**臺北科技大學資訊工程系**

**109學年度實務專題計畫成果報告**

**拍照點名系統**

專題編號：109-CSIE-S022

專題計劃參與人員： 106820003 潘建蒼   
 106820007 段凱文

106820028 陳品睿  
指導教授：陳彥霖教授

執行期間：108年1學期至109年1學期

[**摘要 2**](#_Toc53753239)

[**第一章 緒論 2**](#_Toc53753240)

[**1.1 研究背景 2**](#_Toc53753241)

[**1.2 研究動機 2**](#_Toc53753242)

[**1.3 研究目的 2**](#_Toc53753243)

[**1.4 研究流程 2**](#_Toc53753244)

[**第二章 相關文獻回顧 3**](#_Toc53753245)

[**2.1 背景資訊 3**](#_Toc53753246)

[**2.2 Android 系統架構 3**](#_Toc53753247)

[**2.3 Server端資料庫系統 （使用 MongoDB） 5**](#_Toc53753248)

[**2.4 Server系統(使用 Node.js架構) 7**](#_Toc53753249)

[**2.5 Retrofit 7**](#_Toc53753250)

[**第三章 研究方法 8**](#_Toc53753251)

[**3.1 研究方法 8**](#_Toc53753252)

[**3.2 方法架構 8**](#_Toc53753253)

[**3.2.1 Android與伺服器資料傳輸之實作方法 8**](#_Toc53753254)

[**3.2.2 Server 9**](#_Toc53753255)

[**3.2.3 User（Client） 9**](#_Toc53753256)

[**3.3 人臉偵測及辨識方法 12**](#_Toc53753257)

[**3.3.1. Google ML 模型 12**](#_Toc53753258)

[**3.3.2. TensorFlow Lite 模型 13**](#_Toc53753259)

[**3.4 系統架構圖 14**](#_Toc53753260)

[**3.5 系統資料流程圖 15**](#_Toc53753261)

[**第四章 研究結果 16**](#_Toc53753262)

[**4.1 開發與測試環境 16**](#_Toc53753263)

[**4.2 伺服器端資料庫系統欄位設計 16**](#_Toc53753264)

[**4.3 應用程式設計 18**](#_Toc53753265)

[**4.4.1. 應用程式之用戶登錄和註冊頁面 18**](#_Toc53753266)

[**4.4.2. 應用程式之教師操作 19**](#_Toc53753267)

[**4.4.3. 應用程式之學生操作 20**](#_Toc53753268)

[**4.4.4. 應用程式之學生人臉偵測畫面 22**](#_Toc53753269)

[**第五章 結論 23**](#_Toc53753270)

[**5.1 模型探討 23**](#_Toc53753271)

[**5.2 未來展望 23**](#_Toc53753272)

[**參考文獻 24**](#_Toc53753273)

[**附錄 26**](#_Toc53753274)

# 摘要

主要是藉由TensorFlow的框架來偵測人臉的特徵，並且藉由Android app來做為教授上課點名的輔助系統，且點名的同時會同步建立資料庫記錄下學生的出缺席狀況，以供日後查詢和評分。

# 緒論

## 研究背景

在電腦科學蓬勃且迅速發展的今日，大數據與機器學習是近來在資訊科學領域進展快速且擁有龐大潛力的一門領域，尤其是在已發展了三十餘年的機器學習，讓電腦能夠透過分析龐大數據資料且利用演算法來進行學習從中找出數據的規律和模式，最後能透過學習的結果來達到對新取得且未知的資料進行分類或預測，其概念自 Fisher 於 1936 年發明的線性判別分析(LDA)直至今日的類神經網路(Neural Network , NN)的演變，已被大量應用於解決許多領域的問題，如資料探勘、醫學影像辨識、機器人產業等，已成為各領域發展中解決各式問題不可或缺的技術。

## 研究動機

最近因為全球性新冠肺炎影響，因此學校也有開始使用數位化上課，而在大班級當中時常因為點名而花費大量的時間，所以想研究看看是否能節省無效率的時間消耗。

## 研究目的

利用人臉辨識的技術，藉由訓練好的Model來辨識每個同學的出缺席的狀況。並且為了每個教授的方便性，期望使用Android app作為媒介來點名。

## 研究流程

以Android Studio製作行動裝置之介面，並使其擁有登入、註冊、上傳照片、查看照片、點名……等功能，再建立一個伺服器用於接收點名紀錄等資料，並能用APP將記錄回傳至行動裝置。

# 相關文獻回顧

## 背景資訊

2020是新冠肺炎肆虐的一年，為了防堵疫情的擴散，各個學校紛紛祭出點名的方式來追蹤學生的出缺席狀況，以利追蹤學生足跡之餘，也有利於校園衛生安全的防護；然而點名之動作是為手動進行的，常常浪費許多時間，因此如何節省時間成為相當重要的課題。

結合現今科技，在行動裝置上提供協助點名的方式，目的是為了協助使用者快速的點名，而非取代傳統的手動點名，達到節省時間的效果。

在電子化資訊領域中，因行動裝置及嵌入式裝置有著便於攜帶、使用的優點，民眾日常活動及隨身攜帶上較為便利，其中普及性高的行動裝置以智慧型手機為主流。

截至目前為止，在 Android Google Play平台上能解決此課題的應用非常稀少，再加上是因應疫情所需，是故需要具備戴口罩之人臉偵測和辨識之能力，在難度上又大大的加深。

人臉辨識是進年被熱烈探討的議題。各式模型如：FaceNet、YOLO、VGG16……應運而生。於是我們希望能藉由這些模型來分析人臉，達到連戴口罩之人臉也能達到偵測和辨識之效果，最終目標是自己訓練出模型，再搭配架設的伺服器和撰寫的手機應用程式來組件出具實際應用的系統，解決此議題。

## Android 系統架構

Android系統是一個基於 Linux 核心的 Open Source行動作業系統，由Google公司在 2007年公布的智慧型手機作業系統，主要用於智慧型手機和平板電腦，是透過 C語言及 C++作為系統與 API、使用 Java語言作為用戶介面的編碼，以圖形化界面為主的操作方式，讓用戶能直覺性地方便操作。如今許多掌上型裝置以智慧型手機最為多數，其中又以 Android作業系統市占約達 82.1%（圖1），由此可見 Android用戶為手機市場大宗。



圖1

Google 將大部份的 Android程式碼以開放原始碼 Apache 授權方式釋出，Android 開放源碼專案主要工作是維護與設計新功能， 整個 Android 以 Linux 的核心做為底層，在這個底層中加上中介函式庫與 API，函式庫大多以 C 語言設計，並在這函式庫裡添入應用程式框架，以 Java 相容的程式撰寫之。 在此框架內才是平時手機上安裝的應用程式， 整體系統架構（圖2）如下圖：

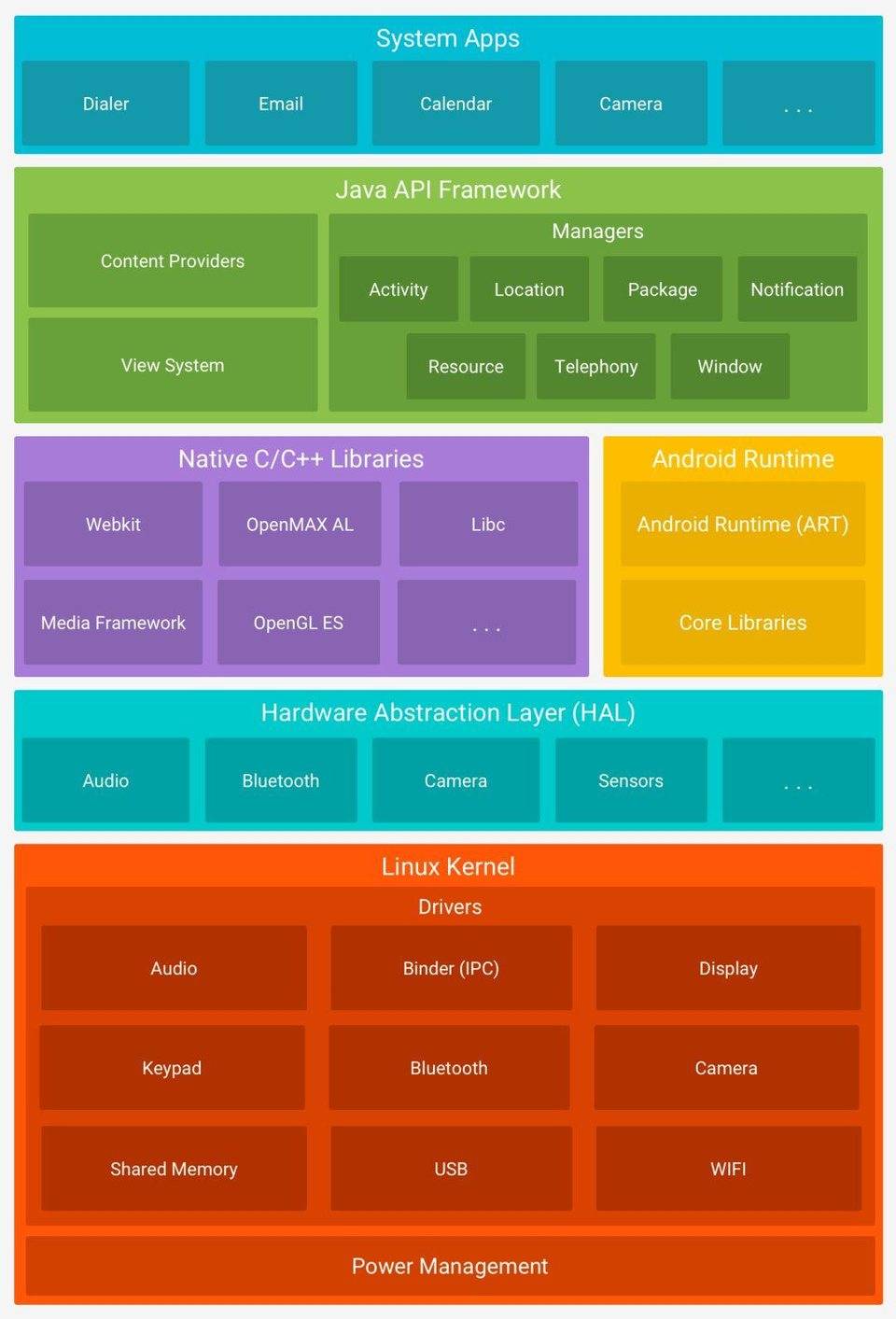


圖2

Android應用程式設計將畫面端程式碼(xml檔，為階層式架構)與執行端程式碼(java檔)分為兩個不同的頁面，分離畫面布局(layout)及後端程式，並且有著分離多語系(string.xml)設計，讓頁面設計師及程式設計師可以將程式分工作業。在應用程式開始執行時會先載入布局檔案再執行接下來的程式。

Android多個布局檔之間的切換，可透過程式內之意圖(Intent)來切換Activity。

## Server端資料庫系統 （使用 MongoDB）

MongoDB是一款為web應用程式和網際網路基礎設施設計的資料庫管理系統。同時也是10gen開發出來的的NoSQL資料庫，而NoSQL是Not Only SQL 的縮寫，即不只是SQL的意思，希望結合SQL優點並混用關聯式資料庫和NoSQL資料庫來達成最佳的儲存效果。

一般我們將NoSQL依其資料模型（圖3）分類為「Key-value Stores」、「Document Databases」、「Graph Databases」三種，

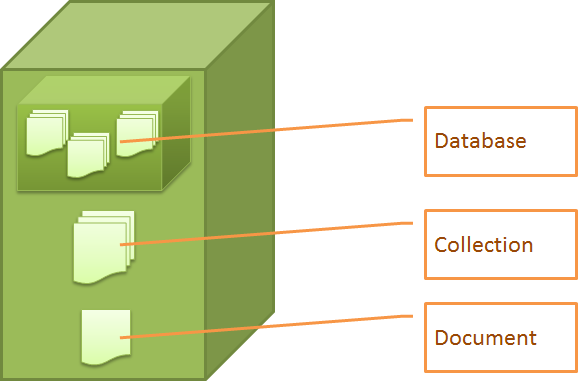


圖3

具有以下優點：

（1）MongoDB提出的是文檔、集合的概念，資料體結構是以 Key,Value組合的，使用BSON（類JSON）（圖4）作為其數據模型結構，其結構是面向對象的而不是二維表，存儲一個用戶在MongoDB中是這樣子的。

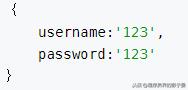


圖4

使用這樣的數據模型，使得MongoDB能在生產環境中提供高讀寫的能力，吞吐量較於mysql等SQL資料庫大大增強。

（2）易伸縮，自動故障轉移。易伸縮指的是提供了分片能力，能對數據集進行分片，數據的存儲壓力分攤給多台伺服器。自動故障轉移是副本集的概念，MongoDB能檢測主節點是否存活，當失活時能自動提升從節點為主節點，達到故障轉移。

（3）數據模型因為是面向對象的，所以可以表示豐富的、有層級的數據結構，比如博客系統中能把「評論」直接懟到「文章「的文檔中，而不必像myqsl一樣創建三張表來描述這樣的關係。

## Server系統(使用 Node.js架構)

Node.js 是一個高效能、易擴充的網站應用程式開發框架 (Web Application Framework)。它誕生的原因，是為了讓開發者能夠更容易開發高延展性的網路服務，不需要經過太多複雜的調校、效能調整及程式修改，就能滿足網路服務在不同發展階段對效能的要求。在眾多後台語言中，選擇 Node.js 的原因有以下幾點：

* 可自己寫 web server
* 高效能「非同步IO」不會延遲等待
* 低耗能(需要時再取得資源)
* 第三方模組支援
* 入門簡單

## Retrofit

是一個RESTful 的 HTTP網路請求框架（圖5）的封装。它並不是一個網路請求框架，主要原因在於網路請求的工作並不是Retrofit来完成的。Retrofit 2.0開始內置 OkHttp，前者專注於接口的封装，后者專注於網路請求的高效，二者分工合作。

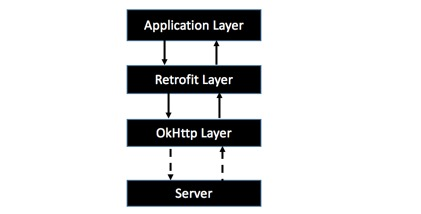


圖5

應用程序通過Retrofit請求網路，實際上是使用Retrofit接口層封裝請求參數、Header、UIrl等信息，之後由OkHttp完成後續的請求操作，在服務端返回數據之後，OkHttp將原始的結果交給Retrofit，後者根據用戶的需求對結果進行解析的過程。所以Retrofit，其實就是Retrofit OkHttp了。

# 研究方法

## 研究方法

由於本專題的核心在於人臉偵測和辨識模型，故一開始先經由模型下手。先至網站查找適合的模型，先大至對於模型的運作有一定的基礎概念後，嘗試自己訓練模型。

在訓練模型期間碰到不少問題，和學長討論過後，有了一點眉目。由於在網路上缺乏戴口罩之人臉資料集，故一開始先從資料的label開始，再將已經label好的資料分別餵給模型，自行訓練，在分析了多組模型後發覺準確度不佳時，在時間的壓力下迫於使用網路現有的模型。

於是乎，便開始思考操作介面要架設在網路還是移動端，最後考量到方便性的問題，決定採用手機當作應用的介面。

在預想到使用者的需求後，將之分為學生和教授，並各自有各自的功能。在座的過程中發現，由於模型需要在手機端進行運算，故須滿足架構簡單和運算量低等條件，故又重新開始尋找合適的模型。在選定模型之後，開始運用和操作。

## 方法架構

### Android與伺服器資料傳輸之實作方法

首先，先建立 MongoDB 資料庫，並且在伺服器端編寫連接 MongoDB資料庫的js 檔。

每個 Android 教師端可以藉由拍攝照片，經由模型分析，將出現在照片中的學生資料，經由伺服器處理，向 MongoDB資料庫插入檔案出，這些檔案為出缺席紀錄文本資料；學生端也可以從 MongoDB資料庫裡獲取該教師插入之出缺席紀錄文本。

學生端可以藉由拍照或是選取本地端的照片經由伺服器進行建檔，再將URL插入MongoDB進行儲存；學生端也可以藉由送出查看照片請求給伺服器，並回傳相應照片學生端（圖6）。 以上所有資料之傳入和傳出皆由json格式來包裝。

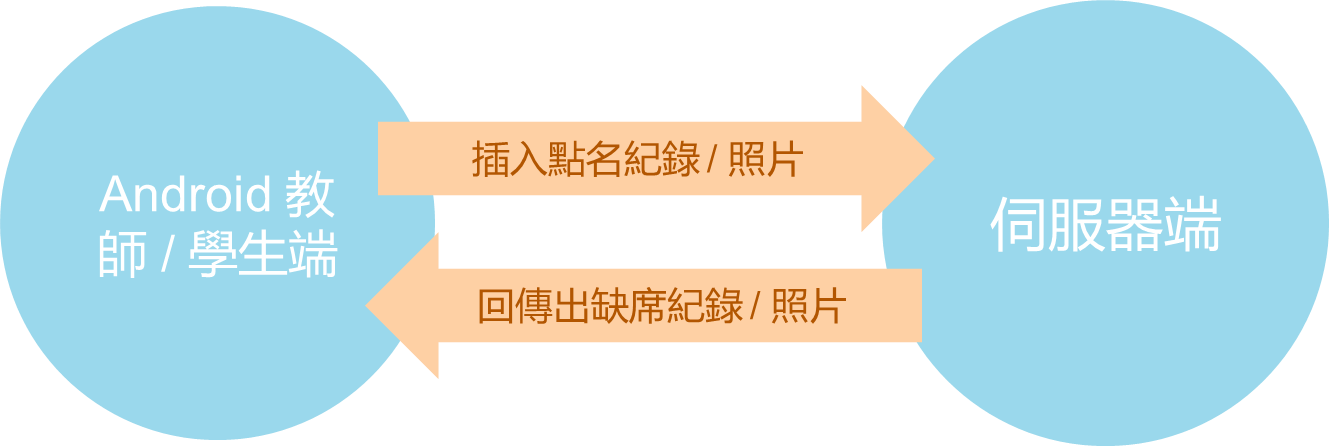


圖6

### Server

使用nodejs+express，當使用者登入時，接收端為app.post(‘/login’, loginFunction(request, response, next))。使用者的登入資訊(email、password)都會存放在request.body，透過nodejs的mongodb模組，我們很可以像在mongodb上直接管理資料庫。使用find()指令查詢資料庫的使用者資訊，並比對email及密碼是否正確。我們把每種結果各用對應的碼及說明存下來，例如若是密碼部正確則為400 + ”Login password error”。最後將結果打包成json格式，使用response.json(myResponse)將結果傳回client端。

當使用者需要上傳圖片時，透過nodejs的multer模組接收圖片並存在伺服器的uploads資料夾。而資料庫中只存取圖片檔名的部分。

當使用者需要接受圖片時，因為圖片是存在伺服器電腦上而非資料庫，所以只要從資料庫中提取圖片名稱，再用express的response.sendFile(${圖片路徑名稱})即可。

### User（Client）

對象為使用此應用程式的所有用戶。

教師可以透過手動點名或是拍照點名進行課程出缺席紀錄，並在手機查看出缺席紀錄。

學生可以透過拍照或選擇照片，上傳照片，且在手機上查看照片和出缺席紀錄。

手機端應用Retrofit:

在手機端主要是使用RxJava作為一個Adapter應用

RxJava 簡介 :

當程式要跟server端互動的時候，我們就會使用asynchronous programming，也就是所謂的響應式開發 ( reactive programming)，可以讓客戶端發出請求並給server端做回覆互動

RxJava是以Java做為開發的環境，並且從Apache 2.0 license所發布，RxJava提供了Java API並且有非對稱programming還提供Observable interface

RxJava 所必要的元件 :

* Observables : 就是提供資料的來源，也就是server端會回覆給我們的response
* Subsrcibers : 主要是監聽server端的response，一旦Observable有資料時會提醒client端做出相應的回覆
* 各種Method 來修改與組裝Observable所需的Data

1. Observables :
   1. Observable就是資料的來源，當subscriber開始監聽資料的時候便開始提供資料給Observable，而Observable也可以提供多筆資料(Object)，也可能是0筆資料，在Observable可以隨時終止，無論server response成功與否，而此時Server可能繼續提供資料而系統沒有接收
2. Subscribers :
   1. 每一個Observable可以有多個Subscriber，只要程式對於這筆資料有關係的話都可以一起subscribe，當Observable有新的資料的時候，可以呼叫onNext() 來呼叫所有的Subscriber來做出回應，而onComplete()是資料傳遞完成時會被呼叫，onError()就是當回覆有錯誤時所會有的回應

那為甚麼要使用asynchronous programming

Reactive programming提供了一個最簡單的方法來達成asynchronous programming, 並且簡化了asynchronously processing所會經過的等待時間，避免UI freeze的情況發生，此外它還可以處理多件事件同時發生的情況，無論是成功event或是Error event都可以Handle，可以有效避免各種exception發生的機會，還有Reactive programming也可以在不同執行緒上面處理不同Task的功能，可以大幅的增加程式運作的效率，或是達成更加複雜的功能一起整合在一起

實際應用:

註冊畫面 :

Server 會回傳整包的Json檔案，而其中Status可以確定加入資料是否成功

還有description可以得到詳細資料給前端通知使用

// 註冊

@POST("register")

@FormUrlEncoded

Observable<String> registerUser(

@Field("email") String email,// 使用者填寫Email位置

@Field("name") String name,// 使用者填寫Name位置

@Field("password") String password, // 使用者填寫Password位置

@Field("\_id") String \_id // 使用者填寫Id位置

);

而前端可以把Json 經由class “JsonParser” 把資料擷取後再給其他class使用

上傳照片 :

再上傳畫面當中，介面會自己判斷現在的使用者是誰，並且記錄下來userId，而圖片可以再經由request傳給後端做處理

@Multipart

@POST("studentUpload")

Observable<ResponseBody> studentUpload(

@Part("userId") RequestBody userId, // 使用者提供Id來給後端做insert

@Part MultipartBody.Part image // 使用者把圖片壓縮過後傳給後端

);

## 人臉偵測及辨識方法

### Google ML 模型

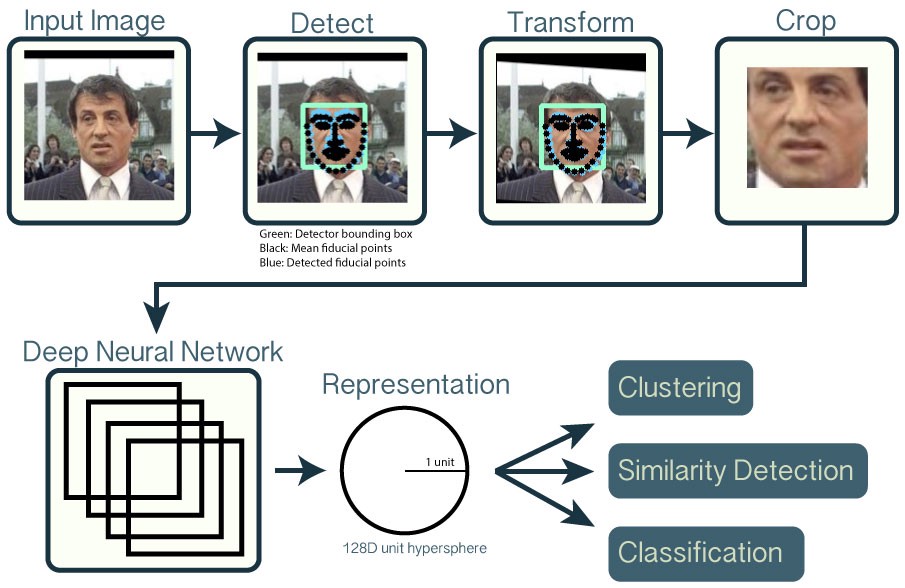


圖7

Google ML作為前端的人臉偵測的主要架構（圖7） 而上圖為偵測部分的流程 以下為說明

1. 將相機的影像先輸入為圖片當作Input資料
2. 把輸入影像與人臉特徵點做標記
3. 如果成功標記人臉之後 再進行裁切的動作 為了給後端作為人臉辨識資料做整理
4. 從後端進行資料庫的比對 並且輸出成果到手機前端做接收

而特徵比對的做法主要就是比較特徵點與距離的差距 如同下列公式（圖8）

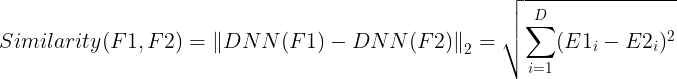


圖8

但是Google ML在手機端並不支援人臉辨識的部分，所以前端只會利用偵測的部分 將擷取下來的圖片與特徵值抓出來之後在給後端處理辨識，也不會對手機有太大的壓力

### TensorFlow Lite 模型

Tensorflow 提供一個人臉辨識訓練的環境，並且有以下的好處 :

* 提供各種Model作為深度學習的環境
* 訓練與預測Model的成果
* 是Open Source : 從Google Brain team開發出來的Tensorflow在2015 九月的時候釋出的，提供在深度學習的環境並且測試的地方

而Tensorflow lite主要是應用已經pretrain model，而且主要的環境是在mobile device上面，在TFLite提供了各種的Interface可以利用，而且最為重要的是tensorflow lite可以在有限的存取空間上面達到做好的平衡

Tensorflow lite有以下特色 : 在手機端可以有低延遲性，對於我們想要做的人臉偵測辨識有很大的關係，因為在做偵測的時候我們最為關注的是整個app的即時性，只要有一點點的時間偏差就會影響整個影像的結果，此外 TFLite也可以把已經訓練好的Tensorflow 模型經過優化變成.tflite的檔案，這個會比整個模型小10多倍，在手機端這種空間有限的環境下也是一個極具重要的特點

以下總結Tensorflow Lite的優點 :

* 容易把tensorflow model直接轉換成tensorflow lite的格式並且對於手機端有優化
* 在IOS與Android端可以輕鬆地開發深度學習程式
* 在mobile model 可以執行server base model
* 提供離線環境使用
* 在做ML的程式可以輕鬆的部屬而且整體的延遲性可以大幅地降低
* Tensorflow還要快很多 並且Weight file也夠小 只有5M

而Tensorflow lite也有其缺點 :

* 並非所有的Model都可以有很好的優化，有些Model還是會太大
* 優化程式效率的缺點就是對於Accuracy的效果會變差

## 系統架構圖

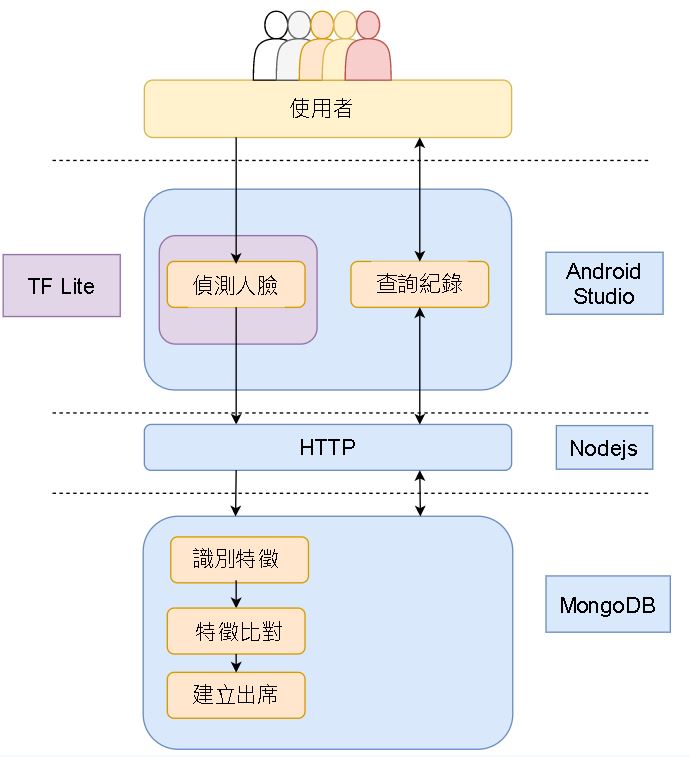


圖9

## 系統資料流程圖

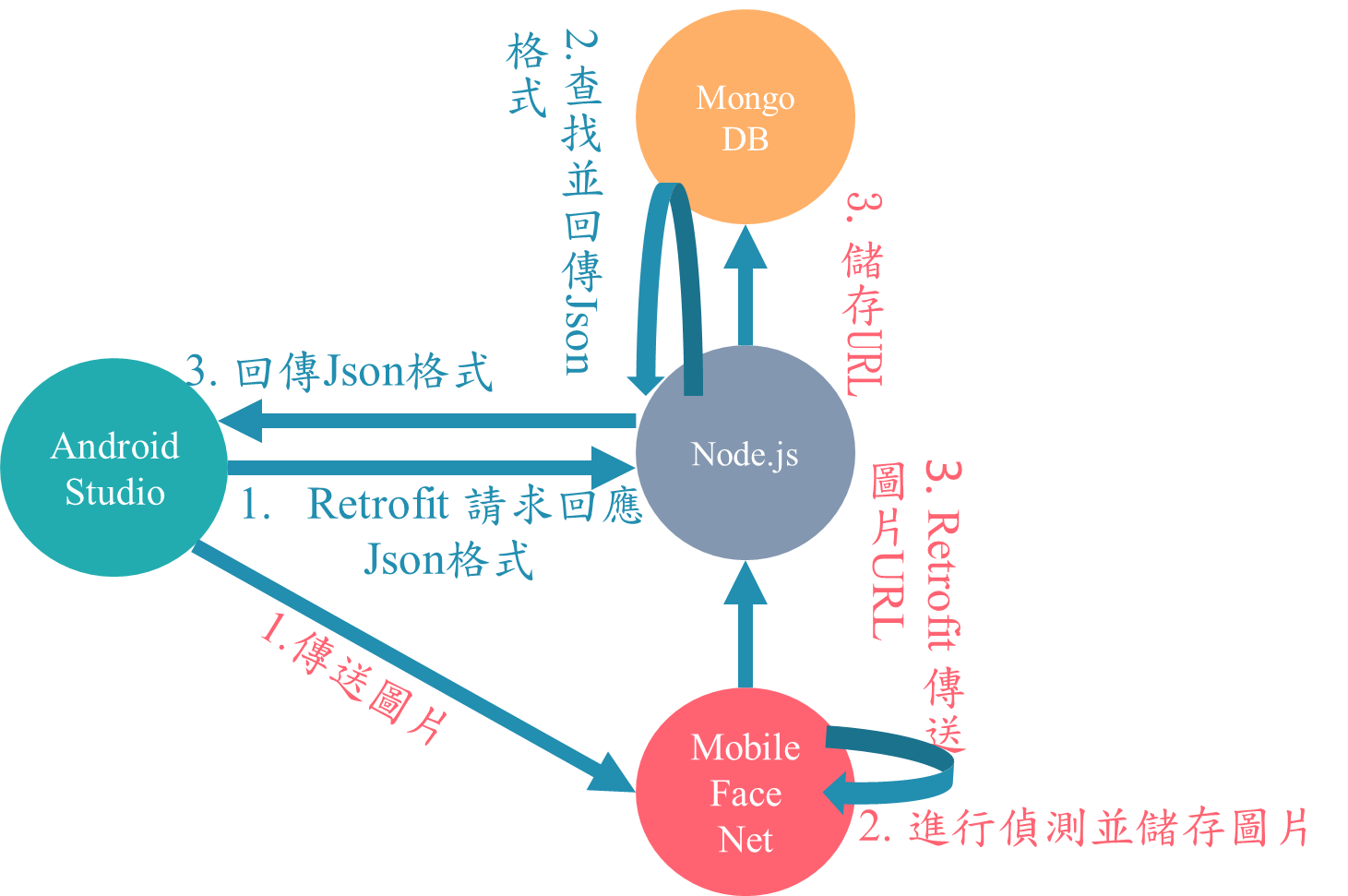


圖10

# 研究結果

## 開發與測試環境

本專題建立於 Windows 10 環境上，使用 Android Studio IDE 進行應用程式的開發，使用 AVD 模擬器及實體手機 Samsung J8 行動裝置做程式測試與執行，並以 Node.js和MongoDB作為儲存和運算資料之伺服器。

## 伺服器端資料庫系統欄位設計

以下圖是 MongoDB伺服器端之資料庫欄位分成八張表(Table)，名稱分別為

course、courseDate、studentCourse、user、avatar、attendance。

course為課程紀錄、courseDate為課程上課日期、studentCourse為學生選課資料、user為使用者資料、avatar為學生照片資料。



圖11：資料庫 course欄位圖

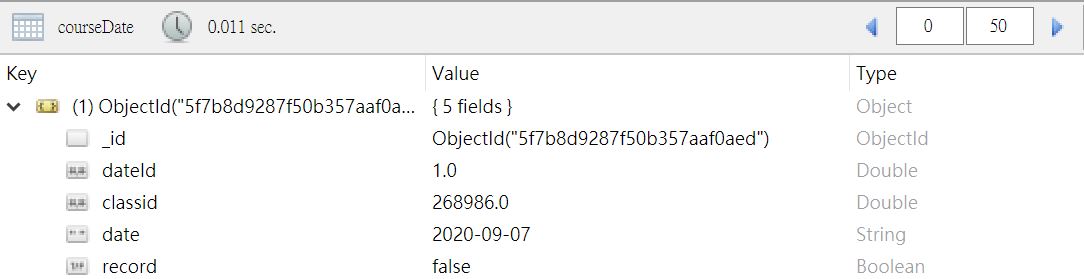


圖12：資料庫 courseDate欄位圖

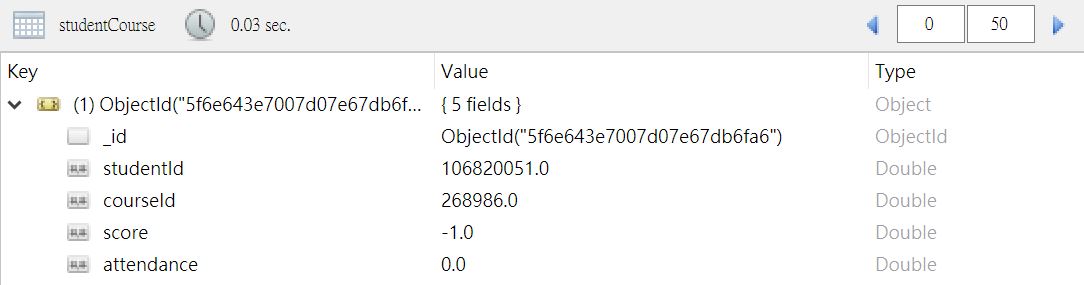


圖13：資料庫 studentCourse欄位圖

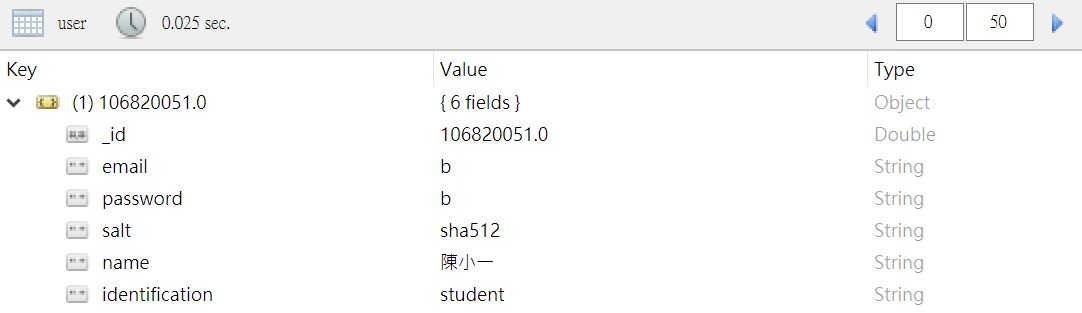


圖14：資料庫 user欄位圖

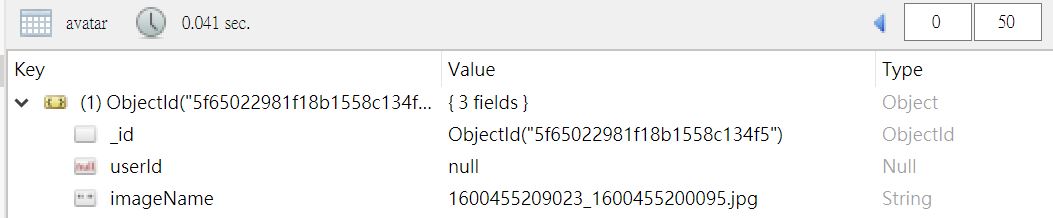


圖15：資料庫 avatar欄位圖

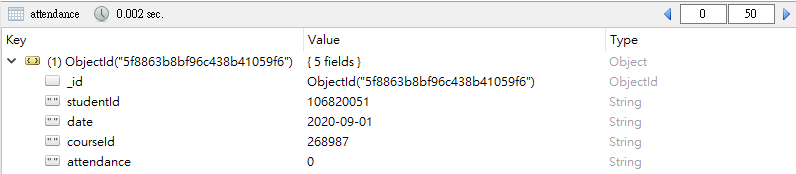


圖16：資料庫 attendance欄位圖

## 應用程式設計

### 應用程式之用戶登錄和註冊頁面

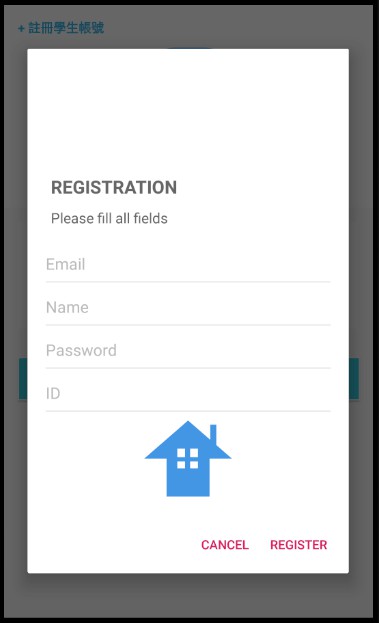
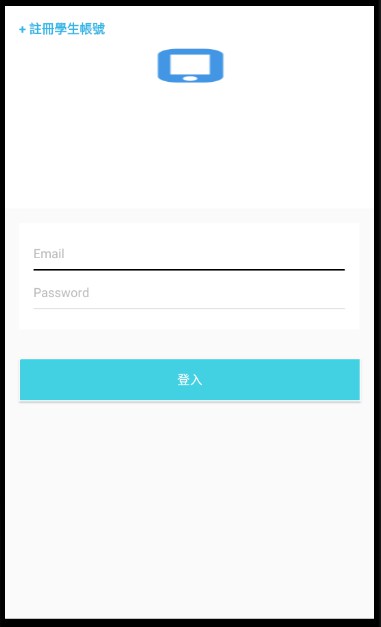
 

圖17：註冊頁面 圖18：登入頁面

### 應用程式之教師操作

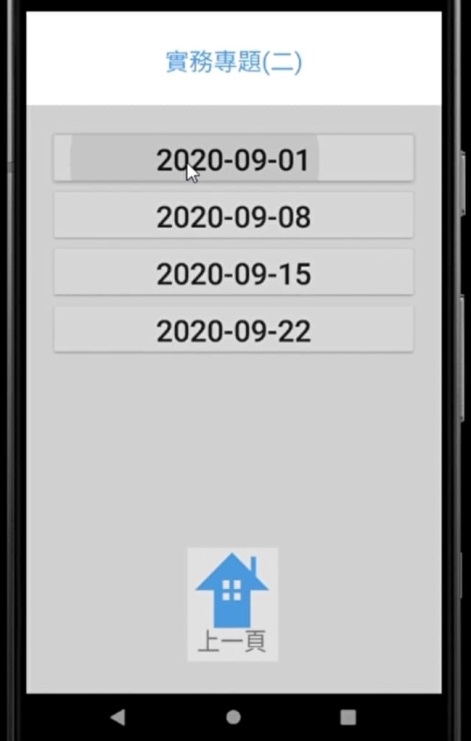
 

圖19：選擇課程頁面 圖20：選擇日期頁面

圖21：選擇模式頁面 圖22：點名頁面



圖23：查看出席紀錄頁面

### 應用程式之學生操作

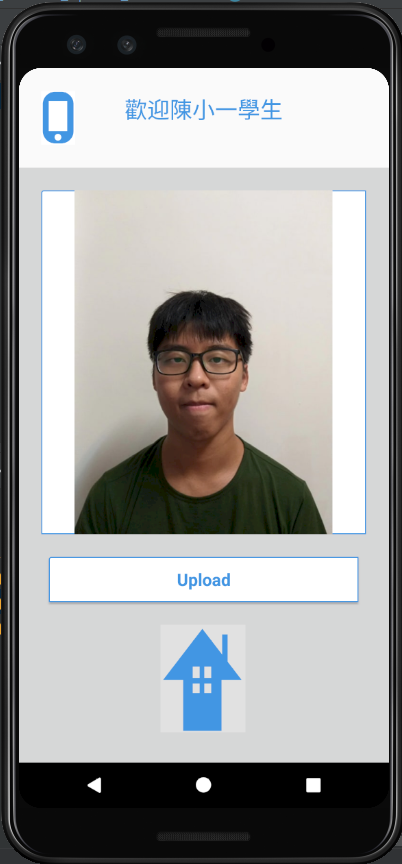
 

圖24：選擇模式頁面 圖25：上傳照片頁面

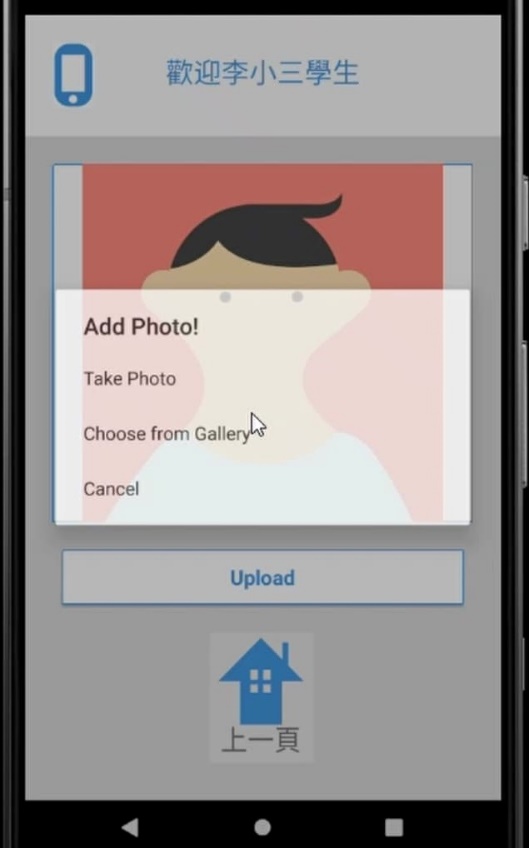
 

圖26：刪除照片頁面 圖27：選擇照片來源頁面

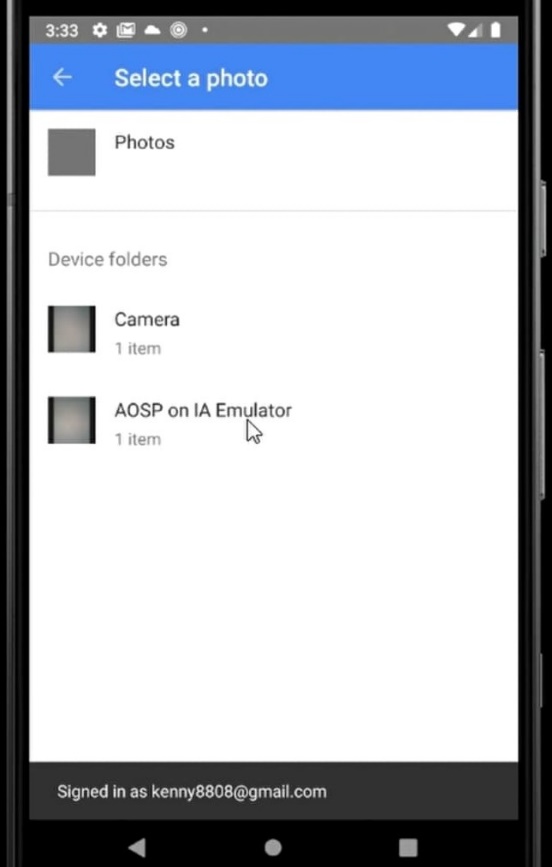
 

圖28：由照片庫選擇照片頁面 圖29：查看出缺席頁面

### 應用程式之學生人臉偵測畫面

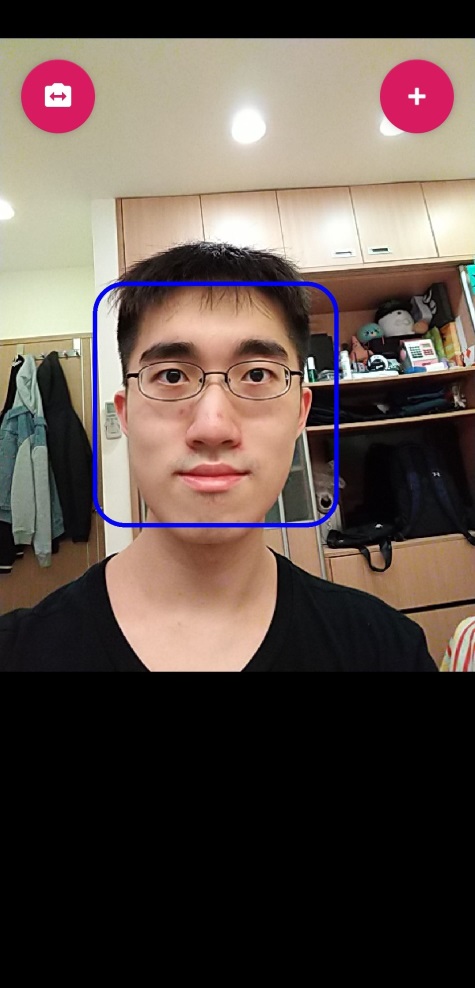
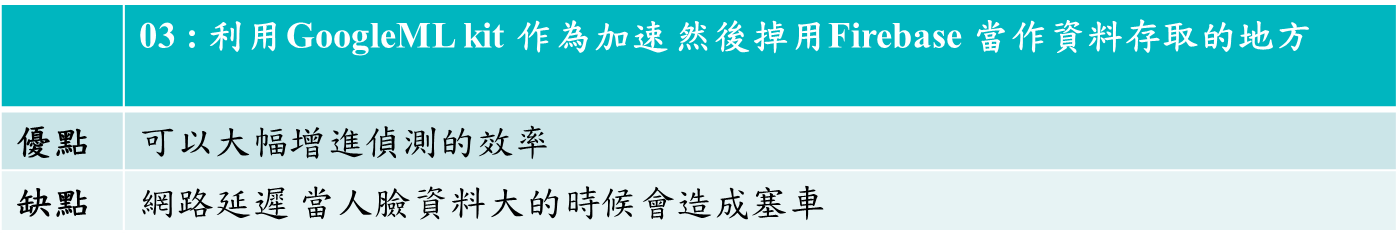
 

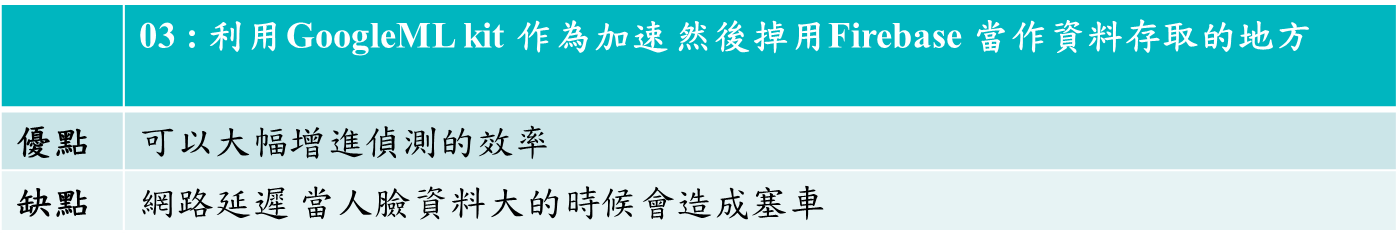
圖30：人臉偵測（無口罩） 圖31：人臉偵測（有口罩）

# 結論

## 模型探討









## 未來展望

最終是希望能夠串接學校DB API協助教授點名，並能獨自訓練出人臉偵測和辨識模型，使之在server端進行辨識之工作。

# 參考文獻

[1] MongoDB介紹網站

http://mongodbcanred.blogspot.com/2015/01/mongodb.html

[2] Android Studio 系統架構

https://www.chainnews.com/zh-hant/articles/856481115718.htm

[3] 手機系統使用率調查

https://3c.ltn.com.tw/news/37478

[4] MongoDB 系統架構

https://chtseng.wordpress.com/2014/05/19/nosql%E8%88%87mongodb/

[5] Node.js 介紹

https://ithelp.ithome.com.tw/articles/10191513

[6] Retrofit 介紹

https://segmentfault.com/a/1190000005638577

[7] Google ml-kit 介紹

https://firebase.google.com/docs/ml-kit/android/detect-faces

[8] 手機人臉辨識介紹

https://github.com/ateymoori/android\_face\_detection

[9] Google ml-kit 版本

https://developers.google.com/ml-kit/vision/face-detection

[10] 人臉分類器模型範例

https://github.com/tensorflow/examples/tree/master/lite/examples/image\_classification/android

[11] tensorflow介紹

https://medium.com/@santhoshkolloju/convert-your-pytorch-models-to-tensorflow-with-onnx-84c3bdd8d722

[12] MobileFaceNet介紹

https://github.com/sirius-ai/MobileFaceNet\_TF/issues/46

[13] facenet介紹

https://github.com/davidsandberg/facenet

[14] openface介紹

https://cmusatyalab.github.io/openface/models-and-accuracies/

[15] opencv4android介紹

https://opencv.org/opencv4android-samples/

[16] opencv-face-recognition介紹

https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/

[17] 人臉辨識運算法介紹

https://www.pyimagesearch.com/2017/05/22/face-alignment-with-opencv-and-python/

[18] 人臉辨識教學

https://www.learnopencv.com/face-recognition-an-introduction-for-beginners/

# 附錄

index.js:

// Register

app.post('/register', async(request, response, next)=>{

var userResponse = {};

var post\_data = request.body;

var plaint\_password = post\_data.password;

var hash\_data = saltHashPassword(plaint\_password);

var password = hash\_data.passwordHash;

var salt = hash\_data.salt;

var name = post\_data.name;

var email = post\_data.email;

var \_id = post\_data.\_id;

var identification = "student";

if (\_id[0] == 'a'){

identification = "teacher"

}

var insertJson = {

'email': email,

'password': plaint\_password, //改成不加密

'salt': salt,

'name': name,

'\_id': \_id,

'identification': identification

};

var db = client.db(dbName);

var numberOfSameEmail = await findUserExistenceUsingEmail();

if (numberOfSameEmail != 0){

userResponse.description = "Register email exist";

userResponse.status = 401;

console.log('Email already exists');

}

var numberOfSameId = await findUserExistenceUsingId();

if (numberOfSameId != 0){

userResponse.description = "Register id exist";

userResponse.status = 405;

console.log('Id already exists');

}

if (numberOfSameEmail == 0 && numberOfSameId == 0){

// Insert Data

db.collection('user').insertOne(insertJson, function(error, res){

if (!error){

userResponse.username = name;

userResponse.userId = \_id;

userResponse.description = "Register success";

userResponse.status = 202;

console.log('Registration success');

}

else {

userResponse.description = "insert data error";

userResponse.status = 406;

console.log("insert data error"); }

})

}

response.json(userResponse);

});

// Login

app.post('/login', async(request, response, next)=>{

userResponse = {};

var post\_data = request.body;

var email = post\_data.email;

var userPassword = post\_data.password;

var numberOfExistence = await findUserExistenceUsingEmail(email);

if (numberOfExistence == 0){

userResponse.description = 'Email not exists';

userResponse.status = "402";

console.log('Login email not exists');

}else{

var user = await findUserDataUsingEmail(email);

if (user){

if (userPassword != user.password){

console.log("login password error");

userResponse.description = 'Login password error !';

userResponse.status = "400";

}

else{

console.log("Login success");

userResponse.description = 'Login success';

if (user.identification == "student"){

console.log("Student login");

userResponse.description = "Student login";

userResponse.status = 200;

}

else if (user.identification == "teacher"){

userResponse.description = "Teacher login";

userResponse.status = 201;

userResponse.courses = {};

var courses = await findCoursesUsingTeacherId(user.\_id);

for (var i = 0; i < courses.length; i++){

userResponse.courses[courses[i].\_id] = courses[i].name;

}

}

userResponse.username = user.name;

userResponse.userId = user.\_id;

console.log("username : " + user.name + " login");

}

}

else {

console.log("user not found!");

}

}

console.log(userResponse);

response.json(userResponse);

});

// student upload images

app.post('/studentUpload', upload.single('image'), async(request, response, next)=>{

console.log(request.body);

console.log(request.file);

console.log(request.file.originalname);

var db = client.db(dbName);

var userResponse = {};

db.collection('avatar').insertOne({'userId': request.body.userId, 'imageName': imageName}, function(error, res){

if (error){

userResponse.status = 406;

userResponse.description = "insert data failed";

}

else {

userResponse.status = 204;

userResponse.description = "insert data success";

}

console.log(userResponse);

response.json(userResponse); }

})

});

// student check avatar

app.post('/studentCheckAvatar', async(request, response, next)=>{

var post\_data = request.body;

var student\_id = post\_data.studentId;

var index = post\_data.index;

var db = client.db(dbName);

var avatars = await db.collection('avatar').find({userId: student\_id}).toArray();

if (avatars.length == 0){

response.json(“no avatar exist”);

}

else {

response.sendFile(\_\_dirname + "/uploads/" + avatars[index].imageName);

}

});

IMyService.java:

// 註冊

@POST("register")

@FormUrlEncoded

Observable<String> registerUser(

@Field("email") String email,

@Field("name") String name,

@Field("password") String password,

@Field("\_id") String \_id

);

// 登入

@POST("login")

@FormUrlEncoded

Observable<String> loginUser(

@Field("email") String email,

@Field("password") String password

);

//學生上傳圖片

@Multipart

@POST("studentUpload")

Observable<ResponseBody> studentUpload(

@Part("userId") RequestBody userId,

@Part MultipartBody.Part image

// @Part("other") RequestBody other

);

//學生查看圖片

@POST("studentCheckAvatar")

@FormUrlEncoded

Observable<ResponseBody> studentCheckAvatar(

@Field("studentId") String studentId,

@Field("index") int index

);

CameraCapture.java:

package org.ntut.faceRecognition.Camera;

import android.app.Activity;

import android.content.Intent;

import android.graphics.Bitmap;

import android.graphics.Bitmap.Config;

import android.graphics.Canvas;

import android.graphics.Color;

import android.graphics.Matrix;

import android.graphics.Paint;

import android.graphics.Paint.Style;

import android.graphics.RectF;

import android.hardware.camera2.CameraCharacteristics;

import android.media.ImageReader.OnImageAvailableListener;

import android.os.Bundle;

import android.util.Size;

import com.google.android.material.floatingactionbutton.FloatingActionButton;

import com.google.mlkit.vision.common.InputImage;

import com.google.mlkit.vision.face.Face;

import com.google.mlkit.vision.face.FaceDetection;

import com.google.mlkit.vision.face.FaceDetector;

import com.google.mlkit.vision.face.FaceDetectorOptions;

import org.ntut.faceRecognition.Camera.customview.OverlayView;

import org.ntut.faceRecognition.Camera.env.ImageUtils;

import org.ntut.faceRecognition.Camera.tflite.SimilarityClassifier;

import org.ntut.faceRecognition.Camera.tflite.TFLiteObjectDetectionAPIModel;

import org.ntut.faceRecognition.Camera.tracking.MultiBoxTracker;

import org.ntut.faceRecognition.R;

import org.ntut.faceRecognition.Utility.ImageSaver;

import org.ntut.faceRecognition.Utility.Utils;

import java.util.LinkedList;

import java.util.List;

public class CameraCapture extends CameraActivity implements OnImageAvailableListener {

// MobileFaceNet

private static final int TF\_OD\_API\_INPUT\_SIZE = 112;

private static final boolean TF\_OD\_API\_IS\_QUANTIZED = false;

private static final String TF\_OD\_API\_MODEL\_FILE = "mobile\_face\_net.tflite";

private static final boolean MAINTAIN\_ASPECT = false;

private static final Size DESIRED\_PREVIEW\_SIZE = new Size(1920, 1080);

OverlayView trackingOverlay;

private Integer sensorOrientation;

private SimilarityClassifier detector;

private Bitmap rgbFrameBitmap = null;

private Bitmap croppedBitmap = null;

private Bitmap cropCopyBitmap = null;

private boolean computingDetection = false;

private Matrix frameToCropTransform;

private Matrix cropToFrameTransform;

private MultiBoxTracker tracker;

private FaceDetector faceDetector;

// here the preview image is drawn in portrait way

private Bitmap portraitBmp = null;

// here the face is cropped and drawn

private Bitmap faceBmp = null;

private FloatingActionButton addButton;

private boolean getPhoto;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

findView();

createFaceDetector();

setAddButton();

}

private void findView() {

addButton = findViewById(R.id.add\_button);

}

private void createFaceDetector() {

FaceDetectorOptions options =

new FaceDetectorOptions.Builder()

.setPerformanceMode(FaceDetectorOptions.PERFORMANCE\_MODE\_FAST)

.setContourMode(FaceDetectorOptions.LANDMARK\_MODE\_NONE)

.setClassificationMode(FaceDetectorOptions.CLASSIFICATION\_MODE\_NONE)

.build();

faceDetector = FaceDetection.getClient(options);

}

private void setAddButton() {

addButton.setOnClickListener(v -> getPhoto = true);

}

@Override

public void onPreviewSizeChosen(final Size size, final int rotation) {

tracker = new MultiBoxTracker(this);

try {

detector =

TFLiteObjectDetectionAPIModel.create(

getAssets(),

TF\_OD\_API\_MODEL\_FILE,

TF\_OD\_API\_INPUT\_SIZE,

TF\_OD\_API\_IS\_QUANTIZED);

} catch (Exception e) {

e.printStackTrace();

Utils.showToast("Classifier could not be initialized", CameraCapture.this);

finish();

}

previewWidth = size.getWidth();

previewHeight = size.getHeight();

sensorOrientation = rotation - getScreenOrientation();

rgbFrameBitmap = Bitmap.createBitmap(previewWidth, previewHeight, Config.ARGB\_8888);

int targetW, targetH;

if (sensorOrientation == 90 || sensorOrientation == 270) {

targetH = previewWidth;

targetW = previewHeight;

} else {

targetW = previewWidth;

targetH = previewHeight;

}

int cropW = (int) (targetW / 8.0);

int cropH = (int) (targetH / 8.0);

croppedBitmap = Bitmap.createBitmap(cropW, cropH, Config.ARGB\_8888);

portraitBmp = Bitmap.createBitmap(targetW, targetH, Config.ARGB\_8888);

faceBmp = Bitmap.createBitmap(TF\_OD\_API\_INPUT\_SIZE, TF\_OD\_API\_INPUT\_SIZE, Config.ARGB\_8888);

frameToCropTransform =

ImageUtils.getTransformationMatrix(

previewWidth, previewHeight,

cropW, cropH,

sensorOrientation, MAINTAIN\_ASPECT);

cropToFrameTransform = new Matrix();

frameToCropTransform.invert(cropToFrameTransform);

trackingOverlay = findViewById(R.id.tracking\_overlay);

trackingOverlay.addCallback(

canvas -> tracker.draw(canvas));

tracker.setFrameConfiguration(previewWidth, previewHeight, sensorOrientation);

}

@Override

protected void processImage() {

trackingOverlay.postInvalidate();

// No mutex needed as this method is not reentrant.

if (computingDetection) {

readyForNextImage();

return;

}

computingDetection = true;

rgbFrameBitmap.setPixels(getRgbBytes(), 0, previewWidth, 0, 0, previewWidth, previewHeight);

readyForNextImage();

final Canvas canvas = new Canvas(croppedBitmap);

canvas.drawBitmap(rgbFrameBitmap, frameToCropTransform, null);

InputImage image = InputImage.fromBitmap(croppedBitmap, 0);

faceDetector

.process(image)

.addOnSuccessListener(faces -> {

if (faces.size() == 0) {

updateResults(new LinkedList<>());

return;

}

runInBackground(

() -> onFacesDetected(faces, getPhoto));

});

}

@Override

protected int getLayoutId() {

return R.layout.camera\_connection\_fragment;

}

@Override

protected Size getDesiredPreviewFrameSize() {

return DESIRED\_PREVIEW\_SIZE;

}

// Face Processing

private Matrix createTransform(

final int srcWidth,

final int srcHeight,

final int dstWidth,

final int dstHeight,

final int applyRotation) {

Matrix matrix = new Matrix();

if (applyRotation != 0) {

// Translate so center of image is at origin.

matrix.postTranslate(-srcWidth / 2.0f, -srcHeight / 2.0f);

// Rotate around origin.

matrix.postRotate(applyRotation);

}

if (applyRotation != 0) {

// Translate back from origin centered reference to destination frame.

matrix.postTranslate(dstWidth / 2.0f, dstHeight / 2.0f);

}

return matrix;

}

private void updateResults(final List<SimilarityClassifier.Recognition> mappedRecognitions) {

tracker.trackResults(mappedRecognitions);

trackingOverlay.postInvalidate();

computingDetection = false;

if (mappedRecognitions.size() > 0) {

SimilarityClassifier.Recognition rec = mappedRecognitions.get(0);

}

}

private void onFacesDetected(List<Face> faces, boolean add) {

cropCopyBitmap = Bitmap.createBitmap(croppedBitmap);

final Canvas canvas = new Canvas(cropCopyBitmap);

final Paint paint = new Paint();

paint.setColor(Color.RED);

paint.setStyle(Style.STROKE);

paint.setStrokeWidth(2.0f);

final List<SimilarityClassifier.Recognition> mappedRecognitions =

new LinkedList<SimilarityClassifier.Recognition>();

// Note this can be done only once

int sourceW = rgbFrameBitmap.getWidth();

int sourceH = rgbFrameBitmap.getHeight();

int targetW = portraitBmp.getWidth();

int targetH = portraitBmp.getHeight();

Matrix transform = createTransform(

sourceW,

sourceH,

targetW,

targetH,

sensorOrientation);

final Canvas cv = new Canvas(portraitBmp);

// draws the original image in portrait mode.

cv.drawBitmap(rgbFrameBitmap, transform, null);

final Canvas cvFace = new Canvas(faceBmp);

boolean saved = false;

for (Face face : faces) {

final RectF boundingBox = new RectF(face.getBoundingBox());

if (boundingBox != null) {

// maps crop coordinates to original

cropToFrameTransform.mapRect(boundingBox);

// maps original coordinates to portrait coordinates

RectF faceBB = new RectF(boundingBox);

transform.mapRect(faceBB);

// translates portrait to origin and scales to fit input inference size

float sx = ((float) TF\_OD\_API\_INPUT\_SIZE) / faceBB.width();

float sy = ((float) TF\_OD\_API\_INPUT\_SIZE) / faceBB.height();

Matrix matrix = new Matrix();

matrix.postTranslate(-faceBB.left, -faceBB.top);

matrix.postScale(sx, sy);

cvFace.drawBitmap(portraitBmp, matrix, null);

//canvas.drawRect(faceBB, paint);

float confidence = -1f;

Integer color = Color.BLUE;

Bitmap crop = null;

if (add) {

crop = Bitmap.createBitmap(portraitBmp,

(int) faceBB.left,

(int) faceBB.top,

(int) faceBB.width(),

(int) faceBB.height());

}

if (getCameraFacing() == CameraCharacteristics.LENS\_FACING\_FRONT) {

// camera is frontal so the image is flipped horizontally

// flips horizontally

Matrix flip = new Matrix();

if (sensorOrientation == 90 || sensorOrientation == 270) {

flip.postScale(1, -1, previewWidth / 2.0f, previewHeight / 2.0f);

} else {

flip.postScale(-1, 1, previewWidth / 2.0f, previewHeight / 2.0f);

}

flip.mapRect(boundingBox);

}

final SimilarityClassifier.Recognition result = new SimilarityClassifier.Recognition(

"0", "", confidence, boundingBox);

result.setColor(color);

result.setLocation(boundingBox);

result.setCrop(crop);

mappedRecognitions.add(result);

if (getPhoto) {

if (crop != null) {

// TODO return !

Intent returnIntent = new Intent();

new ImageSaver(this).

setFileName("captureImage.png").

setDirectoryName("images").

save(crop);

setResult(Activity.RESULT\_OK, returnIntent);

finish();

}

}

}

}

updateResults(mappedRecognitions);

}

}