國立臺灣大學理學院地理環境資源學系

碩士論文

Department of Geography
College of Science
National Taiwan University

Master Thesis

以深度學習方法預測大型活動對臺北捷運運量之影響

Forecasting Passenger Demand for Large Special Events in Taipei Transit System Using Deep Learning Approach

林子鈞

Tzu-Chun Lin

指導教授：孫志鴻 博士

Advisor: Chih-Hong Sun, Ph.D.

中華民國 109 年 7 月

July,2020

摘要

可靠且精準的捷運運量預測對於乘客、捷運公司是非常重要且必要的，特別是當大型活動在城市中舉辦，使參加者在短時間內從城市中往活動舉辦地集中，參加者大多使用大眾運輸作為主要的交通方式，且因為大型活動的特性以及場館區位的特性，參加者到場與離場往往與平日運量相比，在時間上較為集中以及人數也較多，對於乘客的搭乘體驗、列車調度或是場站管理都有可能產生負面的影響。近年資料科學逐漸應用於人類生活中各個領域中，其中在交通預測中因為智慧卡交易時產生大量且複雜的數據，這些資料中隱含了空間與時間的特性。因此本研究欲利用深度學習的方法，以捷運悠遊卡的進出站刷卡資料，透過深度學習的方法，捕捉並預測臺北捷運在不同的日型態以及大型活動時的運量特徵，希望能提供捷運公司管理單位作為場站管理以及行車調度之參考，減輕大型活動的人潮對於大眾運輸系統的衝擊。

關鍵字: 捷運運量預測、大數據、悠遊卡、特殊活動、深度學習、時空分析

Abstract

Reliable and precision ridership forecasting is very important and essential to the passenger and the subway operator. Especially, when large events held in the urban area, the participants concentrated to the event venue in a short time, and most of participants would choose public transit as the main way arriving the venue. Because of the characteristics of large special events and the location of the venues, participants are often more concentrated and more numbers than on weekdays. Therefore, for passengers' ride experience, train scheduling and station management will have a negative impact. In recent years, data science has accordingly applied in many aspects of our daily life. In transit demand forecasting domain, because of the smart cards generated large volume and complex data, furthermore spatial and temporal characteristics are implied in these data. Therefore, this thesis intends to use the method of deep learning to predict the passenger demand of whole Taipei Metro system, in order to capture the feature under different day types and large-scale activities in Taipei Metro system through deep learning method. The prediction model are designed to provide decision-making assistance for MRT company management department and as a reference for station management to mitigate the impact of large special events on the mass transit system.

Keywords: Transit Demand Forecasting, Big Data, EasyCard, Special Events, Deep Learning, Spatiotemporal Analysis

誌謝

感謝孫老師願意指導我的論文，兩年當中承蒙您的照顧以及對論文的悉心指導，讓我的論文逐漸步上軌道到最後開花結果，在往返中研院的路上也分享了許多人生經驗，子鈞衷心感謝老師兩年的照顧。感謝兩位口試委員蔡博文 教授與周學政 副教授，從預口試到口試都給予我的論文肯定與許多實用的建議。

感謝辛勤工作永遠支持我的父母，大學時想從家裡到外面的世界看看，研究所開始珍惜回家與家人相處的時光，也發現父母逐漸上了年紀煩惱開始變多，我只希望你們可以生活得更快樂然後身體健康，有空的時候多出去走走。也希望我的妹妹四年大學可以順順利利，找到自己的興趣。然後是伊棋，妳一路上給的鼓勵和陪伴，你的好脾氣總是能夠包容我的固執，感謝你從考研究所就一直支持我到現在，也在你的見證下完成了最後的口試。謝謝竣翔學長和玫宜學妹在論文實作上的幫忙，遇到你們是我的福氣，沒有你們這篇論文不會這麼順利的完成。感謝臺北飯友雨蒔和大學同學育寬、庭慧、雨青、彥琳，平淡的生活中有你們一起吃飯一起分享生活。感謝高中同學道睿、劉昀、鈺暉、南圻在閒暇之餘可以一起玩遊戲和聊天。

給自己的話，研究所是一個人的旅程與奮鬥，自己的研究從無到有，需要不斷的摸索，才能更接近真理，蚤堡在史詩中唱著:「想起一路走到這裡，你值得為自己驕傲，但未來的路怎麼走，我不免感到焦躁。」或許就是我現在的心情，在論文即將完成之際，想著從研究所之初從零開始學習 Python，到現在完成了自己的論文，又思考著對於未來工作該如何選擇，人生總是充滿疑惑，而對於答案的追求則是永無止境。很開心也很幸運能在台大度過兩年的時光，在學生生涯的最後一站，我滿載而歸。

林子鈞 謹誌

2020.07

目錄

圖目錄

表目錄

第一章 緒論

1.1 研究動機與背景

臺北都會區位於臺北盆地內，為臺灣政治經濟中心，工商活動頻繁，吸引人口往狹小的盆地內集中，因此人口密度為全台之冠，為了解決路面壅塞問題，因而衍生發展捷運的需求。臺北捷運系統的興建及營運，正是紓解臺北都會區長期以來交通問題的一帖良藥，藉此改善都市動線，活絡都市機能，並且促進都市與周邊衛星市鎮再發展(台北大眾捷運股份有限公司, 2010) 。臺北捷運自 1996 年開始營運以來，隨著各線通車與路線延長，涵蓋雙北與桃園的路網逐漸發展成熟，成為臺北與新北地區重要的大眾運輸系統。隨著捷運路網逐漸發展成熟，臺北捷運的運量逐年增加(圖 1.1)，在 2006 年突破每日百萬人次，2015 年隨著所有路線全面通車，次年日均運量突破兩百萬人次。臺北捷運主要使用電子收費系統進行收費，旅客可透過單程電子代幣或是智慧卡進行搭乘，每日平均有兩百萬人次搭乘，也同時意味一日內可以產生大量的電子交易資料，若能善加分析利用，可以更加瞭解乘客使用捷運的行為模式以及便於場站管理與列車調度。工商時報的報導中提到，根據益普索（Ipsos）昨（8 月 31 日）公布台灣電子票證使用行為調查，目前持有率最高的是悠遊卡，有 89％的台灣民眾持有，臺北地區更高達 95％，有 77％受訪者表示悠遊卡是日常生活最常使用的電子票證(邱莉玲, 2015) 。

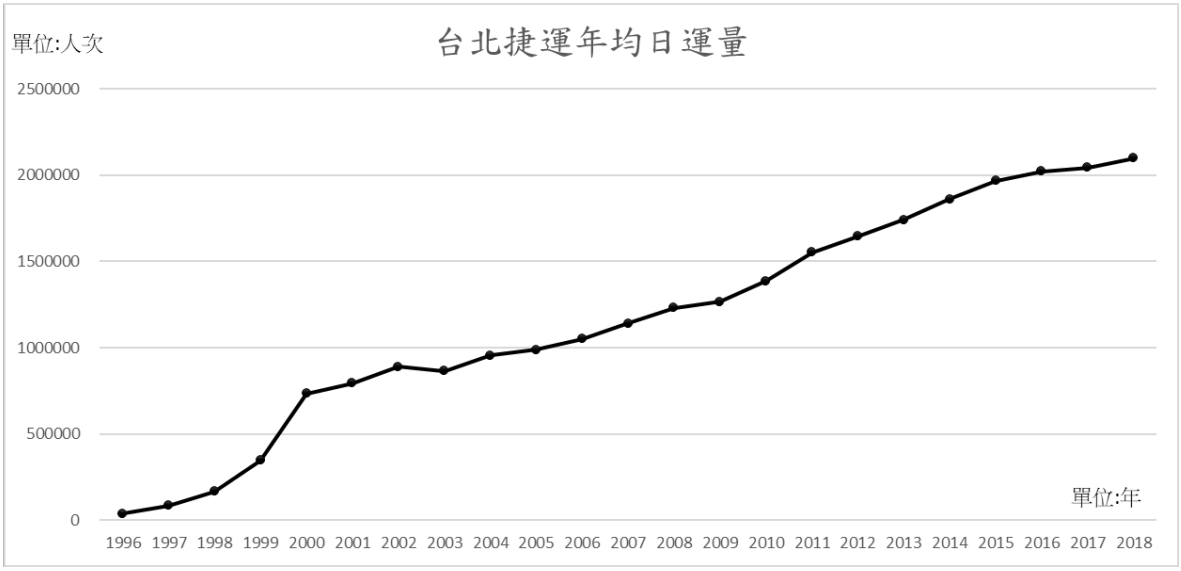


圖 1.1 臺北捷運逐年日均運量圖(作者自行整理)

臺北捷運公司出版的《軌道經營與管理-第 3 期》提到，捷運系統運量預測工作對捷運系統規劃單位而言，係為推動整個都會區捷運路網發展的重要依據，不論是路網規劃、路線可行性研究、運輸走廊研究、定線規劃乃至於財務分析等，運量預測皆存在著重要的關聯性及決定性的影響，其結果往往主導未來營運系統的選擇、車站規模的大小、列車採購規格與數量及列車服務計畫（班距、運行模式）等軟硬體設施規劃。對捷運系統營運機構而言，係為維護系統正常且有效率運轉的重要依據，不論是訂定營運計畫、提升營運效能、控制服務水準、票價訂定乃至於編定財務預算等，皆與運量預測結果緊密依存，故運量預測是決定捷運系統資源投入與效率評估的重要關鍵(臺北大眾捷運股份有限公司, 2008) 。

臺灣現今由於國民教育水準不斷提高、經濟的提升與社會的繁忙，造成個人工作與生活壓力增加，國人為脫離僅滿足基本生活需求的生活型態，而開始藉由參與各式活動來調劑身心與紓解壓力，因此參與提升生活品質、放鬆壓力之活動儼然成為國人生活中不可或缺的一部分。就現代的都市活動狀況而言，大型活動的舉辦已經是相當頻繁，本篇文獻也提及因為臺北捷運的旅行時間較短、旅行成本較低以及路線與站點普及臺北與新北市人口密集處，因此成為民

眾參與大型活動的主要使用運具(呂佩容, 2012) 。

Y. Liu et al. (2006)提到潛在的嚴重事件通常是由人群的集中與交通路網的阻塞所複合產生。而大型活動不管是開始前或結束後，大量的群眾向捷運站集中，人潮進入相對狹小的空間內，使車站以及車廂內產生擁擠的情況，除了可能造成推擠受傷等情況產生外，也可能造成旅客使用捷運的經驗不佳、等待時間太長，或是轉往使用其他大眾運輸工具，但往往捷運公司並不希望公開這些資訊，因為會造成收入減少，但對於正要使用捷運的乘客來說，若能預先知道目的地是否會有因為活動舉辦而產生的擁擠，就能轉往使用其他交通方式抵達。因此，了解乘客搭乘捷運的型態，以及能夠預測出因為大型活動所產生特殊的運量特徵就變得相當重要。

在機器學習領域中，短期的旅運需求預測是新興的研究主題，而短期旅運需求的預測會吸引較多研究興趣是因為大量當代的資料來源，如:智慧卡、交通感測器、探針、GPS 與監視器等，這些資料的結構富有細節與複雜度，其中包含了許多人類活動的行為特徵、時空關係等，對於分析人類活動可以用相較過去更微觀的方式來進行分析，但同時這也是非常具有挑戰性的研究課題，因為大數據資料的複雜度，讓過去傳統在趨勢分析以及時間序列分析使用的模型，難以預測資料中所包含的非線性特徵(Nguyen et al., 2018)。

類神經網路是一個非常受歡迎的電腦學習模型，已經被廣泛的應用在各種不同交通問題，也被大量的應用在運量預測的領域當中。此模型具有儲存經驗知識的特色，在交通研究中，類神經網路已經主要被使用作為資料處理方法，因為它能使用大量不同類型資料的能力、建立模型的彈性、學習與概括化能力、對資料的適應能力以及良好的預測能力(Karlaftis & Vlahogianni, 2011)。但類神經網路並非萬能的解決方案，Khashei and Bijari (2011)提到使用類神經網路模擬線性問題產生了混合的結果，因此盲目地將類神經網路應用於任何類型的數據是不明智的。類神經網路有龐大數量的輸入變數是非常具有威力的機器

學習系統，但過適(overfitting)對這樣的網路是非常嚴重的問題，在訓練資料的階段採樣了雜訊，導致對測試資料的預測不準確(Srivastava et al., 2014)。Kwang-Ho et al. (2000)提到類神經網路的缺點，它們對預測異常運量條件下的特別節日有限制，因為很難獲得足夠的數據集來學習那些特殊的負荷分佈。而且，這些方法不能處理人類預報員的定性知識和經驗，為特殊日子提供相對準確的預測結果。

López et al. (2018)提到可靠與精準的短期捷運乘客預測對於乘客、營運者以及公家機關是重要的。傳統的研究專注於平常運量的預測，以及在預測特殊事件情景下的客流量方面，存在無法預測資料中非線性特徵的缺點。這些大型事件可能會有破壞性的影響，對於大眾運輸系統應該有更多的關注在主動的管理以及即時傳播訊息。因此，對於大型活動運量的短期預測就顯得重要。臺北捷運目前列車班次的管理依靠預先規劃的列車運行計畫，並依靠人工的方式調整該計畫，此方式會有經驗傳承不易以及規劃結果難以驗證的問題(林誌銘 & 王晉元, 2008)。因此若能精準預測出捷運的運量，就能在特殊事件發生時更好的進行列車以及人員的管理。

捷運的運量同時存在線性與非線性的特性，因此過去捷運的運量分析使用一個對於演算法來說相對穩定以及變異不大的資料集來進行預測，會在分析之前先對資料進行雜訊的消除。蔡宗憲 et al. (2006)、黃志偉 (2015)等研究中，在預估短期運量時將國定假日與極端分布的運量數據視為離群值，因此進行實證分析時，在整理原始資料階段，則將離群值從資料中剔除，因為此段時間的資料變因多和變化大，難以找到其規律性。過去的運量研究大多透過不同機器學習的方法來預測捷運平常日的運量，而關注特殊事件的研究相對較少，但往往這樣的事件對於捷運管理人員，或是使用大眾運輸的乘客是較為不便的。因此，本研究欲使用臺北捷運的悠遊卡刷卡資料，透過深度學習的方法建立臺北捷運在大型活動時的運量模型，以期能增進捷運班次與場站的管理以及供乘客

進行決策的參考。

臺北捷運公司出版的《軌道經營與管理-第 14 期》針對臺北小巨蛋以及鄰近的南京東路站(現為南京復興)的加車機制進行探討，文中提到因應大型活動結束的人潮而從末端站派出加開車次常常緩不濟急，主要原因為不知何時需要加派列車以及需要加發幾輛列車，而文章中藉由開發軟體並分類加車條件，嘗試解決這個問題。臺北捷運公司票務部門所建立的旅客進站即時系統是以五分鐘累計為一小時之起訖(OD)運量，計算列車乘載率來計算加車數量。研究成果顯示降低列車乘載率 33.1%，無效加班車比例降低 87.4%。(臺北大眾捷運股份有限公司, 2014)。該篇研究透過加車機制軟體的開發，成功開發出合適的加車機制，但在目前捷運公司的資料當中並未提及或公開相關的資訊，且該研究也只僅限於文湖線，並未廣泛運用到整個捷運系統。

1.2 研究目的

基於前述動機，本研究為解決民眾使用大眾運輸對於捷運擁擠情況不明，降低捷運營運成本(無效派車、乘客抱怨)，以及提供時間解析度更高的運量預測，發展一個運量預測模型，透過提供預測未來的旅客量供管理單位參考，協助捷運公司進行管理與調派車策略上的調整。

本研究目的如下:

- 整理與處理悠遊卡運量資料，透過深度學習方法預測未來運量，並建立臺北捷運運量預測模型。
- 在預測中加入平日與特殊活動特徵，透過均方根誤差(Root Mean Square Error, RMSE)，了解模型對於平日與大型活動舉辦時的運量變化的預測能力。
- 提供預測結果，供捷運管理單位在列車班次調度、場站人員管理參考。

第二章 文獻回顧

本章首先針對大型活動相關文獻來進行討論，了解臺北市舉辦大型活動的相關交通管制規範，並確認出哪些活動以及場館可以作為觀察的選擇，並進行場站相關文獻的回顧，目的是選出要進行預測的場站相關條件為何。接著回顧短期運量的預測模型，了解過去在機器學習領域中如何處理這樣的問題，並進一步回顧能夠預測特殊事件下運量的相關模型，作為後續研究方法與模型建立的參考。

2.1 臺北捷運運量預測回顧

1. 臺北捷運現行運量預測系統

通過詢問臺北捷運公司關於現行運量預測系統，將得到的回覆與第四代臺北都會區運輸需求系統預測模型(以下簡稱 TRTS-4S)進行說明(周汶叡 et al., 2018)。臺北捷運目前營運所需的運量預測主要使用佛拉塔年成長率法，預估短年期之運量，其優點在於運算收斂速度最快，能最快獲得預測成果。此模式假設未來的旅次分布與旅次產生率和旅次吸引成長率成正比，與兩地間阻撓因素成反比。長期運量參考人口預測報告，以及臺北市捷運工程局針對各捷運路線於規劃階段之運量資料，預估捷運運量。在 TRTS-4S 的報告中也針對各路線捷運站間運量進行預測，分析年期包含基年民國 104 年，中間年民國 110 年，目標年民國 120 年、民國 130 年及民國 140 年。分析時段包含晨峰時段、昏峰時段、離峰時段、全日 24 小時、晨峰小時、昏峰小時等六個時段。從全日與各路線晨峰時段的站間量與觀察值進行比較，各站誤差皆在正負 10%以內。從捷運公司的說明可以了解，目前對於捷運運量的預測還是屬於較長時間(年)的預測，以及在路線規劃上的長期預測。而在 TRTS-4S 所做的驗證，雖然有相當好的預測結果，但預測的時間仍以時段作為預測的對象，與捷運公司的回覆相同，皆有無法預測更細緻時間的缺陷。從上述關於臺北捷運運量預測的文獻可以發現，臺北捷運缺少更高時間解析度的預測模型，也並未針對大型活動有個別的預測，因此本研究針對臺北捷運短期運量以及大型活動運量的預測，能對於北捷現有的系統可以提出貢獻。

2. 臺北都會區運輸需求預測模式(Taipei Rapid Transit System, TRTS)

臺北都會區運輸需求預測模式起源於民國 64 年交通部統籌辦理之「臺北都會區大眾捷運系統初步規劃」，並於民國 70 年建立臺北捷運運輸需求預測模式(TRTS-I 模式)，後以 TRTS-I 為基礎，針對臺北捷運特性重新檢討並引進新的預測技術，更新為 TRTS-II 模式。臺北市政府捷運工程局自民國 81 年起，重新針對臺北都會區社經、旅運特性，建立第三代臺北都會區運輸需求預測模型(TRTS-Ⅲ模型)，於民國 83 年 5 月完成。民國 86 年臺北市政府交通局以 TRTS-III 模型為基礎，發展完成 DOT-I 模型，後再進行家庭訪問調查及屏柵線交通量調查，於民國 90 年完成模型校估，完成 DOT-II 模型。於民國 98 年辦理第 1 階段「臺北都會區整體運輸需求預測模式建立-旅次行為調查及旅次發生模組」工作，於民國 99 年 10 月完成成果報告，第 2 階段「臺北都會區整體運輸需求預測模式建立與應用(TRTS-IV)」接續辦理旅次分佈、運具選擇、路網指派等模組建立、參數校估、驗證、預測等工作，並於民國 101 年 12 月完成 TRTS-IV 模型(圖 2.1)。TRTS-IV 模型核心為界內客運模型，此模型以程序性總體需求模式(Aggregate Sequential Demand Model)－旅次發生、旅次分佈、運具分配與路網指派等四大步驟建構，再結合特殊吸引點旅次處理、旅次矩陣轉換、尖峰小時推估模組等作業程序。其運輸需求模式分析時段包含晨峰時段、昏峰時段、離峰、晨峰小時、昏峰小時，旅次目的共有 7 個，主要分析之旅次目的分為：家—工作、家—學校 6-14 歲、家—學校 15 歲以上、家—其他、非家共 5 個(周汶叡 et al., 2018)。

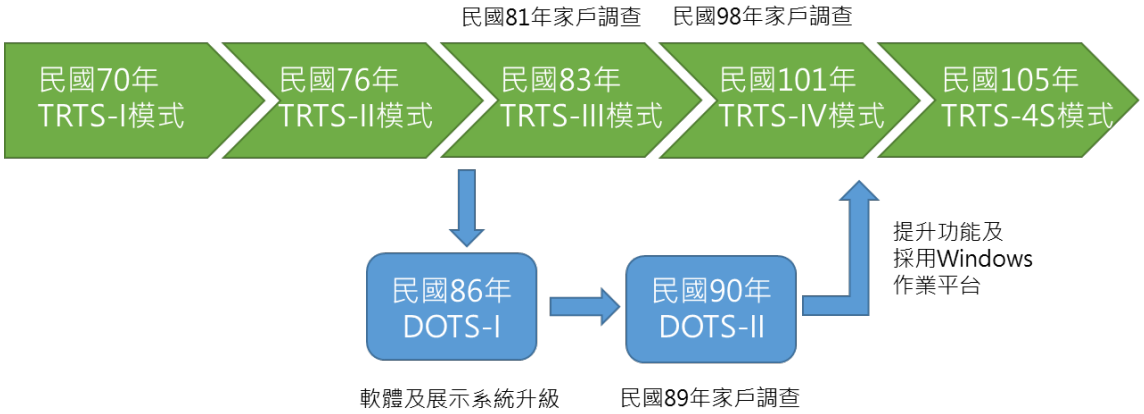


圖 2.1 TRTS 模式發展歷程 (周汶叡 et al., 2018)

臺北都會區運輸需求預測模式使用細緻的資料與參數，來對臺北都會區的交通建立模型並提供各種交通運輸的預測，雖然 TRTS 模型可以對於整體運量有很好的預測，但城市中的交通行為相當複雜，對於短時間內交通運量的變化可能無法掌握，因此本研究能在此方面補上其預測的缺口。

3. 臺北捷運運量預測文獻

林炫洋 (1982) 探討臺北市引進中運量捷運系統後，其捷運路線上運輸需求狀況，並尋求建立一合理的運具分配模式，藉以預估未來目標年之路線運量，以為規劃該捷運系統之參考。以系統分析方法進行研究，利用調查資料推導出羅吉斯運具分配模式，並以之作運量預測。該篇研究利用統計的羅吉特模型以及問卷來預測並驗證臺北捷運的運量。

有別於傳統統計方法來進行運量預測，蘇怡瑄 (2009)使用倒傳遞類神經網路臺北車站出站人數進行預測；林泉亨 (2015) 透過文字探勘方法建立話題特徵及影像偵測方法取得社群網路中影像資料中人臉的的特徵，將其結合機器學習方法中的支持向量機(Support Vector Machine)，隨機森林(Random Forest)，及隨機梯度提升方法(Stochastic Gradient Boosting)，建立預測短期各捷運站的旅客出站數之模式，並以捷運站出站人數進行驗證。

從臺北捷運過去的運量研究中可以發現，對於長期與短期的運量都有相關的研究，本篇研究則將利用深度學習的方法，將過去傳統統計與機器學習方法無法處理的特殊事件下的運量進行預測，期能補上此研究缺口。

2.2 臺北市大型活動發展回顧

本節透過回顧舉辦大型活動的特色，了解大型活動如何產生，以及對於城市的影響。接著回顧臺北市對於舉辦大型活動的規範，了解不同類型活動與使用場館間的差異，作為之後選擇場站的參考。最後回顧大型活動對捷運的影響，了解舉辦活動時人潮移動的特徵。

1.大型活動特性回顧

根據內政部 2015 年所訂定的大型活動安全管理要點中對大型活動的定義:舉辦每場次預計參加或聚集人數達一千人以上，且持續二小時以上之下列活動：一、體育競技活動。二、演唱會、音樂會等演出或類似之娛樂活動（派對、祭、季等）。三、展覽（售）、人才招募會、博覽會等活動。四、燈會、花會、廟會、煙火晚會等活動。五、民俗節慶、原住民慶典等活動。此篇研究觀察與預測的臺北小巨蛋為舉辦演唱會、體育競技活動等娛樂活動的舉辦場所。

Pereira et al. (2015)提到特別事件通常對大眾運輸系統有潛在的破壞性影響，因為這些活動非慣性的行為模式，以至於難以預測與事先規劃。除了非常大型的活動（如：奧運、世界盃足球賽）外，政府因為人工追蹤發生在大城市中的事件耗時費力，再者即使有事件的列表，他們的影響難以估計，特別是同時有一個以上的事件發生的時候，因為這兩個原因而鮮少採取特別規劃的措施。

2.大型活動交通維持計畫

臺北市政府為了降低大型活動期間對交通的衝擊，並且維持交通安全與順暢，訂定了臺北市《大型路外活動交通維持作業辦法》，圖 2.1 為審查交通維持作業的原則。此原則也可以作為本研究選擇場館以及捷運站的根據。雖然訂定了大型活動舉辦時需有交通維持計畫，但對於非活動的參加者往往到達活動發生地的捷運站才發現目的地有活動舉辦，造成擁擠並付出許多時間成本；對於活動參加者而言，亦會在活動開始前後形成人潮造成空間的擁擠。

活動交維層級審查原則

類型	性質	規模		型態	層級	備註
		每日參觀人次	總參觀人次			
路外活動(註 1)	百貨公司或大賣場營業總樓地板面積≧1 萬m^2舉辦開幕式或週年慶	–	–	開幕	幹事	
				週年慶	幕僚	
	體育館場舉辦演唱會或晚會	≧2 萬	–	非例行	幹事	捷運 300m 內
		≧1 萬				捷運 300m 外
	信義及南港展覽館舉辦展覽	≧1.5 萬		例行	幕僚	專業展
		≧3 萬				消費展
	戶外場地大型活動	–	≧30 萬	非例行	委員	1、非例行：非屬經常性舉辦活動。 2、例行：屬歷年經常性舉辦活動。
				例行	幹事	
道路活動(註 1)	–	≧3 萬		非例行	委員	
				例行	幹事	
	–	1 萬≦人<3 萬		非例行	幹事	
				例行	幕僚	
	–	<1 萬		–	幕僚	

圖 2.2 臺北市大型活動交通維持作業審查原則

(臺北市政府交通局, 2009)

2.3 運量預測方法回顧

本節首先回顧在運量預測中機器學習方法與統計方法如何進行運量的預測，其中機器學習擅長處理非線性資料的特性，使預測結果更符合真實世界運量發生的特徵。深度學習作為機器學習的一部分，在現代的資料科學中被使用，透過多層的結構，損失函數與優化器調整模型權重，來讓預測達到理想的成果。

1. 運量預測方法概述

蔡宗憲 (2006)研究提到，運輸預測模式的建立主要著重於中長期的總體預測模式，而短期預測模式的重要性，在於其預測結果可以提供後續短期營運規劃所用。交通資料最常使用兩種方法來建立模型:統計方法或機器學習方法。

統計方法為收集、組織以及解釋數學數據的方法。統計數據具有可靠且被廣泛接受的數學基礎，以及可以提供有關創建數據機制的見解。但捷運短期的運量複雜且具有非線性的特徵，統計分析則沒有能力處理這樣的問題。

機器學習方法結合學習、適應、演化與模糊邏輯的元素來建立智能的模型，

是從結構性資料到非結構化的開端(Karlaftis & Vlahogianni, 2011)。自從進入電腦時代以來，研究者一直致力於讓電腦擁有學習能力，解決這個問題一直是人工智慧中最有挑戰性以及引人入勝的長期目標。在現在的機器學習研究領域中，將焦點放在三個領域:1.任務導向的研究:開發及分析機器學習系統來改善對一個預定的問題的表現。2.認知模擬:調查與電腦模擬人類的學習歷程。3.理論分析:獨立於應用領域可能的學習方法和算法的空間理論探索(Michalski et al., 2013)。在機器學習領域中，短期旅運需求的預測因為大量當代的資料來源如:智慧卡、交通感測器、探針、GPS 與監視器等，是結構富有細節與複雜度的資料(Nguyen et al., 2018)而吸引了許多人的研究興趣，對於分析人類活動可以用相較過去更微觀的方式來進行分析。

2. 深度學習預測架構

深度學習是機器學習中的一個子集合:從數據中學習表示形式的新方法，重點是學習越來越具有意義的表示形式的連續層狀架構，深度學習中資料的特徵都是透過多層類神經網路(Neural Network)來學習。

損失函數(loss function)是神經網路中用來量測模型輸出與實際值差異的函數，此函數會計算出一個距離分數(distance score)來顯示模型預測的結果好壞。深度學習的根本就是使用這個分數做為回饋訊號，朝著更低的損失分數來調整模型權重，而執行此動作的功能為優化器(optimizer)，優化器能夠應用反向傳播法(backpropagation)來調整權重。訓練的最初，模型的權重被隨意的指派，資料在模型當中被隨意的連結，也因為如此輸出往往跟理想相去甚遠而得到非常高的 loss score，模型透過每次的迭代訓練得到不同的輸出，優化器會調整模型權重來讓每次輸出的 loss score 盡可能的變小，最後得到一個權重調整後有最少損失值的模型(Chollet, 2018)。

2.4 短期交通預測研究回顧

本節回顧以不同機器學習方法進行捷運運量預測的文獻，了解不同方法在如何預測運量。回顧的文獻皆考慮到特殊事件以及其他外部因子對於捷運運量的影響。

1.機器學習方法預測文獻回顧

過去的 20 年，人工智慧已經被廣泛的使用在短期運量預測中，已經部份取代過去的時間序列以及數學方法(López et al., 2018)。在交通研究中，類神經網路已經主要被使用作為資料處理方法，因為它能使用大量多面向資料的能力、建立模型的彈性、學習與概括化能力、對資料的適應能力以及良好的預測能力(Karlaftis & Vlahogianni, 2011)。國內亦有魏健宏等人整理利用類神經網路建立的相關交通預測如圖 2.3，分別有分類與歸類、最佳化以及預測問題。

類神經網路功能	交通運輸課題
分類與歸類	車種、車牌辨識
	道路維修
	道路事件偵測
最佳化	交通控制
	方案評估
	路線與排程問題
預測	駕駛人行為模擬
	旅運預測
	車輛操控

圖 2.4 類神經網路在交通領域之應用 (魏健宏 & 陳奕志, 2001)

Li et al. (2017)使用多尺度徑相基函數網路預測特殊事件下的北京地鐵乘客，此方法無須與事件相關的背景知識，並且只依靠有限的捷運運量數據。提出的方法包含線性子模型與非線性子模型，線性子模型可以用來追蹤輸入與輸出之間的線性關係，以及 MSRBF 子模型可以被用來捕捉非線性系統的效應，並使用最小平方正交匹配追蹤法(Matching Pursuit Orthogonal Least Squares, MPOLS)，選擇最顯著的條件建立一個精簡的線性變數模型。此篇研究預測三個捷運站在不同情境下的運量，並以 MSRBF 與支援向量機、決策樹以及單尺度徑向基模型進行比

較，並以平均百分誤差(mean absolute percentage error, MAPE)、方差絕對百分誤差(variance of absolute percentage error, VAPE)、均方根誤差(root mean square error ,RMSE)三種標準來驗證模型精準度，預測結果顯示此篇研究提出的多尺度徑向基函數比其他三種方法有較高的精準度，並可以提前預測 30 分鐘的運量發生，且特別是在緊急情況發生時能準確預測人流。文中也提到此篇研究所提出的模型只適用於北京地鐵，無法應用在其他研究當中。在這篇研究中所討論的特殊運量只限於特殊事件開始前，而未考慮到演奏會結束後散場的人潮是更加集中的，對於特殊事件的人流預測應該包含事件開始與結束後都必須進行預測。

Chen et al. (2019)使用 GARCH(Generalized Autoregressive Conditional Heteroscedasticity)模型使用中國南京奧林匹克體育中心附近兩個捷運站的智慧卡資料，預測特殊事件下的進出站人潮，本篇研究專注於針對突然發生的進站或出站人流建立模型，並探討如何辨識特殊事件下乘客運量時間序列資料中的分布殘差，不同的殘差分布可以被更好的來詮釋在特殊事件下的預測殘差。使用統計方法的 GARCH 模型來預測非線性資料的波動性，配合 ARIMA 模型預測線性資料。此篇研究使用了四個的 ARIMA-GARCH 模型與一個傳統 ARIMA 模型來對兩個捷運站進行預測，並使用交叉效率驗證(cross-validation effectiveness)來針對平均百分誤差(mean absolute percentage error, MAPE)、均方根誤差(root mean square error, RMSE)與平均預測區間長度(mean prediction interval length, MPIL)這三個誤差進行比較(見圖 2.5)，預測結果顯示 ARIMA-NAGARCH 在奧林匹克東站(Olympic-Stadium-East, OSE)與奧林匹克體育中心站(Olympic-Sports- Center, OSC)皆有最低的預測誤差。此篇文章基於點預測和區間估計，幫助交通機構監控捷運的性能，並提前採取適當的對策以應對意外的人潮，並能估計場站擁擠的程度，減少耽誤旅客並防止服務中斷。本篇研究考慮到不同捷運路線所搭乘的乘客屬性不同，以及預測時間選擇活動容易發生的下午一點半到晚間十一點半，忽略其他時間帶來的影響以及減少資料的複雜度。

Station	Model	MAPE(%)	RMSE	MPIL
OSE	ARIMA	27.971	99.033	1177.994
	ARIMA-SGARCH	16.420	72.439	582.174
	ARIMA-EGARCH	11.305	49.107	660.472
	ARIMA-GJRGARCH	15.471	70.938	625.871
	ARIMA-NAGARCH	9.056	40.906	543.684
OSC	ARIMA	58.229	37.538	353.857
	ARIMA-SGARCH	23.334	36.476	244.657
	ARIMA-EGARCH	21.513	29.041	291.778
	ARIMA-GJRGARCH	22.279	34.555	233.588
	ARIMA-NAGARCH	17.714	24.180	223.607

圖 2.5 交叉驗證模型預測表現 Chen et al. (2019)

Ding et al. (2016)使用梯度提升決策樹方法來預測地鐵運量與捕捉獨立變數間的關係，本篇特別之處在於討論其他研究鮮少討論的公車轉乘與時間屬性和運量間的關係。文中提到預測精準度與解釋力是預測模型最重要的兩個目標，梯度提升決策樹可以有別於其他機器學習方法，可以識別並評價影響因子的影響程度，並且保持高的預測精準性，圖 2.6 依照不同樹的複雜度(Tree Complexity)以及不同限縮(Shrinkage)係數來進行變數選擇。由此研究可知梯度提升決策樹在選擇輸入變數有很好的效果，並且具有解釋力。

Variable	DWL Subway Station		FXM Subway Station		HLG Subway Station	
	Rank	Relative Importance (%)	Rank	Relative Importance (%)	Rank	Relative Importance (%)
$METRO_{t-1}$	1	82.03	1	85.06	1	92.28
$METRO_{t-2}$	4	1.71	4	0.95	4	0.20
$METRO_{t-3}$	5	1.65	5	0.77	3	0.46
BUS_t	2	9.41	3	4.42	8	0.08
BUS_{t-1}	6	0.55	6	0.44	6	0.11
BUS_{t-2}	8	0.40	7	0.34	9	0.06
BUS_{t-3}	7	0.41	8	0.27	5	0.16
Time of day	3	3.55	2	7.59	2	6.54
Date of month	10	0.12	9	0.10	7	0.10
Day of week	9	0.17	10	0.06	10	0.01

圖 2.6 不同變數對運量的影響(Ding et al., 2016)

2.深度學習預測方法文獻回顧

近年深度學習在類神經網路、人工智慧、圖形辨識、最佳化問題與訊號處理

的領域中快速發展。學術界給了深度學習更細緻的描述，大致來自於兩點:由多層隱藏層或多個非線性資訊處理組成，或是在更高、更抽象維度中利用監督或非監督式學習方法來學習資料的表徵，因為乘客人流的預測是一個複雜且非線性問題(L. Liu & Chen, 2017)。在此部分主要回顧深度學習方法如何解決時空預測問題，並進行相關的文獻回顧，了解深度學習方法如何解決此類問題。透過回顧深度學習的相關研究，了解目前機器學習中近代主流的深度學習方法，並且了解 CNN 與 RNN-LSTM 演算法能夠各自捕捉空間與時間關係，進一步回顧兩者融合使用的 ConvLSTM 演算法，能夠有效的捕捉真實世界中的時空關係，適合用來預測捷運短期運量。

Yann LeCun 等人在 1995 年提出了卷積神經網路(Convolutional Neural Network, CNN)來改善過去在標準的全連接神經網路(Fully connected Neural Network, FNN)中圖形辨識的問題。過去在圖形辨識中使用全連接網路，計算負擔大且運算速度慢，且並未考慮到像素的空間分布，只針對圖片中每個單一像素去作判斷，完全捨棄重要的影像特徵。圖 2.7 標準的卷積神經網路，卷積神經網路有三大特徵:感受視野(local receptive fileds)、神經元共享權重(shared weights)以及空間或時間的降採樣(spatial or temporal subsampling)，也被稱為池化(pooling)。隱藏層神經元和輸入層 n*n 個神經元相連，而這個 n*n 的大小就是感受視野，透過這個「視野」的移動會產生特徵圖(feature maps)，可以找出圖案中的基本視覺特徵(如:邊界、點、角落)。由輸入層產生特徵圖進入隱藏層運算時，每個特徵圖有共享權重的特徵，因為每個特徵圖都是原圖元素的一部分，因此神經元中的權重與誤差(bias)都必須一樣。最後的降取樣是為了簡化出卷積層輸出的結果，通常使用最大池化(Max Pooling)，可以避免過適並保留影像主要特徵(LeCun & Bengio, 1995)。

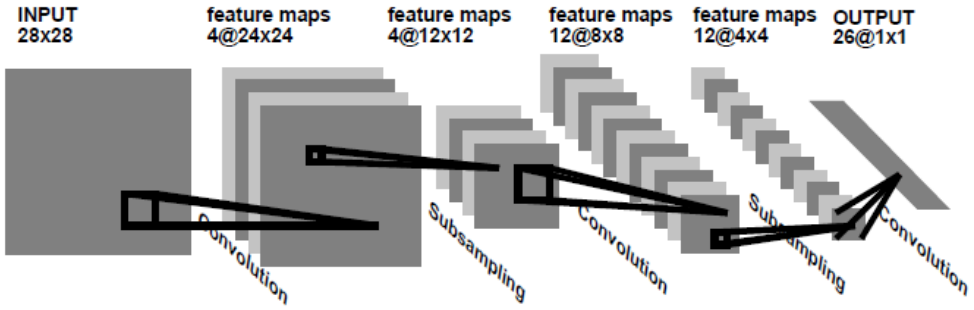


圖 2.7 CNN 應用於手寫辨識流程圖(LeCun & Bengio, 1995)

遞歸神經網路(Recurrent Neural Network,RNN)也是近年深受歡迎的深度學習方法，其優點是可以將先前的預測結果考慮到目前的預測當中，因此 RNN 被廣泛運用在語音辨識、翻譯、預測字詞等領域。但在一般的情況下前後的資料不一定有關係，而是和較先前的資料有關，因此 Hochreiter and Schmidhuber (1997)利用長短期記憶(Long Short-Term Memory, LSTM)與 RNN 結合解決了這個問題，此架構對於許多問題有非常好的表現，因此被廣泛的使用。圖 2.8 與圖 2.9 為 Gers et al. (1999)將原有的 RNN-LSTM 架構中增加遺忘門(forget gate layer)，讓神經元可以決定要記住多少的資訊量進到輸入門中，將舊的數值更新為新的狀態。

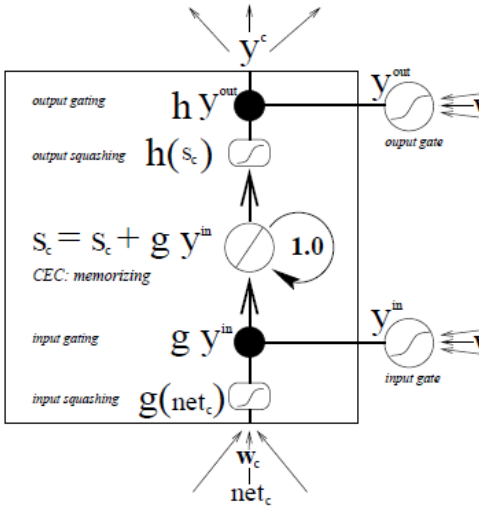


圖 2.8 原本的 LSTM-RNN 神經元

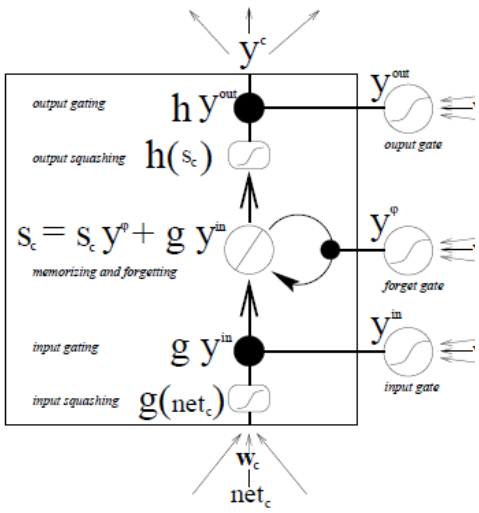


圖 2.9 改進的 LSTM-RNN 神經元

由上述文獻回顧可知，CNN 擅長捕捉圖片中像素的空間關係，LSTM-RNN 擅長預測具有順序性(Sequential)的資料，但捷運運量的預測涉及人在不同時間在空間中的移動，也就是為一時空序列(spatiotemporal sequence)問題，此問題在深

度學習中一直以來都是重大的挑戰。

Zhang et al. (2017)提出 ST-ResNet(Spatial Temporal Residual Network)深度學習架構(見圖 2.10)，分別預測進入的以及離開的兩種人流，整體性預測城市的人流。如下圖所示，考慮外部因子(天氣、特殊活動)，並且分別建立三個 CNN 模型分別為時間接近度，週期和趨勢，最後聚合(aggregate)成為輸出。本篇分別研究使用北京的計程車以及紐約的公共單車的人流，將此篇提出的 ST-ResNet 和其他 6 個模型比較，在預測北京計程車時提出了 7 個不同的模型架構以及參數(見圖 2.11)，研究結果 7 個 ST-ResNet 皆有比傳統模型更佳的表現；在紐約自行車的預測中(見圖 2.12)，這邊加入了不同的 DeepST 模型進行比較，ST-ResNet 仍有最好的預測表現。

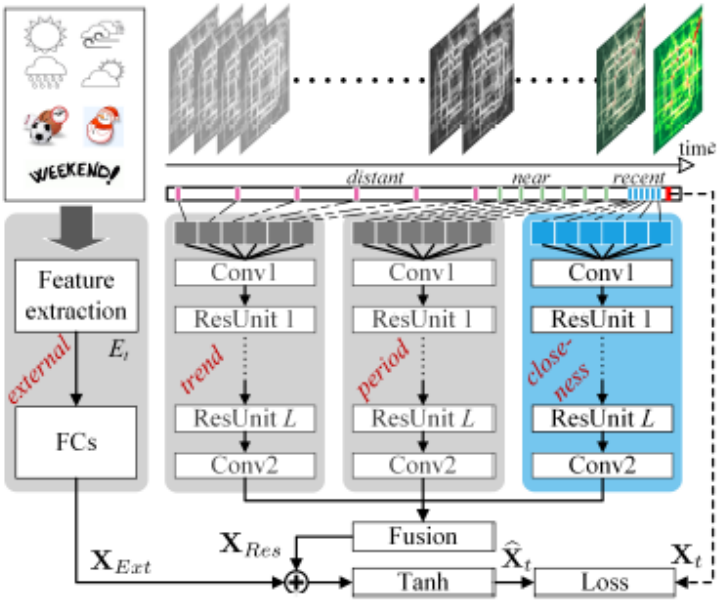


圖 2.10 ST-ResNet 架構圖(Zhang et al., 2017)

Model		RMSE
HA		57.69
ARIMA		22.78
SARIMA		26.88
VAR		22.88
ST-ANN		19.57
Deep-ST		18.18
	ST-ResNet	
L2-E	2 residual units + E	17.67
L4-E	4 residual units + E	17.51
L12-E	12 residual units +E	16.89
L12-E-BN	L12-E with BN	16.69
L12-single-E	12 residual units(1 conv) + E	17.40
L12	12 residual units	17.00
L12-E-noFusion	12 residual units + E without fusion	17.96

圖 2.11 北京計程車預測結果比較

Model	RMSE
ARIMA	10.07
SARIMA	10.56
VAR	9.92
DeepST-C	8.39
DeepST-CP	7.64
DeepST-CPT	7.56
DeepST-CPTM	7.43
ST-ResNet	6.33

圖 2.12 紐約公共單車預測結果比較

Xingjian et al. (2015)提出 ConvLSTM 神經網路的架構(圖 2.13)來預測降雨預報的時空分布，研究團隊研究證實這種神經網路能比傳統 FC-LSTM(Fully-connected LSTM)更有效處理時空序列的問題。

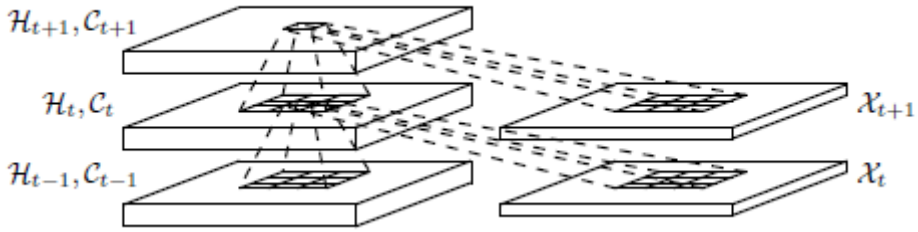


圖 2.13 ConvLSTM 神經網路架構圖 Xingjian et al. (2015)

马晓磊 (2017)使用 ConvLSTM 架構預測北京地鐵乘客運量，圖 2.14 為此篇研究提出的模型架構圖。此篇研究將地鐵路網轉換為網格圖片，得到捷運站的網格座標後，將該時刻的運量作為該網格的像素值，映射到網格圖片中，其餘無站點處設為 0，分別產生不同時刻下的運量圖片，先輸入到卷積層中分析空間特徵，再輸入到長短期預測層中進行時間序列分析。預測結果與 8 種機器學習方法比較 MSE(Mean Square Error)，此研究使用的 ConvLSTM 方法有最低的 MSE 59.078，為表現最好的預測方法。

名稱	尺寸	激勵函數
輸入層	(60, 60, 1)	
卷積層(16)	(3, 3)	
池化層(16)	(2, 2)	relu
卷積層(32)	(3, 3)	
池化層(32)	(2, 2)	relu
卷積層(64)	(3, 3)	
池化層(64)	(2, 2)	relu
全連接層	1335	
重塑層	(5, 267)	
LSTM層	2000	tanh
LSTM層	2000	tanh
Dropout層	0.2	
全連接層	1335	

圖 2.14 運量預測模型架構(马晓磊, 2017)

2.5 ConvLSTM 演算法與模型

本節介紹本研究所採用卷積長短期記憶神經網路(Convolutional Long-Short Term Memory, ConvLSTM)的理論與發展過程，從該神經網路使用的 TensorFlow 架構到此模型所屬的 Keras 套件，最後進行模型介紹。

1. Keras

Keras 為 Python 編寫的一個深度學習框架，旨在提供一個便利的方式來定義與訓練任何深度學習模型。Keras 最早開始為了快速執行實驗而設計給研究人員使用，是 ONEIROS（開放式神經電子智慧機器人作業系統）專案研究工作的部分產物。其特色在於:

(1). 可以在 CPU 與 GPU 上運行

(2). API 易於使用，可以快速地建立深度學習模型

(3). 支援內建的卷積神經網路(用於電腦視覺)、遞歸神經網路(用於序列資料處理)或是兩者的任意組合

(4). 支援任意神經網路架構:複數的輸入或輸出模型、隱藏層共享或模型共享等，這代表 Keras 適合建立任何深度學習網路模型。

(5). Keras 在 MIT 的規範底下發布，意味著它可以在任何商業企劃中免費使用

(6). 支援 Python 2 與 Python3

Keras 是一個模型級的套件庫為深度學習模型提供高級模型建構區塊，他不需要處理低階的程式運行像是張量(tensor)操作與分化，相反的 Keras 依靠特殊且經過良好優化的張量套件作為後端引擎來處理張量。Keras 並沒有選擇單一張量套件並將 Keras 綁定到該庫，而是以模組化的方式處理問題。目前三個主流使用的後端為 TensorFlow、Theano 和 Microsoft Cognitive Toolkit(CNTK)，任何使用 Keras 程式碼都可以在其所支援的後端中運行。圖 2.15 中可以看到，Keras 可以將多個不同的後端引擎無縫地接入。以 TensorFlow 為例，在 CPU 上有 Eigen 套件處理張量；在 GPU 上有 NVIDIA 開發的 CUDA 來進行運算(Chollet, 2018) 。

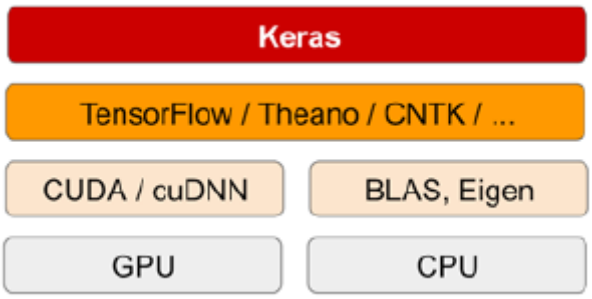


圖 2.15 Keras 軟體與硬體架構

2. TensorFlow

TensorFlow 最初為 Google Brian 所開發，為一開源軟體庫。在 2015 時 Google 將之開源，為現今重要的深度學習框架之一，它支援各式不同的深度學習演算法，並已應用於各大企業服務上，Ex: Google, Youtube, Airbnb, Paypal ... 等。此外，TensorFlow 也支援在各式不同的 device 上運行深度學習 Ex: TensorFlow Lite 、 Tensorflow.js 等等。TensorFlow 為目前最受歡迎的機器學習、深度學習開源專案。不管是 github fork 的數量、論文的使用次數以及熱門程度，均比其他的框架來的多(Dan, 2019) 。

3. ConvLSTM 模型

ConvLSTM 模型是由 FC-LSTM(Fully Connected Long Short Term Memory)演變而來，與 FC-LSTM 不同的是內部的矩陣運算被卷積運算所代替，因此輸入從一維變成了三維。一般而言，LSTM 的輸入為(samples, time steps, features)，而 ConvLSTM 的輸入為三維的圖片，因此圖片的輸入為四維(samples, channels, rows, cols)，而 LSTM 的特徵值會被替換成圖片的特徵，因此 ConvLSTM 整體的輸入則為五維(samples, time steps, channels, rows, cols)。ConvLSTM 的輸出依靠 return_sequence 參數來決定，若此參數為真，輸出為一五維張量(samples, time steps, filters, rows, cols)；若為非，輸出為序列的最後一個值，為一個四維向量(samples, filters, rows, cols) (Xavier, 2019) 。

此外，在模型學習資料中的特徵時，模型的參數也會對預測的結果有所影響，以下介紹在模型設定中的重要參數(Chollet, 2015):

(1). 激勵函數(Activation function):激勵函數定義了該節點在得到特定的輸入或輸入的集合後所得到的輸出，此機制類似於電腦訊號的 0(關)與 1(開)，使用激勵函數的目的在於引入非線性的性質到網路之中，而目前最被廣泛接受的激勵函數為 ReLU。

(2). 損失函數(Loss function): 為模型編譯(compile)需要具備的參數之一。指一種將一個事件（在一個樣本空間中的一個元素）映射到一個表達與其事件相關的經濟成本或機會成本的實數上的一種函數，藉此直觀表示的一些"成本"與事件的關聯，而深度學習的目標是尋求最佳化的解，因此深度學習的目標即為將損失函數降到最小。

(3). 優化器(Optimizer): 為模型編譯(compile)需要具備的參數之一，最常被使用的算法為梯度下降法(Gradient Decent)，此算法使用各參數的梯度值來最大(目標函數為負)或最小化(目標函數為正)損失函數，透過尋找損失函數最小值，調整該層神經元的權重(weight)與偏值(bias)，最終使模型收斂。

2.6 小結

文獻回顧的開頭首先回顧了政府對於大型活動的定義為何，以及大型活動舉辦時必須申請交通維持計畫的標準。但對於乘客來說，即使知道某地將有大型活動舉辦，卻對於捷運站擁擠程度不了解，因此若能依靠一個精準的運量預測模型，能夠分析大型活動時民眾使用捷運的特徵(pattern)，就能供乘客能有更多交通方式的選擇以及出行計畫的參考。

大型活動或特殊事件下的運量預測問題對於傳統的統計方法與新興的機器學習方法都是相當具有挑戰性的問題，學界提出了非常多不同的模型來處理此類問題，也都有不錯的成果。但過去的短期運量研究中大多著重於時間序列以及運量的預測上，智慧卡資料中還包含了起迄站資料，這其中包含了地理空間的訊息，透過深度學習結合圖形辨識模型與長短期記憶模型，可以將運量資料投射到地圖網格當中進行時空分析，可以更加瞭解一個城市中大型事件發生時，人流如何流動的過程。

第三章 研究方法

3.1 研究流程

本研究針對臺北捷運中因為大型活動所產生的短期運量，以深度學習的方法進行預測，透過時空分析的深度學習方法，預測未來 15 分鐘運量，透過驗證方法與不同預測任務，了解大型活動時民眾在空間中的移動特徵，提供民眾使用大眾運輸的決策，以及預測結果供捷運公司在場站的營運與管理參考。依據前述目標，擬定研究流程(見圖 3.1)。

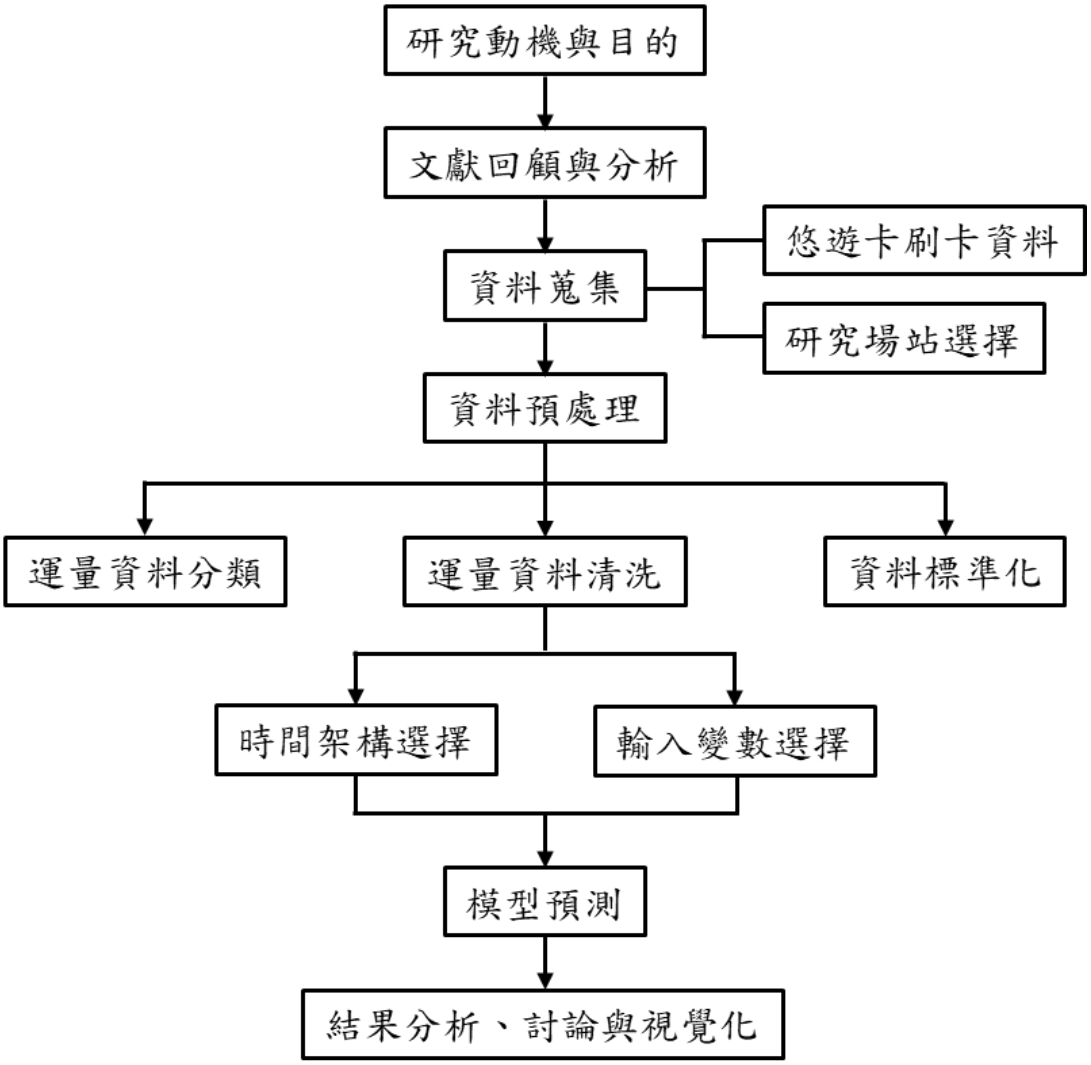


圖 3.1 研究流程圖

3.2 研究範圍

本研究以臺北捷運路網作為研究範圍(未包含機場捷運、貓空纜車與淡海輕軌) 見圖 3.2，其路網範圍涵蓋臺北市與新北市，主要共有 5 條主線與 2 條單站支線，路線里程共 136.6 公里，營運車站共 117 站，扣除 9 個路線分離的轉乘站，實際站數為 108 站。2018 年日均運量達兩百萬餘人次，為臺北大眾交通運輸骨幹。本研究採用全站運量作為模型輸入。

圖 3.2 臺北捷運路網圖

資料來源: (臺北大眾捷運公司，2019)

3.3 資料說明

1.悠遊卡刷卡資料

圖 3.3 為本研究主要使用之運量資料，由臺北市交通局運輸資訊科所授權提供之每五分鐘一筆的悠遊卡運量資料，時間為 2016/1/1 1:05 至 2019/9/29 23:55 的悠遊卡全部共 130 站(包含重複站名但場站代號不同以及未開放場站代碼)之刷卡資料。本研究採用 2016/1/1 6:00 至 2019/7/01 00:00，每日早上 6 點至晚間 12 點之運量紀錄來建構資料集。2016 年後捷運路網就成形為今日的樣貌，未再新增任何任何站點，這對於預測資料來說，可以讓資料集是穩定且有助於模型預測的；而選擇開始與結束時間是參照各路線首/末班車的發車時間，也便於每日資料的預測與分割。

資料蒐集時間	場站代號	進站人數	出站人數	整批資料更新時間	存入DB時間	該筆資料更新時間	該筆資料更新時間
2016/1/13 00:00	83	7	29	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	67	1	5	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	97	2	13	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	48	2	32	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	122	1	30	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	34	0	4	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	62	4	1	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	108	14	0	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	133	3	1	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	25	0	1	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	174	0	1	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	59	4	14	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	86	54	72	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	76	0	7	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	127	4	25	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	31	0	2	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	92	18	10	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	8	9	28	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	79	1	37	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	107	3	1	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	128	2	10	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	42	10	0	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	56	7	11	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	89	11	8	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	98	0	20	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	11	6	13	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13
2016/1/13 00:00	47	3	36	10:01.2	10:01.2	2016/1/13 00:00	2016/1/13

圖 3.3. 每五分鐘捷運站進出資料

資料來源: (臺北市交通局運輸資訊科)

2.資料預處理

臺北捷運各路線於早上 6 點自各端點發車，並於晚間 12 點左右發出末班車，本

研究將採用此時間區段的資料作為製作資料集材料，並根據交通局提供的運量資料以 5 分鐘作為一個時步(time step)，則每站每日將產生 216 筆進/出站運量資料，時間內全系統共 29,927,772 筆資料。在過去研究中時步長度的選擇沒有一個統一的標準，因為針對不同的運輸系統或是班距也都不一定相同，在本研究中選擇五分鐘作為基本的時步，因為臺北捷運除文湖線為中運量路線外，其餘為高運量路線，而在高運量路線中各路線離峰時段班距多為 4~12 分鐘之間，而尖峰班距在 4~7 分鐘之間，因此選擇五分鐘作為資料基本的時步，輸入模型前會依照不同的需要，將此五分鐘的運量製作成本研究主要使用的 15 分鐘預測資料以及依照預測需要不同時步的資料集。

使用 Python 3.7 語言在 Jupyter notebook 環境中進行資料處理。首先讀取運量資料，賦予欄位名稱以及將時間資料轉換為 datetime 的格式以利之後進行時間的計算與處理，刪除資料更新與儲存時間等無關欄位，最後將三份運量資料合併(見圖 3.4)。在此份資料中只有保留進/出站大於 0 的資料，因此必須建立一個完整的時間序列補足資料的空缺，並裁切出前述的時間區間。將運量資料處理完成後需要將資料整理為(樣本數,時步,特徵值,行,列)的 5 維矩陣，即完成輸入資料的處理。

3.大型事件與預測場站選擇

本次研究選擇臺北小巨蛋站作為特殊事件活動的資料來源，臺北小巨蛋為一多功能體育館，除體育活動外亦經常作為演唱會、頒獎典禮等大型活動的舉辦場地，該場館在網站公布過去與未來每場活動相關資訊，利於在建立資料集時蒐集並標記活動日期。此資料從 2013 年開始記錄至 2019 年 10 月共 569 筆(天)，在本次研究所選擇的時間內的活動共 349 筆(天)。

為了增加特殊活動資料的筆數，讓特殊活動的特徵可以在運量資料中被模型學習，加入了國定假日與普通例假日也作為特殊活動資料集的樣本，此特殊活動共包含 563 筆(天)。

3.4 分析流程

圖 3.4 為本研究研擬的分析流程，本研究透過深度學習方法進行悠遊卡資料分析，以 ConvLSTM 產生圖片並進行時空序列分析(spatiotemporal analysis)，透過建構以過去 15 分鐘預測未來 15 分鐘的運量資料集預測臺北捷運運量，了解在不同的日形態下對於捷運系統以及都市內部人口移動的影響，並提供精準的運量預測供管理單位在列車調度上能有更精準的派車策略。

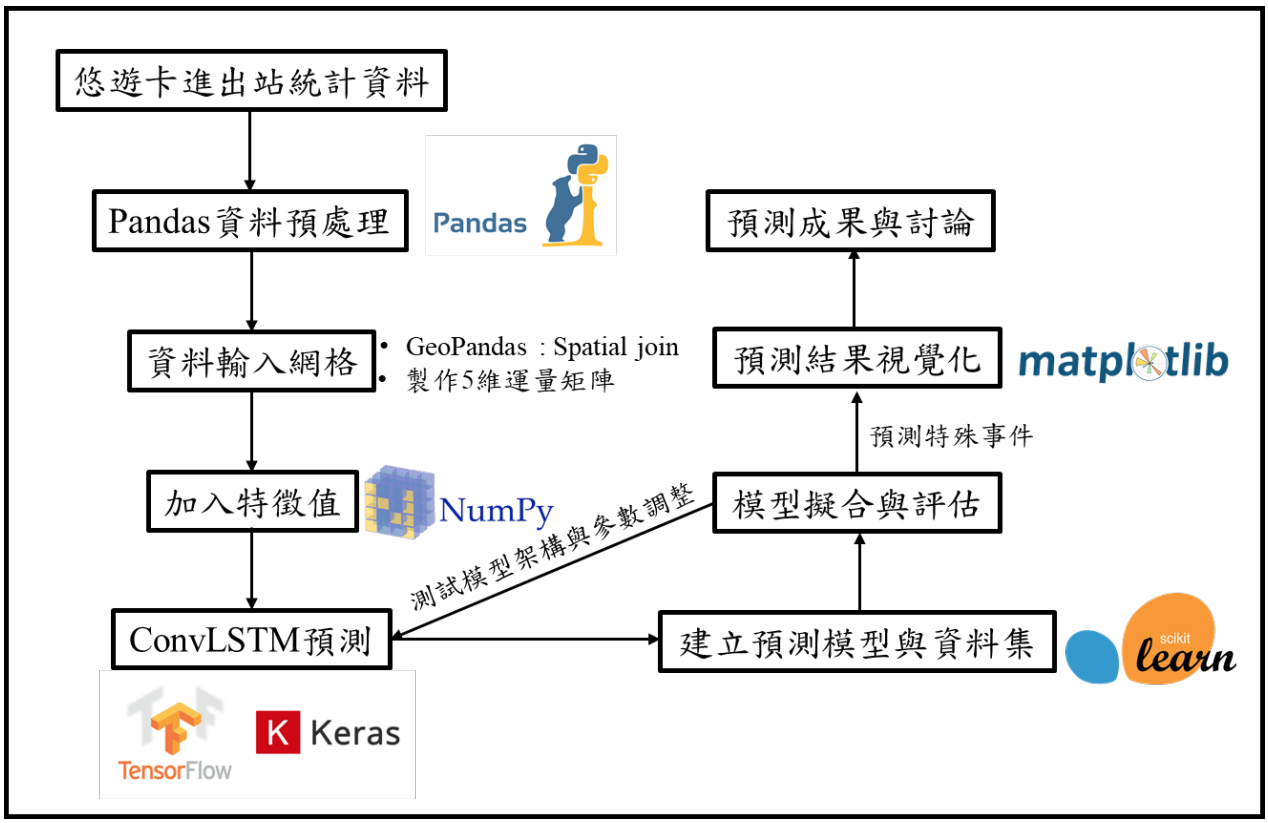


圖 3.4 分析流程圖(自行整理)

1. Pandas 資料預處理

(1). 資料清洗與整理

由於悠遊卡運量統計資料屬於原始資料(raw data)，其中有許多不必要的資料成為往後分析時的雜訊，因此必須在此階段將不相關的欄位、超出觀測時間的刷卡紀錄以及同站進出的紀錄進行刪除。

(2). 資料輸入網格(Convert data to grid)

因為 ConvLSTM 輸入的格式為圖片的形式，而運量資料為一維的數據，因此必須

將運量資料轉換為圖片的形式，需要根據資料需求的尺寸建立一個值為 0 的矩陣，將運量加入到矩陣中進行輸入，運量在這邊可以理解為圖片中的像素值，每個網格中只會有唯一的像素值。因為此方法將現實世界的空間映射到數值的陣列中，因此須將每個捷運站以空間結合(Spatial join)的方式對應到矩陣中的位置。首先必須先決定圖片的範圍大小，表 3.1 為臺北捷運在四方位極點的捷運站，可知南北相差約 0.21 度，東西亦相差 0.21 度，後續處理會依照此間隔與網格大小進行考量，選擇適合的網格大小以避免鄰近的捷運站共用同一網格，而造成該站的資料消失。

站名	經度	緯度	位置
淡水站	121.45°	25.17°	極北
新店站	121.54°	24.96°	極南
迴龍站	121.41°	25.02°	極西
南港展覽館站	121.62°	25.06°	極東

表 3.1 臺北捷運極點

2. ConvLSTM 預測架構

马晓磊 (2017)的研究中提到，在傳統的捷運運量預測中，運量都被視為一個單純的向量數據，但會因此失去了許多特徵訊息進而影響預測，因為北京的路網規劃為單中心環形分布，因此會產生明顯的空間與時間的差異。而在臺北與新北市，雖然臺灣的居住型態為住商混合，但因為臺北仍有商業區與居住區的區別，加上市中心與城市周邊房價的差異，使民眾在不同時間朝特定方向移動，因此也會產生運量空間分布的差異；除了空間相關性外，捷運運量也表現出明顯的時間相關性，在第四代臺北都會區運輸需求系統預測模型(TRTS-4S)的研究中也提到特定時段捷運會出現高峰，因此考慮時間的相關性是必須的，因此本研究選擇 ConvLSTM 來捕捉空間中隨時間變化的運量特徵。

本次運量的預測問題使用 ConvLSTM 神經網路來進行，而根據前面所提及的，本研究關注的是一個時空序列的問題，圖 3.5 的公式說明了我們的問題是會根據過去 J 個觀測，來預測接下來最可能發生的 K 個序列。ConvLSTM 相似於 LSTM 模

型，但是輸入轉換以及遞歸轉換都是卷積的，一般的 LSTM 輸入的是一維張量，但我們的輸入是三維的空間網格(spatial grid)的資料格式 (samples, time steps, features)。而我們在真正進行計算時，因為輸入的是圖片，而一般 CNN 的資料格式為 4 維張量(samples, channels, rows, columns)， 輸入 ConvLSTM 的資料因為是一組隨時間變化的圖片，因此將會是一個 5 維張量(samples, time steps, channels, rows, cols)。

$$\tilde{\mathcal{X}}_{t+1},\ldots,\tilde{\mathcal{X}}_{t+K} = \underset{\mathcal{X}_{t+1},\ldots,\mathcal{X}_{t+K}}{\arg\max}\ p(\mathcal{X}_{t+1},\ldots,\mathcal{X}_{t+K} \mid \hat{\mathcal{X}}_{t-J+1},\hat{\mathcal{X}}_{t-J+2},\ldots,\hat{\mathcal{X}}_{t})$$

圖 3.5 預測問題表示法(Xingjian et al., 2015)

圖 3.6 為一個 ConvLSTM cell 的數學表示，中 i_t 代表輸入門(input gate layer)，這個階段接收前一個資料所傳來的特徵輸入到 cell 中；f_t 代表遺忘門(forget gate layer)選擇性忘記不重要的資訊，控制前一個資料 C_{t-1} 哪些資訊該被遺忘，經過前兩個門則會得到可傳給下一個狀態的 C_t。o_t 代表輸出門(output gate layer)決定哪些資訊會成為輸出，最後通過一個 tanh 激勵函數成為 H_t 輸出，$X_1,\ldots,X_t$ 代表輸入、$C_1,\ldots,C_t$ 為輸出，W 代表權重，b 代表偏值，”*”代表卷積運算子，”。” 代表阿達瑪乘積(Hadamard product)使矩陣運算後依然維持原本的大小，圖 3.7 即是將 ConvLSTM 數學式轉化為圖形。本研究以過去的 15 分鐘(3 個時步)來預測未來 15 分鐘後(3 個時步)的運量，在資料預測的階段會視預測效果優劣，對於預測目標進行調整。ConvLSTM 相似於 LSTM 模型，但是輸入轉換以及遞歸轉換都是卷積的。

$$\begin{aligned}
i_t &= \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i)\\
f_t &= \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{ci} \circ C_{t-1} + b_f)\\
C_t &= f_t \circ C_{t-1} + i_t \circ tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c)\\
o_t &= \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \circ C_t + b_o)\\
H_t &= o_t \circ tanh(C_t)
\end{aligned}$$

圖 3.6 ConvLSTM 演算法 (Xingjian et al., 2015)

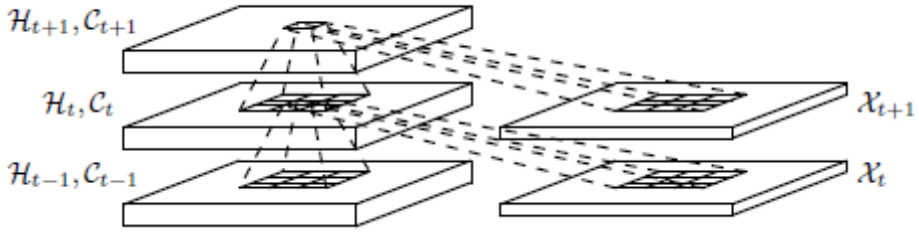


圖 3.7 ConvLSTM 模型 (Xingjian et al., 2015)

模型的輸入格式為一個 5 維向量 (samples, time, channels, rows, cols)，分別為樣本數、時間步長(timesteps)、通道數、行、列；輸出若設定為「以序列輸出(return_sequence)」，則此時的輸出也是 5 維的(samples, time, filters, output_row, output_col)。損失函數(loss function)採用均方誤差(Mean Square Error, MSE)，由於運量預測為迴歸分析，MSE 的好處在於求解時能夠得到一個穩定的結果，也是在迴歸問題中較常被使用的損失函數。優化器(Optimizer)選擇 Adam，它有對於學習率(learning rate)自行調整的特色，加上 Adam 有做參數的「偏離校正」，讓每次的學習率都會有確定的範圍，使模型參數的更新較為平穩。

圖 3.8 為本次研究所使用的模型，輸入層使用一層 ConvLSTM 接收資料，隱藏層使用三層 ConvLSTM，每個 ConvLSTM cell 中採用 20 個濾波器來提取圖形特徵，在輸出層採用一層 Flatten 層加上一層全連接層(Dense)的組合，Flatten 層將模型的輸出壓縮為 1 維的矩陣，而 Dense 層透過倒傳遞(back propagation)來達

到學習模型特徵，在此設定 108 個神經元，代表輸出的 108 個站的運量值。

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lst_m2d (ConvLSTM2D)    (None, 3, 20, 44, 44)     6800
_________________________________________________________________
batch_normalization (BatchNo (None, 3, 20, 44, 44)     176
_________________________________________________________________
conv_lst_m2d_1 (ConvLSTM2D)  (None, 3, 20, 44, 44)     12880
_________________________________________________________________
batch_normalization_1 (Batch (None, 3, 20, 44, 44)     176
_________________________________________________________________
conv_lst_m2d_2 (ConvLSTM2D)  (None, 3, 20, 44, 44)     12880
_________________________________________________________________
batch_normalization_2 (Batch (None, 3, 20, 44, 44)     176
_________________________________________________________________
conv_lst_m2d_3 (ConvLSTM2D)  (None, 3, 20, 44, 44)     12880
_________________________________________________________________
batch_normalization_3 (Batch (None, 3, 20, 44, 44)     176
_________________________________________________________________
flatten (Flatten)            (None, 116160)            0
_________________________________________________________________
dense (Dense)                (None, 108)               12545388
=================================================================
Total params: 12,591,532
Trainable params: 12,591,180
Non-trainable params: 352
_________________________________________________________________
```

圖 3.8 本研究預測模型結構圖

3.5 預測結果驗證

為了驗證此模型預測結果是否正確，，使用交叉驗證與均方根誤差，這些方法都被廣泛的被使用在模型的驗證當中。

(1). K-折交叉驗證(K-fold cross validation)

假設有個未知模型具有一個或多個未知的參數，且有一個數據集能夠反映該模型的特徵屬性（訓練集）。擬合的過程是對模型的參數進行調整，以使模型儘可能反映訓練集的特徵。如果從同一個訓練樣本中選擇獨立的樣本作為驗證集合，當模型因訓練集過小或參數不合適而產生過度擬合時，驗證集的測試予以反映。交叉驗證用於防止模型過度複雜而引起的過度擬合，在統計學上將數據樣本切為較小子集的方法。而 K-折交叉驗證法是最常被使用的驗證方法，將訓練資料分為 k 份，依序將其中 1 到 K 等分作為驗證資料，而其餘 k-1 份作為訓練資料，將 k 次訓練的預測精準度(accuracy)平均，則能說此數值為此模型預測的正確率，而最常被採用的 k 值為 10，本研究也採用此數值作為分割資料的大小(圖 3.9)。

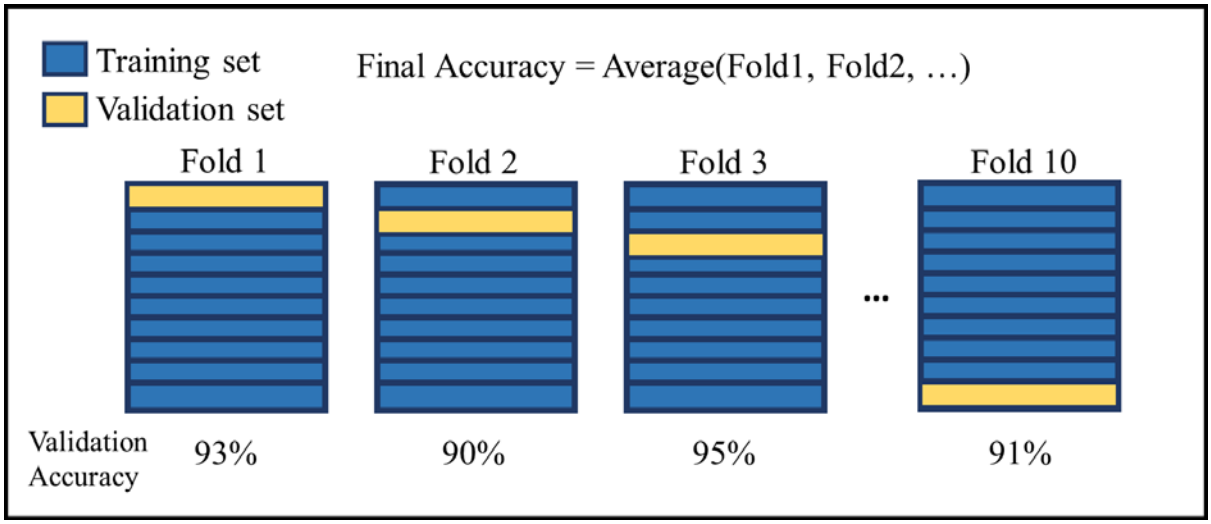


圖 3.9　K-折交叉驗證法示意圖(自行整理)

(2). 均方根誤差(Root Mean Square Error, RMSE)

均方根誤差定義為估計量($\hat{\theta}$)與實際值(θ)之差期望值(E)的平方根(公式見圖 3.9)，均方根誤差常用來量度測量數值之間的差異，其數值常為模型預測的值或是被觀察到的估計量。均方根誤差代表預測值和觀察值之差的樣本標準差（sample

standard deviation），當這些差值是以資料樣本來估計時，他們通常被稱為殘差；當這些差值不以樣本來計算時，通常被稱為預測誤差（prediction errors）。均方根誤差主要用來聚集預測中誤差的大小，在不同的時間下，以一個值來表現其預測的能力。在本篇研究中使用正規化後的均方根誤差(coefficient of variation RMSE, CV RMSE)，正規化的均方根誤差可以使得不同數值範圍的資料集更易於比較。雖然目前並沒有一個一致性的方法來正規化均方根差，但較常用平均值或是資料的範圍來正規化被量測的資料(公式見圖 3.11)。這個值常被用來指正規化的方均根偏移或誤差，同時也常常被表示成比例。當比例的值較低時，代表較少的殘差變異，在取較小的樣本的時候，樣本的範圍容易被樣本的大小影響，其準確度可能就受到影響。本研究會將算出的均方根誤差值除以該筆資料對應的 y_test 平均值，以百分比的方式呈現此數值。

$$\mathrm{RMSE}(\hat{\theta}) = \sqrt{MSE(\hat{\theta})} = \sqrt{E\left((\hat{\theta} - \theta)^2\right)}$$

圖 3.10 均方根誤差公式

$$\mathrm{CV(RMSE)} = \frac{RMSE}{\bar{y}}$$

圖 3.11 正規化均方根誤差公式

透過三個情境:A. 同一時間不同站點的預測精準度，與 B. 平日與特殊日下，同一時間不同站點的預測精準度，以及 C. 同一站點在不同時間下預測精準度比較，來觀察與比較 ConvLSTM 在不同情境下的預測精準度。

3.6 研究限制

1. 都市活動的複雜性

本研究專注於捷運運量的研究，但在都市中的交通包含更多大眾運輸工具以及自有運具，且也會有運具轉乘的狀況，因此並未考慮到其他交通所帶來的影響。

2. 運量資料代表性

無法從運量資料判斷乘客是否為活動參加者，因此無法確認該次產生的運量與活動的關聯，只能從時間來進行判斷。

3. 運量資料不完整

本次研究從臺北市交通局所申請的運量資料為悠遊卡進出站的運量統計資料，而不包含以其他方式(其他智慧卡、單程票)進出站的運量，因此無法包含完全真實的進出站情況，但悠遊卡還是主要的使用者，可以代表大部分時候的運量。

第四章 資料集製作與模型搭建

4.1 Pandas 資料預處理

這個部分主要使用 Python 語言處理大型活動資料的爬蟲，運量資料的處理，以及訓練資料集的製作。

1. 大型活動日資料處理

在臺北具指標性且有完整紀錄活動的場館只有臺北小巨蛋，同時他們也將過去的活動紀錄公布在官方網站，因為本研究的目的之一是希望預測模型可以預測特殊活動的人潮，因此我們必須在運量資料中加入「該日是否有大型活動舉辦」的特徵值，因此利用 Pandas 讀取網頁的功能將網頁資料讀取到 Jupyter Notebook 內(圖 4.1)。從 2016 年到 2019 年 7 月 1277 天中共有 349 天為活動日，為了避免模型無法捕捉特殊日特徵，利用人事行政總處的「中華民國政府行政機關辦公日曆表」，加入了國定假日以及周休二日增加特殊日期的樣本數，透過將兩者資料進行聯集，最後得到 520 天特殊日。

	編號	標題	發布日期
0	1	2019/11/10~11/17《2019台北海碩網球公開賽》	108-10-08
1	2	2019/11/02《仙角百老匯8-人生開麥拉》	108-10-02
2	3	2019/10/20《2019台新銀行貴賓之夜》	108-09-27
3	4	2019/11/23《八三夭 想見你 想見你 想見你：重返小巨蛋演唱會》	108-09-21
4	5	2019/12/07~12/08《絕色其文蔚 世界巡迴演唱會 巔峰之旅 台北站》	108-09-19
...	...	...	...
564	565	郭富城舞林正傳世界巡迴演唱會2009臺灣站 2	102-03-01
565	566	新好男孩2010台北演唱	102-03-01
566	567	蕭敬騰 洛克先生 Mr.ROCK 演唱會	102-03-01
567	568	縱貫線 Super Band 世界巡迴演唱會 [終點站]	102-03-01
568	569	鴻海科技集團年終聯歡會	102-03-01

569 rows × 3 columns

圖 4.1 臺北小巨蛋活動紀錄

因為發布日期並非為該場活動舉辦日期，以及編號無法提供有效的訊息，因此將此兩個欄位刪除。觀察活動名稱可以發現，活動日期後面都接著「《」的符號，因此透過 split 的功能，將欄位分為日期與活動名稱(圖 4.2)，因為在後續的處理

中，必須依照日期「逐一」將資料標記是否為活動舉辦日，因此在日期欄位中可以發現日期的格式以不同的格式呈現，因為格式較複雜，這邊將資料儲存後讀取到 excel 進行處理。

	日期	活動名稱
11	2019/09/03、09/04、09/05、09/06、09/07、09/08	2019YONEX中華台北羽球公開賽》
30	2019/07/20、07/21、07/23、07/24、07/25、07/26、07/27...	百老匯音樂劇-獅子王》
33	2019/04/13、04/14	薛之謙"摩天大樓"世界巡迴演唱會—臺北站》
34	2019/06/08、06/09	王力宏 龍的傳人2060 世界巡迴演唱會》
41	2019/02/14、02/15、02/16、02/17	JJ林俊傑聖所世界巡迴演唱會 台北站》
45	2019/02/01、02/02	Mr.Children Tour 2018-19 重力與呼吸 Live in Taiwan》
48	2018/12/14、12/15	張韶涵2018《旅程》世界巡迴演唱會》
49	2018/12/29、12/30	林宥嘉 ido世界巡迴演唱會 台北初登場》
50	2018/11/11、11/12、11/13、11/14、11/15、11/16、11/17...	2018台北海碩網球公開賽》
51	2018/10/27、10/28	2018年第七屆世界盃太極拳錦標賽》
52	2019/05/15、05/17、05/18、05/19	費玉清2019台北小巨蛋演唱會》
53	2019/02/08、02/09	費玉清2019台北小巨蛋演唱會》
54	2019/04/26、04/27、04/28	盧廣仲 11週年 台北小巨蛋演唱會 大人中》
58	2018/12/01、12/02	2018徐佳瑩小巨蛋演唱會》
63	2018/10/02、10/03、10/04、10/05、10/06、10/07	2018中華臺北羽球公開賽》
65	2018/09/22、09/23	關西傑尼斯8台北演唱會》

圖 4.2 活動欄位拆分

在 excel 中透過篩選的功能修改日期資料，並且依照活動舉辦的天數來將活動所舉辦的日期一一標註，處理結果如圖 4.3。

	活動日期	活動名稱
0	2019-12-08	絕色其文蔚世界巡迴演唱會巔峰之旅台北站
1	2019-12-07	絕色其文蔚世界巡迴演唱會巔峰之旅台北站
2	2019-11-30	蕭煌奇2019「人生劇場放映中」小巨蛋演唱會
3	2019-11-23	八三夭想見你想見你想見你：重返小巨蛋演唱會
4	2019-11-17	2019台北海碩網球公開賽
5	2019-11-16	2019台北海碩網球公開賽
6	2019-11-15	2019台北海碩網球公開賽
7	2019-11-14	2019台北海碩網球公開賽
8	2019-11-13	2019台北海碩網球公開賽
9	2019-11-12	2019台北海碩網球公開賽

圖 4.3 小巨蛋活動日期整理

從網站下載中華民國政府行政機關辦公日曆表後，先透過 Excel 選取目標時間，讀入 Jupyter Notebook 後可以看到該表以 1 和 0 來做為分別是否放假的表示

方式，並在後面標示假期的描述(見圖 4.4)，以此表與小巨蛋的活動表透過 merge 的方式進行合併，參數選擇 outer 就可進行聯集，最後得到 520 日特殊活動日。

```
special_event = pd.merge(event,holiday,how='outer',on='活動日期
',sort=True)
```

	活動日期	isHoliday	holidayCategory
0	2016-01-01	1	放假之紀念日及節日
1	2016-01-02	1	星期六、星期日
2	2016-01-03	1	星期六、星期日
3	2016-01-09	1	星期六、星期日
4	2016-01-10	1	星期六、星期日
...	...	...	...
417	2019-06-16	1	星期六、星期日
418	2019-06-22	1	星期六、星期日
419	2019-06-23	1	星期六、星期日
420	2019-06-29	1	星期六、星期日
421	2019-06-30	1	星期六、星期日

422 rows × 3 columns

圖 4.4 週休二日與國定假日處理

2. 運量資料整理

Pandas 是 python 的一個數據分析套件，2009 年底透過開源(open source)的方式分享，提供高效能、簡易使用的資料格式(Data Frame)讓使用者可以快速操作及分析資料。在資料處理的階段都會透過 Pandas 套件來進行相關的計算與整理。

將運量資料與場站代號讀入後，從運量資料中利用 grouupby 以場站代號欄位進行分類並用 count 功能將場站代號以外的欄位進行計數(圖 4.5)，發現運量紀錄中一些場站包含過少筆數的進出站紀錄，後與交通局提供的捷運場站代碼表對照發現有多餘的場站代碼如圖 4.6 與圖 4.7，刪除錯誤的代碼後剩下 117 站，與代碼表相符。

	資料蒐集時間	場站代號	進站人數	出站人數	人數總和
0	2016-01-01 01:05:00	55	16	7	23
1	2016-01-01 01:05:00	101	1	0	1
2	2016-01-01 01:05:00	24	118	62	180
3	2016-01-01 01:50:00	43	29	6	35
4	2016-01-01 01:50:00	51	1	0	1
...	...	...	...	...	...
36763042	2019-09-29 23:55:00	30	8	11	19
36763043	2019-09-29 23:55:00	83	18	55	73
36763044	2019-09-29 23:55:00	22	11	12	23
36763045	2019-09-29 23:55:00	130	68	94	162
36763046	2019-09-29 23:55:00	90	0	1	1

36763047 rows × 5 columns

	捷運站名稱	捷運站代碼	緯度	經度
0	大安	11	25.032943	121.543551
1	大安	102	25.032943	121.543551
2	中山	53	25.052685	121.520392
3	中山	106	25.052685	121.520392
4	台北車站	51	25.046255	121.517532
...	...	...	...	...
112	龍山寺	85	25.035280	121.499826
113	雙連	54	25.057805	121.520627
114	關渡	68	25.125633	121.467102
115	蘆洲	174	25.091554	121.464471
116	麟光	14	25.018535	121.558791

117 rows × 4 columns

圖 4.5 讀取運量與場站資料

資料來源:作者自行整理

```
mrt_count = mrt4.groupby("場站代號")
b=list(set(mrt4['場站代號'])-set(col_name['捷運站代碼']))
```

場站代號				
0	2	2	2	2
1	4	4	4	4
3	1	1	1	1
7	305426	305426	305426	305426
8	311678	311678	311678	311678
...	...	...	...	...
196	327	327	327	327
209	2	2	2	2
210	19	19	19	19
253	1	1	1	1
255	82	82	82	82

```
In [10]: b=list(set(mrt4['場站代號'])-set(col_name['捷運站代碼']))
         b=pd.Series(b)
         b

Out[10]: 0       0
         1       1
         2     193
         3       3
         4     195
         5     196
         6     255
         7     209
         8     210
         9     118
         10     87
         11    253
         12    191
         dtype: int64
```

圖 4.6 以場站代號進行 groupby(左)　　圖 4.7 多餘的場站代號(右)

資料來源:作者自行整理

對照場站代碼表可以發現有站名重複但場站代碼不同的車站(大安、中山、臺北車站、民權西路、忠孝復興、忠孝新生、松江南京、南京復興、南港展覽館)，因為這些車站為路線轉乘站，在路線設計上給予不同的代號代表不同路線，但在進行資料處理時若一站多號會造成該站權重錯誤，因此針對重複的代碼一律替代為較小的數字，並透過 groupby 將重複站的運量進行合併，最後將場站中文名稱與原本的運量 DataFrame 透過 for 迴圈進行接合(圖 4.8)。

使用replace取代值

站名重複但代號不重複的站有9站，大安、中山、台北車站、民權西路、忠孝復興、忠孝新生、松江南京、南京復興、南港展覽館

合併重複代碼

```
In [5]: record['場站代號'].replace({102:11,106:53,52:51,129:55,90:10,133:89,132:107,108:9,98:31},inplace = True)
        record.head()
```

Out[5]:

	資料蒐集時間	場站代號	進站人數	出站人數	人數總和	站名
0	2016-01-01 01:05:00	55	16	7	23	民權西路
1	2016-01-01 01:05:00	101	1	0	1	信義安和
2	2016-01-01 01:05:00	24	118	62	180	港墘
3	2016-01-01 01:50:00	43	29	6	35	小南門
4	2016-01-01 01:50:00	51	1	0	1	台北車站

圖 4.8 重複站名處理

在資料預處理中還需要檢查資料是否有遺失或是缺少，透過搜尋進(出)站人數發現，在此份運量資料中僅記錄進站或出站人數大於 1 的時間區段，也就是說當該站在某時間區間內無人進出，則無該站紀錄。為了建立完整的時間序列，必須透過產生 0 進出量的完整時間序列，與原運量資料進行聯集，方可填補缺失時間區段。

首先利用函式建立一連串時間序列還原的方法，透過站名選取運量資料後，將每站的資料與預先建立的從 2016/01/01 00:00:00 到 2019/09/30 00:00:00 每五分鐘一筆的時間資料，透過 merge 的方法取得兩者時間的聯集，還原了資料的時間序列，合併後會保留原先已有的運量資料，而新增的運量資料會以 NaN 表示，則以 0 填入此遺失值，並加回該站在合併過程中遺失的站名與場站代號，則此函式就完成了一站的處理，後透過 for 迴圈輸入每個站的站名，函式接收到站名後就開始執行單站的資料處理並輸出，將最後所有的成果與一個空的 DataFrame 使用 append 方法合併，即完成此步驟(圖 4.9)。

```
def merger(sta):
    #與完整時間進行聯集
    b = record_groupby[record_groupby['站名']==sta]
    merge_time = pd.merge(time_sep2,b[['資料蒐集時間','進站人數','出站人數','人數總和']],how='outer')
    merge_time = merge_time.fillna(0)
    #站名、代號復原
    sta_count = b.groupby(["站名","場站代號"],as_index = False).count()
    merge_time['站名']=float("NaN")
    merge_time['場站代號']=float("NaN")
    merge_time['站名'].fillna(value=sta_count['站名'][0],inplace=True)
    merge_time['場站代號'].fillna(value=sta_count['場站代號'][0],inplace=True)
    return merge_time
#迴圈建立每個站的完整時間
for i in range(len(station_name)):
    merge_time = merger(station_name[i])
    a = pd.concat([a,merge_time])
```

	資料蒐集時間	進站人數	出站人數	人數總和	站名	場站代號
0	2016-01-01 00:00:00	0.0	0.0	0.0	七張	35.0
1	2016-01-01 00:05:00	0.0	0.0	0.0	七張	35.0
2	2016-01-01 00:10:00	0.0	0.0	0.0	七張	35.0
3	2016-01-01 00:15:00	0.0	0.0	0.0	七張	35.0
4	2016-01-01 00:20:00	0.0	0.0	0.0	七張	35.0
...	...	...	...	...	...	...
42550375	2019-09-29 23:40:00	87.0	26.0	113.0	龍山寺	85.0
42550376	2019-09-29 23:45:00	204.0	145.0	349.0	龍山寺	85.0
42550377	2019-09-29 23:50:00	146.0	137.0	283.0	龍山寺	85.0
42550378	2019-09-29 23:55:00	20.0	10.0	30.0	龍山寺	85.0
42550379	2019-09-30 00:00:00	0.0	0.0	0.0	龍山寺	85.0

42550380 rows × 6 columns

圖 4.9 建立運量資料完整時間序列

如 3.3.1 所述由於本次研究需要，在建構完整時間序列後需要選取早上 6 點至晚上 24 點之間的運量資料，因此必須透過迴圈將資料刪除。首先使用比較運算符篩選出 2016/01/01 早上 6 時至 2019/07/01 晚上 24 時期間內的資料，將時間資料設定為資料的索引(index)，這個步驟是為了之後要能選擇出目標時間區段。之後建立一函式，與前面的方法相同，此函式皆針對單站資料進行處理，後以 for 迴圈輸入每站的站名來使函式運行，最後將每站的運行結果合併為處理完成的資料。在擷取特定時間的時候，主要使用了 between_time 的功能，將資料蒐集時間欄位作為 DataFrame 的索引(index)，選取出指定的起始時間(早上 6 點)與結束時間(晚間 12 點)，最後就得到裁切完成的運量資料。在處理完成的資料中，預期將有每站一小時有 12 筆五分鐘的資料，一天採計 18 小時，加上 2019-07-01 0 時的一筆資料，108 個站共 1277 天的資料，(12*18+1)*108*1277=29927772 筆資料(圖 4.10)。

```
#選擇日期
mask1 = merge_time['資料蒐集時間']<='2019-07-01 00:00:00'
mask2 = merge_time['資料蒐集時間']>'2016-01-01 00:00:00'
merge_time=merge_time[(mask1 & mask2)]
merge_time.index=merge_time['資料蒐集時間']
#建立單站時間選擇函式
def time_selection(station):
    single_sta=merge_time[merge_time['站名']==station]
    single_sta['資料蒐集時間'] = single_sta[['資料蒐集時間
']].between_time(start_time='06:00:00', end_time='00:00:00')
    single_sta = single_sta.dropna()
    qwer = single_sta.reset_index(drop=True)
    return qwer
#迴圈完成每個站的時間選擇
for i in range(len(station_name)):
    time_slec = time_selection(station_name[i])
    b = pd.concat([b,time_slec])
```

	資料蒐集時間	進站人數	出站人數	人數總和	站名	場站代號
0	2016-01-01 06:00:00	0	23	23	松山機場	7
1	2016-01-01 06:00:00	96	318	414	中山國中	8
2	2016-01-01 06:00:00	841	456	1297	南京復興	9
3	2016-01-01 06:00:00	356	267	623	忠孝復興	10
4	2016-01-01 06:00:00	303	277	580	大安	11
...	...	...	...	...	...	...
29927767	2019-07-01 00:00:00	25	156	181	徐匯中學	176
29927768	2019-07-01 00:00:00	135	322	457	三和國中	177
29927769	2019-07-01 00:00:00	103	88	191	三重國小	178
29927770	2019-07-01 00:00:00	59	84	143	迴龍	179
29927771	2019-07-01 00:00:00	61	116	177	丹鳳	180

29927772 rows × 6 columns

圖 4.10 運量資料時間處理

3. 製作訓練資料集

在前一小節已經將運量資料整理完成，在輸入到模型計算以前，必須將資料整理為模型接受的樣態，而根據前面所述本研究所使用的 ConvLSTM 模型接受的為 5 維的形狀，本次模型使用的輸入方式為 channel_first 的方式，會將特徵值(feature)放在行列資料前面，因此必須將資料整理為(sample,timesteps,feature,rows,cols)的形狀，在製作單筆資料的時候必須決定(timesteps, feature, rows, cols)，如 3.4 所述本研究預計使用過去 15 分鐘的資料來預測未來 15 分鐘的運量。本研究採用輸入五維矩陣來預測 108 個站 15 分鐘的平均運量(一維)，因為若採用原本的模型方法是以五維圖片去預測五維圖片，預測出的結果會因為 softmax 激勵函數而將預測出的數值壓縮到 0 到 1 之間的小數，得到的結果為與原圖片相似的圖片而非捷運站位置的數值，因此以此方法預測無法得到直觀的運量預測結果。採用以五維預測一維的方式，再依照預測出的結果將預測出的 108 個數值重新輸入到空的矩陣當中還原運量圖。

在 3.4.1 提到，本次的研究方法是將現實的空間映射至虛擬的網格之上，並在捷運站在網格上的映射位置加入該捷運站的特徵值作為圖片輸入模型。若選取相較

於捷運站太大的範圍作為模型輸入會降低訊息的解析度，因此必須根據捷運站所在的經緯度來決定範圍。在 Jupyter Notebook 中讀入 TWD97 的臺灣鄉鎮市區圖，以及將捷運站轉為 TWD97 的格式繪製點資料(圖 4.11)。

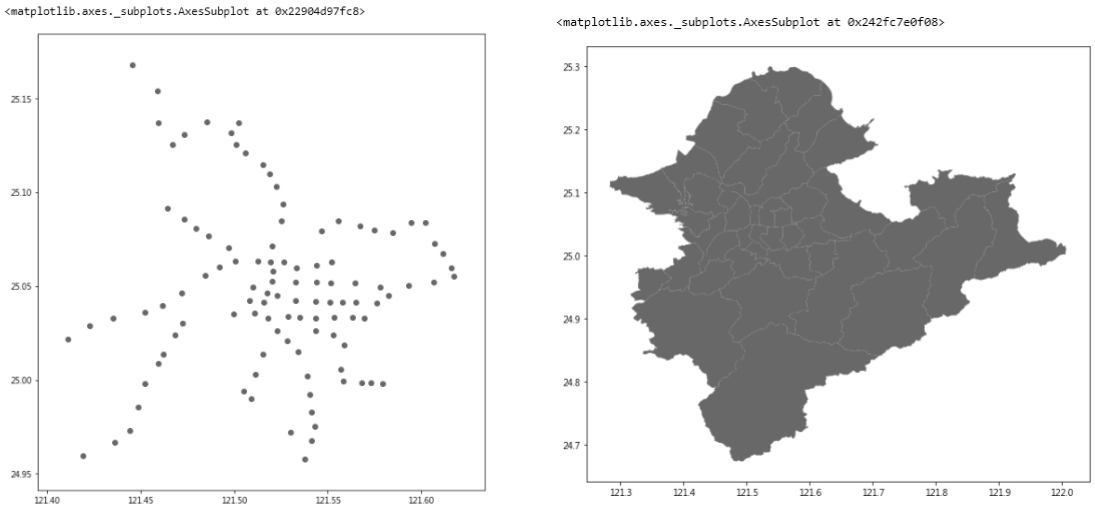


圖 4.11 匯入空間圖資

將圖資匯入後，將資料進行套疊，並使用 within 的功能，透過迴圈判斷點資料與是否在面資料的範圍中，若未包含捷運站的區域則被刪除，最後剩下包含捷運站的區域，圖 4.12 中灰色區域為包含捷運站的區，黑點則為捷運站。

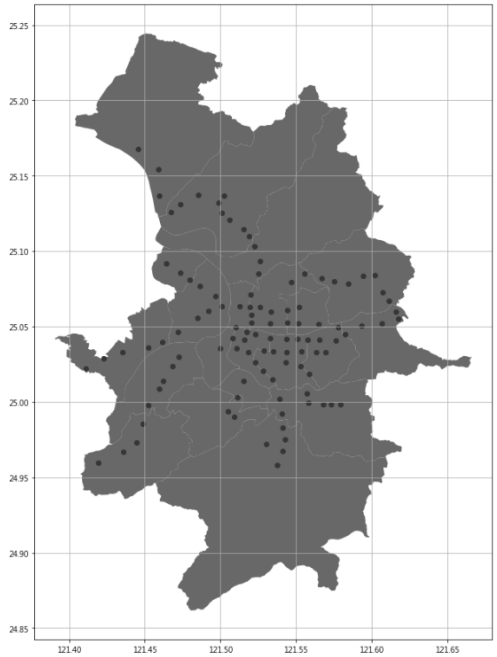


圖 4.12 捷運站分布圖

了解了捷運站空間上的分布後，在地圖上繪製網格，根據各捷運站的位置來調整網格密度，來使每一個捷運站都能在一個格子中，後選擇以 0.005°(約 500m)作為網格間隔*，取出一個 44*44 (22*22km^2)大小的網格，若兩站在同一網格中或是該網格位置偏移，則選擇距離該捷運站最近的網格移至網格內，而在資料中南港展覽館(121.6175001°, 25.055288°)與南港軟體園區(121.615953°, 25.059905°)以及南勢角(121.509237°, 24.990045°)與景安站(121.505113°, 24.993905°)，因為站距過近而在進行 spatial join 時無法指定到不同的矩陣位置，為了將場站分開，將南港展覽館(121.6175001°, 25.0525°)與南勢角站(121.5075°,24.9875°)經緯座標調整 (圖 4.9)。

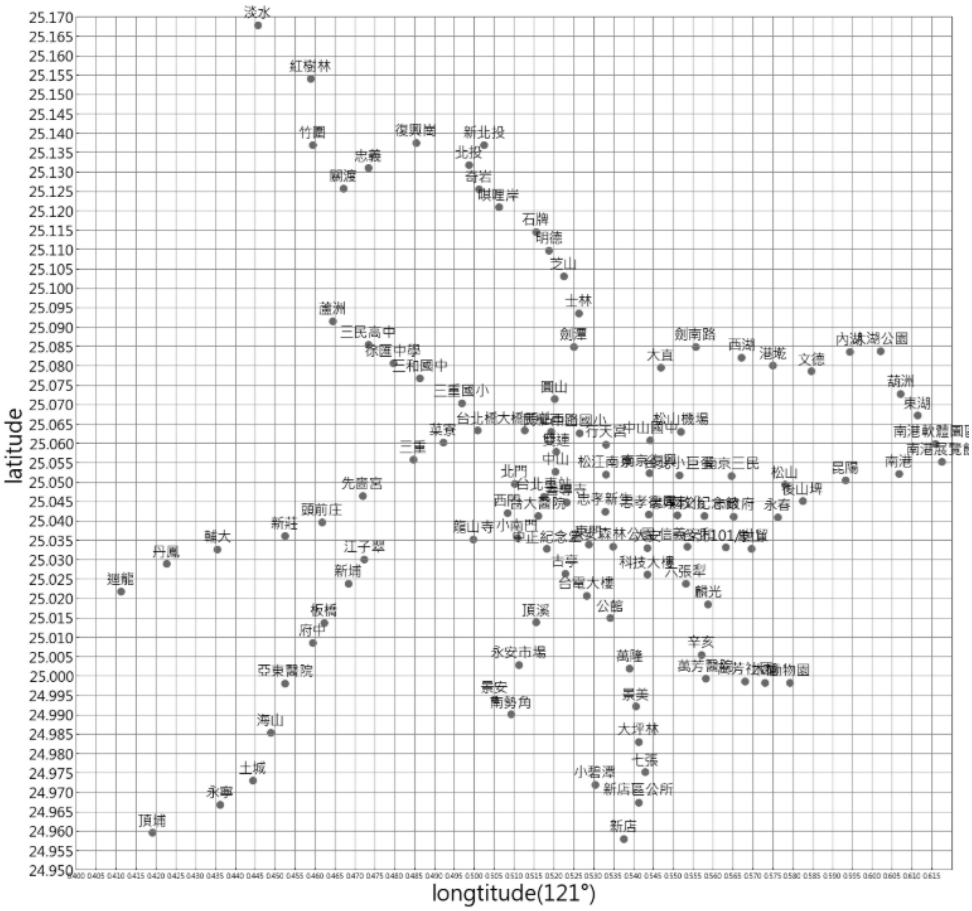


圖 4.13 捷運站網格套疊圖

*註:雖然經緯度長度不相同，但各取相同的經緯度長度只是為了讓網格為正方形，方便資料的輸入，捷運站位置的偏移不會影響到預測的結果

在製作訓練資料集時，實際輸入的其實是矩陣(matrix)，因此我們需要將前面處理好的 5 分鐘運量資料，輸入到一個形狀 44*44 的矩陣中，會得到(277109,44,44)的三維矩陣，這裡的每筆資料為五分鐘的運量矩陣，取前三年資料進行資料集製作，將三筆矩陣利用迴圈相加得到(79295, 3, 44, 44)的矩陣，最後加上代表運量的一維特徵，將此矩陣轉為五維矩陣(79295, 3, 1, 44, 44)即完成輸入資料製作。

在實作的部分，首先利用迴圈建立一個 geodataframe，內容為每 0.005° 為間隔 44*44 的網格，每個格子都對應到真實世界相對應的座標。接著將每個站的經緯度座標加入每筆運量資料，並將運量資料從 dataframe 轉換為 geodataframe，並與網格進行 Spatial join，會得到每個站在網格中對應的 index，根據這些 index 就可以使用迴圈建立每五分鐘的運量圖，將同時刻每站的運量資料加入一個 1936 維的 0 矩陣中，再 reshape 為 44*44 的矩陣，原本共有 29927772 筆運量資料，轉換為矩陣後為 277109 筆資料。下圖為將運量矩陣視覺化的結果，圖中顏色越白代表運量越高，而黑色的地方數值為 0(圖 4.10)。

```
#劃出邊界
xmin,ymin,xmax,ymax = (121.4,24.95,121.625,25.17)
width, height = (44,44)
rows = np.arange(ymin,ymax,0.005)
cols = np.arange(xmin,xmax,0.005)
geomatrix = []
for y in rows:
    for x in cols:
        coord = [x,y]
        geomatrix.append(coord)
geomatrix = np.array(geomatrix).reshape(45,45,2)
#指定經緯度到每個方格
matrix = []
polygons = []
for j,r in enumerate(geomatrix):
    if j > 43:
        break
    for i,c in enumerate(r):
        if i > 43:
            continue
        else:
            matrix.append([geomatrix[j+1][i],geomatrix[j+1][i+1]
,r[i+1],c])
            polygons.append(Polygon([geomatrix[j+1][i],geomatrix
[j+1][i+1],r[i+1],c]))
#轉為帶有地理資訊的 Dataframe(geoDataframe)
crs = {'init': 'epsg:4326'}
grid = gpd.GeoDataFrame(matrix,crs = crs,geometry=polygons)
grid.columns = ['p1','p2','p3','p4','geometry']
```

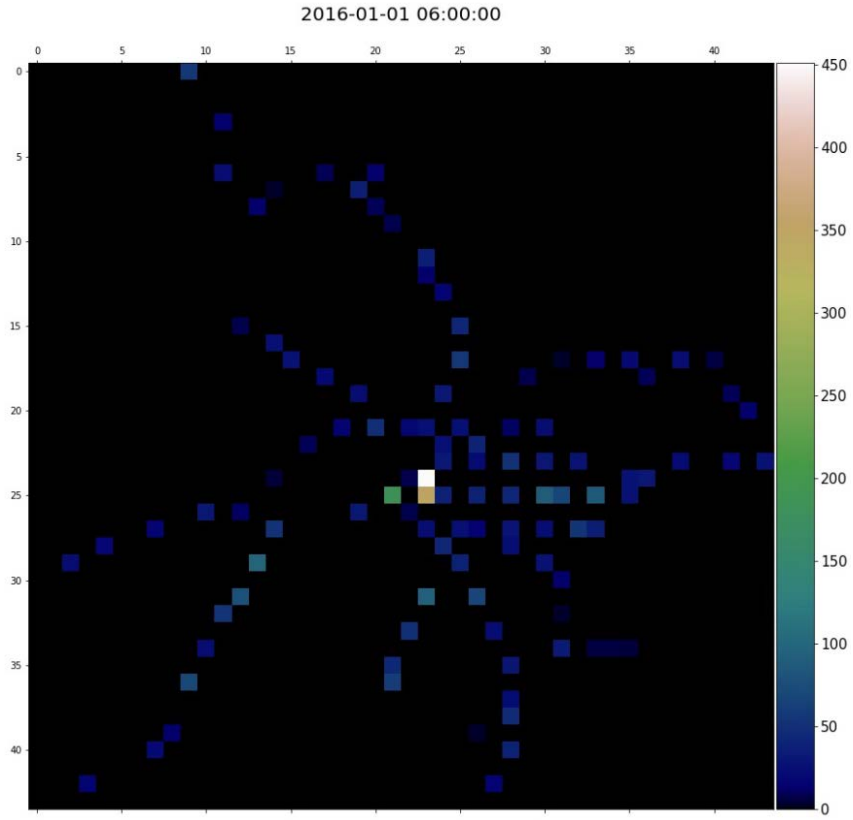


圖 4.14 運量矩陣視覺化

4. 加入日型態特徵值

為了讓模型能更好的分辨出平日與特殊日的運量，在矩陣內值為 0 的位置加入一個固定的數值，並以不同的形狀來表示兩種日型態的差別，希望在資料輸入模型時能最早被模型分開，分別在計算時能根據特徵值去預測不同日型態的運量。在圖 4.16 中可以看到分別為左側的 2016 年 1 月 1 日 6 點與右側 2016 年 1 月 5 日 6 點的矩陣圖，因為沒有捷運站而選擇右上角作為特徵值加入的地方，為了讓特徵值能被模型當作特徵而非雜訊，其值必須大於某個值才會被模型採納，因此計算平日與特殊日的運量 75 百分位數，分別為 215 與 200，因此選擇整數 200 作為輸入的值。選擇一個 4*4 的範圍，在左上與右下填入三角形的特徵值，分別

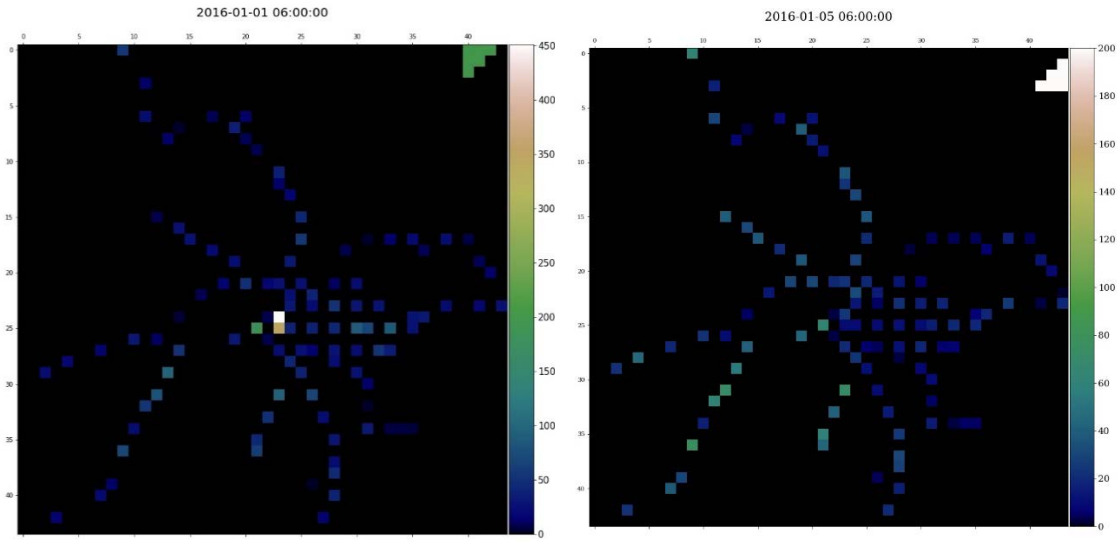


圖 4.16 加入日型態特徵

4.2 搭建訓練模型

由於要進行的是多維矩陣的運算，需要強大的硬體設備才能加速運算的進行，因此使用 Google 下的 Google Colaboratory 來進行模型的搭建與訓練。Google Colaboratory(簡稱 Colab)是一個基於 Jupyter Notebook 的雲端開發環境，透過雲端可以將資料在 Google 的虛擬機上進行運算，此平台最大的特色在於提供了免費且強大的 GPU 資源讓使用者進行 GPU 加速運算，並且能連結 Google Drive 來存放/開啟資料，對於使用者來說相當具有便利性與可近性。

在模型結構上，使用了一層 ConvLSTM 作為輸入層，並在每層 ConvLSTM 層後加上批量標準化層(Batch Normalization)，將每個 batch 在前一層的輸出重新標準化，使其輸出的平均值為 0，標準差接近 1，其優點在於能增加學習率、不會對預設值過度反應以及控制過度學習(減少 dropout 層的使用)，模型隱藏層為三層 ConvLSTM 與 Batch Normalization 來達到深度學習的模型架構，最後連結到變平層(Flatten)將預測結果壓縮為一維，最後連接全連結層(Dense)並以 Adam 作為優化器，Adam 可以對於梯度的速度方向以及學習率(learning rate)進行調整，使模型參數的更新較為平穩，輸出層連結前面模型的輸出並加入非線性特徵，將結果進行分類。

```
model4=tf.keras.Sequential()
######################InputLayer
model4.add(tf.keras.layers.ConvLSTM2D(filters=20,kernel_size=(3,
3),input_shape=(3,1,44,44),data_format='channels_first',recurren
t_activation='hard_sigmoid',activation='tanh',padding='same',ret
urn_sequences=True))
model4.add(tf.keras.layers.BatchNormalization())
###################### HiddenLayer
model4.add(tf.keras.layers.ConvLSTM2D(filters=20,kernel_size=(3,
3),data_format='channels_first',recurrent_activation='hard_sigmo
id',activation='tanh',padding='same',return_sequences=True))
model4.add(tf.keras.layers.BatchNormalization())

model4.add(tf.keras.layers.ConvLSTM2D(filters=20,kernel_size=(3,
3),data_format='channels_first',recurrent_activation='hard_sigmo
id',activation='tanh',padding='same',return_sequences=True))
model4.add(tf.keras.layers.BatchNormalization())

model4.add(tf.keras.layers.ConvLSTM2D(filters=20,kernel_size=(3,
3),data_format='channels_first',recurrent_activation='hard_sigmo
id',activation='tanh',padding='same',return_sequences=True))
model4.add(tf.keras.layers.BatchNormalization())
######################OutputLayer
model4.add(tf.keras.layers.Flatten(data_format='channels_first')
)
model4.add(tf.keras.layers.Dense(108))
####################
opt=tf.keras.optimizers.Adam(lr=0.01)
model4.compile(loss='mse',metrics=['accuracy','mse'],optimizer=o
pt)
####################
early_stopping=EarlyStopping(monitor='val_loss',mode='min',verbo
se=1)
model4.fit(x_train4,y_train4,batch_size=100,epochs=10,validation
_split=0.3,verbose=1,callbacks=[early_stopping])
model4.fit(x_train4,y_train4,batch_size=100,epochs=10,validation
_data=(x_test4,y_test4),verbose=1,callbacks=[early_stopping])
```

模型預測將使用加入與未加入日型態特徵的運量資料進行預測，皆使用 15 分鐘運量資料集做為預測對象，因為捷運離峰班距為 8~12 分鐘，為了能包含此時間區間且兼顧 Colab 所能容納的資料大小，模型預測將主要以 3 個時步進行，在模型預測使用交叉驗證，將 10 次的預測結果之平均作為模型最後的預測成果，預測結束後將進行模型的比較與驗證，首先比較特徵值的有無對於模型預測的正確性，接著選擇預測表現較好的模型與 30 分鐘、60 分鐘之預測進行比較，最後針對 15 分鐘的預測模型進行交叉驗證與均方根誤差的驗證。

第五章 研究成果與討論

本研究的目的希望透過深度學習模型 ConvLSTM 來預測大型活動下的臺北捷運運量，讓模型具有預測大型活動運量的能力。本章將介紹使用 ConvLSTM 預測的過程，並對預測的結果進行分析，對於模型在不同情境下的預測能力進行驗證與討論。

1. ConvLSTM 運量預測

本研究採用全站運量對 ConvLSTM 模型進行輸入，以前三個時步(十五分鐘)的運量預測後三個時步的平均運量作為基本的預測模型，使用不同輸入資料與不同時長來觀察模型在不同情況下的預測能力。此小節以製作 3 個時步的資料集過程進行說明。ConvLSTM 透過過去 n 筆的圖片產生 n 筆的預測圖片，但本研究目標是要預測出特定位置上的數值，因此採用一維各站運量作為模型輸出，為了維持模型以過去 n 筆資料來預測未來 n 筆資料的規則，就選擇以未來運量的平均做為預測的輸出。

1. 預測任務

本研究將進行有無日型態特徵值預測、不同時步預測(15 分鐘、30 分鐘、60 分鐘)以及特殊活動日預測，並且透過交叉驗證、均方根誤差以及實際活動運量比較來驗證 ConvLSTM 是否能夠對於活動的運量成功預測。

2. 輸入資料集製作

在第四章將資料輸入網格後加入了特徵值，完成五分鐘一筆的運量矩陣，而因為預測任務的需要必須製作出不同時步的資料集，以下以製作 3 個時步的資料集為例進行說明。此部分製作:有/無日型態特徵之 15 分鐘運量資料集、30 分運量特徵資料集、60 分運量特徵資料集，共四組資料。

資料集需要包含輸入(x)與輸出(y)，也就是每筆輸入都要對應一個輸出。運量矩陣共有 277109 筆，因為要取前 3 筆製作輸入(x)以及取後 3 筆製作輸出(y)，在保證輸入與輸出資料及長度相同的情況下，選擇從第 1 筆取到第 277105 筆來製

作輸入資料集，得到輸入為(92368, 3, 44, 44)的 4 維矩陣。此時的矩陣與 ConvLSTM 所接受的輸入格式(sample, timesteps, feature, rows, cols)還差了特徵值，而在矩陣中的特徵值只有一個一維的運量值，因此可以直接將輸入矩陣的形狀重塑為(92368, 3, 1, 44, 44)，在全部三年半的輸入資料中截取三年的資料共 79275 筆。輸出資料為後三筆純運量的平均，每筆資料中為全站於該時刻的運量共 108 個值，代表每個捷運站 15 分鐘運量的平均值，因此輸出資料為一個(79275, 108)的矩陣。

將輸入/出資料準備完成後，需要使用 Python 中提供機器學習與資料集合(data clustering)的 Scikit-learn(SKlearn)當中分割訓練資料與預測資料的 train_test_split，透過這個套件將資料集依照比例隨機分為訓練與資料集，就可以產生預測與訓練資料集，分別為訓練資料集的輸入 x 與輸出 y，以及測試資料集的輸入 x 與輸出 y。資料分割的重要性在於不將所有的資料全部輸入到模型中，避免模型學習所有資料而造成過度擬合(overfitting)，過度擬合意味著模型訓練得太好，與訓練資料集過於接近，這通常發生在模型過於複雜的情況下，模型在訓練資料上非常的準確，但對於未訓練資料或者新資料會很不準確，因此過度擬合對於模型預測是必須避免的狀況。對於資料的分割比例沒有一定的標準，而在機器學習與深度學習中慣用的資料分割比例有兩種，第一種為訓練資料與測試資料 7:3 的比例，另外一種為 8:2，本研究採用最被廣泛使用的 7:3 作為資料分割的依據。將訓練資料輸入模型，讓模型學習資料間的關係來更新模型權重，而測試資料則用來評估模型預測的精準度。

3. 模型編譯(compile)與參數設定

曾任 Google 深度學習研究團隊聯合創始人、百度首席科學家，吳恩達曾對於深度學習有這樣的評價:” Deep Learning has also been overhyped. Because neural networks are very technical and hard to explain, many of us used to explain it by drawing an analogy to the human brain. But we have pretty much no idea how the

biological brain works.” (Ng, 2016)，此段話說明了在深度學習的問題中，我們往往只能透過不同的參數組合，找到「較好」的結果，因此好的模型與參數組合必須透過試誤法(Trial and error)來尋找。

模型可分為輸入層、隱藏層與輸出層三個部分，以下針對重要參數進行說明。輸入層與隱藏層皆由 ConvLSTM2D 組成，其中重要函數為:

(1). 濾波器(filters):將輸入的圖片轉化為特徵圖的個數，通常為 4 的倍數。

在本模型中設定為 20。

(2). 卷積核大小(kernel_size):一般為正方形且邊長為奇數，便於尋找中心點，此部分決定以多大的尺寸來提取卷積特徵。

本模型選擇 3*3 作為卷積核的大小。

(3). 激發函數(activation):一個節點的激勵函數定義了該節點在給定的輸入或輸入的集合下的輸出，卷積運算後會使用激發函數加上非線性特徵，得到特徵圖(feature maps)。

選擇 tanh 函數，tanh 能夠讓加速資料的收斂。

(4). 補零方式(padding):卷積層取週邊 kernel_size 的滑動視窗時，若超越邊界時，是否要放棄這個 output 點(valid)、一律補零(same)、還是不計算超越邊界的 Input 值(causal)。

本模型選擇 same，讓周邊的特徵值也能被考慮進去。

(5). 資料格式(data_format):一般預設為” channels_last”，而在本模型中選擇” channels_first”，將通道資料放在第二維。

4. 模型擬合與預測結果

模型擬合(fitting)使用 train_x 做為輸入，train_y 作為輸出，以全體資料的 70%進行訓練。訓練完畢後，使用 evaluate 評估 x_test 與 y_test 可以得到模型的損耗(loss)與預測正確率，透過 predict 指令輸入訓練資料集的 x 預測出 y 值，將預測出的 y 與訓練資料集的 y 比較，就能了解模型實際的預測成效如何。在訓練

過程中為了防止模型產生過度擬合的狀況使用了以下兩種方法:

(1). 驗證資料分割(Validation split):從訓練資料中提取一部分的資料作為驗證資料，驗證資料主要的功能是調整參數，協助模型降低訓練產生的誤差。在模型中使用 30%的測試資料作為驗證資料分割的用途。首先在模型擬合時使用 validation split 進行參數調整，調整完參數後再使用全部的訓練資料進行擬合。

(2). 提前停止(Early stopping): 是一種在使用如梯度下降之類的疊代優化方法時，可防止過度擬合的正則化(Regularization)方法。在某個節點之前，更好地擬合訓練集使得模型在訓練集之外的數據上表現得更好；但在該節點之後，過度的擬合訓練集會增加誤差。提前停止提供停止疊代的條件，以便在模型開始過擬合之前停止疊代優化。在本研究中使用 min 模式來尋找損耗(loss)最小的參數組合，在驗證資料的損耗(loss)停止下降後停止模型繼續擬合，代表模型已無法繼續學習。

2. **預測結果分析與討論**

首先對 15 分鐘的預測模型進行 K-折交叉驗證與均方根誤差在不同情境下的評估結果進行分析與討論。接著進行不同預測任務的分析與討論，有/無加入特徵值的 15 分鐘資料集的預測結果，後將較佳的結果與 30 分鐘、60 分鐘的預測結果進行比較，觀察模型對於不同時步資料的預測能力。最後將使用先前預留半年的資料，針對臺北 101 跨年與演唱會進行 15 分鐘運量預測，測試模型在未學習資料中對於大型活動的預測能力。

1. K-折交叉驗證(K-fold cross validation)

K 折交叉驗證常被使用在判斷驗證機器學習與深度學習模型是否有過度擬合的方法。使用 scikit-learn 中 K-fold 套件來進行驗證，在每個測試中依然使用早停法來防止過度擬合以及加快模型預測速度，在每個預測結束後將該次權重進行預測(predict)，並用預測出的 y 值與真實的 test_y 計算均方根誤差做為判斷該次預測

結果的好壞。10 個 fold 的平均為 0.9887，平均的均方根誤差為 45.7112，與模型原本預測出的均方根誤差 28.94 有所差異。表 5.2 為 10 次預測的精準度與均方根誤差結果。

fold	Accuracy	RMSE
1	0.9968466	31.02937250454618
2	0.9871342	36.00027522916915
3	0.9889001	43.61372336509496
4	0.9837285	30.142483711687515
5	0.9918012	33.995130062220234
6	0.98751104	86.64170883708664
7	0.98776335	31.572175147028616
8	0.98776335	59.82805012826649
9	0.9896556	40.68658583228812
10	0.9863757	30.52928052305188
average	0.9887479722499848	45.711183482190506

表 5.1　K 折交叉驗證結果

2.　均方根誤差(Root Mean Square Error, RMSE)

以三種情境探討在不同狀況下模型的預測能力，情境分別為:同一時間不同站點的預測精準度，同一站在不同時間下的預測精準度，以及平日與特殊日下，同一時間不同站點預測精準度比較。在此部分以正規化的均方根誤差(coefficient of variation, CV)來進行計算，以百分比的方式來表示兩數據的差異程度。對預測出的 y 與 y_test 進行整體的驗證，整體的相對誤差為 17.04%，以下透過情境分析，針對預測結果與當中的誤差做更深入的探討。

(1). 同一時間不同站點

此部分比較的是在同一個時步當中，108 個站的預測與實際值之差異。圖 5.3

為全部驗證資料做出的相對誤差繪製成直方圖，折線圖為核密度估計(kernel density estimation, KDE)，將直方圖轉換為 KDE 可以清楚的看到資料以連續的線段被呈現，也能清楚的看到資料集中的狀況。x 軸為 CV RMSE 值，y 軸為核密度估計值。可以看到大部分的資料集中在相對誤差小於 50 的地方，因為整體的誤差為 17%，則以 20%的誤差來作為判斷預測好壞的標準，則在測試資料中相對誤差在 0%到 20%間的資料共有 17897 筆，約占測試資料的 75%。

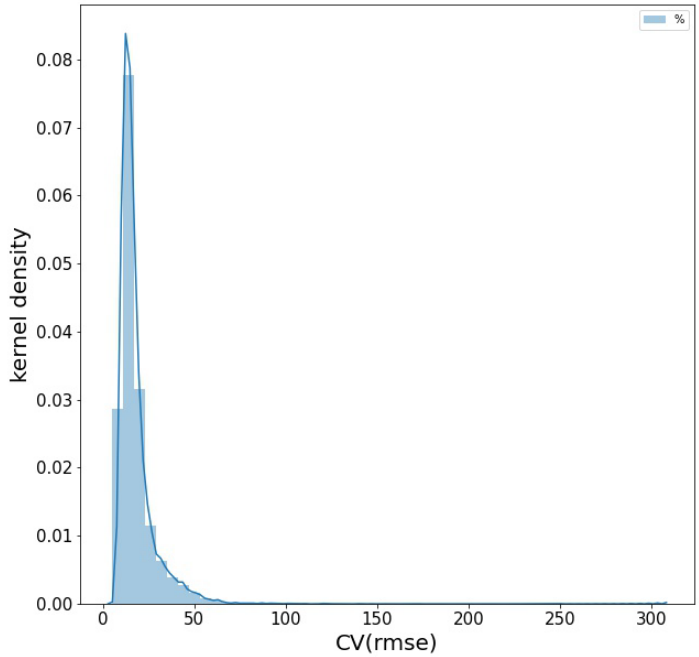


圖 5.1 同一時間不同站點相對誤差圖

(2). 特殊日與平日下，同一時間不同站點

這個部分使用原模型的權重，輸入三年中所有特殊日資料共 31827 筆來進行預測。特殊日資料的相對誤差為 18.97%，比整體資料還要高，在這邊依然使用 20%來作為判斷預測好壞的指標。在特殊日資料中有 20991 筆資料小於等於 20%的誤差，約佔資料的 66%。在圖 5.4 中可以發現折線圖也相對較寬，代表資料中數值差異較大較分散。

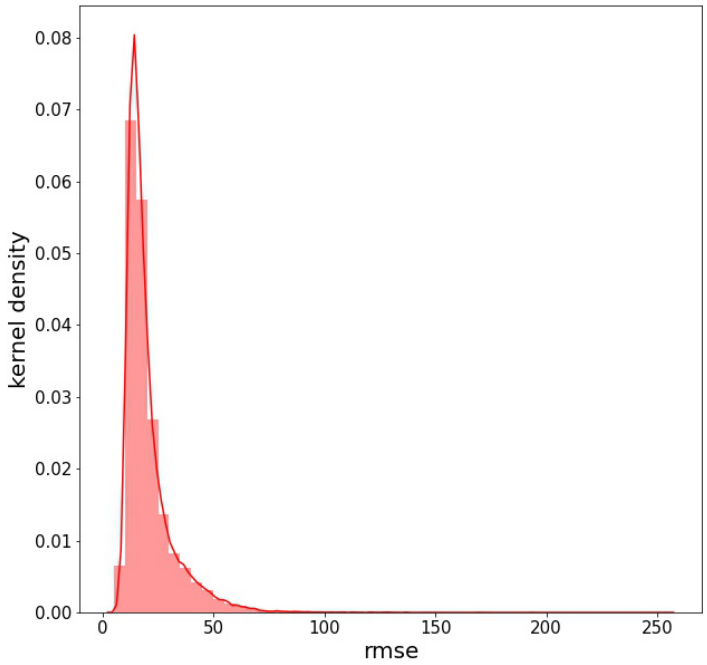


圖 5.2 特殊日運量相對誤差圖

平日的資料共 47449 筆，亦使用三年中所有平日資料來作為預測的資料。平日資料整體的相對誤差為 15.70%，可以發現比整體以及特殊日的誤差還要低，代表模型對於平日資料的預測有較好的結果。在 20%的相對誤差標準下，有 39001 筆為預測良好的資料，約佔資料 82%。圖 5.5 中可以發現，資料相對集中，代表誤差歧異度較低。

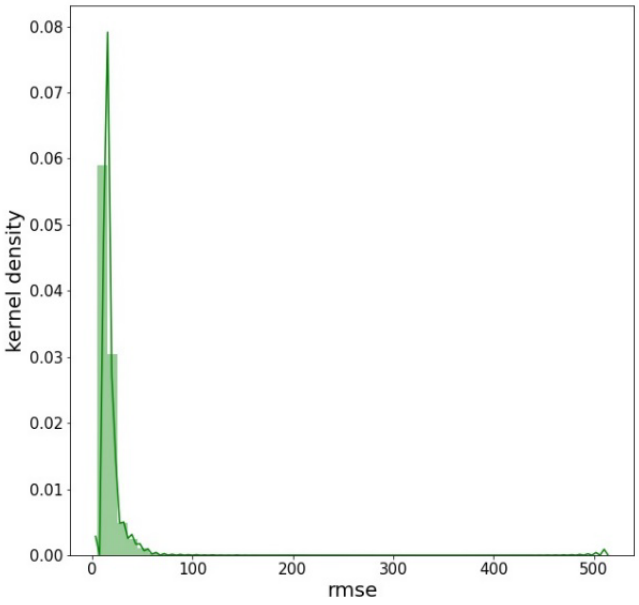


圖 5.3 平日運量相對誤差圖

(3). 同一站點不同時間

觀察每個站在不同時間下的預測誤差，此處使用一般的均方根誤差，若要使用正規化的均方根誤差需要除以 y_test 的均值，而在這邊的 y_test 為一站的值，有可能為 0，計算容易出現錯誤，因此選擇一般的均方根誤差進行驗證。計算完每站的 RMSE，共有(108, 23783)筆資料。將每站的資料進行平均，得到 108 站的 RMSE 平均值，當中均方根誤差前 10%高的站點分別為: 南京復興、忠孝復興、南港展覽館、中正紀念堂、臺北車站、中山、淡水、西門、市政府、松江南京、臺北小巨蛋，共 11 站(表 5.3)。從得到的結果可以發現這些站大多為路線交會的捷運站，且這些站點周圍大多有舉辦活動的會場，因此可以推測因為活動或是旅遊等帶有特定目的的旅次，又這些旅次並非固定旅次，讓該站運量變化較大而難以預測。

站名	RMSE
臺北車站	104.4245469
西門	45.38325695
市政府	43.27448177
忠孝復興	34.4943447
淡水	34.44599083
中正紀念堂	32.80128663
松江南京	32.39688012
南京復興	30.54669302
南港展覽館	30.04238321
中山	29.71782366
臺北小巨蛋	29.63856536

表 5.2 相對誤差較大的站點

3. 有/無特徵值運量預測比較

在資料預處理時，針對特殊日與平日不同的日型態，在矩陣的角落加入梯形與倒梯形的特徵值，試圖讓模型可以根據形狀的不同，將不同的日型態分開後進行後續資料的連結，以得到更好的預測精準度，此部分以三個時步的預測進行說明。無加入特徵運量預測精準度為 99.10%，均方誤差(MSE)為 1095.1470，均方根誤差(RMSE)為 33.0930；而加入特徵值的運量預測結果為 99.03%，均方誤差為 837.5446，均方根誤差為 28.9403。從預測結果來看，雖然兩種運量資料在預測中都達到 99%的預測精準度，但加入特徵值的運量在 MSE 與 RMSE 的誤差較少，因此採用有特徵值的運量作為後續使用的資料。為了能更直觀的展示 ConvLSTM 的預測能力，選擇累積運量最多的臺北車站作為預測的對象，臺北車站為累積運量最多之捷運站，為板南線與淡水信義線交會的轉運站，運量的標準差(取整數)為 685 為全捷運系統之冠(第二為西門站，標準差為 331)，表示臺北車站的運量變化是全系統中最高，也就代表其運量的預測難度是較高的。圖 5.1 中挑選臺北車站的 test_y 與預測出的 y 各 100 筆，上方圖為有特徵值的運量比較圖，下方為無特徵值的比較，可以看到不管有或無特徵值的運量都與真實值相當接近，但在有加入特徵值的資料中，其折線圖與真實資料的折線圖較接近，與均方根誤差的量測結果相同。

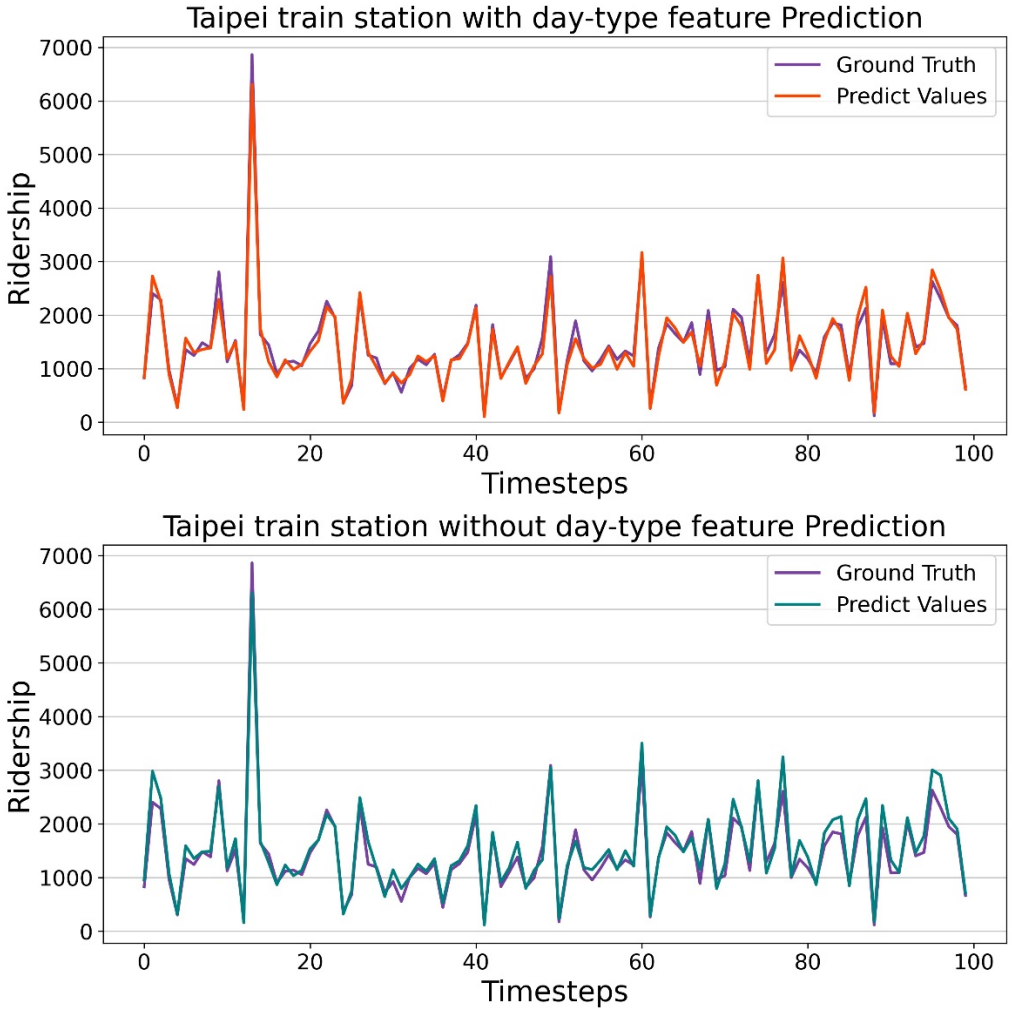


圖 5.4 有無特徵值運量比較

4. 不同時步運量預測比較

在此部分比較 ConvLSTM 模型在不同時步下的預測能力，採用的輸入資料集分別為三個時步(15 分鐘)、六個時步(30 分鐘)、十二個時步(60 分鐘)，輸出資料集為 108 個站時間長度的平均運量，以 7:3 的比例分為訓練與測試資料，三個資料的總筆數分別為 79275、39637、19818 筆。在表 5.1 的結果可以看到，隨著預測時間越長，均方誤差也越高，但本來預期模型在更長的預測中，預測精準度會跟著時間變長而下降，反而隨時間變長而上升，另外在 30 與 60 分鐘的訓練中，驗證資料的精準度大於訓練資料的精準度，有輕微過度擬合的狀況。

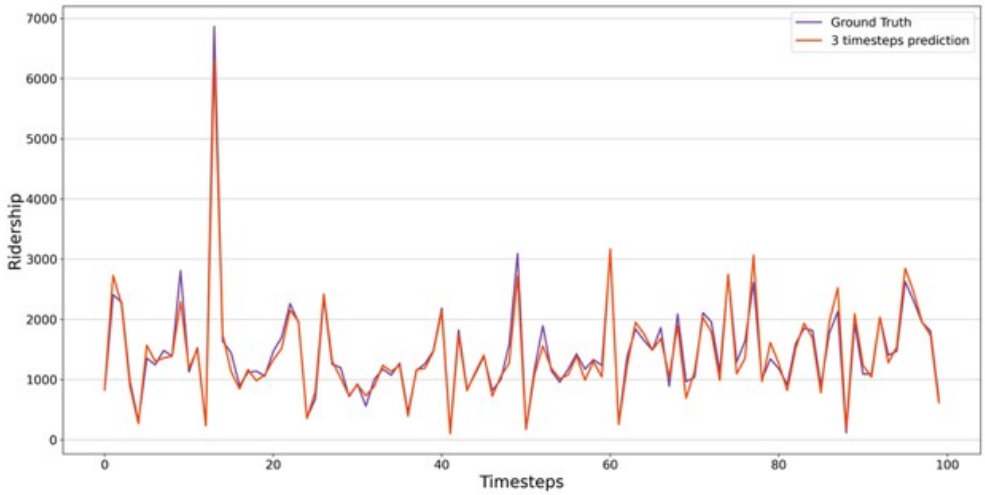
從資料與驗證結果可以推測資料筆數太少造成均方誤差隨著時步變長而增加，預測精準度增加與本模型預測的方式有關，在 15 分鐘的預測裡，每筆 x 僅

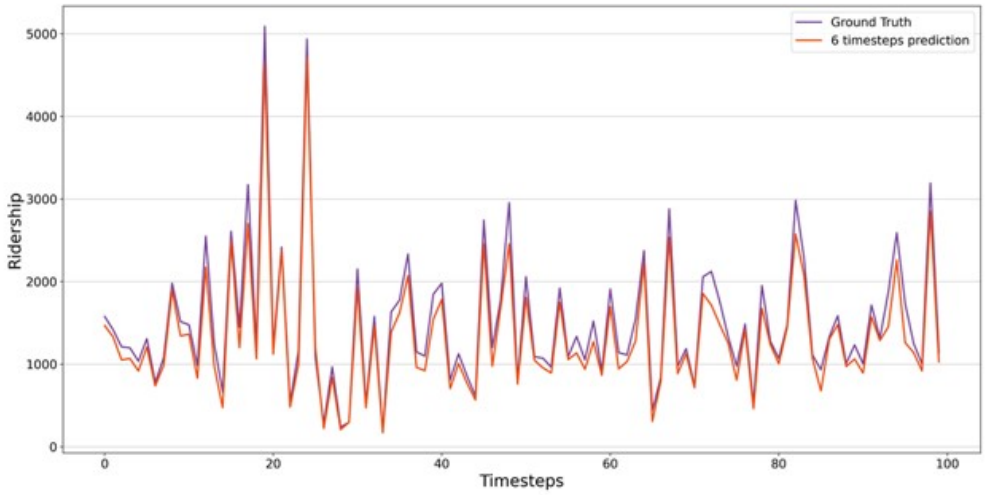
有三筆五分鐘運量矩陣，而隨著時步變長，x 資料中的矩陣數量變多，而 y 資料是運量的平均，只有一維的運量資料，因此會隨著預測時步變長而提升預測精準度。

	預測精準度	均方誤差
15 分鐘	99.03%	837.5446
30 分鐘	99.60%	875.6776
60 分鐘	99.88%	3892.7908

表 5.3 不同時步運量預測比較

圖 5.2 為前 100 筆不同時步下臺北車站的運量相對誤差折線圖，可以看到隨著時步變長，預測與真實值的誤差越大，符合均方誤差所展示的結果，但可以發現即使在極高的運量下模型仍然能預測出相當接近的數值，顯示本模型擬合結果相當良好。三種預測的結果中依然是十五分鐘的預測對於活動有最好的預測能力。





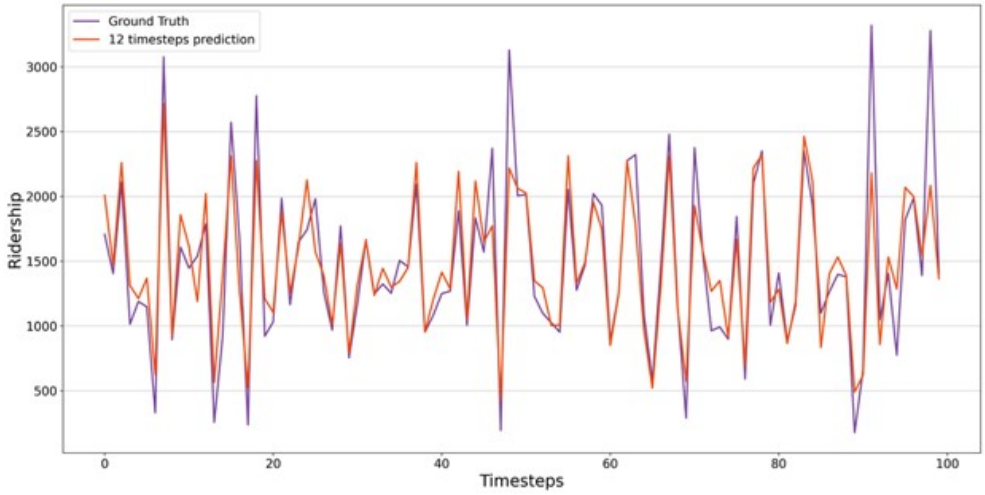


圖 5.5 不同時步運量相對誤差折線圖

5. 特殊節日預測驗證

本研究的貢獻為訓練模型使其具有預測大型活動運量的能力。為了驗證模型預測大型活動的能力，在本小節使用 2018 的跨年晚會以及 2019 年林俊傑演唱會作為驗證的大型活動。

2018 年跨年活動採用 2018 年 12 月 31 日早上 6 點的跨年晚會到隔日的 2019 年 1 月 1 日晚間 11:25 來作為跨年活動以及驗證的時間，此段時間捷運公司推出了「42 小時不收班」的疏運方案，因此將此時間視為跨年疏運時間，將這段時間的運量進行預測，觀察均方根誤差(相對誤差)隨時間的變化，以及在同一時間下哪一站的預測誤差最高，觀察是否符合捷運公司所建議民眾散場搭乘的場站(國父紀念館站、永春站、信義安和站、象山站、南京三民站)以及是否符合臺北 101 周圍之捷運站(臺北 101 站、市政府站)。

圖 5.6 為跨年活動運量預測的均方根誤差隨時間的變化折線圖，可以發現誤差的高峰分別出現在第 71~73 與第 88~95 個時步，換算成真實時間為 2018/12/31 晚間 23:30 到 2019/1/1 的 00:15，以及凌晨 4 點至 5 點 45。這兩段時間正好為跨年煙火施放以及 4:30 開始的元旦升旗典禮，因為使用捷運人數增加而造成誤差增大。

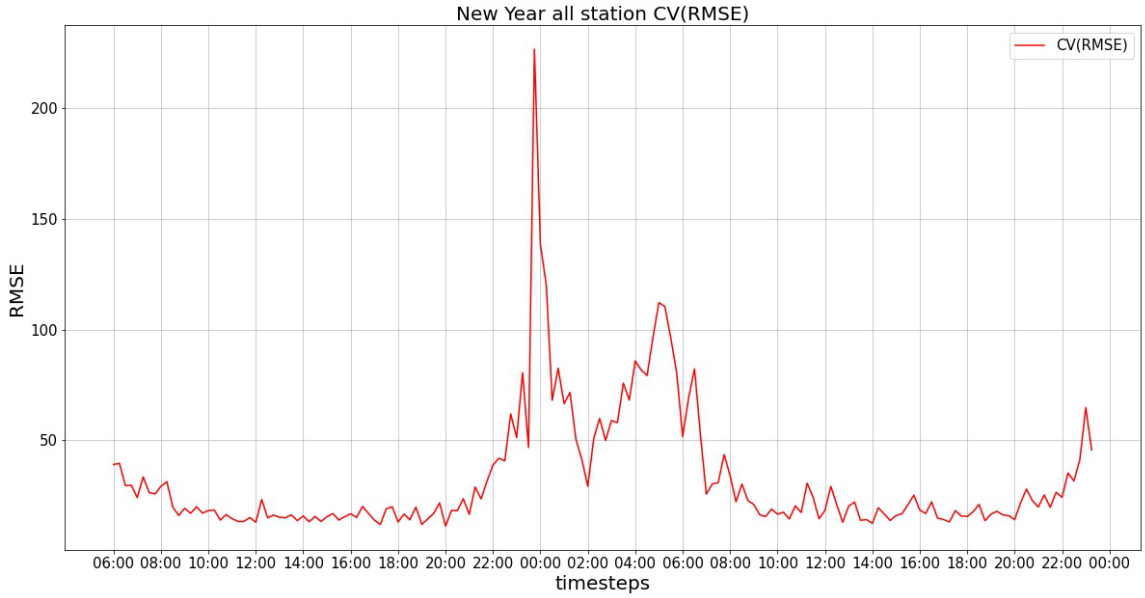


圖 5.6 跨年活動全站預測相對誤差折線圖

表 5.4 從每個站在活動時間期間的誤差來觀察，可以發現與捷運公司宣導疏運的站點不同，這些站的位置也與跨年活動場地相距較遠，其中萬芳社區、大湖公園、紅樹林、先嗇宮、辛亥、南港軟體園區為全捷運系統運量倒數 20 名的站，可以推測誤差較大的場站在跨年活動與元旦升旗時產生比平常更多的運量，因此模型並未學習到突然增加的運量而造成誤差。

站名	相對誤差(RMSE_CV)
萬芳社區	45.040946
大湖公園	35.995886
紅樹林	33.183119
象山	32.176964
先嗇宮	30.955455
辛亥	27.921647
信義安和	26.418883
台大醫院	25.397678
南港軟體園區	24.130315
小南門	21.818514

表 5.4 跨年單站誤差排名

圖 5.7 跨年活動所指定的疏運場站以及臺北 101 周圍的場站的預測結果，紫線為真實運量，藍線為預測運量，從各站的運量變化來看，主要的跨年站點(國父紀念館、臺北 101、臺北市政府、象山)真實的運量，在午夜 12 點達到最低，前後都有高峰，呈現一致的變化；而從運量變化可以看出永春、信義安和與南京三民為主要的疏運場站，運量在午夜 12 點過後急遽上升。

站名	相對誤差(RMSE_CV)
國父紀念館站	18.781857
市政府站	12.384147
永春站	15.562621
象山站	32.176964
臺北 101 站	14.661963
信義安和站	26.418883
南京三民站	11.409694

表 5.5 跨年活動周邊場站誤差

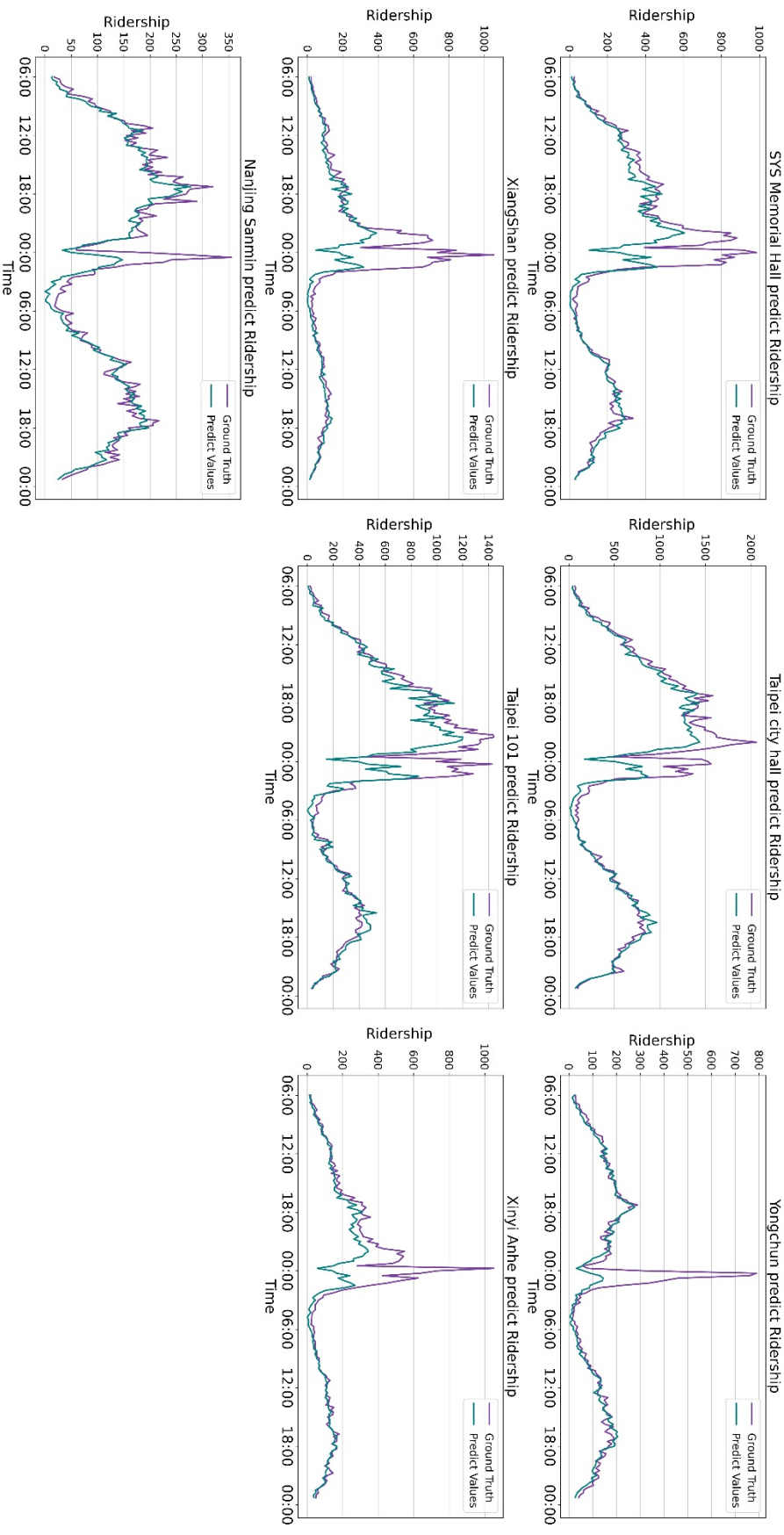


圖 5.7 跨年活動周邊場站誤差折線圖

因為在特殊日的運量中加入了小巨蛋活動的資料，挑選一個於小巨蛋舉辦且具有規模的活動做為驗證的事件。選擇於 2019 年 2 月 14 至 17 日舉辦的「JJ 林俊傑聖所世界巡迴演唱會臺北站」作為驗證的大型活動，此場活動為林俊傑於亞洲巡迴演唱會的最終場，四天共 4 萬人進場欣賞演唱會。圖 5.8 為臺北小巨蛋站四天演唱會運量比較折線圖，紫線為真實運量值，藍線為預測運量值，紅色虛線為每天運量的間隔，時間間隔因為晚間 12 點接續隔日早晨 6 點的運量，因此統一以 00:00 來表示。演唱會於晚間 7:30 開始，晚間 6:00 開放進場，預計於晚間 10:30 結束。從折線圖可以看出，雖然依然存在散場人潮難以預測的缺點，但模型準確的預測到了演唱會開始前聚集的人潮，也預測到了散場急升的人潮走勢，也可以從四天的運量看出 2/14、2/15 為平常日，而 2/16、2/17 為假日，兩者運量呈現相當明確的分別。

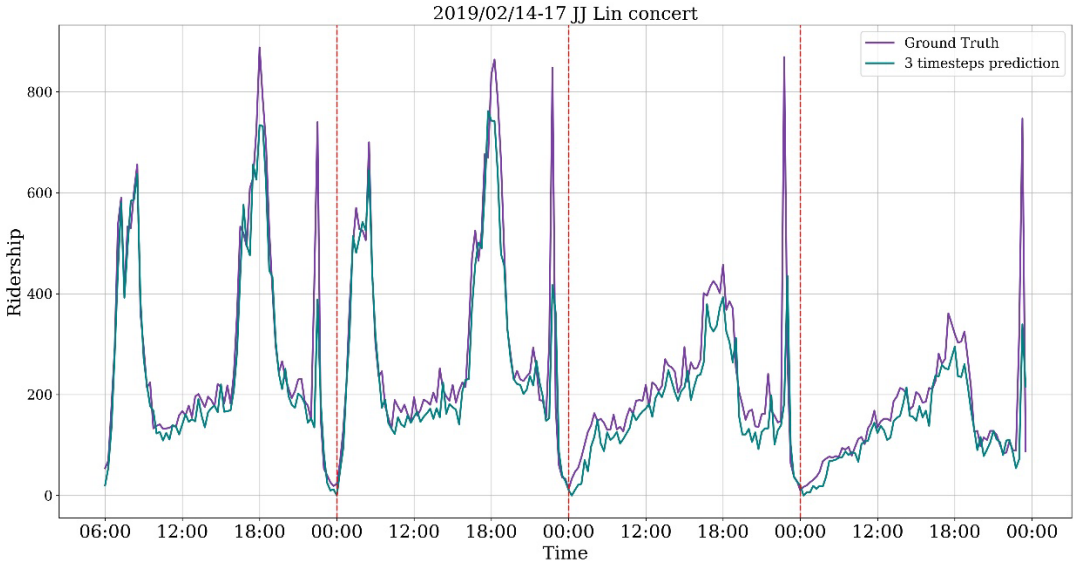


圖 5.8 林俊傑演唱會運量比較折線圖

驗證結果說明模型對於平常的運量有非常良好的預測能力，但模型對於突發的巨量人潮有低估的現象，也就是模型對於活動散場的人潮預測能力較差，在實際應用上可以輔以人工來對於預測量進行調整。

因為使用 15 分鐘或是更長時間的預測有可能會將活動散場的運量稀釋，因此使用同樣的林俊傑演唱會的資料，以 5 分鐘與 10 分鐘的運量進行預測。從圖 5.9 可以看出五分鐘的預測運量在原先預測較差的散場運量有了更好的預測結果，在十分鐘的預測也發生了低估運量的情況，因此確實證實了因為輸出採用三個時步的平均，因此會有稀釋短時間內突發運量的狀況。在跨年活動中也呈現類似的情況，在十分鐘的預測當中對於突發運量仍有低估現象，而五分鐘則有非常接近的預測結果。

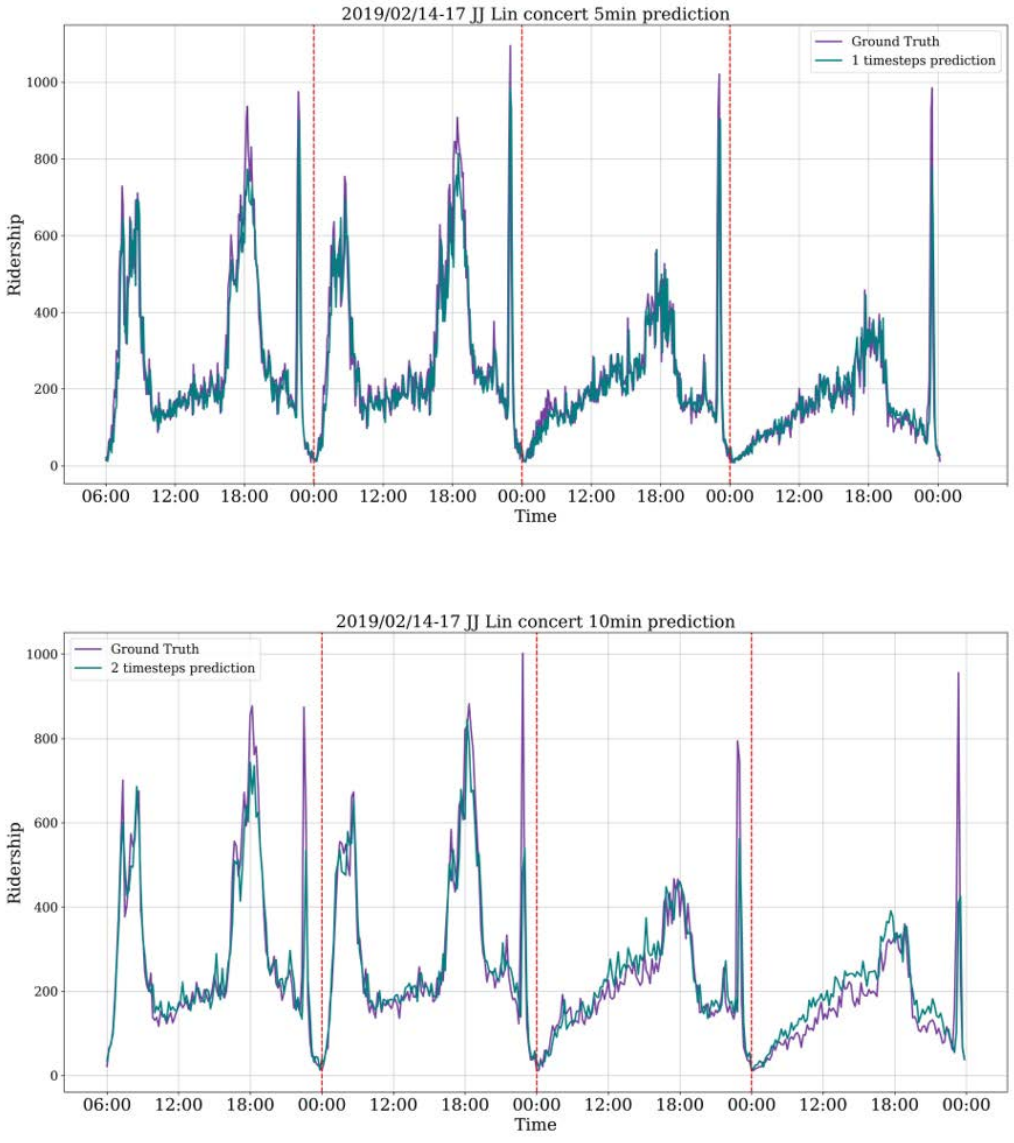


圖 5.9 5 分鐘與 10 分鐘演唱會運量折線圖

6. 小結

本章從模型參數的設定與實際預測開始，利用不同運量資料與不同情境進行預測的驗證。以變化最大的臺北車站進行有無加入特徵值的預測，有加入特徵值的運量取得了較佳的預測結果。接著使用未受過訓練的元宵燈會開幕式作為不同時步下模型預測的驗證，預測結果為 15 分鐘運量有最好的預測結果，但對開幕活動帶來的運量捕捉仍有待加強。K 折交叉驗證法驗證模型是否有過度擬合的狀況，驗證結果顯示雖然預測精準度並無太大的差異，但均方根誤差卻有較大的差異，表示在模型中確實有過度擬合的狀況產生。在均方根誤差驗證中，以三個情境來分析預測與實際值的差異，利用正規化的均方根誤差去除樣本數帶來的誤差，以 20%的誤差做為門檻，呈現更加可信與直觀的分析結果，此部分結果顯示模型對於大部分的資料還是有預測的能力，以及模型對於平日、運量變化較少的站有更好的預測能力。最後以跨年與演唱會作為大型特殊活動的驗證，跨年活動以臺北捷運的 42 小時疏運作為時間範圍，當中包含跨年以及元旦升旗典禮，從驗證結果來說，跨年活動的極高運量對於預測模型是難以預測的，雖然預測模型能夠捕捉到運量增加的特徵，但無法準確預測驟升的運量，也造成了在每站的試驗當中，一些平常運量較低的站點，也出現了比平常高出許多的運量而造成較大的誤差。在演唱會的驗證中，模型準確的預測演唱會前聚集的人潮，也預測到演唱會結束後人數驟升的趨勢。對於跨年活動以及演唱會結束的人潮模型模型僅能預測到突然升高的趨勢，而並非預測到準確的值，透過觀察兩場活動的折線圖可以發現，在突發的運量之前的運量都是較低的，而並非隨著活動有相對升高的趨勢，且活動結束的人潮是在短時間內進入到捷運站中，相對於活動前的運量人潮聚集的時間較長，短暫時間升高的運量讓模型是難以預測的。總體來說，ConvLSTM 對於運量的預測是相當出色的，能夠準確捕捉大部分運量及其趨勢，但對於極高的人潮會有低估的狀況，實務上可以人工來對該數值進行調整。

第六章 結論與建議

6.1 結論

隨著社會普遍經濟狀況的提升與現代社會的繁忙，國人對於休閒的需求提升，大型活動舉辦日漸增加。而資料科學在近年成為熱門的研究領域，並被廣泛的應用在各種研究中，而在交通預測中也被用來預測運量與訊號辨識等領域。臺北捷運目前建置的旅客預測系統只能預測一小時後的運量，對於列車的調度與場站因為突發人潮造成的擁擠無法處理，列車調度需要更高的時間解析度，避免造成派出的車輛太早或太晚徒增營運成本；場站的擁擠造成旅客等待時間增加，並有發生推擠等危險事故的潛在問題。本研究利用 2016 年至 2019 年每五分鐘一筆的悠遊卡進出站運量資料，針對共 3000 多萬筆運量紀錄進行資料清洗、整理，將運量資料利用空間結合(spatial join)的方法，將運量資料根據各捷運站對應的位置，映射至以真實世界為範圍的網格中以製作輸入模型的圖片(矩陣)。將製作完成的矩陣聚集(aggregate)成每 15 分鐘一筆的輸入資料集，並以未來 15 分鐘 108 站之平均運量做為對應的輸出資料集。應用由 LSTM 發展而來的 ConvLSTM 模型，利用其捕捉空間與考慮長短期運量的特色進行預測，選擇臺北小巨蛋的活動以及假日來製作特殊日資料集，在運量資料中加入平日與特殊日的特徵值，模型預測結果的精準度為 99%。預測完成後利用交叉驗證、均方根誤差以及實際案例來進行驗證，分別以不同的情境與事件來進行分析，呈現本研究的預測在不同層面的結果。本研究最大貢獻與主要目標在於建立可預測大型活動運量之模型，驗證結果顯示模型對於不同的日型態都能有相當良好的預測，對於大型活動的預測也能準確捕捉因為活動而增加的運量，而活動結束的人群因為帶來太大量的運量，因此僅能抓到運量驟升的趨勢，加上人工調整即能對到來的人流有所掌握。

6.2 建議

1. 納入更多大眾運輸工具

建議納入更多使用智慧卡的大眾運輸工具如公車、ubike，並將尺度提升為在選定範圍內(方格)進出的人群，這樣可以達到監控都市人流的效果，可以在交通安全與道路擁擠程度等方面提出貢獻。

2. 納入更多票種

本研究僅用到悠遊卡刷卡資料，但使用捷運的部分民眾會使用單程票或是其他票種。透過納入更多票種，能得到更完整的進出站紀錄，更多的紀錄意味著更加穩定的預測結果，也更貼近真實的運量情況。

3. 使用其他深度學習模型或架構

在本研究中，雖然精準度非常高，但在資料驗證的部分可以看出此模型有過度擬合的狀況，因此建議未來要進行相關研究可以嘗試不同模型架構，或是以更深的模型架構來進行訓練。

4. 大型活動預測方式

為了讓模型對於活動的運量有更好的預測，可針對活動前後加入特徵值，透過標註特徵值讓模型對於到來的活動能有更好的預測。或建立一般與特殊運量模型，透過設定運量閾值，當運量到達某個條件則使用活動的預測方式，若非則使用一般運量預測模型。

5. 建議可以加入運量過去的趨勢，或是同期的歷史資料，有助於模型預測

參考文獻

Chen, E., Ye, Z., Wang, C., & Xu, M. (2019). Subway Passenger Flow Prediction for Special Events Using Smart Card Data. *IEEE Transactions on Intelligent Transportation Systems*.

Chollet, F. (2015). Keras Documentation. from https://keras.io/

Chollet, F. (2018). *Deep Learning with Python* (C. Taylor Ed.). New York: Manning Publications Co.

Dan. (2019). Tensorflow 介紹 及 Tensorflow 2.0 相關知識. from https://ithelp.ithome.com.tw/articles/10215969

Ding, C., Wang, D., Ma, X., & Li, H. (2016). Predicting short-term subway ridership and prioritizing its influential factors using gradient boosting decision trees. *Sustainability, 8*(11), 1100.

Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation, 9*(8), 1735-1780.

Karlaftis, M. G., & Vlahogianni, E. I. J. T. R. P. C. E. T. (2011). Statistical methods versus neural networks in transportation research: Differences, similarities and some insights. *19*(3), 387-399.

Khashei, M., & Bijari, M. (2011). A novel hybridization of artificial neural networks and ARIMA models for time series forecasting. *Applied Soft Computing, 11*(2), 2664-2675. doi: 10.1016/j.asoc.2010.10.015

Kwang-Ho, K., Hyoung-Sun, Y., & Yong-Cheol, K. (2000). Short-term load forecasting for special days in anomalous load conditions using neural networks and fuzzy inference method. *IEEE Transactions on Power Systems, 15*(2), 559-565. doi: 10.1109/59.867141

López, M., Sans, C., Valero, S., & Senabre, C. J. E. (2018). Empirical Comparison of Neural Network and Auto-Regressive Models in Short-Term Load Forecasting. *11*(8), 2080.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks, 3361*(10), 1995.

Li, Y., Wang, X., Sun, S., Ma, X., & Lu, G. (2017). Forecasting short-term subway passenger flow under special events scenarios using multiscale radial basis function networks. *77*, 306-328.

Liu, L., & Chen, R.-C. (2017). A novel passenger flow prediction model using deep learning methods. *84*, 74-91.

Liu, Y., Lai, X., & Chang, G.-L. (2006). Two-Level Integrated Optimization System for Planning of Emergency Evacuation. *132*(10), 800-807.

Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (2013). *Machine learning: An artificial intelligence approach*: Springer Science & Business Media.

Ng, A. (2016). What does Andrew Ng think about Deep Learning? , from https://www.quora.com/What-does-Andrew-Ng-think-about-Deep-Learning

Nguyen, H., Kieu, L.-M., Wen, T., & Cai, C. (2018). Deep learning methods in transportation domain: a review. *IET Intelligent Transport Systems, 12*(9), 998-1004. doi: 10.1049/iet-its.2018.0064

Pereira, F. C., Rodrigues, F., & Ben-Akiva, M. (2015). Using data from the web to predict public transport arrivals under special events scenarios. *Journal of Intelligent Transportation Systems, 19*(3), 273-288.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. J. T. J. o. M. L. R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *15*(1), 1929-1958.

Xavier, A. (2019). An introduction to ConvLSTM. from

https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7

Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., & Woo, W.-c. (2015). *Convolutional LSTM network: A machine learning approach for precipitation nowcasting.* Paper presented at the Advances in neural information processing systems.

Zhang, J., Zheng, Y., & Qi, D. (2017). *Deep spatio-temporal residual networks for citywide crowd flows prediction.* Paper presented at the Thirty-First AAAI Conference on Artificial Intelligence.

马晓磊, 张., 丁川,刘剑锋. (2017). China Patent No. 北京航空航天大学.

台北大眾捷運股份有限公司. (2010). 使命與願景. from https://web.archive.org/web/20120326134542/http://www.trtc.com.tw/ct.asp?xItem=1315925&CtNode=24529&mp=122031

呂佩容. (2012). 特殊活動之運輸需求行為分析-以台北小巨蛋演唱會為例. *中原大學土木工程研究所學位論文*, 1-120.

周汶叡, 柯召璇, 李思葦, 陳譽晏, & 李宗益. (2018). 第 4 代臺北都會區運輸需求預測模式更新版(TRTS-4S)成果. *捷運技術, 53*.

林泉亨. (2015). *基於社群網路資料之捷運運量預測 MRT Demand Prediction through Social Media.* Retrieved from http://ntur.lib.ntu.edu.tw/handle/246246/277956

林炫洋. (1982). *台北市中運量捷運系統運量預測模式之研究*. (碩士), 國立交通大學. Retrieved from http://hdl.handle.net/11536/51636

林誌銘, & 王晉元. (2008). *應用基因演算法於捷運列車運行計畫之研究*.

邱莉玲. (2015). 電子票證鏖戰 一卡通揮軍北上. *工商時報*. Retrieved from https://www.chinatimes.com/newspapers/20150901000065-260202?fbclid=IwAR2k9SvS8jNBPU9r4YxDDN7maRa_eQoqr1t1CGIMu4YmYnVBJFRM2fCbzds&chdtv

黃志偉. (2015). *運用人工神經網路探討短期高雄捷運班次之運能*. 成功大學. Available from Airiti AiritiLibrary database. (2015 年)

臺北大眾捷運股份有限公司. (2008). 從營運公司觀點介紹臺北捷運運量推估方式. *軌道經營與管理, 3*.

臺北大眾捷運股份有限公司. (2014). 文湖線加車啟動機制 － 以臺北小巨蛋為例. *軌道經營與管理, 14*.

臺北市大型路外活動交通維持作業辦法 (2009).

蔡宗憲. (2006). *類神經網路模式於短期列車旅運量需求預測之應用*. 成功大學. Available from Airiti AiritiLibrary database. (2006 年)

蔡宗憲, 李治綱, & 魏健宏. (2006). 短期列車旅運需求之類神經網路預測模式建構與評估. [Artificial Neural Networks for Short-Term Railway Passenger Demand Forecasting]. *35*(4), 475-505. doi: 10.6402/tpj.200611.0475

魏健宏, & 陳奕志. (2001). 類神經網路模式在國內交通運輸研究之成果評析. *運輸計劃季刊, 30*(2), 323-347.

蘇怡瑄. (2009). *運用類神經網路預測捷運車站之運量*. 國立交通大學. Retrieved from http://140.113.39.130/cdrfb3/record/nctu/#GT079736519

附錄一 運量資料預處理程式碼

```
#讀入套件
%matplotlib inline
import os
import pandas as pd
import numpy as np
from google.colab import drive
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import colors
import math
from mpl_toolkits.axes_grid1 import make_axes_locatable
from scipy import stats
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
from keras.models import Model
from keras.models import Sequential
from keras.layers.convolutional_recurrent import ConvLSTM2D
from keras.layers.convolutional import Conv3D
from keras import optimizers
from tensorflow import keras
from tensorflow.keras.callbacks import EarlyStopping
from keras.layers import Input, Dense, MaxPooling2D, MaxPooling3D, Dro
pout, BatchNormalization, Flatten, Reshape
from keras.models import model_from_json
from sklearn import datasets
from sklearn.model_selection import train_test_split,KFold,cross_val_s
core
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn import metrics
from scipy.stats import zscore
```

```
#連結 google drive
drive.mount('/content/gdrive')


#讀入運量資料與場站名稱
mrt1 = pd.read_csv("D:/NTU/臺北市交通局授權_悠遊卡進出站資料
/mrt_ticket_five_01.csv",encoding='utf-8',header=None)
mrt2 = pd.read_csv("D:/NTU/臺北市交通局授權_悠遊卡進出站資料
/mrt_ticket_five_02.csv",encoding='utf-8',header=None)
mrt3 = pd.read_csv("D:/NTU/臺北市交通局授權_悠遊卡進出站資料
/mrt_ticket_five_03.csv",encoding='utf-8',header=None)
col_name = pd.read_csv("D:/NTU/臺北市交通局授權_悠遊卡進出站資料/捷運場站名
稱.csv",encoding='big5')
#加上欄位名稱
mrt1.columns=['資料蒐集時間','場站代號','進站人數','出站人數','整批資料更新時間
','存入 DB 時間','該筆資料更新時間','該筆資料更新時間 1']
mrt2.columns=['資料蒐集時間','場站代號','進站人數','出站人數','整批資料更新時間
','存入 DB 時間','該筆資料更新時間','該筆資料更新時間 1']
mrt3.columns=['資料蒐集時間','場站代號','進站人數','出站人數','整批資料更新時間
','存入 DB 時間','該筆資料更新時間','該筆資料更新時間 1']

#將資料蒐集時間從"Object"轉為"datetime"型態
mrt1['資料蒐集時間'] = pd.to_datetime(mrt1['資料蒐集時間'],format='%Y-%m-
%d %H:%M:%S')
mrt2['資料蒐集時間'] = pd.to_datetime(mrt2['資料蒐集時間'],format='%Y-%m-
%d %H:%M:%S')
mrt3['資料蒐集時間'] = pd.to_datetime(mrt3['資料蒐集時間'],format='%Y-%m-
%d %H:%M:%S')

#將表格連接
mrt4 = mrt1.append([mrt2, mrt3], ignore_index=True)

#刪除不需要的欄位
mrt4.drop(['整批資料更新時間','存入 DB 時間','該筆資料更新時間','該筆資料更新時間
1'], axis=1, inplace=True)
#將資料依照時間排序，並重新指派資料 index
mrt4 = mrt4.sort_values(by=['資料蒐集時間'], ascending=True)
mrt4.index = range(len(mrt4.index))
```

```
#添加人數總和的欄位
mrt4["人數總和"]=mrt4.進站人數+mrt4.出站人數

#篩選未在場站代號上的站點
b=list(set(mrt4['場站代號'])-set(col_name['捷運站代碼']))
b=pd.Series(b)
mrt4.index = range(len(mrt4))
mrt5=mrt4
mrt5.head()
mrt5.to_csv('mrt5.csv',encoding='utf-8')
#加上中文站名
empty_df=pd.DataFrame(columns = ["資料蒐集時間","場站代號" ,"進站人數", "出
站人數", "人數總和","捷運站名稱"])
def sta_name(sta):
    a = mrt5[mrt5['場站代號']==sta]
    a['捷運站名稱']=col_name[col_name['捷運站代碼']==sta]['捷運站名稱
'].values[0]
    return(a)

for i in col_name['捷運站代碼'].values:
  single_sta = sta_name(i)
  empty_df = pd.concat([empty_df,single_sta])

empty_df=empty_df.sort_values(by=['場站代號','資料蒐集時間
'], ascending=True)
empty_df.index=range(len(empty_df))
mrt_done=empty_df.to_csv('mrt_done.csv',encoding='utf-8')
#合併重複(轉運站)場站代碼
empty_df['場站代號
'].replace({102:11,106:53,52:51,129:55,90:10,133:89,132:107,108:9,98:3
1},inplace = True)
#合併重複站點運量
record_groupby = empty_df.groupby(["資料蒐集時間","捷運站名稱","場站代號
"],as_index = False).sum()
record_groupby = record_groupby.sort_values(by=['場站代號','資料蒐集時間
'], ascending=True)
record_groupby.index = range(len(record_groupby.index))
```

```
record_groupby_save=record_groupby.to_csv('record_groupby.csv',encodin
g='utf-8')

#裁切每日早晨 6 點到晚間 12 點之運量紀錄
b = pd.DataFrame(columns = ["資料蒐集時間", "進站人數", "出站人數", "人數總
和","站名","場站代號"])
mask1 = merge_time['資料蒐集時間']<='2019-07-01 00:00:00'
mask2 = merge_time['資料蒐集時間']>'2016-01-01 00:00:00'
merge_time=merge_time[(mask1 & mask2)]
merge_time.index=merge_time['資料蒐集時間']

def time_selection(station):
    single_sta=merge_time[merge_time['站名']==station]
    single_sta['資料蒐集時間'] = single_sta[['資料蒐集時間
']].between_time(start_time='06:00:00', end_time='00:00:00')
    single_sta = single_sta.dropna()
    qwer = single_sta.reset_index(drop=True)
    return qwer

for i in range(len(station_name)):
    time_slec = time_selection(station_name[i])
    b = pd.concat([b,time_slec])

b.index = range(len(b))
time_selection_save=b.to_csv('time_selection.csv',encoding='utf-8')

#根據捷運站經緯度選擇真實世界範圍
col_name = pd.read_csv("D:/NTU/臺北市交通局授權_悠遊卡進出站資料/捷運場站名稱
2.csv",encoding='big5'
mrt_point = [Point(xy) for xy in zip(col_name.經度, col_name.緯度)]
crs = {'init': 'epsg:3826'}
gdf = gpd.GeoDataFrame(col_name, crs=crs, geometry=mrt_point)
#讀入全台縣市 shapefile
tw = gpd.read_file('D:/NTU/臺北市交通局授權_悠遊卡進出站資料
/mapdata201911260954/TOWN_MOI_1081121.shp',encoding='utf-8')
tw.crs= {'init' :'epsg:3826'}
#選擇雙北行政區
tp_city = tw[(tw.COUNTYNAME=='臺北市')|(tw.COUNTYNAME=='新北市')]
```

```
tp_city.index=range(len(tp_city))
#套疊捷運站點與行政區圖
for i in range(len(tp_city)):
    if gdf.geometry.within(tp_city.geometry[i]).any()==False:
        tp_city.drop([i],inplace=True)
ax=tp_city.plot(figsize=(15,15))
gdf.plot(figsize=(15,15),color='red',ax=ax)
plt.grid(True)
plt.savefig('地圖套疊.jpg')
#網格映射(加入中文站名)
x = col_name.經度-121
y = col_name.緯度

plt.figure(figsize=(22,22))

plt.xlim(0.40, 0.62)
plt.ylim(24.95, 25.17)
plt.xticks(np.arange(0.40,0.625,0.005),size=10)
plt.yticks(np.arange(24.95,25.17,0.005),size=20)


plt.xlabel("longtitude(121°)",size=30)
plt.ylabel("latitude",size=30)

plt.tick_params(direction='in', colors='black')

q = col_name["捷運站代碼"]
e = col_name['捷運站名稱']

for a,b,c in zip(x,y,e):
    plt.text(a, b+0.001,s=c, ha='center', va= 'bottom',fontsize=20)


plt.scatter(x, y,s=100)
plt.grid(color='grey',linewidth=1)
plt.savefig('網格映射.jpg')

#資料輸入網格
```

```
col_name = pd.read_csv("D:/NTU/臺北市交通局授權_悠遊卡進出站資料/捷運場站名稱
3.csv",encoding='big5')
time_selec = pd.read_csv('time_selec.csv',encoding='utf-8')
time_selec['資料蒐集時間'] = pd.to_datetime(time_selec['資料蒐集時間
'],format='%Y-%m-%d %H:%M:%S')
time_selec.drop(['Unnamed: 0'], axis=1, inplace=True)
time_selec.drop(['進站人數','出站人數',], axis=1, inplace=True)
#產生網格
#劃出邊界
xmin,ymin,xmax,ymax = (121.4,24.95,121.625,25.17)
width, height = (44,44)
rows = np.arange(ymin,ymax,0.005)
cols = np.arange(xmin,xmax,0.005)
geomatrix = []
for y in rows:
    for x in cols:
        coord = [x,y]
        geomatrix.append(coord)
geomatrix = np.array(geomatrix).reshape(45,45,2)
matrix = []
polygons = []
for j,r in enumerate(geomatrix):
    if j > 43:
        break
    for i,c in enumerate(r):
        if i > 43:
            continue
        else:
            matrix.append([geomatrix[j+1][i],geomatrix[j+1][i+1],r[i+1
],c])
            polygons.append(Polygon([geomatrix[j+1][i],geomatrix[j+1][
i+1],r[i+1],c]))

crs = {'init': 'epsg:4326'}
grid = gpd.GeoDataFrame(matrix,crs = crs,geometry=polygons)
grid.columns = ['p1','p2','p3','p4','geometry']
#替每個網格加上 xy 座標
```

```
sta_xy = pd.DataFrame(columns = ["資料蒐集時間", "人數總和","站名","場站代號
","x","y"])
def corr_station(station):

    single_sta = time_selec[time_selec['站名']==station]
    index_out = col_name[col_name['捷運站名稱']=='頂溪'].index
    lon = col_name[col_name['捷運站名稱']==station]['經度'].item()
    lat = col_name[col_name['捷運站名稱']==station]['緯度'].item()
    target = np.array([lon,lat])
    x = np.tile(target[0],277109)
    y = np.tile(target[1],277109)
    single_sta['x'] = list(x)
    single_sta['y'] = list(y)

    return single_sta
for i in range(len(col_name)):
    single_sta = corr_station(col_name['捷運站名稱'][i])
    sta_xy = pd.concat([sta_xy,single_sta])

sta_xy = sta_xy.sort_values(by=['資料蒐集時間','場站代號
'], ascending=True)
sta_xy.index = range(len(sta_xy))
geometry = [Point(xy) for xy in zip(sta_xy.x, sta_xy.y)]
target_gis = gpd.GeoDataFrame(sta_xy, crs=crs, geometry=geometry)
target_gis_save=target_gis.to_csv("target_gis.csv",encoding='utf-8')#存
檔

#空間結合 spatial join
col_name = pd.read_csv("D:/NTU/臺北市交通局授權_悠遊卡進出站資料/捷運場站名稱
3.csv",encoding='big5')
index_right = pd.read_csv("D:/NTU/臺北市交通局授權_悠遊卡進出站資料
/index_right.csv",encoding='utf-8',header=None)
target_gis = pd.read_csv('target_gis.csv',encoding='utf-8')
target_gis['資料蒐集時間'] = pd.to_datetime(target_gis['資料蒐集時間
'],format='%Y-%m-%d %H:%M:%S')
target_gis.drop(['Unnamed: 0'], axis=1, inplace=True)
#spatial join
join = gpd.sjoin(target_gis,grid,how='inner',op='within')
```

```
join.sort_values(by=['資料蒐集時間','場站代號
'], ascending=True, inplace=True)
#取得捷運站在 0 矩陣中的位置
index_right = join[:108].index_right
index_right_save=index_right.to_csv("index_right.csv",encoding='utf-
8')
#運量資料加入網格
target_gis = target_gis.join(col_name.set_index('捷運站名稱'),on='站名')
data_matrix = []
for i in range(1,int((len(target_gis)+1)/108)+1):
    # print(i)
    npmatrix_test = np.zeros(1936)
    npmatrix_test[[index_right[1]]] = target_gis.人數總和[108*(i-
1):108*i]
    if sum(npmatrix_test) >= 0:
        npmatrix_test = npmatrix_test.reshape(44,44)
        data_matrix.extend([npmatrix_test,target_gis.資料蒐集時間
[108*(i-1)]])
    else:pass
data_matrix = np.array(data_matrix)
np.save('data_matrix', data_matrix)
#篩選運量矩陣
rider_matrix=[]
for i in range(0,len(data_matrix),2):
    rider_matrix.append(data_matrix[i])
rider_matrix = np.array(rider_matrix)
np.save('rider_matrix',rider_matrix)
#特殊節日處理
target_gis = pd.read_csv('target_gis.csv',encoding='utf-8')
target_gis['資料蒐集時間'] = pd.to_datetime(target_gis['資料蒐集時間
'],format='%Y-%m-%d %H:%M:%S')
target_gis.drop(['Unnamed: 0'], axis=1, inplace=True)
holiday = pd.read_csv('政府行政機關辦公日曆表
_000291815490406155525 9.csv',encoding='utf-8')
holiday['活動日期'] = pd.to_datetime(holiday['活動日期'],format='%Y-%m-
%d')
holiday.drop(['name','description'], axis=1, inplace=True)
holiday['isHoliday'].replace({'否':0,'是':1},inplace = True)
```

```
event = pd.read_excel('預先嘗試//日期分割_complete.xls',names=['活動日期
','活動名稱'])
event['活動日期'] = pd.to_datetime(event['活動日期'],format='%Y-%m-%d')
event.sort_values(by=['活動日期'], ascending=True, inplace=True)
event.index =range(len(event))
special_event = pd.merge(event,holiday,how='outer',on='活動日期
',sort=True)
special_event['isHoliday'].fillna(1,inplace=True)
not_holiday = special_event[special_event['isHoliday']==0]
special_event = special_event[special_event['isHoliday']==1]
special_event.drop(['holidayCategory','活動名稱'],axis=1,inplace=True)
special_event.index=range(len(special_event))
#蒐集特殊日時間
data_matrix = np.load('data_matrix.npy',allow_pickle=True)
time_list=[]
for i in range(1,len(data_matrix),2):
    time_list.append(data_matrix[i])
time_list1 = pd.DataFrame(time_list)
time_list1[0] =time_list1[0].dt.strftime('%Y-%m-%d %H:%M:%S')
special_event['活動日期'] =special_event['活動日期'].dt.strftime('%Y-%m-
%d')
holiday_time = pd.DataFrame()
for i in range(len(special_event['活動日期'])):
    holiday_five = time_list1[time_list1[0].str.contains(special_event
['活動日期'][i])]
    holiday_time = pd.concat([holiday_time,holiday_five])
holiday_time_save=holiday_time.to_csv('holiday_time.csv',encoding='utf
-8')
#運量加入平日與特殊日特徵值
rider_matrix = np.load('rider_matrix.npy',allow_pickle=True)
def feature_normal(a):
    rider_matrix[a][42][43]+=200
    rider_matrix[a][41][42]+=200
    rider_matrix[a][41][43]+=200
    rider_matrix[a][40][41]+=200
    rider_matrix[a][40][42]+=200
    rider_matrix[a][40][43]+=200
def feature_special(b):
```

```
    rider_matrix[b][41][40]+=200
    rider_matrix[b][42][40]+=200
    rider_matrix[b][42][41]+=200
    rider_matrix[b][43][40]+=200
    rider_matrix[b][43][41]+=200
    rider_matrix[b][43][42]+=200
all_index=list(range(0,277109))
normal_index = list(set(all_index)-set(holiday_index))
for i in holiday_index:
    feature_special(i)
for i in normal_index:
    feature_normal(i)
np.save('rider_matrix_feature',rider_matrix)
np.save('holiday_index',holiday_index)
#製作 108 站一維運量
target_gis = pd.read_csv('/content/gdrive/My Drive/target_gis.csv')
target_gis.drop(['Unnamed: 0','x','y'],axis=1,inplace=True)
oneD_rider=[]
for i in range(0,len(target_gis['人數總和']),108):
  one_step=target_gis['人數總和'][i:i+108].values
  oneD_rider.append(one_step)
np.save('/content/gdrive/My Drive/oneD_rider',oneD_rider)
#製作輸入/出資料集
rider_matrix_feature = np.load('/content/gdrive/My Drive/rider_matrix_
feature.npy',allow_pickle=True)
oneD_rider = np.load('/content/gdrive/My Drive/oneD_rider.npy',allow_p
ickle=True)

##取三年資料長度
rider_matrix_feature = rider_matrix_feature[:237832]
oneD_rider = oneD_rider[:237832]
#x data
rider_feature_3ts = []##創建空白 list 接收資料
for i in range(0,len(rider_matrix_feature)-7,3):
    sample = rider_matrix_feature[i:i+3]##[0:3]->[0,1,2]
    rider_feature_3ts.append(sample)

rider_feature_3ts = np.array(rider_feature_3ts)#轉換為 numpy.array
```

```
rider_feature_3ts = rider_feature_3ts.reshape(79275,3,1,44,44)#reshape
成五維向量

print(rider_feature_3ts.shape)##檢查資料大小是否正確

np.save('/content/gdrive/My Drive/rider_feature_3ts',rider_feature_3ts
)##存檔，前面放路徑，逗點後為要儲存的變數
#y data
oneD_rider_3ts_mean=[]##創建空白 list 接收資料

for i in range(0,len(oneD_rider)-7,3):
    answer = oneD_rider[i+3]+oneD_rider[i+4]+oneD_rider[i+5]##矩陣相加,
由[i+n]開始
    oneD_rider_3ts_mean.append(answer//3)##取平均，除以 n

oneD_rider_3ts_mean = np.array(oneD_rider_3ts_mean)#轉換為 numpy.array

print(oneD_rider_3ts_mean.shape)##檢查資料大小是否正確

np.save('/content/gdrive/My Drive/oneD_rider_3ts_mean',oneD_rider_3ts_
mean)##存檔，前面放路徑，逗點後為要儲存的變數
```

附錄二 運量預測程式碼

```
#讀取資料
x = np.load('/content/gdrive/My Drive/x_feature.npy',allow_pickle=True
)
oneD_y = np.load('/content/gdrive/My Drive/oneD_rider_3ts_mean.npy',al
low_pickle=True)
#選取三年資料長度
x=x[:79276]
oneD_y = oneD_y[:79276]
#分割訓練、測試資料集
x_train4, x_test4, y_train4, y_test4 = train_test_split(x,oneD_y, test
_size=0.3, random_state = 1234)
#建立模型
model4 = tf.keras.Sequential()
###################### Input Layer
model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3),in
put_shape=(3,1,44,44)
                      , data_format='channels_first'
                      , recurrent_activation='hard_sigmoid'
                      , activation='tanh'
                      , padding='same'
                      , return_sequences=True))
model4.add(tf.keras.layers.BatchNormalization())
######################
model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                      , data_format='channels_first'
                      , recurrent_activation='hard_sigmoid'
                      , activation='tanh'
                      , padding='same'
                      , return_sequences=True))
model4.add(tf.keras.layers.BatchNormalization())
######################
model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                      , data_format='channels_first'
                      , recurrent_activation='hard_sigmoid'
                      , activation='tanh'
                      , padding='same'
```

```
                    , return_sequences=True))
model4.add(tf.keras.layers.BatchNormalization())
######################
model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                    , data_format='channels_first'
                    , recurrent_activation='hard_sigmoid'
                    , activation='tanh'
                    , padding='same'
                    , return_sequences=True))
model4.add(tf.keras.layers.BatchNormalization())
#####################   Output Layer
model4.add(tf.keras.layers.Flatten(data_format='channels_first'))
model4.add(tf.keras.layers.Dense(108))
####################
opt = tf.keras.optimizers.Adam(lr=0.01)
model4.compile(loss='mse',metrics=['accuracy', 'mse'],optimizer=opt)
model4.summary()
####################
early_stopping = EarlyStopping(monitor='val_loss', mode='min', verbose
=1)
model4.fit(x_train4, y_train4, batch_size=100, epochs=10, validation_s
plit=0.3,verbose=1,callbacks=[early_stopping])
model4.fit(x_train4, y_train4, batch_size=100, epochs=10, validation_d
ata=(x_test4,y_test4),verbose=1,callbacks=[early_stopping])
score = model4.evaluate(x_test4, y_test4)
print('loss:', score[0])
print('正確率', score[1])
#利用測試資料集進行預測
predict = model4.predict(x_test4)
predict = predict.round()
#保存模型權重
model_json = model4.to_json()
open('/content/gdrive/My Drive/convlstm_model4_arch.json',
     'w').write(model_json)
model4.save('/content/gdrive/My Drive/model4.h5')  # creates a HDF5 fi
le 'my_model.h5'
```

附錄三 資料驗證程式碼

```
#資料驗證
#讀入模型權重
model4 = keras.models.model_from_json(open('/content/gdrive/My Drive/c
onvlstm_model4_arch.json').read())
model4.load_weights('/content/gdrive/My Drive/model4.h5')
model4.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=0.01)
)
#讀入資料
x = np.load('/content/gdrive/My Drive/x_feature.npy',allow_pickle=True
)
oneD_y = np.load('/content/gdrive/My Drive/oneD_rider_3ts_mean.npy',al
low_pickle=True)
x=x[:79276]
oneD_y = oneD_y[:79276]

x_train4, x_test4, y_train4, y_test4 = train_test_split(x,oneD_y, test
_size=0.3, random_state = 1234)
predict = model4.predict(x_test4)
predict = predict.round()
#RMSECV 相對誤差
def mean_relative_error(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())/targets.mean
()
mean_relative_error(predict,y_test4)
#不同時間 RMSE 比較
dist_re=[]
for i in range(len(predict)):
  single = mean_relative_error(predict[i],y_test4[i])
  dist_re.append(single)
dist_re = np.array(dist_re)*100
dist_re = dist_re.tolist()
#篩選好/壞預測結果
bad_pred=[]
good_pred=[]

for i in range(len(dist_re)):
```

```
  if dist_re[i] >20:
    bad_pred.append(dist_re.index(dist_re[i]))
  elif (dist_re[i] >= 0 and dist_re[i]<20):
    good_pred.append(dist_re.index(dist_re[i]))
#繪圖
plt.figure(figsize=(10,10))

plt.xlabel("CV(rmse)" , size=20)
plt.ylabel('kernel density',size=20)

plt.xticks(size=15)
plt.yticks(size=15)
sns.distplot(dist_re)
plt.legend('%')
plt.savefig('/content/gdrive/My Drive/all_station_cv.jpg')
#讀取特殊日/平日運量
holiday_matrix_3ts_3yr = np.load('/content/gdrive/My Drive/holiday_mat
rix_3ts_3yr.npy',allow_pickle=True)
holiday_oneD_y = np.load('/content/gdrive/My Drive/holiday_oneD_y.npy'
,allow_pickle=True)
normal_matrix_3ts = np.load('/content/gdrive/My Drive/normal_matrix_3t
s.npy',allow_pickle=True)
normal_oneD_y = np.load('/content/gdrive/My Drive/normal_oneD_y.npy',a
llow_pickle=True)
#特殊日/平日運量預測
predict_holiday = model4.predict(holiday_matrix_3ts_3yr)
mean_relative_error(predict_holiday,holiday_oneD_y)#整體 RMSECV
#不同 timesteps
spe_cv=[]

for i in range(len(predict_holiday)):
  single = mean_relative_error(predict_holiday[i],holiday_oneD_y[i])
  spe_cv.append(single)

spe_cv = np.array(spe_cv)*100
spe_cv = spe_cv.tolist()
spe_bad_pred=[]
spe_good_pred=[]
```

```
for i in range(len(spe_cv)):
  if spe_cv[i] >20:
    spe_bad_pred.append(spe_cv.index(spe_cv[i]))
  elif (spe_cv[i] >= 0 and spe_cv[i]<20):
    spe_good_pred.append(spe_cv.index(spe_cv[i]))
plt.figure(figsize=(10,10))
plt.xlabel("rmse" , size=20)
plt.ylabel('kernel density',size=20)
plt.xticks(size=15)
plt.yticks(size=15)
sns.distplot(spe_cv,color='r')
plt.savefig('/content/gdrive/My Drive/spe_cv.jpg')
#平日
predict_normalday = model4.predict(normal_matrix_3ts)
mean_relative_error(predict_normalday,normal_oneD_y)
nor_cv=[]

for i in range(len(predict_normalday)):
  single = mean_relative_error(predict_normalday[i],normal_oneD_y[i])
  nor_cv.append(single)

nor_cv = np.array(nor_cv)*100
nor_cv = nor_cv.tolist()
nor_bad_pred=[]
nor_good_pred=[]

for i in range(len(nor_cv)):
  if nor_cv[i] >20:
    nor_bad_pred.append(nor_cv.index(nor_cv[i]))
  elif (nor_cv[i] >= 0 and nor_cv[i]<20):
    nor_good_pred.append(nor_cv.index(nor_cv[i]))
plt.figure(figsize=(10,10))
plt.xlabel("rmse" , size=20)
plt.ylabel('kernel density',size=20)
plt.xticks(size=15)
plt.yticks(size=15)
sns.distplot(nor_cv,color='g')
```

```
plt.savefig('/content/gdrive/My Drive/nor_cv.jpg')
#各站不同時間的 RMSECV
def single_sta_rmse(sta):

  single_sta = []
  for i in range(len(y_test4)):
    single_sta.append(rmse(predict[i][sta],y_test4[i][sta]))
  single_sta = np.array(single_sta)
  return single_sta

single_station=[]

for i in range(108):
  single_station.append(single_sta_rmse(i))
single_station = np.array(single_station)
single_rmse=[]

for i in range(108):
  single_rmse.append(np.average(single_station[i]))
rmse_matrix = []

for i in range(108):
  loc=index_right[1][i]
  npmatrix_test = np.zeros(1936)
  npmatrix_test[loc] = single_rmse[i]
  npmatrix_test = npmatrix_test.reshape(44,44)
  rmse_matrix.extend([npmatrix_test])
rmse_matrix = np.array(rmse_matrix)

for i in range(1,108):
  rmse_matrix[0]+=rmse_matrix[i]
#畫圖
plt.figure(figsize=(10,10))
colors.Normalize(rmse_matrix[0].min(),rmse_matrix[0].max())
fig = plt.matshow(np.flipud(rmse_matrix[0]),fignum=1,cmap='gray')
ax = plt.gca()
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
```

```
cb = plt.colorbar(fig, cax=cax)

tick=[0]
for i in range(0,11):
    a = rmse_matrix[0].max()/100
    a = math.ceil(a)
    a = round(10*a*(i+1))
    tick.append(a)


cb.set_ticks(tick)
cb.ax.tick_params(labelsize=15)
#篩選預測不佳場站
bad_single_value=[]

for i in bad_single_rmse:
  bad_single_value.append(single_rmse[i])
sta_name['捷運站名稱'][bad_single_rmse]
high_rmse = []
for i in bad_single_rmse:
  loc=index_right[1][i]
  npmatrix_test = np.zeros(1936)
  npmatrix_test[loc] = single_rmse[i]
  npmatrix_test = npmatrix_test.reshape(44,44)
  high_rmse.extend([npmatrix_test])


high_rmse = np.array(high_rmse)
high_rmse.shape
for i in range(1,11):
  high_rmse[0]+=high_rmse[i]

plt.figure(figsize=(10,10))
colors.Normalize(high_rmse[0].min(),high_rmse[0].max())
fig = plt.matshow(np.flipud(high_rmse[0]),fignum=1,cmap='gray')
ax = plt.gca()
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cb = plt.colorbar(fig, cax=cax)
```

```
tick=[0]
for i in range(0,11):
    a = high_rmse[0].max()/100
    a = math.ceil(a)
    a = round(10*a*(i+1))
    tick.append(a)


cb.set_ticks(tick)
cb.ax.tick_params(labelsize=15)
plt.savefig('/content/gdrive/My Drive/high_rmse.jpg')

#半小時預測
six_ts_x= np.load('/content/gdrive/My Drive/X&Y data/rider_feature_6ts
.npy',allow_pickle=True)
six_ts_y = np.load('/content/gdrive/My Drive/X&Y data/oneD_rider_6ts.n
py',allow_pickle=True)
six_ts_x = six_ts_x.reshape(39637,6,1,44,44)
x_train6, x_test6, y_train6, y_test6 = train_test_split(six_ts_x,six_t
s_y, test_size=0.3, random_state = 1234)
model6 = keras.models.model_from_json(open('/content/gdrive/My Drive/c
onvlstm_model6_arch.json').read())
model6.load_weights('/content/gdrive/My Drive/model6.h5')
model6.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=0.001
))
model6 = tf.keras.Sequential()
###################### Input Layer
model6.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3),in
put_shape=(6,1,44,44)
                      , data_format='channels_first'
                      , recurrent_activation='hard_sigmoid'
                      , activation='tanh'
                      , padding='same'
                      , return_sequences=True))
model6.add(tf.keras.layers.BatchNormalization())
######################
model6.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                      , data_format='channels_first'
```

```
                    , recurrent_activation='hard_sigmoid'
                    , activation='tanh'
                    , padding='same'
                    , return_sequences=True))

model6.add(tf.keras.layers.BatchNormalization())
######################
model6.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                    , data_format='channels_first'
                    , recurrent_activation='hard_sigmoid'
                    , activation='tanh'
                    , padding='same'
                    , return_sequences=True))

model6.add(tf.keras.layers.BatchNormalization())
######################
model6.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                    , data_format='channels_first'
                    , recurrent_activation='hard_sigmoid'
                    , activation='tanh'
                    , padding='same'
                    , return_sequences=True))

model6.add(tf.keras.layers.BatchNormalization())
#####################   Output Layer
model6.add(tf.keras.layers.Flatten(data_format='channels_first'))
model6.add(tf.keras.layers.Dense(108))
####################
opt = tf.keras.optimizers.Adam(lr=0.001)
model6.compile(loss='mse',metrics=['accuracy', 'mse'],optimizer=opt)
early_stopping = EarlyStopping(monitor='val_loss', mode='min', verbose
=1)
model6.fit(x_train6, y_train6, batch_size=100, epochs=10, verbose=1, v
alidation_split=0.3,callbacks=[early_stopping])
model6.fit(x_train6, y_train6, batch_size=100, epochs=10, validation_d
ata=(x_test6,y_test6),verbose=1,callbacks=[early_stopping])
model_json = model6.to_json()
open('/content/gdrive/My Drive/convlstm_model6_arch.json',
```

```
    'w').write(model_json)
model6.save('/content/gdrive/My Drive/model6.h5')  # creates a HDF5 fi
le 'my_model.h5'
predict6 = model6.predict(x_test6,verbose=1)
model6.evaluate(x_test6,y_test6)
#30 分鐘運量比較圖
plt.figure(figsize=(20,10))
plt.title
plt.plot(y_test6[:100,41],color='indigo',label='Ground Truth', alpha=0
.75,linewidth=2.0)
plt.plot(predict6[:100,41],color='orangered',label='6 timesteps predic
tion',linewidth=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('Timesteps',fontsize=20)
plt.ylabel('Ridership',size=20)
plt.savefig('/content/gdrive/My Drive/圖/30 分鐘運量比
較.jpg', dpi=300,format='jpg')
#一小時運量預測
tw_ts_x= np.load('/content/gdrive/My Drive/X&Y data/rider_feature_12ts
.npy',allow_pickle=True)
tw_ts_y = np.load('/content/gdrive/My Drive/X&Y data/oneD_rider_12ts.n
py',allow_pickle=True)
tw_ts_x = tw_ts_x.reshape(19818,12,1,44,44)
x_train12, x_test12, y_train12, y_test12 = train_test_split(tw_ts_x,tw
_ts_y, test_size=0.3, random_state = 1234)
model12 = tf.keras.Sequential()
###################### Input Layer
model12.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3),i
nput_shape=(12,1,44,44)
                     , data_format='channels_first'
                     , recurrent_activation='hard_sigmoid'
                     , activation='tanh'
                     , padding='same'
                     , return_sequences=True))
model12.add(tf.keras.layers.BatchNormalization())
```

```
#####################
model12.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                     , data_format='channels_first'
                     , recurrent_activation='hard_sigmoid'
                     , activation='tanh'
                     , padding='same'
                     , return_sequences=True))

model12.add(tf.keras.layers.BatchNormalization())
#####################
model12.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                     , data_format='channels_first'
                     , recurrent_activation='hard_sigmoid'
                     , activation='tanh'
                     , padding='same'
                     , return_sequences=True))

model12.add(tf.keras.layers.BatchNormalization())
######################
model12.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                     , data_format='channels_first'
                     , recurrent_activation='hard_sigmoid'
                     , activation='tanh'
                     , padding='same'
                     , return_sequences=True))

model12.add(tf.keras.layers.BatchNormalization())
#####################   Output Layer
model12.add(tf.keras.layers.Flatten(data_format='channels_first'))
model12.add(tf.keras.layers.Dense(108))
####################
opt = tf.keras.optimizers.Adam(lr=0.01)
model12.compile(loss='mse',metrics=['accuracy', 'mse'],optimizer=opt)
early_stopping = EarlyStopping(monitor='val_loss', mode='min', verbose
=1)
model12.fit(x_train12, y_train12, batch_size=100, epochs=10, verbose=1
, validation_split=0.3,callbacks=[early_stopping])
```

```
model12.fit(x_train12, y_train12, batch_size=100, epochs=10, validatio
n_data=(x_test12,y_test12),verbose=1,callbacks=[early_stopping])
predict12 = model12.predict(x_test12)
model12.evaluate(x_test12,y_test12)
model_json = model12.to_json()
open('/content/gdrive/My Drive/convlstm_model12_arch.json',
     'w').write(model_json)
model12.save('/content/gdrive/My Drive/model12.h5')  # creates a HDF5
file 'my_model.h5'
#畫圖
plt.figure(figsize=(20,10))
plt.plot(y_test12[:100,41],color='indigo',label='Ground Truth', alpha=
0.75,linewidth=2.0)
plt.plot(predict12[:100,41],color='orangered',label='12 timesteps pred
iction',linewidth=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('Timesteps',fontsize=20)
plt.ylabel('Ridership',size=20)
plt.savefig('/content/gdrive/My Drive/圖/60 分鐘運量比
較.jpg', dpi=300,format='jpg')

#K-cross validation
model4 = keras.models.model_from_json(open('/content/gdrive/My Drive/c
onvlstm_model4_arch.json').read())
model4.load_weights('/content/gdrive/My Drive/model4.h5')
model4.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=0.01)
)
x = np.load('/content/gdrive/My Drive/x_feature.npy',allow_pickle=True
)
oneD_y = np.load('/content/gdrive/My Drive/oneD_rider_3ts_mean.npy',al
low_pickle=True)

x=x[:79275]
oneD_y = oneD_y[:79275]
```

```
x_train4, x_test4, y_train4, y_test4 = train_test_split(x,oneD_y, test
_size=0.3, random_state = 1234)
kf = KFold(10 , random_state=1234)

oos_y = []
oos_pred = []
eva=[]
score=[]

fold = 0
for train, test in kf.split(x,oneD_y):
  fold+=1
  print(f"Fold #{fold}")

  x_train = x[train]
  y_train = oneD_y[train]
  x_test = x[test]
  y_test = oneD_y[test]

  model4 = tf.keras.Sequential()
  ###################### Input Layer
  model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3),
input_shape=(3,1,44,44)
                      , data_format='channels_first'
                      , recurrent_activation='hard_sigmoid'
                      , activation='tanh'
                      , padding='same'
                      , return_sequences=True))
  model4.add(tf.keras.layers.BatchNormalization())
  ####################
  model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                      , data_format='channels_first'
                      , recurrent_activation='hard_sigmoid'
                      , activation='tanh'
                      , padding='same'
                      , return_sequences=True))

  model4.add(tf.keras.layers.BatchNormalization())
```

```
  ######################
  model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                      , data_format='channels_first'
                      , recurrent_activation='hard_sigmoid'
                      , activation='tanh'
                      , padding='same'
                      , return_sequences=True))

  model4.add(tf.keras.layers.BatchNormalization())
  ######################
  model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                      , data_format='channels_first'
                      , recurrent_activation='hard_sigmoid'
                      , activation='tanh'
                      , padding='same'
                      , return_sequences=True))

  model4.add(tf.keras.layers.BatchNormalization())
  ######################   Output Layer
  model4.add(tf.keras.layers.Flatten(data_format='channels_first'))
  model4.add(tf.keras.layers.Dense(108))
  ####################
  opt = tf.keras.optimizers.Adam(lr=0.01)
  model4.compile(loss='mse',metrics=['accuracy', 'mse'],optimizer=opt)
  ####################
  early_stopping = EarlyStopping(monitor='val_loss', mode='min', verbo
se=1)
  model4.fit(x_train, y_train, batch_size=100, epochs=10, validation_d
ata=(x_test,y_test),verbose=1,callbacks=[early_stopping])

  pred_y = model4.predict(x_test)
  eva.append(model4.evaluate(x_test,y_test)[1])

  oos_y.append(y_test)
  oos_pred.append(pred_y)

  # Measure this fold's RMSE
  score.append(np.sqrt(metrics.mean_squared_error(pred_y,y_test)))
```

```
  print(f"Fold score (RMSE): {score}")
  print(eva)

  del(x_train ,y_train,x_test,y_test)

# Build the oos prediction list and calculate the error.
oos_y = np.concatenate(oos_y)
oos_pred = np.concatenate(oos_pred)
final_score = np.sqrt(metrics.mean_squared_error(oos_pred,oos_y))
print(f"Final, out of sample score (RMSE): {final_score}")

#無特徵值預測
x = np.load('/content/gdrive/My Drive/x.npy',allow_pickle=True)
y = np.load('/content/gdrive/My Drive/oneD_rider_3ts_mean.npy',allow_p
ickle=True)
x=x[:79275]
y = y[:79275]
kf = KFold(10 , random_state=1234)

oos_y = []
oos_pred = []
eva=[]
score=[]

fold = 0
for train, test in kf.split(x,y):
  fold+=1
  print(f"Fold #{fold}")

  x_train = x[train]
  y_train = y[train]
  x_test = x[test]
  y_test = y[test]

  model4 = tf.keras.Sequential()
  ###################### Input Layer
  model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3),
input_shape=(3,1,44,44)
```

```
                    , data_format='channels_first'
                    , recurrent_activation='hard_sigmoid'
                    , activation='tanh'
                    , padding='same'
                    , return_sequences=True))
model4.add(tf.keras.layers.BatchNormalization())
#####################
model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                    , data_format='channels_first'
                    , recurrent_activation='hard_sigmoid'
                    , activation='tanh'
                    , padding='same'
                    , return_sequences=True))

model4.add(tf.keras.layers.BatchNormalization())
#####################
model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                    , data_format='channels_first'
                    , recurrent_activation='hard_sigmoid'
                    , activation='tanh'
                    , padding='same'
                    , return_sequences=True))

model4.add(tf.keras.layers.BatchNormalization())
######################
model4.add(tf.keras.layers.ConvLSTM2D(filters=20, kernel_size=(3,3)
                    , data_format='channels_first'
                    , recurrent_activation='hard_sigmoid'
                    , activation='tanh'
                    , padding='same'
                    , return_sequences=True))

model4.add(tf.keras.layers.BatchNormalization())
#####################   Output Layer
model4.add(tf.keras.layers.Flatten(data_format='channels_first'))
model4.add(tf.keras.layers.Dense(108))
####################
opt = tf.keras.optimizers.Adam(lr=0.01)
```

```
  model4.compile(loss='mse',metrics=['accuracy', 'mse'],optimizer=opt)
  ####################
  early_stopping = EarlyStopping(monitor='val_loss', mode='min', verbo
se=1)
  model4.fit(x_train, y_train, batch_size=100, epochs=10, validation_d
ata=(x_test,y_test),verbose=1,callbacks=[early_stopping])

  pred_y = model4.predict(x_test)
  eva.append(model4.evaluate(x_test,y_test)[1])

  oos_y.append(y_test)
  oos_pred.append(pred_y)

  # Measure this fold's RMSE
  score.append(np.sqrt(metrics.mean_squared_error(pred_y,y_test)))
  print(f"Fold score (RMSE): {score}")
  print(eva)

  del(x_train ,y_train,x_test,y_test)

# Build the oos prediction list and calculate the error.
oos_y = np.concatenate(oos_y)
oos_pred = np.concatenate(oos_pred)
final_score = np.sqrt(metrics.mean_squared_error(oos_pred,oos_y))
print(f"Final, out of sample score (RMSE): {final_score}")
model_json = model4.to_json()
open('/content/gdrive/My Drive/model4_nonfeature.json',
     'w').write(model_json)
model4.save('/content/gdrive/My Drive/model4_nonfeature.h5')  # create
s a HDF5 file 'my_model.h5'

#大型活動驗證
#2018-2019跨年活動
#讀取資料
index_right = pd.read_csv("/content/gdrive/My Drive/index_right.csv",h
eader=None)
index_right.drop(0,axis=1,inplace=True)
```

```
model4 = keras.models.model_from_json(open('/content/gdrive/My Drive/c
onvlstm_model4_arch.json').read())
model4.load_weights('/content/gdrive/My Drive/model4.h5')
model4.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=0.001
))
ny_x = np.load('/content/gdrive/My Drive/new_year_3ts.npy',allow_pickl
e=True)
ny_y = np.load('/content/gdrive/My Drive/newyear_3ts_y.npy',allow_pick
le=True)
sta_name = pd.read_csv('/content/gdrive/My Drive/捷運場站名稱
4.csv',encoding='utf-8')
sta_name.drop(['Unnamed: 0'],axis=1,inplace=True)
#預測
predict_ny = model4.predict(ny_x)
predict_ny = predict_ny.round()
for i in range(len(predict_ny)):
  for j in range(0,108):
    if predict_ny[i][j]<0:
      predict_ny[i][j]=0
    elif 1>predict_ny[i][j]>0:
      predict_ny[i][j]=1
predict_ny = np.array(predict_ny)
ny_y = np.array(ny_y)
#RMSE CV
def mean_relative_error(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())/targets.mean
()*100
cv_rmse=[]
for i in range(166):
  single = mean_relative_error(predict_ny[i],ny_y[i])
  cv_rmse.append(single))
#畫圖-全站運量 RMSE-CV 折線圖
ticks_3=np.arange(0,176,8)
label3=['06:00','08:00','10:00','12:00'
,'14:00','16:00','18:00','20:00','22:00','00:00'
,'02:00','04:00','06:00','08:00','10:00','12:00'
,'14:00','16:00','18:00','20:00','22:00','00:00']
```

```
plt.figure(figsize=(20,10))
plt.title('New Year all station CV(RMSE)',fontsize=20)
plt.plot(cv_rmse,color='red',label='CV(RMSE)')
plt.xticks(ticks_3,label3,fontsize=15)
plt.yticks(fontsize=15)
plt.grid(alpha=0.75)
plt.legend(loc='best',fontsize=15)
plt.xlabel("timesteps",fontsize=20)
plt.ylabel("RMSE",fontsize=20)
plt.savefig('/content/gdrive/My Drive/new_year_cv.jpg')
#跨年周邊站點驗證
single_sta=[]

for i in range(108):
  single = mean_relative_error(sum(predict_ny[:,i])/166,sum(ny_y[:,i])
/166)
  single_sta.append(single)
print(len(single_sta))

sta_name = pd.read_csv('/content/gdrive/My Drive/捷運場站名稱
4.csv',encoding='utf-8')
sta_name.drop(['Unnamed: 0'],axis=1,inplace=True)
sta_name['rmse_cv']=single_sta
sta_name['std']=sta_std
sta_name['ny_std']=nt_std
sta_name.sort_values(by=['ny_std'],ascending=False)[:40]

ny_sta_num=[75,76,77,81,82,83,88]
for i in ny_sta_num:
  print(i,single_sta[i])
#預測不佳場站
for i in range(108):
  if single_sta[i] >20:
    print(i,single_sta[i])
#跨年場站運量比較折線圖
label_ny=['06:00','12:00','18:00','00:00','06:00','12:00','18:00','00:
00']
ticks_ny=np.arange(0,192,24)
```

```
plt.figure(figsize=(30,20))
plt.subplot(4, 3 ,1)
plt.title('SYS Memorial Hall predict Ridership',fontsize=20)
plt.plot(ny_y[:,75],color='indigo',label='Ground Truth', alpha=0.75,li
newidth=2.0)
plt.plot(predict_ny[:,75],color='teal',label='Predict Values',linewidt
h=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(ticks_ny,label_ny,fontsize=20)
plt.yticks(fontsize=15)
plt.xlabel('Time',fontsize=20)
plt.ylabel('Ridership',size=20)
########################
plt.subplot(4, 3 ,2)
plt.title('Taipei city hall predict Ridership',fontsize=20)
plt.plot(ny_y[:,76],color='indigo',label='Ground Truth', alpha=0.75,li
newidth=2.0)
plt.plot(predict_ny[:,76],color='teal',label='Predict Values',linewidt
h=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(ticks_ny,label_ny,fontsize=20)
plt.yticks(fontsize=15)
plt.xlabel('Time',fontsize=20)
plt.ylabel('Ridership',size=20)
########################
plt.subplot(4, 3 ,3)
plt.title('Yongchun predict Ridership',fontsize=20)
plt.plot(ny_y[:,77],color='indigo',label='Ground Truth', alpha=0.75,li
newidth=2.0)
plt.plot(predict_ny[:,77],color='teal',label='Predict Values',linewidt
h=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(ticks_ny,label_ny,fontsize=20)
plt.yticks(fontsize=15)
```

```
plt.xlabel('Time',fontsize=20)
plt.ylabel('Ridership',size=20)
########################
plt.subplot(4, 3 ,4)
plt.title('XiangShan predict Ridership',fontsize=20)
plt.plot(ny_y[:,81],color='indigo',label='Ground Truth', alpha=0.75,li
newidth=2.0)
plt.plot(predict_ny[:,81],color='teal',label='Predict Values',linewidt
h=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(ticks_ny,label_ny,fontsize=20)
plt.yticks(fontsize=15)
plt.xlabel('Time',fontsize=20)
plt.ylabel('Ridership',size=20)
########################
plt.subplot(4, 3 ,5)
plt.title('Taipei 101 predict Ridership',fontsize=20)
plt.plot(ny_y[:,82],color='indigo',label='Ground Truth', alpha=0.75,li
newidth=2.0)
plt.plot(predict_ny[:,82],color='teal',label='Predict Values',linewidt
h=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(ticks_ny,label_ny,fontsize=20)
plt.yticks(fontsize=15)
plt.xlabel('Time',fontsize=20)
plt.ylabel('Ridership',size=20)
########################
plt.subplot(4, 3 ,6)
plt.title('Xinyi Anhe predict Ridership',fontsize=20)
plt.plot(ny_y[:,83],color='indigo',label='Ground Truth', alpha=0.75,li
newidth=2.0)
plt.plot(predict_ny[:,83],color='teal',label='Predict Values',linewidt
h=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(ticks_ny,label_ny,fontsize=20)
```

```
plt.yticks(fontsize=15)
plt.xlabel('Time',fontsize=20)
plt.ylabel('Ridership',size=20)
########################
plt.subplot(4, 3,7)
plt.title('Nanjing Sanmin predict Ridership',fontsize=20)
plt.plot(ny_y[:,88],color='indigo',label='Ground Truth', alpha=0.75,li
newidth=2.0)
plt.plot(predict_ny[:,88],color='teal',label='Predict Values',linewidt
h=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(ticks_ny,label_ny,fontsize=20)
plt.yticks(fontsize=15)
plt.xlabel('Time',fontsize=20)
plt.ylabel('Ridership',size=20)
plt.tight_layout()
plt.savefig('/content/gdrive/My Drive/圖/101 運量比
較.jpg', dpi=300,format='jpg')

#有無特徵值運量比較
model4 = keras.models.model_from_json(open('/content/gdrive/My Drive/c
onvlstm_model4_arch.json').read())
model4.load_weights('/content/gdrive/My Drive/model4.h5')
model4.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=0.01)
)
model5 = keras.models.model_from_json(open('/content/gdrive/My Drive/m
odel4_nonfeature.json').read())
model5.load_weights('/content/gdrive/My Drive/model4_nonfeature.h5')
model5.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=0.01)
)
x_fea = np.load('/content/gdrive/My Drive/x_feature.npy',allow_pickle=
True)
oneD_y = np.load('/content/gdrive/My Drive/oneD_rider_3ts_mean.npy',al
low_pickle=True)
x_non = np.load('/content/gdrive/My Drive/x.npy',allow_pickle=True)

x_fea = x_fea[:79275]
```

```
x_non = x_non[:79275]
oneD_y = oneD_y[:79275]
x_train4, x_test4, y_train4, y_test4 = train_test_split(x_fea,oneD_y,
test_size=0.3, random_state = 1234)
x_train5, x_test5, y_train5, y_test5 = train_test_split(x_non,oneD_y,
test_size=0.3, random_state = 1234)
pred_y_fea=model4.predict(x_test4)
pred_y_non=model5.predict(x_test5)
#15 分鐘運量比較圖
plt.figure(figsize=(20,10))
plt.plot(y_test4[:100,41],color='indigo',label='Ground Truth', alpha=0
.75,linewidth=2.0)
plt.plot(pred_y_fea[:100,41],color='orangered',label='3 timesteps pred
iction',linewidth=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('Timesteps',fontsize=20)
plt.ylabel('Ridership',size=20)
plt.savefig('/content/gdrive/My Drive/圖/15 分鐘運量比
較.jpg', dpi=300,format='jpg')

#有無特徵運量比較
plt.figure(figsize=(10,10))
plt.subplot(2, 1 ,1)
plt.title('Taipei train station with day-
type feature Prediction',fontsize=20)
plt.plot(y_test4[:100,41],color='indigo',label='Ground Truth', alpha=0
.75,linewidth=2.0)
plt.plot(pred_y_fea[:100,41],color='orangered',label='Predict Values',
linewidth=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('Timesteps',fontsize=20)
plt.ylabel('Ridership',size=20)
```

```
########################
plt.subplot(2, 1 ,2)
plt.title('Taipei train station without day-
type feature Prediction',fontsize=20)
plt.plot(y_test4[:100,41],color='indigo',label='Ground Truth', alpha=0
.75,linewidth=2.0)
plt.plot(pred_y_non[:100,41],color='teal',label='Predict Values',linew
idth=2.0)
plt.legend(loc = 'best',fontsize=15)
plt.grid(axis='y', alpha=0.75)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('Timesteps',fontsize=20)
plt.ylabel('Ridership',size=20)
plt.tight_layout()
plt.savefig('/content/gdrive/My Drive/圖/有無特徵運量比
較.jpg', dpi=300,format='jpg')

#特殊日矩陣圖
plt.figure(figsize=(15,15))
colors.Normalize(x_fea[0][0][0].min(),x_fea[0][0][0].max())
fig = plt.matshow(np.flipud(x_fea[0][0][0]),fignum=1,cmap='gist_earth'
)
ax = plt.gca()
plt.title('2016-01-
01 06:00:00',fontsize=20,color='black',loc='center')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cb = plt.colorbar(fig, cax=cax)

tick=[0]
for i in range(0,11):
    a = x_fea[0][0][0].max()/100
    a = math.ceil(a)
    a = round(10*a*(i+1))
    tick.append(a)


cb.set_ticks(tick)
```

```
cb.ax.tick_params(labelsize=15)
plt.savefig('/content/gdrive/My Drive/圖/special_day_feature.jpg')

#平常日矩陣圖
plt.figure(figsize=(15,15))
colors.Normalize(x_fea[650][0][0].min(),x_fea[650][0][0].max())
fig = plt.matshow(np.flipud(x_fea[650][0][0]),fignum=1,cmap='gist_eart
h')
ax = plt.gca()
plt.title('2016-01-
04 06:00:00',fontsize=20,color='black',loc='center')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cb = plt.colorbar(fig, cax=cax)

tick=[0]
for i in range(0,11):
    a = x_fea[650][0][0].max()/100
    a = math.ceil(a)
    a = round(10*a*(i+1))
    tick.append(a)
cb.set_ticks(tick)
cb.ax.tick_params(labelsize=15)
plt.savefig('/content/gdrive/My Drive/圖/normal_day_feature.jpg')

#林俊傑演唱會驗證
model4 = keras.models.model_from_json(open('/content/gdrive/My Drive/c
onvlstm_model4_arch.json').read())
model4.load_weights('/content/gdrive/My Drive/model4.h5')
model4.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=0.001
))
#產生 x、y data
rider_feature_3ts = []##創建空白 list 接收資料
for i in range(24955,25606-6,3):
    sample = rider_matrix_feature[i:i+3]##[0:3]->[0,1,2]
    rider_feature_3ts.append(sample)

rider_feature_3ts = np.array(rider_feature_3ts)#轉換為 numpy.array
```

```
rider_feature_3ts = rider_feature_3ts.reshape(215,3,1,44,44)#reshape 成五維向量

print(rider_feature_3ts.shape)##檢查資料大小是否正確

np.save('/content/gdrive/My Drive/JJ_concert_3ts',rider_feature_3ts)##存檔，前面放路徑，逗點後為要儲存的變數

############################################################

oneD_rider_3ts_mean=[]##創建空白 list 接收資料

for i in range(24955,25606-6,3):
    answer = oneD_rider[i+3]+oneD_rider[i+4]+oneD_rider[i+5]##矩陣相加,由[i+n]開始
    oneD_rider_3ts_mean.append(answer//3)##取平均，除以 n

oneD_rider_3ts_mean = np.array(oneD_rider_3ts_mean)#轉換為 numpy.array

print(oneD_rider_3ts_mean.shape)##檢查資料大小是否正確

np.save('/content/gdrive/My Drive/JJ_concert_oneD_3ts',oneD_rider_3ts_mean)##存檔，前面放路徑，逗點後為要儲存的變數
#利用權重預測
pred_jj = model4.predict(jj_rider)
pred_jj = pred_jj.round()
for i in range(len(pred_jj)):
  for j in range(0,108):
    if pred_jj[i][j]<0:
      pred_jj[i][j]=0
    elif 1>pred_jj[i][j]>0:
      pred_jj[i][j]=1
pred_jj = np.array(pred_jj)

#演唱會運量比較折線圖
label_jj=['06:00','12:00','18:00','00:00','12:00','18:00','00:00','12:00','18:00','00:00','12:00','18:00','00:00','12:00','16:00','00:00']
ticks_jj=np.arange(0,302,24)
```

```
plt.figure(figsize=(20,10))
plt.title('2019/02/14-17 JJ Lin concert',fontsize=20)
plt.plot(jj_oneD_rider[:,87],color='indigo',label='Ground Truth', alph
a=0.75,linewidth=2.0)
plt.plot(pred_jj[:,87],color='teal',label='3 timesteps prediction',lin
ewidth=2.0)
plt.axvline(72,0, 2, linestyle= '--',color='red')
plt.axvline(144,0, 2, linestyle= '--',color='red')
plt.axvline(216,0, 2, linestyle= '--',color='red')
plt.legend(loc = 'best',fontsize=15)
plt.grid(alpha=0.75)
plt.xticks(ticks_jj,label_jj,fontsize=20)
plt.yticks(fontsize=15)
plt.xlabel('Time',fontsize=20)
plt.ylabel('Ridership',size=20)
plt.savefig('/content/gdrive/My Drive/圖/JJ Lin concert.jpg',dpi=300)
```