機器學習_服裝辨識

08160971 黃凱勵
08160422 孫振寧
08160723 鄧晨言
08160130 林采葳

目錄

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。

研究方法

資料

透過MNIST的數據庫讀取資料，

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

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

屬性

服飾的特徵

每個點的灰階資料

模型

1. Sequential模型

2. 用來構建深度神經網絡

```
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)])
```

第一層是Flattening layer(展平層)，第二層為全連接層，並設置128個神經元，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

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。

損失函數

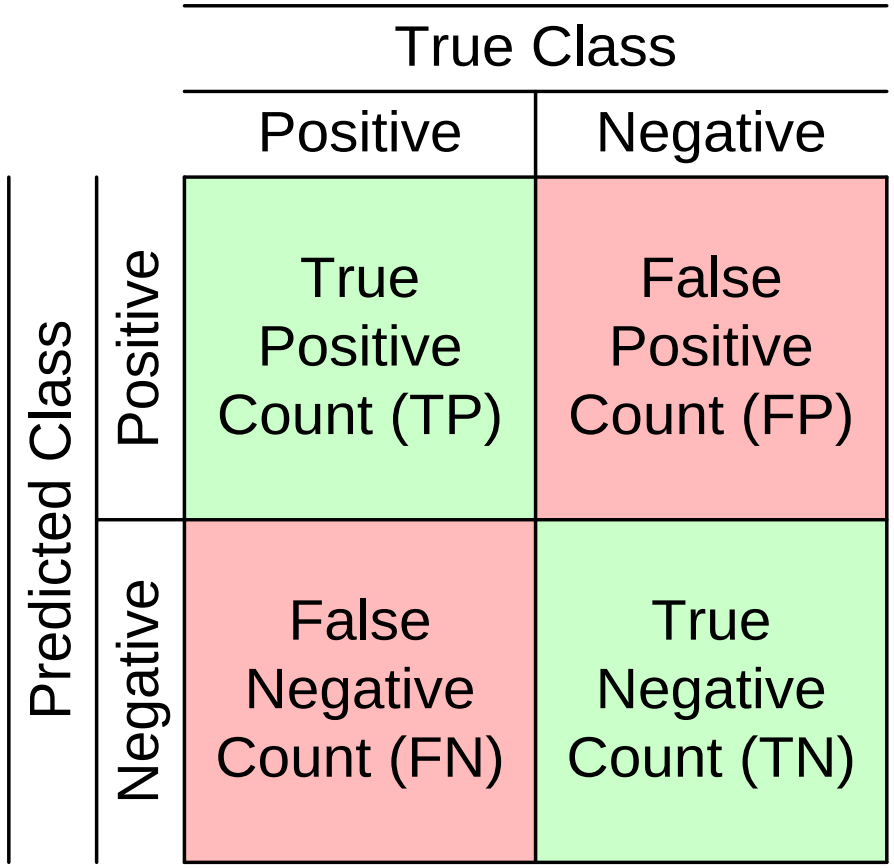
損失函數爲' sparse_categorical_crossentropy'，就是**交叉熵**，而 categorical_crossentropy 和 sparse_categorical_crossentropy這二者都是針對多分類任務。

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

成效衡量指標

成效衡量指標則是'accuracy'

(tp+tn)/(tp+fp+fn+tn)



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。

```
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=5)     #訓練5個訓練集
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4989 - accuracy: 0.8241
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3753 - accuracy: 0.8649
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3366 - accuracy: 0.8771
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3144 - accuracy: 0.8854
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2968 - accuracy: 0.8902
```

訓練數據（epochs）的設置

將epochs 設為50，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

```
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)
```

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

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

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

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

實驗1

實驗平台

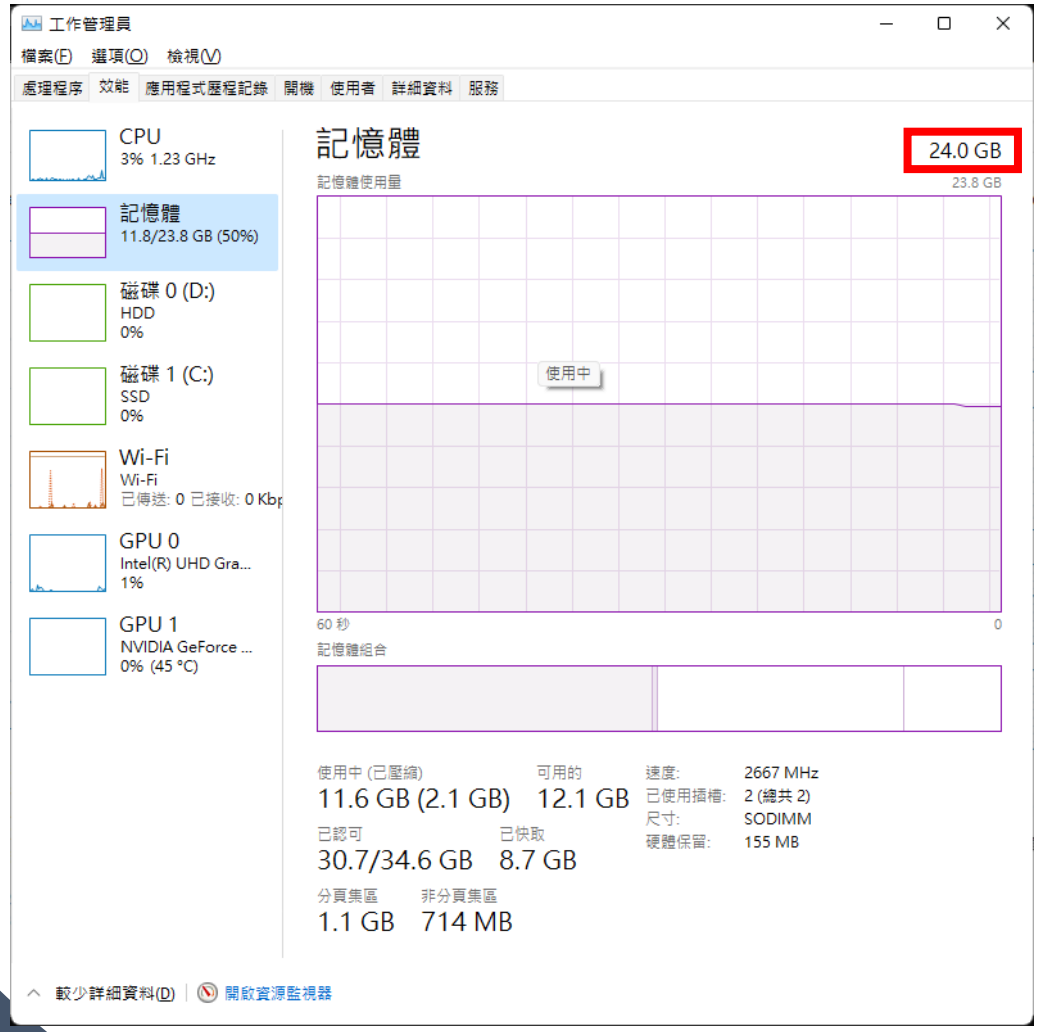
1. 使用 **python** 進行程式的撰寫。

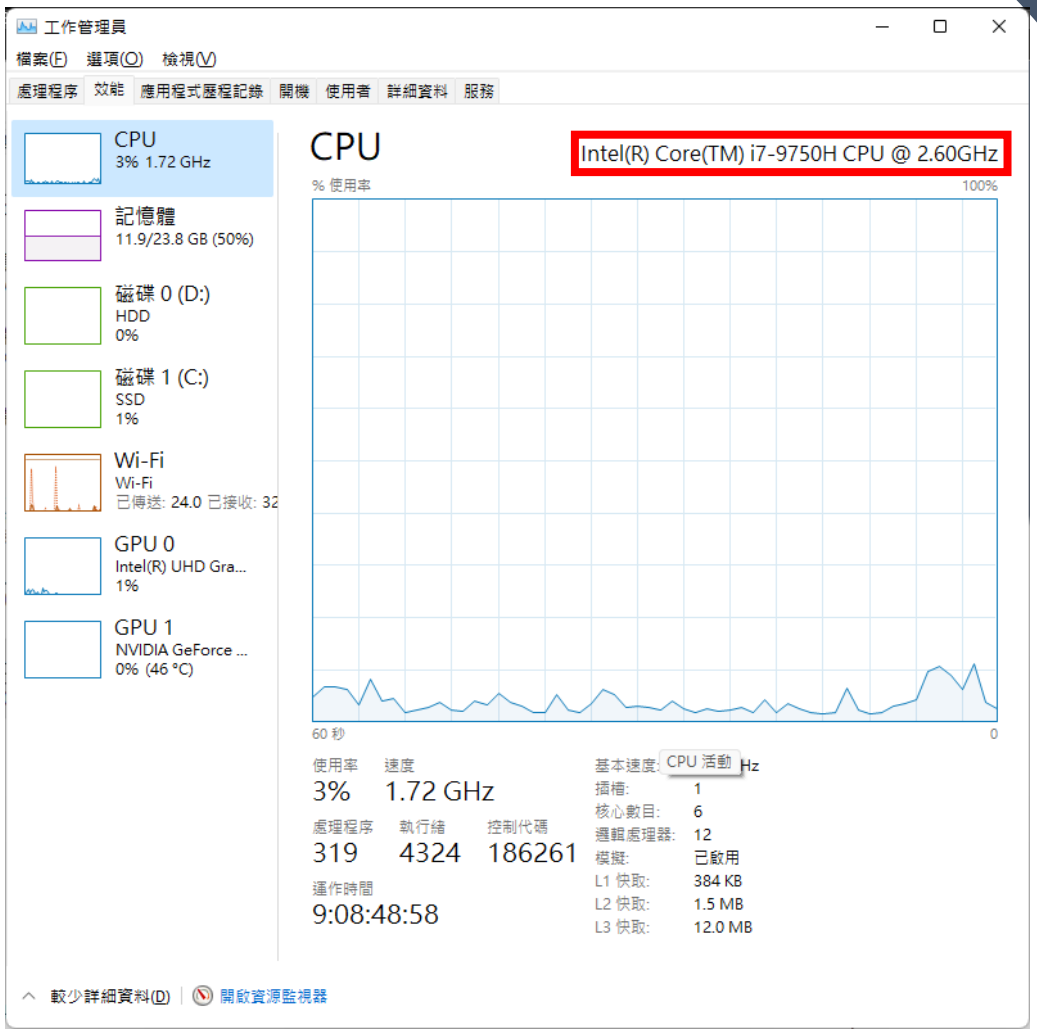
2. 能夠把**軟體代碼、計算輸出、解釋文檔、多媒體資源**整合在一起的多功能科學運算平台。

3. 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。

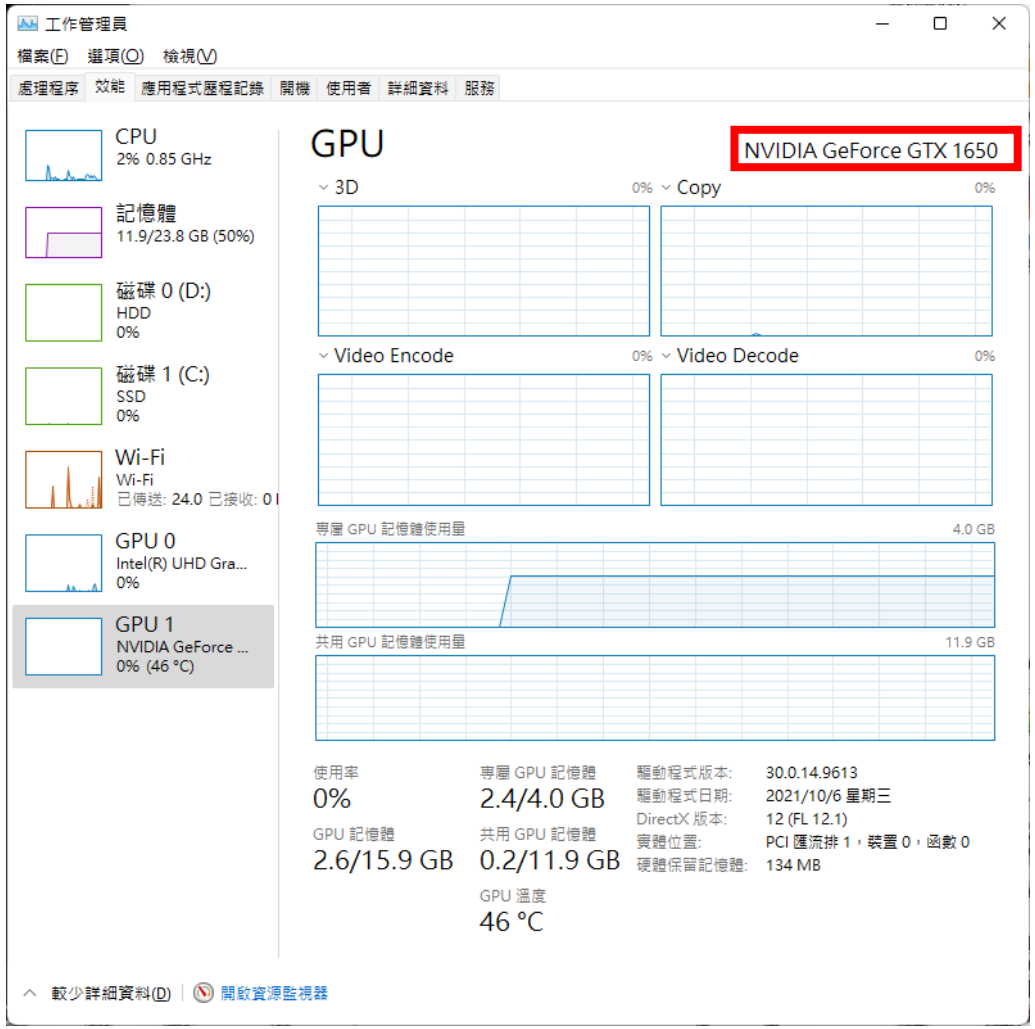
4. 每次實驗可以只跑一小個 Cell 裡的代碼，在代碼下面**立刻**就可以**看到結果**。

實驗環境


工作管理員
檔案(F) 選項(O) 檢視(V)
處理程序 效能 應用程式歷程記錄 開機 使用者 詳細資料 服務
CPU
3% 1.23 GHz
記憶體
11.8/23.8 GB (50%)
磁碟 0 (D:)
HDD
0%
磁碟 1 (C:)
SSD
0%
Wi-Fi
Wi-Fi
已傳送: 0 已接收: 0 Kbp
GPU 0
Intel(R) UHD Gra...
1%
GPU 1
NVIDIA GeForce ...
0% (45 °C)
記憶體
24.0 GB
記憶體使用量
23.8 GB
使用中
60 秒
0
記憶體組合
使用中 (已壓縮)
11.6 GB (2.1 GB)
可用的
12.1 GB
速度: 2667 MHz
已使用插槽: 2 (總共 2)
尺寸: SODIMM
硬體保留: 155 MB
已認可
30.7/34.6 GB
已快取
8.7 GB
分頁集區
1.1 GB
非分頁集區
714 MB
較少詳細資料(D)
開啟資源監視器


工作管理員
檔案(F) 選項(O) 檢視(V)
處理程序 效能 應用程式歷程記錄 開機 使用者 詳細資料 服務
CPU
3% 1.72 GHz
記憶體
11.9/23.8 GB (50%)
磁碟 0 (D:)
HDD
0%
磁碟 1 (C:)
SSD
1%
Wi-Fi
Wi-Fi
已傳送: 24.0 已接收: 32
GPU 0
Intel(R) UHD Gra...
1%
GPU 1
NVIDIA GeForce ...
0% (46 °C)
CPU
Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
% 使用率
100%
60 秒
0
使用率
3%
速度
1.72 GHz
處理程序
319
執行緒
4324
控制代碼
186261
運作時間
9:08:48:58
基本速度: CPU 活動 Hz
插槽: 1
核心數目: 6
邏輯處理器: 12
模擬: 已啟用
L1 快取: 384 KB
L2 快取: 1.5 MB
L3 快取: 12.0 MB
較少詳細資料(D)
開啟資源監視器

實驗環境


工作管理員
檔案(F) 選項(O) 檢視(V)
處理程序 效能 應用程式歷程記錄 開機 使用者 詳細資料 服務
CPU
2% 0.85 GHz
記憶體
11.9/23.8 GB (50%)
磁碟 0 (D:)
HDD
0%
磁碟 1 (C:)
SSD
0%
Wi-Fi
Wi-Fi
已傳送: 24.0 已接收: 0
GPU 0
Intel(R) UHD Gra...
0%
GPU 1
NVIDIA GeForce ...
0% (46 °C)
GPU
NVIDIA GeForce GTX 1650
3D 0%
Copy 0%
Video Encode 0%
Video Decode 0%
專屬 GPU 記憶體使用量 4.0 GB
共用 GPU 記憶體使用量 11.9 GB
使用率
0%
專屬 GPU 記憶體
2.4/4.0 GB
驅動程式版本: 30.0.14.9613
驅動程式日期: 2021/10/6 星期三
DirectX 版本: 12 (FL 12.1)
實體位置: PCI 匯流排 1，裝置 0，函數 0
硬體保留記憶體: 134 MB
GPU 記憶體
2.6/15.9 GB
共用 GPU 記憶體
0.2/11.9 GB
GPU 溫度
46 °C
較少詳細資料(D)
開啟資源監視器

實驗 函式庫簡介

```
import tensorflow as tf
import tkinter as tk

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

import tensorflow as tf

用於**機器學習**和**深度神經網路方面**的研究

import tkinter as tk

用來在 Python 中**建構 GUI 圖形介面**程式

from tkinter import filedialog

開啟**檔案對話框**用法

import matplotlib.pyplot as plt

用來**繪圖**、**圖表呈現**及**數據表示**

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

import random

匯入標準模組庫中(standard library)的**亂數模組(random)**

實驗 資料來源

訓練資料與測試資料數量皆為MNIST數據庫

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

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

實驗 資料筆數

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

```
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,)
```

程式碼講解

```
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
```

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

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

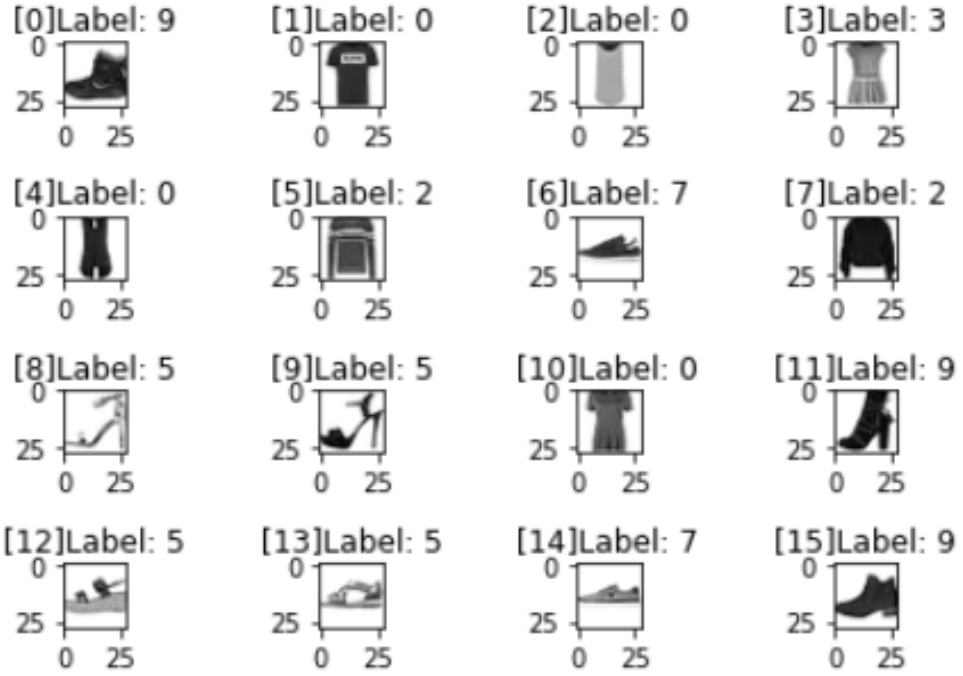
```
print( 'training_image' + str(training_images.shape)) #資料大小
print( 'training_label' + str(training_labels.shape))
print( 'test_image' + str(test_images.shape))
print( 'test_label' + str(test_labels.shape))
```

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

程式碼講解

```
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()
```

取前16張圖來檢查處理過的資料是否正常可顯示

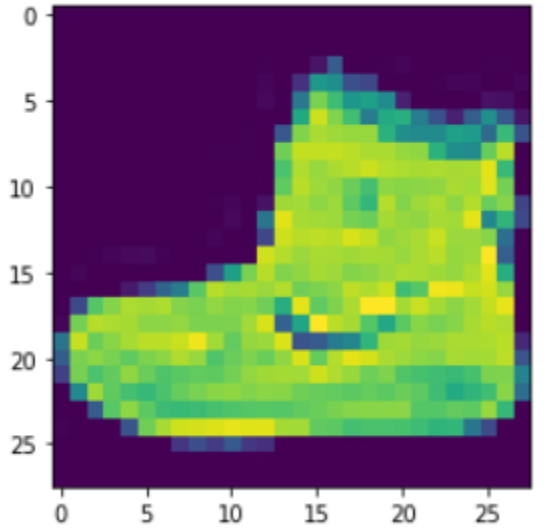


```python
plt.imshow(training_images[0])      #訓練圖
def printMatrixE(a):
    rows = a.shape[0]
    cols = a.shape[1]
    for i in range (0,rows):
        str1=""
        for j in range(0,cols):
            str1=str1+("%3.0f" % a[i,j])
        print(str1)
    print("")

printMatrixE(training_images[0])
```

以第一張圖為例

顯示每行每列的特徵值(圖像的RGB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  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  0193228218213198180212210211213223220243202  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  6  0 99244222220218203198221215213222220245119167 56
  0  0  0  0  0  0  0  0  0  4  0  0 55236228230228240232213218223234217217209 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 62145204228207213221218208211218224223219215224244159  0
  0  0  0  0 18 44 82107189228220222217226200205211230224234176188250248233238215  0
  0 57187208224221224208204214208209200159245193206223255255221234221211220232246  0
  3202228224221211211214205205205220240 80150255229221188154191210204209222228225  0
98233198210222229229234249220194215217241 65 73106117168219221215217223223224229 29
75204212204193205211225216185197206198213240195227245239223218212209222220221230 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  0  0  0  0  0
```



程式碼講解

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

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

```
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)])
```

建立模型(Model)：

確立Input格式 => Sequential()

要經過幾層處理 =>3層

每一層要作什麼處理

第一層=>Flattening layer(展平層)，

第二層=>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）。

此模型輸入一個向量（為校正點中心的 28*28 服飾圖片)

第三層=>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

程式碼講解

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)

```
model.compile(optimizer = tf.keras.optimizers.Adam(),
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

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

訓練50個訓練集,從測試集劃分80%給訓練集,測試的間隔為20次

```
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
```

程式碼講解

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

顯示目前網路架構

```
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
```

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

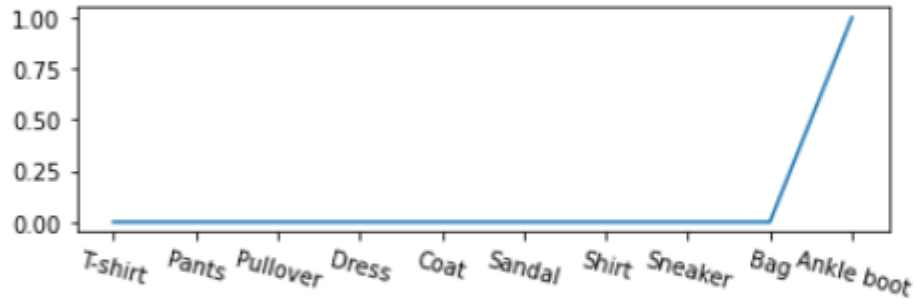
評估(Evaluation)：訓練完後，計算成效

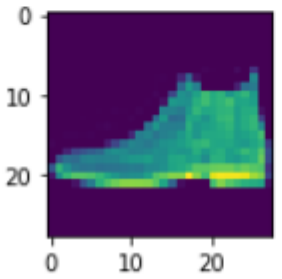
```
313/313 [==============================] - 1s 2ms/step - loss: 0.5888 - accuracy: 0.8843

[0.5887990593910217, 0.8842999935150146]
```

程式碼講解

```
classificat
print("預測
print("")
x=[0,1,2,3,
y=classific
values = ['
plt.subplot
plt.plot(x,
plt.xticks(
max=0
for i in ra
    if max<
        max
        ans
for i in ra
    if ans=
        if

plt.subplot
plt.tight_l
plt.imshow(
```

```
預測值: [3.4754174e-20 1.2578888e-25 1.1903199e-15 4.5160290e-25 6.3640621e-23
 3.6289090e-11 1.6995387e-18 2.1865737e-05 2.3209635e-20 9.9997818e-01]

預測是 Ankle boot 正確答案是 Ankle boot

<matplotlib.image.AxesImage at 0x19179262d60>
```



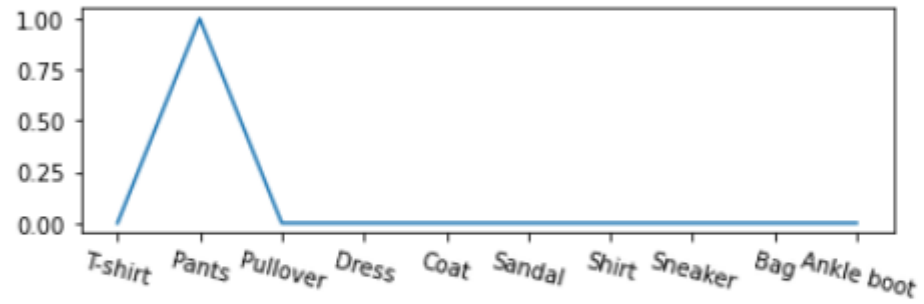


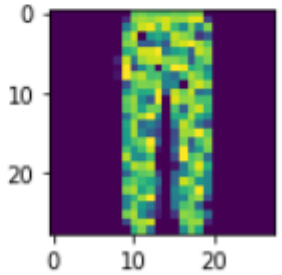
程式碼講解

```
classificati
test_random=
print("預測值
print("")
x=[0,1,2,3,4
y=classifica
values = ['T
plt.subplot(
plt.plot(x,y
plt.xticks(x
max=0
for i in ran
    if max<y
        max=
        ans=
for i in ran
    if ans==
        if t

plt.subplot(
plt.tight_la
plt.imshow(t
```

```
預測值: [2.4082777e-19 1.0000000e+00 7.3702905e-27 2.4595492e-16 7.9145728e-24
 5.2057285e-33 5.9310782e-20 1.4854267e-38 4.0079257e-25 0.0000000e+00]

預測是 Pants 正確答案是 Pants

<matplotlib.image.AxesImage at 0x1917927b7c0>
```

```
'Ankle boot']
```





實驗 不同參數下的結果

- 改變神經元數量

```
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)])
```

```
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
```

實驗2

研究方法

資料

透過MNIST的數據庫讀取資料，

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

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

屬性

服飾的特徵

每個點的灰階資料

模型

1. RNN模型循環神經網絡
2. 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

```
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)
])
```

優化器

接下來選擇優化器，我們使用’ AdamOptimizer()’優化器，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()

- 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。

- 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。

損失函數

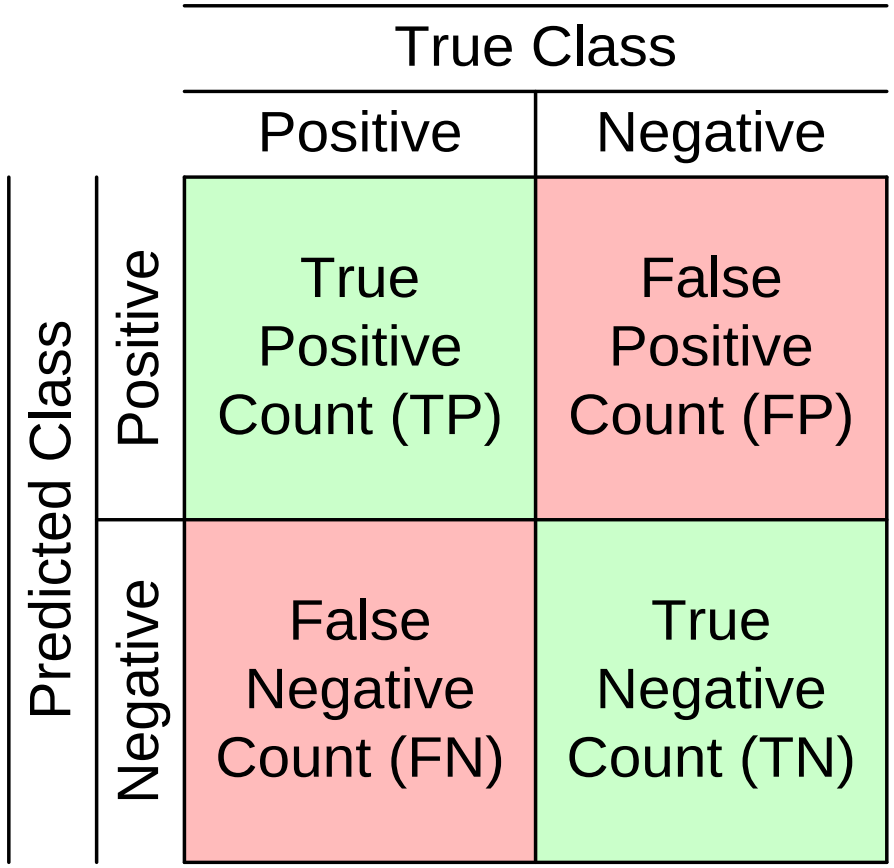
損失函數爲' sparse_categorical_crossentropy'，就是**交叉熵**，而 categorical_crossentropy 和 sparse_categorical_crossentropy這二者都是針對多分類任務。

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

成效衡量指標

成效衡量指標則是'accuracy'

(tp+tn)/(tp+fp+fn+tn)



實驗2

實驗平台

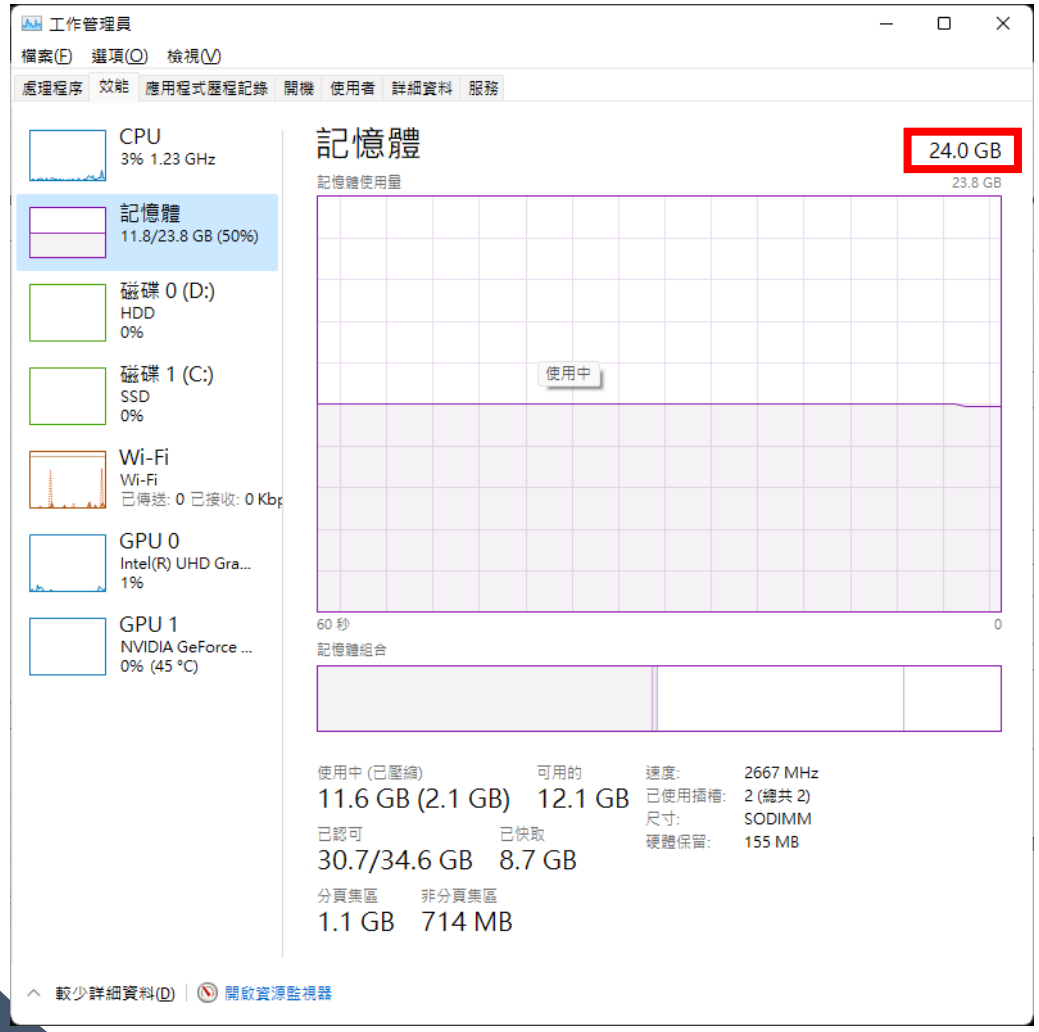
1. 使用 **python** 進行程式的撰寫。

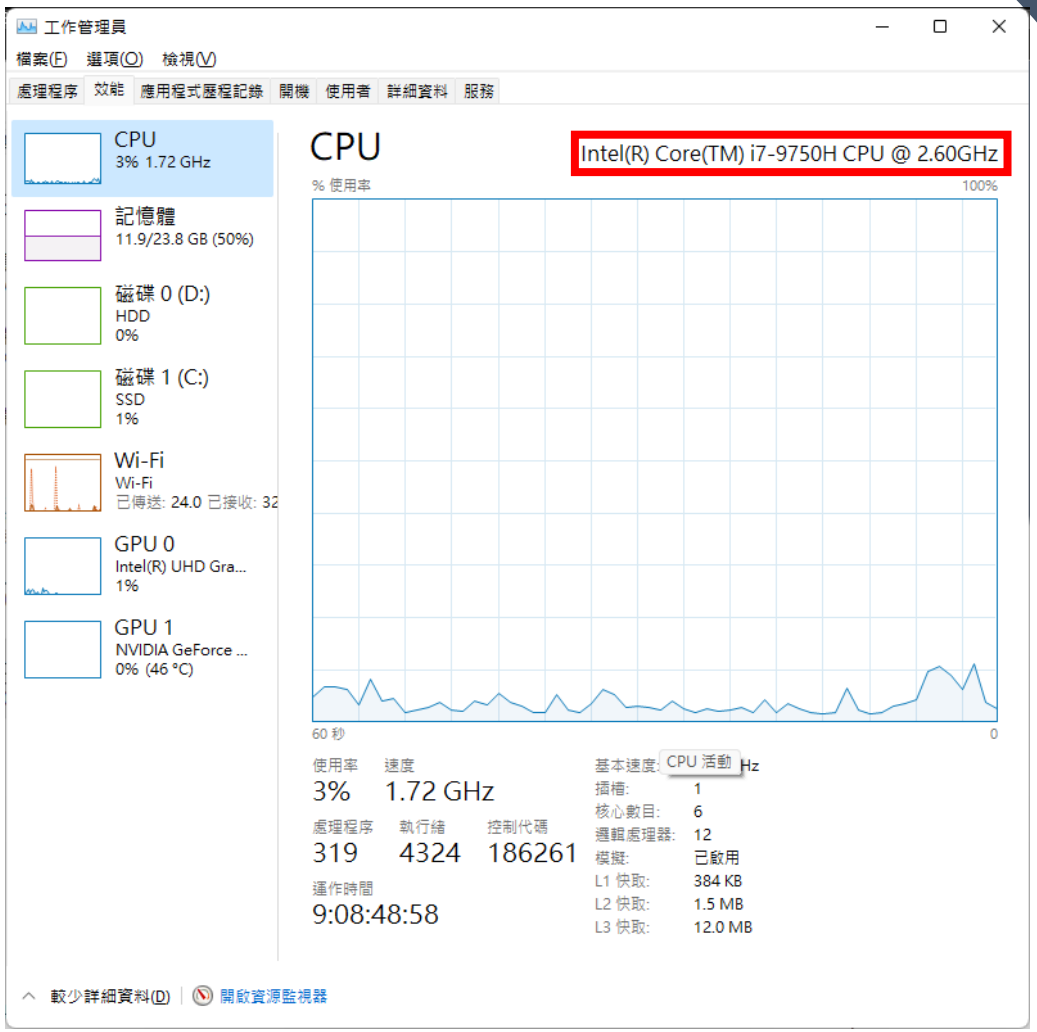
2. 能夠把**軟體代碼、計算輸出、解釋文檔、多媒體資源**整合在一起的多功能科學運算平台。

3. 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。

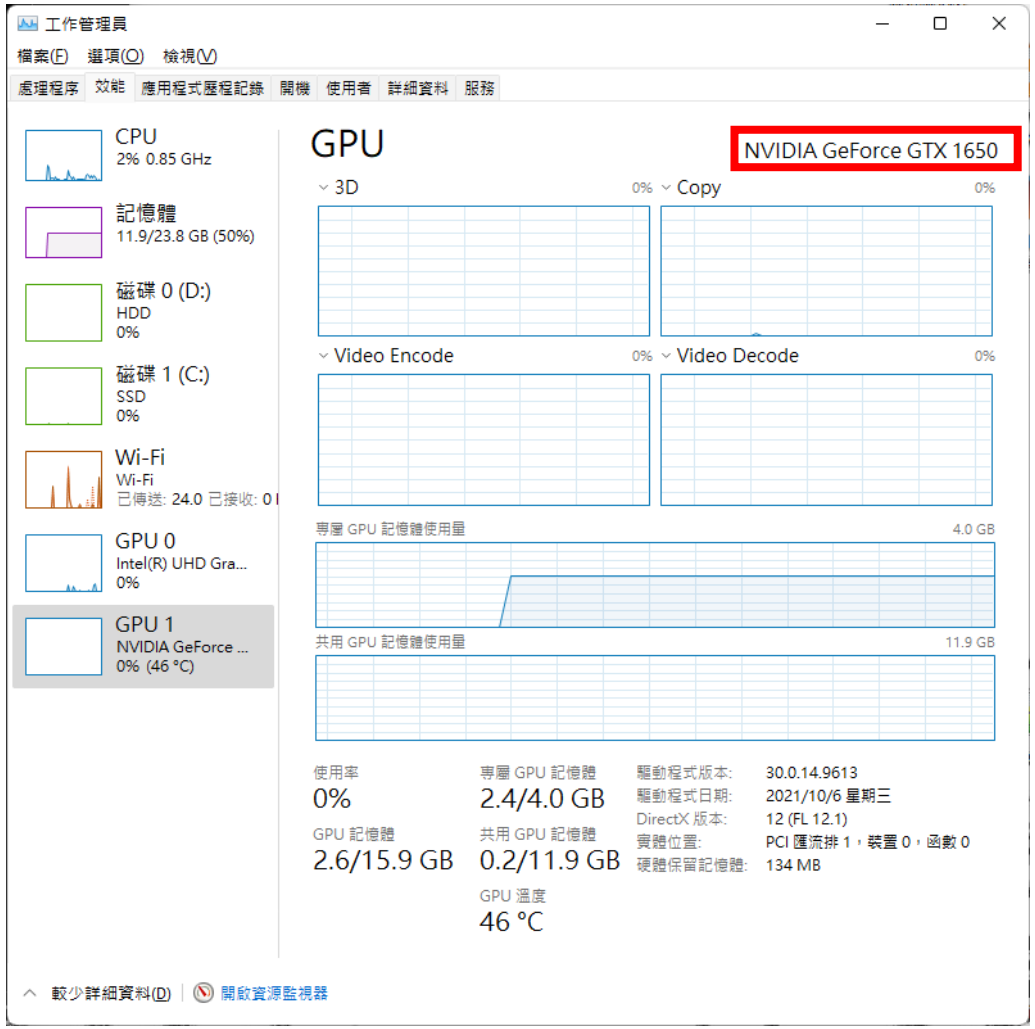
4. 每次實驗可以只跑一小個 Cell 裡的代碼，在代碼下面**立刻**就可以**看到結果**。

實驗環境


工作管理員
檔案(F) 選項(O) 檢視(V)
處理程序 效能 應用程式歷程記錄 開機 使用者 詳細資料 服務
CPU
3% 1.23 GHz
記憶體
11.8/23.8 GB (50%)
磁碟 0 (D:)
HDD
0%
磁碟 1 (C:)
SSD
0%
Wi-Fi
Wi-Fi
已傳送: 0 已接收: 0 Kbp
GPU 0
Intel(R) UHD Gra...
1%
GPU 1
NVIDIA GeForce ...
0% (45 °C)
記憶體
24.0 GB
記憶體使用量
23.8 GB
使用中
60 秒
0
記憶體組合
使用中 (已壓縮)
11.6 GB (2.1 GB)
可用的
12.1 GB
速度: 2667 MHz
已使用插槽: 2 (總共 2)
尺寸: SODIMM
硬體保留: 155 MB
已認可
30.7/34.6 GB
已快取
8.7 GB
分頁集區
1.1 GB
非分頁集區
714 MB
較少詳細資料(D)
開啟資源監視器


工作管理員
檔案(F) 選項(O) 檢視(V)
處理程序 效能 應用程式歷程記錄 開機 使用者 詳細資料 服務
CPU
3% 1.72 GHz
記憶體
11.9/23.8 GB (50%)
磁碟 0 (D:)
HDD
0%
磁碟 1 (C:)
SSD
1%
Wi-Fi
Wi-Fi
已傳送: 24.0 已接收: 32
GPU 0
Intel(R) UHD Gra...
1%
GPU 1
NVIDIA GeForce ...
0% (46 °C)
CPU
Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
% 使用率
100%
60 秒
0
使用率
3%
速度
1.72 GHz
處理程序
319
執行緒
4324
控制代碼
186261
運作時間
9:08:48:58
基本速度: CPU 活動 Hz
插槽: 1
核心數目: 6
邏輯處理器: 12
模擬: 已啟用
L1 快取: 384 KB
L2 快取: 1.5 MB
L3 快取: 12.0 MB
較少詳細資料(D)
開啟資源監視器

實驗環境


工作管理員
檔案(F) 選項(O) 檢視(V)
處理程序 效能 應用程式歷程記錄 開機 使用者 詳細資料 服務
CPU
2% 0.85 GHz
記憶體
11.9/23.8 GB (50%)
磁碟 0 (D:)
HDD
0%
磁碟 1 (C:)
SSD
0%
Wi-Fi
Wi-Fi
已傳送: 24.0 已接收: 0
GPU 0
Intel(R) UHD Gra...
0%
GPU 1
NVIDIA GeForce ...
0% (46 °C)
GPU
NVIDIA GeForce GTX 1650
3D 0%
Copy 0%
Video Encode 0%
Video Decode 0%
專屬 GPU 記憶體使用量 4.0 GB
共用 GPU 記憶體使用量 11.9 GB
使用率
0%
專屬 GPU 記憶體
2.4/4.0 GB
驅動程式版本: 30.0.14.9613
驅動程式日期: 2021/10/6 星期三
DirectX 版本: 12 (FL 12.1)
實體位置: PCI 匯流排 1，裝置 0，函數 0
硬體保留記憶體: 134 MB
GPU 記憶體
2.6/15.9 GB
共用 GPU 記憶體
0.2/11.9 GB
GPU 溫度
46 °C
較少詳細資料(D)
開啟資源監視器

實驗 函式庫簡介

```
import tensorflow as tf
import tkinter as tk

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

import tensorflow as tf

用於**機器學習**和**深度神經網路方面**的研究

import tkinter as tk

用來在 Python 中**建構 GUI 圖形介面**程式

from tkinter import filedialog

開啟**檔案對話框**用法

import matplotlib.pyplot as plt

用來**繪圖**、**圖表呈現**及**數據表示**

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

import random

匯入標準模組庫中(standard library)的**亂數模組(random)**

程式碼講解

In [1]:
```
from tensorflow import keras
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"
```

匯入函式庫

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

程式碼講解

訓練資料與測試資料數量皆為MNIST數據庫

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

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

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

計算時會先展開結構

程式碼講解

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

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。

這種方式可以減少特徵檢測器（隱層節點）間的相互作用

程式碼講解

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

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

程式碼講解

Learning_rate

學習速率，值越大則表示權值調整動作越大

訓練數據（epochs）的設置 = 20

```
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]])
```

```
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
```

程式碼講解

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

顯示目前網路架構

```
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
```

評估(Evaluation)：訓練完後，計算成效

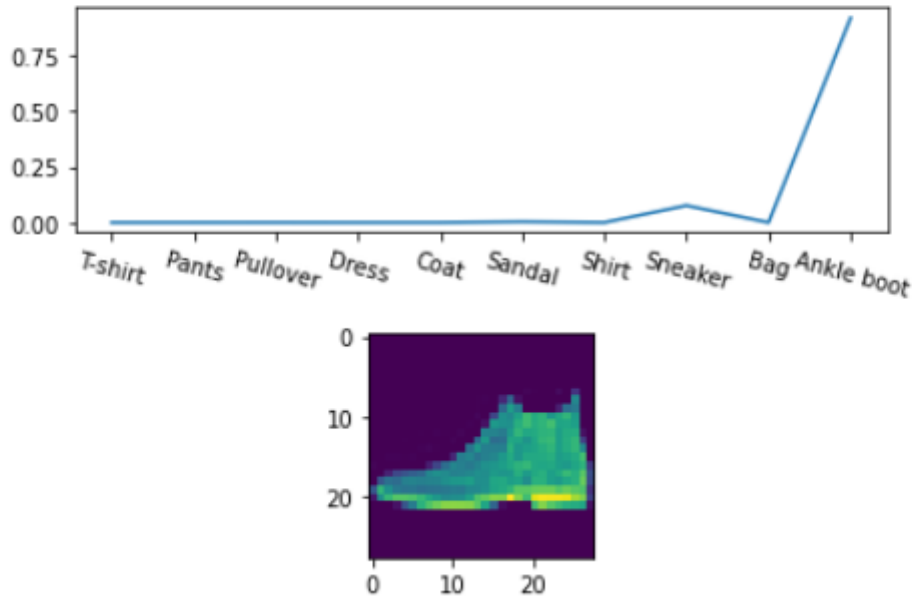
```
313/313 [==============================] - 2s 6ms/step - loss: 0.6035 - accuracy: 0.7787
Out[8]: [0.6034534573554993, 0.7786999940872192]
```

程式碼講解

In [9]:
```
cla
pri
pri
x=[
y=c
val
plt
plt
plt
max
for



for



plt
plt
plt
```

```
預測值: [1.2093998e-05 9.1252994e-05 1.8276297e-04 8.3967525e-06 9.3354560e-05
 3.4848407e-03 1.0527573e-04 7.6334335e-02 1.3659794e-04 9.1955113e-01]

預測是 Ankle boot 正確答案是 Ankle boot
```

Out[9]: <matplotlib.image.AxesImage at 0x1bbfc831ac0>



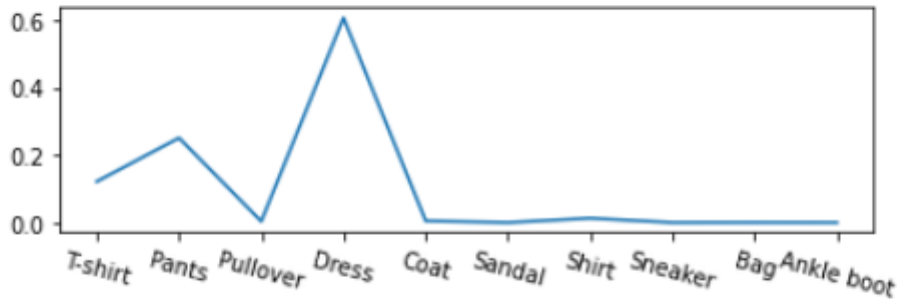
ion)模型

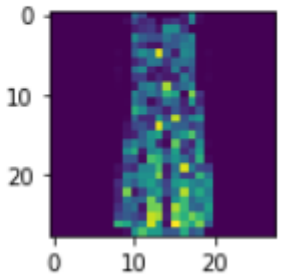
nkle boot']

程式碼講解

```
In [10]: cl
         te
         pr
         pr
         x=
         y=
         va                                                    Ankle boot']
         pl
         pl
         pl
         ma
         fo


         fo


         pl
         pl
         pl
```

```
預測值: [1.2153149e-01 2.5077564e-01 3.5328087e-03 6.0626566e-01 4.8769363e-03
 9.4167171e-06 1.2736082e-02 9.3998387e-06 2.5064842e-04 1.1907671e-05]

預測是 Dress 正確答案是 Dress

Out[10]: <matplotlib.image.AxesImage at 0x1bbfc810a00>
```





實驗 不同參數下的結果

- 改變神經元數量

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)
])
```

```
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
```

In [12]:
```
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 [==============================] - 19s 10ms/step - loss: 7.6190 - accuracy: 0.0996 - val_loss: 10.5844 - val_accurac
y: 0.0870
Epoch 2/5
1875/1875 [==============================] - 17s 9ms/step - loss: 7.3047 - accuracy: 0.1002 - val_loss: 7.7192 - val_accuracy:
0.0970
Epoch 3/5
1875/1875 [==============================] - 17s 9ms/step - loss: 7.2255 - accuracy: 0.1002 - val_loss: 9.1829 - val_accuracy:
0.1110
Epoch 4/5
1875/1875 [==============================] - 17s 9ms/step - loss: 7.1361 - accuracy: 0.1009 - val_loss: 6.1045 - val_accuracy:
0.1070
Epoch 5/5
1875/1875 [==============================] - 17s 9ms/step - loss: 7.2337 - accuracy: 0.1002 - val_loss: 7.1112 - val_accuracy:
0.0970
```

```
In [17]: print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 512)               276992

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 10)                5130

=================================================================
Total params: 282,122
Trainable params: 282,122
Non-trainable params: 0
_________________________________________________________________
None
```

```
In [14]: model.evaluate(test_images, test_labels)  #測試
```

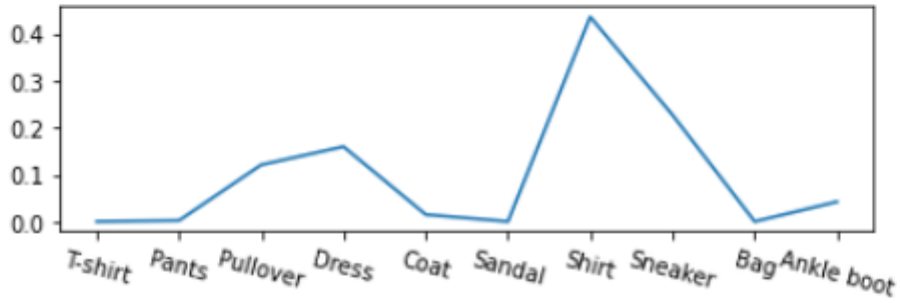
```
313/313 [==============================] - 2s 5ms/step - loss: 7.1248 - accuracy: 0.1000
```

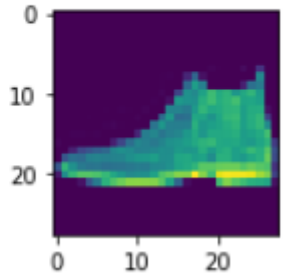
```
Out[14]: [7.12483024597168, 0.10000000149011612]
```

```
In [15]: cla
         pri
         pri
         x=[
         y=c
         val
         plt
         plt
         plt
         max
         for



         for



         plt
         plt
         plt
```

```
預測值: [3.5040360e-10 2.0731122e-03 1.2051660e-01 1.5909921e-01 1.4752876e-02
 5.6580193e-06 4.3502459e-01 2.2662988e-01 2.3756234e-08 4.1898012e-02]
```

```
Out[15]: <matplotlib.image.AxesImage at 0x1d96fa194f0>
```

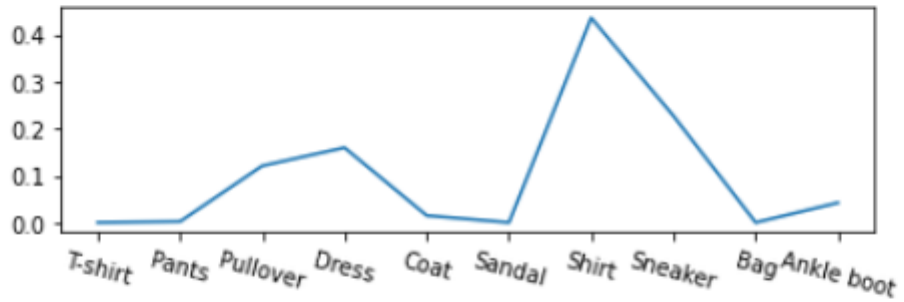


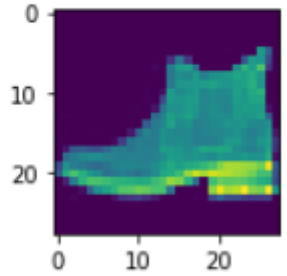



```
Ankle boot']
```

In [16]:
```
預測值: [3.5040360e-10 2.0731122e-03 1.2051660e-01 1.5909921e-01 1.4752876e-02
 5.6580193e-06 4.3502459e-01 2.2662988e-01 2.3756234e-08 4.1898012e-02]
```

Out[16]: <matplotlib.image.AxesImage at 0x1d8a4612130>






```
nkle boot']
```

謝謝大家