2018 年臺灣國際科學展覽會
優勝作品專輯

作品編號　190009

參展科別　電腦科學與資訊工程

作品名稱　使用深度學習構建足球競賽預測模型之研究

得獎獎項　大會獎：三等獎

比利時科學博覽會 BSE 正選代表

就讀學校　新北市私立康橋高級中學

指導教師　吳浩銘

作者姓名　謝之貽

關鍵詞　足球競賽預測、深度學習、類神經網路

作者簡介

我是謝之貽，目前就讀於康橋高中十一年級，從國中開始接觸程式設計後就非常喜歡寫程式，未來也希望能朝這個方向發展。平常除了參加程式設計競賽之外，課餘時間我會去學很多電腦科學的技術並做一些專題，讓平常所學的演算法與資料結構和生活應用相結合。這次的研究主題是我非常有興趣的足球比賽，過程雖然辛苦，但也讓我對人工智慧更加了解，獲益良多。

摘要

大數據時代促成運動賽事分析更加蓬勃發展，一般來說數據分析需要三個要素：資料、分析演算法和應用領域知識；隨著開放資料分享的普遍化，人們可以更輕易的取得資料並運用各式分析演算方法來針對有興趣的應用問題做出更精準的預測。

本研究以 kaggle 平台上所提供之歐洲職業足球比賽之公開資料集為基礎，使用目前最具分析潛力的深度學習技術—結合卷積神經網路（CNN）和全連接型神經網路的設計出一個五層的學習架構，建置出足球比賽結果的預測模型。此模型可直接預測主隊勝、負以及平手等三種結果，實驗結果亦展示出本研究所建置的 SoccerNet 預測模型優於過往的研究，有著更佳的預測能力；同時也驗證了使用公開資料集與 CNN 技術在球賽分析的可能性。而本研究所提的 SoccerNet 模型不僅可以運用於賽前的結果預測，亦可運用於球隊經營管理等決策，頗具有商業價值。

Abstract

The booming of big data has led to the development of sport games result prediction. In data analysis, the three main elements are: data, algorithms, and domain knowledge. As open data began to popularize, people all around the world are now able to build projects, such as game prediction, through applying algorithms to the open data.

In our research, we used data from kaggle and constructed a five-layers deep learning neural network model with Convolutional Neural Network (CNN) and Fully-Connected Neural Network to predict the result of European professional soccer games. Our model can directly predict if the home team would win, draw, or lose. The result of our experiment has proven that our SoccerNet performs better than previous work and demonstrates the possibility of sport games result prediction using open data and CNN. This SoccerNet model not only can predict the result, but it can also provide more information for team managers; therefore, it has commercial value.

壹、 前言

一、 研究動機

運動產業向來是一個有利可圖的生意，同時也對全世界的經濟有著巨大的影響，所以儘管運動競賽被認定是一項難以預測的任務之一，它仍然吸引著無數研究者前仆後繼的努力。而足球長久以來一直是全球最受歡迎的運動之一，因此足球競賽的預測堪稱為一個攸關全世界二百多個國家千百億營收的產業[17]，著眼於此，本研究擬將研究主題聚焦於足球競賽結果的預測。而為了更精準的預測競賽的結果，各球隊、球員和比賽等資料不間斷的被收集與分析，其中不乏運用統計或應用經濟的思維在分析資料，而與之搭配的資料分析工具主要是統計或人工智慧中的機器學習演算方法。

然而當我們在建置模型以預測足球競賽結果的時候，許多學者致力於以統計方法來研究各變數對球賽結果的影響，以期找到最關鍵的變因進行更為準確的預測；然而在機器學習的領域之中，分析資料的方法本質上傾向於使用演算法或模型去建構變數間的關連關係，而不是探討變數間的因果關係。除此之外，部分統計模型分析因果變因的研究中所使用的變數，有些資訊並不是我們可輕易的從公開資料(open data)中取得的，因此難以被其它研究者複製以進行更多的探索。所幸人工智慧機器學習的演算方法，在本質上就是設計來解決這種事前不容易做精準變數挑選的問題，其中深度類神經網路學習是近期與可見未來中表現最為突出的技術之一。

此外隨著大數據時代的來臨，多樣足球數據的收集也成指數倍速成長，更進一步促成了運動科技的快速發展，愈來愈多的決策試著更科學的倚賴實證資料(Evidence-based)來降低不確定性以提升各種預測的品質，期望更精確的掌握投資效益。但由於這些透過資訊新科技或應用所收集的數據資料往往更多更亂，在這樣的情境之下，使用機器學習方法來達成預測模型的建置不失為一個好方法。基

於上述緣由，本研究擬採深度類神經網路學習的方法，使用 kaggle 開放資料平台所提供的公開資料來進行預測模型的建置。

二、 研究目的與構想

本研究目的主要是聚焦於運用卷積神經網路(CNN)深度學習的技術於 kaggle 平台所提供之國際足球總會(FIFA)球賽和球員的公開資料，以建置出一個**預測足球競賽結果：主隊贏(Home Win)、平手(Draw)和主隊輸(Home Lose)的模型**，並藉此**實證採用公開資料集可以建置出準確的競賽結果預測模型**。以下簡單說明本研究建置模型時的研究構想：

（一） 球賽之勝負結果會受參賽球員之能力特性、雙方球隊之綜合能力特性、以及最近比賽成績影響。

（二） 球員之能力特性可以被一般化(generalize)，而成為通用的特徵，當使用這些通用的特徵來表現球隊中的球員時，就可以用共享的模型參數來進行學習，以增加數據的數量，減少參數的個數，降低過度適應(overfit)的機會，此亦即是採用卷積神經網路(CNN)的優點。

（三） 比賽雙方球隊的能力特性，亦可以被一般化，而構成中間層神經網路單元的通用特徵，亦即使用另一層獨立的卷積神經網路(CNN)來學習。

（四） 球員及球隊之外的其他資料，主要是各球隊過往的比賽結果資料，由於這部分是採用開賽前最近的特定滾動(rolling)次數進行建模，非為通用特徵，故不適合使用卷積神經網路(CNN)，因此將此訓練資料分開，使用全連接模型的神經單元，匯入於上層的神經網路。

三、 文獻探討

本小節擬對過去的研究者針對建置的足球預測模型的相關研究做簡單的文獻整理，接下來再針對本研究所使用到的類神經網路深度學習的技術做介紹。

（一） 足球預測

根據國際足球總會(FIFA)近期的估計─全世界有 2.65 億的人口在玩足球[20]，是故隱含在各職業、非職業或國家代表隊的足球賽預測，所帶來的龐大經濟利益不言可喻。概括來說，過去研究主要有兩種不同的方向：第一種是建構模型來預測比賽時參賽隊伍會進幾球與被進幾球，並針對這些可能的結果估算其相對應的機率；而第二類的作法則是直接針對比賽結果「贏、平手或輸 (win-draw-lose)」來建置預測模型。

第一類的研究由 Moroney [26]於 1956 年率先提出，研究中使用卜瓦松機率分配與負二項機率分配來進行足球得分的預測，後續研究如[3]實證這兩種模型的效用。Macher [22]則於 1982 年使用卜瓦松機率分配來預測主隊(Home Team)和客隊(Away Team)的得分，所採取的變數包括主客隊的攻擊、防守參數以及主場優勢等，後續研究[5]延續了這個模型並實證了卜瓦松機率分配比負二項機率分配有更好的預測成效。但由於上述研究假設各球隊的得分會依循卜瓦松機率分配是一個可被挑戰的議題，也就是說有不少研究者認為球隊的得分應該不止跟時間有關連。此外，過去的研究中亦指出攻擊或防守參數並不能完全代表得分能力[29]，[6]則進一步實證球隊和隊員排名對得分的機率有所關連。

而早在 1995 年時，Wilson 就曾使用類神經網路對美國大專足球隊進行排名的預測[34]；後續的研究[28]採用監督式多層神經網路(multilayer perceptron)來預測國家美式足球聯盟(National Football League)；[13]於 2010 年則更進一步採用倒傳遞(Back-Propagation)多層神經網路來預測 2006 年的世界盃足球賽的勝率，該研究也深入的探討各種類神經網路學習時的參數設定。

然而近期的研究則比較多傾向直接預測球賽輸贏的第二類研究，Koning [18]在 2000 年提出一個多元概率比回歸模型亦稱為序位多元概率比(Ordered Probit)迴歸模型來預計荷蘭足球競賽；[9][7]延續這個模型針對英格蘭足球賽資料與博彩公司的預測進行比較，實驗結果顯示所建置的模型不亞於博彩公司的預估。然而隨著資料探勘與機器學習領域的發展，統計迴歸模型以外的分類模型也被運用

在預測足球比賽的結果(win-draw-lose)，Byungo 使用單純貝氏分類器(Naive Bayes Classifier，簡稱 NBC)來預測足球競賽的結果[25]，Kampakis 則運用隨機森林(Random Forest)的方法來建置預測模型。[16] 以上的研究都實證了分類方法在預測足球競賽的結果十分有用。

不論是第一類或第二類的研究，過往文獻有不少從事這些對比賽結果可能有因果關係的變數進行探討的，[33]曾對這些研究仔細做統整，簡單描述相關研究中曾經考慮過的變數歸整如下：本季表現、同競爭對手的過去比賽結果、連贏/輸、主場優勢、明星球員、足球技能、攻擊/防禦策略、晉級/淘汰賽、賭盤賠率、球隊預算、球隊被升級前的成績、離比賽場地的距離、管理團隊的更替、疲累度、專家預測和國家隊的球員等十六項。上述研究一般都有綜合考慮其中幾項，而非擺放全部的變因在模型之中，最主要的原因是某些變項的資料可能要額外再收集和計算或做特定定義才會有資料可用，如：球隊預算、離比賽場地的距離、管理團隊的更替、疲累度、專家預測…等等，致使這些變項往往無法直接從公開資料集中取得。再者，就一般統計模型或機器學習模型在處理這麼多變項時，常常要利用一些降維的方法(reduction)如主成份分析，把變數先歸整成一個代表變數，才能間接進行預測。

而針對第一類和第二類研究的預測效果：2005 年時[8]曾針對此二種的模型進行比較，研究的結論顯示，雖然當時的研究者普遍認為第一類模型由於採計較多的細節資訊應該可以表現勝出，但實驗的成果顯示採用第一類的各種共變數來預測第二類模型的依變數會達到最好的預測輸贏成果。而在 2013 年時，[31]所進行的實驗則更進一步實證直接預測比賽結果的模型比第一類間接預估法來得更為準確。因此，我們將研究的主題設定在第二類直接建置模型來預測球賽的輸贏結果。

[4] 曾指出足球賽可算是最難預測競賽結果的運動賽事之一，有許多的變數如角球、射球、傳球、違規、舉牌、鏟球和得分…等都是有可能對比賽輸贏有潛在的影響，除此之外，若要考量列入球隊近期的競賽表現恐怕會有無限的排列組

合需納進入模型的相依變數之中。因此如果要對所有的變數可能的排列組合一一建置所屬的模型來進行檢驗恐怕會陷入研究困境。

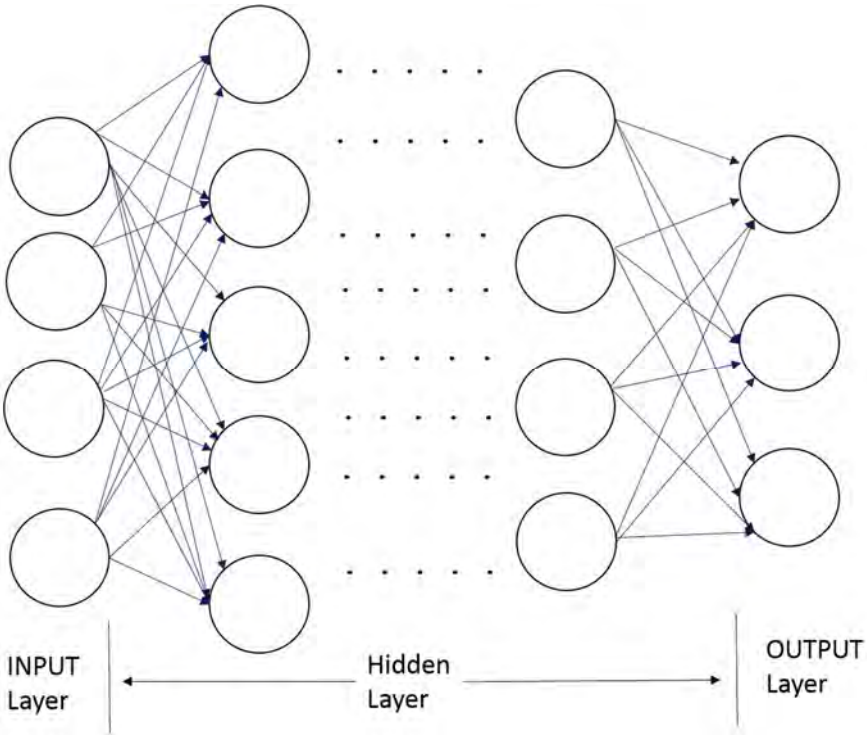
所幸隨著大數據時代的來臨，大量的資料被收集與分析，由於巨量資料的特色包括擁抱不精確，放寬允許的誤差值，利用手中更多的資料看到全貌，從精確走向可能性。所以巨量資料在不容易辨識因果關係的同時，其實只要找到有用的指標和相關性，就能觀察的更快也更清楚[24]。而在眾多數據分析的技術之中，使用類神經深度學習網路來建置高維度(多自變數)的分類模型時顯得特別有效率，相關技術的介紹將在下一小節中呈現。

（二） 類神經學習網路

人工神經網路的研究可以追溯到 1957 年 Rosenblatt 提出的感知器模型(Perceptron)。但一直到 80 年代產學界才重新對人工神經網路感到興趣，也致使神經網路的復興。人工神經網路(Artificial Neural Network)，簡稱神經網路(Neural Network)或類神經網路，主要是模仿生物神經網路特別是大腦的結構和功能之數學模型或計算模型，用於對函式進行估計或近似。它反映了人腦功能的基本特徵並將其簡化抽象和模擬。

神經網路越來越受到關注，主要是因為它為解決高複雜度問題提供了一種相對有效且簡單方法，神經網路可以輕易的解決具有上百個參數的問題。在結構上，可以把一個神經網路劃分為輸入層、輸出層和隱含層。圖一中的圓圈代表的是類神經網路單元亦稱神經元(Neuron)，輸入層的每個一神經元對映到各可能影響最終結果的自變數，而輸出層則用來表示所欲學習的目標變數，中間隱含層則代表從輸入到輸出結果中間的神經學習網路，這部分對一般使用系統的人們來說是個黑箱作業。而中間隱含層數的多寡和各層的神經元個數決定了神經網路的複雜度。過去或早期的研究礙於電腦的運算能力或模型參數調整的問題，類神經網路解決問題的成效有限，而近年來隨著電腦運算能力大幅的提升，密集且多隱含層的層數與各層神經元數可被設定與計算，深度學習的概念在 2006 年首度被 Hinton 所

提出。[11]



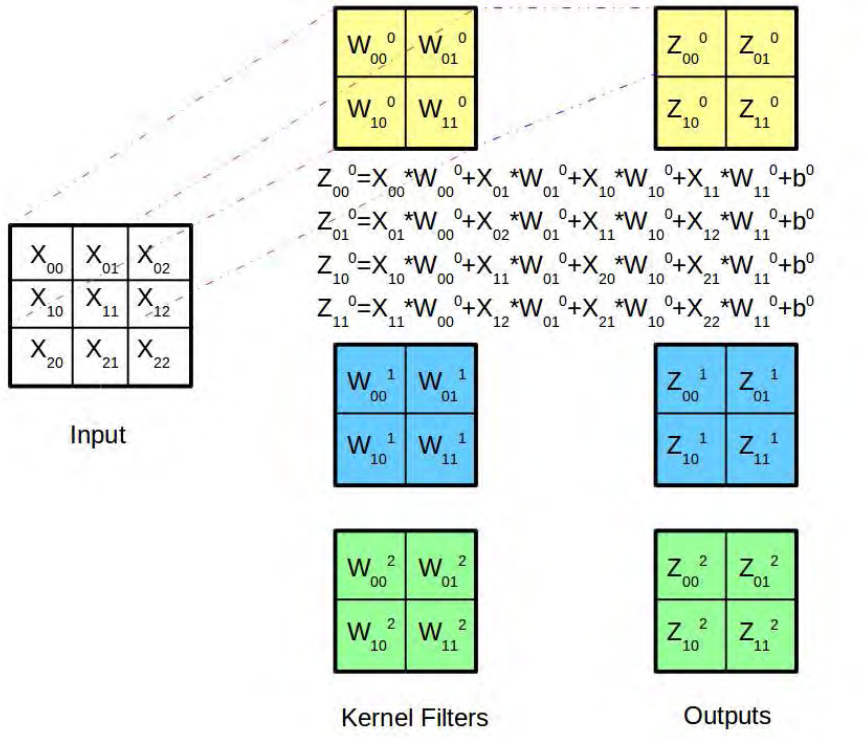
圖一、類神經網路圖

2016 年 3 月 AlphaGo 以四比一擊敗李世乭後，AlphaGo 所使用的深度學習技術引起了各界的關注。但從 iPhone 的語音助理軟體 Siri 說的語音辨識到 facebook 可以自動找出相片中的人臉，深度學習的模型在電腦視覺及語音識別都已經達到了顯著效果[30]。

簡單的說，深度學習是類神經網路學習的延伸，因應大數據時代量大且龐雜的資料量，加上電腦硬體運算能力的增強，以及對於如何訓練學習模型知識的增加，人們可以把更多更複雜的資料當作輸入，透過多層且多神經元的隱含層之運算，自動學習以找出有用的計算函數(function)。深度神經網路常被認為是具有 3 層以上有參數的神經網路，與淺層神經網路類似，深度神經網路也能夠為複雜非線性系統提供建模，但更多的層次為模型提供了更高的抽象層次，因而提高了模型的能力。

而其中卷積神經網路(Convolutional Neuron Networks，CNN)首次被 LeCun 等人在 1998 年提出[21]，它是當時唯一可以用在三層以上的神經網路來進行影

像分析的分類模型；而在深度學習的概念被提出之後[12]，2012 年 Hinton 等進一步把 CNN 結合在深度學習之中[19]，在圖像辨識上帶來跳躍式的進步，從此圖像辨識便開始大量使用 CNN 的技術。接著，近年來商業公司如微軟等也實證出 CNN 在語音辨識上能夠得出更優的結果。此模型也可以使用反向傳播演算法進行訓練，相較其他神經網路，卷積神經網路(CNN)需要估計的參數更少，使之成為近年來被公認為最具潛力深度學習結構。



圖二、卷積神經網路概念示意圖

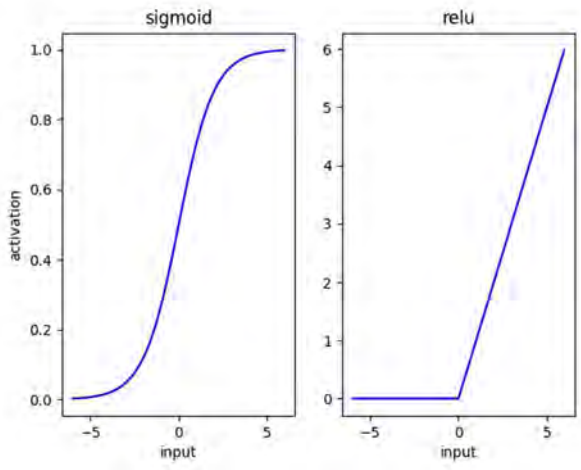
圖二假設使用 3*3 的小圖作為資料輸入，而且假設這隱藏層有 3 個單元(卷積核過濾器 Convolution Kernel Filter，由對應的權值向量 W 和 偏差 b 來呈現)。每個隱藏單元的輸入是用自己的權值向量 W 與 2*2 的小圖做內積，最後再與偏差項 b 相加。當要從中提取特徵，必須將 3*3 的大圖分解成若干 2*2 的小圖並分別提取它們的特徵。分解方法就是：從大圖的左上角起點始分別作小圖，依次(1, 1)、(1, 2)、(2, 1)、(2, 2)等 4 個點分別提取這 4 張小圖的特徵。所以，每個

隱藏單元將有 4 個輸入，不同於之前的 1 個，然後再分別導入激發函數計算後輸出，這一層的卷積特徵提取的工作就完成了。原則上愈複雜愈非線性的問題需要更多(深度)的隱藏層，來提取特徵以建構出更準確的模型。

基本上一個神經元的運算可以看成有兩個部份，即線性計算與非線性計算。圖二所說明的，便是其線性計算部份，而激發函數則是其非線性計算部份。如果缺少非線性計算，神經元只能用線性組合來提供預測能力，就沒有辦法處理複雜的問題。最早廣為使用的激發函數是邏輯函數(sigmoid function)，直到 Nair 和 Hinton 在 2010 年提出修正線性單元(rectified linear unit, relu)[27]後，由於其更適合較深的類神經網路，很快地就取代邏輯函數成為最廣為使用的深度學習之激發函數，因此本研究擬採用 relu 作為神經元的激發函數。圖三可展示 sigmoid 和 relu 兩函數的差異。

$sigmoid(x) = \frac{1}{(1+e^{-x})}$, where x is the input variable.

$relu(x) = \max(0, x)$, where x is the input variable.

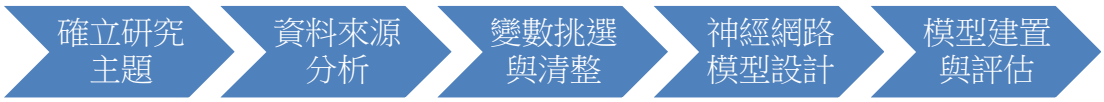


圖三、激發函數 sigmoid *vs* relu

最後，依據我們回顧文獻的結果，目前尚未有研究以 CNN 技術來建置球類競賽的預測模型。有鑑於此，本論文擬以卷積深度學習的神經網路來建置足球比賽結果預測的分類器，以期在不需要額外的內部資訊下，借助新的資料分析方法達到更好的預測成效。

貳、 研究方法與過程

本研究的流程如圖四所示，本章節將針對流程中前四部分的細節進行介紹，而最後「模型建置與評估」則在下一章中介紹。



圖四、研究流程

一、 確立研究主題

本研究的主題定義為利用公開資料進行足球競賽結果的預測，所採用的技術方法融合現今最熱門且具未來性的卷積深度學習的神經網路(CNN)，期待建置出更為準確的分類器預測模型。

二、 資料來源分析

由於本研究是採用 kaggle 上所提供的開放資料，所以建置模型時以公開資料集上的資料表格與欄位做為候選變數的挑選拄據，有關 kaggle 和本研究所使用的公開資料說明於後。

（一） kaggle：數據建模和數據分析競賽平台

kaggle 於 2010 年創立於澳洲，之後總公司移往美國舊金山，2017 年 google 買下部分股權使之成為公司人工智慧未來發展的利器之一。kaggle 是目前全世界最具規模的資料分析競賽平台，來自各地的數據分析好手或剛入門的新手都可以在這個平台上免費的進行學習或分享，是資料分析師培養實力與展示功力的社群，估計 kaggle 已擁有近八十萬用戶(李飛飛，2017) [1]。kaggle 專注於舉辦資料科學相關的線上競賽，各組織可在平台上不定期將實務難題拋出，藉由提供競賽獎金以吸引各方好手參與，獲勝者除可得到優厚報酬外，更可藉此建立自己的

專業口碑。

目前 kaggle 平台除了提供競賽之外，還有公開資料集(datasets)和運算模型(kernel)的分享，前者提供公開的資料讓各界好手鑽研他們有興趣的研究或實作主題，後者則是讓資料科學家分享他們針對特定資料集的模型建置成果與程式碼，以提供有興趣的人進行後續的研究。此次本研究所採用的資料即是從網站上的資料集：European Soccer Database 下載後進行資料的分析與建模。

（二） kaggle 資料集介紹：European Soccer Database[23]

由於對足球競賽做資料分析的研究者為數眾多，因此 FIFA 提供應用程式介面(Application Programming Interface)供有興趣的人爬取所需資訊，而陳列在 kaggle 的資料集則是由來自英國的 Hugo Mathien 透過 API 到 FIFA 等三個網站爬取資料後，再進一步清整最後再上傳到 kaggle 平台上。這個資料集主要含有超過 25,000 場的比賽資料、超過 10,000 筆球員資料、來自 11 個歐洲國家的頂尖球隊聯賽，從 2008 年到 2016/10/16 總計約八年的比賽資料。詳見網頁介紹。我們依據研究的主題主要是取得比賽、球員和球隊：Match、Player、Player Attribute、Team 和 Team Attributes 等五個資料表。

三、 變數挑選與清整

如同前述，本研究所採用的資料是來自 kaggle 的公開資料集，為了實證公開資料可以準確的預測出足球競賽的結果，我們在變數的挑選上擬從可獲取的資料集中的欄位進行，而非特定的收集或取得其它的資訊。這一小節中我們會介紹本研究所挑選的應變數，雖然挑選變數的過程或多或少是研究者基於特定目的的選擇(ad-hoc choice)，但鑑於卷積深度學習的神經網路(CNN)可以海納高維度的變數，原則上我們都儘量保留各資料表上的欄位變數。而本研究在挑選變數時最主要依循下列原則─基於所建置的模型是競賽前的預測，當該欄位變數是屬於比賽前不能預知的落後指標，就會被排除在建模變數的清單之外。各資料表的筆數、

變數個數與最後被採計的欄位數如表格一所示。另由於 CNN 擅長處理數值型資料，故當欄位有“ *k*”個類別型值時，我們就會把它編碼成一位有效編碼(one-hot encoding)，換句話說就是“ *k*”個布林欄位。表格一最右一欄則是用來表示編碼後的變數總數，這部分會在第四小節中再做說明。

表格一、資料表清單

資料表名稱	總筆數	欄位的總數	建模挑定的欄位總數	CNN 所用的變項數
Match	25,979	115	66	9*2=18
Player	11,060	7	0	0
Player Attributes	183,978	42	38	45
Team	299	5	0	0
Team Attributes	1,458	25	12	58

由於這五個資料表中 Player 和 Team 兩個資料表僅儲存鍵值和最基本的個資資訊，所以本研究並未直接使用這兩個資料表，取而代之的是分別以“Player Attributes”和“Team Attributes”兩個資料表的欄位變數為主，各資料表欄位挑選與遺缺值(missing value)的清整填補方式說明如下。

（一） 比賽(Match)資料表

1. 欄位挑選

(1) 主鍵(Primary Key)和外來鍵(Foreign Key)的欄位 (建模時不採計)

id、country_id、league_id、season、stage、date、match_api_id、home_team_api_id、away_team_api_id

(2) 得分欄位 (賽後資訊；故本研究於建模時不採用)

主隊得分 home_team_goal、客隊得分 away_team_goal

(3) 陣型欄位 (11*2*2=44 個)

home_player_X1~X11、away_player_X1~X11、home_player_Y1~Y11、away_player_Y1~Y11

(4) 先發隊員欄位 (11*2=22 個)

home_player_1~11、away_player_1~11

(5) 射門欄位 (賽後資訊；故本研究於建模時不採用)

射門次數 goal、射準次數 shoton、沒射準次數 shotoff

(6) 其它 (賽後資訊；故本研究於建模時不採用)

罰球數 foulcommit、黃紅牌數 card、鏟球數 cross、角球數 corner、控球時間 possession

(7) 賭盤相關欄位 (本研究不採計)

B365H、B365D、B365A、BWH、BWD、BWA、IWH、IWD、IWA、LBH、LBD、LBA、PSH、PSD、PSA、WHH、WHD、WHA、SJH、SJD、SJA、VCH、VCD、VCA、GBH、GBD、GBA、BSH、BSD、BSA

2. 遺缺值的清整填補

本資料表的資料尚稱完整，所挑選的球員陣型和球員雖有遺缺值，但究其原因實屬正常現象，換言之在某些比賽之中，確實會有先發球員少於 11 人的情況發生，故所挑選的變數中若出現遺缺值不特別做調整。

（二） 球員(Player)基本資料表

1. 欄位挑選

(1) 主鍵(Primary Key)和外來鍵(Foreign Key)的欄位(建模時不採計)

id、player_api_id、player_name、player_fifa_api_id

(2) 球員最基本的個資欄位(本研究建模時不採計)

birthday、height、weight

2. 遺缺值的清整填補

本資料表沒有直接參與建模故不需考慮此步驟。

（三） 球員屬性(Player Attributes)資料表

1. 欄位挑選

(1) 主鍵(Primary Key)和外來鍵(Foreign Key)的欄位(建模時不採計)

id、player_fifa_api_id、player_api_id、date(球員能力會隨時間調動),

(2) 球員能力相關的欄位 (38 個)

綜合表現 overall_rating、潛力 potential、慣用腳 preferred_foot、攻擊效率 attacking_work_rate、防守效率 defensive_work_rate、橫傳 crossing、完成攻擊任務 finishing、頭槌準確率 heading_accuracy、短傳 short_passing、抽射 volleys、運球 dribbling、曲球 curve、自由球準確 free_kick_accuracy、長傳 long_passing、控球 ball_control、加速度 acceleration、衝刺速度 sprint_speed、敏捷度 agility、反應力 reactions、平衡力 balance、射門力道 shot_power、彈跳力 jumping、持久力 stamina、強度 strength、遠射 long_shots、活躍度 aggression、攔截力 interceptions、卡位 positioning、臆測球向 vision、罰球 penalties、盯人能力 marking、站立攔截 standing_tackle、鏟球 sliding_tackle、守門員飛撲擋球 gk_diving、守門員接球 gk_handling、守門員踢球 gk_kicking、守門員卡位 gk_positioning、守門員反應力 gk_reflexes

2. 遺缺值的清整填補

本資料表中若欄位中有遺缺值時，我們採用全部球員在該欄位中的能力值進行平均，以此平均值取代原先的遺缺值。

（四） 球隊(Player)基本資料表

1. 欄位挑選

(1) 主鍵(Primary Key)和外來鍵(Foreign Key)的欄位(建模時不採計)

id、team_api_id、team_fifa_api_id

(2) 球隊名稱欄位(本研究建模時不採計)

team_long_name、team_short_name

2. 遺缺值的清整填補

本資料表沒有直接參與建模故不需考慮此步驟。

（五） 球隊屬性(Team Attributes)資料表

1. 欄位挑選

(1) 主鍵(Primary Key)和外來鍵(Foreign Key)的欄位(建模時不採計)

id、team_fifa_api_id、team_api_id、date

(2) 球隊各種能力相關的欄位 (12 個)

發動攻擊的速度 buildUpPlaySpeed、~~buildUpPlaySpeedClass~~

發動攻擊的運球 buildUpPlayDribbling、~~buildUpPlayDribblingClass~~

發動攻擊的傳球 buildUpPlayPassing、~~buildUpPlayPassingClass~~,

發動攻擊的卡位 buildUpPlayPositioningClass

有進球機會的傳球 chanceCreationPassing、~~chanceCreationPassingClass~~

有進球機會的橫傳 chanceCreationCrossing、~~chanceCreationCrossingClass~~

有進球機會的射門 chanceCreationShooting、~~chanceCreationShootingClass~~

有進球機會的卡位 chanceCreationPositioningClass

防守給的壓迫感 defencePressure、~~defencePressureClass~~

防守時的活躍度 defenceAggression、~~defenceAggressionClass~~

防守時的戰線寬度 defenceTeamWidth、~~defenceTeamWidthClass~~

防守時的防線 defenceDefenderLineClass

從以上的欄位名稱看起來我們不難發現有部分欄位是成對冗餘存在的，舉例來說 buildUpPlaySpeed 和 buildUpPlaySpeedClass 這兩個欄位，buildUpPlaySpeed 是以原始數值資料存在的，但所應的 buildUpPlaySpeedClass 則是原數值資料離散化(discretize)的結果，由於類神經網路本來就擅長處理數值型資料，所以我們直接採用原始的資料欄位捨棄離散化後的欄位(如刪除線所示)。

2. 遺缺值的清整填補

在這個資料表中各能力變數如果有遺缺值時，基本上都是採用平均數來進行資料的填補；一般來說所有成對的冗餘欄位遺缺值都是同時發生的，但 buildUpPlayDribbling 和 buildUpPlayDribblingClass 這兩個欄位情況較為特殊，我們觀察到不少筆資料在 buildUpPlayDribbling 欄位中有遺缺值的現象，但卻在 buildUpPlayDribblingClass 中有類別值，因此我們便將與該筆資料相同歸類的資料，將其所對應的原始數值欄位計算其平均數，最後再用此平均數填補回 buildUpPlayDribbling 的欄位內。

最後，由於球隊近期的表現與當下的賽事預測絕對是息息相關的[14]，因此我們進一步使用了比賽(Match)資料表中的訊息，計算其輸贏結果當作模型建置中的應變數，因此我們定義了 6 個類 18 個欄位來當作所欲預測的競賽中雙方隊伍的近期表現，計算方式說明如表格二所示。

表格二、主客兩隊的近期表現變數

變數名稱	計算方式
主隊近期表現	主隊最近的前 30 場球賽的勝率、和率、敗率
客隊近期表現	客隊最近的前 30 場球賽的勝率、和率、敗率

主隊近期同為主隊的表現	主隊最近同為主隊的前 20 場球賽的勝率、和率、敗率
客隊近期同為客隊的表現	客隊最近同為客隊的前 20 場球賽的勝率、和率、敗率
主客兩隊近期對賽的表現	主客兩隊近期對賽的前 10 場球賽的勝率、和率、敗率
主客兩隊近期對賽且同為主客場角色的表現	主客兩隊近期對賽且同為主客場角色的 5 場球賽的勝率、和率、敗率

而當我們在計算這些近期表現的數值時，有可能會遇到兩隊的近期比賽次數不足的情況，為了填補這類型的遺缺值，我們考量使用最早期的數據來填補更早期的資料是比較合宜的方式，所以我們會以所能取得最早期的數據來將所有往前不存在的遺缺值填滿。舉例來說，以主客兩隊近期對賽的表現來看，如果之前對賽資料不滿 10 場，只有 8 場那填補後的資料如表格三所示。

表格三、近期表現變數的填補

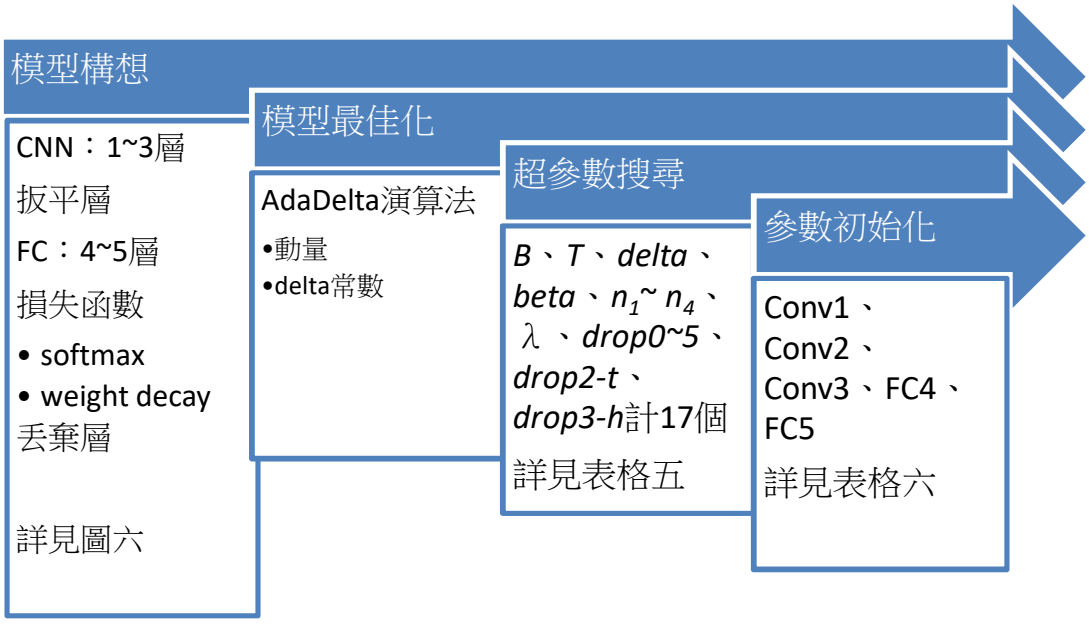
近期 1	近期 2	近期 3	近期 4	近期 5	近期 6	近期 7	近期 8	近期 9	近期 10
主勝	主勝	客勝	主勝	客勝	主勝	主勝	客勝	客勝	客勝

這種填補通常是發生在較新的球隊，而新球隊的勝負預測本來就比較困難，所以預測效果可能多少受到影響。表格三中近期 1 代表是最近一次的比賽，近期 8 則代表是最早期的對賽資料，近期 9 和 10 原本是遺缺值，但經過遺缺值填補之後變成“客勝”，雖然不難看出填補前的勝率是 5/8，填補後的勝率是 5/10，但由於近期 8 是最能代表近期 9 和 10 當時的兩隊實力之差距，故我們仍決定捨棄

平均值的填補方法而採取最接近時間點的填補方法。

四、 神經網路模型設計

本研究使用 kaggle 平台所提供之歐洲 2008-2016 職業足球比賽之公開資料集：European Soccer Database，運用深度學習於此資料集來建構類神經網路模型，我們將其命名成 SoccerNet，本節主要在描述本研究如何設計和架構深度學習的神經網路，擬先依序說明 SoccerNet 的模型構想、模型最佳化、超參數的搜尋以及參數初始化。建置流程如圖五所示。



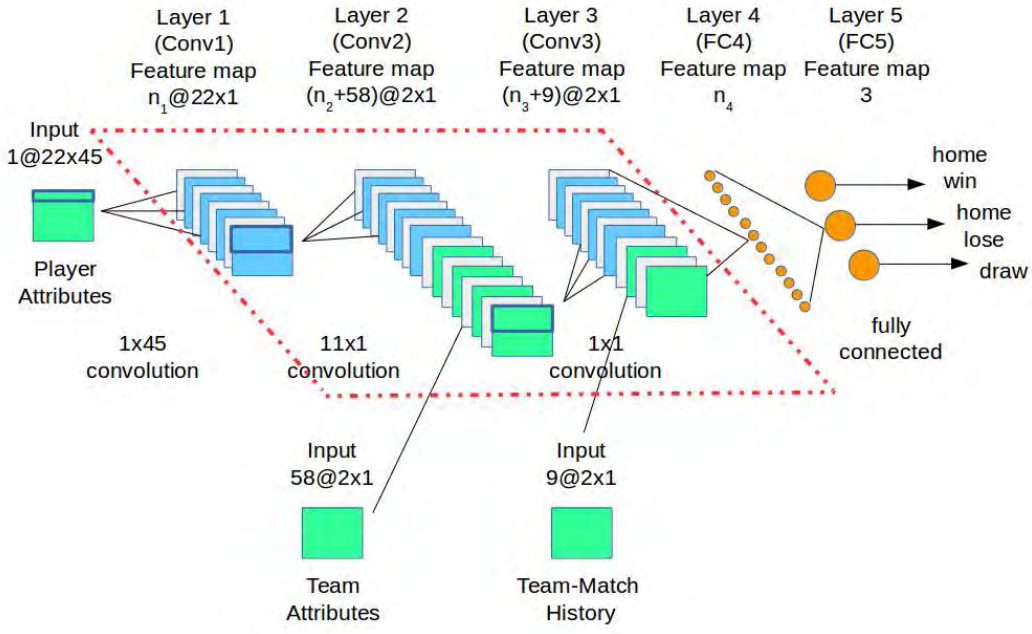
圖五、SoccerNet 神經網路模型建置流程

（一） 模型構想

基本上每一場歐洲職業足球比賽都是由主隊(Home Team)及客隊(Away Team)進行對決，除非某競賽隊伍有特殊情況，否則每隊在球場上都有 11 位先發球員，雖說這些球員在場上可能有被替換或被舉牌出局造成球員不足等狀況，但由於這些訊息在賽前不可得知且在公開資料集中也沒有出現，所以我們在建置

CNN 模型時是以先發球員為主。一旦球隊因特殊狀況導致先發球員不足 11 位時，我們會將遺缺值用“0”來代入模型。

再者，一般來說，職業球員在其職業生涯中，常會有轉換球隊的狀況，一旦有所謂的“明星”球員轉隊時，往往對球隊日後的比賽成績產生重大的影響。但從另一方面來看，當有新的明星球員加入某個球隊時，卻也未必都能如預期地提升球隊的成績。由此可見，比賽的勝負最基本就是由球員及球隊的綜合特性所共同影響。除此之外，球隊近期的成績，亦是被認定為會影響到比賽成績的因素，例如連勝或連敗，或者某個實力較好的球隊總是輸給某個實力較差的球隊，這種情形亦被球迷戲稱為“魔咒”。



圖六、SoccerNet 模型構想

因此本研究在建置模型時，由於強調使用公開資料集進行深度學習，所以最主要是以球員屬性、球隊屬性以及最近比賽成績來建構，詳細的變數如前小節所介紹。而文獻[33]中所歸整提及的變項，如果不是公開資料集可以取得，就不被放入本研究的變項之中，我們會在下一章節裡以實驗結果實證出僅使用這些公開資料集也可以得出準確的預測效果。

而由於本研究所採用的公開資料－球員屬性、球隊屬性以及最近比賽成績，這三種資料各有不同的特性，所以難以被設定成單一神經元來塑模。為了解決這種問題，本研究提出 SoccerNet 的類神經模型建構的設計，除了輸入層與輸出層外，中間層包括二大部分：前三層屬第一部分，使用 CNN 來建構，主要將這三種資料的饋入節點分開，於類神經網路的不同層饋入，再連結成相容的抽象特徵，作為下一層的輸入，如圖六中虛線四邊框所示。第二部分則是採用全連接型(fully connected)的二層神經網路接續 CNN 做學習。由於我們要設定的 SoccerNet 架構是用來預測一場比賽的輸贏，所以我們在圖六及以上的說明是先以一筆比賽(match)資料做說明。

1. 第一層 (Layer 1 Feature Map, Conv1)

第一層的 CNN 神經元主要是用來萃取球員屬性的特徵(feature)，由於每一場比賽的主隊及客隊的球員共計 22 人，而我們所取得的球員資料表中每個球員有 45 個屬性，這樣的資料特性可以被類比是一個 22*45 像素的單色影像資料，因此很適合使用 CNN 神經元來萃取球員的通用特徵，在這裡我們先假定會有 n_1 個神經元(特徵)。為了要應用 CNN 的概念，由於我們相鄰的球員之間，並沒有一般影像資料的空間關係存在(影像像素間存有相鄰空間關係)，因此我們不能使用影像辨識時常用的二維(2D)卷積核過濾器，故改用一維(1D)卷積核過濾器(kernel filter)，是故這一層的卷積核過濾器設為 1*45(="1"維*"45"個屬性)，並且每次移動時只向下一列移動 1 步(stride)，由於有 22 個球員資料是故會往下移動 22 列。這樣一來，對於這一層的每一個神經元來說每一個球員會產生 1 個值，我們以第一位球員資料為例，計算公式如下，這個值會被當成這個神經元的輸出。

$$\sum_{j=1}^{45} a_{1j} * w_j + b$$

所以當我們在第一層設定 n_1 個神經元來說(n_1 的決定容後說明)，會產

生出 n_1*(22*1)的輸出，這樣的輸出可以被視為有 n_1 個通道(channel)的 22*1 像素之影像資料，或者說是有 n_1 種顏色的 22*1 像素影像資料，我們統一使用 n_1@22*1(通道@列*行)來表示。第一層神經元對應的參數 w_j 的總個數，係由此層卷積核過濾器的維度大小(“1”D)及數量(45)決定，亦即為 n_1*1*45 或 45*n_1 個參數，偏差(bias)參數 b 的數量則是跟著神經元的個數一樣，僅有 n_1 個，故可以忽略不計。

2. 第二層 (Layer 2 Feature Map, Conv2)

第二層的 CNN 神經元主要是設計用來讓球員的通用特徵和球隊屬性相容，同時也為球員屬性資料提供更高的非線性度，基本上 CNN 的層數愈多非線性就愈高，這樣也愈能模擬複雜的問題。而為了要讓第二層的輸出可以和球隊屬性相容，第二層的輸出必須設計分成主客兩隊，因此我們此時把球隊的 58 個屬性設計表示成 58 個不同的通道，亦即可以標記為 58@2*1(通道@列*行)像素影像資料。另一方面來說，由於我們要將第一層的球員屬性和本層的球隊屬性相容，對於 n_2 個的第二層神經元來說，必需要產生出 n_2@2*1；所以我們把此層的卷積核過濾器設計為 11*1，並且設定其每次移動時向下一列移動 11 步，這樣的輸出可以就符合 n_2@2*1 而進一步相容。是故我們將兩者相連之後，第三層的輸入會是(n_2+58)@2*1，或者說是有 n_2+58 種顏色的 2*1 像素影像資料。除了相容性的因素之外，由於 CNN 的共享參數(shared parameters)之特性，第二層的卷積核過濾器所萃取出來的球隊通用特徵，也會和主客兩隊的角色無關，亦即也是一般化後的特徵。第二層參數的個數，係由此層卷積核過濾器的大小、維度及第一層的通道數決定，亦即為 n_2*11*(1* n_1)或 11* n_1*n_2 個參數；同前一層的設計，偏差(bias)參數 b 的數量則是跟著神經元的個數一樣，僅有 n_2 個，故可以忽略不計。

3. 第三層 (Layer 3 Feature Map, Conv3)

第三層的 CNN 神經元主要是用來第一層的球員屬性與第二層的球隊

屬性的通用特徵和球隊最近比賽成績相容在一起，同時多加一層也可以為模型提供更高的非線性度讓預測更為準確。為了要讓第三層的輸出可以和球隊最近比賽成績相容，我們運用與第二層相同的原理來設計模型，是故我們把球隊最近比賽成績的 9 個屬性表示成 9 個不同的通道，即可以視為 9@2*1 像素影像資料，對於 n_3 個第三層神經元來說，要產生出 n_3@2*1，才可以和球隊最近比賽成績的資料相連在一起。此層的卷積核過濾器設計為 1*1，並且每次移動時向下一列移動 1 步，這樣的輸出就是 n_3@2*1，轉變成和球隊最近比賽成績也相容。兩者相連之後，第四層的輸入會是 (n_3+9)@ 2*1，或者說是有 n_3+9 種顏色的 2*1 像素影像資料。第三層參數的個數，係由此層卷積核過濾器的大小及數量決定，亦即為 n_3*1*1*(n_2+58) 或 n_2*n_3+58*n_3 個參數，偏差(bias)參數 b 的數量則是跟著神經元的個數一樣，僅有 n_3 個，故可以忽略不計。

4. 扳平層 (Flatten Layer)

自第三層之後，我們的 SoccerNet 模型要利用主隊和客隊的不同角色，萃取能用以預測比賽結果的特徵，對於此種目的而言，全連接型神經元比較適合。為了能讓 CNN 神經元銜接到全連接型神經元，模型在兩層之間需插人另一層沒有參數的扳平(flatten)層，第三層原來的輸出(n_3+9)@2*1 被扳平之後，變成 2*(n_3+9)個輸出。由於此層並沒有參數，所以此層不計入模型深度中。

5. 第四層 (Layer 4 Feature Map, FC4)

第四層的全連接型神經元用來萃取能用以預測比賽結果的特徵，對於第四層 n_4 個神經元來說，每一個神經元都連接到所有的輸入節點；換言之，第四層的每一個神經元都可以連接到第三層主客隊的特徵，並依此產生出可以判斷比賽結果的特徵。第四層參數的個數，係由此層的輸入節點數量及神經元個數決定，亦即為 n_4*2*(n_3+9)或 2* n_3*n_4+18*n_4 個參數，偏差參數 b 的數量較少，可以忽略不計。

6. 第五層 (Layer 5 Feature Map, FC5)

第五層的全連接型神經元用來產生和比賽結果正相關的邏輯迴歸(logit)，本模型要預測的類別共有主隊勝(Home Win)、主隊敗(Home Lose)以及平手(Draw)三種，因此第五層的神經元只有三個，分別產生對應類別的邏輯迴歸。第五層參數的個數，係由此層的輸入節點數量及神經元個數決定，亦即為 3*n_4 個參數，偏差參數 b 的數量較少，可以忽略不計。

表格四、SoccerNet 架構總覽

Layer Name	# of Neurons	Input Size	Output Size	Kernel Filter	Parameters 參數數量
Conv1	n_1	1@22*45	n_1@22*1	1*45	45*n_1
Conv2	n_2	n_1@22*1	n_2@2*1	11*1	11*n_1*n_2
Conv3	n_3	(n_2+58)@2*1	n_3@2*1	1*1	n_2*n_3+58*n_3
FC4	n_4	2*n_3+18	n_4	-	2*n_3*n_4+18*n_4
FC5	3	n_4	3	-	3*n4

7. 模型的損失函數

在機率理論中 Softmax 的產出可以用來代表類別的分布，所以我們採用被大量研究運用的 Softmax 交叉熵(Cross Entropy)來當做損失函數，以將邏輯迴歸值換算成 Softmax 機率值，三種類別的 Softmax 機率值之總合即為 1，其計算方程式為：

$\mathrm{q}(x_i) = e^{x_i} / \sum_{i=1}^{3} e^{x_i}$; where x_i is the logit，

$\mathrm{q}(x_i)$代表三種類別的估計值之機率分佈函數。

如果我們以$\mathrm{p}(x_i)$代表三種類別的真實狀況(ground truth)之機率分佈函數，交叉熵損失函數即是用以代表這兩種機率分佈函數的不一致性，愈不一致，則損失愈大。其計算方程式為

$$\mathrm{CrossEntrophy}(\mathrm{p},\mathrm{q}) = -\sum_{i=1}^{3}(p(x_i) * \log(q(x_i))$$

除此之外，權值衰減(weight decay)的損失函數亦需要被考量；權值衰減是一種正規化(regularization)的方法，使用權值衰減的目的是為了要減少過 度適應(overfitting)的情形發生，其計算方程式為

$$\text{WeightDecay(w)} = \lambda * 0.5 * \sum_{w \epsilon W} w^2;$$

λ是常數，W 是除了偏差參數外的所有參數的集合。換言之，從第一層到第五層，除了偏差參數之外的所有參數，都要納入以上的權值衰減。

8. 丟棄層(Dropout Layers)

除了使用權值衰減之外，為了要更進一步減少過度適應，從第一層到第五層，鄰兩層之間都再插入一層丟棄(dropout)層[32]。此外，三種資料的饋入節點在進入類神經網路之前也都插入一層丟棄層；最後，第五層和損失函數之間也插入一層丟棄層，每一層丟棄層都有各自獨立的丟棄比率(dropout ratio)，用來決定隨機丟棄的資料比率，後續會再介紹這個超參數如何決定。這些丟棄比率只有在訓練模型的時候有作用，測試模型的時候則無作用，不再丟棄資料。

（二） 模型最佳化

梯度下降法(Gradient Descent)是目前最廣為使用的類神經網路最佳化演算法(optimizer)之始祖，但它在使用上也存在一些困難的挑戰。

首先是學習率(learning rate)，使用梯度下降法時需要選擇學習率，選擇大一點的學習率可以學習比較快，但是到了最佳點附近時，可能會發散(diverge)而不會收斂(converge)到最佳點。相反地，選擇小一點的學習率比較不會發生發散，但是要花更長的時間來學習。如果一開始使用大一點的學習率，一段時間之後再換小一點的學習率，可以兼具兩者的優點，但是如何做學習率排程(learning rate schedule)則是要解決的挑戰。

其次是局部最佳點(local minimum)，局部最佳點附近的梯度都是上升的，單

純只看梯度很容易陷在局部最佳點無法跳脫所以往往找不到更好的解(solution)。針對這個困難，目前最常見的方法是除了梯度之外再加入動量(momentum)的考量。動量的意義是最近梯度的加權平均，當到達一個局部最佳點時，動量可以提供一個學習的方向向量，即使附近的梯度都是上升的，只要加上動量之後仍是下降的，最佳化演算法就不會陷在局部最佳點不動。

因此，有數種基於梯度下降法來變形而成的最佳化演算法，AdaDelta[35]便是其中的一種。AdaDelta 使用過去一段固定時間內梯度平方的移動平均值(moving average)，以及梯度平方的動量，來決定學習率的大小。在梯度大的地方，學習率就會大一點；反之，在梯度小的地方，學習率就會小一點。在局部最佳點的時候，梯度平方的動量可以幫助 AdaDelta 跳離，同時也避免了需要做學習率排程的挑戰，只有動量的權重大小需要決定。此外，AdaDelta 為了避免梯度平方的移動平均值為 0 時，可能出現數值不穩定的情況，因此加上一個很小的常數 delta，一般可以用 10^{-6}。

基於上述，本研究使用 AdaDelta 作為最佳化演算法，唯一調整的超參數(hyper parameter)是動量的權重，並且固定使用 10^{-6} 作為常數 delta。

（三） 模型超參數的搜尋方法

最佳化演算法可以用來尋找最佳的模型參數，然而深度學習的模型有許多無法使用最佳化演算法來尋找最佳解的參數，為有別於能使用最佳化演算法來尋找最佳解的參數，這樣的參數稱為超參數。

有些超參數可以根據過去的研究直接決定，例如本研究直接使用 10^{-6} 作為 AdaDelta 的常數 delta，即使如此，深度學習的模型仍有許多超參數待搜尋。超參數的搜尋方法[2]可分為自動選擇、半自動選擇以及手動選擇(manual selection)。一般被公開且可以自由取得的深度學習框架(framework)，例如：Tensorflow、Torch、Caffe 等[15]，並不支援自動選擇或半自動選擇，本研究使用的深度學習框架是 Caffe，亦必須採用手動選擇。

用手動選擇來搜尋超參數時，必須先將訓練模型用的資料集分割成訓練(training)資料集、驗證(validation)資料集以及測試(testing)資料集。當選擇某組超參數的值之後，使用訓練資料集以及最佳化演算法來尋找最佳的模型參數，然後再用驗證資料集和剛才得到的最佳模型參數，計算驗證正確率(validation accuracy)。接著選擇另一組超參數的值，再次使用訓練資料集以及最佳化演算法來尋找最佳的模型參數，並且用驗證資料集和新得到的最佳模型參數，再次計算驗證正確率，較高的驗證正確率即代表該組超參數較佳。這樣的優化超參數過程一直反覆進行，直到沒有更高的驗證正確率為止，此時模型所有的參數以及超參數就都被決定了。最後，用測試資料集以及所有被決定的參數和超參數，計算測試正確率(testing accuracy)，此測試正確率即代表模型的效能。

為了優化超參數，本研究將歐洲 2008-2016 職業足球比賽之公開資料隨機分割成訓練資料集佔 70%、驗證資料集佔 15%和測試資料集佔 15%。

本研究所提出之模型共有以下 17 個超參數 ：

1. 批次大小(mini-batch size)：B，批次大小主要的限制是電腦圖形加速器(GPU)的記憶體大小，批次大小愈大，運算效率愈高。
2. 訓練迭代次數(number of training iterations)：T，訓練迭代次數主要取決於最佳化演算法的收斂速度以及過度適應的現象是否趨向明顯，一旦過度適應的現象產生，可以提前結束(early stopping)，減少訓練迭代次數，避免結果變得更差。
3. AdaDelta 常數：*delta*。
4. 動量權重(momentum)：beta。
5. 第一層 CNN 神經元個數：n1。
6. 第二層 CNN 神經元個數：n2。
7. 第三層 CNN 神經元個數：n3。
8. 第四層全連接型神經元個數：n4。
9. 權值衰減權重(weight decay)：λ。

10. 球員資料和第一層之間的丟棄比率：drop0。

11. 第一層和第二層之間的丟棄比率：drop1。

12. 第二層和第三層之間的丟棄比率：drop2。

13. 球隊資料和第三層之間的丟棄比率：drop2-t。

14. 第三層和第四層之間的丟棄比率：drop3。

15. 球隊最近比賽成績資料和第四層之間的丟棄比率：drop3-h。

16. 第四層和第五層之間的丟棄比率：drop4。

17. 第五層和損失函數之間的丟棄比率：drop5。

以上 17 個超參數可能值所構成的空間相當大，即使每個超參數只嘗試兩種不同的值，仍有數萬種的組合。為了有效地縮小搜尋空間，本研究使用以下 4 個策略：

1. 先選擇可以固定使用的超參數: B=256、delta=1e-6[35]、beta=0.95[35]、λ=0.0005[15]。

2. 先不使用丟棄層，亦即讓所有的丟棄比率為 0%，選擇可以讓得到訓練正確率超過 90%的神經元個數 n1、n2、n3、n4 以及訓練迭代次數 T。增減神經元個數時，依模型各層的參數數量表(表格四)，挑選參數數量影響最大的那一層，並以加倍或減半的方式進行。如果訓練正確率低於 90%，則先增加訓練迭代次數 T，並以一次增加 5000 的方式進行。利用這個策略，可以快速地找出足以描述問題的適當模型大小及訓練迭代次數。本研究從 n1=64、n2=64、n3=64、n4=256、T=10000 開始，依以下的順序搜尋。

   2.1 n_1=64、n_2=64、n_3=64、n_4=256、T=10000：得到訓練正確率 91.11%，驗證正確率 50.96%。訓練正確率仍大於 90%，挑選第二層減半。

   2.2 n_1=64、n_2=32、n_3=64、n_4=256、T=10000：得到訓練正確率 98.21%，驗證正確率 49.48%。訓練正確率仍大於 90%，挑選第四層減半。

   2.3 n_1=64、n_2=32、n_3=64、n_4=128、T=10000：得到訓練正確率 98.21%，驗證正確率 49.29%。訓練正確率仍大於 90%，挑選第一層減半。

   2.4 n_1=32、n_2=32、n_3=64、n_4=128、T=10000：得到訓練正確率 92.87%，

驗證正確率 51.11%。訓練正確率仍大於 90%，挑選第四層減半。

2.5 n_1=32、n_2=32、n_3=64、n_4=64、T=10000：得到訓練正確率 95.64%，驗證正確率 49.80%。訓練正確率仍大於 90%，挑選第二層減半。

2.6 n_1=32、n_2=16、n_3=64、n_4=64、T=10000：得到訓練正確率 88.41%，驗證正確率 50.52%。訓練正確率低於 90%，增加訓練迭代次數 5000。

2.7 n_1=32、n_2=16、n_3=64、n_4=64、T=15000：得到訓練正確率 94.94%，驗證正確率 50.08%。訓練正確率仍大於 90%，挑選第三層減半。

2.8 n_1=32、n_2=16、n_3=32、n_4=64、T=15000：得到訓練正確率 82.07%，驗證正確率 50.99%。訓練正確率低於 90%，增加訓練迭代次數 5000.

2.9 n_1=32、n_2=16、n_3=32、n_4=64、T=20000：得到訓練正確率 85.21%，驗證正確率 49.01%。訓練正確率低於 90%，第三層加倍恢復大小，並恢復訓練迭代次數，改挑選參數數量略少的第四層減半。

2.10 n_1=32、n_2=16、n_3=64、n_4=32、T=15000：得到訓練正確率 86.19%，驗證正確率 50.55%。訓練正確率低於 90%，增加訓練迭代次數 5000.

2.11 n_1=32、n_2=16、n_3=64、n_4=32、T=20000：得到訓練正確率 86.19%，驗證正確率 50.55%。訓練正確率低於 90%，第四層加倍恢復大小，並恢復訓練迭代次數，改挑選參數數量略少的第一層減半。

2.12 n_1=16、n_2=16、n_3=64、n_4=64、T=15000：得到訓練正確率 83.20%，驗證正確率 46.12%。訓練正確率低於 90%，增加訓練迭代次數 5000。

2.13 n_1=16、n_2=16、n_3=64、n_4=64、T=20000：得到訓練正確率 88.92%，驗證正確率 50.74%。訓練正確率低於 90%，第一層加倍恢復大小，並恢復訓練迭代次數，改挑選參數數量最少的第二層減半。

2.14 n_1=32、n_2=8、n_3=64、n_4=64、T=15000：得到訓練正確率 68.73%，驗證正確率 49.23%。訓練正確率低於 90%，增加訓練迭代次數 5000。

2.15 n_1=32、n_2=8、n_3=64、n_4=64、T=20000：得到訓練正確率 81.73%，驗證正確率 53.00%。訓練正確率低於 90%，第二層加倍恢復大小，並恢復訓練迭代次數。

2.16 n_1=32、n_2=16、n_3=64、n_4=64、T=15000：得到訓練正確率 93.75%，驗證正確率 49.83%。訓練正確率仍大於 90%，因此可以決定 n_1=32、n_2=16、n_3=64、n_4=64。

3. 在前項搜尋過程中，可以看出驗證正確率和訓練正確率有顯著的差距，表示有過度適應的問題。依據過去的研究[32]，當輸入資料層的丟棄比率為 20%，類神經網路隱藏層(hidden layer)的丟棄比率為 50%時，常是接近最佳的選擇。使用此初始值之後，驗證正確率便具有參考性，可以用來搜尋其它的超參數，亦即訓練迭代次數 T 和動量權重 *beta*，挑選能得到較高驗證正確率的值。決定之後，再用最好的值來搜尋入資料層

的丟棄比率，最後再搜尋隱藏層的丟棄比率。

3.1 比較 *T*=30000 和 *T*=45000 的驗證正確率，前者較佳，以此可以決定 *T*=30000。

T	*beta*	訓練正確率	驗證正確率
30000	0.95	58.65%	**57.87%**
45000	0.95	58.59%	57.55%

3.2 比較 *beta*=0.95，*beta*=0.97，*beta*=0.97 的驗證正確率，*beta*=0.95 較佳，以此可以決定 *beta*=0.95。

T	*beta*	訓練正確率	驗證正確率
30000	0.95	58.65%	**57.87%**
30000	0.97	58.93%	57.11%
30000	0.99	58.34%	57.30%

3.3 比較 *drop0*，*drop2-t*，*drop3-h* 各自為 10%及 20%的驗證正確率，以此可以決定 *drop0*=10%，*drop2-t*=20%，*drop3-h*=10%。

drop0	*drop2-t*	*drop3-h*	訓練正確率	驗證正確率
20%	20%	20%	58.65%	57.87%
10%	10%	10%	60.09%	57.52%
10%	10%	20%	55.86%	54.69%
10%	20%	10%	58.68%	**58.05%**
20%	10%	10%	60.22%	57.87%
20%	20%	10%	59.15%	57.17%
20%	10%	20%	58.43%	57.17%
10%	20%	20%	55.70%	54.85%

3.4 比較 *drop1*=50%，*drop1*=25%的驗證正確率，以此可以決定 *drop1*=50%。

drop1	訓練正確率	驗證正確率
25%	58.62%	57.55%
50%	58.68%	**58.05%**

3.5 比較 *drop2*=50%，*drop2*=25%的驗證正確率，以此可以決定 *drop2*=50%。

drop2	訓練正確率	驗證正確率
25%	59.37%	57.43%
50%	58.68%	**58.05%**

3.6 比較 *drop3*=50%和 *drop3*=25% 的驗證正確率，以此可以決定 *drop3*=50%。

drop3	訓練正確率	驗證正確率
25%	60.31%	56.95%
50%	58.68%	**58.05%**

3.7 比較 *drop4*=50%和 *drop4*=25%的驗證正確率，以此可以決定 *drop4*=50%。

drop4	訓練正確率	驗證正確率
25%	60.60%	56.86%
50%	58.68%	**58.05%**

3.8 比較 drop5=50%和 drop5=25%的驗證正確率，以此可以決定 drop5=50%。

drop5	訓練正確率	驗證正確率
25%	60.89%	56.78%
50%	58.68%	**58.05%**

4. 依照之前的過程，所有的超參數都可以被決定了。但由於提高丟棄比率可能會使得更大的模型表現更好，因此最後再嘗試把 n_1，n_2，n_3，n_4 加倍， 並以更大的訓練迭代次數進行比較，以免更大的模型需要更多的訓練次數才能收斂。結果如下表所示，兩者皆未得到更好的結果，因此可以決定前述結果即為最終的模型超參數。

T	訓練正確率	驗證正確率
30000	59.69%	57.21%

45000	60.97%	57.05%

經過上述超參數的探索，我們將 SoccerNet 建模時所有的超參數設定值統整如表格五所示。

表格五、SoccerNet 超參數設定值

超參數名稱	設定值
批次大小(mini-batch size)：*B*	256
訓練迭代次數(number of training iterations)：*T*	30000
AdaDelta 常數：*delta*	1e-6
動量權重(momentum)：*beta*	0.95
第一層 CNN 神經元個數：n_1	32
第二層 CNN 神經元個數：n_2	16
第三層 CNN 神經元個數：n_3	64
第四層全連接型神經元個數：n_4	64
權值衰減權重(weight decay)：λ	0.0005
球員資料和第一層之間的丟棄比率：*drop0*	10%
第一層和第二層之間的丟棄比率：*drop1*	50%
第二層和第三層之間的丟棄比率：*drop2*	50%
球隊資料和第三層之間的丟棄比率：*drop2-t*	20%
第三層和第四層之間的丟棄比率：*drop3*	50%
球隊最近比賽成績資料和第四層之間的丟棄比率：*drop3-h*	10%
第四層和第五層之間的丟棄比率：*drop4*	50%
第五層和損失函數之間的丟棄比率：*drop5*	50%

除了模型超參數之外，參數初始化(weight initialization)也會對最佳化演算法有所影響[10]，所有的模型參數皆以高斯分佈初始化，依照文獻的推導平均數皆為 0，標準差則為$\sqrt{\frac{2}{nl}}$，nl 是接到該層神經元的輸入數量，計算結果如表格六所示。最後有關偏差(bias)參數 b 則不需要如此，都以 0 為初始值。

表格六、參數初始化

Layer Name	分佈函數	平均數	標準差
Conv1	高斯分佈	0	0.21
Conv2	高斯分佈	0	0.075
Conv3	高斯分佈	0	0.16
FC4	高斯分佈	0	0.12
FC5	高斯分佈	0	0.18

接下來的章節中，我們將以這些模型構想、模型最佳化、超參數和參數初始化的值進行本研究的模型建置。

參、 研究結果與討論

本章依據前一章的模型構想(圖六)模型超參數的設定(表格五)與參數初始化(表格六)，以 kaggle 平台所提供之歐洲 2008-2016 職業足球比賽之公開資料集(European Soccer Database)進行模型的建置，有關結果的評估與討論呈現於後。我們先介紹研究所使用的硬軟體，接著呈現我們的實驗結果，最後一節再跟其它相關研究進行比較與討論。

一、 實驗設備與器材

（一） 硬體

筆記型電腦乙台用來進行程式的撰寫與執行，配備如下：

CPU： Intel Core i7-6700HQ，2.60GHz

RAM： 8GByte

GPU：nVidia GeForce GTX 960M

OS: Ubuntu 16.04.2 LTS

（二） 資料來源

kaggle 平台：職業足球比賽之公開資料集(European Soccer Database)

資料時間：2008-2016 年

Match 資料表：25,979 筆、Player 資料表：11,060 筆、

Player Attributes 資料表：183,978 筆、Team 資料表：299 筆、

Team Attributes 資料表：1,458 筆

（三） 軟體

本研究主要使用 Python 2.7 來進行程式撰寫，分兩部分進行說明：第一部分先說明資料前處理的清整程式，第二部分則說明深度學習工具的設定與操作。

1. 資料前處理

依照前述章節所說明的資料清整方法，本研究使用 python2.7 實作了三個程式：sql.py、tensor.py 和 normalization.py，簡要說明如下。

sql.py

此程式的目的是在做資料的清整：篩選、編碼與填寫遺缺值，詳細做法請見第貳章的「三、變數挑選與清整」小節。

程式主要是使用 python 所支援的 SQL Lite 引擎，查詢原始資料中的各種表格(table)，類別型值也是由此程式編碼成一位有效編碼(one-hot coding)，同時也清整填補遺缺值，並且計算出球隊近期表現的指標數據。在程式的最後部分，從 SQL Lite 資料庫取出的資料被轉成 python 的 list，並且儲存到適合 python list 的 pickle 檔案。

tensor.py

這個程式銜接 sql.py 的結果，從 pickle 檔案中讀出 python list，根據本研究建構的類神經網路模型之構想(詳見第貳章的「四、神經網路模型設計」)，轉換成其所需要的維度之張量(tensor)。所謂的張量，其實就是高維度的陣列，同時也是 python 的 numpy 數值分析(numerical analysis)使用的資料型態。轉換成張量之後，這個程式將其儲存到適合 python numpy 的 npy 檔案。

normalization.py

這個程式銜接 tensor.py 的結果，從 npy 檔案中讀出張量。這些張量在餵入類神經網路模型工具 Caffe 之前必須做常模化(normalization)，這個程式便是執行這個最後的前處理工作。

常模化的過程是分別對訓練資料的每一個維度計算平均值和標準差，然後將所有的值減去平均值再除以標準差。換言之，就是把每一個維度的值轉換成統計學(statistics)上的標準常態分佈的 z 分數，並且將這些平均值和標準差儲存起來，因為以後要使用這個類神經網路

模型來做預測時，輸入模型的資料就要在各維度做相同的常模化，這時所用到的平均值和標準差就是這些從訓練資料計算得到的。

另外建模時的第一步驟需將全部的資料分割成訓練資料、驗證資料和測試資料，也是這個程式的工作，本研究所設定的比率為70/15/15。最後，這個程式也利用剛提及的訓練資料計算的平均值和標準差，先對驗證資料和測試資料做常模化。所有的資料常模化之後，再儲存到 python 和 Caffe 通用的 h5py 檔案，這樣資料前處理任務就告完成了。

2. 深度學習模型的建置

Caffe是一個開發深度學習模型的平台，研究人員透過描述模型，即可利用 Caffe 支援的各種神經單元、深度學習演算法等，來訓練其模型，不需要額外再實作程式。這個平台是 UC Berkeley 的 BAIR(Berkeley Artificial Intelligence Research)實驗室所開發及維護，自 2014 年以來已經累計有 5487 篇學術論文使用 Caffe 來訓練其模型。其深度演算法的正確性，已經被大量地驗證，同時更因為 Caffe 是開源軟體(open source)，其程式碼亦被這些使用它的研究人員不斷地除錯及改善，因此不論是在執行效率，或是正確性方面，都比研究人員自行實作要好。此外，Caffe 支援使用 GPU 做運算，可以大幅度地加速模型的訓練。

因此本研究採用 Caffe 1.0.0 的工具來進行模型的建置，我們需要搭配兩個輸入檔案給 Caffe：soccernet-train.protxt 和 soccernet-solver.protxt。soccernet-train.protxt 的設定內容是依據第貳章的「四、神經網路模型設計」的所有細節從各層神經元的構想、模型最佳化、超參數和參數起始化等等進行設定，總計有 25 個用來定義模型的語法區塊，統整如表格七。而有關模型最佳化 AdaDelta 的部

分相對簡單在此不贅述 (soccernet-solver.protxt)。詳細的設定語法請見附錄。

表格七、SoccerNet_Caffe 設定

用途	對應的模型定義之語法區塊名稱
訓練/驗證/測試	train、valid、l6-test_out
卷積神經網路層	l1-conv、l2-conv l3-conv
扳平層	l4-fl、
全連接網路層	l4-fc、l5-fc
激發函數	l1-relu、l2-relu、l3-relu、l4-relu
連接	l3-concat、l4-concat
丟棄比率	l0-drop 、 l1-drop 、 l2-drop 、 l2-drop-t 、 l3-drop、l3-drop-h、l4-drop、l5-drop
損失函數	l6-sm
正確率	l6-acc

二、 實驗結果

由於模型參數初始化是一個隨機過程，每次都隨機採用不同的亂數種子 (random seed) ，所以會使得每次訓練模型的結果略有差別，依照[2]的建議，本研究以訓練資料集訓練 10 次模型參數，並以其中測試資料集的正確率最高者為最終的模型，10 次的實驗結果討論於後。我們首先介紹一下常用來評估分類結果的混淆矩陣(Confusion Matrix)，接下來再說明常用的指標數據。

在監督式學習的分類模型中，混淆矩陣被用來檢視所建的分類模型的效果，如表格八所示。矩陣的每一欄代表一個目標類別(主隊勝/平手/主隊輸)的模型預測結果是對(正例)或錯(負例)之筆數，而每一列表示一個目標類別(主隊勝/平手/主隊輸)的真實狀況是對(正例)或錯(負例)之筆數。在機器學習領域，混淆矩陣也

常被稱為列聯表或誤差矩陣。

表格八、混淆矩陣

模型預 / 真實狀況	＋ 正例	－ 負例
＋	正確正例 (True Positive)	錯誤負例 (False Negative)
－	錯誤正例 (False Positive)	正確負例 (True Negative)

（一） 精確率(Precision)

$$\text{精確率(Precision)}=\frac{\text{正確正例(TP)}}{\text{正確正例(TP)+錯誤正例(FP)}}$$

用以衡量當分類器模型判定為正例時，真正是正例的比率有多少。

（二） 召回率(Recall)

$$\text{召回率(Recall)}=\frac{\text{正確正例(TP)}}{\text{正確正例(TP)+錯誤負例(FN)}}$$

用以衡量能被分類器模型正確判定為正例佔總真實正例的比重。

（三） F 度量(F-measure)

$$\text{F 度量(F-measure)}=2*\frac{\text{準確率(Precision)}*\text{召回率(Recall)}}{\text{準確率(Precision)+召回率(Recall)}}$$

用以綜合考量召回率和準確率的綜合指標。

（四） 正確率(Accuracy)

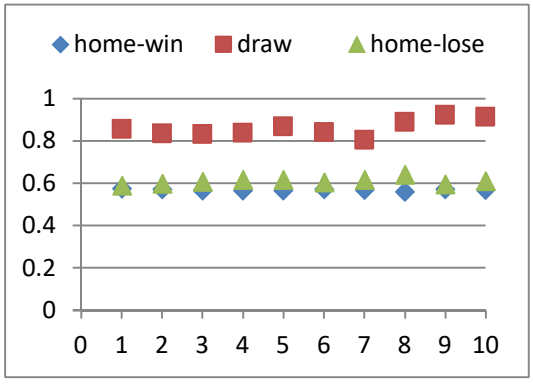
$$\text{正確率(Accuracy)}=\frac{\text{正確正例(TP)+正確負例(TN)}}{\text{正確正例(TP)+錯誤正例(FP)+錯誤負例(FN)+正確負例(TN)}}$$

反映了分類器對整體資料的判定能力，也就是可以將正例的判定為正，負例的判定為負的比例。

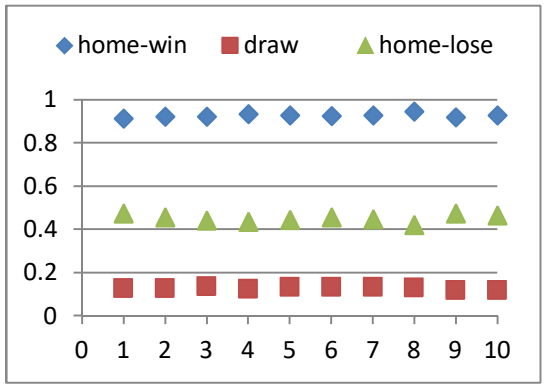
以上四個指標各有其應用的目的，比如說如果我們比較關心該如何在開賽前下注，那分類器所預測出來的類別(主隊勝/平手/主隊輸)的精確率(Precision)就很

重要。但在本研究中，重心是放在分類器對整體資料的判定能力，而且過往文獻也主要使用正確率(Accuracy)來代表模型好壞，所以我們在第貳章的超參數搜尋與本章的實驗成果呈現都以正確率作為挑選標準。讀者則可依應用目的之不同自行挑選合適的判定指標。

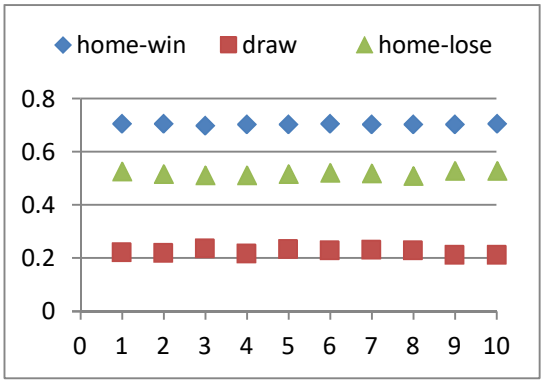
接下來我們展示採用第貳章的「四、神經網路模型設計」，包括類神經網路模型之構想、模型最佳化、超參數值和參數初始化進行十次不同亂數種子(random seed)的測試，我們利用這十次所產出的測試資料集之混淆矩陣，計算其精確率(圖七)、召回率(圖八)、F 度量(圖九)和正確率(圖十)，並討論如後：



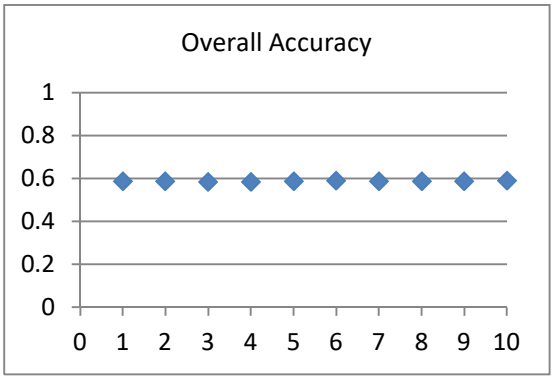
圖七、精確率比較



圖八、召回率比較



圖九、F 度量比較



圖十、正確率比較

從這十次實驗結果來看，首先我們可以發現不論是何種指標十次隨機種子的數值都變異不大。但讀者可以看出不同類別的預測表現相當不一樣，例如：預測「主隊勝」的召回率很高，然而預測「平手」的精確率很高。

一般來說，精確率是用來表現資訊系統預測出來的各類別是否精確，以購買

運動彩卷的目的而言，如果模型預測的結果是「平手」，因為它的精確率很高，選擇相信模型預測的勝算就很大。反之，如果模型預測的結果是「主隊勝」，因為它的精確率不夠高，則該選擇保守看待預測的結果—不要購買運動彩卷為宜。所以儘管預測「平手」的召回率很低，以此應用目的來說，就不構成問題。

另外從召回率來看，因為預測主隊勝的召回率很高，那表示當主隊會真的勝時，幾乎都會被系統預測出來，所以如果當系統預測主隊不會勝時，主隊的經營管理者或後援會可能就不需要準備慶祝勝利的活動，以這種應用目的來看，其他的指標就相對較不重要。

除此，如果兩個指標一起看時；以「主隊勝」精確率低但召回率高的狀況為例，它代表的意思是當主隊會真的勝時，幾乎都會被系統預測出來；但當系統預測是「主隊勝」則可能有一定比率是誤判(把真實的平手和主隊輸誤判成主隊勝)。這樣的資訊可以讓主隊的經營管理者或後援會在系統預測「主隊勝」時，保守準備慶祝勝利的各種活動或相關商品。

F 度量和正確率兩者都是用來觀看整體模型的指標，F 度量是利用定義的公式計算精確率和召回率的綜合影響，它的好處是每個預測類別都有分開的數據可以參考；而正確率則是不分類別只提供單一數據去表達整體模型可以正確預測所屬類別的比例。由於我們下一節要比較的文獻都是使用正確率做為比較基準，是故我們從這十次實驗中挑出測試資料集中正確率最高的模型做為本研究的實驗成果。最佳測試是發生在正確率為 59.03%，其所得到的混淆矩陣(confusion matrix)如下：

表格九、最佳模型之混淆矩陣

Predicted / Actual	Home-win	Draw	Home-lose
Home-win	1357	2	104
Draw	535	95	168

Home-lose	489	7	428

三、 評估與比較

依照表格九，我們製作出 SoccerNet 模型預測三種類別結果的指標數據：精確率(precision)、召回率(recall)、F 值(F-measure)和正確率(Accuracy)，如表格十所示。

表格十、SoccerNet 模型預測評測數據

類別 / 指標	Home-win	Draw	Home-lose
精確率(precision)	56.99%	91.35%	61.14%
召回率(recall)	92.76%	11.91%	46.32%
F 值(F-measure)	70.60%	21.06%	52.71%
正確率(Accuracy)	59.03%		

接下來我們進一步將實驗結果與之前的研究[33]相比，採用該文獻做為比較基準的理由有兩點：其一，該研究是近期且刊登在頗具公信力的學術期刊之中；其次該研究統整了相當多先前文獻所提出的各種機器學習方法與參數變數的處理方法，所以堪稱相當完整地涵蓋過往研究的實驗結果。

該文獻在實驗的第一部分先使用公開資料建立數種預測模型，包括測試了九種不同機器學習共 15 種參數的設定，搭配 8 種不同變數的選擇，建了 120 種不同的模型，所得出的正確率從 48.324% ~ 54.702%不等，其中有四個模型達到相同的最佳測試正確率 54.702%。該文獻第二部分另改採用賭盤資料再建立 15 種預測模型，其正確率為 47.459% ~ 55.297%不等；最後該文獻混合使用公開資料及賭盤資料，再次建立前述的 120 種預測模型，其正確率為 48.324% ~ 56.054%。由此可以觀察出 SoccerNet 的正確率達 59.03%，優於文獻[33]的結果。而由於本

研究所提出之模型僅使用公開資料中不含賭盤的資訊，所以接續段落擬以SoccerNet與[33]第一部分的4個最佳模型做比較(54.702%)：NB3、MPA3、MPA7、MPH3 (全名分別為 NaiveBayes w. 3PCs, Multilayer Perceptron all w. 3PCs, Multilayer Perceptron all w. 7PCs, and Multilayer Perceptron half w. 3PCs)。

表格十一、各指標值與文獻[33] 比較

Predicted / 指標	NB3 [33]	MPA3 [33]	MPA7 [33]	MPH3 [33]	SoccerNet [本研究]
正確率(accuracy)	54.702%	54.702%	54.702%	54.702%	**59.03%**
主隊勝					
精確率 (precision)	57.43%	**57.50%**	**57.50%**	57.33%	56.99%
召回率(recall)	83.78%	83.44%	83.44%	84.33%	**92.76%**
F 值(F-measure)	68.14%	68.09%	68.09%	68.26%	**70.60%**
平手					
精確率 (precision)	37.50%	NA	NA	NA	**91.35%**
召回率(recall)	0.57%	0%	0%	0%	**11.91%**
F 值(F-measure)	1.13%	NA	NA	NA	**21.06%**
主隊輸					
精確率 (precision)	24.57%	24.63%	24.63%	25.10%	**61.14%**
召回率(recall)	30.59%	31.53%	31.53%	31.06%	**46.32%**
F 值(F-measure)	27.25%	27.66%	27.66%	27. 76%	**52.71%**

***資料來源：本研究與文獻[33] 附錄 B

從表格十一可以看出，本研究 SoccerNet 所建置的模型大部分的指標數據都優於文獻[33]中四個正確率最高的預測模型，僅有在主隊勝的精確率小輸 0.5%。另外，值得注意的是關於「平手」的預測。文獻[33]所建立的所有預測模型顯然無法預測或幾乎無法預測和局，根據該文獻中最好的「平手」的預測數據，精確率和召回率分別為 37.50%以及 0.57%，該研究之作者分析後推測可能是公開資料中並沒有足以預測和局的特徵。但本研究所提出之模型，預測「平手」的精確率和召回率分別達到 91.35%以及 11.91%，建立模型所使用的亦是公開資料，可見公開資料中仍隱藏有關和局的相關資訊，惟本研究所提出之模型尚無法得到令人滿意的召回率，有關和局的預測，仍有待後續研究與努力。

準確的預測足球競賽的結果，向來是一個有趣但困難的問題，文獻中揭示出的成果或許未臻完美，但隨著數據資料分析工具的精進也期待能為足球預測建置出更新更好的模型。本節用以與文獻[33]實驗結果的比較基準，或許可能因所用的資料集有所不同比較起來不一定絕對公平，但後續我們的研究可以進一步運用 SoccerNet 的模型概念與方法套用到文獻[33]所使用的資料集上，相信屆時也可以得出令人滿意的結果。

肆、 結論與應用

一、 結論

運動競賽的預測長久以來因問題十分有趣與其背後所影響的巨大商機，吸引了無數的產學業研究人員的投入，而隨著各種資訊系統與應用的普及，各種數據可被輕易的收集或擷取，人們得以便利且快速的依據這些資料來進行更為準確的預測。

本研究嘗試以 kaggle 平台上所提供之歐洲 2008-2016 職業足球比賽之公開資料集(European Soccer Database)，搭配卷積深度神經網路(CNN)學習技術，建置出足球競賽結果的預測模型。實驗成果展示出本研究所建置的 SoccerNet 預測模型與過往的研究成果相比，有著更佳的預測能力。

研究的貢獻除了具體提供更準確的足球勝負預測模型，以供各方相關人員做出更高品質的決策外；SoccerNet 也實證了使用公開資料以及目前最具潛力的卷積深度神經網路學習技術，在運動競賽的勝負預測中的運用成效。

二、 未來展望與應用

本研究所建置的預測模型，儘管在實驗結果與之前文獻相比有不錯的成績，但未來仍有許多可努力與改善的空間：

（一） 本研究所提出之模型，相當適合用來預測由多位球員組成的兩球隊之競賽，值得後續針對不同應用進行模型的設計與研究。

（二） 球隊的競賽歷史資料亦頗值得後續研究，可以更深入考量時序關係與 CNN 技術結合，以學習到更關鍵的重要特徵，提供更好的預測能力。

（三） 球隊經營者通常擁有比開放資料更豐富的資訊，若能結合這些內部資訊應該能創造出更準確的預測模型。

伍、 參考文獻及附錄

一、 參考文獻

[1] 黃亦筠(2007)。Google 的 AI 祕密武器。天下雜誌，檢自: http://www.cw.com.tw/article/article.action?id=5081385 [Mar. 13, 2017]

[2] Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. arXiv: 1206.5533v2.

[3] Benjamin, B., Charles, R., & Pollard, R. (1971). Skill and chance in ball games. Journal of the Royal Statistical Sociaety, 623-629.

[4] Boldrin, B. (2017). Predicting the result of English Premier League soccer games with the use of poison models. Undergraduate Thesis, Stetson University.

[5]Dixon, M., & Stuart, G. (1997). Modelling association football scores and inefficiencies in the football betting market. Journal of the Royal Statistical Society, 265-280.

[6] Dyte, D. & Clarke, S. (2000). A ratings based Poisson model for World Cup soccer simulation. Journal of the Operational Research Society, 993-998.

[7] Goddard, J., & Asimakopoulos, I. (2004). Forecasting Football Results and the Efficiency of Fixed-odds Betting. Journal of Forecasting, 23, 51-66.

[8] Goddard, J. (2015). Regression models for forecasting goals and match results in association football. International Journal of Forecasting, 21(2), 331-340.

[9] Graham, I., & Stott, H. (2008). Predicting bookmaker odds and efficiency for UK football. Applied Economics, 40(1), 99-109.

[10] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. arXiv: 1502.01852.

[11] Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A Fast Learning Algorithm for Deep Belief Nets. Neural Computation, 18(7), 1527-1554.

[12] Hinton, G. E., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., …, & Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The shared views of four research groups. IEEE Signal Processing Magazine, 29(6), 82-97.

[13] Huang, K.-Y., & Chen, K.-J. (2011). Multilayer Perceptron for Prediction of 2006 World Cup Football Game. Advances in Artificial Neural Systems, 1-8.

[14] Hucaljuk, J., & Rakipovic, A. (2011). Predicting football scores using machine learning techniques. In MIPRO, 2011 Proceedings of the 34th International Convention, IEEE, 1623-1627.

[15] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., & Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding, arXiv: 1408.5093.

[16] Kampakis, S., & Adamides, A. (2014). Using Twitter to predict football outcomes. CoRR, abs/1411.1243.

[17] Keogh, F., & Rose, G. (2013). Football betting-the global gambling industry worth billions. Available From: http://www.bbc.com/sport/football/24354124 [May 15, 2017]

[18] Koning, R. H. (2000). Balance in competition in Dutch soccer. Journal of the Royal Statistical Society: Series D (The Statistician), 49(3), 419-431.

[19] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 1, 1097-1110.
[20] Kunz, M. (2007). Big Count. FIFA Magazine, 10-15.
[21] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. In Proceedings of the IEEE, 86(11), 2278-2324.
[22] Maher, M. J. (1982). Modelling association football scores. Statistica Neerlandica, 36(3), 109–118.
[23] Mathien, H., (2017). European Soccer Database. Kaggle Inc, Available From: https://www.kaggle.com/hugomathien/soccer [May 9, 2017].
[24] Mayer-Schönberger, V., & Cukier, K. (2014). BIG DATA. London, John Murray.
[25] Min, B., Kim, J., Choe, C., Eom, H., & McKay, R. I. (2008). A compound framework for sports results prediction: A football case Study. Knowledge-Based systems, 551-562.
[26] Moroney, M. J. (1956). Facts from figures. London, Penguin Books.
[27] Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In ICML, 807-814.
[28] Purucker, M. C. (1996). Neural network quarterbacking. IEEE Potentials, 15(3), 9-15.
[29] Ridder, G. (1194). Down to ten: estimating the effect of a red card in soccer. Journal of the American Statistical association, 1124-1127.
[30] Sainath, T. N., Mohamed, A. r., Kingsbury, B., & Ramabhadran, B. (2013). Deep convolutional neural networks for LVCSR. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, 8614-8618.
[31] Snyder, J. (2013). What actually wins soccer matches: Prediction of the 2011-2012 premier league for fun and profit. Undergraduate Theses, Princeton University.
[32] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15, 1929-1958.
[33] Tax, N., & Joustra, Y. P. (2015). Predicting the Dutch football competition using public data: A machine learning approach. Trans. Knowl. Data Eng., 10(10), 1-13.
[34] Wilson, R. (1995). Ranking college football teams: A neural network approach. Interfaces, 44-59.
[35] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. arXiv: 1212.5701.

二、 附錄

（一） 定義 SoccerNet 模型：soccernet-train.prototxt

```
name: "soccerPrediction"
layer {
  name: "train"
  type: "HDF5Data"
  top: "players"            # (256, 1, 22, 45)
  top: "teams"               # (256, 58, 2, 1)
  top: "teamsHis"           # (256, 9, 2, 1)
  top: "labels"
  include {
    phase: TRAIN
  }
  hdf5_data_param {
    source: "train_loc.txt"
    batch_size: 256
  }
}

layer {
  name: "valid"
  type: "HDF5Data"
  top: "players"            # (256, 1, 22, 45)
  top: "teams"               # (256, 58, 2, 1)
  top: "teamsHis"           # (256, 9, 2, 1)
  top: "labels"
  include {
    phase: TEST
  }
  hdf5_data_param {
    source: "test_loc.txt"
    batch_size: 3185
  }
}


layer {
  name: "l0-drop"
  type: "Dropout"
  bottom: "players"
  top: "drop0"
  dropout_param {
    dropout_ratio: 0.1
  }
}

layer {
  name: "l1-conv"
  type: "Convolution"
  bottom: "drop0"
  top: "conv1"               # (256, num_output, 22, 1)
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 1 decay_mult: 0 }
  convolution_param {
    num_output: 32         # learn 64, 32, or 16 neurons
    kernel_h: 1
    kernel_w: 45           # each filter is 1x45
```

```
    stride: 1
    weight_filler {
      type: "gaussian" # initialize w from a Gaussian
      std: 0.21        # distribution with stdev sqrt(2/n_fan_in)
      #n = 45 (kernel_h*kernel_w*channel_input, 1x45x1)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}

layer {
  name: "l1-relu"
  type: "ReLU"
  bottom: "conv1"
  top: "relu1"              # (256, 32, 22, 1)
}

layer {
  name: "l1-drop"
  type: "Dropout"
  bottom: "relu1"
  top: "drop1"
  dropout_param {
    dropout_ratio: 0.5
  }
}

layer {
  name: "l2-conv"
  type: "Convolution"
  bottom: "drop1"
  top: "conv2"              # (256, num_output, 2, 1)
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 1 decay_mult: 0 }
  convolution_param {
    num_output: 16      # learn 64, 32, or 16 neurons
    kernel_h: 11
    kernel_w: 1         # each filter is 11x1
    stride_h: 11        # step 11 from home team to away team
    stride_w: 1
    weight_filler {
      type: "gaussian" # initialize w from a Gaussian
      std: 0.075        # distribution with stdev sqrt(2/n_fan_in)
      #n = 352 (kernel_h*kernel_w*channel_input, 11x1x32)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}

layer {
  name: "l2-relu"
  type: "ReLU"
  bottom: "conv2"
  top: "relu2"              # (256, 16, 2, 1)
```

```
}

layer {
  name: "l2-drop"
  type: "Dropout"
  bottom: "relu2"
  top: "drop2"
  dropout_param {
    dropout_ratio: 0.5
  }
}

layer {
  name: "l2-drop-t"
  type: "Dropout"
  bottom: "teams"
  top: "drop2-t"
  dropout_param {
    dropout_ratio: 0.2
  }
}

layer {
  name: "l3-concat"
  type: "Concat"
  bottom: "drop2"          # (256, 16, 2, 1)
  bottom: "drop2-t"       # (256, 58, 2, 1)
  top: "cc3"                # (256, 74, 2, 1)
  concat_param {
    axis: 1
  }
}

layer {
  name: "l3-conv"
  type: "Convolution"
  bottom: "cc3"
  top: "conv3"              # (256, num_output, 2, 1)
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 1 decay_mult: 0 }
  convolution_param {
    num_output: 64        # learn 64, 128, 256, or 512 neurons
    kernel_size: 1        # each filter is 1x1
    stride: 1
    weight_filler {
      type: "gaussian" # initialize w from a Gaussian
      std: 0.16            # distribution with stdev sqrt(2/n_fan_in)
      #n = 74 (kernel_size*kernel_size*channel_input, 1x1x74)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}

layer {
  name: "l3-relu"
  type: "ReLU"
  bottom: "conv3"
```

```
    top: "relu3"                # (256, 64, 2, 1)
}

layer {
    name: "l3-drop"
    type: "Dropout"
    bottom: "relu3"
    top: "drop3"
    dropout_param {
        dropout_ratio: 0.5
    }
}

layer {
    name: "l3-drop-h"
    type: "Dropout"
    bottom: "teamsHis"
    top: "drop3-h"
    dropout_param {
        dropout_ratio: 0.1
    }
}

layer {
    name: "l4-concat"
    type: "Concat"
    bottom: "drop3"          # (256, 64, 2, 1)
    bottom: "drop3-h"        # (256, 9, 2, 1)
    top: "cc4"               # (256, 73, 2, 1)
    concat_param {
        axis: 1
    }
}

layer {
    name: "l4-fl"
    type: "Flatten"
    bottom: "cc4"
    top: "fl4"               # (256, 146)
    flatten_param {
        axis: 1
        end_axis: -1
    }
}

layer {
    name: "l4-fc"
    type: "InnerProduct"
    bottom: "fl4"
    top: "fc4"               # (256, num_output)
    param { lr_mult: 1 decay_mult: 1 }
    param { lr_mult: 1 decay_mult: 0 }
    inner_product_param {
        num_output: 64       # learn 256, 128, 64, 32, or 16 neurons
        weight_filler {
            type: "gaussian" # initialize w from a Gaussian
            std: 0.12        # distribution with stdev sqrt(2/n_fan_in)
            #n = 146 (input, 146)
        }
        bias_filler {
```

```
        type: "constant"
        value: 0
      }
    }
}

layer {
  name: "l4-relu"
  type: "ReLU"
  bottom: "fc4"
  top: "relu4"                # (256, 64)
}

layer {
  name: "l4-drop"
  type: "Dropout"
  bottom: "relu4"
  top: "drop4"
  dropout_param {
    dropout_ratio: 0.5
  }
}


layer {
  name: "l5-fc"
  type: "InnerProduct"
  bottom: "drop4"
  top: "fc5"                  # (256, num_output)
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 1 decay_mult: 0 }
  inner_product_param {
    num_output: 3           # learn 3 neurons for home_win, draw, and home_lose
    weight_filler {
      type: "gaussian" # initialize w from a Gaussian
      std: 0.18             # distribution with stdev sqrt(2/n_fan_in)
      #n = 64 (input, 64)
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}

layer {
  name: "l5-drop"
  type: "Dropout"
  bottom: "fc5"
  top: "drop5"
  dropout_param {
    dropout_ratio: 0.5
  }
}

layer {
  name: "l6-acc"
  type: "Accuracy"
  bottom: "drop5"
  bottom: "labels"
```

```
    top: "accuracy"
    include {
      phase: TEST
    }
}

layer {
    name: "l6-sm"
    type: "SoftmaxWithLoss"
    bottom: "drop5"
    bottom: "labels"
    top: "loss"
    include {
      phase: TRAIN
    }
}

layer {
    name: "l6-test_out"
    type: "HDF5Output"
    bottom: "fc5"
    bottom: "labels"
    hdf5_output_param {
      file_name: "l6-test.h5"
    }
    include {
      phase: TEST
    }
}
```

（二） 定義 SoccerNet 模型最佳化的參數：soccernet-solver.prototxt

```
net: "train.prototxt"
type: "AdaDelta"
base_lr: 1.0
lr_policy: "fixed"
momentum: 0.95
weight_decay: 0.0005
display: 100
max_iter: 30000
snapshot: 5000
snapshot_prefix: "soccer"
solver_mode: GPU
delta: 1e-6
```

【評語】190009

此作品建立一個足球比賽預測神經網路。實驗結果顯示比現有的預測系統有更高的精準度。此作品著重在模型建立過程中，如何有效地挑選模型參數，加速訓練。建議對此過程，能有更有系統地呈現。與相關研究的比較，要更深入。