淡江大學資訊工程學系 110 學年度

專題實驗成果報告

專題名稱：模仿遊戲—人臉表情相似度挑戰

專題組員：

組長：許儒怡

組員：陳燮涵 李承謙 林伯翰 周卓盈 王靖雄

指導教授：張志勇

目錄

一、動機與目的 3

1.研究背景 3

2.研究動機 3

3.研究目的 4

二、使用工具 4

三、功能描述 5

四、架構圖 5

五、實驗階段 6

第一個階段：確定遊戲玩法 6

第二個階段：設計遊戲系統框架與核心算法研究 6

1.遊戲 ui 與機制框架 6

2.表情相似度算法思路： 9

第一步：面部關鍵點的提取：Dlib 10

第二步：運用 Procrustes Analysis 的算法進行人臉對齊 11

第三步：表情相似度算法 15

第四步：運用高斯濾波穩定結果 22

3. 遊戲內容的呈現 24

六、結論 29

一、開發難點 29

二、未來展望 29

三、心得 29

七、海報 30

八、參考文獻 31

一、動機與目的

1. 研究背景

人工智慧，作爲一項誕生與 1950 年代的技術，近年來由於數據的爆炸，和底層技術的不斷成熟，迎來了其第二春，并且逐漸開始展現其在各種領域的强大潛能。作爲一項電腦相關的技術，其解決問題的能力之强和適用範圍之廣無不令人驚嘆。現今，無論是在醫療行業、娛樂行業、金融行業都存在著其身影，甚至在大衆眼中同樣神秘的圍棋領域也一樣卓有成效，在人們以爲 AI 無法觸及的辯論領域也能有所建樹。

影像處理，作爲人工智慧中的一個重要分支在研究領域和我們的生活中存在著重要的價值。隨著相應技術的不斷成熟，大量不同且各具特色的 AI 衍生應用和技術也不斷的湧現。而這一强大的工具也自然被充滿好奇心的人們應用在對於人體本身的探索當中。其中，作爲人體中變化最爲豐富的部位以及人類情緒表達的重要途徑之一，人臉及其相關的面部表情也自然受到了高度重視。在各方努力的探索和研究之下，人臉識別和情緒識別的技術也因此愈發的成熟，致使市面上出現了形形色色的不同套件和技術成果。不僅如此，伴隨著基本人臉識別技術的不斷成熟和完善，相關應用面的產品也在不斷出現，例如市面上常見的表情游戲就是其中之一。可惜的是市面上的表情游戲大多都是對於固定的幾個表情的呆板匹配，而人臉識別在更具重大意義的醫療方面的應用也并不豐富，因此在相關領域進行進一步的研究和嘗試也是頗具意義和趣味性的。

2. 研究動機

在經過了一定時間的對於人工智慧基礎知識的瞭解和學習之後，我們便有了想要開發和研究 AI 相關應用的想法，其中人臉識別、情緒識別便率先進入了我們的視野。在進行了大量的資料查找之後，我們發現做一個小小的表情識別和相似度匹配的小游戲頗具趣味。

Figure 1 研究動機

我們發現市面上的人臉匹配的游戲多是對固定的幾個表情的匹配，而且多爲單人游玩，顯得過於的乏味。於是我們便想到要研究一款可以跳脫出固定的表情樣板的表情游戲，并且開發雙人模式。如此一來其可玩性和趣味性便得以大大的提高，對於玩家能更有吸引力，讓病患能更有堅持的動力以及通過和病友或醫師一起游玩獲得更好的正反饋。

3. 研究目的

此研究旨在解決市面上表情遊戲採用單一表情標籤進行表情匹配的趣味性不足問題。為此，我們運用機器學習的思想，利用 python 製作表情相似度遊戲。透過對比不同表情之間的相似度，期望提高遊戲趣味性，本研究也希望能夠提供一種新的基於人臉關鍵點的表情相似度計算方法，以應用於更廣泛的領域。

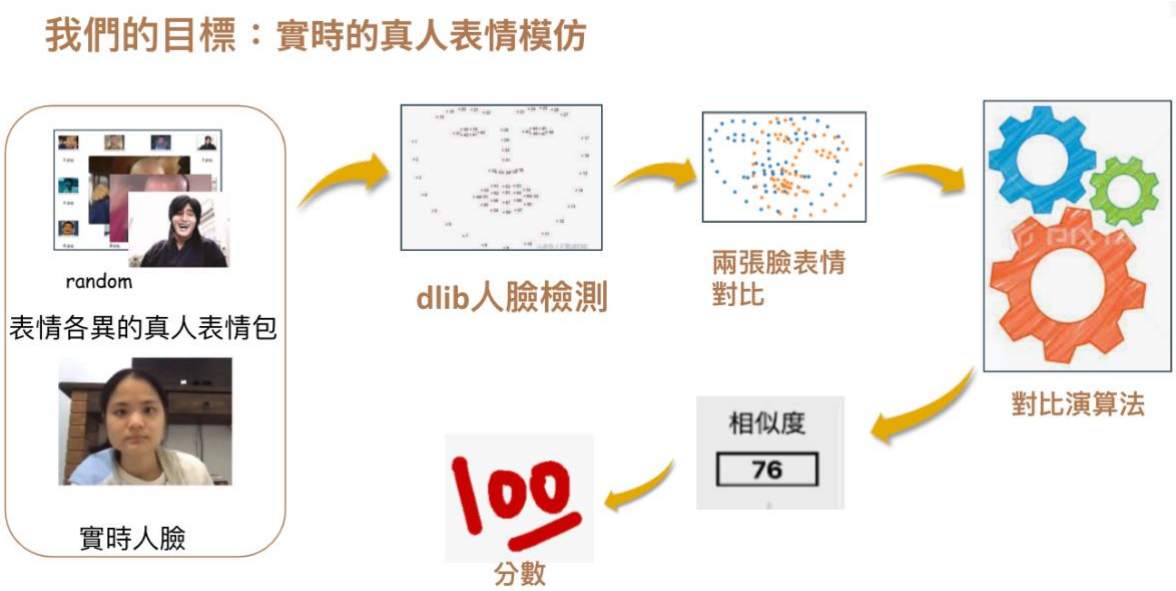


Figure 2 研究思路

二、使用工具

1. OpenCV:

   OpenCV 的全稱是 Open Source Computer Vision Library，是一個跨平台的電腦視覺庫，可用於開發即時的圖像處理以及圖型識別程式。本項目運用 opencv 開啟攝像頭，捕捉攝像頭畫面，處理圖片等圖片處理功能。

2. Dlib:

   Dlib 是一套包含了機器學習、計算機視覺、圖像處理的函式庫。本項目應用了 dlib 的人臉辨識模塊，運用 dlib 官方訓練的人臉 68 特徵點模型（shape_predictor_68_face_landmarks.dat.bz2）來檢測人臉特徵點。

3. PyQt5、Qt Designer

   PyQt 是 Python 語言的 GUI 編程解決方案，QtDesigner 是 Qt 所包含的視覺化 UI 設計器，它使用拖拉操作來設計圖形介面。在設計的同時，還能夠直接預覽最終的表單效體。當表單很複雜或者整個程式需要大量的表單時，Qt 設計器可以節省大量的代碼。本項目運用 QtDesigner 輔助設計 UI 並生成 python 程式碼。

4. numpy 数据結構，本項目引用 numpy 來生成矩陣，變換矩陣。

5. SciPy 數學工具，本項目引用此工具內建的函數來計算歐式距離，餘弦距離。

三、功能描述

是一款人臉表情識別小遊戲，我們將人臉表情識別和遊戲結合，做出個可以鍛鍊人臉部肌肉的小遊戲。遊戲分為單人模式和雙人模式:
（1）單人模式中玩家一共可以挑戰三個關卡，根據系統所產生出的表情圖片來做出相對應的表情，然後系統會根據玩家做出來的表情相似度來進行打分數。
（2）雙人模式中玩家間可以選擇誰模仿誰的表情，計時 15s，最後畫面按相似度給分。

我們需要解決的程式功能包括:

1. 出題 - 隨即從程式庫取出一張表情包

2. 圖像捕捉 - OpenCV 捕捉畫面幀

3. dlib 獲取和繪製人臉特徵點

4. 計算相似度

5. 根據難度設置倒計時和圖片數量

6. 判斷倒計時時間和計算分數

7. 玩家選擇遊戲模式，暫停/開始音樂，返回選擇頁面，結束遊戲， - pyqt5 建立按鈕事件

四、程式架構圖

程式結構架構圖:

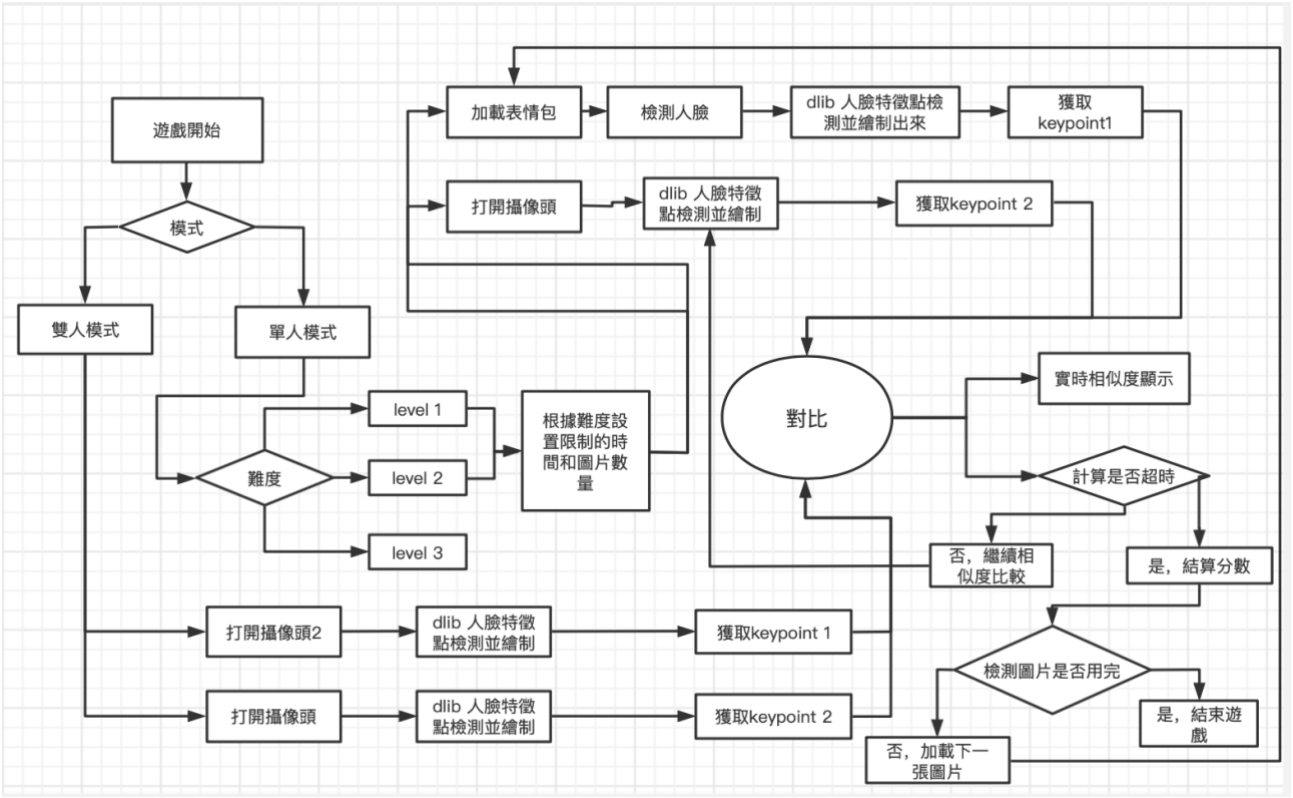


Figure 3 程式架構圖

五、專題實驗階段

第一個階段：確定遊戲玩法

在該階段，我們確定了我們研究的方向——表情識別。

對於這個大方向，我們先各自了解了下市面上對於表情識別的一些應用和使用到的技術。基本都是透過 opencv 技術和其套件，來獲取人臉的特徵。在大致了解並學習了 opencv 的用法後，我們通過組內討論確定出我們的專題題目。

根據專題的題目，我們也了解了下網路上能找到的和表情有關的小遊戲。蒐集匯總了下這類遊戲的玩法，並最終討論出我們遊戲的玩法和遊戲核心機制是什麼。
接著我們開始分工合作，一部分人策劃遊戲玩法及遊戲數值；一部分人收集遊戲所需的表情包，建立表情包庫；一部分人研究遊戲核心演算法；一部分人測試遊戲裡會使用到的套件。

第二個階段：設計遊戲系統框架與核心算法研究

1. 遊戲 ui 與機制框架

1）UI 介面

❶ ui設計與美術素材收集和實現

Figure 4 UI 製作流程

在 PyQt 中編寫 UI 界面可以直接通過代碼來實現，也可以通過 Qt Designer 來完成。Qt Designer 的設計符合 MVC 的架構，其實現了視圖和邏輯的分離，從而實現了開發的便捷。Qt Designer 中的操作方式十分靈活，其通過拖拽的方式放置控件可以隨時查看控件效果。Qt Designer 生成的.ui 文件（實質上是 XML 格式的文件）也可以通過 pyuic5 工具轉換成.py 文件。

QtDesigner 隨 PyQt5-tools 包一起安裝，其安裝路徑在“Python 安裝路徑\Lib\site-packages\pyqt5-tools”下。

2）UI 概念設計：

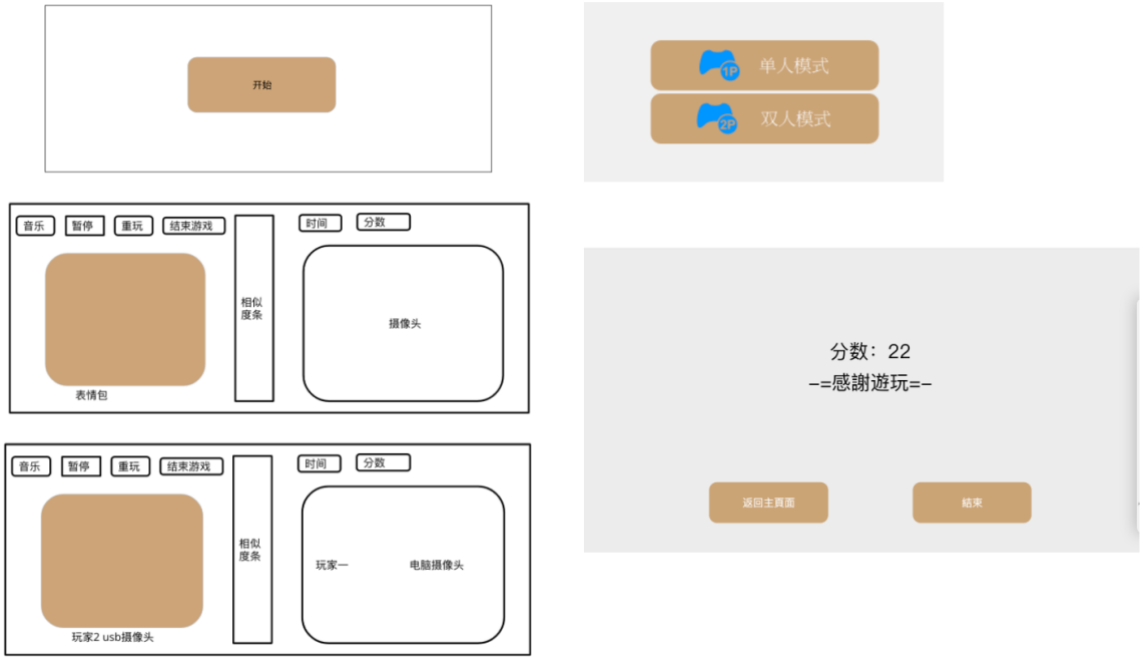


Figure 5 UI 設計圖

3）Qt designer 運作

1. 工具箱:有各種可用的物件（容器、輸入輸出視窗、互動按鈕等等）
2. 畫面編輯區:設計出來的畫面
3. 物件內容區:包含物件圖層樹狀圖、物件的屬性視窗、及物件的事件編輯視窗

Figure 6 Qt Designer 介面製作

2. 遊戲機制設計

Figure 7 遊戲機制

1）遊戲機制的函數架構關係圖：

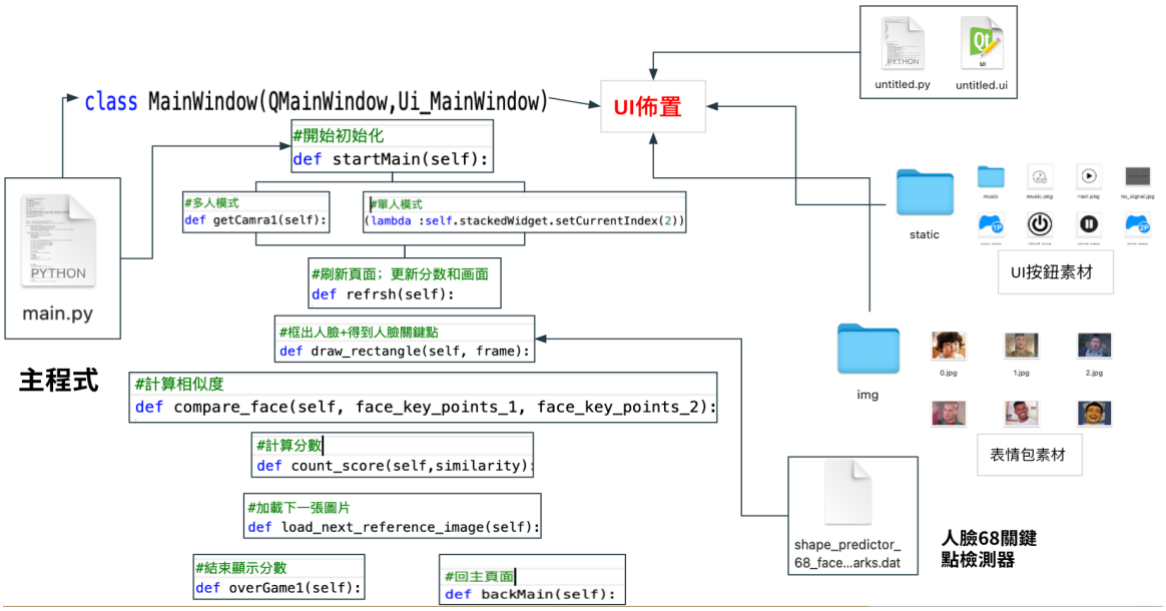


Figure 8 函數架構關係圖

2、表情相似度算法思路：

1）思路圖：

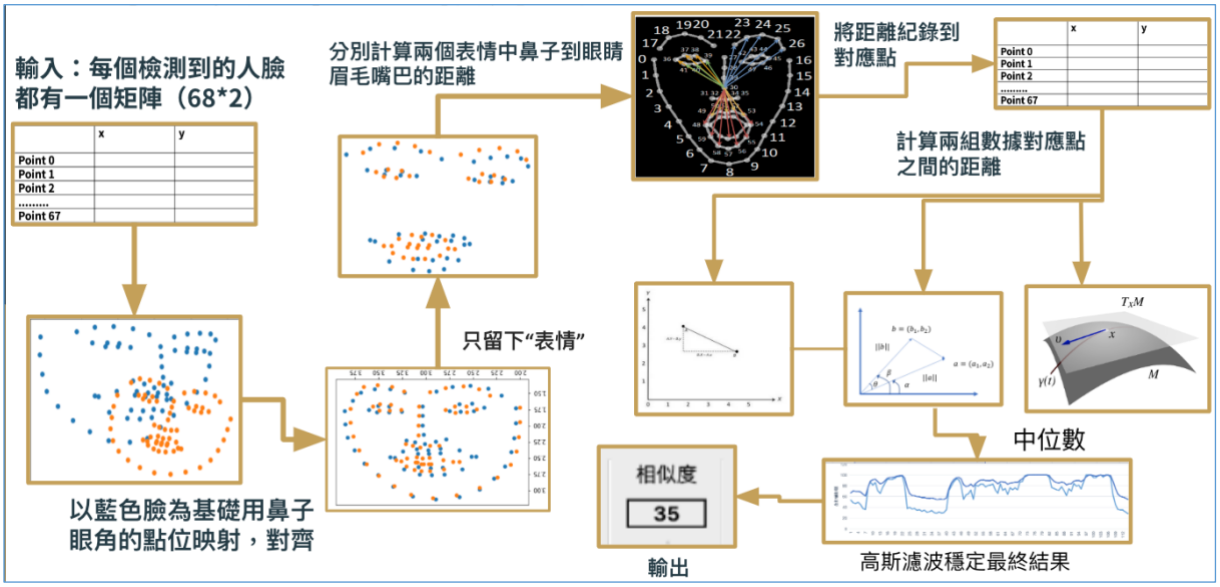


Figure 9 相似度對比流程圖

2）臉部表情的比較步驟

第一步：dlib 獲取人臉 68 關鍵點

第二步：運用 Procrustes Analysis 的算法进行人脸对齐

第三步：提取表示面部签名的关键点之间的向量，構建向量矩陣

第三步 2：運用歐幾里德距離、餘弦距離及黎曼度量距离以得到表情的相似度

第四步：用高斯濾波穩定結果

第一步：面部關鍵點的提取：Dlib

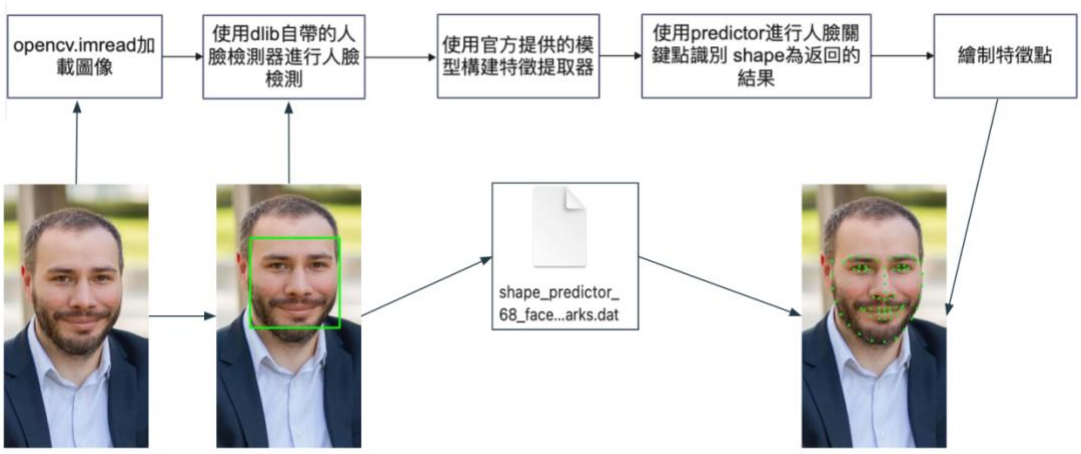


Figure 10 dlib 人臉標誌檢測流程

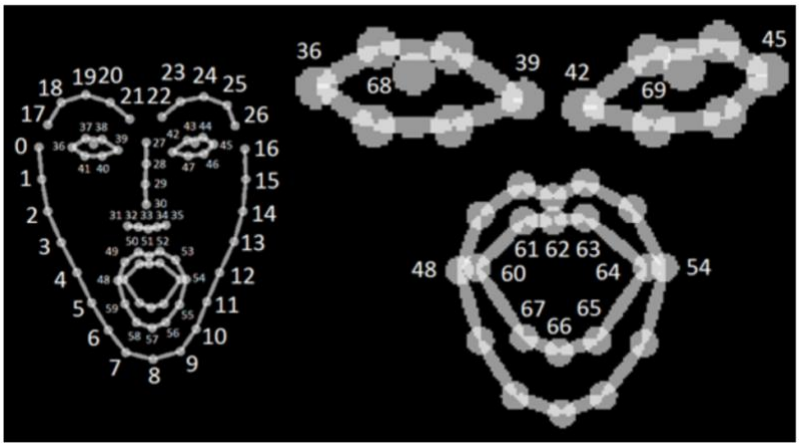


Figure 11 dlib 人臉標誌檢測圖

該系統為我們提供了矩陣中的結果。矩陣結構如下：

#增加一行以方便矩陣變換計算

*每個檢測到的人臉都有一個矩陣（68*3）*

	x	y	x,y 在畫面上的概率
Point 0			1
Point 1			1
Point 2			1
.....			.....
Point 67			1

```
[[237 200   1]
 [237 227   1]
 [239 254   1]
 [243 279   1]
 [251 304   1]
 [264 326   1]
 [282 346   1]
```

Table: dlib 68 人臉特徵數據輸出

第二步：運用 Procrustes Analysis 的算法進行人臉對齊

在這一步中，我們保存在參考圖像和網路攝像頭幀中檢測到的人臉矩陣，以計算人臉表情距離。
此步驟的目標是：
1. 消除自然人臉差異（如胖臉和瘦臉）和玩家與攝像頭距離的影響。
2. 將兩張臉對齊成同一個角度，同一個比例，同一個位置。

方法一：opencv 三點法

OpenCV 內置的仿射變換算法一般是通過將待變換的圖像和目標圖像中的對應特徵點（如人臉關鍵點）進行配准，從而得到仿射變換矩陣，然後使用該矩陣對待變換圖像進行變換，使其與目標圖像對齊。

具體來說，OpenCV 中可以使用 cv2.getAffineTransform()函數計算 2D 仿射變換矩陣，該函數需要輸入原始圖像中的三個點和目標圖像中的三個對應點。得到變換矩陣後，可以使用 cv2.warpAffine()函數對原始圖像進行變換。對於人臉對齊，一般會先檢測出人臉關鍵點，然後根據預定義的關鍵點位置，選取對應的點作為變換的參考點。OpenCV 內置的仿射變換僅僅只要了三個點（左眼，右眼，左右眼中心點）：

左右眼距離確定縮放比例和角度、左右眼中心點確定旋中心點

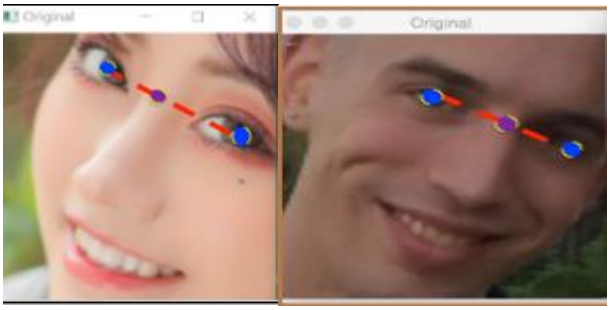


Figure 12 三點法

產生的問題：

OpenCV 的算法是通過對齊關鍵點進行仿射變換，所以如果關鍵點檢測不準確或者檢測到的關鍵點數量不足，就會導致對齊結果不準確。

方法二：Procrustes Analysis 的算法（多點對齊）

Procrustes Analysis 是一種經典的形狀分析算法，它主要用於將不同形狀的物體的坐標點集進行對齊。在人臉識別中，該算法通常用於將不同人臉的關鍵點坐標點集進行對齊，通過在特徵向量上計算仿射變換矩陣，更加靈活，可以適應不同的人臉表情和姿態，且對關鍵點的檢測要求不高，因此在人臉對齊的效果上更加優秀。

我們先分別對待比較的兩個人臉的特徵點進行一定的預處理，然後分別使用 Procrustes Analysis 對兩個人臉進行對齊：

1. 通過將第一個人臉的中心點對準坐標系原點，並對該人臉的臉部輪廓點進行等比例縮放，得到一個標準的人臉形狀。
2. 通過尋找第二個人臉的臉部輪廓點和第一個人臉的標準臉部輪廓點之間的相應關係，計算出一個仿射變換矩陣，用該矩陣將第二個人臉的特徵點進行對齊。

1. 點位分組

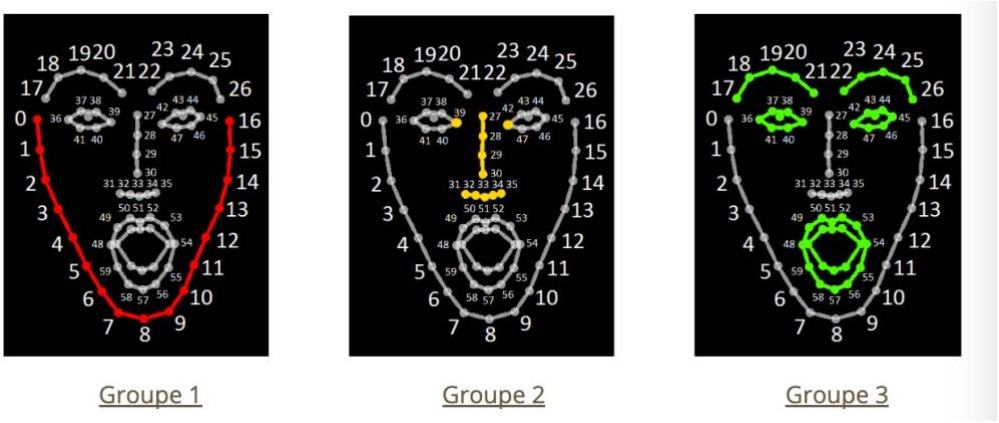


Figure 13 分組示意圖

第一組：無關點（0~16）
這些點與面部表情或面部位置沒有直接關係。另一方面，它們與面部的自然形狀相關。

第二組：定位點 [27, 28, 29, 30(鼻子鼻尖) 31, 32, 33, 34, 35(鼻子下缘) 36, 39, 42, (左右眼眼角)]
這些點用於定位/旋轉臉部。 我們將在對齊步驟中使用它們。

第三組：特徵點（17~26，36~59）
這些點與面部表情相關，我們可以根據這些點識別容易的表情。

2. 計算變換矩陣：

為了使我們的演示算法對縮放、變換和旋轉具有魯棒(穩健)性(Robustness)，我們將構建一個仿射變換矩陣(T)來對齊兩個面：

$$\begin{bmatrix} \vec{y} \\ 1 \end{bmatrix} = \begin{bmatrix} A & \vec{b} \\ 0, \ldots, 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}$$

Figure 14 仿射變換矩陣公式

相似變換，即是將人臉通過使用旋轉、平移、等比縮放，最大可能的使第二張圖片適合第一張圖片。變換後，兩個面將具有相同的位置、相同的比例和相同的旋轉角度：

A. 對於表情包中的人臉特徵點，通過計算兩個眼睛的距離來對其進行縮放。

目的：

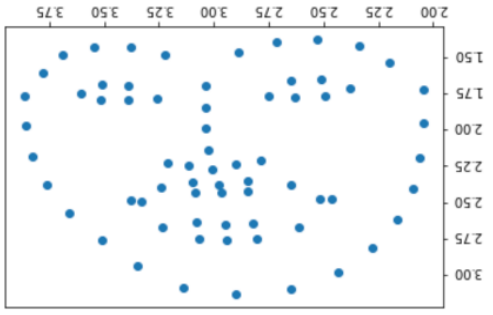
1）將人臉進行縮放是為了進行歸一化，即將不同大小的人臉縮放到相同的尺寸，從而便於進行比較和識別

2）使用兩個眼睛的距離來進行縮放，是因為眼睛在人臉中位置比較固定，而兩個眼睛的距離相對於人臉大小是一個相對固定的比例，因此可以用這個比例來進行縮放，從而使得不同尺寸的人臉在眼睛位置大小上更為一致

3）由於眼睛距離是在人臉平面上的距離，因此使用二維歐幾里得距離來計算兩個眼睛的距離。

縮放矩陣

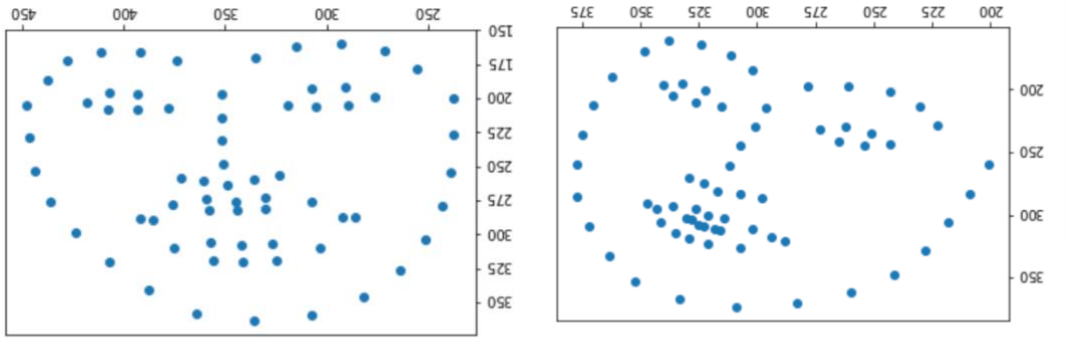
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x\, x \\ s_y\, y \\ s_z\, z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$




```
# alignement_1 對齊1
#縮放
a = 1/(distance.euclidean(face_key_points_1[30, :2], face_key_points_1[8, :2]))
T_1 = np.array([[a,0,0],[0,a,0],[0,0,1]])
face_key_points_1_transformed = np.transpose(np.dot(T_1, np.transpose(face_key_points_1)))
```

Figure 15 縮放對齊人臉的程式實現

B. 定義 Q 矩陣為輸入圖像的坐標點組成的矩陣，S 為目標臉坐標點組成的矩陣， M 為將輸入映射到目標臉的仿射矩陣



2

S的關鍵點　　Q的關鍵點

C. 公式推導:

則有 $Q' = QM$,

而我們需要最小化的是 $minimize \sum (Q' - S)^2$ ，即當 M 取什麼值時，才能使得映射後的 Q’ 與 S 最接近

- 即 $minimize \sum (QM - S)^2$

- 通過最小二乘法我們可得 $M = (Q^T Q)^{-1} Q^T S$

```
M = np.linalg.inv(Q.T @ Q) @ Q.T @ S
```

D. 實驗結果:

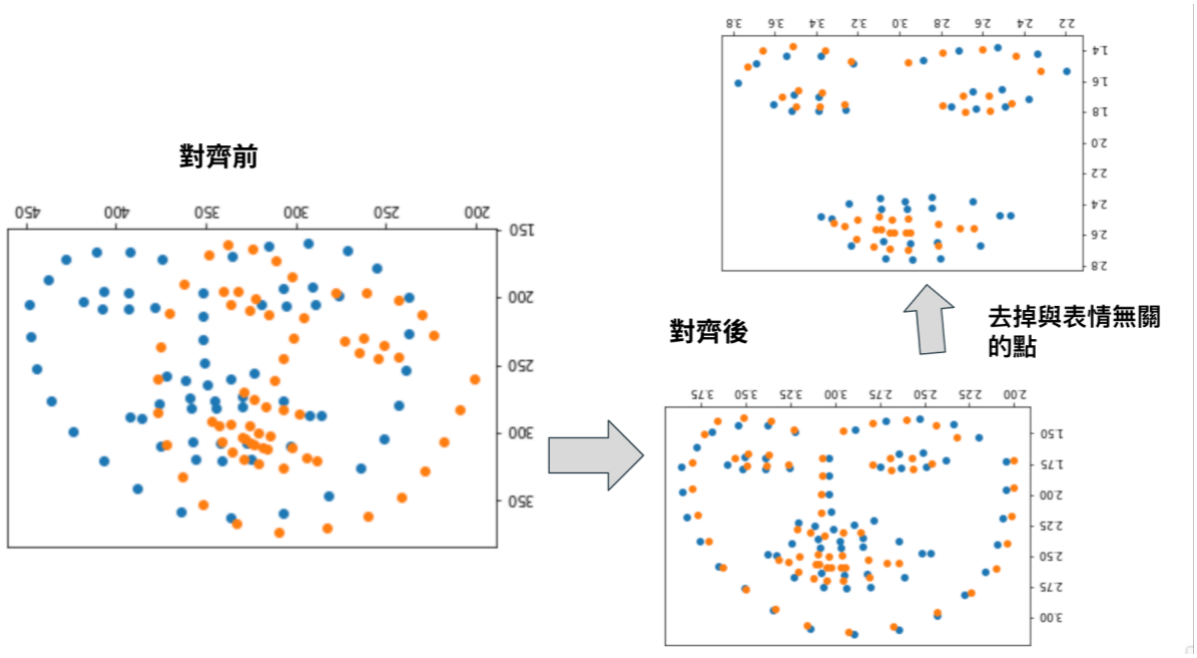


Figure 16 對其前後差異

E、實驗結論：

與使用 OpenCV 內置的仿射變換算法相比，這種方法的優點在於它不會破壞人臉的結構，因為它使用的是對齊點而不是任意選取的點進行變換。另外，這種方法也不需要過多的參數調整，因為它只需要人臉的關鍵點來實現對齊和變換。

第三步：表情相似度算法

0. 此步驟目的：計算兩張臉的特徵點之間的位置差異，並轉換成相似度

1. 構建向量矩陣

對於每個人臉，我們將構建一個簽名矩陣，其向量從參考點（30：鼻尖）開始，到所有第 3 組點結束。

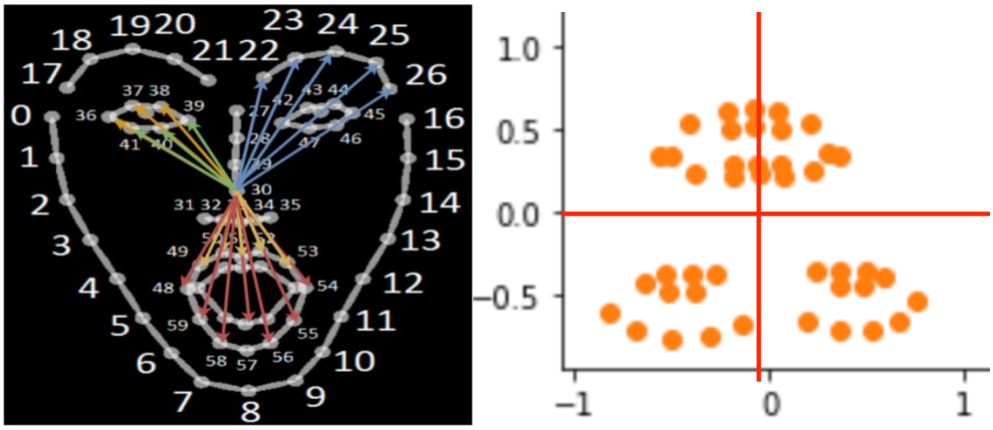


Figure 17 向量矩陣示意圖

```
vectors_1 = face_key_points_1_signature_aligned[:,:2]-face_key_points_1_center_aligned[:2]
print("vectors_1:",vectors_1)
```

```
#矩陣2
vectors_2 = face_key_points_2_signature_aligned[:,:2]-face_key_points_2_center_aligned[:2]
print("vectors_2:",vectors_2)
```

Figure 18 向量矩陣程式實現

輸出向量矩陣：

Table 1 向量矩陣

	x	y
Point 17		
Point 18		
Point 19		
.....		
Point 59		

```
vectors_1: [[-0.82777078 -0.60358286]
 [-0.68980898 -0.71567682]
 [-0.50011151 -0.75878988]
 [-0.31041404 -0.74154466]
 [-0.1379618  -0.67256376]
 [ 0.19832008 -0.65531853]
 [ 0.3535271  -0.70705421]
 [ 0.51735674 -0.70705421]
 [ 0.66394114 -0.65531853]
 [ 0.75016727 -0.52597935]
 [-0.64669592 -0.422508  ]
 [-0.51735674 -0.48286629]
 [-0.37939494 -0.47424367]
 [-0.27592359 -0.37077233]
 [-0.39664016 -0.36214972]
 [-0.53460196 -0.37077233]
 [ 0.23281053 -0.3535271 ]
 [ 0.36214972 -0.43975323]
 [ 0.48286629 -0.44837584]
 [ 0.57771502 -0.38801755]
 [ 0.4914889  -0.34490449]
 [ 0.36214972 -0.34490449]
 [-0.5604698   0.33628188]
 [-0.37939494  0.24143314]
 [-0.18107486  0.21556531]]
ors_2: [[-0.75560186 -0.60852063]
 .63778224 -0.7086002 ]
 .47491439 -0.75272958]
 .28335563 -0.72781281]
 .1181096  -0.66457646]
 .15500062 -0.67619607]
 .28230552 -0.74361246]
 .43853641 -0.7709261 ]
 .5845768  -0.74407272]
 .65523254 -0.6396119 ]
 .61429038 -0.39959899]
 .50666126 -0.44551155]
 .38314498 -0.44663327]
 .28285803 -0.38889989]
 .39048715 -0.34298733]
 .51068495 -0.35027348]
 .1884479  -0.39241847]
 .2963121  -0.46727443]
 .4134265  -0.48052381]
 .48787086 -0.44235765]
 .41912316 -0.38156595]
 .30532723 -0.37672445]]
```

2. 計算這倆個向量矩陣之間的歐基里德距離、餘弦距離及黎曼度量距離

兩張臉的表情差異圖:

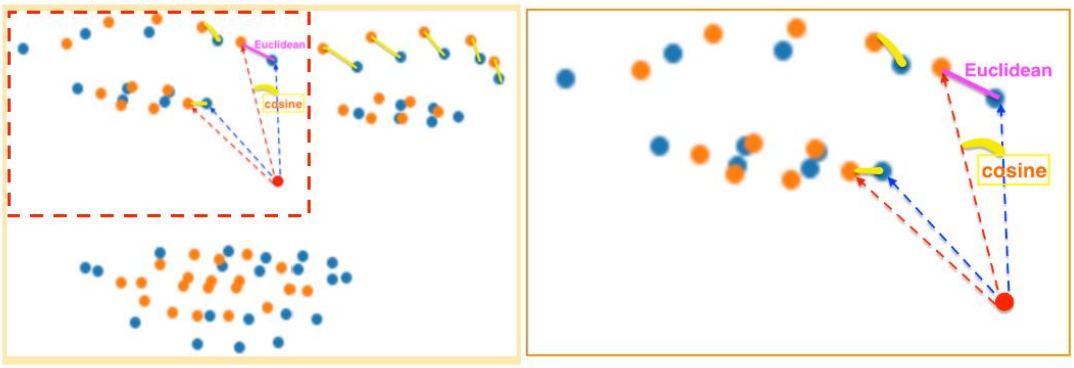
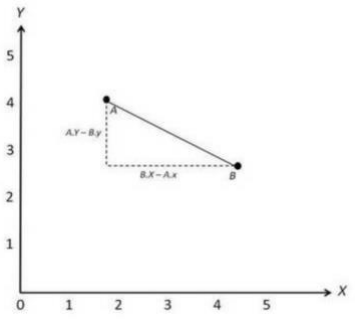


Figure 19 表情對比示意圖

A. 歐幾里德距離:



$$d(x,y)=\sqrt{(\sum(x_i-y_i)^2)}$$

$$sim(x,y)=\frac{1}{1+d(x,y)}$$

Figure 20 歐幾里德距離公式與示意圖

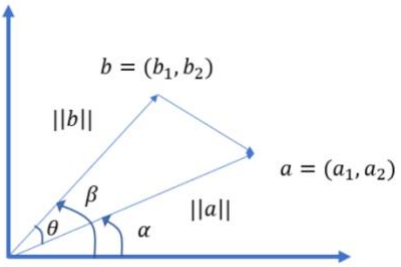
以和鼻尖的距離作為坐標軸，將兩個向量矩陣數據繪制到坐標系上，並計算他們彼此之間的直線距離。

```
# squared Euclidean distance 平方欧几里得距离
squared_euclidean_distance = 0
for i in range(len(signature_indexes)):
  squared_euclidean_distance += (distance.euclidean(vectors_1[i, :], vectors_2[i, :])
```

Figure 21 歐幾里德程式實現

B. 餘弦距離計算：distance = 1- cos θ

餘弦相似度，顧名思義，藉由測量兩個向量夾角的餘弦值，來度量它們之間的相似性。向量之間夾角越小，表示兩個向量的方向越接近



$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}.$$

Figure 22 餘弦距離公式及示意圖

```
cos_distance = 0
for i in range(len(signature_indexes)):
    cos_distance += distance.cosine(vectors_1[i, :], vectors_2[i, :])
```

Figure 23 餘弦距離程式實現

餘弦相似度與歐式距離最大不同之處在於，餘弦相似度的衡量不會受到向量大小的影響，因為在計算上會除以本身向量大小，類似標準化的動作。

C: 黎曼度量距離（Riemann 度量）

由于人臉是非歐幾里德空間，所以前面的兩種方法得出的結果在實際應用中並不準確。人臉可以被視為高維流形空間中的一個點。在這個流形空間中，每個人臉的特徵可以看作一個向量，而這些向量在高維空間中構成了一個流形。

受文獻（*A Novel Space-Time Representation on the Positive Semidefinite Cone for Facial Expression Recognition*）的啟發，我們想出基於黎曼度量距離計算人臉表情相似度的方法。

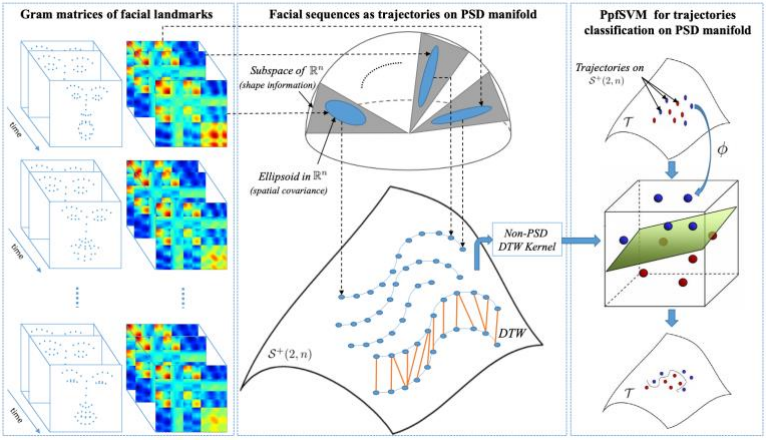
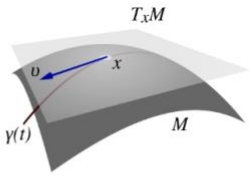


Figure 24 論文圖示：左側的子图是一张人脸图像，中间的子图是将图像转换为矩阵形式，右側的子图是将矩阵嵌入到黎曼流形中的结果。

這篇論文提出了一種基於黎曼流形的人臉表情識別方法，主要是通過將人臉表情數據表示成矩陣形式，然後將矩陣嵌入到黎曼流形中進行處理。論文圖示：左側的子圖是一張人臉圖像，中間的子圖是將圖像轉換為矩陣形式，右側的子圖是將矩陣嵌入到黎曼流形中的結果。

那麼如何把矩陣投影到流形中呢？文獻里推理為使用极坐标分解的方式计算黎曼流形上的距离，這種方法常應用在处理对称正定矩阵（SPD）的问题上（圖示為該文獻部分摘要）：

通過使用切空間和 Riemannian metric，我們可以定義黎曼距離，它是沿著 M 上的曲線的最短路徑。對於 M 中的兩個點 x_1 和 x_2，從 x_1 到 x_2 的最短路徑被稱為黎曼幾何中的地球線。



The tangent space $T_{(U,R^2)}(V_{n,2} \times \mathcal{P}_2)$ consists of pairs (M, N), where M is a $n \times 2$ matrix satisfying $M^T U + U^T M = 0$ and N is any 2×2 symmetric matrix. Bonnabel and Sepulchre define a *connection* (see [22, p. 63]) on the principal bundle $\Pi : V_{n,2} \times \mathcal{P}_2 \to \mathcal{S}^+(2,n)$ by setting the horizontal subspace $\mathcal{H}_{(U,R^2)}$ at the point (U, R^2) to be the space of tangent vectors (M, N) such that $M^T U = 0$ and N is an arbitrary 2×2 symmetric matrix. They also define an inner product on $\mathcal{H}_{(U,R^2)}$: given two tangent vectors $A = (M_1, N_1)$ and $B = (M_2, N_2)$ on $\mathcal{H}_{(U,R^2)}$, set

$$span : V_{n,2} \to \mathcal{G}(2,n)$$

that sends an $n \times 2$ matrix with orthonormal columns U to their span $span(U)$. Given two subspaces $\mathcal{U}_1 = span(U_1)$ and $\mathcal{U}_2 = span(U_2) \in \mathcal{G}(2,n)$, the geodesic curve connecting them is

$$span(U(t)) = span(U_1 \cos(\Theta t) + M \sin(\Theta t)), \quad (4)$$

Figure 25 參考文獻擷取

在該文中，作者使用極坐標來表示子空間之間的測地線，並通過奇異值分解來計算它們之間的距離和角度。具體來說，假設有兩個子空間 U1 和 U2，可以使用極坐標來表示它們之間的測地線。這是通過將 U1 和 U2 投影到它們的共同正交補空間上得到的。這個正交補空間可以使用奇異值分解來計算，並且可以得到它們之間的主角度。然後，通過在這兩個子空間之間移動，可以得到一條連接它們的測地線。

此外，通過計算在每個子空間中取的基向量之間的奇異值分解，可以計算出它們之間的距離和角度。具體而言，可以通過計算這些基向量之間的內積和奇異值來計算它們之間的距離和角度。

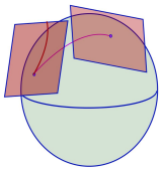


Figure 26 Geodesic 測地線

而通過該論文的啟發，我們應用了對應的算法，運用極坐標分解及奇異值分解（SVD）的方式將矩陣嵌入到黎曼流形中並計算了兩個矩陣的黎曼度量距離：

```
# Riemannian geometry
u1, r1 = polar(vectors_1)
u2, r2 = polar(vectors_2)
u, s, v = np.linalg.svd(np.dot(np.transpose(u1),u2), full_matrices=True)
sigma1 = s[0]
sigma2 = s[1]
theta1 = np.arccos(sigma1)
theta2 = np.arccos(sigma2)
theta = np.array([[theta1, 0],[0, theta2]])
theta_distance = np.linalg.norm(theta)**2
```

Figure 27 黎曼度量距離程式實現

具體地說，該方法有以下步驟：

1. 將第一個向量矩陣轉換為極坐標形式，並計算其極角

```
u1, r1 = polar(vectors_1) '''將第一個向量 vectors_1 轉換為極坐標形式，並計算其極角'''
u2, r2 = polar(vectors_2)'''u1 表示向量 vectors_1 的极角（弧度制），r1 表示向量 vectors_1 的模长'''
```

在極坐標系中，每個點由其到原點的距離和與正半軸的夾角表示。因此，將向量轉換為極坐標形式意味著將向量的大小和方向分別表示為極坐標中的距離和角度，這樣可以更方便地計算角度的相似度。

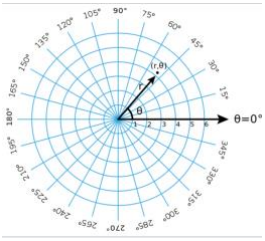
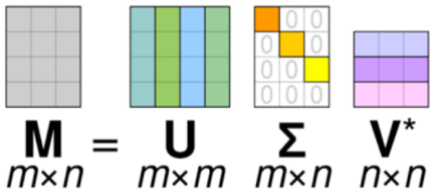


Figure 28 極坐標

2. 計算兩個向量 u1 和 u2 之間的余弦相似度，它使用奇異值分解（SVD）來實現。



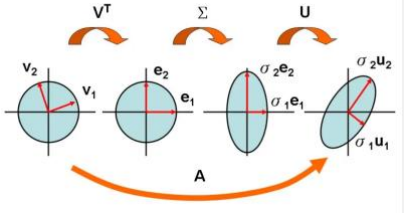


Figure 29 奇異值分解（SVD）示意圖

* 對角矩陣 Sigma 的對角線元素即為矩陣 M 的奇異值，它代表了矩陣 M 在左奇異向量組 U 和右奇異向量組 V 張成的子空間中的縮放因子。在幾何上，這些奇異值代表了矩陣 M 在左奇異向量組和右奇異向量組張成的子空間中的伸縮程度。

```
u, s, v = np.linalg.svd(np.dot(np.transpose(u1),u2), full_matrices=True)
sigma1 = s[0]
sigma2 = s[1]
theta1 = np.arccos(sigma1)
theta2 = np.arccos(sigma2)
```

Figure 30 建立子空間之程式實現

*上面這段代碼計算了兩個向量的奇異值分解結果中的前兩個奇異值，並將它們轉換為對應的角度值。

具體而言，sigma1 和 sigma2 是 s 數組中的前兩個元素，表示兩個向量的奇異值分解結果中的前兩個奇異值。這裡的奇異值可以被視為向量在相應方向上的長度縮放系數。由於奇異值分解結果中的奇異值是按照從大到小的順序排列的，因此 sigma1 表示向量在最重要的方向上的長度縮放系數，sigma2 則表示在第二重要的方向上的長度縮放系數。

然後，theta1 和 theta2 分別通過 np.arccos 函數計算了 sigma1 和 sigma2 所對應的角度值。這裡的角度值表示兩個向量在相應方向上的夾角大小，是通過余弦定理計算得出的。

3. 計算兩個向量 u1 和 u2 之間的旋轉角度，即 theta，以及 theta 的距離。

```
sigma1 = s[0]
sigma2 = s[1]
theta1 = np.arccos(sigma1)
theta2 = np.arccos(sigma2)
theta = np.array([[theta1, 0],[0, theta2]])
theta_distance = np.linalg.norm(theta)**2
```

Figure 31 黎曼度量之論文參考

where $\Theta = diag(\theta_1, \theta_2)$ is a 2×2 diagonal matrix formed by the *principal angles* between $\mathcal{U}_1$ and $\mathcal{U}_2$, while the matrix M is given by the formula $M = (I_n - U_1 U_1^T) U_2 F$, with F being the pseudoinverse $diag(\sin(\theta_1), \sin(\theta_2))$. The Riemannian distance between $\mathcal{U}_1$ and $\mathcal{U}_2$ is given by

$$d_{\mathcal{G}}^2(\mathcal{U}_1, \mathcal{U}_2) = \|\Theta\|_F^2. \quad (5)$$

Figure 32 計算黎曼距離之程式實現

theta1 和 theta2 分別代表 u1 和 u2 的奇異值向量中的第一項和第二項，theta 是由 theta1 和 theta2 組成的一個矩陣。具體來說，theta 矩陣描述了將一個向量旋轉到另一個向量所需的旋轉角度。theta1 和 theta2 是 u1 和 u2 的奇異值分解中的奇異值，它們分別表示兩個向量的長度和方向的差異。通過將這兩個值放入一個矩陣中，就可以描述旋轉兩個向量所需的旋轉角度。

```
theta_distance = np.linalg.norm(theta)**2
```

$$||A||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

Figure 33 倆子向量間的相似性公式

theta distance 是 theta 矩陣的 Frobenius 範數的平方，是矩陣元素平方和的平方根表示了 theta 矩陣的大小，也表示了 theta 矩陣內的兩個向量的旋轉角的幾何意義，即可以用來衡量两个向量之间的相似性。

4. 與傳統的余弦相似度計算相比，黎曼度量距離具有以下優勢：

1. 考慮了向量的方向：傳統的余弦相似度只考慮了向量的大小，而沒有考慮方向。而這個算法考慮了向量的方向，因此更能準確地反映向量之間的相似度。
2. 能夠處理長度不同的向量：由於在計算余弦相似度時只考慮了向量的大小，因此無法處理長度不同的向量。而這個算法通過將向量轉換為極坐標形式，從而能夠處理長度不同的向量。
3. 能夠處理負向量：在傳統的余弦相似度中，如果兩個向量之間的夾角為鈍角或者是直角，那麼它們的余弦相似度就是負數。而這個算法能夠處理負向量，因此能夠更全面地反映向量之間的相似度。

黎曼度量距離可以用於計算人臉表情的相似度，原因在於它使用了基於形狀和紋理特徵的相似度計算方法。人臉表情的相似度可以通過比較它們的形狀和紋理特徵來進行計算，這正是該算法所涉及的內容。在該算法中，使用了基於黎曼度量的距離計算公式，這是一種非常有效的方法，可以捕捉到形狀和

紋理特徵之間的複雜關係，從而提高相似度計算的準確性。

4. 距離轉換成相似度

1）歐幾里德距離：

要將相似度鎖在 0-1 之間： 初設方法： similarity = 1/1+d

問題：這個方法非常簡單粗暴，但我們發現了以下問題：

1. 距離加大之後相似度的變化過於微小，在結果上給玩家不夠直觀的表現，所以用線型函數比較直觀
2. 距離在很小之後，x 要在趨近於 0 距離才可能有高相似度，考慮到 r 人臉之間的差異，我們需要設定一個寬限值
3. 這個函數的特性使得我們最終的結果容易徘徊在中間值（也就是 60%-40%之間）

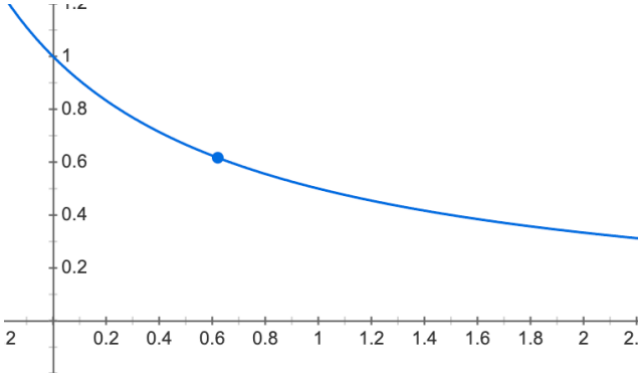


Figure 34 similarity = 1/1+d 函數視覺化

解決問題：
我們自定義兩個個距離：
當 d<0.2, 相似度設為 1
當 d>0.9, 相似度為 0

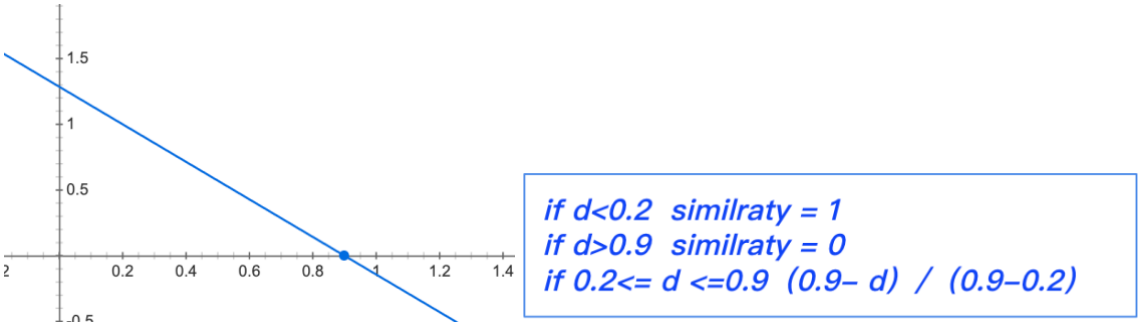


Figure 35 限制範圍後 similarity = 1/1+d 函數

2）餘弦距離：

與歐式距離有著相似的問題，它是 x 要在趨近於 0 距離才可能有高相似度，而差距較大的時候才等於 0

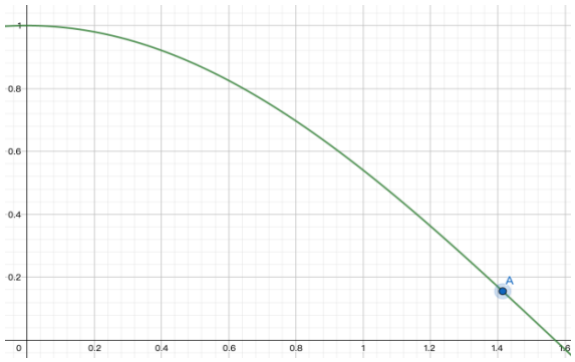


Figure 36 餘弦距離視覺化

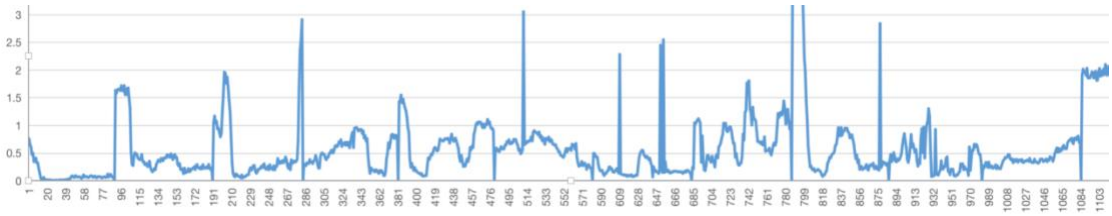


Figure 37 餘弦距離相似度顯示幅度（横軸：實驗次數，縱軸：餘弦距離）

通過實驗數據發現，不同人臉存在大概 0.2-0.3 的差距，所以 d<0.2,相似度設為 1。而餘弦距離的趨近於零的距離是相對合理的，實驗數據得知超過 1.5 就已經差距很大了，所以 d>1.5 時，相似度等於 0。

最終相似度：取中位數：

取中位數的方法是為了將三種距離度量綜合起來，得到更全面的相似度評估。其中，歐幾里得距離和余弦距離都可以被轉化為黎曼度量距離，但它們的計算方式和物理意義不同。因此，綜合考慮這三種距離度量能夠更全面地反映出人臉表情相似度。

第四步：運用高斯濾波穩定結果

1）發現問題：

雖然相似度的結果非常精準，但是變換過於快速，每次相似度的計算都是按視頻禎數來顯示的，加上面部特徵點的效能，攝像頭的高速畫面，使得結果摻雜著雜訊，導致結果非常不穩定。

2）解決問題：

影像處理的領域中，存在許多不同種類的濾波器，我們需要將結果平滑化處理，濾除雜訊，於是有

A. 均值濾波器

B. 中值濾波器

C. 高斯濾波(Gaussian Filter)

最後使用高斯濾波的結果最穩定：

A. 采樣

采樣上我們使用最近的三個 similarity:

```
self_last_similarity_0 = 0 // 本次的 similarity
```

```
self_last_similarity_1 = 0 // 上次的 similarity
self_last_similarity_2 = 0 // 上上次的 similarity
```

B. 計算權重

之後采用高斯濾波中的高斯方程的計算方法來計算在高斯分佈（常態分佈）曲綫中每次采樣所應該占有的比重。

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/(2\sigma^2)}$$

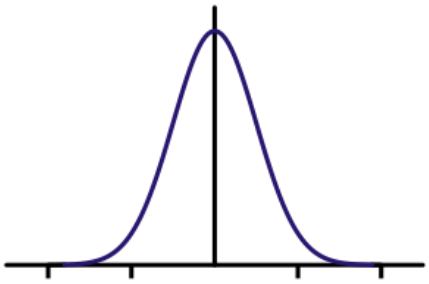


Figure 38 高斯濾波公式及示意圖

在這個基礎的一維情況下的高斯方程中，我們將 deviation 設置爲 1，從而方便計算。并且以 similarity 產生的時間作爲其權重分佈的依據，因此 x 軸的意義就是每個 similarity 距離本次計算的 similarity 的時間差。

具體來說：

X0 = 0(當次，處於高斯分佈中心)

X1 = -1（上一次）

X2 = -2（上上次）

以此方法，帶入公式計算后得到對應的值在做加總，並以此作爲分母計算每個對應 similarity 的權重，以達到權重之和為 1 的效果。計算結果轉換成矩陣后如下：

```
gauss filter = [7/74, 26/74, 41/74]
```

之後，便使用這個矩陣在每次算出新的 similarity 后與新形成的 similarity 的 matrix 進行點乘，從而減少極端情況造成的雜訊的干擾。

```
# Gauss filter
gauss_filter = [7/74, 26/74, 41/74]
self.last_distance_2 = self.last_distance_1
self.last_distance_1 = self.last_distance_0
self.last_distance_0 = squared_euclidean_distance
squared_euclidean_distance = np.dot(gauss_filter, [self.last_distance_2, self.last_distance_1,
self.last_distance_0])
```

Figure 39 高斯濾波程式碼

以下就是經過高斯濾波處理前後的對比圖(淺藍色為處理前)

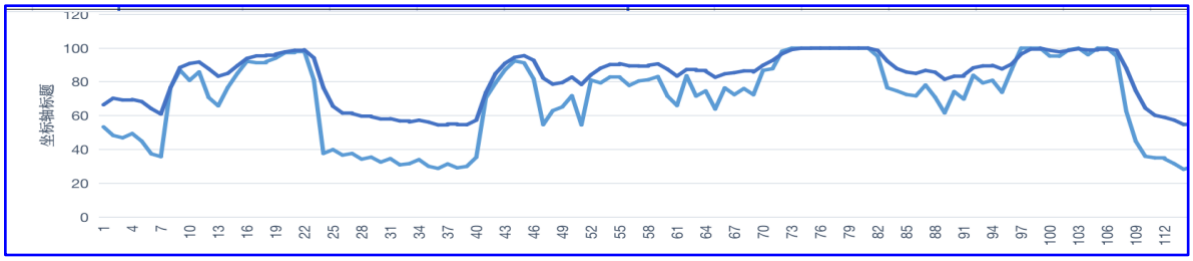


Figure 40 高斯濾波處理前後對比，淺藍為處理前，深藍為處理後

3. 遊戲內容的呈現

介面展示

1. 開始介面：

Figure 41 開始介面

2. 選擇模式：可選擇單人模式或雙人模式

Figure 42 單雙人選擇介面

3. 單人模式選擇關卡難易度:

遊戲關卡選擇介面:

可在此選擇關卡難易度

level 1: 每張圖片 10 秒，共 10 張圖片。

level 2: 每張圖片 7 秒，共 14 張圖片。

level 3: 每張圖片 4 秒，共 25 張圖片。

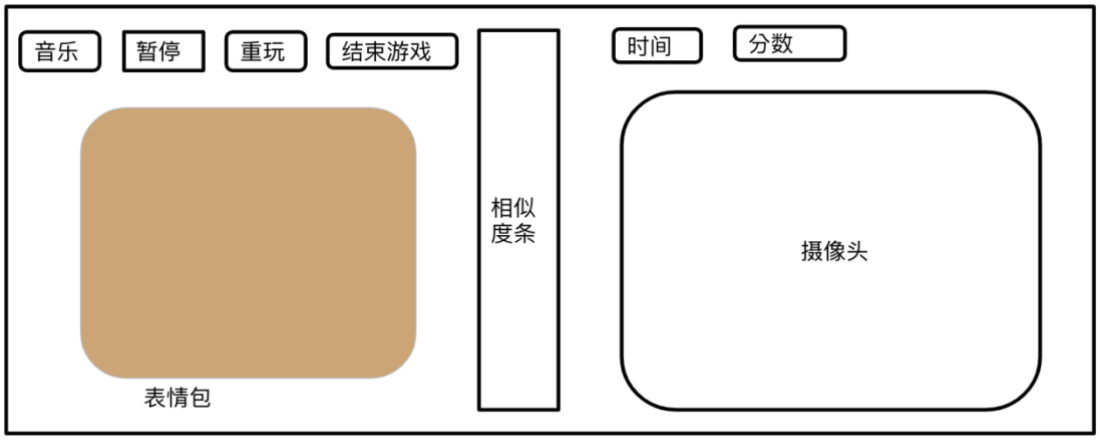
Figure 43 關卡難度選擇

4. 遊戲介面:

左側隨機出現表情包或其他玩家，並根據玩家於右側攝影機顯示的表情進行對比後評分，越接近左側情緒分數越高。

- 隨機出現表情包
- 音樂按一下繼續音樂
- 重玩回到開始頁面
- 結束遊戲去到結束頁面
- 時間顯示倒計時
- 暫停鍵暫停音樂

單人模式:



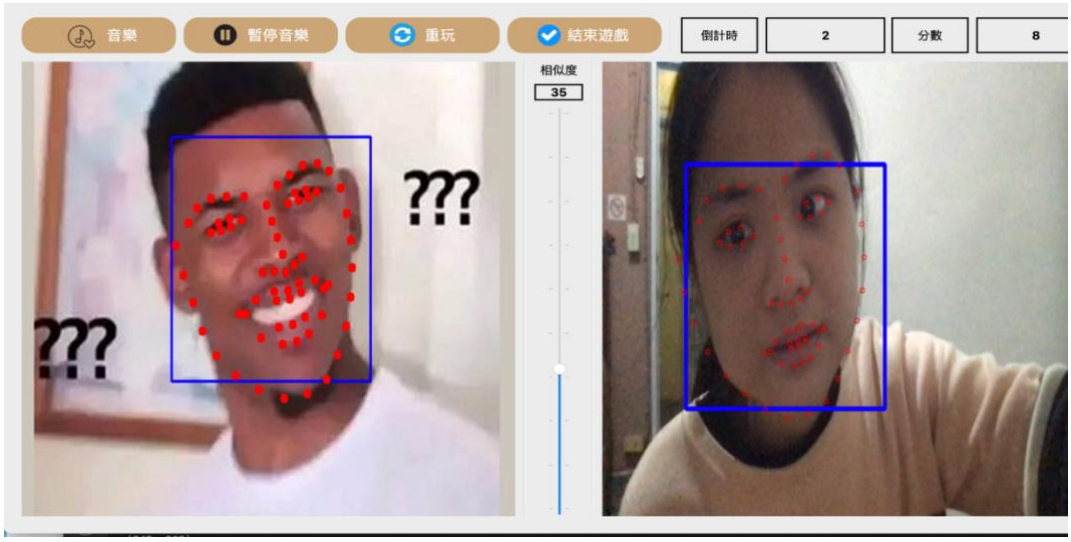


Figure 44 單人模式介面

雙人模式:

兩邊玩家有 15s 做表情，15s 結束後比對相似度並給出分數

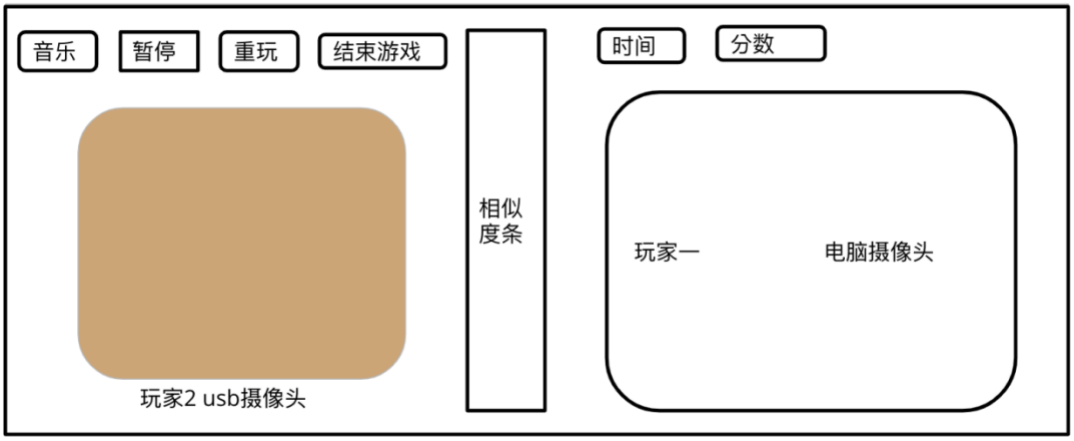


Figure 45 雙人模式介面

雙人模式左邊沒有檢測到攝像頭的話，將以：no_signal.jpg 代替畫面

分數規則：

根據表情的匹配程度給分：

匹配度達 80%及以上為 perfect，perfect 分數為 10 分；

匹配度達 60～80%為 excellent，excellent 分數為 7 分；

匹配度達 30～60%為 good，good 分數為 4 分；匹配度小於 30%為 loss，loss 沒有分數。

5. 遊戲結束介面：在此顯示最終所獲得的分數。

Figure 46 結束介面

表情包:

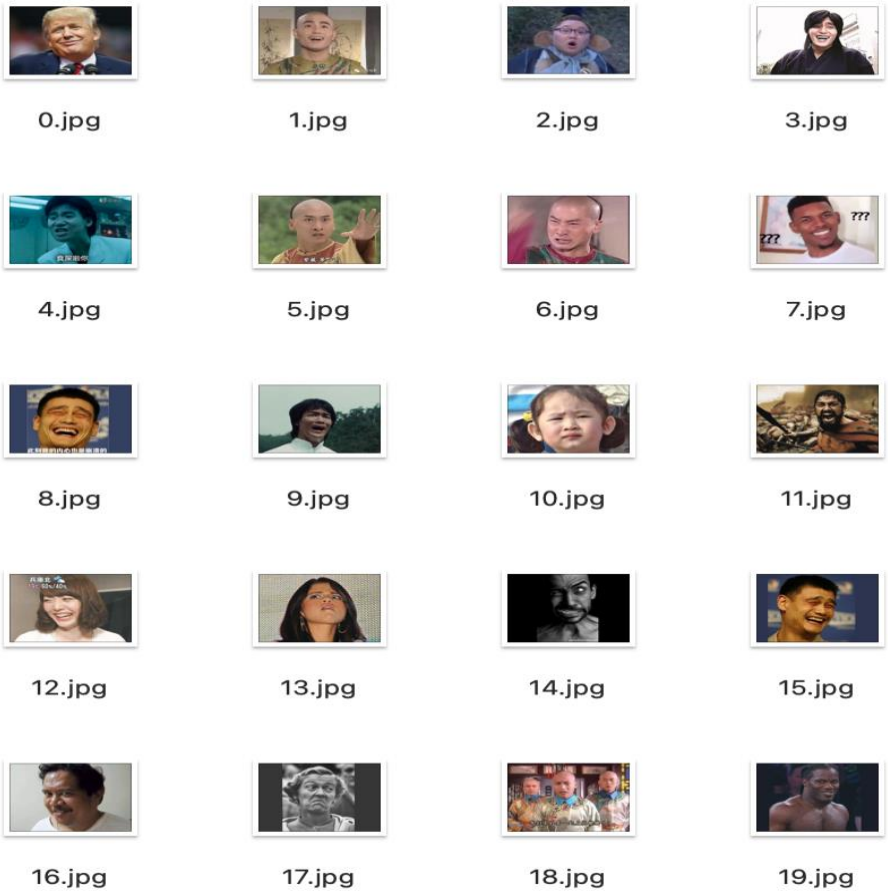


Figure 47 表情包列表

六、結論

1. 開發難點

1） 實時畫出關鍵點的問題
2） 如何消除人臉差異問題
3） 對齊步驟的演算法：缩放矩陣和角度對齊如何统一
4） 相似度的計算公式：黎曼幾何的數學設計
5）如何平滑化處理結果：高斯濾波器的數值

2. 未來展望

在專題的製作與學習當中，我們學到了表情識別的方法。我們透過 dlib 來抓取人臉特徵點，並且對這些特徵點進行操作和計算。雖然說現階段我們對動作幅度大一些的表情可以有結果出來，可是對於一些微表情和動作幅度不怎麼大的表情，測試出來的結果還是不是那麼的準確。

在查閱論文和查找相關研究後，我們了解到的人臉表情識別技術運用範圍比較侷限，基本上只能識別出較為誇張的面部表情，但我們的成果對於微表情的識別範圍極其有限。

希望未來人臉識別技術在微表情上面能有新的突破。那這樣未來人臉表情識別能應用到更加廣泛的領域，也能給人們的工作和生活帶來更大的便利。

3. 心得

這次的專題製作，讓我們了解到現在人臉識別這類的一系列豐富成果及這些成果所應用到的技術。一開始這些技術都讓還未深度了解過人臉識別的我們感到十分新鮮。也激勵著我們想要去了解這些有趣的成果到底是怎樣做出來的。

我們是先通過 OpenCV 這個技術來接觸人臉識別的。並且在一步步探索與學習裡，我們了解到現在人臉表情識別大多都是通過標籤來定義該表情是開心還是傷心；或是直接人為規定這個表情的數值。在某些情況下就顯得有些死板，我們覺得這樣比較無法展示出表情的豐富性。

通過組內討論，我們決定透過數學方法，來確定兩個表情的相似度。取消了原本表情庫裡的 emoji 表情，改為不被標籤定義死的、網路上面常見的表情包，增加了遊戲的趣味性。通過計算表情包和玩家表情的相似度，來給出一定的分數。但這個方向性的改變，也相對應增加了演算法設計的難度。

在解決兩個表情相似度方面，我們先是通過 dlib 獲取人臉關鍵點，接著做了仿射變換來對齊表情包和玩家表情，然後通過歐幾里得、餘弦距離、黎曼度量距離來計算出相似度。在這過程我們也學習到很多在影像處理上面會用到的數學，更加深入地了解數學思維在影像處理方面的重要性。

雖然專題中途也頻繁出現一些問題，但經過組員的互相幫忙，問題也被一一解決。在這過程裡，大家也對自己做的專題有各自的體會，對應用到的技術也有更深入的理解。

七、海報

Figure 9 相似度對比流程圖

Figure 48 海報

組員分工

許儒怡 ： 前端撰寫、算法研究

周卓盈：遊戲設計、海報製作

李承謙：前端測試、套件測試

陳燮涵：算法研究、前端測試

林伯翰：查找資料、套件學習

王靖雄：查找資料、投影片製作

八、參考文獻

1. Anis Kacem, Mohamed Daoud, Boulbaba Ben Amor, Juan Carlos Alvarez Paiva, A Novel Space-Time Representation on the Positive Semidefinite Cone for Facial Expression Recognition, DOI:10.1109/ICCV.2017.345
2. Mo, S., & Huang, Y. (2019, January 23). *2018Fall-FaceDanceMachine*. GitHub. https://github.com/NTUEE-ESLab/2018Fall-FaceDanceMachine
3. Cheng, C. (2020, February 9). 歐氏距離與餘弦相似度的比較. Medium. https://medium.com/qiubingcheng/歐氏距離與餘弦相似度的比較-c78163ad51b
4. 手写 ai. (2021, August 9). *人脸识别中的对齐技术，相似变换 Similarity Transformation*. Zhihu. https://zhuanlan.zhihu.com/p/393037076
5. 3blue1brown. (2016, August 8). *Linear Transformations and Matrices Chapter 3, Essence of Linear Algebra*. YouTube. https://www.youtube.com/watch?v=kYB8IZa5AuE
6. OpenCV 仿射变换：https://docs.opencv.org/4.x/da/d54/
7. Huang, Z., Wang, R., Shan, S., & Chen, X. (2015). Projection Metric Learning on Grossmann Manifold with Application to Video Based Face Recognition. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://openaccess.thecvf.com/content_cvpr_2015/papers/Huang_Projection_Metric_Learning_2015_CVPR_paper.pdf