美和學校財團法人美和科技大學

108 年度教師產學合作計畫

結案報告書

計畫名稱：利用卷積神經網路辨識手指姿勢

計畫編號：108- FI-DIT-IAC-R-001

計畫期間：108 年 6 月 1 日 ~ 108 年 10 月 31 日

計畫主持人：游義地

共同主持人：

研究助理：

經費總額：　　60,000　　元

經費來源：標準桿實業有限公司

摘　要

以人體的姿態或手勢操作硬體設施是資訊科技應用的重要項目，除了體感遊戲外，對於銀髮族或部分行動不方便人士而言，利用手指姿勢操作家電設備，可有效提高生活便利性。一般手勢辨識以影像辨識為主，利用標記演算法及邊界演法求取圖形特徵，過程十分繁瑣。本研究目的是開發手指姿勢辨識系統，採用卷積神經網路求取手指姿勢特徵，主要是判斷手指個數，其結果可作為手勢控制家電之依據。

本研究主要分為兩部分，第一部分為建立手指姿勢圖片，為了確保所開發之程式能有效辨識手指個數，本研究拍製手指姿勢圖片約 1200 張，其中 900 張訓練集為單一測試者，另外 300 張為測試集分屬三位不同測試者。為了提高特徵值擷取，本研究採用膚色二值化處理，先將彩色圖片轉成二值化，再進行深度學習；第二部分為卷積神經網路學習過程，本研究採用 Python 語言，以 Keras 作為軟體開發平台。Keras 是屬於高階的深度學習程式庫，可以使用較少的程式碼，花費較少的時間，就可建立深度學習模型，並進行神經元訓練。經實際測試結果，膚色二值化處理可有效提高手部辨識率，三組測試準確度皆達 90%以上。

關鍵字：深度學習、Keras、手指姿勢辨識。

第一章 緒論

1.1 研究動機

隨著科技的進步和人類追求便利與健康樂活的需求，近年來機器人研究的重點，從早期的工業用機器人，逐漸發展到各種服務型機器人。無論是娛樂、服務、照護或是基本的操作型機器人，都開始朝向智慧型機器人發展，以求能理解環境的狀況來做出相對應的行為，而非只是進行單純重複性的工作。

為了增進機器人判斷人類肢體語言之能力，本文實現深度學習技術在手指姿勢影像辨識的可行性，讓人類只需藉由手指的姿勢就能達到與機器人之溝通。

1.2 研究目的

本研究主要的目的為以下三點：

1. 本文提出一個以卷積神經網路用於手指姿勢之辨識，實現深度學習在手指姿勢辨識研究的可行性。
2. 比較使用膚色二值化後的手指圖像訓練出來的模型和使用原始圖像訓練出來的模型來比對三位測試者的預測準確率哪個較高。
3. 透過本研究從數據收集、處理到訓練神經網路模型來讓自己對深度學習及 CNN 的運作有更進一步的認知。

1.3 文獻探討

利用手勢與機器互動是最近的課題，藉由訓練與學習，手勢辨識的結果可以使用在許多人機互動的應用。在人類與電腦的互動人機介面發展上，人類可以藉由手勢取代滑鼠與其他手持設備作為輸入電腦的方法。同樣的，手勢也可以取代搖桿和按鈕，直接控制電腦機具[1]。在航太科學和軍事用途的研究上，遠端操控機器是一種典型的應用，透過影像獲取遠端資訊，以手勢來進行控制而達到遠端操控的效果[2]。虛擬實境科技研發上，常常以手勢作為與虛擬環境互動的方式[3]。而在視覺監視系統的應用中，可以藉由手勢判斷出觀察對象的行為模式，

而判斷是否將進行不被允許的動作[4]。在醫學上，透過觀察病人的手勢動作評估病人的情緒與壓力程度變化，來做為關護照看的依據[5]。手勢的變化也可以做為偵訊犯人時的測謊依據[6]。近年來的電視遊樂器平台上，以手勢等體感作為遊戲方式的遊戲也非常受到歡迎[7]。在機器人領域中，以手勢作為控制機器人的方法也有許多的相關研究[8]。

第二章 深度學習

深度學習是機器學習的分支，兩者關係如圖 2.1 所示。深度學習是人工智慧中，成長最快的領域，深度學習模擬人類神經網路的運作方式，常見的深度學習架構，如多層感知器（Multilayer Perceptron）、深度神經網路 DNN（Deep Neural Network）、卷積神經網路 CNN（Convolution Deep Neural Network）、遞迴神經網路 RNN（Recurrent Neural Network）。深度學習特別應用於視覺辨識、語音識別、自然語言處理、生物醫學等領域，取得非常好的效果。

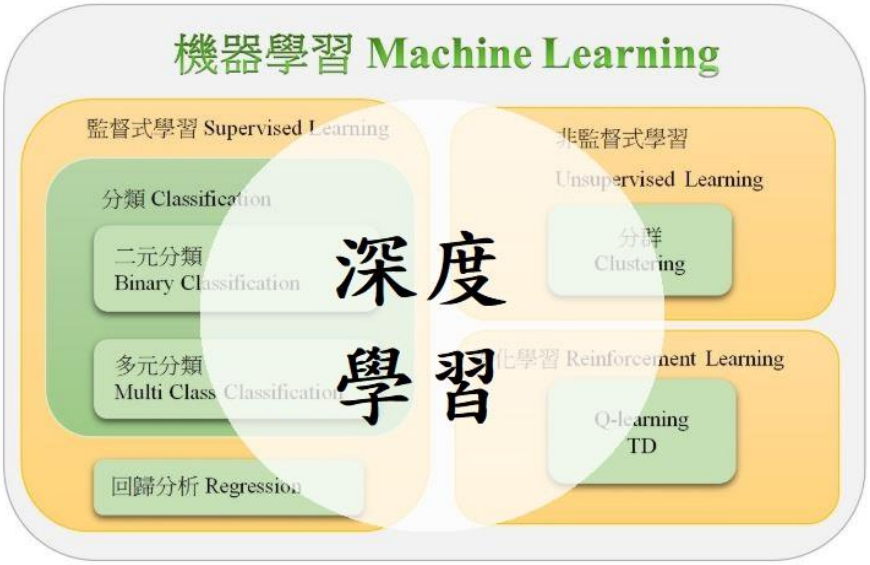


圖 2.1：機器學習與深度學習的關係

人類神經網路結構非常複雜，預估具有 860 億，以及超過 100 兆條神經相連，形成的網路比最先進的超級電腦還要強大。為了方便電腦模擬，將神經元分為多層次，來模擬神經網路。通常會有一個輸入層，一個輸出層、隱藏層可以非常多層，如圖 2.2 所示，所以稱為深度學習。

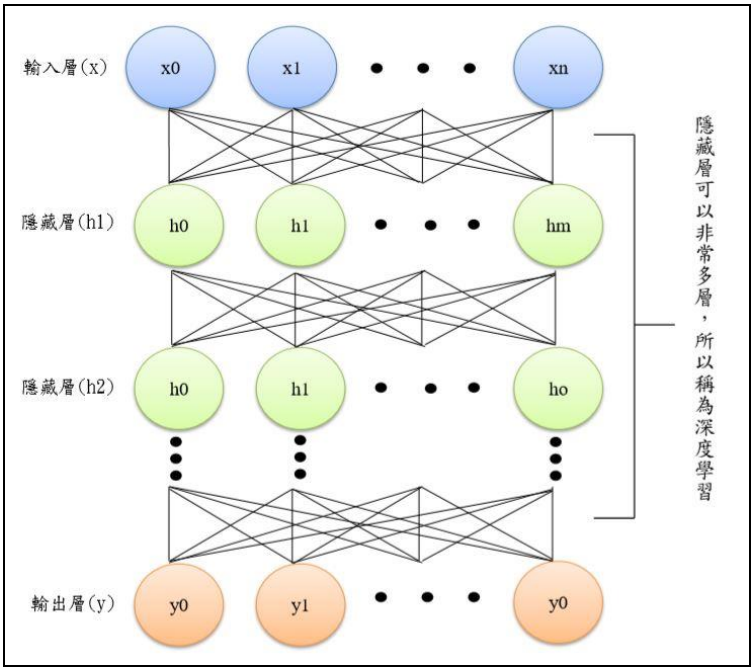


圖 2.2：神經網路

2.1 神經傳導原理介紹

神經傳導運作很複雜，在此本文中簡略介紹其概念，如下圖：

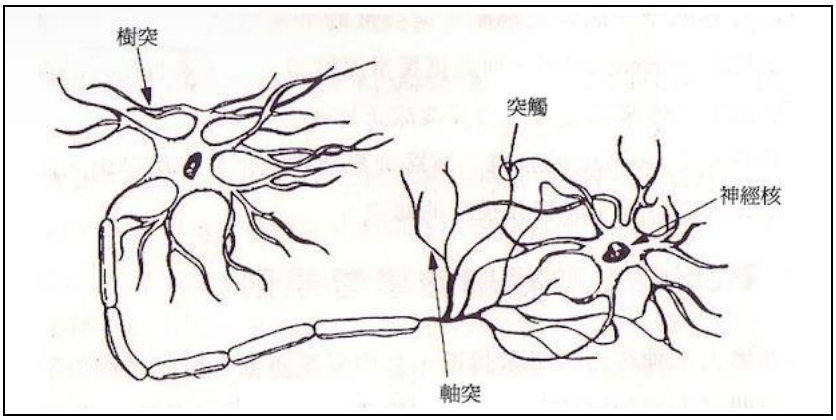


圖 2.3：生物神經元構造

- 軸突傳送訊息：神經元長出一細長條的軸突，以電流方式將訊息傳遞給另一個神經元。軸突最長可達一公尺，最短只有幾十分之一毫米。
- 樹突接收訊息：樹突主要功能是，接收其他神經元傳來的電化學訊息，再傳遞給本身的細胞。
- 突觸是輸入與輸出的神經元傳遞的機制：輸入與輸出的神經元之間發展出特

殊的結構稱為突觸，神經元透過釋放化學物質來傳遞訊息，當電壓達到臨界值，就會透過軸突，傳送電脈衝動作電位至接收神經元。

為了將神經元以電腦來模擬，將神經元傳導，以數學公式來表示。以下圖 2.4 說明。

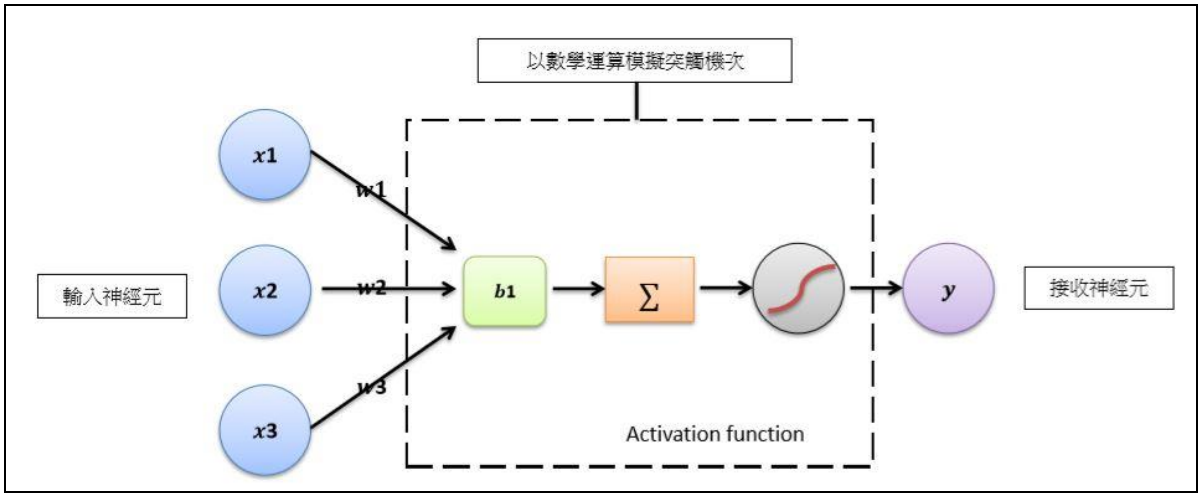


圖 2.4：一個接收神經元的神經網路

圖 2.4 可以整理公式(2-1)如下：

$$y = activation\ function(x1 \times w1 + x2 \times w2 + x3 \times w3 + b1) \quad (2\text{-}1)$$

公式(2-1)詳細說明如下：

- 輸入x：x模擬輸入神經元，接收外界傳送訊息，如圖 2.4 中，共有三個輸入神經元：$x1$、$x2$、$x3$。
- 接收y：模擬接收神經元，如圖 2.4 中，只有一個接收神經元：y。
- 權重w(weight)：權重w模擬軸突，連續輸入與接收神經元，負責傳送訊息。如圖 2.4 中，共有三個軸突：$w1$、$w2$、$w3$。

- 偏權值b(bias)：偏權值b模擬突觸的結構，代表接收神經元容易被活化的程度，偏權值越高，越容易被活化並傳遞訊息。如圖 2.4 中，因為接收神經元只有一個，所以也只有一個偏權值：$b1$。
- 活化函數（activation function）：活化函數模擬神經傳導的運作方式，當接收神經元，接受刺激的總和：$(x1 \times w1 + x2 \times w2 + x3 \times w3 + b1)$，經過活化函數的運算，大於臨界值會傳遞至下一個神經元。常見的活化函數，例如：sigmoid、relu。

以上活化函數可以模擬神經傳導的運作方式，將上一層神經元訊號傳遞至下一層。活化函數（activation function）通常為非線性函數，加入了活化函數，讓神經網路可以處理比較複雜的線性問題。以圖 2.5 說明線性與非線性函數：

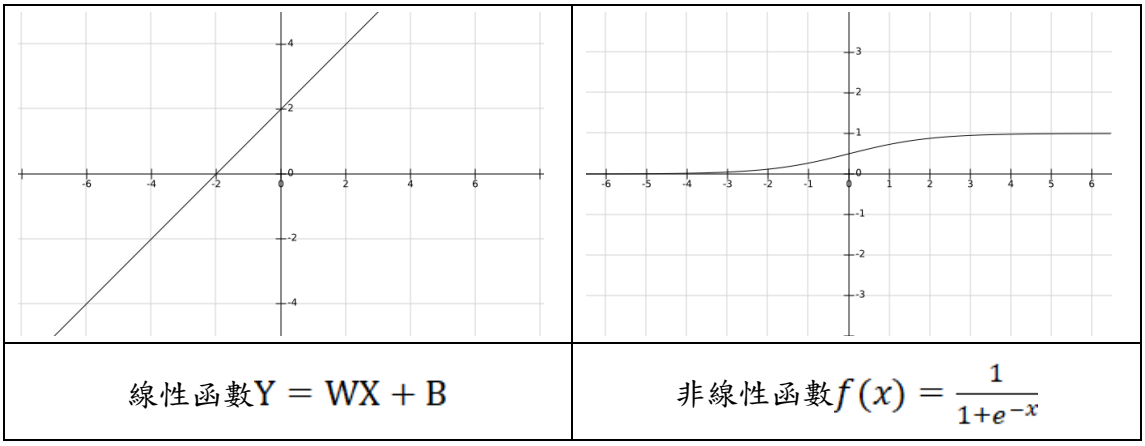


圖 2.5：線性函數與非線性函數

Keras 與 Tensorflow 支援很多活化函數，不過我們介紹最常用的兩種函數：Sigmoid 與 relu（activation function 有很多翻譯方式，例如：活化函數、激勵函數、激活函數，本文統一翻譯為活化函數，其意義是接收神經網路刺激使神經元活化之意）。

1. sigmoid 活化函數

常用的活化函數 sigmoid 如下，式子(2-2)。

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2-2}$$

Sigmoid 活化函數，其實與人類感覺神經，對訊號的接收類似，如圖 2.6 所示，例如：當接收神經元，接受刺激的總和：

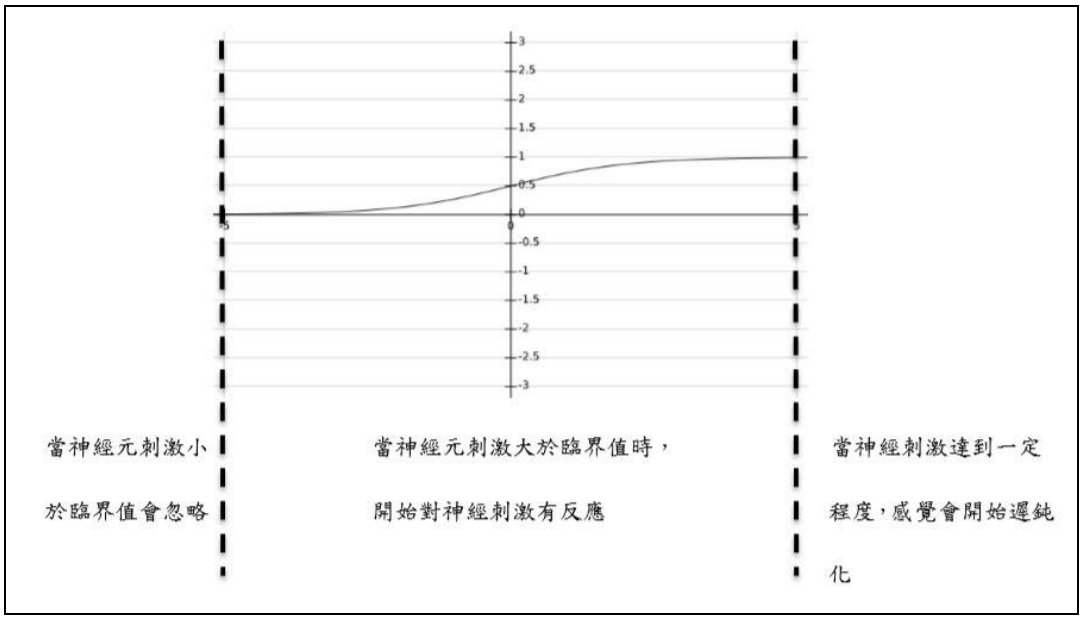


圖 2.6：sigmoid 活化函數

- 小於臨界值，會忽略此刺激：當x小於 5，輸出y接近 0。
- 大於臨界值，開始接收神經刺激：當x範圍再-5 與 5 之間，隨著x數值加大，y的數值也加大。
- 當神經刺激達到一定程度，感覺會開始鈍化：即使更大的刺激，感覺仍維持不變，即當x小於 5，y的數值趨近於 1。

2. Relu 活化函數

另一個很常見的活化函數 relu，如圖 2.7：

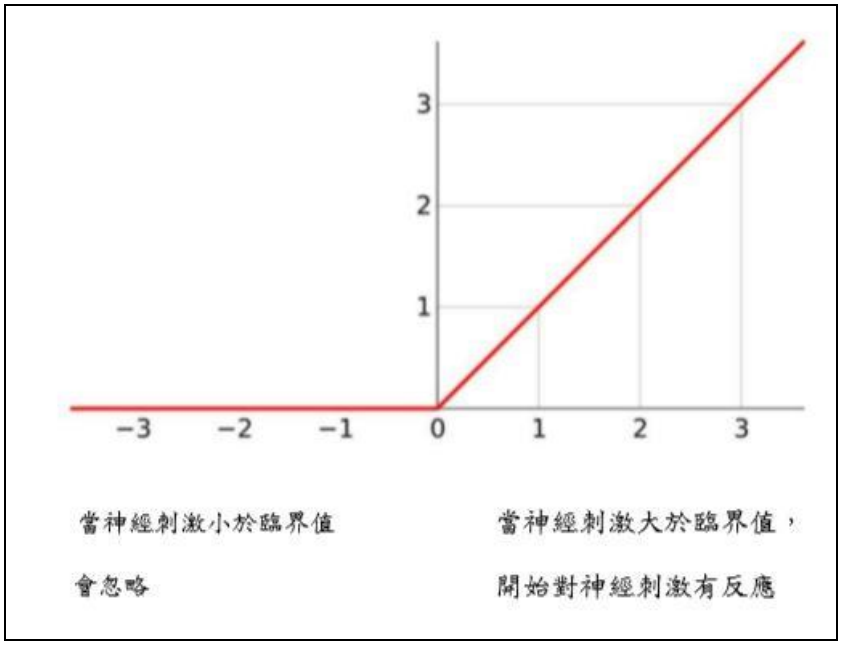


圖 2.7：Relu 活化函數

當接收神經元，接受刺激的總和：

- 小於臨界值，會忽略此刺激：輸入x小於 0，y值是 0。
- 大於臨界值，開始接收神經刺激：輸入x大於 0，y等於x。

2.2 以矩陣運算模擬神經網路

本文上一節中，是以數學運算模擬單一接收神經元，本節將說明以矩陣模擬兩個接收神經元。

多個接收神經元的神經網路，如圖 2.8 所示：

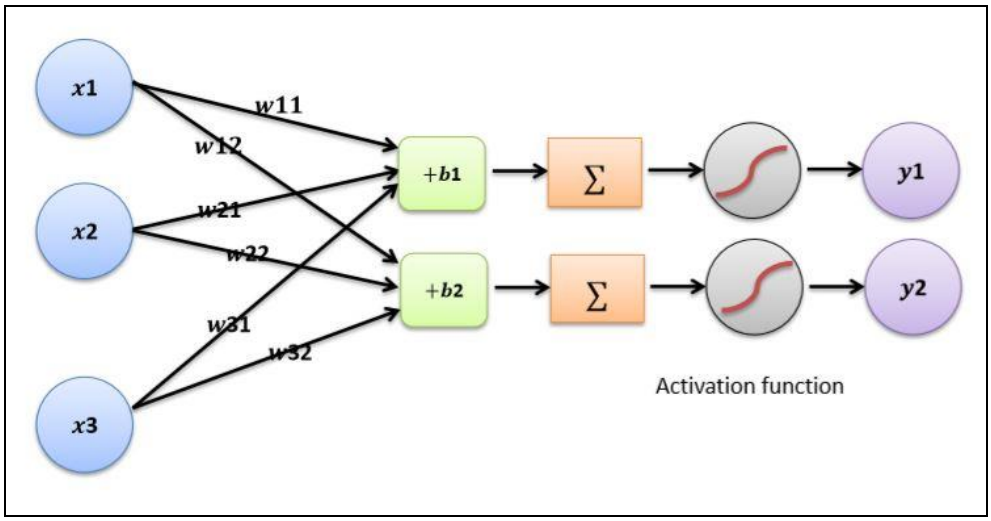


圖 2.8：多個接收神經元的神經網路

從圖 2.8 中，可以整理數學式子(2-3)、(2-4)：

$$y1 = activation\ function(x1 \times w11 + x2 \times w21 + x3 \times w31 + b1) \quad (2\text{-}3)$$

$$y2 = activation\ function(x1 \times w12 + x2 \times w22 + x3 \times w32 + b1) \quad (2\text{-}4)$$

矩陣形式以式子(2-5)表示：

$$Y = activation(X \times W + b) \quad (2\text{-}5)$$

- 輸入X：X模擬輸入神經元，接收外界傳送訊息，如圖 2.8 中，共有三個輸入神經元：$x1$、$x2$、$x3$。
- 接收Y：模擬接收神經元，如圖 2.8 中，共有兩個輸入神經元：$y1$、$y2$。
- 權重W：權重模擬神經元的軸突，連結輸入與接收神經元，負責傳送訊息。因為要完全連結輸入與接收神經元，共需要（輸入 3）×（接收 2）＝ 6 個軸突。

  $w11$、$w21$、$w31$，負責傳送訊息給$y1$。

  $w12$、$w22$、$w32$，負責傳送訊息給$y2$。

- 偏權值b（bias）：偏權值b模擬突觸的結構，代表接收神經元容易被活化的程度，偏權值越高，越容易被活化並傳遞訊息。如圖 2.8 中，因為接收神經元有兩個，所以也有兩個偏權值：$b1$、$b2$。
- 活化函數（activation function）：活化函數模擬神經傳導的運作方式，例如：當接收的神經元$y1$，接收刺激的總和：

  $(x1 \times w11 + x2 \times w21 + x3 \times w31 + b1)$，經過活化函數的運算，大於臨界值會傳遞至下一個神經元。

2.3 多層感知器模型（Multilayer Perceptron）

以多層感知器模型辨識本研究手指姿勢影像，說明多層感知器的運作方式，如圖 2.9 所示。

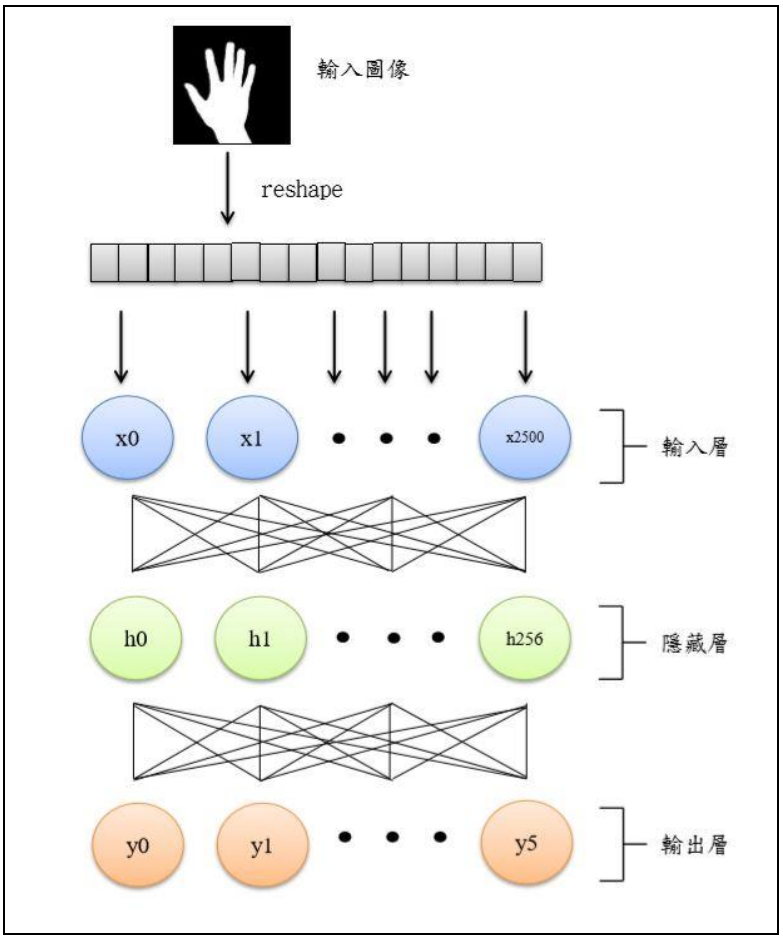


圖 2.9：以多層感知器辨識手指姿勢影像

- 輸入圖像：例如手指姿勢 Five 影像，是 50×50 的 2 維影像，以 reshape 轉換為 1 維向量，也就是 2500 個 float 數字，作為 2500 個神經元的輸入。
- 輸入層（Input layer）：2500 個的輸入神經元，接收外界訊號。
- 隱藏層（Hidden layer）：模擬內部神經元，共有 256 個隱藏神經元。
- 輸出層（Output layer）：6 個輸出神經元，就是預測結果。對應到希望的數字，從 0~5 就有 6 個結果。

建立輸入層與隱藏層數學式子(2-6)：

$$h1 = Relu(X \times W1 + b1) \tag{2-6}$$

以式子 2-6 模擬多層感知器模型運作，如圖 2.10 詳細說明如下：

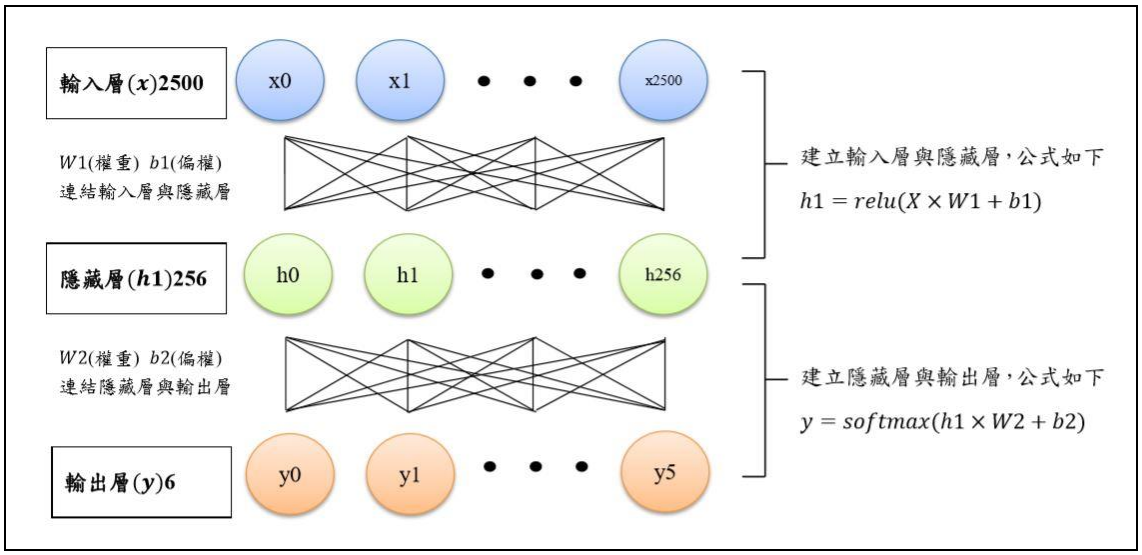


圖 2.10：多層感知器模型運作

- 輸入層x：x模擬輸入神經元，接收外界傳送訊息，如圖 2.10 所示，共有 2500 個神經元。
- 隱藏層$h1$：隱藏層$h1$模擬內部神經元，共有 256 個神經元。

- 權重$W1$：權重模擬神經元軸突，連結輸入與接收神經元，負責傳送訊息。連結輸入層（2500 個神經元）與隱藏層（256 個神經元），為了讓二層的每一個神經元都互相連結，總共需要 2500×256 ＝ 640000 個軸突，所以權重（$W1$）必須是 2500×256 的矩陣，用來模擬這些軸突的功能。
- 偏權值$b2$：偏權值$b2$模擬突觸的結構，代表接收神經元容易被活化的程度，偏權值越高，越容易被活化並傳遞訊息。如圖 2.10 所示，因為隱藏層共 256 個神經元，所以偏權值是長度為 256 的向量。
- 活化函數（activation function）：活化函數模擬神經傳導的運作方式，在此使用 relu 函數，接受刺激的總和：$(X \times W1 + b1)$，經過活化函數 relu 的運算，大於臨界值會傳遞給下一個神經元。

建立隱藏層與輸出層數學式子(2-7)：

$$y = Softmax(b1 \times W2 + b2) \tag{2-7}$$

以式子 2-7 模擬多層感知器模型運作，如圖 2.10 詳細說明如下：

- 隱藏層$h1$：隱藏層$h1$模擬內部神經元，共有 256 個神經元。
- 輸出層y：模擬輸出神經元，就是預測的結果，共有 6 個輸出神經元。對應到我們希望預測的數字，從 0~5 共有 6 個結果。
- 權重$W2$：權重模擬神經元軸突，連結輸入與接收神經元，負責傳送訊息。連結隱藏層（256 個神經元）與輸出層（6 個神經元），為了讓二層的每一個神經元都互相連結，總共需要 256×6 ＝ 1536 個軸突，所以權重（$W2$）必須是 256×6 的矩陣，用來模擬這些軸突的功能。

- 偏權值b：偏權值b模擬突觸的結構，代表接收神經元容易被活化的程度，偏權值越高，越容易被活化並傳遞訊息。如圖 2.10 所示，因為接收神經元是輸出層（6 個神經元），所以偏權值是長度為 6 的向量。
- 活化函數（Activation function）：在輸出層我們使用 Softmax 活化函數，接受刺激的總和：$(b1 \times W2 + b2)$，經過 Softmax 運算後的輸出，是一個機率分布，共有 6 個輸出，數值越高代表機率越高，以本研究手指姿勢辨識舉例：輸出結果由 0 算起第 6 個數字最高，代表預測結果是 Five。

2.4 倒傳遞（Back Progation）演算法進行訓練介紹

倒傳遞法是訓練人工神經網路的常見方法，並且與最優化方法（Optimizer），例如梯度下降法結合使用。倒傳遞是一種監督式學習
方法，必須輸入特徵值（Features）與真實的值（Labels）。

倒傳遞演算法簡單的來說就是「從錯誤中學習」。多層感隻模型辨識本研究手指姿勢影像，本文將結果整理如圖 2.11。

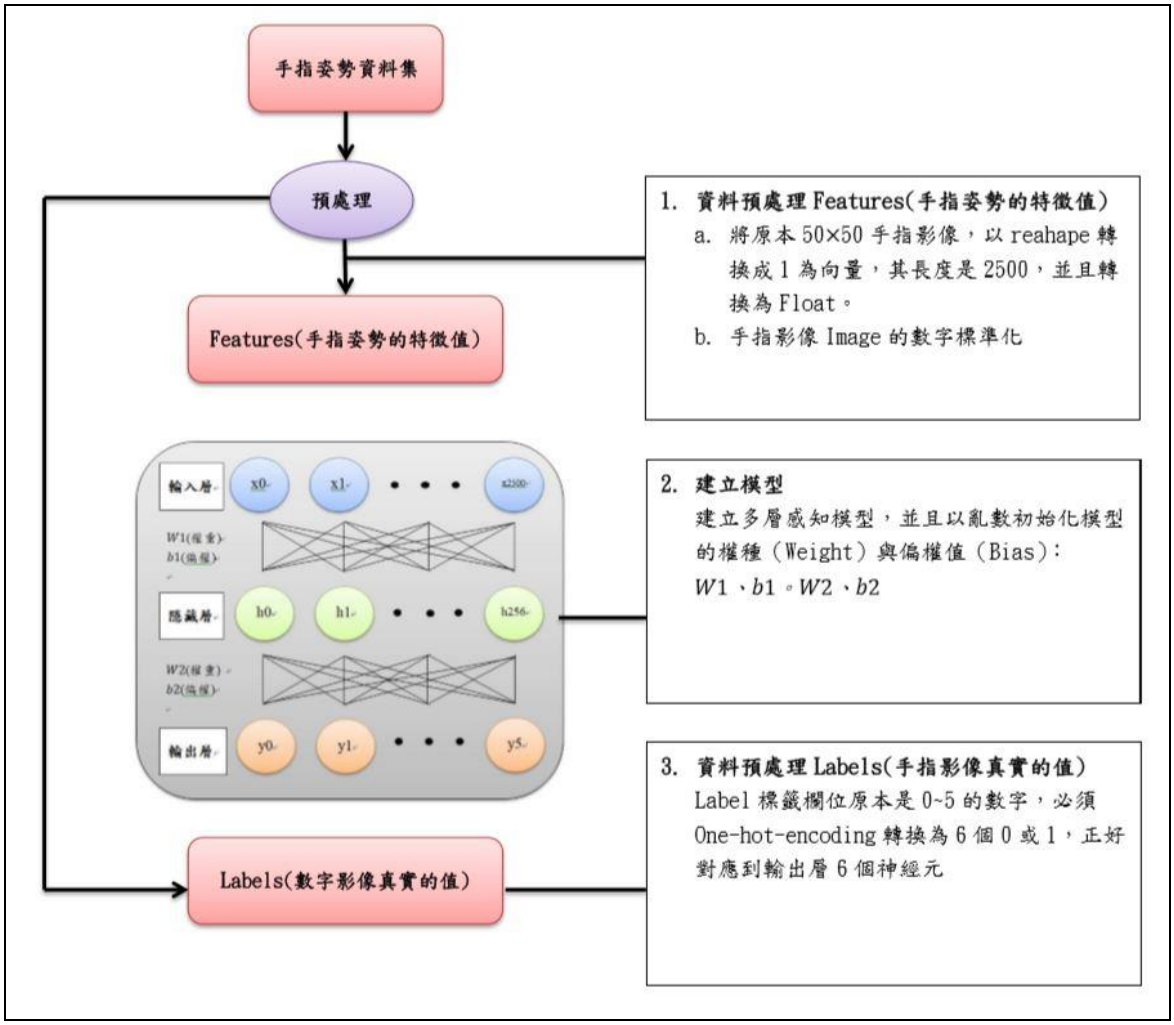


圖 2.11：多層感知模型辨識手指姿勢影像

1. 資料預處理：手指姿勢資料集經過預處理產生手指姿勢的特徵值（Features）、手指姿勢影像真實的值（Labels），作為後續訓練使用。
2. 建立模型：建立多層感知模型，且以亂數初始化模型的權重（Weight）與偏權值（Bias）：$W1$、$b1$、$W2$、$b2$。

進行訓練時，資料分為多個批次，以本研究手指姿勢為例，每一批次 72 筆資料，然後每次讀取一批次資料進行倒傳遞演算法訓練：

重複一下傳遞（Propagation）和權重更新（Weight update），直到誤差（Loss）收斂，如圖 2.12 所示。

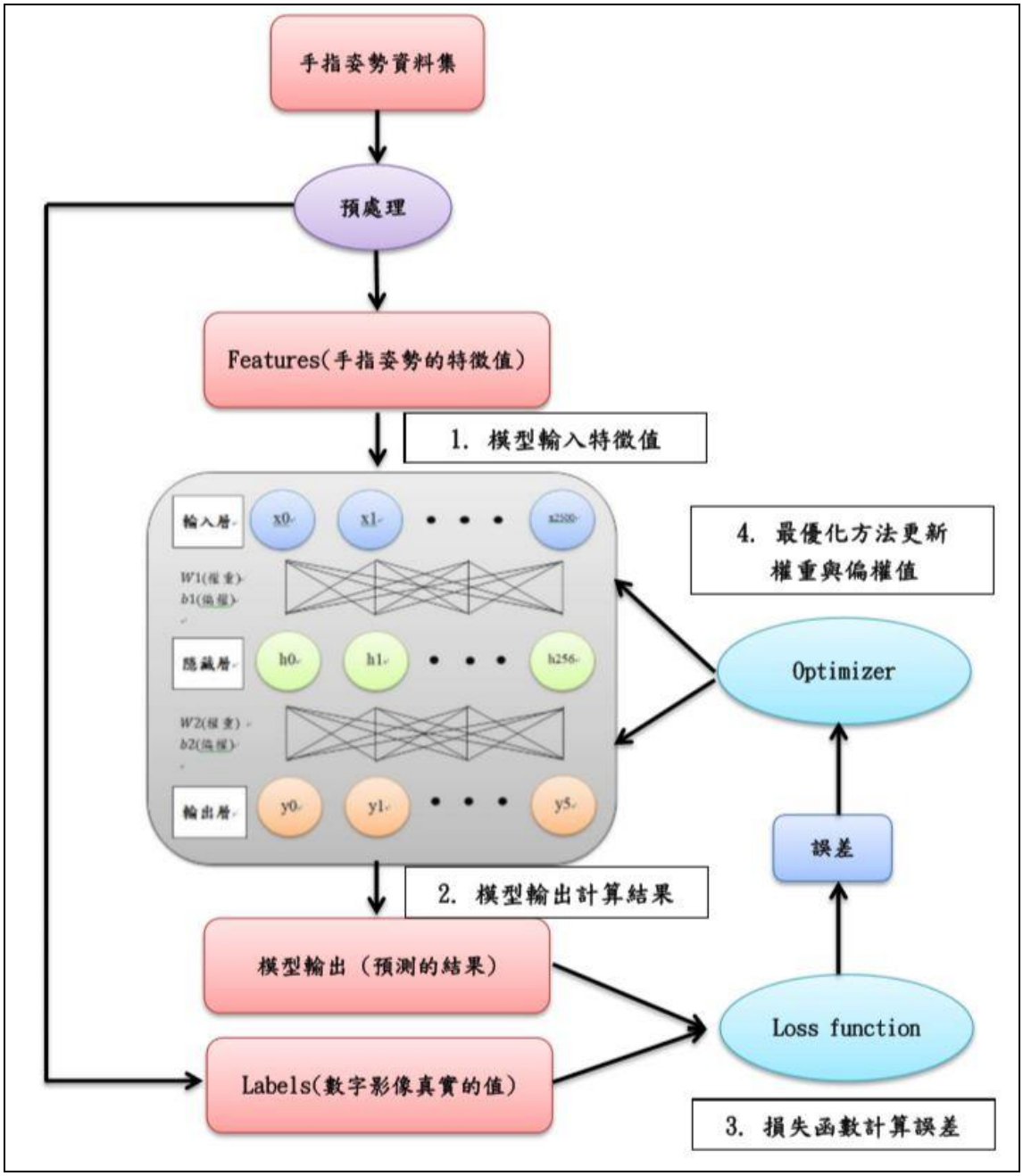
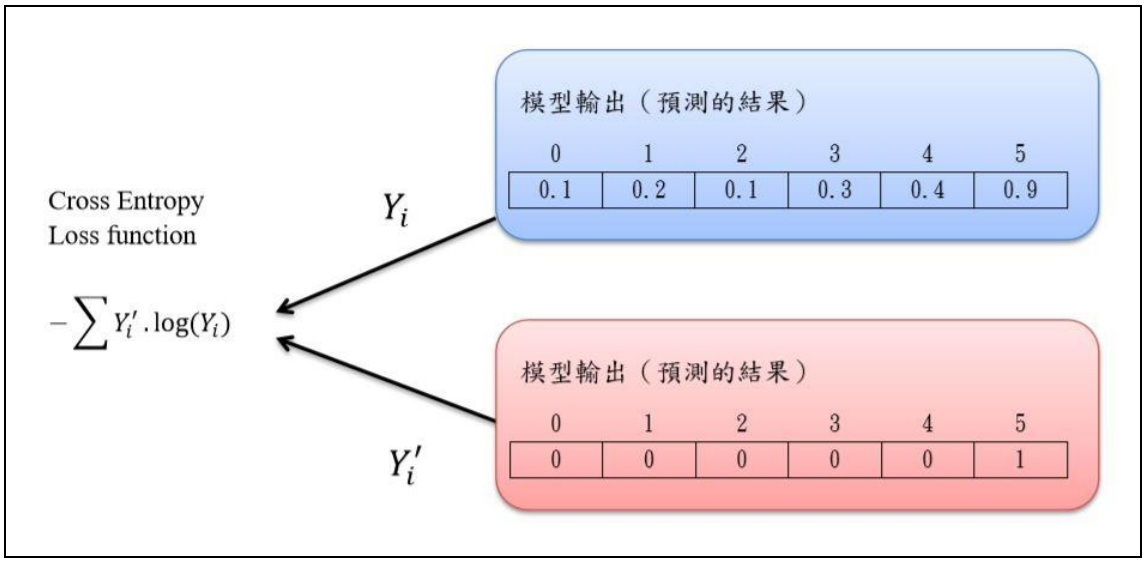


圖 2.12：倒傳遞演算法流程圖

1. 模型輸入特徵值：手指姿勢影像的特徵值（Features），輸入到神經網路計算。
2. 模型輸出計算結果：經過多個隱藏層進行計算，逐層向前傳遞，最後到達輸出層，產生神經網路的輸出。
3. 損失函數（Loss Function）計算誤差：使用損失函數計算，模型輸出（預測的結果）與數字影像真實的值（Labels）之間的誤差值。

4. 最優化方法（Optimizer）更新權重與偏權值：依照誤差值，更新神經元連結的權重（Weight）與偏權值（Bias），盡量使損失函數的誤差值最小化。

   倒傳遞算法，簡單的說就是「從錯誤中學習」，而損失函數就是計算誤差，Cross Entropy 是深度學習常用的損失函數，說明如圖 2.13。



圖：2.13 Cross Entropy

圖 2.13 損失函數計算：模型輸出（預測的結果）與手指影像真實的值（Labels）之間的誤差值，都是以 One-Hot-Encoding 表示，以本研究手指姿勢為例，預測手指姿勢為 5（Five）。

- 手指影像真實的值（Labels）：由 0 算起第 6 個數字是 1，其他都是 0。
- 模型輸出（預測的結果）：預測結果 0（Zero）是 0.1（10%），預測結果 1（One）是 0.2（20%）等等，預測 5（Five）是 0.9，代表預測手指姿勢 Five 有 90% 的機率，其他機率都不高。

最佳化方法（Optimizer）就是使用某種數值方法，在不斷批次訓練中，不斷更新權重（Weight）與偏權值（Bias），使損失函數的誤差值最小化。並且最終找出誤差值最小的「權重與偏差的組合」。

在深度學習中，通常是使用準確率梯度下降法（Stochastic Gradient Descent，SGD）來對「權重與偏差的組合」進行最佳化。準確率梯度下降法可以想像成，

是在所有「權重與偏差的組合」所組成的高維度空間中，每個訓練批次，沿著每個維度下降的方向走一小步，經過許多次步驟，就可以找到最佳化「權重與偏差的組合」。

真實的深度學習中權重（Weight）與偏權值（Bias）數量很多，會形成非常多維度的空間。本文為了方便說明，假設最簡單的情況，只有兩個權重 weight1（$w1$）與 weight2（$w2$），所以$w1$與$w2$可以畫出二維圖形，如圖 2.14。

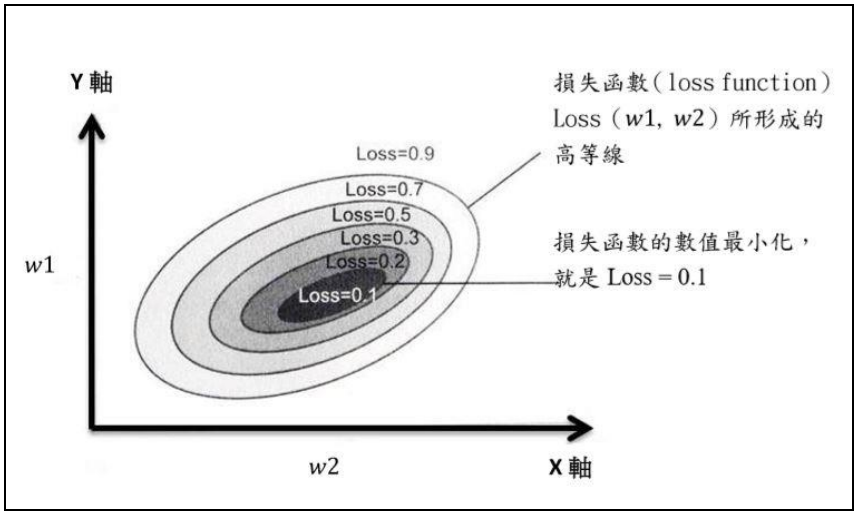


圖 2.14 $w1$與$w2$二維圖形

- X 軸是 weight1（$w1$）。
- Y 軸是 weight2（$w2$）。
- 損失函數（Loss Function），輸入兩個參數$w1$與$w2$。

依照 Loss（$w1$,$w2$）的數值，可以畫出由多個橢圓形所行刑的高等線，顏色越深，代表 Loss 數值越小。

SGD 準確率梯度下降法，就是每次沿著 Loss 下降的方向，每次訓練批次走一小步，經過許多訓練步驟，就能下降到 Loss=0.1，損失函數的誤差最小化。

2.5 Keras 介紹

Keras 是一個開放原始碼，高階深度學習程式庫，使用 Python 編寫，能夠運行在 Tensorflow 或 Theano 上。其主要作者維護者是 Google 工程是 François Chollet，以 MIT 開放原始碼。

Keras 可以使用最少的程式碼，花費最少的時間，就可以建立深度學習模型，並進行訓練、評估準確率，進行預測。相對的使用 Tensorflow 這樣低階的程式庫，雖然可以完全控制各種深度學習模型的細節，但是需要更多程式碼，花費更多時間開發，才能達成。

Keras 是一個（Model-level）模型級深度學習程式庫，Keras 只處理模型的建立、訓練、預測等功能。深度學習底層的運作，例如張量（矩陣）運算，Keras 必須配合使用「後端引擎」進行運算。目前 Keras 提供了兩種後端引擎（Backend engine）：Theano 與 Tensorflow。

如圖 3.1 所示，使用 Keras 時，只需專注於建立模型，至於底層操作細節，例如：張量（矩陣）運算，Keras 會幫你轉化為 Theano 或 Tensorflow 相對指令

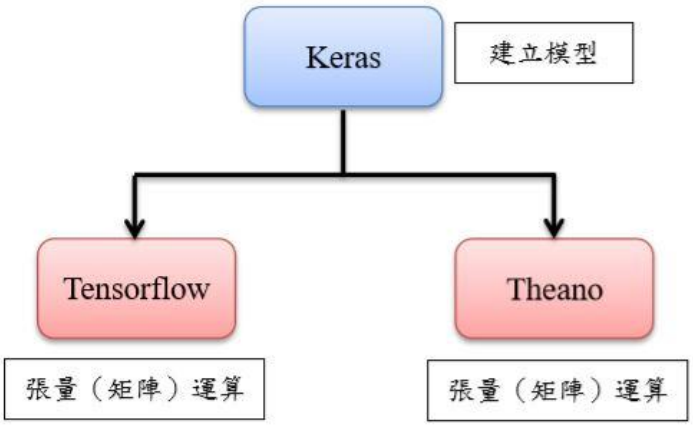


圖 2.15：Keras 運作方式

本研究手指姿勢辨識使用 Keras 處理模型的建立、訓練、預測等功能，Tensorflow 則為後端引擎。

2.6 Keras 特色

Keras 深度學習程式庫特色：

- 簡單快速的建立原型 Prototyping：Keras 具備友善的使用者介面、模組化設

計、可擴充性。

- 已經內建各式類神經網路層級，例如：卷積層 CNN、RNN，可以幫助本研究快速建立神經網路模型。
- 透過後端引擎：Theano 與 Tensorflow，可以在 CPU 與 GPU 運行。
- 以 Keras 開發的程式碼，更簡潔、更可讀性、更容易維護、更具生產力。
- Keras 的說明文件與非常齊全，官網上提供範例，相當淺顯易懂。

第三章 研究方法

本章節研究方法分為五節，分別為開發平台簡介、建立手指姿勢圖片、環境架設、卷積神經網路架構設計、最後是學習參數設定。

3.1 開發平台簡介

本研究採用個人電腦 Intel(R) Core(TM) I7-8750H CPU @ 2.20 GHz，NVIDIA GeForce GTX1060，16GB 的 RAM，以 Keras 作為軟體開發平台。Keras 是屬於高階的深度學習程式庫，可以使用較少的程式碼，花費較少的時間，就可建立深度學習模型，並進行神經元訓練，撰寫語言為 Python 程式，編譯器為 Jupyter Notebook，能讓使用者在瀏覽器撰寫及執行程式，作業系統為 Microsoft Windows 10 家用版。

3.2 建立手指姿勢圖片

本研究是透過訓練神經網路來進行手指姿勢之辨識，手指姿勢是根據人類使用一隻手伸出不同數量的手指來表示 0~5 的數字，本研究要辨識的手指姿勢圖片為訓練資料，辨識的手指姿勢分為 6 種類型，並將 6 種類型的手指姿勢分別放置於資料夾中，總影像樣本數為 900 張。

為了提高特徵值擷取，本研究採用膚色二值化處理。傳統上分別針對顏色 R、G 以及 B 等三原色設定一定的閥值來對膚色與其他非膚色的部分進行區分是很常見的提取手部圖像的方法，然而在影像辨識系統中，光線照射強度與所產生的陰影是一個會影響系統表現的重要因素，因此單純只靠對三種顏色設定閥值是無法充分的描述在不同光照之下膚色的統計特性。YCbCr 色彩空間比起 RGB 空間對於光照的影響比較沒這麼敏感，我們先將原始圖片轉換至 YCbCr 空間，並且在 YCbCr 空間建立閥值而能夠分離出手的部分。我們以式子(3-1)對於轉換後的圖像進行閥值的設定。

$$B = \begin{cases} 255, & if\ 50 \le Y \le 255; \\ & 90 \le Cb \le 155; \\ & 135 \le Cr \le 180; \\ 0, & otherwise. \end{cases} \tag{3-1}$$

以圖 3.1 為例，可以看到使用式子(3-1)對於轉換後的圖像，能夠分離出手的部分。

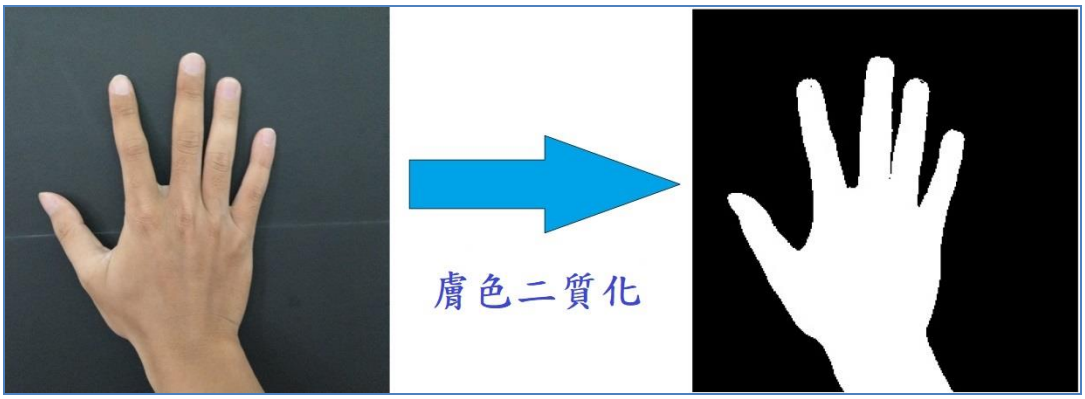
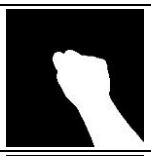
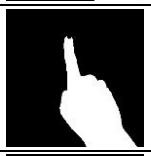
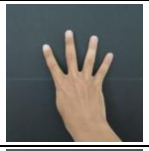


圖 3.1：膚色二質化

本研究使用手機拍攝規格為 3008x3008 像素大小的照片 900 張，其中每種類型的手指姿勢為 150 張，共為 6 類，如表 3.1 所示。為了減少學習模型訓練的參數個數，我們將照片縮放成為 50x50 像素大小，這樣原本一張照片可以大幅減少必須輸入的參數。本研究採用總影像樣本的訓練集 80%與驗證集 20%比例去做影像的收集，其中為訓練集為 720 張、驗證集為 180 張。

表 3.1 建立手指姿勢圖片

特徵 (手指數量)	資料夾名稱	樣本數	訓練資料 原始圖	訓練資料 經膚色二值化
0	a_zero_finger	150		
1	b_one_finger	150		
2	c_two_fingers	150		
3	d_three_fingers	150		
4	e_four_fingers	150		
5	f_five_fingers	150		

本研究為了驗證因應不同人所造成的手指姿勢影響會有不同的辨識結果，而選擇三位測試者來進行比對，準備 3 組測試者的 6 種手指姿勢照片，第一組為 85 張照片，放入 Taster_1 資料夾中，為測試者 1 號之測試集，第二組為 70 張照片，放入 Taster_2 資料夾中，為測試者 2 號之測試集，第三組為 145 張照片，放入 Taster_3 資料夾中，為測試者 3 號之測試集。最後本研究的資料結構為圖 3.2 所示。

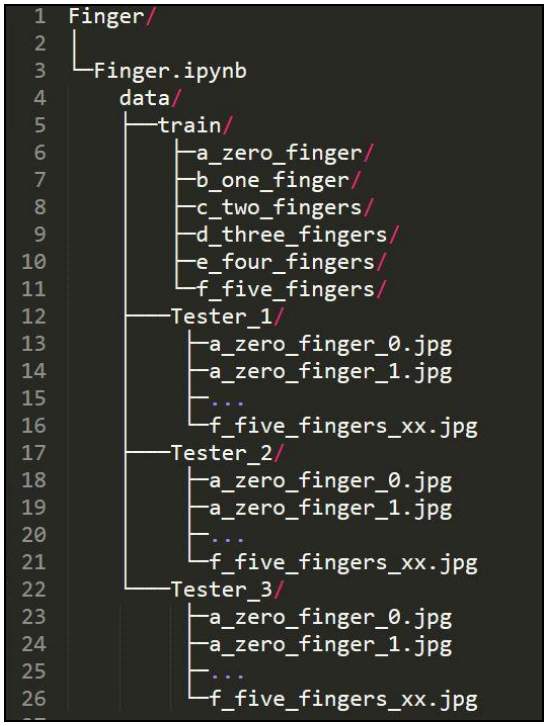


圖 3.2：資料結構

3.3 環境架設

本研究使用 Anaconda 官方下載安裝檔，Anaconda 是個開源軟體庫利用它來下載本研究需要的環境軟體及 API 套件，如圖 3.3 所示，主要的套件如：Tensorflow、Keras、Python、Jupyter Notebook、Opencv、Numpy、Matplotlib 等，建立完整的訓練環境架構。

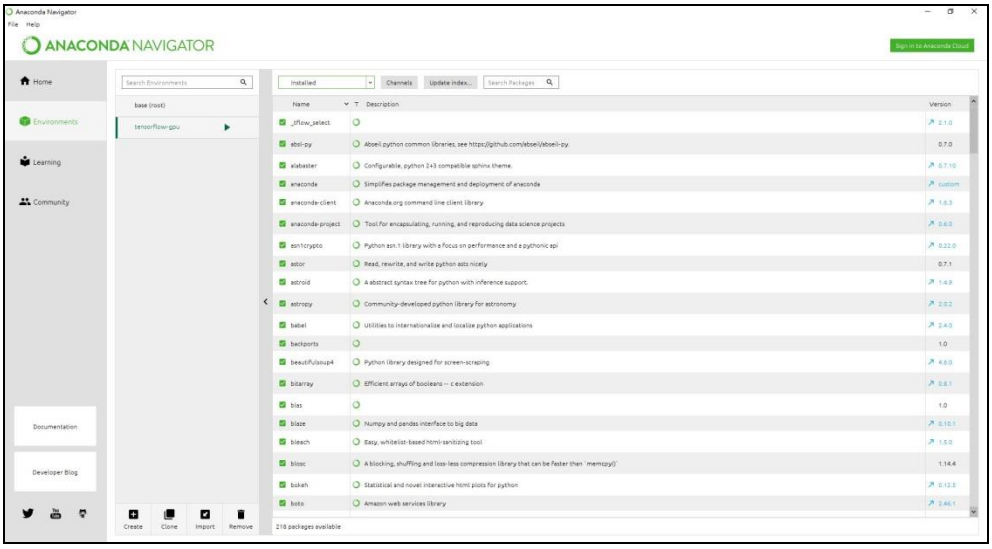


圖 3.3：Anaconda Navigator 介面

開啟命令提示字元輸入指令 conda create -n tensorflow--gpu pip python=3.7 來創建一個名為 tensorflow-gpu 的 conda 環境，python=3.7 代表這個虛擬環境的 python 版本，輸入完按下確認鍵，一段時間後顯示 Proceed ([y]/n)？，輸入 y，按下確認鍵，結果如圖 3.4。

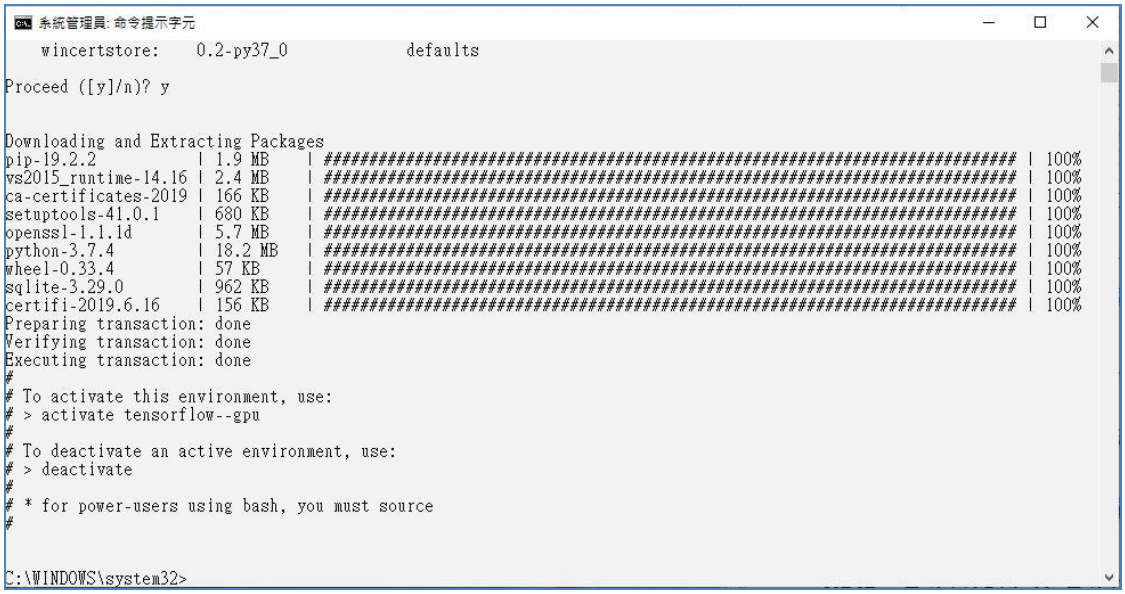


圖 3.4：建立 conda 環境

環境架設完成後，接著輸入 activate tensorflow--gpu，按下確認鍵，即可進入 conda 環境，結果如圖 3.5，後續安裝 API 套件以及套件更新都會在此環境中。

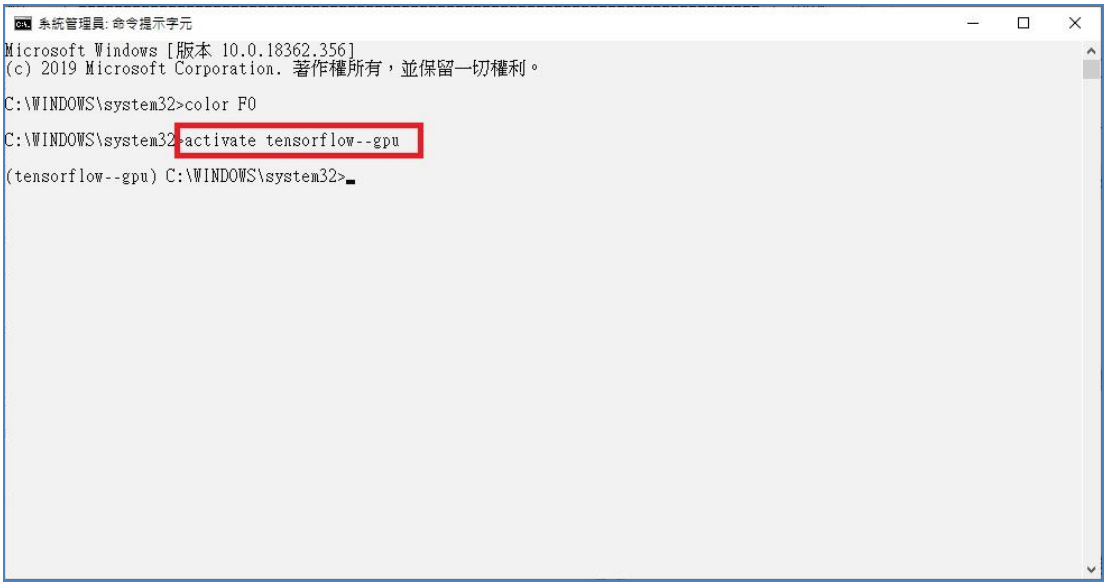


圖 3.5：進入 conda 環境

輸入以下指令 pip install --ignore-installed --upgrade tensorflow--gpu，在名稱為 tensorflow--gpu 的 conda 環境中安裝 GPU 版本 TensorFlow，並指定安裝版本為最新版，安裝時畫面如圖 3.6 所示。

圖 3.6：GPU 版本 Tensorflow 安裝指令

完成 GPU 版本 TensorFlow 安裝後，接著輸入指令 pip install keras 安裝 Keras，安裝時畫面如圖 3.7 所示。

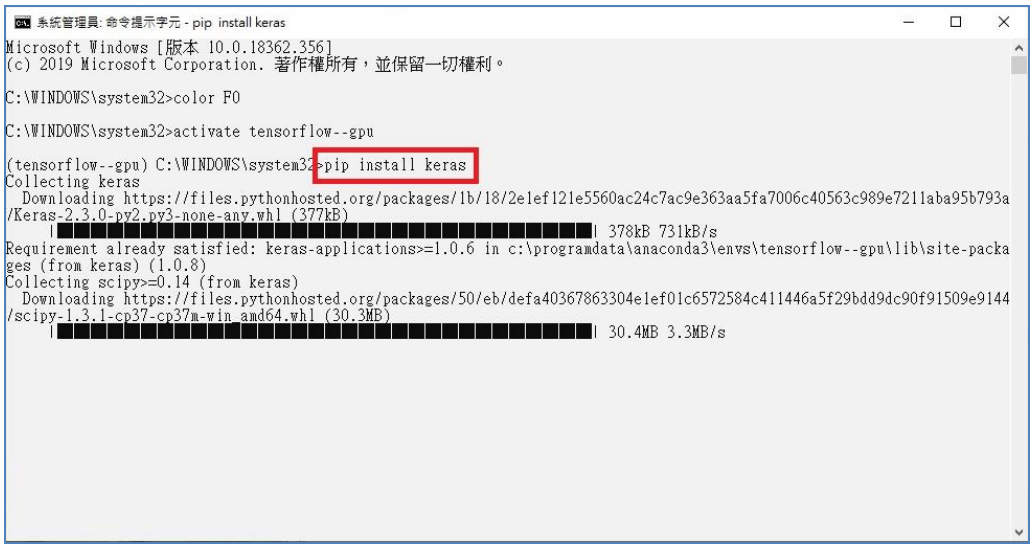


圖 3.7：Keras 安裝指令

按照以上安裝步驟，使用同樣的方式將本研究會使用到的套件安裝在 tensorflow--gpu 環境底下。所需安裝套件及指令如表 3.2 所示。

表 3.2：所需安裝套件及指令

pip install numpy
pip install matplotlib
pip install pickle
pip install opencv-python
pip install h5py
pip install glob2
pip install sklearn

完成以上 tensorflow--gpu 環境及套件安裝後，開啟新的命令提示字元並進入 tensorflow--gpu 環境底下，輸入指令 jupyter notebook 開啟 Web 瀏覽器，成功開啟畫面如圖 3.8 所示。

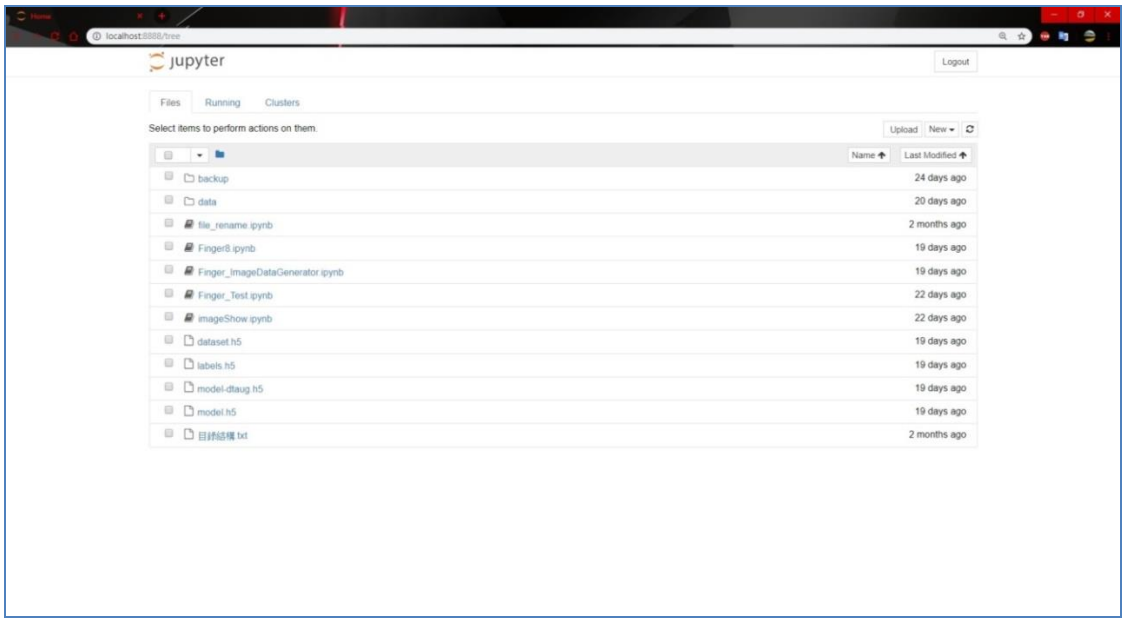


圖 3.8：開啟 Jupyter Notebook

3.4 卷積神經網路架構設計

在 Minist 辨識手寫數字中，使用了 2 層卷積層及 1 層隱藏層，辨識 Minist 中的手寫數字，其分類精確度接近為 99%〔林大貴(2017)。Tensorflow+Keras 深度學習人工智慧實務應用〕。不過這只是單色手寫數字辨識，相對來說比較簡單。本研究使用卷積神經網路，辨識手指姿勢，相對於辨識手寫數字來說難度較為高，參考了 Minist 辨識手寫數字的網路架構將卷基層的層數調整為 3 層及 2 層隱藏層，且在卷積層間使用 Dropout 來防止神經網路訓練時產生“過度凝合(Overfitting)”之情形。

表 3.3 的 Input(50x50)代表輸入圖片的大小為 50x50 的數字手指姿勢圖像。Conv-3x3-16，代表使用 3x3 的濾鏡 16 個，設定步幅為 1。Pooling-2x2，代表使用大小為 2x2 的池化層，且步幅為 2，以上為一次的卷積運算，卷積層一共為 3 層。Flatten 為平坦層，Dense-256，代表為 256 個神經元的隱藏層。經過 2 個隱藏層後，最後以 Softmax-6 作為輸出 (Output)，Softmax 可以將神經元的輸出，轉換為預測每一個影像類別的機率，如方程式(3-2)所示。

$$Softmax(s) = \frac{e^{s^i}}{\Sigma_j e^{s^j}} \tag{3-2}$$

表 3.3：卷積神經網路架構

Input (50x50)
ConV-3x3-16
ReLU
Pooling-2x2
Dropout-0.2
ConV-3x3-32
ReLU
Pooling-2x2
Dropout-0.2
ConV-3x3-64
ReLU
Pooling-2x2
Dropout-0.2
Flatten
Dense-256
ReLU
Dropout-0.2
Dense-256
ReLU
Dropout-0.2
Softmax-6
Output

3.5 學習參數設定

1. 損失函數（loss funtion）

神經網路會根據一個指標，尋找最佳參數，而這個指標就是「損失函數」(loss function)。損失函數是代表神經網路表現的「好壞」指標，它可以數值表示網路表現的效果好不好，神經網路會以此數值去調整參數，以下我們介紹幾種常用的損失函數。

本研究的損失函數（loss funtion）選擇使用交叉熵誤差(cross entropy error)，如方程式(3-3)，這裡用y_k代表神經網路的輸出，t_k是訓練資料的 label，k表示資料的維度，當L越小時，與正確答案誤差越小。

$$L = -\sum t_k \log y_k \tag{3-3}$$

2. 最佳化（Optimizers）

在前面的部分，我們已經得到損失函數（loss function），在最佳化理論使損失函數（loss function）越小越好，也就是找極小值。

為了找出極小值，本研究使用 SGD (stochastic gradient descent)計算參數梯度(微分)，重複執行，利用參數的梯度，往梯度方向更新參數，最後便會逐漸接近極小值。這樣的方法就是 SGD，中文稱作準確率梯度下降法，如方程式(3-4)。

$$W^{n+1} = W^n - \eta \frac{\partial L}{\partial W} \tag{3-4}$$

W^{n+1}為更新後的權重值，$\frac{\partial L}{\partial W}$為損失函數梯度，$\eta$為學習率，本文採用 0.01。

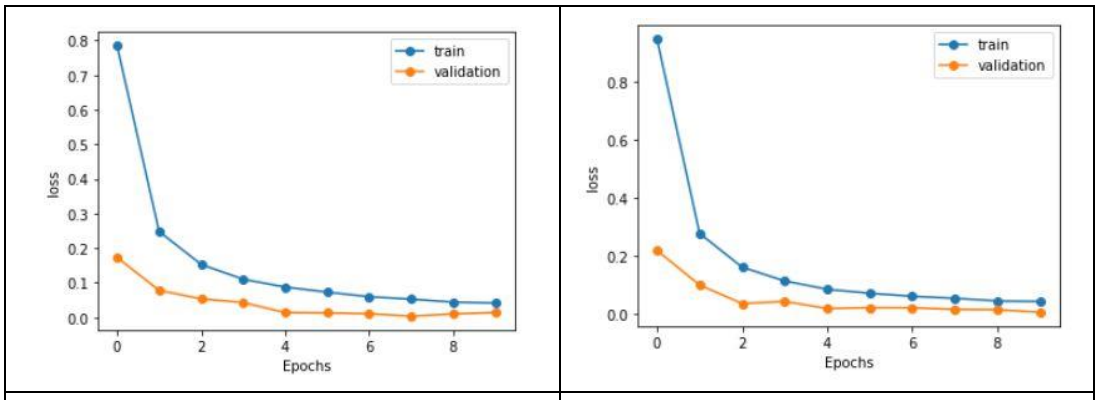
第四章 結果與討論

本章節結果與討論分為三節，分別為學習結果、圖像前處理對手指姿勢辨識率的影響以及預測結果視覺化。

4.1 學習結果

本研究共選用 1200 張圖片，其中 720 張為訓練集（train set），180 張為驗證集（validation set）以及三位測試者共 300 張為測試集（test set）。深度學習採批次訓練，批次大小設定為 72，訓練次數（epoch）設定為 10 次。測試結果討論如下：

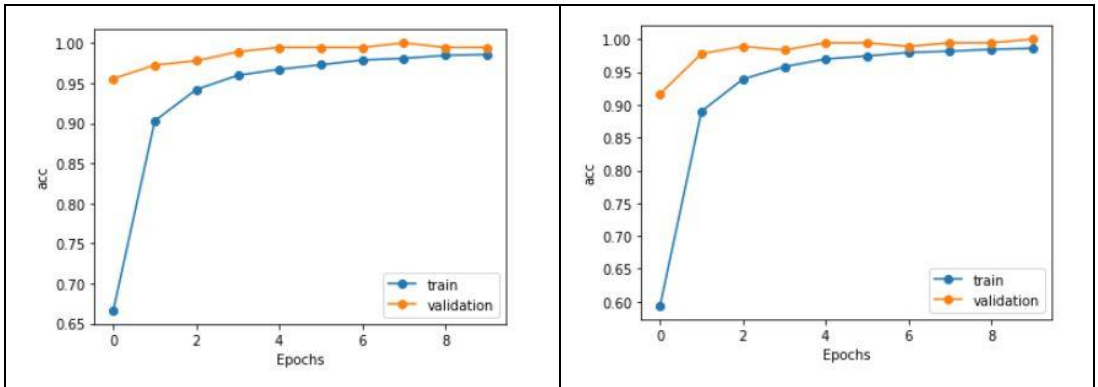
圖 4.1 為損失函數收斂結果，藍色曲線為「訓練集的誤差」，黃色曲線為「驗證集的誤差」。從圖中可以發現，不論訓練集與驗證集的誤差值皆越來越低，符合收斂的趨勢。



(a) 原始 RGB 圖像收斂結果　(b) 膚色二值化圖像收斂結果

圖 4.1：學習訓練收斂結果

在圖 4.2 為準確率結果，藍色曲線為「訓練集的準確率」，黃色曲線為「驗證集的準確率」。從圖中可以發現，不論訓練集與驗證集的準確率皆越來越高。



(a) 原始 RGB 圖像準確率結果　(b) 膚色二值化圖像準確率結果

圖 4.2：學習訓練準確率結果

4.2 圖像前處理對手指姿勢辨識率的影響

混淆矩陣（confusion matrix），也稱為誤差矩陣（error matrix），是一種特定的表格顯示方式，能以視覺化的方式，看出驗證結果是否混淆了兩個類，將其中一個手指姿勢的種類預測成另一個種類，對角線則為預測正確。本研究利用混淆矩陣來分別觀察 3 位測試者原始 RGB 圖像驗證的結果。如表 4.1 所示。以及 3 位測試者的二值化圖像驗證的結果。如表 4.3 所示

表 4.1：原始 RGB 手指姿勢圖像驗證結果

測試者 1 號混淆矩陣

	a_zero_finger	b_one_finger	c_two_fingers	d_three_fingers	e_four_fingers	f_five_fingers
a_zero_finger	14	0	0	0	0	0
b_one_finger	0	15	0	0	0	0
c_two_fingers	0	0	12	0	0	0
d_three_fingers	0	0	0	12	0	0
e_four_fingers	0	0	0	3	12	2
f_five_fingers	0	0	0	0	9	6

測試者 2 號混淆矩陣

	a_zero_finger	b_one_finger	c_two_fingers	d_three_fingers	e_four_fingers	f_five_fingers
a_zero_finger	10	0	0	0	0	0
b_one_finger	2	13	0	0	0	0
c_two_fingers	0	3	4	0	0	0
d_three_fingers	0	0	1	12	1	0
e_four_fingers	0	0	0	3	5	2
f_five_fingers	0	0	0	0	8	6

測試者 3 號混淆矩陣

	a_zero_finger	b_one_finger	c_two_fingers	d_three_fingers	e_four_fingers	f_five_fingers
a_zero_finger	21	0	0	0	0	0
b_one_finger	1	24	0	0	0	0
c_two_fingers	0	6	16	0	0	0
d_three_fingers	0	0	0	21	0	0
e_four_fingers	0	0	0	7	18	2
f_five_fingers	0	0	0	0	8	21

經由混淆矩陣試算結果可獲得不同圖型之學習結果，如表 4.2 所示。

表 4.2：原始 RGB 手指姿勢準確率結果

測試者 1 號							
種類	**Zero**	**One**	**Two**	**Three**	**Four**	**Five**	整體準確率
實際數量(張)	14	15	12	12	17	15	
預測正確(張)	14	15	12	12	12	6	
混淆數量(張)	0	0	0	3	9	2	
個別準確率(%)	100	100	100	89	63	52	83.5
測試者 2 號							
種類	**Zero**	**One**	**Two**	**Three**	**Four**	**Five**	整體準
實際數量(張)	10	15	7	14	10	14	
預測正確(張)	10	13	4	12	5	6	

混淆數量(張)	2	3	1	3	9	2	確率
個別準確率(%)	91	84	67	83	42	55	71.4
測試者 3 號							
種類	**Zero**	**One**	**Two**	**Three**	**Four**	**Five**	整體準確率
實際數量(張)	21	25	22	21	27	29	
預測正確(張)	21	24	16	21	18	21	
混淆數量(張)	1	6	0	7	8	2	
個別準確率(%)	98	87	84	86	68	81	83.4

表 4.3：二值化手指姿勢圖像驗證結果

測試者 1 號混淆矩陣

	a_zero_finger	b_one_finger	c_two_fingers	d_three_fingers	e_four_fingers	f_five_fingers
a_zero_finger	14	0	0	0	0	0
b_one_finger	0	15	0	0	0	0
c_two_fingers	0	0	10	0	2	0
d_three_fingers	0	0	0	12	0	0
e_four_fingers	0	0	0	0	17	0
f_five_fingers	0	0	0	0	2	13

測試者 2 號混淆矩陣

	a_zero_finger	b_one_finger	c_two_fingers	d_three_fingers	e_four_fingers	f_five_fingers
a_zero_finger	10	0	0	0	0	0
b_one_finger	0	14	1	0	0	0
c_two_fingers	0	1	6	0	0	0
d_three_fingers	0	0	0	10	4	0
e_four_fingers	0	0	0	0	10	0
f_five_fingers	0	0	0	0	1	13

測試者 3 號混淆矩陣

	a_zero_finger	b_one_finger	c_two_fingers	d_three_fingers	e_four_fingers	f_five_fingers
a_zero_finger	21	0	0	0	0	0
b_one_finger	0	25	0	0	0	0
c_two_fingers	0	3	19	0	0	0
d_three_fingers	0	0	0	20	0	1
e_four_fingers	0	0	0	4	23	0
f_five_fingers	0	0	0	0	3	26

經由混淆矩陣試算結果可獲得不同圖型之學習結果，如表 4.4 所示。

表 4.4：二值化手指姿勢準確率結果

測試者 1 號							
種類	**Zero**	**One**	**Two**	**Three**	**Four**	**Five**	整體準確率
實際數量(張)	14	15	12	12	17	15	
預測正確(張)	14	15	10	12	17	13	
混淆數量(張)	0	0	0	0	4	0	
個別準確率(%)	100	100	91	100	89	93	95.2
測試者 2 號							
種類	**Zero**	**One**	**Two**	**Three**	**Four**	**Five**	整體準
實際數量(張)	10	15	7	14	10	14	
預測正確(張)	10	14	6	10	10	13	

混淆數量(張)	0	1	1	0	5	0	確率
個別準確率(%)	100	93	86	83	80	96	90
測試者 3 號							
種類	**Zero**	**One**	**Two**	**Three**	**Four**	**Five**	整體準確率
實際數量(張)	21	25	22	21	27	29	
預測正確(張)	21	25	19	20	23	26	
混淆數量(張)	0	3	0	4	3	1	
個別準確率(%)	100	94	93	89	87	93	92.4

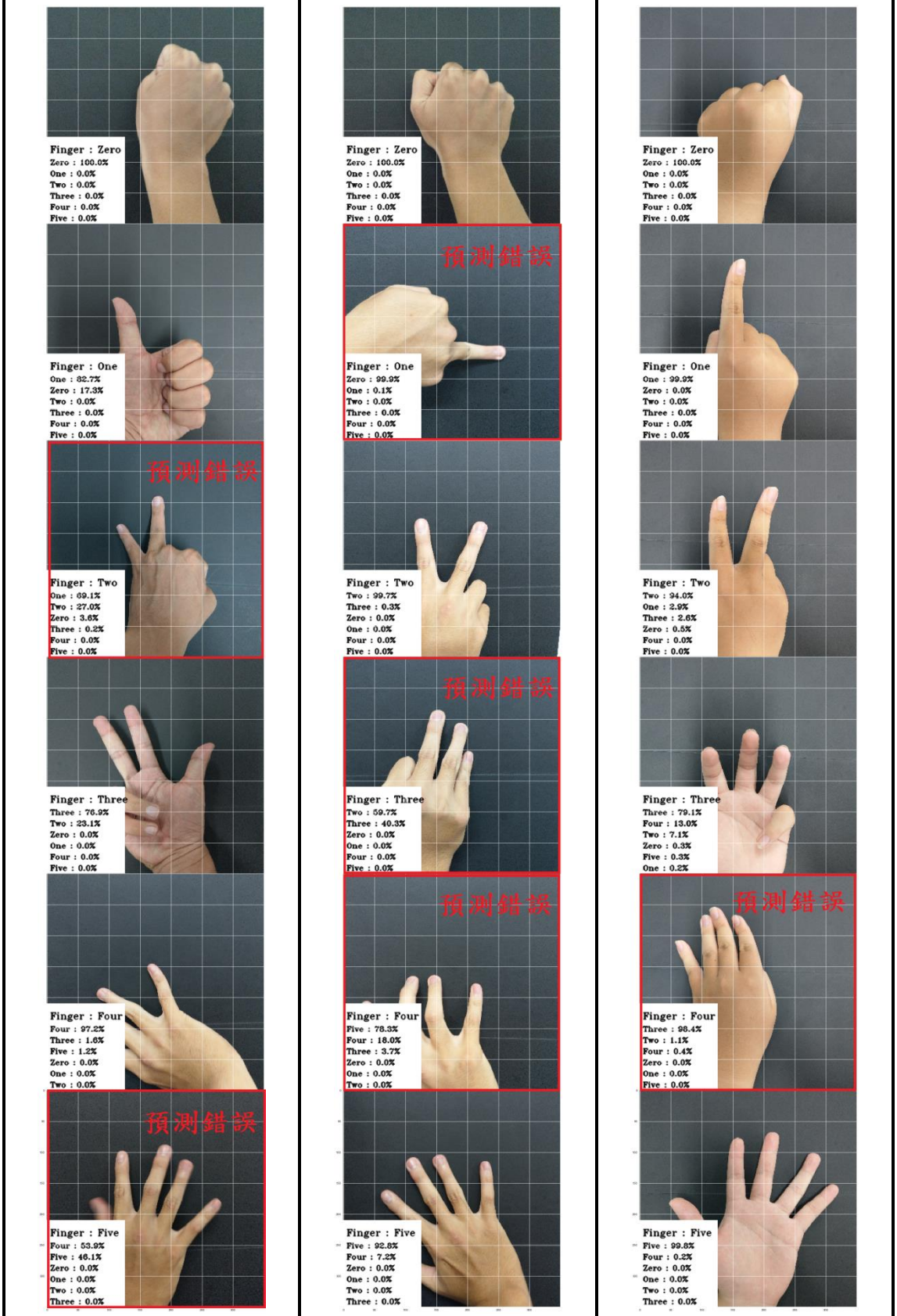
4.3 預測結果視覺化

將三位測試者的 RGB 手勢圖像及二值化手勢圖像預測的結果視覺化，並從 6 種手勢中分別隨機挑出 1 張並預測準確率。

從圖 4.3 中，本研究發現到即使背景是實驗室中很單純固定好光照環境的桌面上，由於三位測試者手的大小和擺放角度，很容易會有不同光影的影響，各種手勢都被包裹在影子之中，導致拍出來的影像對於手型的輪廓有著相當大的雜訊干擾，就算實際手型差異性很大，造成三位測試者 CNN 的整體準確率皆低於 85%，最低為 71.4%，所以辨識效果不夠好。

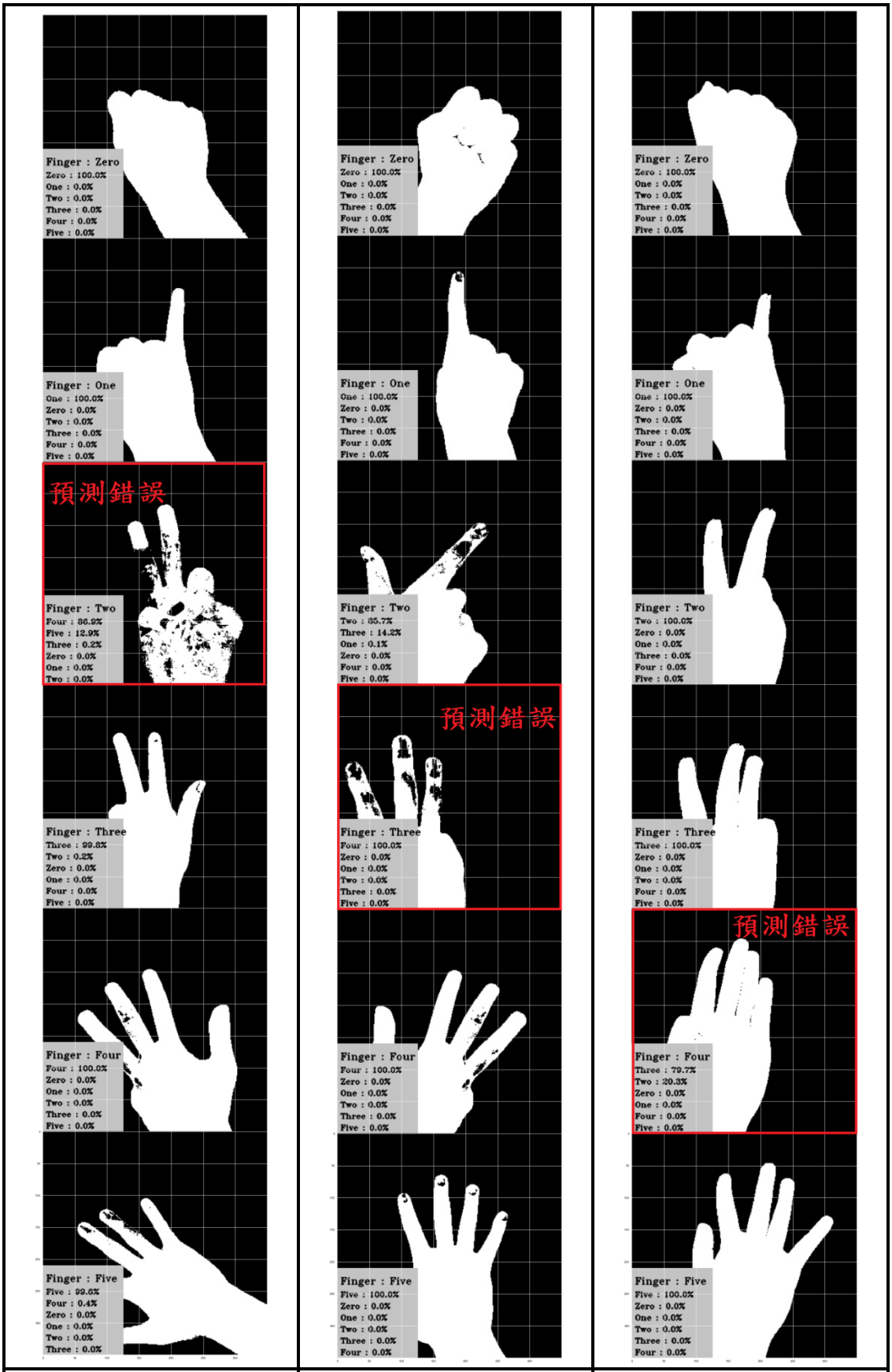
為了解決光影干擾的問題，圖 4.4 為我們對原始 RGB 手指圖像去除背景處理之後的二值化手指圖像，三位測試者 CNN 的整體準確率皆達到了 90%，特別是在手型差異極小的手指姿勢（Three）與手指姿勢（Four）的預測準確率結果中，手指姿勢的誤判個數有明顯減少。

從圖 4.4 中，可以發現二值化圖像預判錯誤可能的原因，第一個原因為原始圖像模糊，或者是測試者的膚色差異，而造成經過二值化後圖像邊緣有缺口，甚至是斷裂而造成預判錯誤。第二個原因為手指併攏及彎曲會影響個別準確率，或者是預判錯誤。



(a) 測試者 1 號	(b) 測試者 1 號	(c) 測試者 3 號

圖：4.3 RGB 手指姿勢預測結果視覺化



(a) 測試者 1 號 (b) 測試者 2 號 (c) 測試者 3 號

圖：4.4 二值化手指姿勢預測結果視覺化

第五章 結論與展望

本研究提出基於卷積神積網路之手指姿勢辨識，在圖像前處理時進行膚色二值化後進行訓練與學習，驗證了圖像前處理對於手指姿勢辨識率的影響有著明顯的差別，整體影像的預測的準確率上升幅度高達約 19%，且三位測試者的整體預測的準確率均達到 90%以上，實現深度學習技術在手指姿勢影像辨識研究的可行性。

本研究提出的手指姿勢辨識缺點為只能用於靜態手指辨識，在應用層面上有許多的不足，若能辨識動態中的手指姿勢，將有效提高應用層面。

在 2019GTC（GPU Technology Conference）大會中， NVIDIA 宣布一款嵌入式 AI 平台 Jetson Nano，硬體規格上提供完整的 Cuda GPU，Jetson Nano 有足夠的運算能力，快速執行現代人工智慧演算法，並可同時執行多個神經網路，以及同步處理數個高解析度感應器。在軟體方面則支援景深量測、物件識別、姿勢量測、手勢識別、路徑規劃等功能，另外也支援 TensorRT、cuDNN 等深度學習框架，以及 VisionWorks OpenCV 等電腦視覺應用。

未來將利用本研究所開發的卷積神經網路數據，結合嵌入式 AI 平台 Jetson Nano 開發以手指姿勢操控家電設備，以提高銀髮族或部分行動不方便人士之生活品質與便利性。

文獻探討

[1] V. I. Pavlovic, R. Sharma and T. S. Huang, "Visual interpretation of hand gesturesfor human-computer interaction: a review," Pattern Analysis and Machine Intelligence,IEEE Transactions On, vol. 19, pp. 677-695, 1997.

[2] G. V. Paul, G. J. Beach, C. J. Cohen, and C. J. Jacobus, Tracking and Gesture Recognition System Particularly Suited to Vehicular Control Applications, 2006.

[3] C. Maggioni, "A novel gestural input device for virtual reality," in Virtual Reality Annual International Symposium, 1993., 1993 IEEE, 1993, pp. 118-124.

[4] W. Hu, T. Tan, L. Wang, and S. Maybank, "A survey on visual surveillance of object motion and behaviors," Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions On, vol. 34, pp. 334-352, 2004.

[5] S. Mitra and T. Acharya, "Gesture Recognition: A Survey," Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions On, vol. 37,pp.311-324, 2007.

[6] A. Vrij and G. R. Semin, "Lie experts' beliefs about nonverbal indicators of deception," J. Nonverbal Behav., vol. 20, pp. 65-80, 1996.

[7] W. T. Freeman, K. Tanaka, J. Ohta, and K. Kyuma, "Computer vision for Computer games," in Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference On, 1996, pp. 100-105.

[8] J. Triesch and C. Von Der Malsburg, "A gesture interface for human-robot-interaction," in 2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG),

1998, pp. 546-546.