻沒有將原圖的細節：例如，臉的顏色、頭髮的顏色考慮進來畫。另一種可能是因為，loss 的權重並沒有調整好，導致出來結果長這樣。GAN真的不是隨便想train就可以成功的程式呢。

後兩頁為CycleGAN實驗當中左有的比較圖，總共有兩張大表格，每張大表格有兩欄位比較圖。

第二題第二小題所有結果的比較，完整表格如下所示：

Animation to Cartoon (right : Real Animation , left : Fake Cartoon)

Real Animation 1	Fake Cartoon 1
	
Real Animation 2	Fake Cartoon 2
	
Real Animation 3	Fake Cartoon 3
	
Real Animation 4	Fake Cartoon 4
	
Real Animation 5	Fake Cartoon 5
	

Real Animation 6	Fake Cartoon 6
	
Real Animation 7	Fake Cartoon 7
	
Real Animation 8	Fake Cartoon 8
	
Real Animation 9	Fake Cartoon 9
	
Real Animation 10	Fake Cartoon 10
	

Cartoon to Animation (right : Real Cartoon · left : Fake Animation)

Real Cartoon 1	Fake Animation 1
	
Real Cartoon 2	Fake Animation 2
	
Real Cartoon 3	Fake Animation 3
	
Real Cartoon 4	Fake Animation 4
	
Real Cartoon 5	Fake Animation 5
	

Real Cartoon 6



Fake Animation 6



Real Cartoon 7



Fake Animation 7



Real Cartoon 8



Fake Animation 8



Real Cartoon 9



Fake Animation 9



Real Cartoon 10



Fake Animation 10

