1. 環境介紹與流程
   1. 數據內容  
      我們利用App取得騎車者的加速度值並將數據分成訓練數據以及測試數據，其中訓練數據有92筆而測試數據則有41筆。
   2. 使用工具  
      所使用的程式語言為python，而training model我們是用TensorFlow來實現，繪圖的部分是利用matplotlib library來繪製。
   3. 流程圖

測試

載入資料

總加速度計算

有效資料篩選

資料預處理

統計分析

加速度是否大於threshold

True

摔車

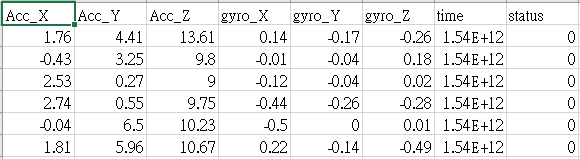
未摔車

False

深度神經網路

深度神經網路

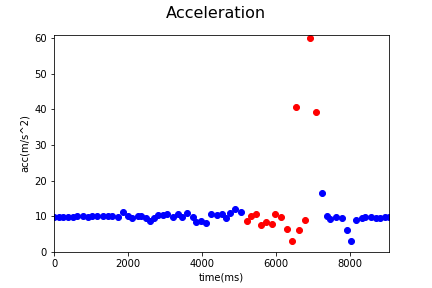
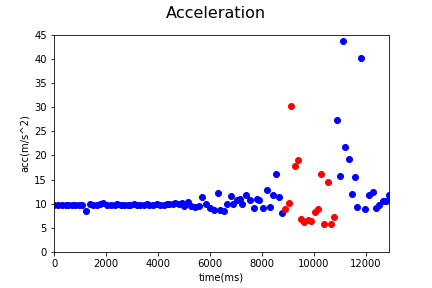
1. 資料預處理
   1. 總加速度計算   
      在App中我們紀錄了腳踏車行駛中的X, Y, Z三軸的加速度、時間以及狀態，如圖一，我們透過程式利用X, Y, Z的加速度來計算出總加速度，其計算方法為



圖一、實驗數據內容

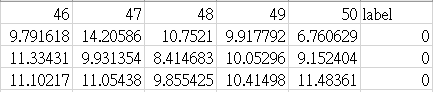
* 1. 有效資料選擇

因為我們從App上獲取的資料量長度都不一致，為了使後續的實驗能更公平且一致所以我們將前面步驟計算出來的總加速度與時間畫出2D關係圖(圖二)後，觀察到可以保留長度50筆的加速度量即可看出是否有摔車，因此我們刪除了前後的資料使資料長度變短，此作法不但達到了資料的公平性也縮短了後續需要處理的時間。



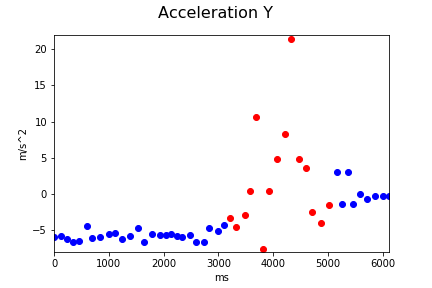
圖二、加速度與時間關係圖

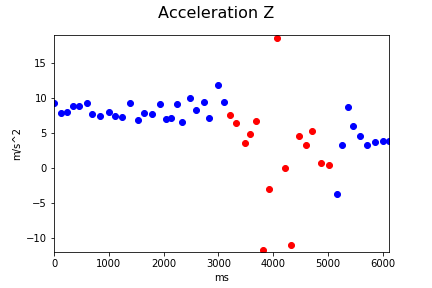
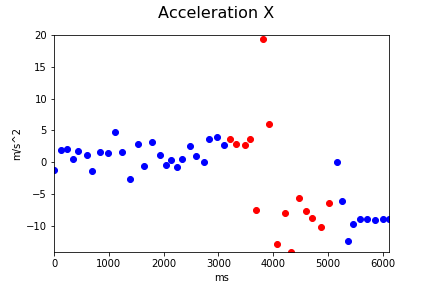
最後我們將所有數據的加速度輸出到同一個檔案裡，在此檔案中的資料包含50筆加速度以及一個狀態，此狀態表示摔車(1)以及未摔車(0)，部分資料內容如圖三。



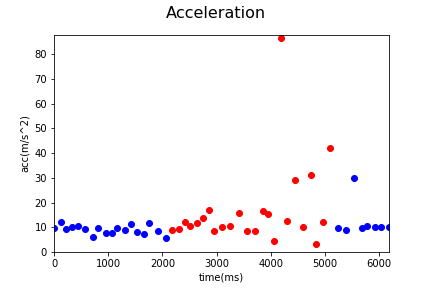
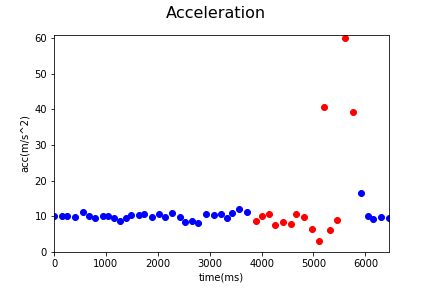
圖三、train\_data

1. 資料分析
   1. 加速度與時間關係圖  
      首先，我們將App取得的X, Y, Z三軸的加速度與時間分別繪製出二維的圖，如圖四，另外，我們利用X, Y, Z三軸加速度取出總加速度並繪製其與時間關係圖，如圖五。





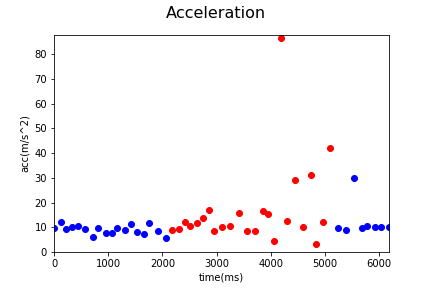
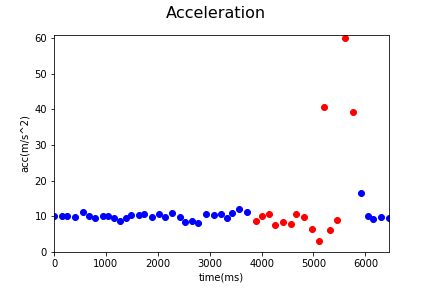
圖四、X, Y, Z三軸加速度與時間關係圖



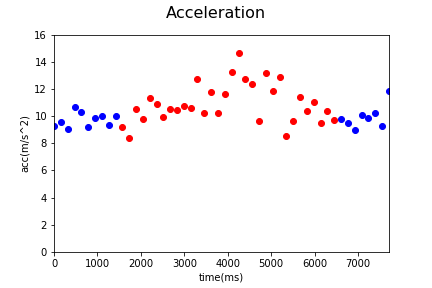
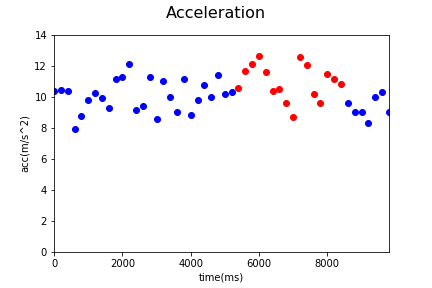
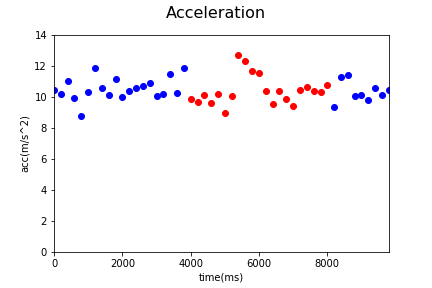
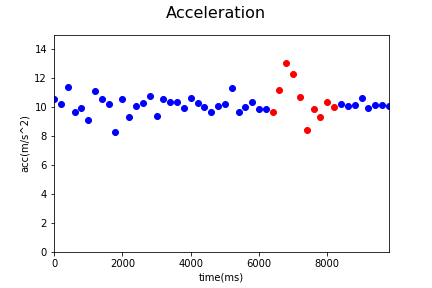
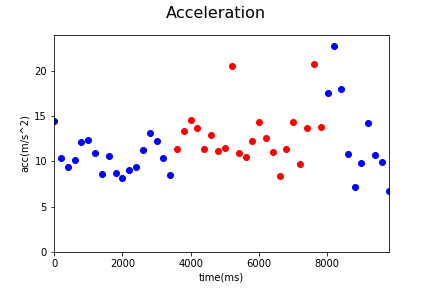
圖五、總加速度與時間關係圖

其中圖中的紅色點表示摔車狀態而藍色點表示未摔車狀態。

* 1. 數據種類  
     我們透過手機上的App獲取的數據，數據種類分成兩大類型，摔車以及未摔車，而其中摔車又分成向左側摔車以及向右側摔車，而未摔車的也分成了五類，分別是加速、煞車、左轉彎、右轉彎和迴轉，此目的是要使數據盡可能多樣化模擬各種可能與摔車有相關的情況，再利用前面提到的加速度與時間關係圖來辨別其中的差異以利後續做判斷使用。
  2. 數據分析  
     我們將這七種類型的資料繪製出加速度與時間的關係圖，並從關係圖中我們可以看出使用者有摔車(圖六)，以及使用者沒有摔車(圖七)之間的差異處。



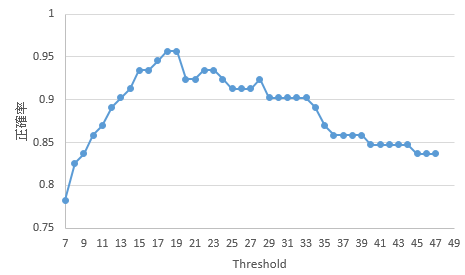
圖六、摔車加速度與時間關係圖，左圖為左摔而右圖為右摔



圖七、未摔車加速度與時間關係圖，由上到下由左至右分別是加速、煞車、左轉彎、右轉彎和迴轉

透過以上這兩大類型的加速度與時間的關係圖，我們觀察到當發生那些事件的當下所造成的加速度變化量，在發生摔車的圖中其加速的變化量會比未發生摔車的圖中來的大，因此我們認為可以利用統計的方法來實做，實作的辦法後面統計分析中會詳細介紹。而除了透過單純的統計分析來判斷摔車外，我們也利用了深度神經網路分類器來訓練出模組，藉此來比較兩者之間的好與壞，而其實作方法後面的深度神經網路中也會詳細的介紹。

1. 統計分析
   1. Threshold  
      根據前一步驟的數據，我們可以透過加速與時間的關係圖做分析，我們透過統計的方式歸納出摔車發生時其發生當下的加速度會突然的大幅增加，而未摔車的情況雖然加速度也有增加，但與摔車時增加的幅度比較是有明顯差異的，為了證實我們觀察到的結果，我們將訓練資料的加速度依據摔車與否分成兩大類並計算其最大和最小加速度差的平均值，得到的結果是發生摔車時其加速度差平均為48.65986 m/s2，而未發生摔車時其加速度差平均則為7.937981 m/s2，由此兩個數據我們可以明顯看到其差異。  
      此外，此兩數據在物理意義上也表示摔車與未摔車的數據大致分散在他們的平均值的附近，而我們為了取得更精確的分布情況，我們計算了他們各自的標準差，得到的結果是摔車加速度差的標準差為22.91112 m/s2，而未摔車加速度差的標準差則為5.581405 m/s2，因此我們可以透過加速度差平均值以及加速度差的標準差說明其分布情況，摔車時加速度差平均為48.65986 m/s2，而其加速度差的標準差為22.91112 m/s2，代表著絕大部分的摔車數據資料分布範圍在48.65986+22.91112 m/s2到48.65986-22.91112 m/s2的區間內，而相同的未摔車的加速度差平均為7.937981 m/s2，而其加速度差的標準差為5.581405 m/s2，代表著絕大部分的未摔車數據資料分布範圍在7.937981+5.581405 m/s2到7.937981-5.581405 m/s2的區間內，根據這兩個區間我們可以找出兩區間的中間值當作我們判斷摔車與未摔車的Threshold，其值為19.63406 m/s2，而我們為了驗證我們計算出來這個Threshold是否合理，我們將訓練數據根據不同的Threshold(從7到49)驗證其正確率曲線，如圖八



圖八、各Threshold對訓練數據造成的正確率

由圖八我們看出正確率最高的Threshold為18到19區間，因此我們可以驗證前面計算Threshold方法的可行性，所以我們就將Threshold設為19.63406 m/s2。

* 1. 正確率  
     利用上述統計分析得出的Threshold，我們首先使用原本的訓練資料做測試我們得到正確率為92.3913%，而後我們將測試資料放進model中做測試，我們會依據時間每十筆資料找出這區間內最大加速度以及最小加速度，並計算其差，若此加速度差大於Threshold則表示此筆資料為摔車數據反之則沒有摔車，最後所得到的正確率達到87.8048%。

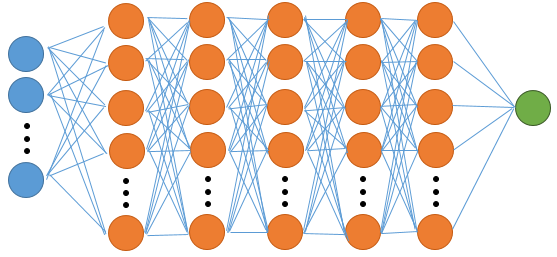
1. 深度神經網路
   1. 神經網路架構  
      我們利用最基本的神經網路架構，如圖九，其中輸入有50個資料，輸出有1個神經元，而hidden layers共有五層，各層中有20-50-50-20-10個神經元。

Input

(50 neurons)

5\*Hidden layers

(20-50-50-20-10 neurons)



Output

(1 neuron)

圖九、深度神經網路架構圖

實作方法我們使用TensorFlow中的DNNClassifier來實現，其中iteration為1500，並將跑完的model儲存起來。

* 1. 正確率  
     利用上述所提到的架構做完訓練後，我們首先將原本的訓練資料放入model中做測試我們得到正確率為100%，而後我們將測試資料放進model中做測試所得到的正確率達到87.8049%。

1. 結果分析  
   我們將兩種方法的正確率做比較，發現兩種方法對測試資料的正確率並沒有很大的差異，但由於深度神經網路我們可以根據新的資料去更新我們的model進而增加訓練資料量來提升正確率，而在實際生活中應用時我們會提供使用者做事後的反饋，當我們系統誤判時使用者可以回饋資料的正確判斷，我們也可以藉由此舉動來更新我們最初訓練好的model，經過長期使用下當使用者越多，此model也會跟著提升準確性，而反之雖然統計分析的方法其測試資料的正確性與深度神經網路的結果差不多，但在訓練資料上他並無法達到百分之百，引此總體表現上統計分析的方法還是沒有優於深度神經網路的方法，而在未來的更新方面我們即使可以改變他的Threshold使其更精確的判斷，但正確率很難會有更大的提升，因此比較下來若要能使得未來能更加精確的預測深度神經網路的架構會是較好的選擇。
2. 實作  
   我們透過TensorFlow並利用92筆訓練數據訓練出最初的model，並將此model檔案儲存起來，之後我們將此model檔案放置我們的App，而其在實際環境中運作的方法是，我們會將使用者行駛時所得到的加速度先計算其總加速度，之後當累積10筆數據(數據1-10)後會將此數據透過我們的model給出一個預測值，此預測值表示是否有摔車，接下來會繼續將下十筆數據(數據2-11)放入model中預測，直到發生摔車現象則我們就會回報或者使用者停車則我們會停止預測。