

國立嘉義大學應用數學系
專題製作

Department of Applied Mathematics
National Chiayi University
Project on Mathematics

運用深度學習模型辨識
經傳立葉處理之圖片

學 生：陳範耘、潘陣恩
李涵碩、黃尚崑

指導老師：鄭富國 老師

中 華 民 國 112 年 1 月

目錄

壹、序論	3
一、研究動機及目的	3
二、致謝	5
貳、研究內容	7
一、選擇及使用工具	7
1. Python	7
2. PyCharm	8
3. Spyder	9
4. TensorFlow	11
5. CIFAR-100	12
6. VGG 16	13
二、開發環境	15
1. 模型訓練之環境	15
2. 主程式執行環境	15
三、深度學習及卷積神經網路	16
1. 深度學習	16
2. 卷積神經網路	18
四、傅立葉轉換	26

1. 一維傅立葉轉換	26
2. 二維傅立葉轉換	27
五、Python 程式	29
1. CNN 模型	29
2. 主程式	35
參、結論	41
一、執行結果	41
二、結果分析	43
三、未來展望	48
肆、參考文獻	50
伍、附錄	51

壹、序論

一、研究動機及目的

自從 AI 在圍棋上擊敗人類好手後，AI 開始為人所知，不再是從前只能在科幻電影中才能接觸到的高科技產品，而是開始慢慢走進了我們的生活。也因為如此，數學系的應用之一——大數據也開始逐漸成為人們討論的話題，有鑑於此，我們想從人工智慧基礎的圖像辨識入門，學習機器學習、深度學習等等的 AI 基礎，並嘗試把我們在應用數學系這將近四年的時間裡，學習到的各種數學專業知識，融入我們的專題中，讓我們人工智慧的專題中出現充滿數學與資訊的內容。

在開始研究之前，我們以為這是一個可以以我們現有的能力直接觸及的知識，而在開始研究之後，才發現越深入研究我們越難以負荷，為此我們花費了更多的時間完備了許多資訊相關的專業科目，也因為這樣，我們更確定接下來的道路正是我們的目標。

也是在開始研究之後，我們發現電腦在做如此大量的運算時，會耗費許多時間，所以目前大多數的訓練資料集都是使用低解析度的圖片來進行訓練；但在進行測試時，我們仍會用到高解析度的照片來進行訓練及測試，提高我們的模型準確率以及辨識能力。為了解決高解析度的圖片帶來的負荷，我們對圖片進行傅立葉轉換的加工，會刪除

圖片所帶有的不必要的資訊，所以我們想要觀察加工後的圖片會對神經網路造成何種影響。

我們決定使用深度學習中的一個深度神經網路—卷積神經網路來製作我們的視覺辨識模型，使用 CIFAR-100 來訓練模型，並引入 VGG16 來完備我們的演算法，最後使用至少 100 張的不同圖片來測試結果，並分別比較有無經過二維傅立葉轉換處理的圖片，送入模型預測的結果以及各種參數差異。

二、致謝

感謝帶領著我們專題的鄭富國老師，有鑑於我們的應用數學系上，目前沒有老師是走人工智慧專業的方向，但富國老師還是同意了我們的專題題目，並從旁提供我們寶貴的建議以及適時的幫助，給予學生們非常大的空間撞擊我們的意見、揮灑我們的創意，我們對您的感謝無以言表，在讓我們深摯地向老師致謝之前，我們對這四年來的經歷有些話想先說說。

在大學一開始就讀應用數學系時，讓我們對於應用數學的幻想有嚴重的破滅，原來應用數學只是單純改了名子的純數學系，在茫茫數學的定理、公設下，看不盡那微積分、高等微積分、微分方程等等專業科目的背後，應用數學系真正的「應用」學習從何時到來；也因此，我們學生中有許多同學對持續鑽研單純的數學失去了信心及喜愛，在漫漫三年好課值得一修再修的洗禮下，淡忘了初衷，直至看到專題，才想起當初對應用數學的幻想，又才堅定「無論如何也要把專題做到更好」，以此不愧對我們在數學系就讀的這四年。

那為什麼突然發表以上的經歷呢？其實和我們對富國老師的感情有很大的關係。我們這屆大一下修老師的離散數學必修課，還記得在那時候，富國老師向我們解釋了密碼學的原理與 RSA 演算法所運用到的數學知識，當時，這些知識宛如夜空中的流星，劃入了自從大一

以來，腦部被純數扭曲搥打而乾涸的小溪，「唰！」那一刻透過「數學」鑿壁的山區流入「應用」的源泉，我們對數學的幻想透過富國老師的啟發而萌芽。自從修過離散後，我們組員也都修了富國老師的 java 兩學期課程，期待在一成不變的數學裡，尋找著千變萬化的應用，而那時的我們，說不定早已定下了用程式輔助數學這個專題的目標，也正是如此，我們非常感謝富國老師的教導，因為前方有老師的引路，才有我們今天得以將傅立葉轉換，製作成深度學習的這項成果。

最後感謝國立嘉義大學對我們的教育之恩，給我們一個很好的學習環境，也謝謝許多在過程中幫助我們的好同學，鼓勵我們和分享了我們許多專題的方向，最後也很開心我們這一組彼此勉勵與鼓勵下，終於完成了這項數學與資訊融合的專題，謝謝大家，謝謝我們敬愛的鄭富國老師。

貳、研究內容

一、選擇及使用工具

1、Python

Python，最受歡迎的程式語言之一[1]，是一種廣泛使用的直譯式、進階和通用的程式語言。其設計理念是「優雅」、「明確」、「簡單」，強調程式碼的可讀性和簡潔的語法。

Python 還支援多種程式設計範式，包括函數式、指令式、反射式、結構化和物件導向程式設計，而且擁有動態型別系統和垃圾回收功能，能夠自動管理記憶體使用。Python 還可以通過外界函式介面如標準庫中的 `ctypes` 等，來存取動態連結庫或共享庫中 C 相容資料類型並呼叫其中函式，以便於向用其他語言編寫的程式進行整合和封裝。又因為它本身設計為可擴充的，所以提供給用戶豐富的 API 和工具，方便用戶能夠輕鬆地使用 Python、C、Cython 來編寫擴充模組。

2、PyCharm

PyCharm 是一個用於電腦編程的整合式開發環境(簡稱 IDE) [2]，也就是一個跨平台的程式開發環境，使用 Python 開發，其功能之一也是 Python 編輯器，可以為 Python 程式用戶提供眾多有用的功能。以下為 PyCharm 所擁有的其中幾項功能：

- a. 代碼分析輔助：擁有補全代碼、突顯語法和錯誤提示。
- b. 專案和代碼導航：專門的專案視圖，檔案結構視圖和和檔案、類、方法和用例的快速跳轉。
- c. 重構：包括重新命名，提取方法，引入變數，引入常數、pull，push 等。
- d. 支援 Web 框架：如 Django, web2py 和 Flask。
- e. 整合 Python 除錯器。
- f. 整合單元測試，按行覆蓋代碼。
- g. Google App Engine 下的 Python 開發。
- h. 整合版本控制系統：為 Mercurial，Git，Subversion, Perforce 和 CVS 提供統一的使用者介面，擁有修改以及合併功能。
- i. 遠端開發、支援資料庫與 SQL。

3、Spyder

Spyder 是開放原始碼跨平台科學運算集成開發環境(簡稱 IDE) [3]。其使用 Python 開發。Spyder 整合了 NumPy，SciPy，Matplotlib 與 IPython，以及其他開源軟件，比較容易做數據處理相關工作。

與其他科學數值分析專用 IDE (如 Matlab 或 RStudio) 相比，Spyder 有下列特色：第一，開放原始碼，以 Python 編寫且可以相容於非自由軟體授權。第二，可以使用附加元件擴充，內建互動式工具以處理數據。跨平台的特性使它可以通過 Anaconda，Winpython 和 Python (x,y)。Spyder 有以下功能：

- a. 編輯器：支持多語言，具有函數和類查看器，代碼分析特性（pyflakes 和 pylint 獲得了支持），代碼補全，水平與垂直視窗的分離，直接跳入定義等等。
- b. 交互端口：Python 或 IPython 端口都在工作區可以調整和使用。支持對編輯器里的代碼直接調試。此外整合了 Matplotlib 的圖表顯示。
- c. 文檔瀏覽器：在編輯器或端口中顯示任意類或函數調用的文檔。

- d. 可變的瀏覽窗口：在文件的執行過程中可以創建可變的瀏覽窗口。同時也可以對其進行編輯。
- e. 在文件中查找：支持正則表達式與 Mercurial 倉庫。
- f. 其他擴展使用：Spyder 也可以作為 PyQt4/PySide 的擴展使用（spyderlib 模塊）。例如，Spyder 當中使用的 Python 交互端也可以被你用在自己的 PyQt4/PySide 程序中。

4、TensorFlow

TensorFlow 是一個開源軟體庫[4]，用於各種感知和語言理解任務的機器學習因此也有人稱呼它是一個機器學習框架。其用於研究和生產許多 Google 商業產品，如語音辨識、Gmail、Google 相簿和網頁搜尋。

TensorFlow 充當機器學習的核心平台和函式庫。TensorFlow 的 API 使用 Keras 來允許使用者製作自己的機器學習模型。除了建造和訓練它們的模型之外，TensorFlow 還能幫助裝載資料來訓練模型，並使用 TensorFlow Serving 來部署它。

廣泛的應用程式使用 TensorFlow 作為基礎，其中它已成功實現自動化圖像字幕軟體，例如 DeepDream。另外，2015 年 10 月 26 日，Google 正式啟用了由 TensorFlow 提供支援的 RankBrain，處理了大量的搜尋查詢，替換補充傳統的靜態演算法搜尋結果。

5、CIFAR-100

CIFAR-100 是一個圖像數據集[5]，包含 60000 張 32x32 分辨率的彩色圖像，其中分成 10 個大類，而其下又個別擁有 10 個小類，類別之間的交集為空，也就是不會有相同照片同時存在不同類別這種情況發生。總共 100 個類，每個類包含的 600 個圖像，各有 500 個訓練圖像和 100 個測試圖像。大類別包括：水生哺乳動物、魚類、大型食肉動物、大型人造戶外用品等等。小類別包括：airplane、automobile、bird、cat、deer、dog、frog、horse、ship、truck 等等。

6、VGG-16

VGG 是英國牛津大學 Visual Geometry Group 的縮寫[6]，主要貢獻是使用更多的隱藏層，大量的圖片訓練，提高準確率至 90%。VGG 中根據卷積核大小和卷積層數目的不同，可分為 A，A-LRN，B，C，D，E 共 6 個配置(ConvNet Configuration)，其中以 D、E 兩種配置較為常用，分別稱為 VGG16 和 VGG19。

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

圖(2.1.1) ConvNet Configuration

針對 VGG16 進行具體分析，VGG16 共包含：

- a. 13 個卷積層（Convolutional Layer），用 conv3-XXX 表示。
- b. 3 個全連接層（Fully connected Layer），用 FC-XXXX 表示。
- c. 5 個池化層（Pool layer），用 maxpool 表示。

其中，卷積層和全連接層具有權重係數，因此也被稱為權重層，總數目為 $13+3=16$ ，這即是 VGG16 中 16 的來源（池化層不涉及權重，因此不屬於權重層，不被計數。）。)

二、開發環境

1、模型訓練環境

- 品牌：MSI
- 處理器：Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
- 已安裝記憶體(RAM)：16.0 GB
- 系統類型：64 位元作業系統，x64 型處理器
- 顯示卡：GEFORCE GTX 1050 Ti 7680x4320@60Hz

2、主程式執行環境

- 品牌：Dell
- 處理器：Intel(R) Core(TM) i7-10510U CPU @ 2.30 GHz
- 已安裝記憶體(RAM)：8.0 GB
- 系統類型：64 位元作業系統，x64 型處理器
- 顯示卡：Intel(R) UHD Graphics 1920x1080@59.99Hz

三、深度學習及卷積神經網路

1、深度學習

深度學習（deep learning）是機器學習的分支[7]，是一種以人工神經網路為架構，對資料進行表徵學習的演算法。

深度學習是機器學習中一種基於對資料進行表徵學習的演算法。觀測值可以使用多種方式來表示，如每個像素強度值的向量，或者更抽象地表示成一系列邊、特定形狀的區域等。而使用某些特定的表示方法更容易從實例中學習任務（例如，臉部辨識或面部表情辨識）。深度學習的好處是用非監督式或半監督式的特徵學習和分層特徵提取高效演算法來替代手工取得特徵。表徵學習的目標是尋求更好的表示方法並建立更好的模型來從大規模未標記資料中學習這些表示方法。

至今已有數種深度學習框架，如深度神經網路、卷積神經網路和深度置信網路和迴圈神經網路已被應用在電腦視覺、語音辨識、自然語言處理、音訊辨識與生物資訊學等領域並取得了極好的效果。

我們可以在無人駕駛汽車中看到深度學習技術，它能夠區分紅燈和綠燈，從路邊區分人，甚至可以測量兩輛車之間的距離。

它是一種可以實現手機語音激活、人臉識別和個人設備手勢操作的技術，可以說深度學習已經跟我們生活息息相關。

以下為常見的深度學習庫：

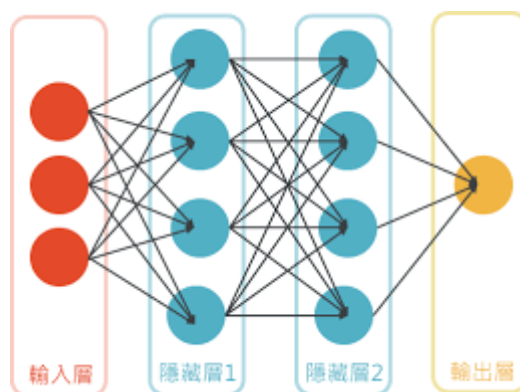
- PyTorch
- ***TensorFlow***
- Theano
- PaddlePaddle
- Deeplearning4j
- Caffe
- roNNie
- ***Keras***
- MXNet

✧ 備註（***粗斜體***）為本專題主要使用的深度學習庫

2、卷積神經網路

每當深度學習又有什麼重大突破時，這些進展幾乎都和卷積神經網路有關。卷積神經網路(Convolutional Neural Networks,CNN)是目前深度神經網路 (deep neural network) 領域的發展主力，在圖片辨別上甚至可以做到比人類還精準的程度。

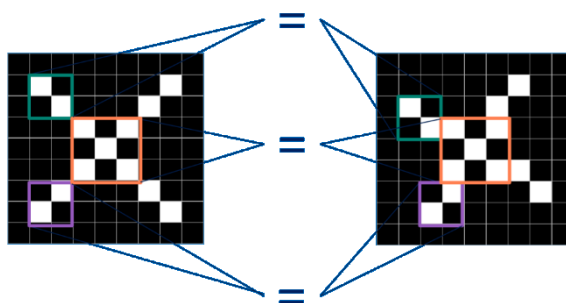
卷積神經網路由一個或多個卷積層和頂端的全連通層組成，同時也包括關聯權重和池化層 (pooling layer)。這一結構使得卷積神經網路能夠利用輸入資料的二維結構。與其他的深度學習結構相比，卷積神經網路在圖像和語音辨識方面能夠給出更好的結果。這一模型也可以使用反向傳播演算法進行訓練。相比較其他深度、前饋神經網路，卷積神經網路需要考量的參數更少，使之成為一種頗具吸引力的深度學習結構。



圖(2.3.1)神經網路結構

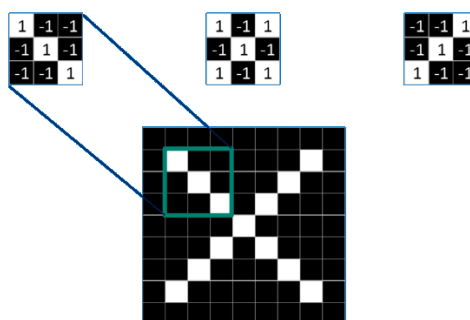
一般 CNN 會經過以下幾頁的重要的流程。

➤ 特徵



圖(2.3.2)比較特徵

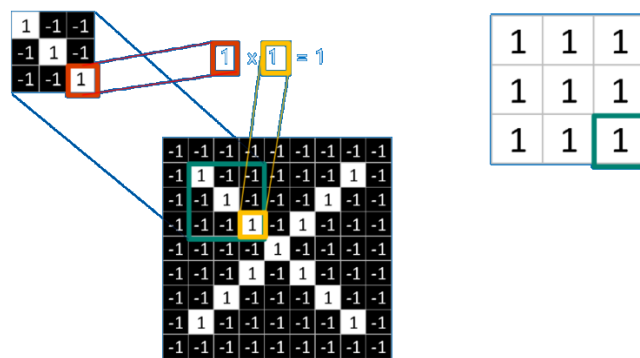
CNN 會比較兩張圖片裡的各個局部，這些局部被稱為特徵 (feature)。比起比較整張圖片，藉由在相似的位置上比對大略特徵，CNN 能更好地分辨兩張圖片是否相同。



圖(2.3.3)捕捉特徵

一張圖片裡的每個特徵都像一張更小的圖片，也就是更小的二維矩陣。這些特徵會捕捉圖片中的共通要素。以叉叉的圖片為例，它最重要的特徵包括對角線和中間的交叉。也就是說，任何叉叉的線條或中心點應該都會符合這些特徵。

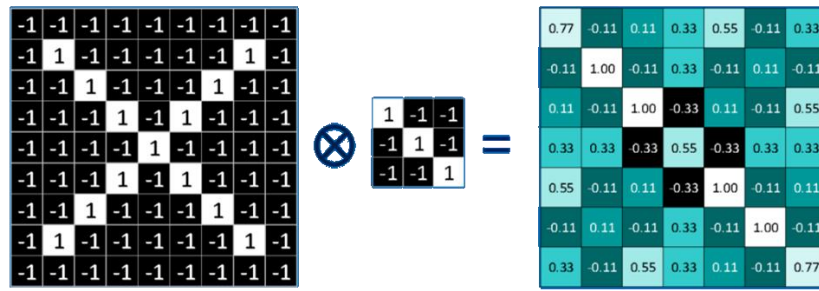
➤ 卷積



圖(2.3.4)卷積運算 1

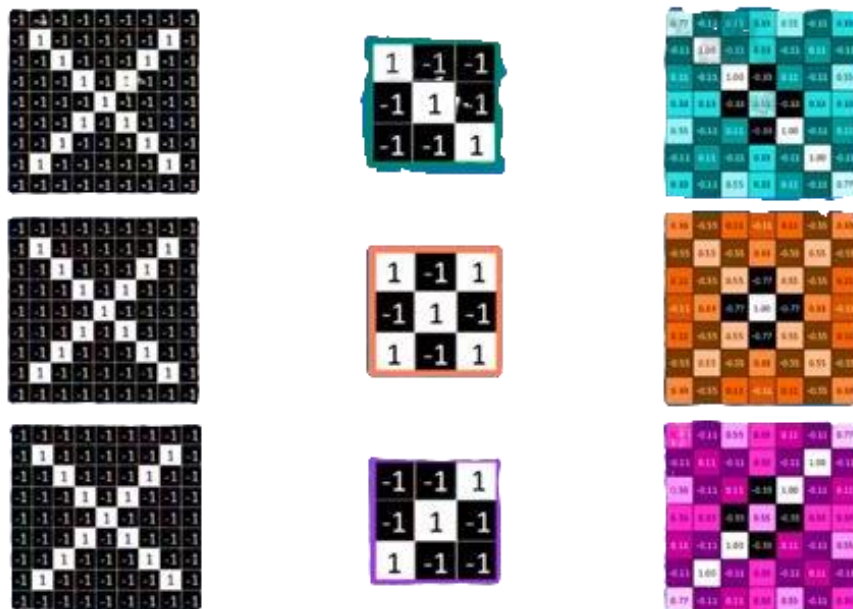
每當 CNN 分辨一張新圖片時，在不知道上述特徵在哪的情況下，CNN 會比對圖片中的任何地方。為了計算整張圖片裡有多少相符的特徵，我們在這裡創造了一套篩選機制。這套機制背後的數學原理被稱為卷積(convolution)，也就是 CNN 的名稱由來。

要計算特徵和圖片局部的相符程度，只要將兩者各個像素上的值相乘、再將總和除以像素的數量。如果兩個像素都是白色(值為 1)，乘積就是 $1 * 1 = 1$ ；如果都是黑色(值為 -1)，乘積就是 $(-1) * (-1) = 1$ 。也就是說像素相符的乘積為 1，像素相異的乘積為 -1。如果兩張圖的每個像素都相符，將這些乘積加總、再除以像素數量就會得到 1；反之，如果兩者的像素完全相異，就會得到 -1。



圖(2.3.5)卷積運算 2

我們只要重複上述過程、歸納出圖片中各種可能的特徵，就能完成卷積。然後，我們可以根據每次卷積的值和位置，製作一個新的二維矩陣。這也就是利用特徵篩選過後的原圖，它可以告訴我們在原圖的哪些地方可以找到該特徵。值越接近 1 的局部和該特徵越相符，值越接近 -1 則相差越大，至於接近值接近 0 的局部，則幾乎沒有任何相似度可言。



圖(2.3.6)卷積運算 3

下一步是將同樣的方法應用在不同特徵上，在圖片中各個部位的

卷積。最後我們會得到一組篩選過的原圖，每一張圖都對應了一個特徵。我們可以將整段卷積運算，簡單想成單一的處理步驟。在 CNN 的運作裡，這個步驟被稱為卷積層，這也代表後面還有更多層。

從 CNN 的運作原理，不難看出它很耗運算資源。雖然我們可以只用一張紙解釋完 CNN 如何運作，但運作途中相加、相乘和相除的數量可以增加得非常快。用數學來表達，可以說這些運算的數量，會隨著：

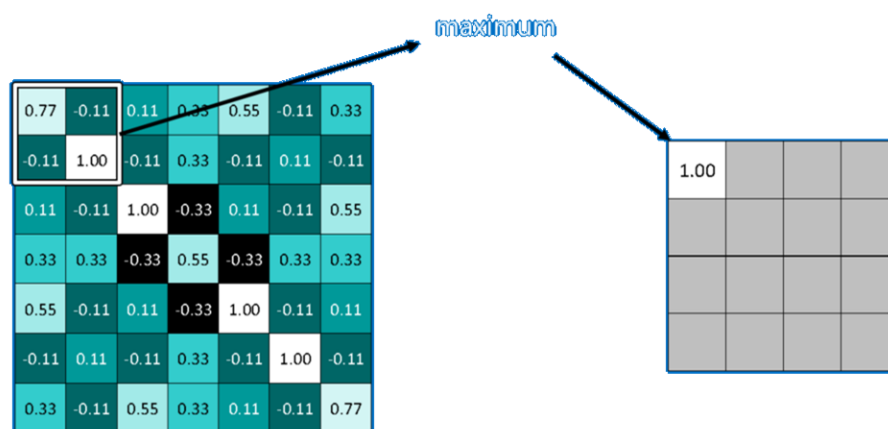
（一）圖片中相素的數量

（二）每個特徵中像素的數量

（三）特徵的數量呈現性增長

有了這麼多影響運算數量的因素，CNN 所處理的問題可以不費吹灰之力地變得非常複雜，也導致一些晶片製造商為了因應 CNN 的運算需求，正在設計和製造專用的晶片。

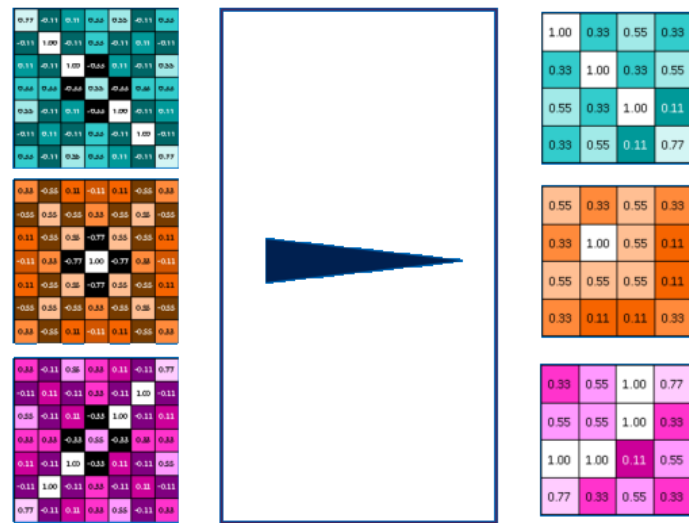
➤ 池化



圖(2.3.7)池化 1

另一個 CNN 所使用的強大工具是池化(pooling)。池化是一個壓縮圖片並保留重要資訊的方法，它的運作原理只需要小二的數學程度就能弄懂。池化會在圖片上選取不同窗口，並在這個窗口範圍中選擇一個最大值。實務上，邊長為二或三的正方形範圍，搭配兩像素的間隔(stride)是滿理想的設定。

原圖經過池化以後，其所包含的像素數量會降為原本的四分之一，但因為池化後的圖片包含了原圖中各個範圍的最大值，它還是保留了每個範圍和各個特徵的相符程度。也就是說，池化後的資訊更專注於圖片中是否存在相符的特徵，而非圖片中哪裡存在這些特徵。這能幫助 CNN 判斷圖片中是否包含某項特徵，而不必分心於特徵的位置。這解決了前面提到電腦非常死板的問題。

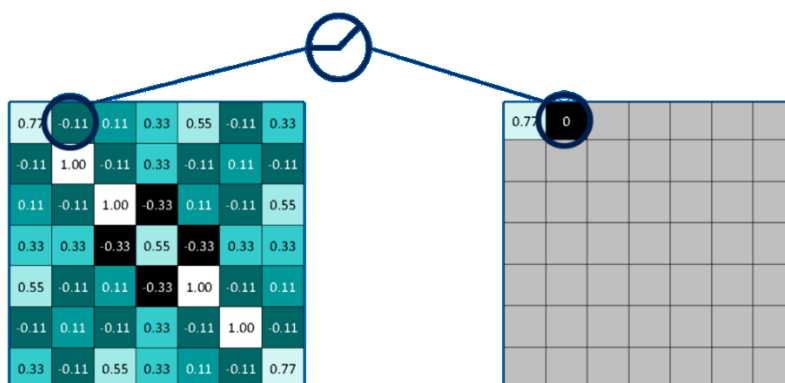


圖(2.3.8)池化 2

所以，池化層的功用是將一張或一些圖片池化成更小的圖片。

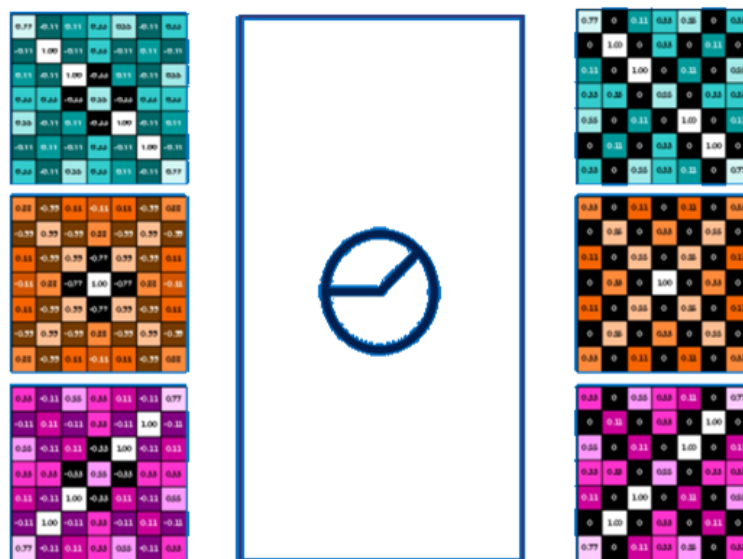
最終我們會得到一樣數量、但包含更少像素的圖片。這也有助於改善剛才提到的耗費運算問題。事先將一張八百萬像素的圖片簡化成兩百萬像素，可以讓後續工作變得更輕鬆。

➤ 線性整流單元



圖(2.3.9) 線性整流單元 1

另一個細微但重要的步驟是線性整流單元 (Rectified Linear Unit, ReLU)，它的數學原理也很簡單，將圖片上的所有負數轉為 0。這個技巧可以避免 CNN 的運算結果趨近 0 或無限大。



圖(2.3.10) 線性整流單元 2

線性整流後的結果和原圖會有相同數量的像素，只不過所有的負值都會被換成零[8][9]。

四、傅立葉轉換

1、一維傅立葉轉換

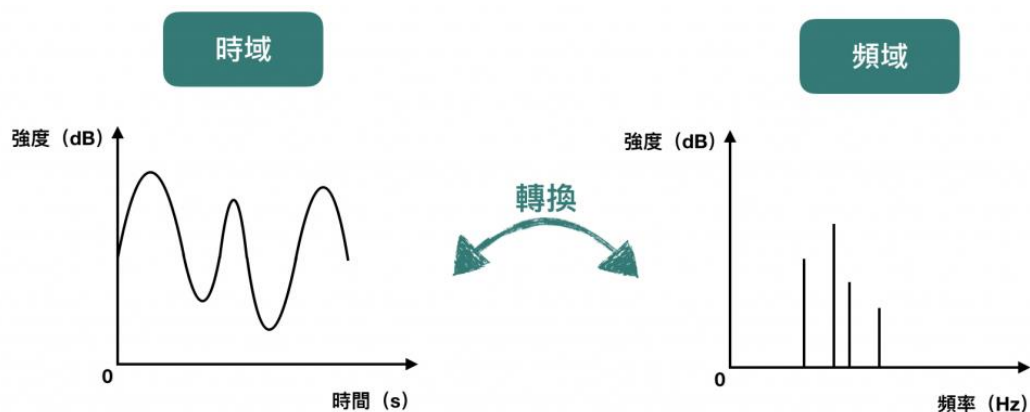
傅立葉轉換可將一個時域訊號，轉換到頻域上去。不管這個訊號是以時域的方式表現（波形, waveform），還是以頻域的方式表現出來（頻譜, spectrum），指的都是同一個訊號，只是藉由傅立葉轉換來改變我們對他的觀點[10]。

➤ 時域

描述數學函數或物理信號對時間的關係。一個信號的時域波形可以表達信號隨著時間的變化。

➤ 頻域

是描述信號在頻率方面特性時用到的一種坐標系。頻域圖顯示了在一個頻率範圍內每個給定頻帶內的信號量。



圖(2.4.1)時域頻域轉換圖[11]

2、二維傅立葉轉換

相較於一維傅立葉轉換，對一張的靜態的平面影像而言，雖然沒有時間的概念在裡面，取而代之的是二維空間域的觀念。二維傅立葉變換的目的是將信號由空間域轉到頻域。

對於平面上的連續非週期信號 $s(t_1, t_2)$ 二維連續傅立葉變換定義為：

$$S(j\omega_1, j\omega_2) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s(t_1, t_2) e^{-j(\omega_1 t_1 + \omega_2 t_2)} dt_1 dt_2$$

影像處理的資料大多不是連續信號，而對於平面上的不連續信號，我們則需使用二維離散傅立葉變換。假設輸入的影像 $s(n, m)$ 水平方向長度是 N ，垂直方向長度是 M ，二維離散傅立葉變換定義為：

$$S(jw_1, jw_2) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} s[n, m] e^{-j(w_1 n + w_2 m)}$$

將以上的概念應用在灰階圖片上，空間域 $f(x, y)$ 表示在位置 (x, y) 的灰階值，二維傅立葉轉換後的結果，我們可以看到它是由哪些灰階度週期變化組成，此時，可以想像許多不同的灰階圖，各個灰階圖視覺上像個小漣漪，這些小漣漪會疊加成最後的

灰階圖片，將此灰階圖進行二維傅立葉轉換，可以得到這些小漣漪的頻率，也就是所謂的圖像頻率。

我們得到圖像頻率後，可以將特定頻率的值設為 0，逆二維傅立葉轉換後可實現濾波功能，例如，圖像的高頻訊號就視覺上看，會讓界線較為明顯，如果可以去除低頻訊號，保留高頻訊號，就有機會保留較多的圖像邊緣。

➤ 逆二維連續傅立葉變換定義為：

$$s(t_1, t_2) = \frac{1}{4\pi^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} S(j\omega_1, j\omega_2) e^{j(\omega_1 t_1 + \omega_2 t_2)} d\omega_1 d\omega_2$$

➤ 逆二維離散傅立葉變換定義為：

$$s[n, m] = \frac{1}{4\pi^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} S(j\omega_1, j\omega_2) e^{j(\omega_1 n + \omega_2 m)} d\omega_1 d\omega_2$$

[12]

這就構成了圖像上的高通濾波器（High pass filter），只不過高頻部份不一定是邊緣，也有可能是高頻雜訊等其他來源，在沒有透過模糊等處理來抑制雜訊的情況下，檢視圖像濾波的結果時，若提高亮度，仍然可以看到一些非邊緣的紋路。

在圖像處理與分析上，傅立葉轉換是個非常有用的工具，除了能認識圖像頻率的意義之外，對於模糊、邊緣等各種處理方式，也會因此而融會貫通[13]。

五、Python 程式

1、CNN 模型

我們運用 Deep learning 的其中一種深度學習框架，也就是上面說的 CNN 來當我們的深度學習模型。

首先我們會需要用來訓練模型的圖片，當然可以使用自己的圖片，甚至建立自己的 Database，但是自己建立圖片庫會產生許多問題，如手機拍的圖片資料量過大、上網找的圖片有版權等；所以我們決定使用 CIFAR-100，關於 CIFAR-100 先前已介紹過，我們直接進入程式步驟：

a. 引入函式庫

```
8 import numpy as np
9 import tensorflow as tf
10 from keras.datasets import cifar100
11 from keras.models import Sequential
12 from keras.layers import Dense
13 from keras.layers import Flatten
14 from keras.layers import Conv2D
15 from keras.layers import MaxPooling2D
16 from keras.layers import Dropout
17 from keras.utils import to_categorical
```

圖(2.5.1)

引入 numpy、tensorflow、keras 等函式庫。

b. 指定亂數種子

```
18  
19     # 指定亂數種子  
20     seed = 7  
21     np.random.seed(seed)  
22     |
```

圖(2.5.2)

深度學習網絡模型中初始的權值參數通常都是初始化成隨機數，而使用梯度下降法最終得到的局部最優解對於初始位置點的選擇很敏感，爲了能夠完全復現作者的開源深度學習代碼，隨機種子的選擇能夠減少一定程度上，算法結果的隨機性，也就是更接近於原始作者的結果，即產生隨機種子意味着每次運行實驗，產生的隨機數都是相同的。

c. 輸入資料集

```
22  
23     #輸入資料集  
24     (X_train, Y_train), (X_test, Y_test) = cifar100.load_data()  
25
```

圖(2.5.3)

d. 正規化

```
25  
26     #正規化  
27     X_train = X_train.astype("float32") / 255  
28     X_test = X_test.astype("float32") / 255  
29
```

圖(2.5.4)

資料正規化是指將原始資料的數據按比例縮放於 $[0, 1]$ 區

間中，且不會改變原本的分佈，我們在進行影像辨識或圖像預處理時，也會習慣將像素值縮放到 $[0,1]$ 之間（即除以 255）。

e. 資料編碼

```
29  
30     #onehot  
31     Y_train = to_categorical(Y_train)  
32     Y_test = to_categorical(Y_test)  
33     model = Sequential()  
34
```

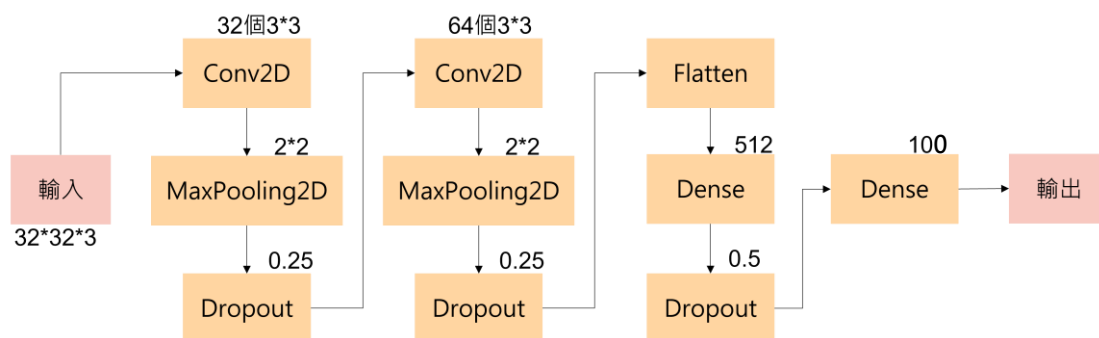
圖(2.5.5)

要訓練模型時，data 必須要是數值形式，所以要將類別型的特徵進行轉換，常用有兩種編碼方式：

- 標籤編碼 (Label Encoding)：類似於流水號，依序將新出現的特徵類別編上新代碼，已出現的類別編上已使用的代碼；適合非深度學習時使用。
- 獨熱編碼 (One Hot Encoding)：改良標籤編碼數字大小沒有意義的問題，將不同特徵類別分別獨立為一欄；適合非深度學習時使用。

f. 定義模型

接著定義神經網路模型，我們規劃的 CNN 如下圖所示（以兩層為例）：



圖(2.5.6) [14]

上圖有 3 組卷積和池化層、3 個 Dropout 層、1 個 Flatten 層、2 個 Dense 層。因為 CIFAR-100 是預測 100 種分類，所以輸出層是 100，程式如下：

```

35     #Define Conv
36     model.add(Conv2D(32, kernel_size = (3, 3), padding = "same",
37                     input_shape = X_train.shape[1:], activation = "relu"))
38
39     model.add(MaxPooling2D(pool_size = (2, 2)))
40     model.add(Dropout(0.25))
41
42     model.add(Conv2D(64, kernel_size = (3, 3), padding = "same",
43                     activation = "relu"))
44
45     model.add(MaxPooling2D(pool_size = (2, 2)))
46
47     #Define Dropout, Flatten, Dense
48     model.add(Dropout(0.25))
49     model.add(Flatten())
50     model.add(Dense(512, activation = "relu"))
51     model.add(Dropout(0.5))
52     model.add(Dense(100, activation="softmax"))
53     model.summary()

```

圖(2.5.7)

此 2 組卷積和池化層分別是 32 個和 64 個 (3,3) 過濾器，最後的 Dense 輸出層物件是 100 個神經元，啟動函數是 softmax 函數。然後呼叫 summary() 函式顯示模型的資訊。

g. 編譯模型

定義好模型後，我們需要編譯模型來轉換成 TensorFlow 計算圖，程式如下：

```
55 #編譯
56 model.compile(loss="categorical_crossentropy",
57               optimizer="adam", metrics=["accuracy"])
58
```

圖(2.5.8)

此 compile() 函式的損失函數是 categorical_crossentropy，優化器是 adam，評估標準是 accuracy 準確度。

h. 訓練模型

在成功編譯模型後，就可以開始訓練模型，程式如下：

```
58
59 #訓練
60 history = model.fit(X_train, Y_train, validation_split= 0.2,
61                    epochs = 50, batch_size = 128, verbose = 2)
62
```

圖(2.5.9)

此 fit() 函式的第 1 個參數是訓練資料集 X_train，第 2 個參數是訓練資料集 Y_train，並且分割出驗證資料集 20%，期訓練週期是 50 次，批次尺寸是 128。

i. 評估與儲存模型

使用訓練資料及訓練模型後，我們可以使用測試資料集來評估模型效能，程式如下：

```
62
63     # 評估模型
64     loss, accuracy = model.evaluate(X_train, Y_train)
65     print(" 訓練資料的準確性")
66     loss, accuracy = model.evaluate(X_test, Y_test)
67     print(" 測試資料的準確性")
68
69     print("saving model: cifar100.h5...")
70     model.save("cifar100.h5")
71
```

圖(2.5.10)

上述程式碼呼叫 `evaluate()` 函式來評估模型，然後使用 `save()` 函式儲存模型結構和權重。

2、主程式

訓練好模型後，我們需要寫一個主程式，用於將不同於 data 圖片的測試圖片，送入我們訓練好的模型進行視覺辨識，在終端輸出執行結果，並直接生成一個 Excel 表格，方便我們進行資料分析。

我們分為兩個程式：一是未將測試圖片經二維傅立葉轉換，直接送入模型辨識；二是將測試圖片經二維傅立葉轉換後，送入模型辨識。

A. 未將測試圖片經二維傅立葉轉換之程式

a. 引入函式庫

```
7
8 import pandas as pd
9 import tensorflow as tf
10 import numpy as np
11 import os
12 import time
```

圖(2.5.11)

引入 pandas、tensorflow、numpy、os、time 等函式。

b. 引入 VGG16

因為前面用於訓練 CIFAR-100 的卷積層數不夠，所以我們引入 VGG16，利用 VGG16 中間層萃取的特徵重新訓練資料。

```
model = tf.compat.v1.keras.applications.vgg16.VGG16()
# model.summary()
```

圖(2.5.12)

VGG16 的卷積和池化層程式如下：

```
# 第1個卷積區塊(block1)
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu',
                  input_shape=input_shape, name='block1_conv1'))
model.add(Conv2D(64, (3,3), padding='same', activation='relu', name='block1_conv2'))
model.add(MaxPool2D((2,2), strides=(2,2), name='block1_pool'))

# 第2個卷積區塊(block2)
model.add(Conv2D(128, (3,3), padding='same', activation='relu', name='block2_conv1'))
model.add(Conv2D(128, (3,3), padding='same', activation='relu', name='block2_conv2'))
model.add(MaxPool2D((2,2), strides=(2,2), name='block2_pool'))

# 第3個區塊(block3)
model.add(Conv2D(256, (3,3), padding='same', activation='relu', name='block3_conv1'))
model.add(Conv2D(256, (3,3), padding='same', activation='relu', name='block3_conv2'))
model.add(Conv2D(256, (3,3), padding='same', activation='relu', name='block3_conv3'))
model.add(MaxPool2D((2,2), strides=(2,2), name='block3_pool'))

# 第4個區塊(block4)
model.add(Conv2D(512, (3,3), padding='same', activation='relu', name='block4_conv1'))
model.add(Conv2D(512, (3,3), padding='same', activation='relu', name='block4_conv2'))
model.add(Conv2D(512, (3,3), padding='same', activation='relu', name='block4_conv3'))
model.add(MaxPool2D((2,2), strides=(2,2), name='block4_pool'))

# 第5個區塊(block5)
model.add(Conv2D(512, (3,3), padding='same', activation='relu', name='block5_conv1'))
model.add(Conv2D(512, (3,3), padding='same', activation='relu', name='block5_conv2'))
model.add(Conv2D(512, (3,3), padding='same', activation='relu', name='block5_conv3'))
model.add(MaxPool2D((2,2), strides=(2,2), name='block5_pool'))

# 前饋全連接區塊
model.add(Flatten(name='flatten'))
model.add(Dense(4096, activation='relu', name='fc1'))
model.add(Dense(4096, activation='relu', name='fc2'))
model.add(Dense(1000, activation='softmax', name='predictions'))
```

圖(2.5.13)

c. 定義陣列

```
array_of_image = []
array_of_image_name = []
column, row = 100, 3
show_svr = [list(range(row)) for _ in range(column)]
```

圖(2.5.14)

array_of_image、array_of_image_name 為儲存測試圖片集的

一維陣列；show_svr 為儲存結果的二維陣列，大小是 3*100。

d. 處理測試圖片

```
def read_directory(path):
    for filename in os.listdir(path):
        img = tf.keras.preprocessing.image.load_img(path + "/" + filename,
                                                    target_size=(224, 224))

        image = tf.keras.preprocessing.image.img_to_array(img)
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        image = tf.compat.v1.keras.applications.vgg16.preprocess_input(image)
        array_of_image.append(image)
        array_of_image_name.append(filename)
```

圖(2.5.15)

load_img 函式為讀取測試圖片，其中 target_size 函數是轉換

圖片大小為 224*224。

img_to_array 函式為將圖片轉成 numpy 形式。

reshape 函式為轉換圖片維度。

preprocess_input 函式為將圖片送入模型。

append 函式為將圖片資料存入陣列。

e. 預測圖片

```
def predict_image(array_of_image, array_of_image_name):
    global show_svr
    svr = [list(range(3)) for _ in range(len(array_of_image))]
    for i in range(0, len(array_of_image)):
        predict2 = model.predict(array_of_image[i])
        label = tf.compat.v1.keras.applications.vgg16.decode_predictions(predict2)
        label = label[0][0]
        svr[i][0] = array_of_image_name[i]
        svr[i][1] = label[1]
        svr[i][2] = label[2] * 100
    for i in range(0, len(array_of_image)):
        for j in range(3):
            show_svr[i][j] = svr[i][j]
```

圖(2.5.16)

第 3 列為宣告一個 local 二維陣列 svr 儲存結果。

model.predict 函式為預測圖片。

decode_predictions 函式為將預測資料轉文字。

第 7~10 列為將結果儲存至 svr 中。

第 11~13 列為將 svr 中的內容複製到 show_svr 中。

f. 輸出影像

```
path = "test_img"
read_directory(path)
start = time.time()
predict_image(array_of_image, array_of_image_name)
print(np.array(show_svr))
end = time.time()
print("執行時間：%f 秒" % (end - start))
```

圖(2.5.17)

第一列為將 path 儲存為測試圖片集之路徑。

呼叫 read_directory 函式並送入 path。

start = time.time() 為儲存預測開始之時間。

呼叫 predict_image 函式。

print(np.array(show_svr)) 為輸出執行結果。

end = time.time() 為儲存預測結束之時間。

print("執行時間：%f 秒" % (end - start)) 為輸出預測時間。

g. 生成 Excel

```
A=np.array(show_svr)
data = pd.DataFrame(A)
writer = pd.ExcelWriter('results_not_ftt.xlsx')
data.to_excel(writer, 'project_nonftt2', float_format='%.5f')
writer.save()
writer.close()
```

圖(2.5.18)

前兩列為讀取執行結果之資料。

第 3 列為將資料存入 Excel 中。

第 4 列為把資料放在叫 project_nonftt2 的頁面中，預設格子寬度為 5。

第 5、6 列為儲存 Excel 並關閉。

B. 將測試圖片經二維傅立葉轉換過之程式

經二維傅立葉轉換過之程式差別地方在於：

a. 處理測試圖片

```
def read_directory(path):
    for filename in os.listdir(path):
        img = tf.keras.preprocessing.image.load_img(path + "/" + filename,
                                                    target_size=(224, 224))

        f = np.fft.fft2(img)
        fshift = np.fft.fftshift(f)
        #設定高通濾波器
        rows, cols = 224, 224
        crow, ccol = int(rows/2), int(cols/2)
        fshift[crow-10:crow+10, ccol-10:ccol+10] = 0
        #傅立葉逆變換
        ishift = np.fft.ifftshift(fshift)
        iimg = np.fft.ifft2(ishift)
        iimg = np.abs(iimg)

        image = tf.keras.preprocessing.image.img_to_array(iimg)
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        image = tf.compat.v1.keras.applications.vgg16.preprocess_input(image)
        array_of_image.append(image)
        array_of_image_name.append(filename)
```

圖(2.5.19)

第 5、6 列為二維傅立葉轉換。

第 8~10 列為設定高通濾波，保留高頻。

第 12~14 列為逆二維傅立葉轉換。

b. 生成 Excel

```
A=np.array(show_svr)
data = pd.DataFrame(A)
writer = pd.ExcelWriter('results_ftt.xlsx')
data.to_excel(writer, 'project_ftt2', float_format='%.5f')
writer.save()
writer.close()
```

圖(2.5.20)

第 4 列為把資料放在 project_ftt2 的頁面中，預設寬度為 5。

參、結論

一、執行結果

1、圖片經二維傅立葉轉換之輸出結果

	原圖片	預測種類	準確率	51			
0	admiral.jpg	admiral	99.99983310699463	52	milk_can.jpg	saltshaker	40.14553427696228
1	alp.jpg	alp	66.05343818664551	53	miniskirt.jpg	bulletproof_vest	54.857999086380005
2	American_alligator.jpg	American_alligator	97.95413613319397	54	mixing_bowl.jpg	mixing_bowl	53.70924472808838
3	American_lobster.jpg	American_lobster	80.28637766838074	55	monitor.jpg	monitor	66.53754711151123
4	Arabian_camel.jpg	Arabian_camel	99.99316930770874	56	motor_scooter.jpg	motor_scooter	53.011274337768555
5	Baby.jpg	lens_cap	10.054999589920044	57	mountain_tent.jpg	mountain_tent	99.78210926055908
6	badger.jpg	badger	99.69845414161682	58	mushroom.jpg	mushroom	59.0970516204834
7	banana.jpg	banana	99.82426166534424	59	orange.jpg	orange	90.51957130432129
8	barn.jpg	barn	99.81101751327515	60	Otter.jpg	otter	77.02960968017578
9	bath_towel.jpg	bath_towel	96.03646993637085	61	parachute.jpg	parachute	96.64106965065002
10	bear.jpg	brown_bear	99.8300850391388	62	passenger_car.jpg	passenger_car	53.79200577735901
11	beaver.jpg	beaver	99.59979057312012	63	picket_fence.jpg	picket_fence	94.61448192596436
12	bee.jpg	bee	32.486692070961	64	pier.jpg	pier	64.83314037322998
13	bell_pepper.jpg	bell_pepper	99.0760087966919	65	plow.jpg	harvester	58.670175075531006
14	boathouse.jpg	boathouse	96.7470645904541	66	pomegranate.jpg	pomegranate	99.62292313575745
15	breakwater.jpg	drilling_platform	21.65105640888214	67	porcupine.jpg	porcupine	97.86823987960815
16	buckeye.jpg	buckeye	99.98562335968018	68	promontory.jpg	cliff	30.786147713661194
17	cardigan.jpg	trench_coat	34.59592163562775	69	puffer.jpg	puffer	99.99992847442627
18	castle.jpg	castle	89.50265645980835	70	raccoon.jpg	badger	34.54991281032562
19	centipede.jpg	centipede	100.0	71	red_fox.jpg	red_fox	82.15382695198059
20	chimpanzee.jpg	chimpanzee	95.66910862922668	72	rhinoceros_beetle.jpg	rhinoceros_beetle	98.59248995780945
21	cloud.jpg	monitor	10.947801172733307	73	school_bus.webp	school_bus	99.84752535820007
22	cockroach.jpg	cockroach	99.99830722808838	74	sea_lion.jpg	sea_lion	78.96134853363037
23	common_iguana.jpg	common_iguana	99.04118180274963	75	skunk.jpg	skunk	99.99935626983643
24	cow.jpg	ox	79.2001724243164	76	snail.jpg	snail	99.92061257362366
25	coyote.jpg	coyote	99.55520033836365	77	sorrel.jpg	head_cabbage	68.3821976184845
26	crutch.jpg	hook	12.408853322267532	78	space_bar.jpg	computer_keyboard	58.53880047798157
27	cup.jpg	cup	90.31460285186768	79	space_shuttle.jpg	space_shuttle	99.99998807907104
28	daisy.jpg	tray	55.032700300216675	80	stone_wall.jpg	stone_wall	98.19059371948242
29	dial_telephone.jpg	dial_telephone	96.61003947257996	81	streetcar.jpg	streetcar	75.98405480384827
30	dining_table.jpg	dining_table	93.99781227111816	82	studio_couch.jpg	studio_couch	99.55763816833496
31	electric_ray.jpg	electric_ray	59.19795036315918	83	sulphur_butterfly.jpg	sulphur_butterfly	99.75589513778687
32	fiddler_crab.jpg	fiddler_crab	49.49847161769867	84	suspension_bridge.jpg	pier	94.85151290893555
33	fig.jpg	fig	99.99979734420776	85	table_lamp.jpg	table_lamp	91.793692111969
34	flower.jpg	daisy	17.343920469284058	86	tank.jpg	tank	99.97336268424988
35	folding_chair.jpg	folding_chair	99.97443556785583	87	tarantula.jpg	tarantula	99.81958270072937
36	fox_squirrel.jpg	fox_squirrel	99.95155334472656	88	television.jpg	monitor	39.09694254398346
37	frilled_lizard.jpg	frilled_lizard	84.9924087524414	89	tench.jpg	kite	92.9796814918518
38	fur_coat.jpg	fur_coat	99.94704127311707	90	tiger.jpg	tiger	83.28307867050171
39	great_white_shark.webp	great_white_shark	89.01240229606628	91	tow_truck.jpg	tow_truck	98.5313892364502
40	greenhouse.jpg	greenhouse	99.7937560081482	92	tricycle.jpg	tricycle	91.63719415664673
41	hay.jpg	hay	99.99998807907104	93	tusker.jpg	African_elephant	39.043548703193665
42	hook.jpg	hook	56.97360634803772	94	vase.jpg	vase	31.15953505039215
43	Indian_cobra.jpg	Indian_cobra	99.99446868896484	95	wallaby.jpg	wallaby	99.91820454597473
44	killer_whale.jpg	killer_whale	99.9585211277008	96	wall_clock.jpg	wall_clock	51.626622676849365
45	lawn_mower.jpg	lawn_mower	66.82392358779907	97	wardrobe.jpg	wardrobe	97.06973433494568
46	leopard.jpg	leopard	95.9805965423584	98	washbasin.jpg	washbasin	94.11744475364685
47	lion.jpg	lion	99.87043142318726	99	water_bottle.jpg	water_bottle	74.3401101722717
48	loggerhead.jpg	loggerhead	57.519710063934326		wood_rabbit.jpg	hare	94.59657073020935
49	magpie.jpg	magpie	97.03837037086487				
50	marmot.jpg	marmot	99.96558427810669				

表(3.1.1)

2、圖片不經二維傅立葉轉換之輸出結果

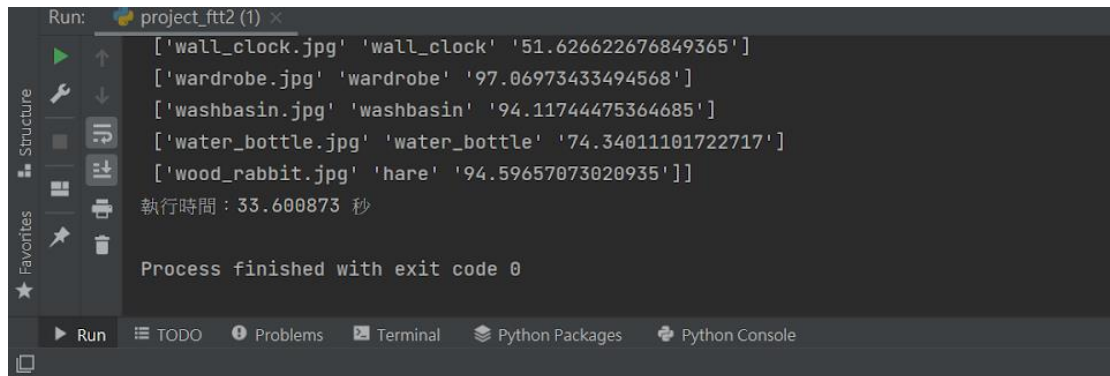
	0	1	2	51			
0	admiral.jpg	admiral	99.99990463256836	52	milk_can.jpg	milk_can	35.66523194313049
1	alp.jpg	alp	41.38382971286774	53	miniskirt.jpg	bulletproof_vest	19.196084141731262
2	American_alligator.jpg	American_alligator	98.06662201881409	54	mixing_bowl.jpg	mixing_bowl	42.181333899497986
3	American_lobster.jpg	American_lobster	50.27668476104736	55	monitor.jpg	monitor	52.107858657836914
4	Arabian_camel.jpg	Arabian_camel	99.98821020126343	56	motor_scooter.jpg	motor_scooter	49.145546555519104
5	Baby.jpg	bib	19.95689421892166	57	mountain_tent.jpg	mountain_tent	99.98325109481812
6	badger.jpg	badger	99.43603873252869	58	mushroom.jpg	mushroom	58.92831087112427
7	banana.jpg	banana	99.2222249507904	59	orange.jpg	orange	91.54325127601624
8	barn.jpg	barn	99.33335185050964	60	Otter.jpg	otter	65.15154242515564
9	bath_towel.jpg	bath_towel	94.98510956764221	61	parachute.jpg	parachute	98.41051697731018
10	bear.jpg	brown_bear	99.5470404624939	62	passenger_car.jpg	passenger_car	53.703951835632324
11	beaver.jpg	beaver	99.11155700683594	63	picket_fence.jpg	picket_fence	92.72710084915161
12	bee.jpg	bee	60.93505024909973	64	pier.jpg	pier	53.11962366104126
13	bell_pepper.jpg	bell_pepper	98.73358607292175	65	plow.jpg	plow	43.94822716712952
14	boathouse.jpg	boathouse	96.56234979629517	66	pomegranate.jpg	pomegranate	99.56195950508118
15	breakwater.jpg	drilling_platform	37.85805404186249	67	porcupine.jpg	porcupine	95.39403319358826
16	buckeye.jpg	buckeye	99.9591052532196	68	promontory.jpg	promontory	59.624290466308594
17	cardigan.jpg	wool	55.29302954673767	69	puffer.jpg	puffer	99.99990463256836
18	castle.jpg	castle	71.437668800354	70	raccoon.jpg	badger	22.875458002090454
19	centipede.jpg	centipede	99.9998807907104	71	red_fox.jpg	red_fox	77.3446798324585
20	chimpanzee.jpg	chimpanzee	94.51034665107727	72	rhinoceros_beetle.jpg	rhinoceros_beetle	99.48524832725525
21	cloud.jpg	hay	12.137100100517273	73	school_bus.webp	school_bus	99.9593436717987
22	cockroach.jpg	cockroach	99.98149275779724	74	sea_lion.jpg	sea_lion	92.9680347442627
23	common_iguana.jpg	common_iguana	98.19921255111694	75	skunk.jpg	skunk	99.99860525131226
24	cow.jpg	ox	74.79037642478943	76	snail.jpg	snail	99.1431474685669
25	coyote.jpg	coyote	99.72014427185059	77	sorrel.jpg	head_cabbage	60.29888391494751
26	crutch.jpg	tripod	34.39693748950958	78	space_bar.jpg	space_bar	44.20389235019684
27	cup.jpg	cup	69.83312368392944	79	space_shuttle.jpg	space_shuttle	100.0
28	daisy.jpg	daisy	75.7570207118988	80	stone_wall.jpg	stone_wall	99.91515874862671
29	dial_telephone.jpg	dial_telephone	95.85604667663574	81	streetcar.jpg	streetcar	63.79164457321167
30	dining_table.jpg	dining_table	96.52280807495117	82	studio_couch.jpg	studio_couch	99.75985884666443
31	electric_ray.jpg	electric_ray	66.29902720451355	83	sulphur_butterfly.jpg	sulphur_butterfly	99.78456497192383
32	fiddler_crab.jpg	fiddler_crab	63.80646824836731	84	suspension_bridge.jpg	suspension_bridge	62.563556432724
33	fig.jpg	fig	99.99996423721313	85	table_lamp.jpg	table_lamp	87.81066536903381
34	flower.jpg	lotion	16.53049886226654	86	tank.jpg	tank	99.95953440666199
35	folding_chair.jpg	folding_chair	99.9532699584961	87	tarantula.jpg	tarantula	99.78711605072021
36	fox_squirrel.jpg	fox_squirrel	99.94351267814636	88	television.jpg	television	36.29558980464935
37	frilled_lizard.jpg	frilled_lizard	84.5142126083374	89	tench.jpg	kite	96.63088321685791
38	fur_coat.jpg	fur_coat	99.92433786392212	90	tiger.jpg	tiger	73.53847622871399
39	great_white_shark.webp	great_white_shark	82.138991355896	91	tow_truck.jpg	tow_truck	98.71180057525635
40	greenhouse.jpg	greenhouse	99.84539747238159	92	tricycle.jpg	tricycle	97.20653295516968
41	hay.jpg	hay	99.99996423721313	93	tusker.jpg	African_elephant	46.72376811504364
42	hook.jpg	hook	52.188920974731445	94	vase.jpg	lotion	42.121073603630066
43	Indian_cobra.jpg	Indian_cobra	99.98692274093628	95	wallaby.jpg	wallaby	99.82166886329651
44	killer_whale.jpg	killer_whale	99.92509484291077	96	wall_clock.jpg	wall_clock	80.85623383522034
45	lawn_mower.jpg	lawn_mower	92.74112582206726	97	wardrobe.jpg	wardrobe	88.50414752960205
46	leopard.jpg	leopard	97.37122654914856	98	washbasin.jpg	washbasin	89.88063335418701
47	lion.jpg	lion	99.11594986915588	99	water_bottle.jpg	water_bottle	63.88157606124878
48	loggerhead.jpg	loggerhead	61.16054654121399		wood_rabbit.jpg	hare	89.43575620651245
49	magpie.jpg	magpie	95.8416998386383				
50	marmot.jpg	marmot	99.99759197235107				

表(3.1.2)

二、結果分析

1、程式執行辨識的執行時間

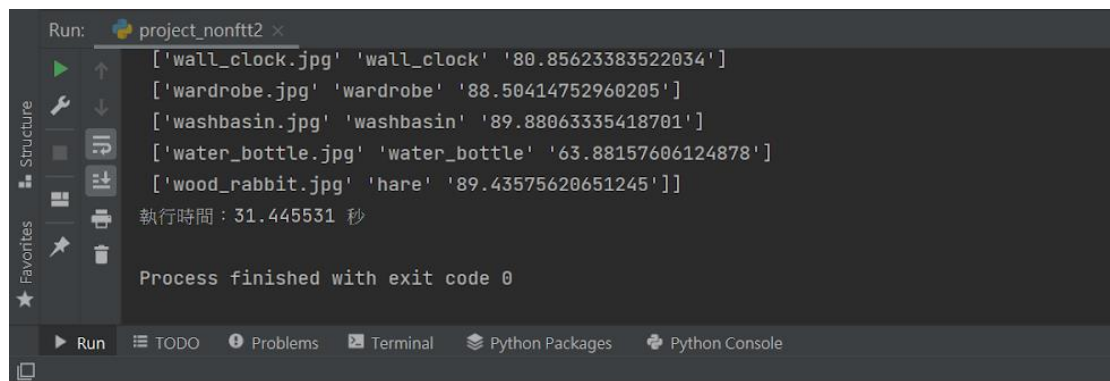
A. 經二維傅立葉轉換：輸出時間為：**33.600873 秒**



```
Run: project_fft2 (1) ×  
['wall_clock.jpg' 'wall_clock' '51.626622676849365']  
['wardrobe.jpg' 'wardrobe' '97.06973433494568']  
['washbasin.jpg' 'washbasin' '94.11744475364685']  
['water_bottle.jpg' 'water_bottle' '74.34011101722717']  
['wood_rabbit.jpg' 'hare' '94.59657073020935']  
執行時間：33.600873 秒  
Process finished with exit code 0
```

圖(3.2.1)

B. 不經二維傅立葉轉換：輸出時間為：**31.445531 秒**



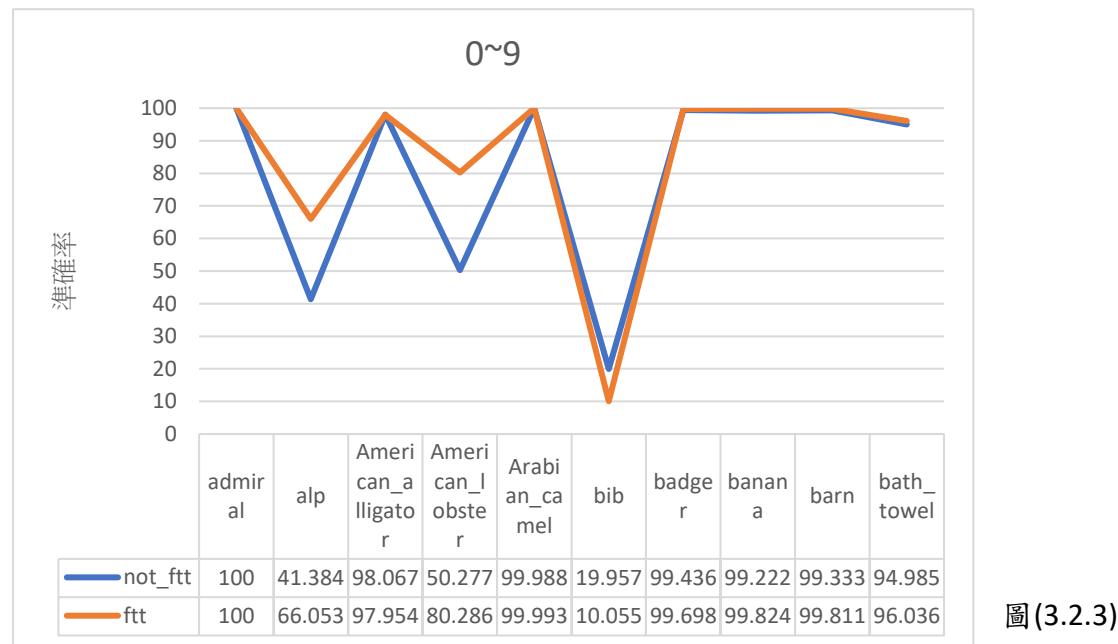
```
Run: project_nonfft2 ×  
['wall_clock.jpg' 'wall_clock' '80.85623383522034']  
['wardrobe.jpg' 'wardrobe' '88.50414752960205']  
['washbasin.jpg' 'washbasin' '89.88063335418701']  
['water_bottle.jpg' 'water_bottle' '63.88157606124878']  
['wood_rabbit.jpg' 'hare' '89.43575620651245']  
執行時間：31.445531 秒  
Process finished with exit code 0
```

圖(3.2.2)

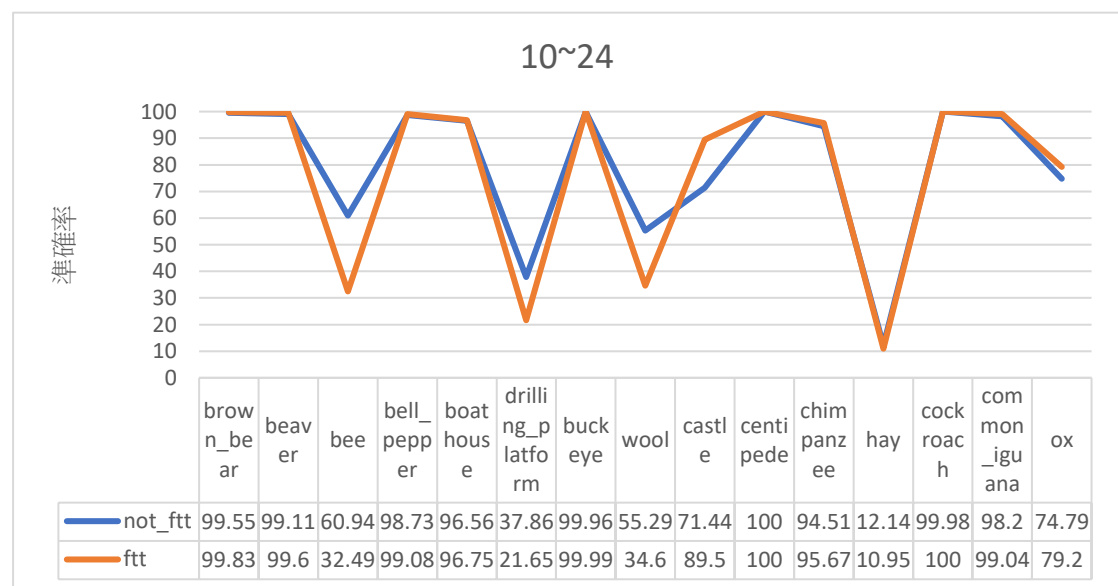
綜合兩者的輸出時間，我們可以得出經二維傅立葉轉換之程式的執行時間高於不經二維傅立葉轉換之程式的執行時間，但是兩者差距僅 2 秒，約 **1.068542 倍**，所以已結果來說兩者差距**可忽略**。

2、準確率對比圖

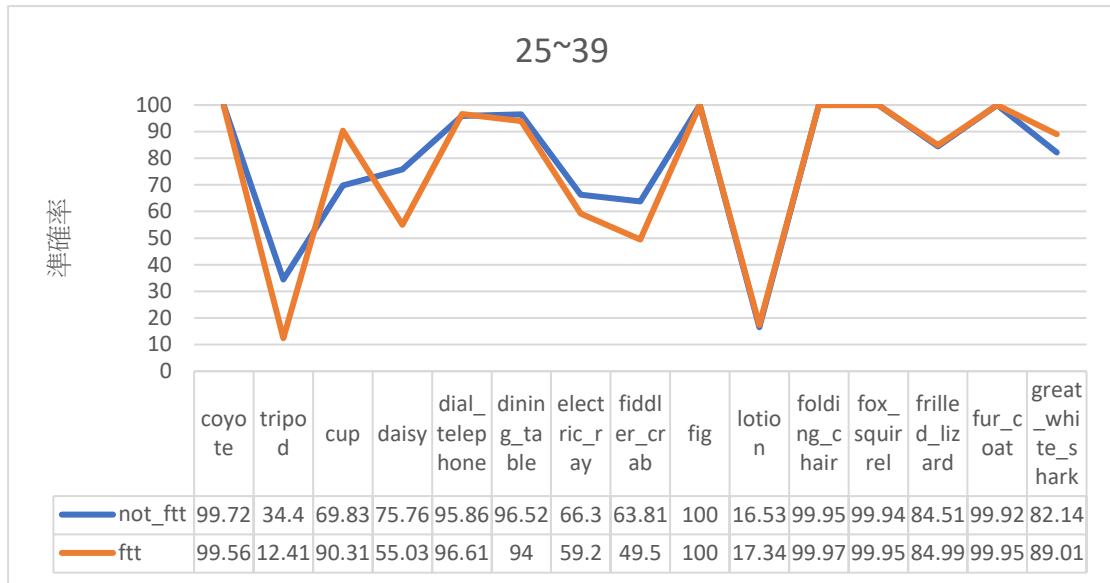
我們將兩個程式輸出的 Excel 做比較，並製作成表格，增加可視度外也方便直接比較：



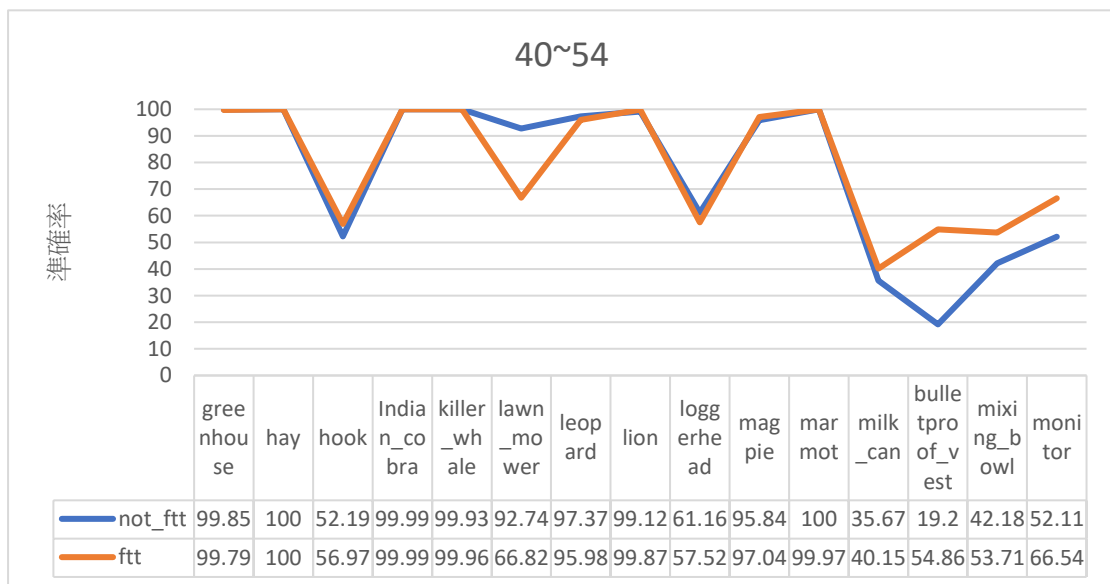
圖(3.2.3)



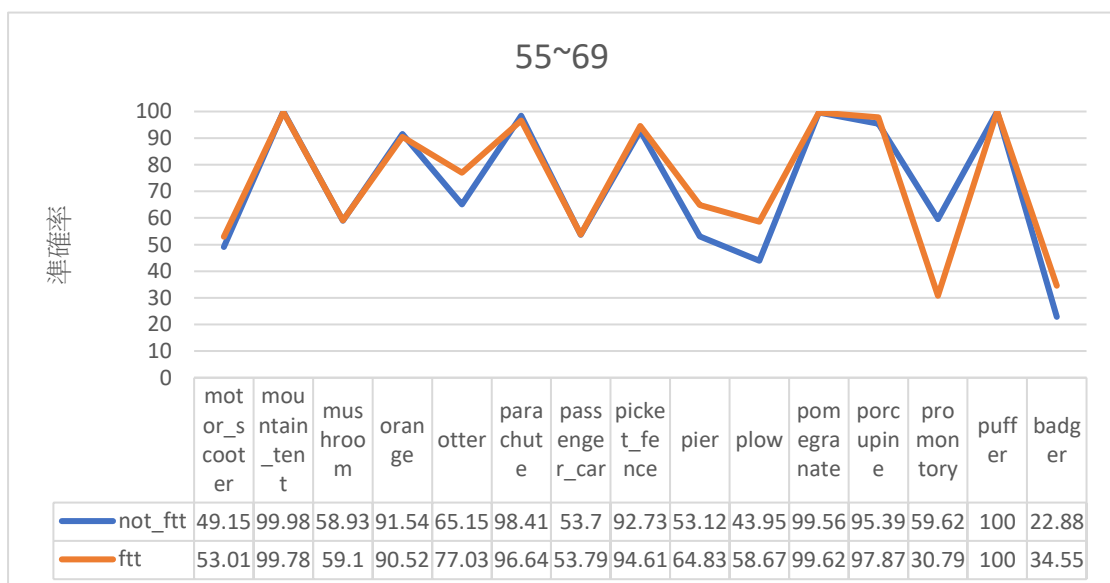
圖(3.2.4)



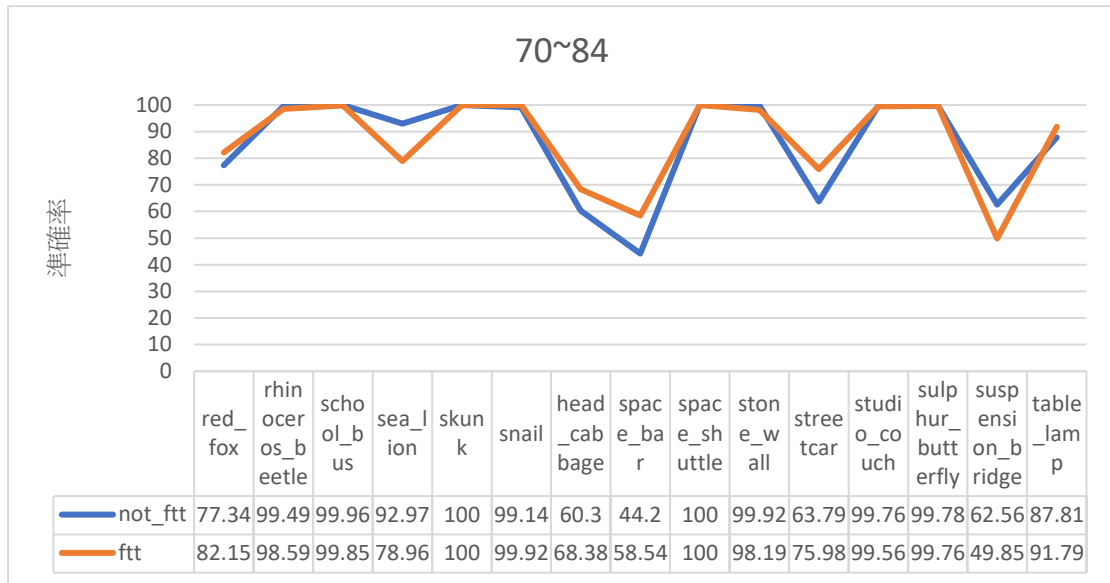
圖(3.2.5)



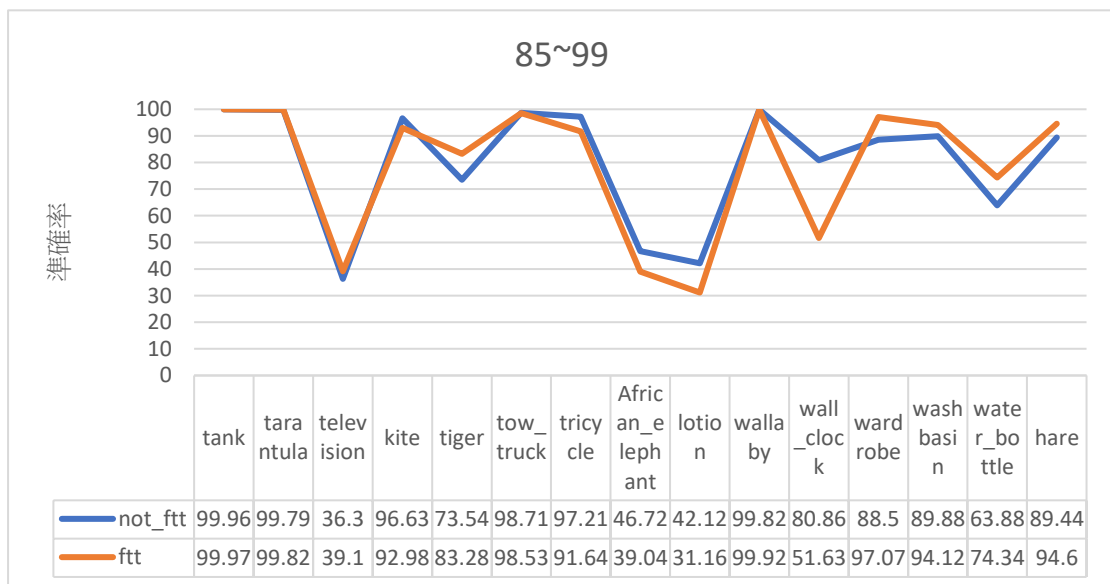
圖(3.2.6)



圖(3.2.7)



圖(3.2.8)



圖(3.2.9)

出現高的次數	出現低的次數	比例差
63	37	1.7027027

圖(3.2.10)

統整之後我們比較準確率發現經二維傅立葉轉換之程式準確率比不經二維傅立葉轉換之程式準確率還高的情況有 63 次，比例差為 1.7027027 倍，經二維傅立葉轉換之程式準確率明顯高於不經二維傅立葉轉換之程式準確率。

綜合上述之比較，時間相差約 1.068542 倍，準確率比較高的次數相差約 1.7027027 倍；以準確率提升的比率來看，我們認為：將測試圖片經二維傅立葉轉換後再送入訓練好的模型進行視覺辨識預測，是比沒有經二維傅立葉轉換的方式好的，透過二維傅立葉轉換的特性來找到圖片頻率，並通過高通濾波保留高頻進行預測來降低資訊量，進而將背景複雜的圖片複雜度降低，大幅提升準確率。

這個情況跟我們執行程式前猜想的結果大致相同，只有執行時間是我們意想之外的結果，執行前的猜想是經二維傅立葉轉換之程式的執行時間會低於沒有經二維傅立葉轉換之程式的執行時間，我們認為造成此結果的原因可能有：

- a. 程式不夠精簡
- b. 硬體設備不夠完備，包括：沒有獨顯、RAM 不足等問題
- c. 演算法不完備

我們會以此為目標繼續完備我們的模型，持續精進程式技巧，完成我們的目標。

三、未來展望

深度學習是未來的趨勢，也是世界趨之若鶩研究的目標，從我們身邊的手機、各種軟體到人類基本一定會碰到的醫學、甚至是還在開發期的自動駕駛都是深度學習的範疇；在這世界以及市場的需求下，人工智慧的知識及研究、與人工智慧的人才，都是不可多得的存在。

我們的專題在於研究圖片辨識更好的演算法，在提升準確率的同時融入在數學系學習的專長，完成數學與資訊融合的成果，以此為基礎往更深的地方研究，只能說延伸是非常非常多的；我們預計尋找更多關於人臉的圖片資料，並運用攝影機時時寫入圖像資料，實現更實用的人臉辨識，加入紅外線儀器可以監測來上課的同學的體溫、加入Live2D 虛擬人偶可以實現透過肢體及臉部表情控制人偶動作、甚至是導入個人資料及臉部詳細資訊可以實現 2D 人臉辨識等延伸，我們希望可以在畢業前完成：能夠時時追蹤攝影機捕捉到的人的臉的模型以及演算法，我們將以這個目標持續努力。

而對於未來的期望，我們研究的專題正是步入人工智慧領域的一個很好的基礎，雖然我們的訓練資料只有上萬筆，測試資料只有幾百筆的情況下，還無法稱是大數據分析，但是當我們的知識及能力上升後，處理的資料就從一萬筆、十萬筆、甚至上千萬筆起跳，這時更能凸顯我們數學系專長應用之一的大數據分析的能力，運用我們比其他

人還要多的數學知識來強化我們的能力，期望我們在未來的工作中，
都能透過記起這個專題的融合性以及實用性，創造出更能造福社會的
作品。

肆、參考文獻

- [1] <https://zh.wikipedia.org/zh-tw/Python>
- [2] <https://zh.wikipedia.org/zh-tw/PyCharm>
- [3] <https://zh.wikipedia.org/zh-tw/Spyder>
- [4] <https://zh.wikipedia.org/zh-tw/TensorFlow>
- [5] <https://www.cs.toronto.edu/~kriz/cifar.html>
- [6] <http://deanhan.com/2018/07/26/vgg16/>
- [7] <https://zh.wikipedia.org/zh-tw/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0>
- [8] https://brohrer.mcknote.com/zh-Hant/how_machine_learning_works/how_convolutional_neural_networks_work.html
- [9] [How Convolutional Neural Networks work - YouTube](#)
- [10] <https://hackmd.io/@jkrvivian/B1wHF21ib?type=view>
- [11] <https://ithelp.ithome.com.tw/articles/10228205>
- [12] <https://zh.m.wikipedia.org/zh-tw/%E4%BA%8C%E7%B6%AD%E5%82%85%E7%AB%8B%E8%91%89%E8%AE%8A%E6%8F%9B>
- [13] <https://www.ithome.com.tw/voice/145429>
- [14] [TensorFlow 與 Keras：Python 深度學習應用實務](#)

伍、附錄

專題網站

http://myweb.ncyu.edu.tw/~s1082574/graduation_project/

