基於行動裝置的眼動儀

Smartphone Eye Tracker

F84086171 黃盈盛

以下將分為四個部分介紹此專題的設計與實作內容說明，第一部分為特徵擷取的實作，第二部分為深度學習模型的主要部分，第三部分為 App 的實作，第四部分為 Django server(用於接收 App 傳來的資料並進行解析、預測，而後回傳結果給 App)

第一部分:

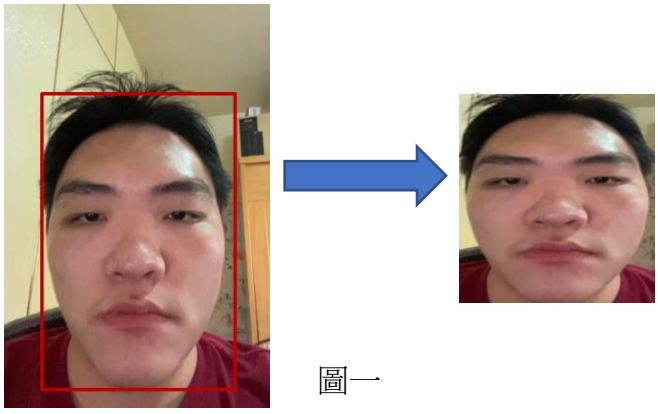
作為前置準備，我們需要到 MIT CSAI Lab Eye Tracking for Everyone 向官方取的 dataset 下載權限，下載後用他們 Github 的 code 先用大量資料訓練出一個 baseline model 才能繼續接下來的動作

接著我們需要一個獲得特徵的方法，此模型共需要四個特徵

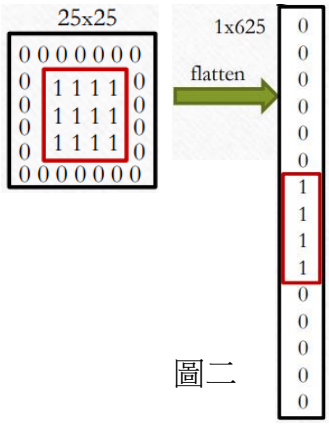
1. face image (224x224)
2. left eye image (224x224)
3. right eye image (224x224)
4. facegrid (1x625)

這邊我使用了 OpenCV 的 facedetector caffemodel，使用此模型可以簡單的做到以下效果

紅框為 facedetector 辨識出的人臉 boundingbox，此框可以獲得兩種 Feature，一為 face image，二為 facegrid，face image 只需把紅框內的圖片擷取出來(如下圖一)而 face grid 則是先用 numpy 創建一個與照片同大小的 array，然後依據紅框，將框外放 0，框內放 1，然後 resize 成 25x25 再 reshape 成 1x625(如下圖二)

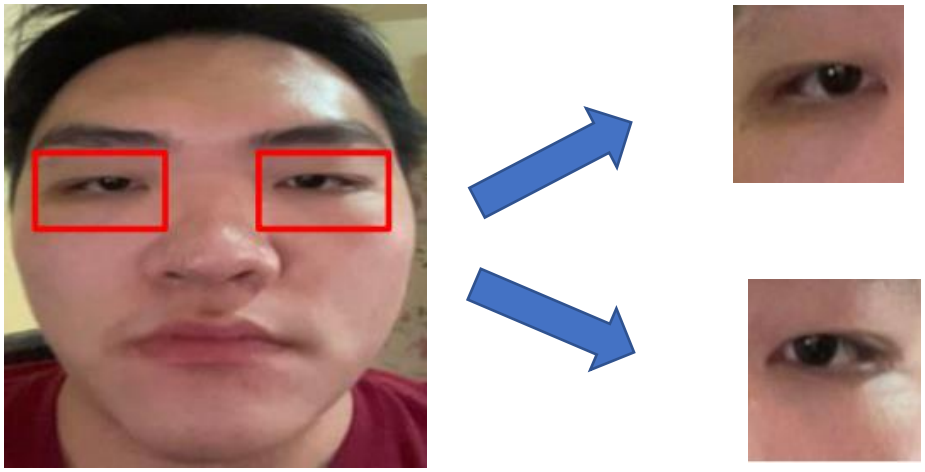


圖一



圖二

左眼以及右眼的部分我則沒有進行額外的 detection 而是直接用固定座標的 boundingbox 框住，並擷取(如下圖)因為大多數人的五官位置都是差不多的，儘管會有微小差異，但通常不會有大到足以影響結果的差異



擷取出來後，如同 face image 一樣，left eye image 與 right eye image 也需 resize 成 224x224 才能順利放到模型中訓練、預測

如此就完成特徵擷取部份了，最後將每張圖片擷取出的 Feature 儲存下來作為後續 transfer learning 的 training data 與 testing data 使用

第二部分:

Pytorch Dataset 製作:透過繼承 Pytorch 的 Dataset 套件並實作相關 method 可以製作自己的 Dataset，在製作 dataset 時需要將所有資料前處理完成，以供模型訓練，以下為此專題有用到的前處理

1. 用 Pillow 套件讀入圖片，然後將圖片做正規化(使用 MIT 提供之 face_mean、leftEye_mean、rightEye_mean)正規化方法只要簡單地將圖片 array 與對應的 mean 相減即可完成正規化

```
imFace = self.loadImage(imFacePath)
imEyeL = self.loadImage(imEyeLPath)
imEyeR = self.loadImage(imEyeRPath)
```

```
imFace = self.transformFace(imFace)
imEyeL = self.transformEyeL(imEyeL)
imEyeR = self.transformEyeR(imEyeR)
```

Transform 為正規化 method(此 method 回傳值已轉為 tensor 後續無須再轉)

2. 將 Label 根據與相機的相對位置做計算後得到 training label

```
gaze = ((258 - self.label[index][0])/60,(18 - self.label[index][1])/60)
```

3. 將所有處理好的資料包成 torch 形式才能使用 pytorch 模型訓練

```
# to tensor
faceGrid = torch.FloatTensor(faceGrid)
gaze = torch.FloatTensor(gaze)
```

接下來是使用 dataloader 將前述的 dataset 的資料 load 進程式使用，格式是一個多維 tuple 陣列(維度取決於 dataset 中回傳的維度，此處為 5，包含四個特徵以及 gaze label)，以 for 迴圈把資料餵給模型訓練

```
train_loader = torch.utils.data.DataLoader(
    trainData,
    batch_size=batch_size, shuffle=True,
    num_workers=workers, pin_memory=True)
```

```
for imFace, imEyeL, imEyeR, faceGrid, gaze in train_loader:
```

再來就是 Transfer learning 的部分，首先我們會需要一個由 MIT 的資料以及其模型訓練出來的 baseline model，此為前置部分，非本專題主要內容，不

多贅述，接著將模型 load 到程式中

```
model = ITrackerModel()

model.load_state_dict(torch.load("model"))
```

此處的”model”即為 baseline model，然後依照訓練 baseline model 時的原始設定(如 Optimizer、Loss function 等)接著，依照前述 dataloader load 進程式的資料，以 for 迴圈做深度學習模型的訓練

```
output = model(imFace, imEyeL, imEyeR, faceGrid)

loss = criterion(output, gaze)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

至此便完成了 transfer learning 的部分，

但是，單單的 transfer learning 是不夠的，依據我的實驗結果，baseline model 的誤差約為 7~10 cm 即使做完了 transfer learning 依然還有 3~4 cm 的誤差，這樣的誤差是無法作使用的，因此針對 transfer learning 的結果還需要搭配一個校正模型作使用才能再進一步縮小誤差

校正模型的訓練:

校正模型的使用參考 Google Eye Tracker 的說法使用 ML model，並且因為資料為數值型資料，需使用 regressor 類(如 RandomForestRegressor、SVR 等)的 ML model 作使用，這裡我先後嘗試了三種不同的 model 分別是

1. LinearRegression:

   最先嘗試使用的模型，非常簡單的 linear regressor，實際效果也並不是非常好，有 2~4cm 誤差，雖然較原始 transfer learning 結果再好一點，但仍無法使用

2. SVR:

既 LinearRegression 後嘗試使用的模型，原理與 SVM 相同，為數值版 SVM，結果為 1.5~3.7 誤差，比 linear regressor 好一點，但是仍不夠精準，無法使用

3. RandomForestRegressor:

最後嘗試的模型，RandomForestRegressor 為一種 ensemble 的 regressor，是因為正好這學期有修的課程有教到，抱著嘗試的心態，我使用了 RandomForestRegressor 作為校正模型，結果意外的很好，順利將誤差壓到 1.2cm 以下(大部分為 0.5~1cm 誤差)因此這也是最後採用的模型

用於訓練校正模型的資料取得:

透過前述的訓練方法，但是只留下這行，剩下四行刪除，意即不更新深度學習的 Neuron 的參數，只獲得 output，並用 list 不斷 extend list 內資料，去紀錄所有資料的預測結果以及其正確 gaze label(如下圖所示)

```
output = model(imFace, imEyeL, imEyeR, faceGrid)

output = output.detach().cpu().numpy()
gaze = gaze.detach().cpu().numpy()

outputs.extend(output)
gazes.extend(gaze)
```

有了所有預測結果及實際 gaze label 的 array 後便可開始訓練 ML model，而 ML model 並不像 deep learning model 一樣，可以透過設定 output 的 Neuron 數達到一次輸出兩個結果的效果，ML 中 Feature 可以很多，但是結果只能有一個，所以 calibration model 的部分需要拆成預測 x 與預測 y 的 model 來做，

這邊選用上述提到的 RandomForestRegressor 來做為模型，而資料的部分，outputs 與 gazes 皆為 n*2 的陣列(如[[0,0],[10,10]])，為了能分離 x 與 y 這裡需要先將 list 轉換為 numpy array(如下圖)

```
outputs = np.array(outputs)
gazes = np.array(gazes)
```

接著就可以利用 numpy array 的用法將 axis=0 的部分分離開來(如下圖)

```
clfx.fit(outputs,gazes[:,[0]])
clfy.fit(outputs,gazes[:,[1]])
```

clfx 與 clfy 分別代表預測 x 與 y 的 model，使用 fit 來訓練模型，以 outputs 做為 feature，再分別以 gazes 的 x 部分、y 部分做為訓練資料，如此便可訓練出一個校正模型

接著就是挑選儲存模型的部分了，眾所周知，deep learning model 存在一種名為 overfitting 的問題，導致模型不夠 general，訓練太多個 epoch 便會很容易產生這種狀況，此時，就會需要 testing data 協助，正常情況應該是由 training data 分一部份資料做 validation，如 5-Fold validation，但考慮到這裡的資料比較特殊，通常一個 gaze 會有數十甚至數百張照片，並且因為取樣時間極短，基本不會有什麼太大的變化，如果在這種情況下仍使用類似於 5-Fold validation 的方法會導致取樣到做為 validation 的 data，training data 仍有一份甚至好幾份幾乎一樣的資料，這樣會使 validation 的功能完全喪失，因為 Error 幾乎都會趨近於 0，因此我這裡是直接拿 testing data 做為驗證，因為 testing data 的蒐集是獨立於 training data 的，因此可以避免資料太相近的情況，接著，用之前提到的 dataset，製作一個 test 的 dataset，並且用前面產生 calibration data 的方法，銜接前一步驟訓練出的 calibration model 去做出一連串的預測，並計算每個預測點的歐式距離，最後取平均做為此輪訓練的平均 testing error，紀錄每個 epoch 的 testing error 的變化，只有當當下的 testing error 小於所有之前的 testing error，

模型才會被儲存，如此一來，模型會被保留再 testing error 最低的狀態，避免 overfitting 的發生

儲存歷史訓練狀況:

為了能方便訓練後能夠針對不同的參數調整結果作分析，在每一輪 testing error 產生後我都會將 error 數值紀錄幾來，並儲存在 log 資料夾內，名稱為時間，以記錄每次訓練的 testing error 走向，分析改變不同參數對於模型訓練狀況、最終表現的影響

第三部分:

App 的部分我實作的是 ios app，在實作 ios app 時有個硬性條件必須滿足才能實作:一台能裝 Xcode 的 macOS 裝置，因為 ios app 必須經過 Xcode 的驗證才能佈署到手機上做測試

App 實作內容:

1. 前鏡頭畫面:

   首先需要調用前鏡頭才能滿足我們要擷取面部圖片得需求，這邊我使用的是 swift 中 ARKit 套件提供的前鏡頭調用方法，再使用 ARSCNView 呈現在螢幕上，因為在實際使用上，為了避免顯示前鏡頭畫面導致眼動測試點顯示範圍遭到限縮，因此，前鏡頭畫面是會被測試點圖層蓋住的，但是實際還是有顯示，只是看不到(這點非常重要，因為後面會需要調用 ARSCNView 中的 snapshot method，如果並沒有正常顯示，而是直接不顯示的話 snapshot 的結果就不會是面部截圖，而是一張全黑圖片)

2. 製作測試畫面:

   根據 Goolge Eye tracker 的說明，他們在校正階段使用了以黑色為底，以綠點作為注視點，以畫圓的方式一次顯示一個綠點，因此，我仿照 Google 提出的校正畫面作了一個一樣的出來，只是此階段是一次顯示所有綠點，不會變化，這部分比較沒什麼難度，就是一些基礎 App layout 的畫面配置而已

3. 測試點移動:

由第二點中我們已經完成了測試畫面中，綠點位置的配置，接下來需要的是讓他們依次出現，一次出現一個點，才能透過使用者觀察某點的圖片，搭配該點座標來取得 data 與 label，在這裡，我調用一個叫 timer 的函式，其作用為每隔一段指定的時間便發出一次 timer interrupt，並執行對應的 handler，這裡我將 timer interrupt 的時間設為最短的 0.0333(30 frame/second)，並使用一個 counter，去紀錄 interrupt 發生的次數，並在固定間隔改變一次顯示點座標，比如我想 2 秒改變一次點位置，我可以將 counter % 120，如果為 0，則改變點位置，所有點位置則被儲存在一個 array 中，利用(counter/間隔)%array_size 做為 index 後取出座標點並顯示於畫面上即可達成每點依序出現的效果

4. 畫面 snapshot:

前面有提到我透過 timer interrupt 定時讓 App 去 call 一個 handler，在這個 handler 中我跟前述綠點一樣都是給定一個時間區間配合 counter 定時 call ARSCNView 的 snapshot，這個 method 會將畫面內容截圖，並回傳 UIImage 資料(Swift 中的圖片 class)，因為是擷取由 ARSCNView 所顯示的 scene，並非是前鏡頭拍照，而是將顯示的前鏡頭畫面截圖下來，因此前面才會說雖然我們看不到，但有沒有實際顯示是至關重要的，否則螢幕截圖就截不到任何東西

5. 傳圖片給 server:

這邊會需要一些 Web server 的相關知識，我這裡是參考了別人的程式碼後實作出來的，大意是我們會需要將 UIImage 其名為 jpegData 的 method，這個 method 可以將 UIImage 轉為 jpegData 格式，如此便能包成 json 得行使用 swift 的 URLSession post 到 server 端，並且這個 jpegData 在 post 到 server 端後可以很簡單的透過 Python 的 Pillow 套件 load 進程式中(注意，此處的 UIImage 在傳輸前須先進行 resize，傳輸的圖片過大會造成傳

輸速度很慢，且沒有意義，因為到 server 端後一樣是要作 resize、clipping 的，降低圖片大小才能增加傳輸速度)

6. 預測點顯示

在將圖片傳送到 server 端後，因為使用的是 post method，因此在傳資料過去的同時也會收到一個回傳的 json，收到的回傳值即為 server 端在預測後得出的結果包成的 json 格式，client 端收到後需要先對收到的資料進行解析，swift 的處理方式比較特殊，我們需要先準備一個與回傳內容對應的 struct，比如回傳值為{ 'x' : 10.0, 'y' : 20.0}，我們會需要一個 struct 具有名為 x 與 y 的成員變數，並指定型別(如下所示)

```
struct data{
    x:CGFloat
    y:CGFloat
}
```

有了這個 struct 後我們可以透過 JsonDetector.decode，參數為這個 struct 以及從 server 端獲取的資料，這個 decoder 會自動將 json 檔中名稱與提供的 struct 的名稱對應，並賦值，這樣就完成了 json 的 parse，最後，只需要透過一個控制點位置的 global variable，更改其 x,y 值便可以達到根據預測結果顯示在螢幕上的效果了(預測結果以紅點顯示在螢幕上)

第四部分:

此部分將講述 django server 的運作方式，與模型的訓練本身沒有關係，但是為在實驗中獲取我的個人臉部資料，以及後續 App 在進行預測時很重要的一部分

利用 python 的 django 套件可以簡單的建立一個模板 server，可以參考以下連結教學: https://ithelp.ithome.com.tw/articles/10233760，建立完模板伺服器後要建立一個傳輸 image 的 http 接口，而 python 有非常簡單的獲得圖片機制，可以直接將圖片由 in memory data(從 client 端傳來的資料)，直

接用 Pillow 套件中 Image class 提供的 open method 直接打開就能當作圖片去操作了，在這樣的操作下我可以很簡單的分成兩個我要實作的功能:

1. 直接用 save method 將圖片儲存起來，再用第一及第二部分的內容將圖片用作 transfer learning，這個是在資料蒐集以及訓練階段會去使用的功能，一般的預測不會需要存圖，避免存圖佔據資源，導致預測速度減慢
2. 直接用於預測，前面有說到傳輸到 server 端的資料可以直接轉為 Pillow 的圖片格式:PIL Image，而前面在 dataset 處理資料的部分也是根據 PIL Image 做處理的，因此，我只需要照般 dataset 的前處理方法，並應用在我由 client 端收到的圖片上，便可以直接放入模型預測，並得到結果，最後再將結果用 Json 的方式回傳給 client 端，讓 App 把收到的座標顯示在螢幕畫面上

Server 的部分並不是絕對需要，是找不到將整套預測方法放上 ios 系統的方法的一個妥協點，但實際結果也還能接受，並且在蒐集圖片上的方便程度非常高(另外，在使用上請確保 client 及 server 連在同一網路上，或是 server 為公開 IP，否則會發生無法連上線的情況)