基於自動生成特徵之條件生成對抗網路臉部合成之研究

Features Automatic Generated Conditional Generation Adversarial Network for Facial Synthesis

介面呈現:

- **功能一:臉部合成**
  - ✓ face_swape
- **功能二:選擇訓練模型生成人臉**
  - ✓ pix2pix_GAN
  - ✓ CycleGAN
  - ✓ StyleGAN(程式碼測試中可以嘗試操作)
- **功能三:選擇模型臉部上妝**
  - ✓ BiSeNet
  - ✓ Dlib
- **功能四:臉部圖像分析**
  - ✓ style_transfer
  - ✓ face_similarity

內容

名稱： 原始圖像

說明： 原始圖像

說明： 訓練不同類型圖像生成後與原始圖像做比對差異

新增圖片

圖片 選擇檔案 未選擇任何檔案 標題：

功能一:臉部合成

○ face_swape

功能二:選擇訓練模型生成人臉

○ pix2pix_GAN ○ CycleGAN ○ StyleGAN(程式碼測試中可以嘗試操作)

CycleGAN請選擇你生成的條件

○ 動漫人物生成真實人臉 ○ 真實人臉生成動漫人物

功能三:選擇模型臉部上妝

○ BiSeNet ○ Dlib

眉毛顏色

紅色R:

功能三:選擇模型臉部上妝

○ BiSeNet ○ Dlib

眉毛顏色

紅色R:

68

綠色G:

54

藍色B:

39

嘴唇顏色

紅色R:

150

綠色G:

0

藍色B:

0

眼睛顏色

紅色R:

255

綠色G:

255

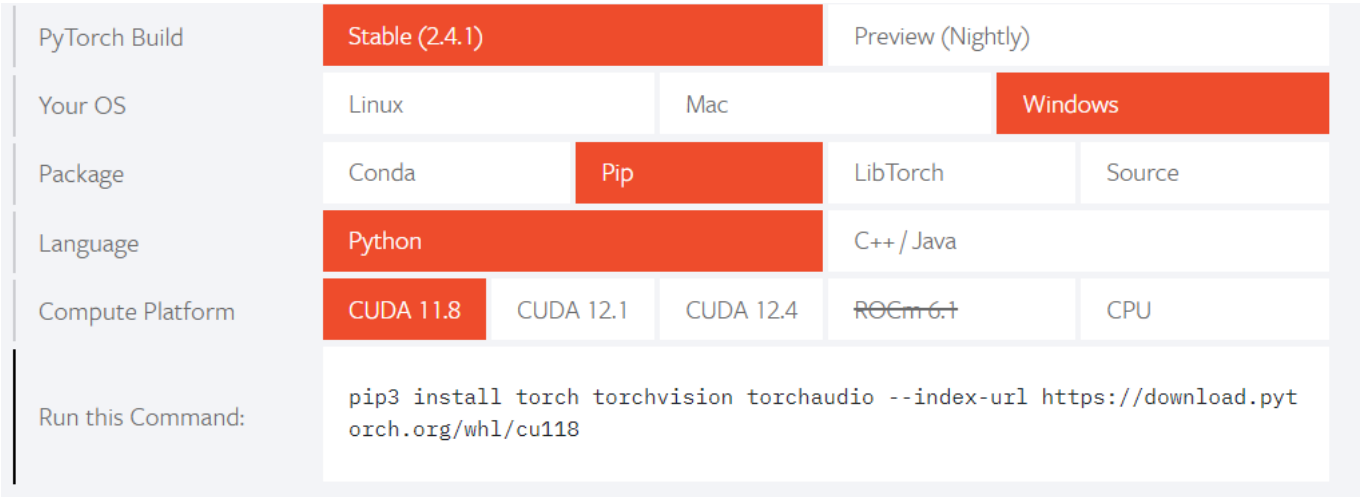
藍色B:

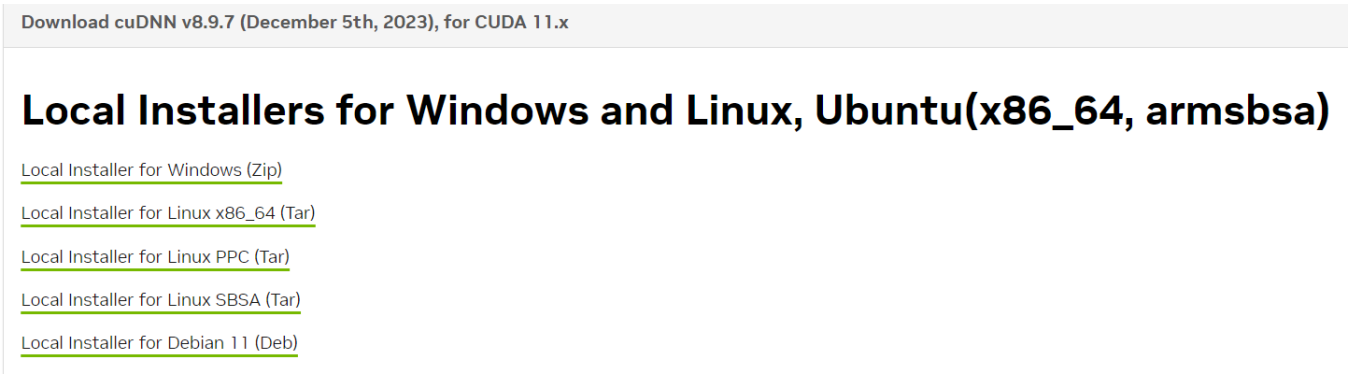
255

功能四:臉部圖像分析

本系統環境[生成式網路應用 GAN]

本系統環境 tensorflow 2.17.0 版本、python 3.12.3、PyTorch 2.4.1、NVIDIA CUDA 11.8、NVIDIA cuDNN NVIDIA 8.9.7 GeForce RTX 3060。





face_swape

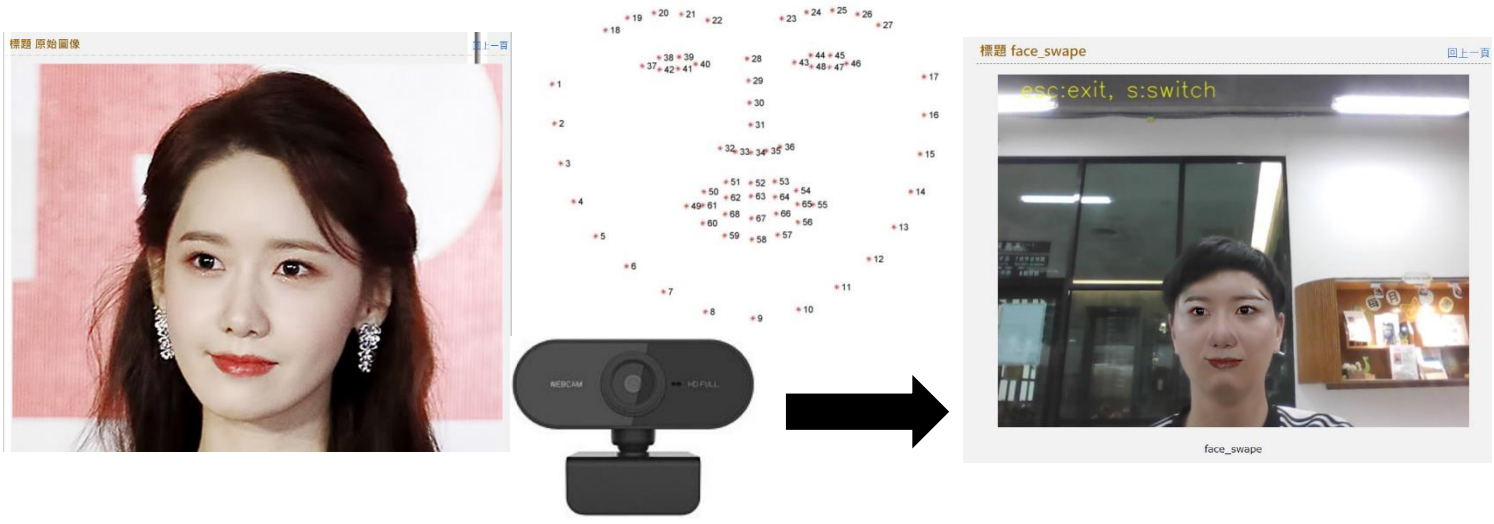
使用 opencv-python 和 dlib 實現的簡單換臉程式

- ✓ 使用 dlib 的 shape_predictor_68_face_landmarks.dat 模型取得人臉圖片 im1 和相機圖片 im2 的 68 個人臉特徵點。

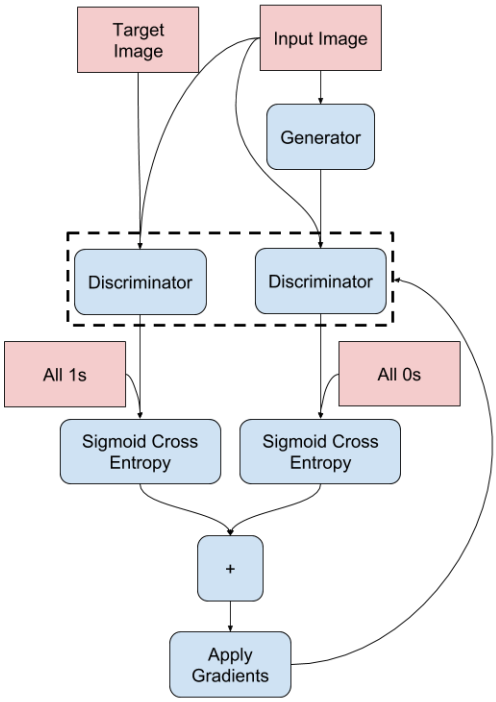
- ✓ 根據上一步驟所獲得的特徵點得到兩張圖片的人臉掩模 im1_mask 和 im2_mask。
- ✓ 利用 68 個特徵點中的 3 個特徵點，對人臉圖片 im1 進行仿射變換使其臉部對準相機圖片中的臉部，得到圖片 affine_im1。

- ✓ 對人臉圖片的掩模 im1_mask 也進行相同的仿射變換得到 affine_im1_mask。
- ✓ 對掩模 im2_mask 和掩模 affine_im1_mask 的掩蓋部分取併集得到 union_mask。

利用 opencv 裡的 seamlessClone 函數對仿射變換後的 affine_im1 與相機圖片 im2 進行泊松融合，掩模為 union_mask，得到融合後的影像 seamless_im。

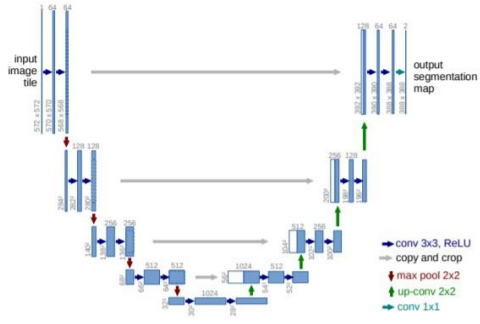


pix2pix_GAN

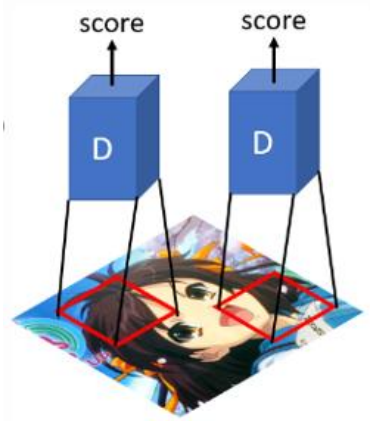


pix2pix 架構的生成器以 U-Net 架構建置，pix2pix 鑑別器網絡採用 PatchGAN ，使用的 PatchGAN 的鑑別器為了能更好地對圖像的局部做判斷，pix2pix 鑑別器採用 PatchGAN 結構，也就是說把圖像等分成多個固定大小的 Patch，分別判斷每個 Patch 的真假， 最後再取平均值作為鑑別器最後的輸出。論文裡有提出的這個 PatchGAN 可以看成另一種形式的紋理損失或樣式損失，在具體實驗時，不同尺寸的 patch，最後發現 70*70 的尺寸比較合適。

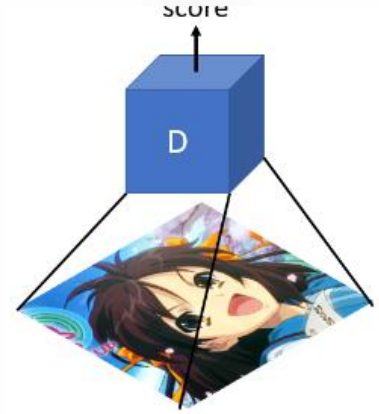
在圖片尺寸稍大的狀況下，如果用整張圖片丟進鑑別器來做檢查很容易導致訓練時間過長、Overfitting 、performance 不好等狀況，因此 PatchGAN 便只檢查整張圖片的 patch。



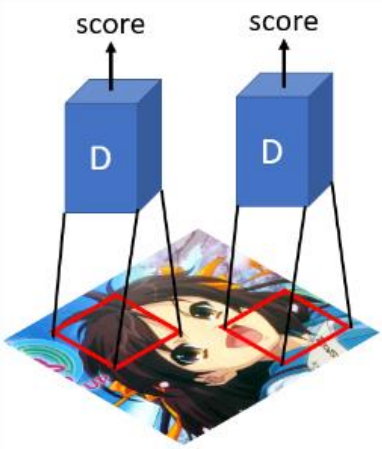
Generator(生成器)

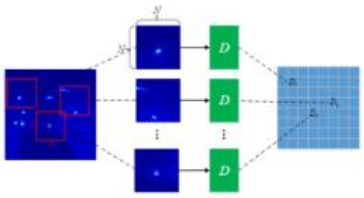


Discriminator(判別器)

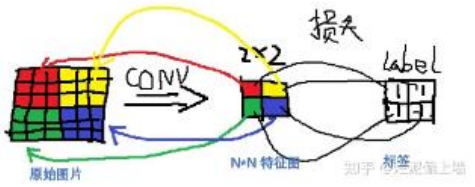


一般GAN
圖片尺寸大時，整張圖片丟進 D
做檢查很易導致訓練時間過長、Overfitting 、performance 不好等狀況
* 最後一層為 Dense layer，輸出只有一個值 [0, 1]，用該值判斷"這整張圖"是否真假





Patch GAN
採取整張 圖片一小部分 **(patch)** 做檢查，
而output的tensor包含多個Patch的機率，
再取平均得到分類結果
損失函數：
據風格判斷(因只取圖片的局部內容，
無法用內容判斷loss)



新增圖片

圖片 選擇檔案 t19.jpg　　標題： apple

功能一:臉部合成

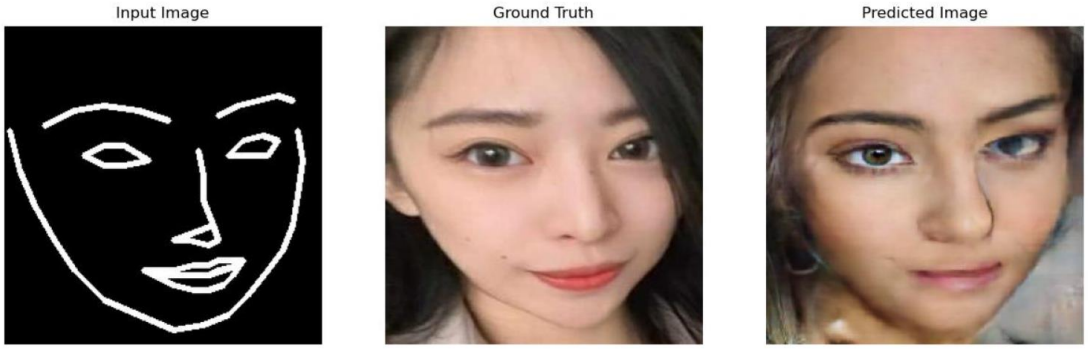
face_swape

功能二:選擇訓練模型生成人臉

pix2pix_GAN　CycleGAN　StyleGAN(程式碼測試中可以嘗試操作)

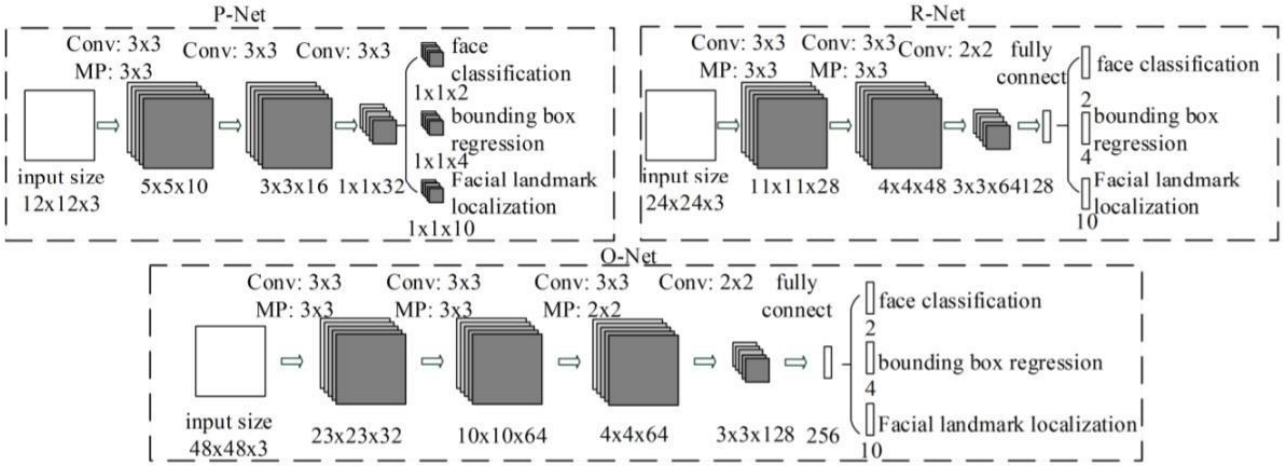
透過你的臉形生成外國人的樣子，首先上傳你的照片

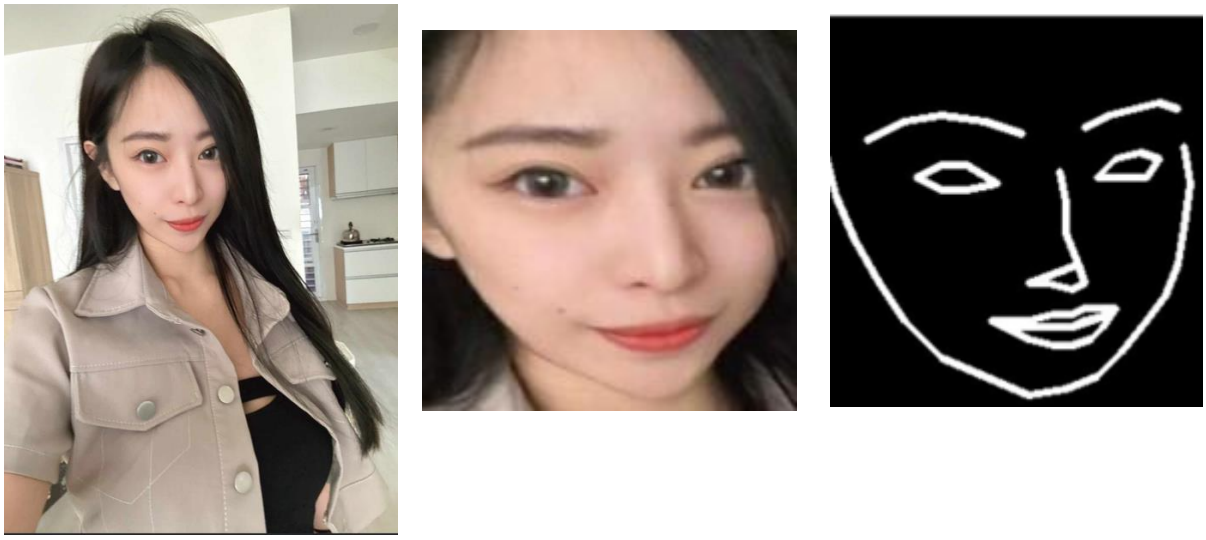
2000 筆訓練資料與 500 測試資料訓練 100 epoch



※　臉部切割範圍

這裡使用 MTCNN 裁切臉部範圍，特徵輪廓是使用 DLIB68 個特徵點連成





CycleGAN

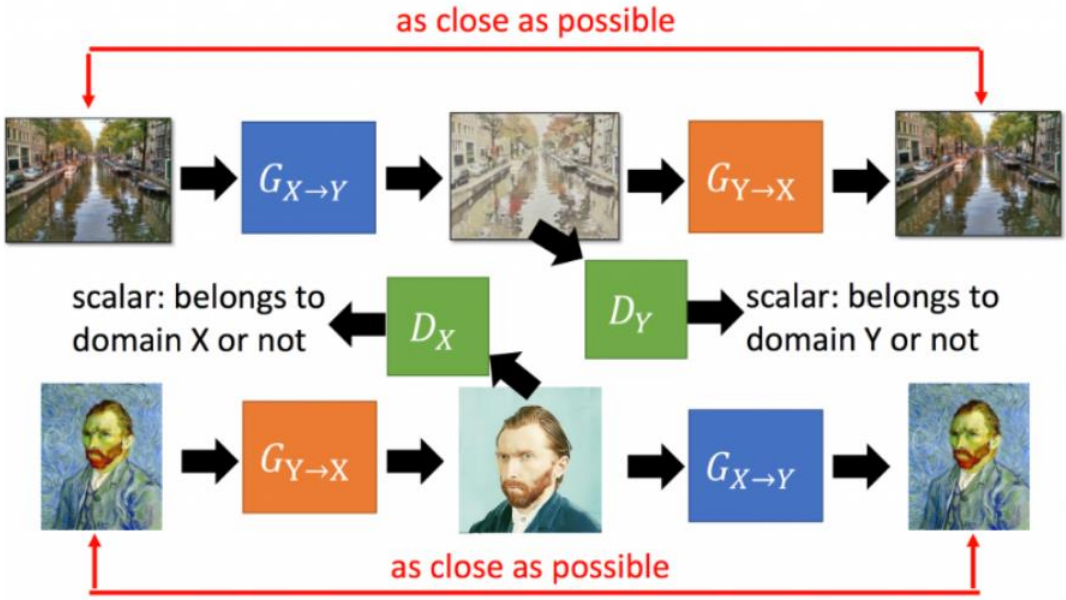
背景知識 - CycleGAN

1.目的：

拋棄掉 Paired Data 的限制，只需要兩種 Domain 的圖片便能做到 Cross Domain Image 轉換

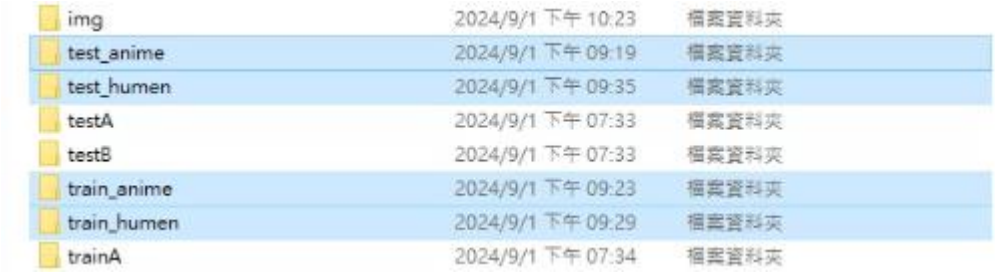
2.方法：

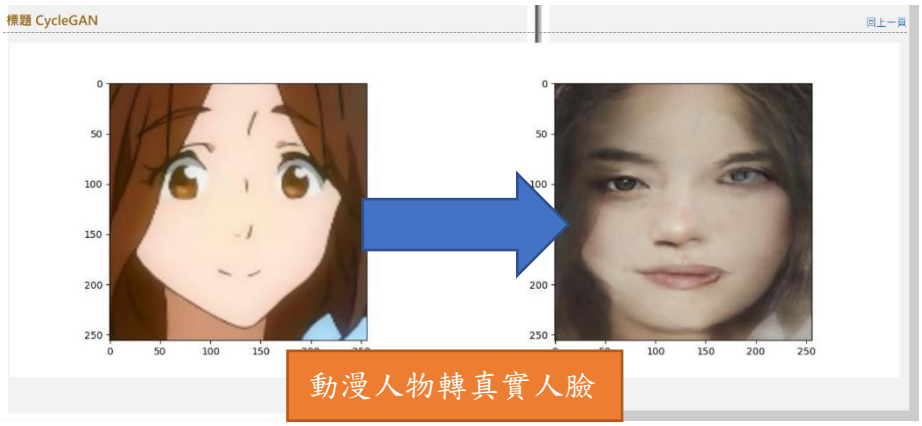
（以下圖片皆擷取自李宏毅老師的授課影片）先從 Figure 2 左圖開始，先利用 Domain Y 圖片訓練出一個 Dy，Dy 能辨識出圖片是否屬於 Domain Y．藉著 Dy 我們便能訓練出從 X 轉到 Y 的 Generator Gxy. 但如果只做這樣，Gxy 可能會不停的產生 Domain Y 的圖片來騙過 Dy，而非產生 Domain X 轉到 Y 的結果，因此作者引入 Cycle Consistency 的概念：如右圖 X → Gxy → Y → Gyx → X’，X 必須與 X’ 一致，藉此我們便能做到 X→ Y. 右圖下半也是相同的概念，做的是 Y→ X 的轉換

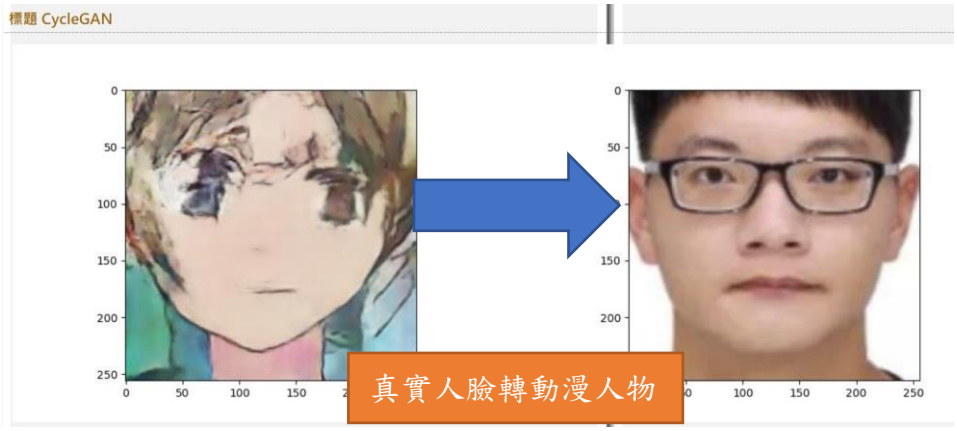


介面呈現：

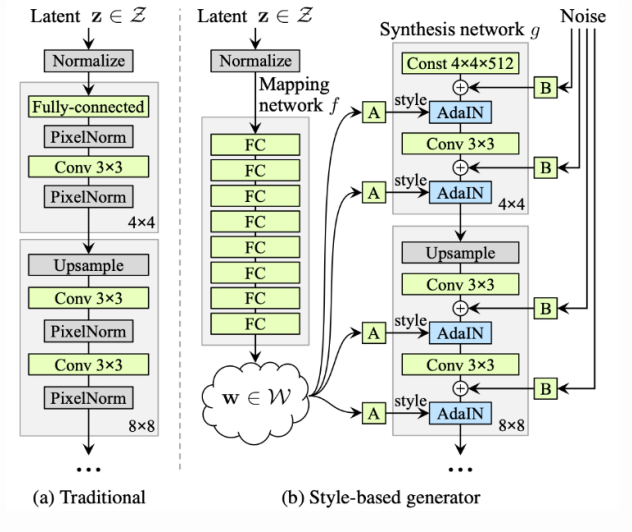
訓練動漫與真實人臉各 2000 張圖，測試資料個 500 張圖







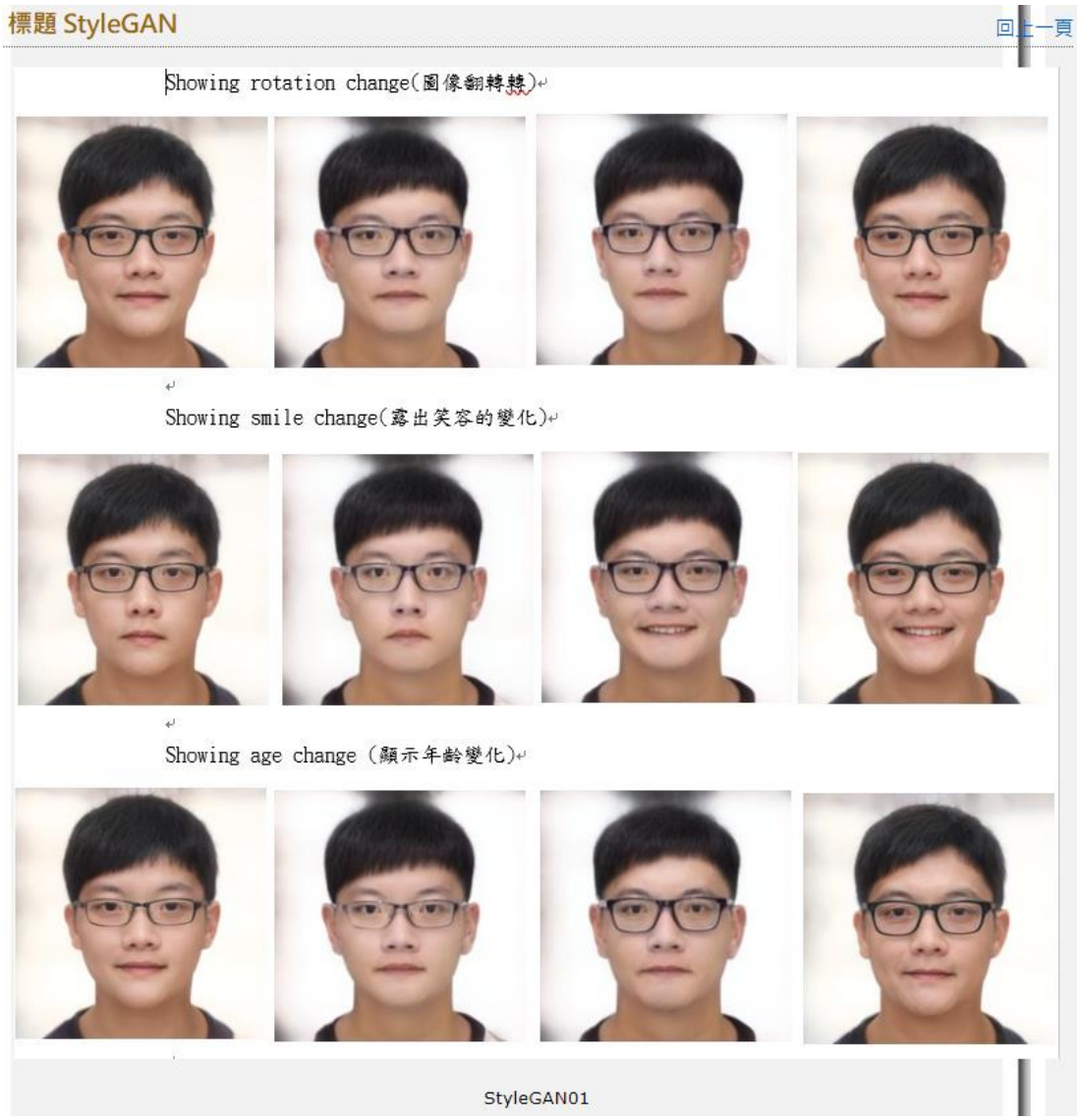
StyleGAN(程式碼測試　　中可以嘗試操作)



StyleGAN 是由 NVIDIA 研究人員於 2018 年 12 月所發表，StyleGAN 受風格遷移 style transfer [2 啟發而設計了一種新的生成器網路結構，網路結構可以透過無監督式的自動學習對圖像更高層屬性或能學習更深層的特徵涵義，例如：臉部圖像的表情和身份，所生成圖像的隨機變化如雀斑和頭髮等，也可以做到一定程度上的控制合成。

映射網路的目標是將輸入向量編碼為中間向量，中間向量的不同元素控制不同的視覺特徵，這是一個非常重要的過程，因為使用輸入向量來控制視覺特徵的能力是非常有限的，因為它必須遵守訓練數據的機率密度，例如，如果黑頭髮的臉部圖像在資料集中比例最高，那麼更多的輸入值將會被映射到該特徵上，更直白地說生成具有黑頭髮的圖片比例較大，因此該模型無法將部分輸入（向量中的元素）映射到特徵上，這一現像被稱為特徵糾纏（features entanglement），然而透過使用另一個神經網路， 該模型可以生成一個不必遵循訓練數據分佈的向量，並且可以減少特徵之間的相關性， 例如：你想控制某個特徵生成盡可能生成你想要的，有可能想單方面控制膚色、眼睛顏色等等。

隨著輸入網路不同的位置，加設噪聲輸入在網路的前半段，特徵變化會與原圖差距較大，例如表情、髮型等等，但噪聲是輸入網路模型的後半段影像的改變會在比較細微部分例如雀斑、顏色、眉毛等等



Dlib 臉部上色上妝(機器學習)

運用 OpenCV 環境建置及 Dlib 函式庫對一張正臉做偵測，圖 39 所偵測到 68 個臉部特徵點，再來把臉部特定位置上編輯上妝

```
# Make the eyebrows into a nightmare
d.polygon(face_landmarks['left_eyebrow'], fill=(color_01,color_02,color_03, 128))
d.polygon(face_landmarks['right_eyebrow'], fill=(color_01,color_02,color_03, 128))
d.line(face_landmarks['left_eyebrow'], fill=(color_01,color_02,color_03, 150), width=5)
d.line(face_landmarks['right_eyebrow'], fill=(color_01,color_02,color_03, 150), width=5)

  # Gloss the lips
d.polygon(face_landmarks['top_lip'], fill=(color_04,color_05, color_06, 128))
d.polygon(face_landmarks['bottom_lip'], fill=(color_04,color_05, color_06, 128))
d.line(face_landmarks['top_lip'], fill=(color_04, color_05, color_06, 64), width=8)
d.line(face_landmarks['bottom_lip'], fill=(color_04, color_05, color_06, 64), width=8)

# Sparkle the eyes
d.polygon(face_landmarks['left_eye'], fill=(color_07, color_08, color_09, 30))
d.polygon(face_landmarks['right_eye'], fill=(color_07, color_08, color_09, 30))

# Apply some eyeliner
d.line(face_landmarks['left_eye'] + [face_landmarks['left_eye'][0]], fill=(0, 0, 0, 110), width=6)
d.line(face_landmarks['right_eye'] + [face_landmarks['right_eye'][0]], fill=(0, 0, 0, 110), width=6)
```

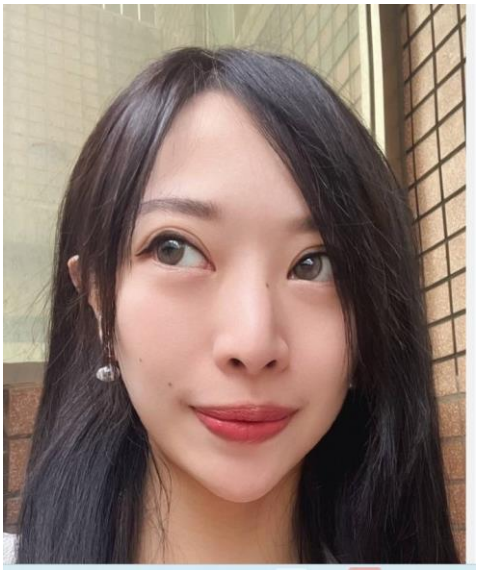
Dlib 提供的方法是在一個已標註好的資料集上做訓練，然後將訓練好的模型給 shape_predictor 方法當參數使用

5 個關鍵點會使用左眼頭尾、右眼頭尾、鼻頭這五個點來做辨識；

68 個關鍵點會使用外輪廓、左眉毛、右眉毛、左眼、右眼、鼻子、嘴巴共 68 個點來做辨識。

而上妝上色部分根據臉部特徵點位置上色。

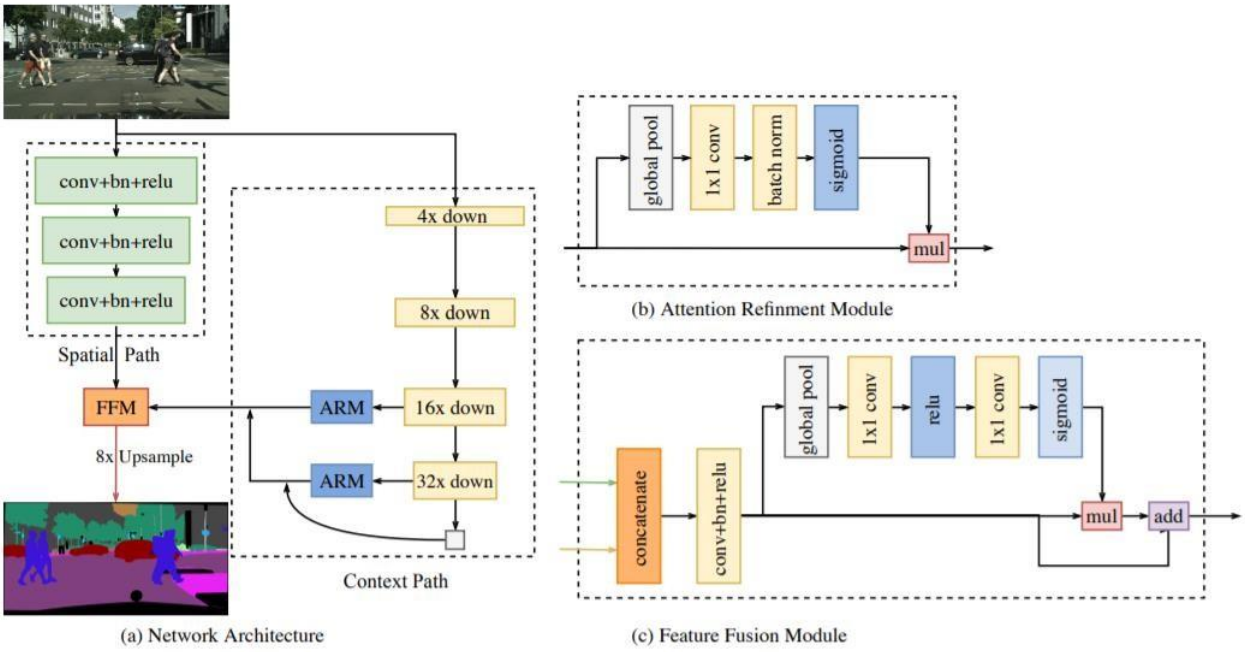
介面呈現:



上妝前

Dlib 上妝後

BiSeNet 臉部上色上妝(深度學習):



BiSeNet 架構圖中 a 部分 Context Path 網路架構部分，在語意分割(semantic segmentation)任務中，接受域(receptive field)對於語意分割(semantic segmentation)至關重要，為增大接受域(Receptive field)一些方法利用金字塔池化模型(Pyramid Pooling Module) ，金字塔型空洞池化（Atrous Spatial Pyramid Pooling,ASPP 或者較大的卷積核(kernel)，但是這些操作比較耗費計算和記憶體，導致速度慢。

出於較大接受域(Receptive field)和計算效率兼得的考量，本文提出 Context Path 網路，它充分利用輕量級模型與全域平均池化(Global Average Pooling,GAP)以提供較大接受域(Receptive field)。

使用輕量級的模型比如 Xception 來編碼高層語意資訊，之後加一層全域性平均池化(Global Average Pooling,GAP)來獲取最大的接受域(Receptive field)，最後將全域性池化(Global Average Pooling,GAP)的結果上採樣(upsampling)並於輕量級模型的輸出特徵相融合。

Spatial Path 網路編碼底層的網路更關注於細節資訊，Context Path 網路編碼高層的網路更關注於語意資訊，而高層的語意資訊能夠幫助我們準確的偵測出目標。

BiSeNet 架構圖中 b 部分中 Attention Refinement Module(ARM)與 Attention Refinement Module(ARM)程式碼，在 Context Path 中，透過全域平均池化(Global Average Pooling,GAP)來抓取全域性語意資訊，並基於此學習一個 Attention 向量，以此來幫助特徵學習。換句話說，即多添了一個分支，用以學習一個向量(weight vector)，來當 feature map 各個通道

(channels)的權重。然後原來的輸出通道，每個通道用對應的權重進行加權(對應通道的每個元素與權重分別相乘)，得到新的加權後特徵，新的特徵稱為 feature recalibration。

這種注意力機制讓模型可以更加關注資訊量最大的通道(channel) 特徵，而抑制那些不重要的 channel 特徵，而這有助於高層的語意資訊能夠幫助我們準確的偵測出

目標。

BiSeNet 架構圖中 b 部分中 Feature Fusion Module(FFM)與 Feature Fusion Module(FFM)機制程式碼，最終在 FFM 前得到的兩條路徑的輸出特徵尺寸不同，因此不能簡單的相加。富含位置資訊的 Spatial Path 特徵和富含語意資訊的 context Path 特徵二者在不同的層次，因此需要 FFM 來融合，對於給定的不同特徵輸入，首先將兩者特徵拼接(concatenate)，然後利用批次標準化(Batch Normalization)來調整特徵的尺度，接著對特徵拼接(concatenate)結果進行平均池化得到一個特徵向量並計算一個權重向量，權重向量可以調整特徵權重，從而結合各特徵。

style_transfer:

採用了 VGG 的模型來對原圖(original image)及風格圖(style image)提取特徵，來實施將一張照片轉換成另一張圖的風格或是畫風，來生成一張新的圖片，讓它同時擁有原圖(original image)的內容以及風格圖(style image)的風格。

原理

1. 利用 VGG 模型分別提取原圖(original image)及風格圖(style image)的特徵。

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

2. 計算原圖(original image)和合成圖間的 loss function，算法如下，其中 F 是合成圖的內容(content)的特徵向量，P 是原圖的內容(content)的特徵向量。

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^{l} - A_{ij}^{l}\right)^2$$

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

3. 計算風格圖(style image)和合成圖間的 loss function，算法如下，其中 G 是 合成圖的風格(style)的特徵向量，A 是原圖的風格(style)的特徵向量。

4. 計算總損失函數(total loss function)=原圖內容的損失函數(content loss) + 風格圖風格的損失函數(style loss)，並利用梯度下降法(Gradient descent)求得總損失函數最小化，當中α和β為權重，看是要合成圖較接近原圖的內容(content)還是風格圖的風格(style)。

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

匯入套件與先設定期望的合成圖中，原圖和風格圖所佔的比重以及生成出來的圖的大小，還有損失總差異預設的 value。

```python
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import time
from PIL import Image



StylePath ='C:\\Users\\user\\Desktop\\django\\AI_generated_face_project_02\\album\\temp\\'
ContentPath = 'C:\\Users\\user\\Desktop\\django\\AI_generated_face_project_02\\album\\temp\\'
content_path = ContentPath+'content_path.jpg'
style_path = StylePath+'style_path.jpg'



# A function to load the input images and set its dimensions to 1024 x 768
def load_image(image_path):
    max_dim=512
    img = tf.io.read_file(image_path)
    img = tf.image.decode_image(img, channels=3)# decodes the image into a tensor
    img = tf.image.convert_image_dtype(img, tf.float32)

    shape = tf.cast(tf.shape(img)[:-1], tf.float32)
    long_dim = max(shape)
    scale = max_dim / long_dim
    new_shape = tf.cast(shape * scale, tf.int32)

    img = tf.image.resize(img, new_shape)
    img = img[tf.newaxis, :]# broadcasting the image array so that it has a batch dimension

    return img
```

1.style_loss(): 讓生成的圖的風格、紋理和風格圖接近。這裡用到一個在紋理生成領域比較常見的 Gram Matrix，來衡量 style image 和生成的合成圖在風格上的相似性。這裡 Gram Matrix 會計算經過 convolutional feature map 後得到的特徵向量(Feature vector)，因此將這些 Feature vector 做 Gram Matrix(原矩陣乘上轉置後的矩陣)得到的乘積越小，那個位置所表示的特徵程度越不重要也越不相關，乘積越大，特徵越重要也越相關，藉此提供了有關圖像紋理之類的信息。

2.content_loss():生成的圖要和原圖裡的內容相近。

3.total_variation_loss():總損失差異，一個正則化的 loss(regularization loss)常被用在損失函數裡，可以起到平滑圖像，去除雜訊(nosie)的作用，以利合成圖的連貫性。

```python
# Compute the gram matrix
# Einsum allows defining Tensors by defining their element-wise computation.
# This computation is defined by equation, a shorthand form based on Einstein summation.
def gram_matrix(input_tensor): # input_tensor is of shape ch, n_H, n_W
    result = tf.linalg.einsum('bijc,bijd->bcd', input_tensor, input_tensor)
    input_shape = tf.shape(input_tensor)
    num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32) # Unrolls n_H and n_W
    return result/(num_locations)
```

```python
def total_cost(outputs):
    style_outputs=outputs['style']
    content_outputs=outputs['content']
    style_loss=tf.add_n([style_weights[name]*tf.reduce_mean((style_outputs[name]-style_targets[name])**2)
                         for name in style_outputs.keys()])
    style_loss*=style_weight/len(style_layers)# Normalize

    content_loss = tf.add_n([tf.reduce_mean((content_outputs[name]-content_targets[name])**2)
                             for name in content_outputs.keys()])
    content_loss*=content_weight/len(content_layers)
    loss=style_loss+content_loss
    return loss
```

在考慮計算 style loss 的時候只用了 VGG19 中 style_layer_names 列出的 5 層才提取特徵而已，而在考慮計算 content loss 的時候只用了”block4_conv2" 一層來提取而已

```
# Define the content image representation and load the model
x=tf.keras.applications.vgg19.preprocess_input(content_img*255)# needs preprocess
x=tf.image.resize(x, (256,256))# the vgg19 model takes images in 256
vgg_model=tf.keras.applications.VGG19(include_top=False, weights='imagenet')
vgg_model.trainable=False
vgg_model.summary()


# Chooose the content and style layers
content_layers=['block4_conv2']
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1']
```

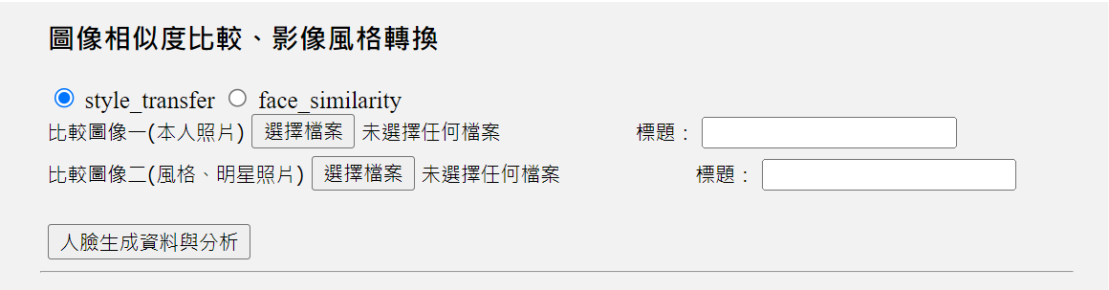
我們為每一層的輸出分配權重，以控制它們對最終影像的風格效果。權重範圍為 0-1。

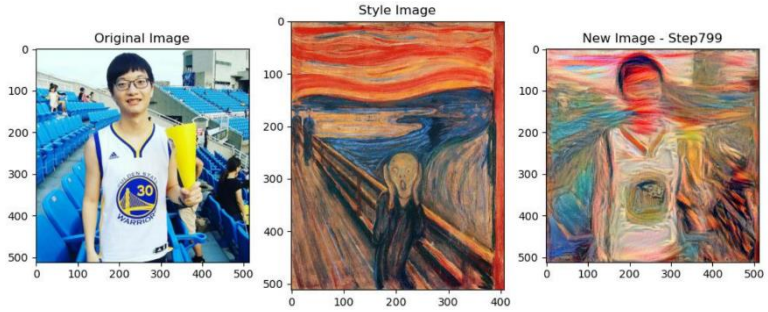
```
style_weight=40
content_weight=10

# Custom weights for different style layers
style_weights = {'block1_conv1': 0.7,
                 'block2_conv1': 0.19,
                 'block3_conv1': 0.24,
                 'block4_conv1': 0.11,
                 'block5_conv1': 0.26}
# style_weights = {'block1_conv1': 0.3,
#                  'block2_conv1': 0.45,
#                  'block3_conv1': 0.15,
#                  'block4_conv1': 0.05,
#                  'block5_conv1': 0.05}
```

然後開始訓練

```
@tf.function()
def train_step(image):
    with tf.GradientTape() as tape:
        outputs = extractor(image)
        loss = total_cost(outputs)

    grad = tape.gradient(loss, image)
    opt.apply_gradients([(grad, image)])
    image.assign(clip_0_1(image))
```

介面操作:





face_similarity

Face Recognition 是在 github 上開源的 python library，作者透過包裝 dlib 的方式來實現快速且簡單的人臉辨識開發，同時提供已經訓練好的 CNN 人臉偵測模型、ResNet 人臉辨識模型。

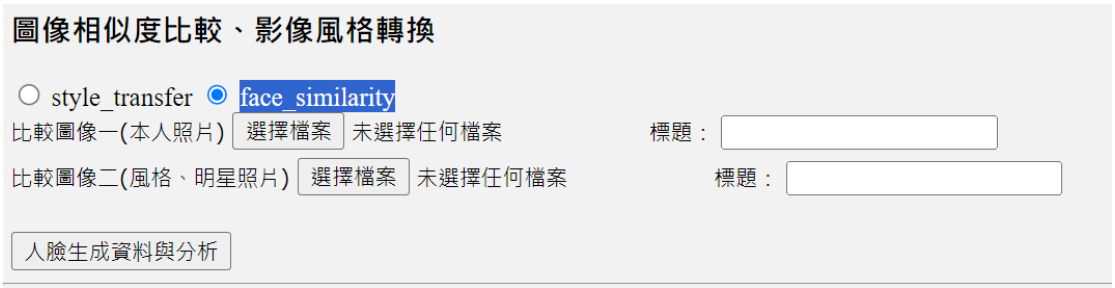
準備好已知的人臉圖片，最好是一張只有一個人臉，且足夠清晰，這樣我們下一步轉成特徵向量比較方便。同時準備一些「要被比對的人臉」的人臉圖片，用來測試人臉辨識的結果。

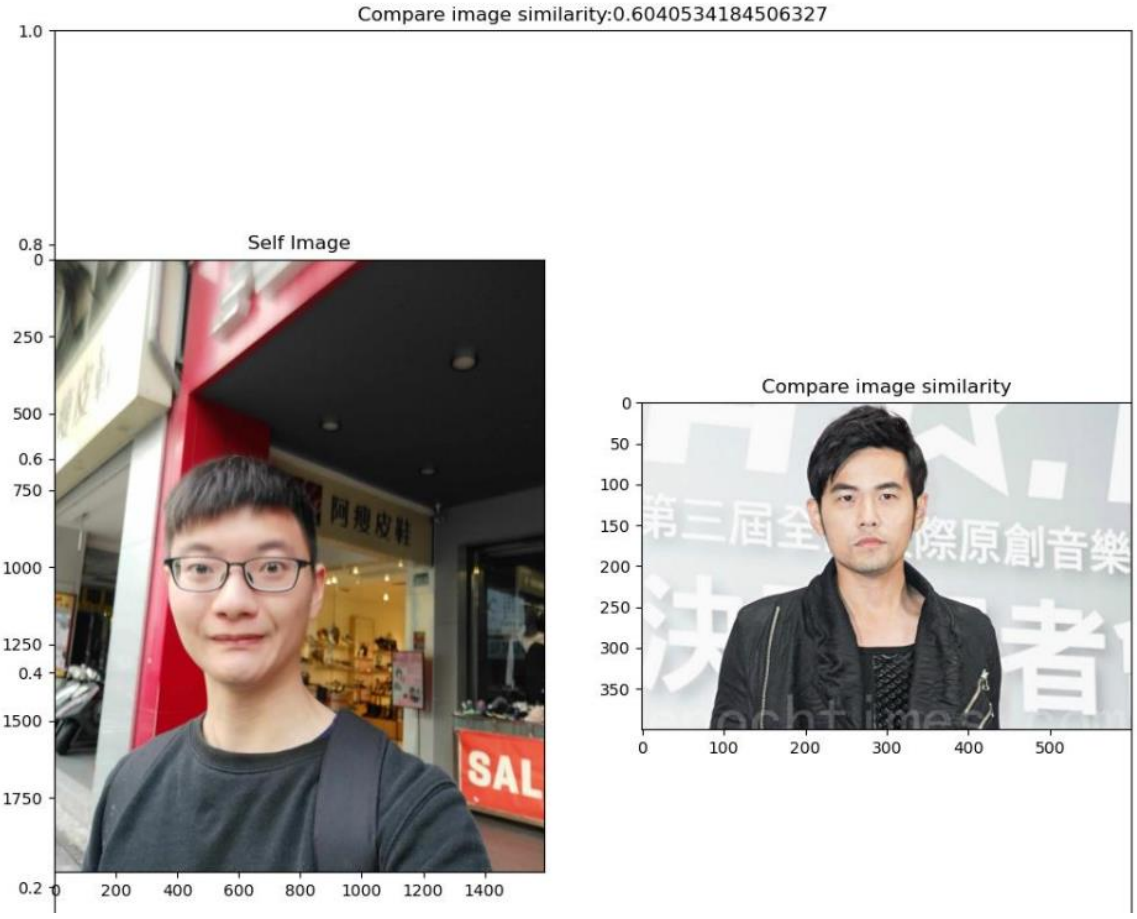
把兩張圖轉成特徵向量 face_encodings()

拿「要被比對的人臉」的特徵向量，一一去比對已知人臉的特徵向量距離得出相似度

```
def face_similarity_compare():
    # 已知人臉位置的臉部編碼
    known_image_path =  "C:\\Users\\user\\Desktop\\django\\AI_generated_face_project_02\\album\\temp\\content_path.jpg"
    known_image      = face_recognition.load_image_file(str(known_image_path))
    known_image_encoding = face_recognition.face_encodings(known_image)[0]
    # 未知人臉位置的臉部編碼
    new_image_path = "C:\\Users\\user\\Desktop\\django\\AI_generated_face_project_02\\album\\temp\\style_path.jpg"
    new_image      = face_recognition.load_image_file(str(new_image_path))
    new_image_encoding = face_recognition.face_encodings(new_image)[0]
    # 進行計算並顯示結果
    distance_img = face_recognition.face_distance([known_image_encoding], new_image_encoding)
```

介面操作:


圖像相似度比較、影像風格轉換
style_transfer
face_similarity
比較圖像一(本人照片)
選擇檔案
未選擇任何檔案
標題：
比較圖像二(風格、明星照片)
選擇檔案
未選擇任何檔案
標題：
人臉生成資料與分析


Compare image similarity:0.6040534184506327
Self Image
Compare image similarity