銘傳大學

資訊傳播工程學系

機器學習導論期末專題報告

服裝辨識

班級：資傳二乙

組長：08160971 黃凱勵

組員：08160422 孫振寧

08160723 鄧晨言

08160130 林采葳

中 華 民 國 一 一 〇 年 十 二 月 二 十 八 日

目錄

圖目錄

第一章　研究動機

一開始在選擇題目的時候參考網路上很多的應用，發現很多的例子，像是手寫辨識、地價預測…….等等的，其中服裝辨識對我們而言是相對好理解的，因此我們最後決定做服裝辨識。

第二章　研究方法

第一節　資料來源

透過 MNIST 的數據庫讀取資料，如圖 2-1-1 所示。

```
mnist = tf.keras.datasets.fashion_mnist

(training_images, training_labels),(test_images,test_labels) = mnist.load_data()
```

圖 2-1-1 讀取 MNIST 數據庫

第二節　方式

採用分類的方式，將服裝分為 10 類，分別為 T 恤/上衣、褲子、套頭衫、禮服、外套、涼鞋、襯衫、運動鞋、包包靴子、長靴。

第三節　資料標籤

- 0 T-shirt/top：T 恤/上衣
- 1 Trouser：褲子；

- 2 Pullover：套頭衫
- 3 Dress：禮服；
- 4 Coat：外套
- 5 Sandal：涼鞋；
- 6 Shirt：襯衫
- 7 Sneaker：運動鞋；
- 8 Bag：包包袋子
- 9 Ankle boot：長靴

第四節　資料屬性

(1) 服飾的特徵。

(2) 每個點的灰階資料。

第五節　模型

(1) Sequential 模型。

(2) 用來構建深度神經網絡。

(3) 第一層是 Flattening layer(展平層)。

(4) 第二層為全連接層，並設置 128 個神經元。

(5) 第三層則輸出 10 維的向量，分別代表這張圖片屬於 0 到 9 的機率，如圖 2-5-1 所示。

```
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(128, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

圖 2-5-1 Sequential 模型

第六節　優化器

接下來選擇優化器，我們使用’Adam’優化器，一般而言比 SGD 模型(隨機梯度下降法)成本低。

第七節　損失函數

損失函數爲’sparse_categorical_crossentropy’，就是**交叉熵**，而 categorical_crossentropy 和 sparse_categorical_crossentropy 這二者都是針對多分類任務。**差別**在於**輸入參數**形式上的區別，在 loss 的計算在本質上沒有區別。

第八節　成效衡量指標

成效衡量指標則是'accuracy’，等於 (tp + tn) / (tp + fp + fn + tn)，如圖 2-8-1 所示。

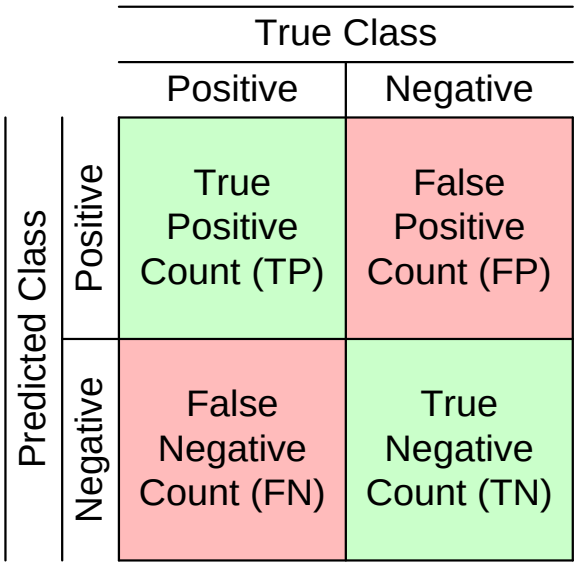


圖 2-8-1 成效衡量指標

第九節　訓練數據（epochs）的設置

通常，epochs 越大，最後訓練的損失值會越小，但是訓練次數過大，會導致過擬合的現象。第一次我們使用 5 次 epochs，可看出訓練後正確率可達 89%；第二次我們將 epochs 設為 50，訓練 50 次訓練集，並從測試集劃分 80%給訓練集，測試的間隔為 20 次，可看出第一次訊聯時的正確率只有 81%，每隔 20 次訓練的正確率分別為 89.08%、89.26%，但當訓練到 50 次的時候，正確率可高達 96%。

第三章　實驗

第一節　實驗 1

第一項 實驗平台

Jupyter 是一個能夠把軟體代碼、計算輸出、解釋文檔、多媒體資源整合在一起的多功能科學運算平台；且不需要切換窗口去找資料，只要看一個文件，就可以獲得項目的所有信息；另外，對於每次實驗可以只跑一小個 Cell 裡的代碼，在代碼下面立刻就可以看到結果，因此我們使用 Jupyter 作為製作此專題的平台，並使用 python 進行程式的撰寫。

圖 3-1-1 Jupyter

第二項 實驗環境

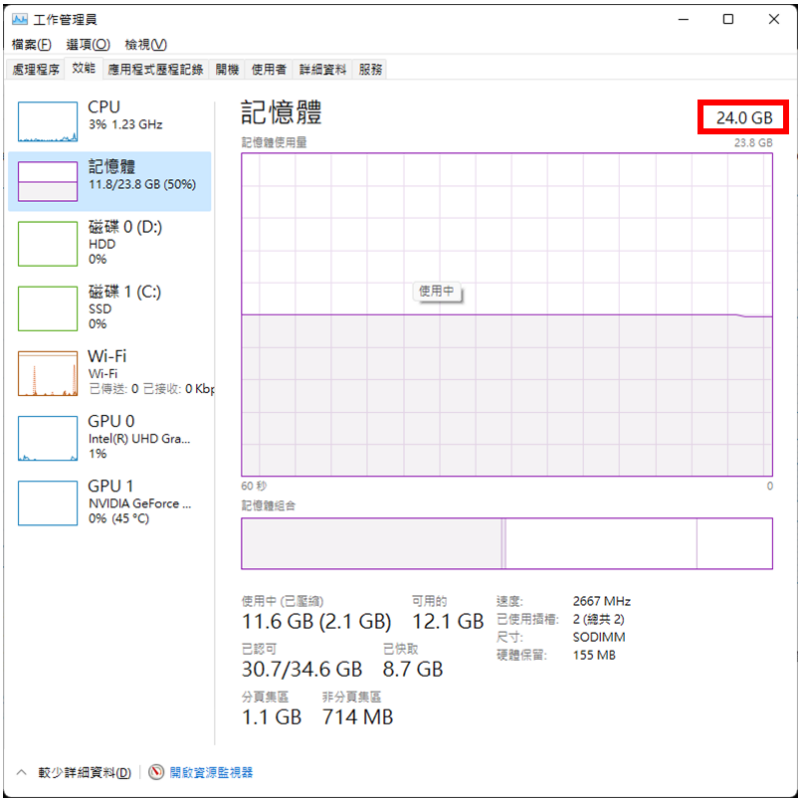


圖 3-1-2 記憶體

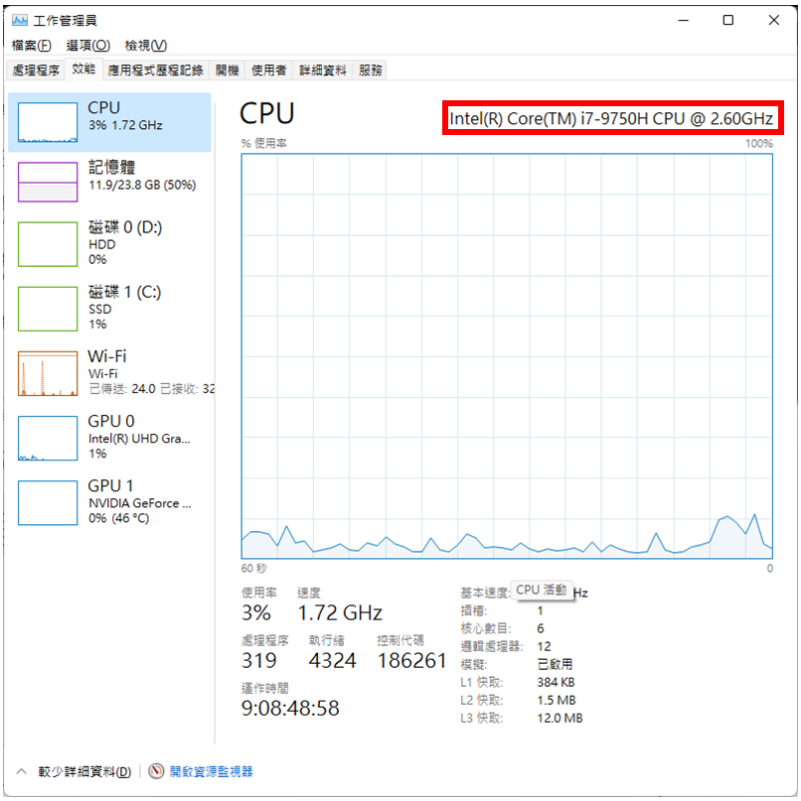


圖 3-1-3 CPU

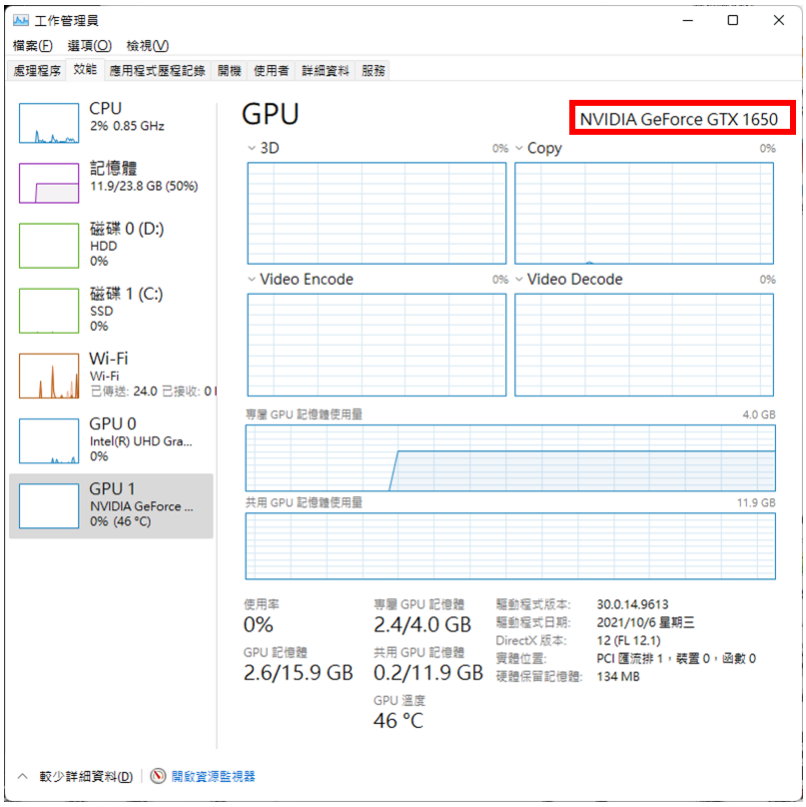


圖 3-1-4 GPU

第三項 實驗函式庫簡介

1. import tensorflow as tf，用於機器學習和深度神經網路方面的研究
2. import tkinter as tk，用來在 Python 中建構 GUI 圖形介面程式
3. from tkinter import filedialog，開啟檔案對話框用法
4. import matplotlib.pyplot as plt，用來繪圖、圖表呈現及數據表示
5. import numpy as np，支援大量的維度陣列與矩陣運算，也針對陣列運算提供大量的數學函式庫
6. import random，匯入標準模組庫中(standard library)的亂數模組(random)

```
import tensorflow as tf
import tkinter as tk

from tkinter import filedialog
import matplotlib.pyplot as plt
import numpy as np
import random
```

圖 3-1-5 函式庫

第四項 實驗資料數

訓練資料數量 有 60000 張 28*28 大小的圖片；測試資料數量 有 10000 張 28*28 大小的圖片。

第五項 程式碼講解

第一步先匯入我們程式所需的函式庫模組，如圖 3-1-6 所示。

```
import tensorflow as tf                                    #匯入模型
import tkinter as tk

from tkinter import filedialog
import matplotlib.pyplot as plt
import numpy as np
import random
```

圖 3-1-6 匯入函式庫模組

第二步讀取 MNIST 的數據庫，如圖 3-1-7 所示。

```
mnist = tf.keras.datasets.fashion_mnist

(training_images, training_labels),(test_images,test_labels) = mnist.load_data()
```

圖 3-1-7 讀取數據

第三步檢查是否正確讀取數據庫，以免資料有所遺失，如圖 3-1-8 所示。

```
print( 'training_image' + str(training_images.shape)) #資料大小
print( 'training_label' + str(training_labels.shape))
print( 'test_image' + str(test_images.shape))
print( 'test_label' + str(test_labels.shape))
```

```
training_image(60000, 28, 28)
training_label(60000,)
test_image(10000, 28, 28)
test_label(10000,)
```

圖 3-1-8 查看資料

第四步為取前 16 張圖片，來檢查處理過的資料是否正常可顯示，如圖 3-1-9 所示。

```
for num in range(0,16):
    plt.subplot(4,4,num+1)
    plt.title('[%d]Label: %d' % (num,training_labels[num]))
    plt.imshow(training_images[num],      cmap=plt.get_cmap('gray_r'))
plt.tight_layout()
plt.show()
```

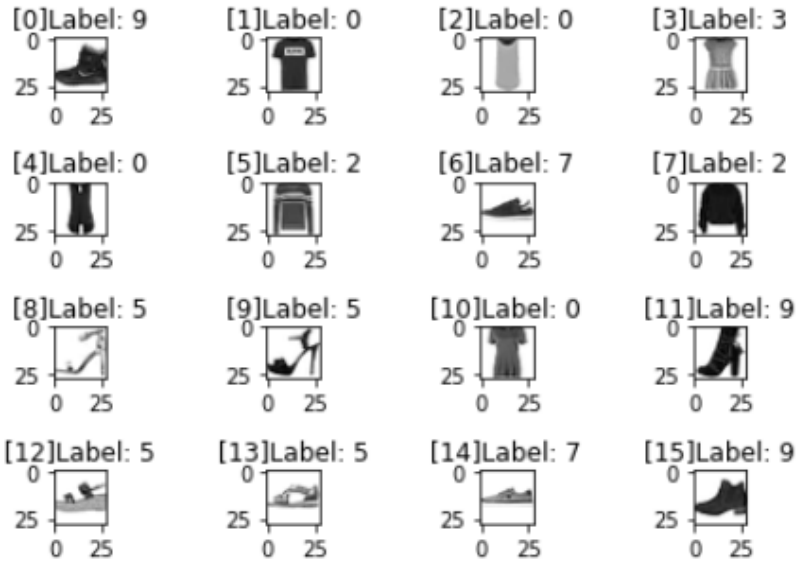


圖 3-1-9 資料樣式

第五步則是以第一張圖為例，顯示每行每列的特徵值(圖像的 RGB 值)，並顯示第一張圖的圖片，如圖 3-1-10 至 3-1-12 所示。

```
plt.imshow(training_images[0])      #訓練圖
def printMatrixE(a):
    rows = a.shape[0]
    cols = a.shape[1]
    for i in range (0,rows):
        str1=""
        for j in range(0,cols):
            str1=str1+("%3.0f" % a[i,j])
        print(str1)
    print("")

printMatrixE(training_images[0])
```

圖 3-1-10 印出色彩值

```
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0 13 73   0   0   1   4   0   0   0   0   1   1   0
  0   0   0   0   0   0   0   0   0   0   0   0   3   0 36136127 62 54   0   0   0   1   3   4   0   0   3
  0   0   0   0   0   0   0   0   0   0   0   0   6   0102204176134144123 23   0   0   0   0 12 10   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0155236207178107156161109 64 23 77130 72 15
  0   0   0   0   0   0   0   0   0   0   0   1   0 692072232182162161631271211221461 41 88172 66
  0   0   0   0   0   0   0   0   0   1   1   1   0200232232233229223232152131641271231962 29   0
  0   0   0   0   0   0   0   0   0   0   0   0   0183225216223228235227224222224221223245173   0
  0   0   0   0   0   0   0   0   0   0   0   0   0193228218213198180212210211213223202432 02   0
  0   0   0   0   0   0   0   0   0   0   1   3   0 122192202122181921692272082182242122261972 09 52
  0   0   0   0   0   0   0   0   0   0   0   6   0 992442222202182031982212152132222202451191 67 56
  0   0   0   0   0   0   0   0   0   0   4   0   0 552362282302282402322132182232342172172 09 92   0
  0   0   1   4   6   7   2   0   0   0   0   0237226217223222219222221216223229215218255 77   0
  0   3   0   0   0   0   0   0   0 621452042282072132212182082112182242232192152242441 59   0
  0   0   0   0 18 44 821071892282202222172262002052112302242341761882502482332382 15   0
  0 57187208224221224208204214208209200159245193206223255255221234221211220232246   0
  3202228224221211211214205205205220240 80150255229221188154191210204209222228225   0
98233198210222229229234249220194215217241 65 731061171682192211521722322322422 9 29
75204212204193205211225216185197206198213240195227245239223218212209222202212 30 67
48203183194213197185190194192202214219221220236225216199206186181177172181205206115
 01222191931791711831962042102132072112102001961941911951911981921761561671772 10 92
  0   0 74189212191175172175181185188189188193198204209210210211188188194192216170   0
  2   0   0   0 66200222237239242246243244221220193191179182182181176166168 99 58   0   0
  0   0   0   0   0   0   0 40 61 44 72 41 35   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

圖 3-1-11 查看色彩值

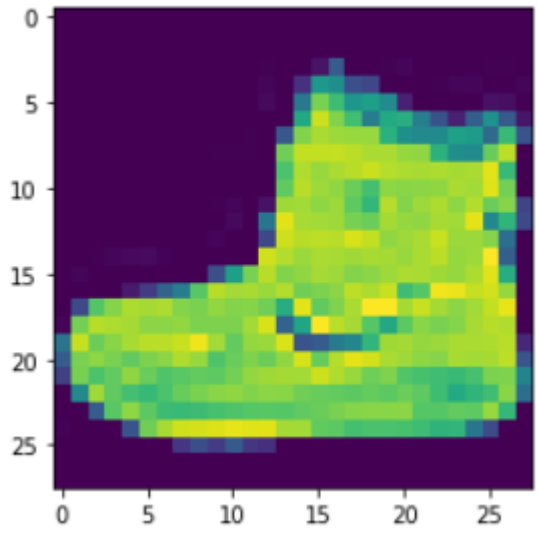


圖 3-1-12 查看原圖

第六步由於灰階影像的值是 0~255，我們選擇將全數除以 255.0 來等比例縮小，如圖 3-1-13 所示。

```
training_images  = training_images / 255.0
test_images = test_images / 255.0
```

圖 3-1-13 等比例縮小

第七步則開始建立模型，確定 Input 格式為 Sequential()，並設定 3 層的處理，接著設定每一層需做的處理：第一層為 Flattening layer(展平層)。第二層為全連接層，設置 128 個神經元，激活函數為非線性激活函數（ ReLU 函數）。第三層則輸出 10 維的向量，分別代表這張圖片屬於 0 到 9 的機率，且機率值介於 [0,1] 之間，總和等於 1，適合多分類使用，如圖 3-1-14 所示。

```
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(128, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

圖 3-1-14 建立模型

第八步確立目標及求解方法：以 compile 函數、定義損失函數(loss)、、優化函數(optimizer)及、成效衡量指標(mertrics)，如圖 3-1-15 所示。

```
model.compile(optimizer = tf.keras.optimizers.Adam(),     #編譯模型 優化函數
              loss = 'sparse_categorical_crossentropy',  #遺失函數
              metrics=['accuracy'])                      #正確率

model.fit(training_images, training_labels, epochs=50,validation_split=0.2,validation_freq=20)
```

圖 3-1-15 建立目標

第九步開始訓練模型，訓練 50 次訓練集，從測試集劃分 80%給訓練集，測試的間隔為 20 次。可看出模型的準確率隨著訓練次數上升，並且沒有發生過度擬合的狀況，如圖 3-1-16 所示。

```
Epoch 1/50
1500/1500 [==============================] - 3s 2ms/step - loss: 0.5174 - accuracy: 0.8185

Epoch 20/50
1500/1500 [==============================] - 3s 2ms/step - loss: 0.1772 - accuracy: 0.9328 - val_loss: 0.3438 - val_accuracy:
0.8908

Epoch 40/50
1500/1500 [==============================] - 3s 2ms/step - loss: 0.1095 - accuracy: 0.9594 - val_loss: 0.4476 - val_accuracy:
0.8926

Epoch 50/50
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0885 - accuracy: 0.9675
```

圖 3-1-16 訓練模型

第十步顯示目前的網路架構，如圖 3-1-17 所示。

```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (32, 784)                 0

 dense (Dense)               (32, 128)                 100480

 dense_1 (Dense)             (32, 10)                  1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
None
```

圖 3-1-17 網路架構

第十一步測試模型，將訓練完成的模型利用 Evaluation()，計算成效，可看出真實的準確率達到 88%，如圖 3-1-18 所示。

```
model.evaluate(test_images, test_labels)  #測試
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.5888 - accuracy: 0.8843

[0.5887990593910217, 0.8842999935150146]
```

圖 3-1-18 測試模型

第十二步則使用第一張圖片進行測試，可由點陣圖看出準確率是很高的，如圖 3-1-19、3-1-20 所示。

```
In [11]: classifications = model.predict(test_images)   #預測
         print("預測值:",classifications[0])
         print("")
         x=[0,1,2,3,4,5,6,7,8,9]  #製圖
         y=classifications[0]
         values = ['T-shirt','Pants', 'Pullover', 'Dress', 'Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot']
         plt.subplot(2,1,1)
         plt.scatter(x,y,c="r")
         plt.xticks(x,values,rotation=-15)
         max=0
         for i in range(10):
             if max<y[i]:
                 max=y[i]
                 ans=i
         for i in range(10):
             if ans==i:
                 if test_labels[0]==ans:
                     print("預測是",values[i],"正確答案是",values[i])
         plt.subplot(2, 1, 2)
         plt.tight_layout()
         plt.imshow(test_images[0])
```

圖 3-1-19 預測模型

```
預測值: [2.4931155e-09 4.4338363e-11 2.1504341e-09 1.6690190e-10 2.6152677e-11
 1.9276184e-05 4.2811646e-08 6.2668854e-03 2.9929328e-08 9.9371380e-01]

預測是 Ankle boot 正確答案是 Ankle boot
```

```
Out[11]: <matplotlib.image.AxesImage at 0x1a86420f460>
```

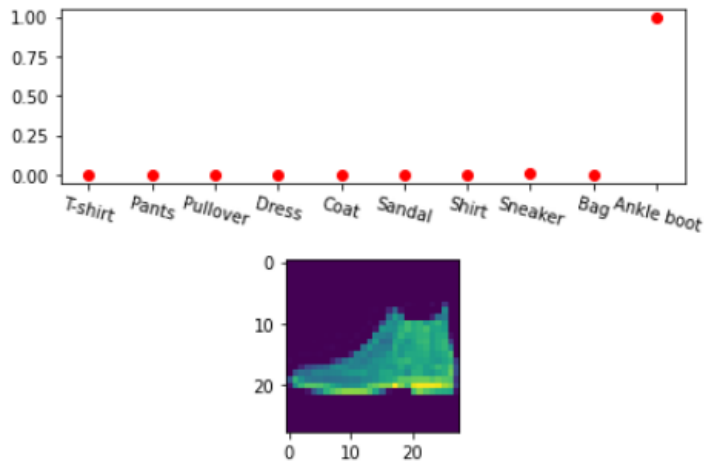


圖 3-1-20 預測結果

第十三步使用隨機亂數，取一張照片進行測試，預測是正確的，由點陣圖看出準確率是很高的，如圖 3-1-21、3-1-22 所示。

```
In [12]: classifications = model.predict(test_images)   #預測
test_random=random.randint(0,9999)
print("預測值:",classifications[test_random])
print("")
x=[0,1,2,3,4,5,6,7,8,9]  #劃圖
y=classifications[test_random]
values = ['T-shirt','Pants', 'Pullover', 'Dress', 'Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot']
plt.subplot(2,1,1)
plt.scatter(x,y,c="r")
plt.xticks(x,values,rotation=-15)
max=0
for i in range(10):
    if max<y[i]:
        max=y[i]
        ans=i
for i in range(10):
    if ans==i:
        if test_labels[test_random]==ans:
            print("預測是",values[i],"正確答案是",values[i])
plt.subplot(2, 1, 2)
plt.tight_layout()
plt.imshow(test_images[test_random])
```

圖 3-1-21 預測模型

```
預測值: [4.5625580e-11 1.1843767e-14 3.7147394e-09 2.2432180e-14 1.0401297e-10
 1.0000000e+00 1.2798133e-10 1.5596427e-10 1.6952010e-08 1.4445947e-14]

預測是 Sandal 正確答案是 Sandal

Out[12]: <matplotlib.image.AxesImage at 0x1a864327070>
```

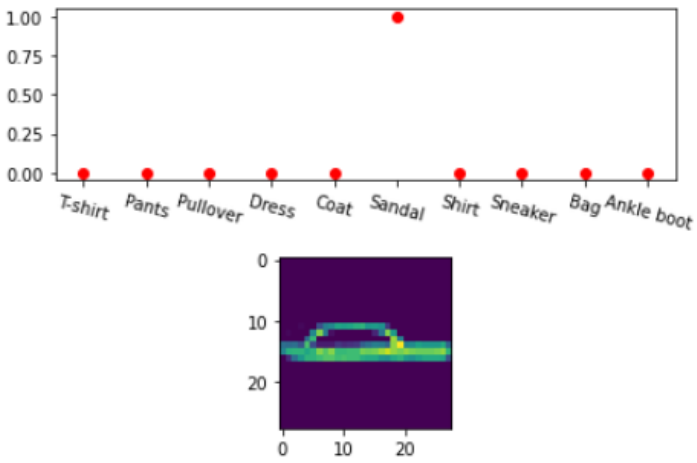


圖 3-1-22 預測結果

第六項不同參數下的結果

首先改變神經元數量，將 128 改為 512，最明顯的差別為時間，訓練時間原本只需要 5~10 分鐘，改變神經元數量之後則需要 10 分鐘以上，但準確率可上升至 97%，整體的準確率也比只有 128 個神經元時還要高，

如圖 3-1-23、3-1-24 所示。

```
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(512, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

圖 3-1-23 改變神經元數量

```
Epoch 1/50
1500/1500 [==============================] - 4s 2ms/step - loss: 0.4953 - accuracy: 0.8236

Epoch 20/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1544 - accuracy: 0.9423 - val_loss: 0.3446 - val_accuracy:
0.8931

Epoch 40/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0904 - accuracy: 0.9664 - val_loss: 0.4918 - val_accuracy:
0.8925

Epoch 50/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0723 - accuracy: 0.9731
```

圖 3-1-24 實驗結果

第二節　實驗 2

對於實驗 2，我們改變實驗的模型、優化器，其餘的實驗平台、環境，以及資料來源皆與實驗 1 相同，因此在此節將會一一說明實驗的模型、優化器。

第一項 實驗模型

在模型的部分我們使用 RNN 模型循環神經網絡，RNN 的 output 不只受上一層輸入的影響，也受到同一層前一個 output 的影響。

第二項 實驗優化器

我們使用’AdamOptimizer()’優化器，為隨機梯度下降算法的擴展式，

近來其廣泛用於深度學習應用中，尤其是計算機視覺和自然語言處理等任務。AdamOptimizer() 優化器的優點，第一個為適應性梯度算法（AdaGrad），可以每一個參數保留一個學習率以提升在稀疏梯度上的性能。第二個為均方根傳播（RMSProp），基於權重梯度最近量級的均值，為每一個參數適應性地保留學習率。

在實驗 2 我們使用的模型為 RNN 模型循環神經網絡，RNN 的 output 不只受上一層輸入的影響，也受到同一層前一個 output 的影響。

第三項 程式碼講解

第一步先匯入我們程式所需的函式庫模組，在最後一行新增“os.environ[“KMP_DUPLICATE_LIB_OK”]=”TURE” ，因為在 windows 顯示圖像的時候可能會遇到一個錯誤，需要添加這個語句才可以正常通過，意思是允許重複加載動態鏈接庫，如圖 3-2-1 所示。

```
In [1]: from tensorflow import keras                                    匯入函式庫
        import tensorflow as tf
        import tkinter as tk

        from tkinter import filedialog
        import matplotlib.pyplot as plt
        import numpy as np
        import random
        import os
        os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
```

圖 3-2-1 匯入函式庫

第二步讀取 MNIST 的數據庫，如圖 3-2-2 所示。

```
In [2]: mnist = tf.keras.datasets.fashion_mnist

        (training_images, training_labels),(test_images,test_labels) = mnist.load_data()
```

圖 3-2-2 讀取數據

第三步展開結構，如圖 3-2-3 所示。

```
In [3]: EAGER = True
```

圖 3-2-3 展開結構

第四步則開始建立模型，確定 Input 格式為 Sequential()，使用 RNN 模型，Input 圖像 大小為 28*28；將神經元數量定為 256 個；因為 RNN 會考慮同一層前面的輸出，當參數 unroll=True 時，表示計算時會先展開結構，它會使用較多的記憶體，但會縮短計算時間；Dropout 設為 0.2，Dropout 可以作為訓練深度神經網路的一種方法，在每個訓練中，通過忽略一半的特徵檢測器（讓一半的隱層節點值為 0），可以明顯地減少過擬合現象；最後輸出 10 維的向量，分別代表這張圖片屬於 0 到 9 的機率，且機率值介於 [0,1] 之間，總和等於 1，適合多分類使用。如圖 3-2-4 所示。

```
In [4]: model = keras.Sequential([
            keras.layers.SimpleRNN(
            input_shape=(28, 28),
            units=256,
            unroll=True),
            keras.layers.Dropout(rate=0.2),
            keras.layers.Dense(10, activation=tf.nn.softmax)
        ])
```

Input 圖像 大小為28*28
神經元數量為256個
計算時會先展開結構

圖 3-2-4 建立模型

第五步等比例縮小，由於灰階影像的值是 0~255，所以我們可以選擇全數除以 255.0 來等比例縮小，如圖 3-2-5 所示。

```
In [5]: training_images  = training_images / 255.0
        test_images = test_images / 255.0
```

圖 3-2-5 比例縮小

第六步確立目標及求解方法：設定 Learning_rate、訓練次數、以 compile 函數、定義損失函數(loss)、、優化函數(optimizer)及、成效衡量指標(mertrics)，其中 Learning_rate 值越大，則表示權值調整動作越大，如圖 3-2-6 所示。

```
In [6]: lr = 0.001
        epochs = 20
        model.compile(optimizer=tf.compat.v1.train.AdamOptimizer(lr),
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])

        model.fit(training_images, training_labels, epochs=epochs,validation_data=[test_images[:1000],test_labels[:1000]])
```

圖 3-2-6 建立目標

第七步開始訓練模型，訓練 20 次訓練集。可看出模型的準確率和訓練次數並不成正比，我們推測可能有過度擬合的狀況或是 RNN 的模型不適合我們的主題，如圖 3-2-7 所示。

```
Epoch 1/20
1875/1875 [==============================] - 20s 10ms/step - loss: 0.7159 - accuracy: 0.7423 - val_loss: 0.5519 - val_accuracy:
0.8110

Epoch 9/20
1875/1875 [==============================] - 18s 10ms/step - loss: 0.5241 - accuracy: 0.8135 - val_loss: 0.4583 - val_accuracy:
0.8410

Epoch 20/20
1875/1875 [==============================] - 19s 10ms/step - loss: 0.6146 - accuracy: 0.7837 - val_loss: 0.5849 - val_accuracy:
0.7910
```

圖 3-2-7 訓練結果

第八步顯示目前的網路架構，如圖 3-2-8 所示。

```
In [7]: print(model.summary())
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 256)               72960

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 10)                2570

=================================================================
Total params: 75,530
Trainable params: 75,530
Non-trainable params: 0
_________________________________________________________________
None
```

圖 3-2-8 網路架構

第九步測試模型，將訓練完成的模型利用 Evaluation()，計算成效，可看出真實的準確率只有 77%，如圖 3-2-9 所示。

```
In [8]: model.evaluate(test_images, test_labels)  #測試
        313/313 [==============================] - 2s 6ms/step - loss: 0.6035 - accuracy: 0.7787
Out[8]: [0.6034534573554993, 0.7786999940872192]
```

圖 3-2-9 測試模型

第十步則使用第一張圖片進行測試，可由點陣圖看出準確率沒有比第一個模型來得準確，讀取到其他的標籤也有蠻大的機率，如圖 3-2-10、3-2-11 所示。

```
In [9]: classifications = model.predict(test_images)   #預測
print("預測值:",classifications[0])
print("")
x=[0,1,2,3,4,5,6,7,8,9]  #製圖
y=classifications[0]
values = ['T-shirt','Pants', 'Pullover', 'Dress', 'Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot']
plt.subplot(2,1,1)
plt.scatter(x,y,c="r")
plt.xticks(x,values,rotation=-15)
max=0
for i in range(10):
    if max<y[i]:
        max=y[i]
        ans=i
for i in range(10):
    if ans==i:
        if test_labels[0]==ans:
            print("預測是",values[i],"正確答案是",values[i])
plt.subplot(2, 1, 2)
plt.tight_layout()
plt.imshow(test_images[0])
```

圖 3-2-10 預測模型

```
預測值: [9.4488947e-05 8.3504810e-06 3.1664444e-05 1.6888807e-04 5.3466479e-06
 7.5738117e-02 1.0259662e-04 1.3226280e-01 5.3760799e-04 7.9105014e-01]

預測是 Ankle boot 正確答案是 Ankle boot
```

Out[9]: <matplotlib.image.AxesImage at 0x1e1053ee640>

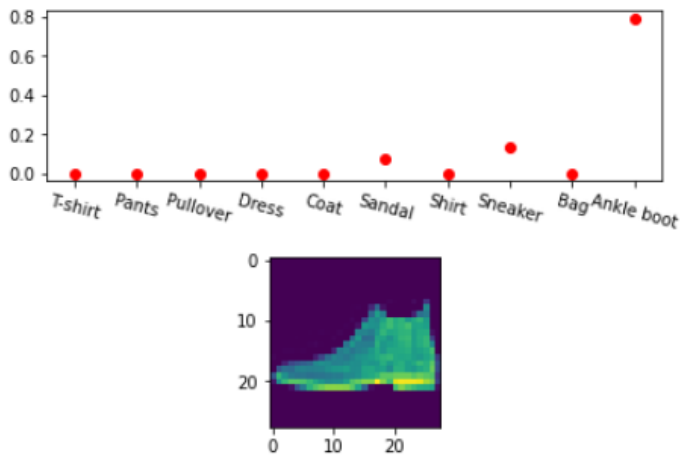


圖 3-2-11 預測結果

第十一步使用隨機亂數，取一張照片進行測試，雖然預測正確，但以點陣圖來看，讀取到其他的標籤也有蠻大的機率，如圖 3-2-12、3-2-13 所示。

```
In [10]: classifications = model.predict(test_images)   #預測
test_random=random.randint(0,9999)
print("預測值:",classifications[test_random])
print("")
x=[0,1,2,3,4,5,6,7,8,9]  #製圖
y=classifications[test_random]
values = ['T-shirt','Pants', 'Pullover', 'Dress', 'Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot']
plt.subplot(2,1,1)
plt.scatter(x,y,c="r")
plt.xticks(x,values,rotation=-15)
max=0
for i in range(10):
    if max<y[i]:
        max=y[i]
        ans=i
for i in range(10):
    if ans==i:
        if test_labels[test_random]==ans:
            print("預測是",values[i],"正確答案是",values[i])
plt.subplot(2, 1, 2)
plt.tight_layout()
plt.imshow(test_images[test_random])
```

圖 3-2-12 預測模型

```
預測值: [0.04801431 0.18924282 0.35494587 0.00326234 0.04657973 0.00407555
 0.3135744  0.00218712 0.03647966 0.00163822]

預測是 Pullover 正確答案是 Pullover
```

```
Out[13]: <matplotlib.image.AxesImage at 0x1e10bc37ee0>
```

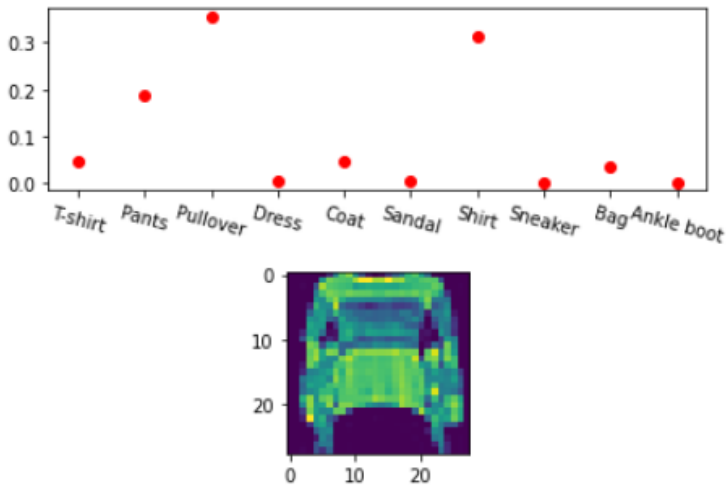


圖 3-2-13 預測結果

第四項不同參數下的結果

第一個改變神經元數量，將 128 改為 512，如圖。最明顯的差別一樣為時間，訓練時間原本只需要 5~10 分鐘，改變神經元數量之後則需要 10 分鐘以上，而對於準確率，卻沒有所提升，甚至比第一次訓練時還低，

如圖 3-2-14、3-2-15 所示。

```
In [4]: model = keras.Sequential([
            keras.layers.SimpleRNN(
            input_shape=(28, 28),
            units=512,
            unroll=True),
            keras.layers.Dropout(rate=0.2),
            keras.layers.Dense(10, activation=tf.nn.softmax)
        ])
```

如圖 3-2-14 改變神經元數量

```
Epoch 1/20
1875/1875 [==============================] - 19s 9ms/step - loss: 0.9824 - accuracy: 0.6395 - val_loss: 0.6625 - val_accuracy:
0.7650

Epoch 5/20
1875/1875 [==============================] - 17s 9ms/step - loss: 0.6723 - accuracy: 0.7586 - val_loss: 0.5692 - val_accuracy:
0.7960

Epoch 20/20
1875/1875 [==============================] - 18s 10ms/step - loss: 0.8068 - accuracy: 0.7063 - val_loss: 0.7021 - val_accuracy:
0.7520
```

如圖 3-2-15 訓練結果

第二個改變 Learning_rate 值與訓練次數，Learning_rate 從 0.001 改為 0.05，訓練次數則改為 5 次，由結果可知，整體的準確率慘不忍睹...，只有 10% 甚至更少，如圖 3-2-16 至 3-2-21 所示。

```
In [6]: lr = 0.05
        epochs = 5
        model.compile(optimizer=tf.compat.v1.train.AdamOptimizer(lr),
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])

        model.fit(training_images, training_labels, epochs=epochs,validation_data=[test_images[:1000],test_labels[:1000]])

        Epoch 1/5
        1875/1875 [==============================] - 15s 8ms/step - loss: 3.7724 - accuracy: 0.1439 - val_loss: 2.5640 - val_accuracy:
        0.1360
        Epoch 2/5
        1875/1875 [==============================] - 14s 8ms/step - loss: 3.6016 - accuracy: 0.1445 - val_loss: 3.0090 - val_accuracy:
        0.1730
        Epoch 3/5
        1875/1875 [==============================] - 14s 8ms/step - loss: 3.6039 - accuracy: 0.1469 - val_loss: 3.4019 - val_accuracy:
        0.0760
        Epoch 4/5
        1875/1875 [==============================] - 15s 8ms/step - loss: 3.6035 - accuracy: 0.1484 - val_loss: 3.2796 - val_accuracy:
        0.1550
        Epoch 5/5
        1875/1875 [==============================] - 15s 8ms/step - loss: 3.6305 - accuracy: 0.1456 - val_loss: 3.4503 - val_accuracy:
        0.1840
```

如圖 3-2-16 改變 Learning_rate 值

```
In [7]: print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 256)               72960

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 10)                2570

=================================================================
Total params: 75,530
Trainable params: 75,530
Non-trainable params: 0
_________________________________________________________________
None
```

```
In [8]: model.evaluate(test_images, test_labels)  #測試
```

```
313/313 [==============================] - 1s 5ms/step - loss: 3.4293 - accuracy: 0.1723
Out[8]: [3.429255485534668, 0.17229999601840973]
```

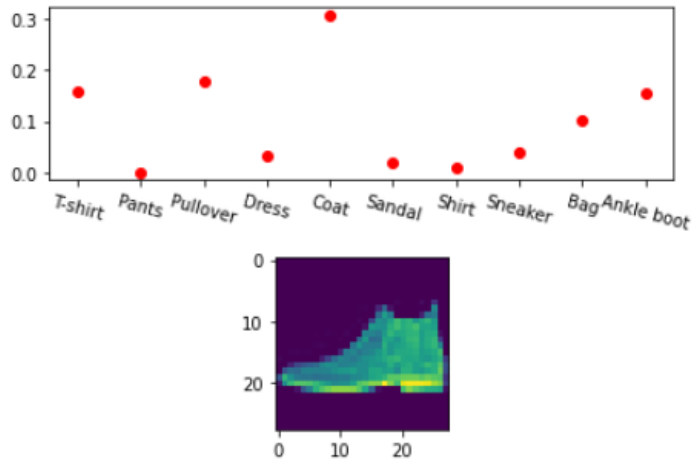
如圖 3-2-17 改變結果

```
In [9]: classifications = model.predict(test_images)   #預測
print("預測值:",classifications[0])
print("")
x=[0,1,2,3,4,5,6,7,8,9]  #製圖
y=classifications[0]
values = ['T-shirt','Pants', 'Pullover', 'Dress', 'Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot']
plt.subplot(2,1,1)
plt.scatter(x,y,c="r")
plt.xticks(x,values,rotation=-15)
max=0
for i in range(10):
    if max<y[i]:
        max=y[i]
        ans=i
for i in range(10):
    if ans==i:
        if test_labels[0]==ans:
            print("預測是",values[i],"正確答案是",values[i])
plt.subplot(2, 1, 2)
plt.tight_layout()
plt.imshow(test_images[0])
```

如圖 3-2-18 測試模型

```
預測值: [1.5795982e-01 1.1987691e-05 1.7710276e-01 3.3767853e-02 3.0725038e-01
 1.9753838e-02 1.0102864e-02 3.8485147e-02 1.0144544e-01 1.5411994e-01]
```

```
Out[9]: <matplotlib.image.AxesImage at 0x29b49f09a60>
```



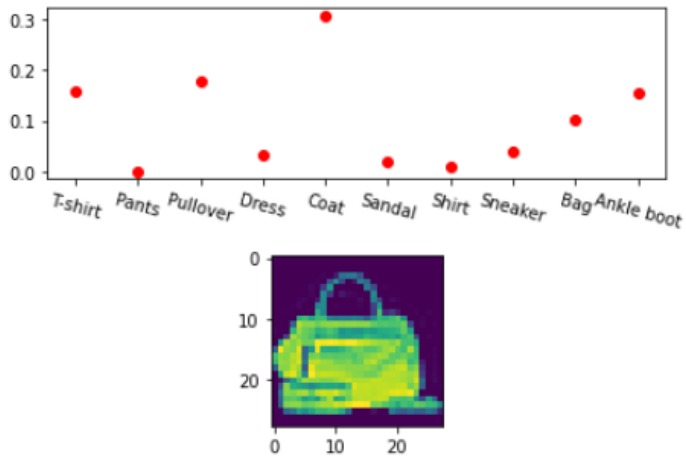
如圖 3-2-19 測試結果

```
In [12]:
classifications = model.predict(test_images)   #預測
test_random=random.randint(0,9999)
print("預測值:",classifications[test_random])
print("")
x=[0,1,2,3,4,5,6,7,8,9]  #製圖
y=classifications[test_random]
values = ['T-shirt','Pants', 'Pullover', 'Dress', 'Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot']
plt.subplot(2,1,1)
plt.scatter(x,y,c="r")
plt.xticks(x,values,rotation=-15)
max=0
for i in range(10):
    if max<y[i]:
        max=y[i]
        ans=i
for i in range(10):
    if ans==i:
        if test_labels[test_random]==ans:
            print("預測是",values[i],"正確答案是",values[i])
plt.subplot(2, 1, 2)
plt.tight_layout()
plt.imshow(test_images[test_random])
```

如圖 3-2-20 測試模型

```
預測值: [1.5795982e-01 1.1987691e-05 1.7710276e-01 3.3767853e-02 3.0725038e-01
 1.9753838e-02 1.0102864e-02 3.8485147e-02 1.0144544e-01 1.5411994e-01]

Out[12]: <matplotlib.image.AxesImage at 0x29b5563b9a0>
```



如圖 3-2-21 測試結果

第四章　結論與未來工作

做完這個專題之後我們對於機器學習更加了解，因為平常都是聽老師講解，也沒有實際打過程式碼或是看過程式碼，所以一開始在決定主題的時候我們苦惱很久。在找到主題之後也一直在思考要底要用什麼平台，要用 Spyder 還是 Jupyter 呢？資料要怎麼收集、怎麼標籤、要用什麼模型……等等的問題。確定主題之後大家開始分工收集資料、找方法、改成我們自己的模型，直到最後終於完成一個模型，也確定是可用的模型之後，大家也越來越上手，於是想到有沒有其他的模型可以用在我們的專題，雖然另一個模型的結果和第一個模型相較起來有落差，但大家對於機器學習也都學習到更多的知識、與實作的經驗。

在未來工作方面，除了機器學習可以選擇以外，還多了影像識別方面的選擇，也因此有了多元化的選擇。

第五章　參考文獻

[1] https://zhuanlan.zhihu.com/p/367066452

[2] https://github.com/zalandoresearch/fashion-mnist#get-the-data

[3] https://officeguide.cc/pytorch-deep-learning-library-fashion-mnist-image-classification-quick-start-tutorial-examples/

[4] https://www.itread01.com/content/1545372006.html

[5] https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-fashion-mnist-clothing-classification/

[6] https://www.kaggle.com/almahmudalmamun/fashion-mnist

[7] https://colab.research.google.com/github/tensorflow/tpu/blob/master/tools/colab/fashion_mnist.ipynb#scrollTo=ESL6ltQTMm05

[8] https://www.itread01.com/content/1544149096.html

[9] https://kknews.cc/zh-tw/news/6jnmq3.html

[10]https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/617324/

[11]https://tengyuanchang.medium.com/%E6%B7%BA%E8%AB%87%E9%81%9E%E6%AD%B8%E7%A5%9E%E7%B6%93%E7%B6%B2%E8%B7%AF-rnn-%E8%88%87%E9%95%B7%E7%9F%AD%E6%9C%9F%E8%A8%98%E6%86%B6%E6%A8%A1%E5%9E%8B-lstm-300cbe5efcc3

[12]https://ithelp.ithome.com.tw/articles/10217112

[13]https://ithelp.ithome.com.tw/articles/10247304

[14]https://ithelp.ithome.com.tw/articles/10191725?sc=iThelpR

[15]https://ithelp.ithome.com.tw/articles/10224345

[16]https://dasanlin888.pixnet.net/blog/post/462728270-%E7%B5%90%E6%A7%8B%E6%96%B9%E7%A8%8Bsem%E6%A8%A1%E5%BC%8F%E9%85%8D%E9%81%A9%E5%BA%A6%E6%8C%87%E6%A8%99(model-fit)%E4%B9%8B%E4%BB%8B%E7%B4%B9%EF%BC%881

[17]https://twagoda.com/entry/8172649

[18]https://docs.aws.amazon.com/zh_tw/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

[19]https://ithelp.ithome.com.tw/articles/10250412

[20]https://ithelp.ithome.com.tw/articles/10228941?sc=rss.qu

[21]https://www.pythonf.cn/read/111510

[22]https://ithelp.ithome.com.tw/articles/10270394?sc=iThelpR