

113-1 雲端運算與邊緣運算應用

End computing and edge computing applications.

Lab2

LeNet Inference and application

授課老師： 王斯弘 老師

學 生： B11123206 陳冠欣

中 華 民 國 1 1 3 年 1 2 月 1 0 日

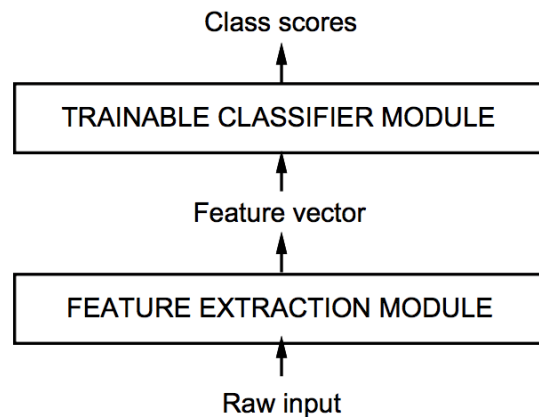
目錄

目錄	-----1
壹、 LeNet 簡介	-----2
貳、 PC 端 模型訓練 & 推理	-----3
參、 Edge 端 操作流程-DevBoard 燒機	-----7
肆、 遇到問題與解決	----- 14
伍、 心得	----- 15

壹、 LeNet 簡介

由於多層類神經網路的架構在高維度資料(手寫辨識)有不錯的效果，而且相較於傳統的辨識方法，不需要有太多的圖片預處理。

傳統的圖像辨識準確度與如何設計特徵提取有著密不可分的關係，而且常常必須針對不同問題而重新設計特徵提取(如下圖)。



傳統的辨識模式是分兩模組執行的；一個固定的特徵擷取器和一個可訓練的分類器

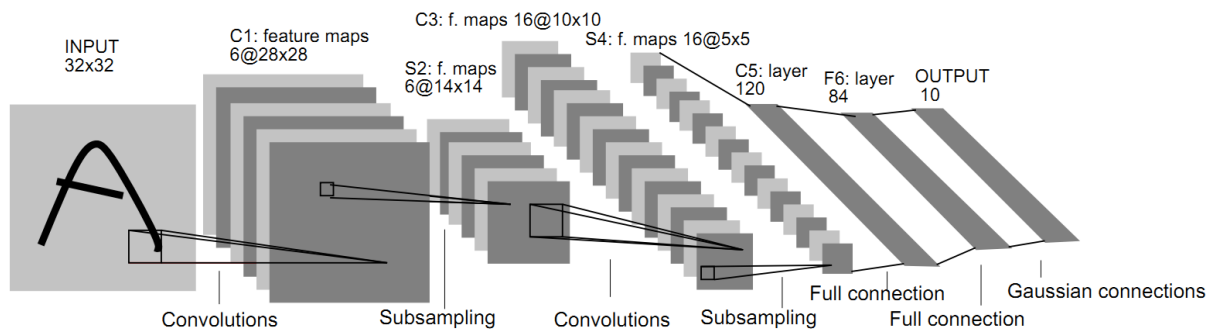


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

上圖為 LeNet-5 的架構，總共有七層。其中 C 代表 Convolution，S 代表 subsampling，也可以稱為 Pooling。簡單介紹一下 Convolution 及 Pooling: Convolution 就是在對圖片去做擷取特徵的動作，找出最好的特徵最後再進行分類。Pooling 就是選取特徵，保留重要資訊，並降低 Overfitting。

貳、PC 端 模型訓練&推理

1. 參數設定、dataset、資料處理

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# 超參數設定
num_class = 10          # 類別數量
batch_size = 256        # 每批次訓練資料的大小
epochs = 500             # 訓練迭代次數
iterations = 30          # 每個 epoch 中的步驟數

# 載入 MNIST 資料集
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# 資料標準化
x_train = x_train / 255.0
x_test = x_test / 255.0
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# 增加通道維度以適配 CNN
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)) # (60000, 28, 28, 1)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], x_test.shape[2], 1))      # (10000, 28, 28, 1)

# 將標籤轉為 one-hot 編碼
y_train = tf.keras.utils.to_categorical(y_train, num_class)
y_test = tf.keras.utils.to_categorical(y_test, num_class)

print("資料預處理完成")
```

2. 定義 convolution(卷積層)

```
# 定義卷積層
def conv(x, filters, kernel_size):
    """
    卷積層：應用指定的濾波器和卷積核尺寸。
    """
    return tf.keras.layers.Conv2D(filters=filters, kernel_size=kernel_size, activation='relu', padding='same')(x)
```

3. 定義 pooling 層(池化層)

```
# 定義池化層
def maxpooling(x):
    """
    最大池化層：降低輸出尺寸以減少計算成本。
    """
    return tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(x)
```

4. 定義 Model

```
# 定義 LeNet 模型架構
def lenet(x):
    """
    LeNet 模型：包含兩個卷積池化區塊和三個全連接層。
    """
    x = conv(x, 6, 5)          # 第一個卷積層 (6個濾波器, 5x5 卷積核)
    x = maxpooling(x)          # 第一個池化層
    x = conv(x, 16, (5, 5))    # 第二個卷積層 (16個濾波器, 5x5 卷積核)
    x = maxpooling(x)          # 第二個池化層
    x = tf.keras.layers.Flatten()(x) # 平坦化層
    x = tf.keras.layers.Dense(120, activation='relu')(x) # 全連接層 1
    x = tf.keras.layers.Dense(84, activation='relu')(x) # 全連接層 2
    x = tf.keras.layers.Dense(10, activation='softmax')(x) # 輸出層
    return x
```

5. 建立模型

```
# 建立模型
img_input = tf.keras.Input(shape=(28, 28, 1)) # 輸入層
output = lenet(img_input) # 構建 LeNet 網路
model = tf.keras.Model(inputs=img_input, outputs=output)
```

6. 編譯模型

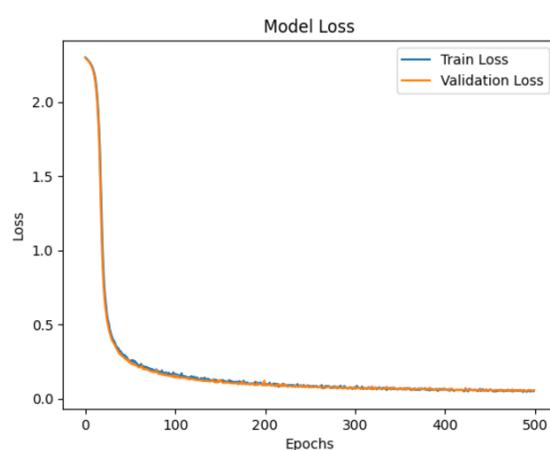
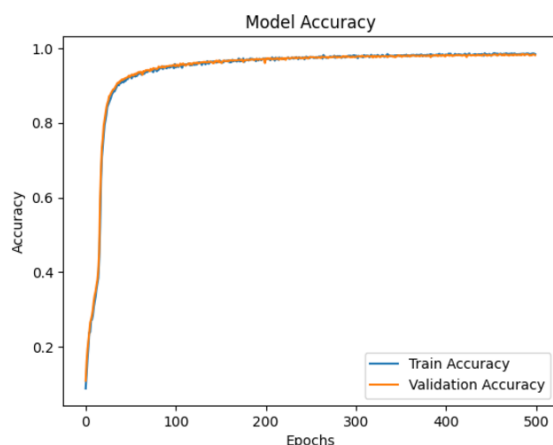
```
# 編譯模型
sgd = tf.keras.optimizers.SGD(learning_rate=0.005) # 優化器：隨機梯度下降 (SGD)
model.compile(optimizer=sgd,
               loss='categorical_crossentropy',
               metrics=['accuracy']) # 損失函數：交叉熵損失
```

7. 訓練模型 & 保存模型

```
# 訓練模型
history = model.fit(x=x_train, y=y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    steps_per_epoch=iterations,
                    validation_data=(x_test, y_test))

# 保存模型
model.save('mnist_lenet.h5')
print("模型已保存為 mnist_lenet.h5")
```

```
Epoch 498/500 0s 10ms/step - accuracy: 0.9858 - loss: 0.0502 - val_accuracy: 0.9827 - val_loss: 0.0533
Epoch 499/500 0s 11ms/step - accuracy: 0.9868 - loss: 0.0428 - val_accuracy: 0.9824 - val_loss: 0.0563
Epoch 500/500 0s 8ms/step - accuracy: 0.9845 - loss: 0.0451 - val_accuracy: 0.9809 - val_loss: 0.0569
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
模型已保存為 mnist_lenet.h5
```



8. 預測圖片

```
[ ] model.save('/content/mnist_lenet.h5')

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using in

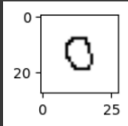
[ ] from PIL import Image

model = tf.keras.models.load_model('/content/mnist_lenet.h5')

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

import matplotlib.pyplot as plt
img = Image.open('/content/1/0.png')

plt.figure(figsize=(1, 1))
plt.imshow(img)
plt.show()
```



```
[ ] img = np.array(img)

img = np.dot(img[..., :3], [0.299, 0.587, 0.114])

img = 255-img
img = img/255.

img = np.reshape(img, (1, 28, 28, 1))

model.predict(img)

1/1 ----- 0s 201ms/step
array([[0.34101295, 0.00412966, 0.01219655, 0.0117046 , 0.00847752,
        0.16703771, 0.00313925, 0.17213851, 0.0015055 , 0.27865767]],
      dtype=float32)

print(np.argmax(model.predict(img)))

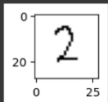
1/1 ----- 0s 18ms/step
0
```

9. 預測多張圖片

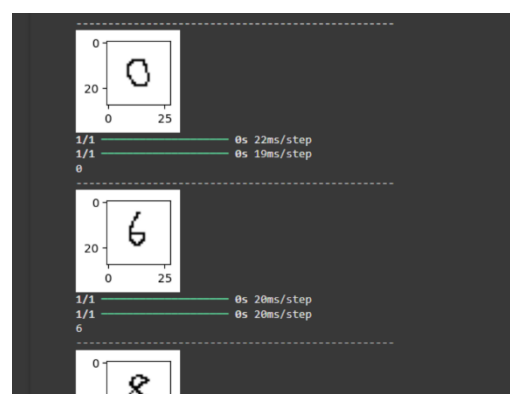
✓ 一次預測多張圖片

```
import glob
import os
file = glob.glob(os.path.join('/content/1', '*.png'))

for f in file:
    img = Image.open(f)
    plt.figure(figsize=(1, 1))
    plt.imshow(img)
    plt.show()
    img = np.array(img)
    img = np.dot(img[..., :3], [0.299, 0.587, 0.114])
    img = 255-img
    img = img/255.
    img = np.reshape(img, (1, 28, 28, 1))
    model.predict(img)
    print(np.argmax(model.predict(img)))
    print('-----')
```

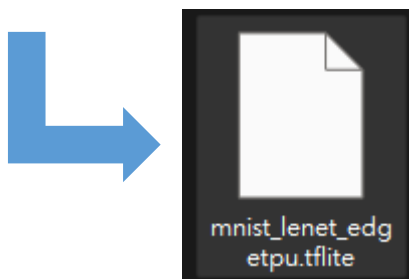


```
1/1 ----- 0s 19ms/step
1/1 ----- 0s 19ms/step
2
```



10. 將 H5 檔轉換成 tflite

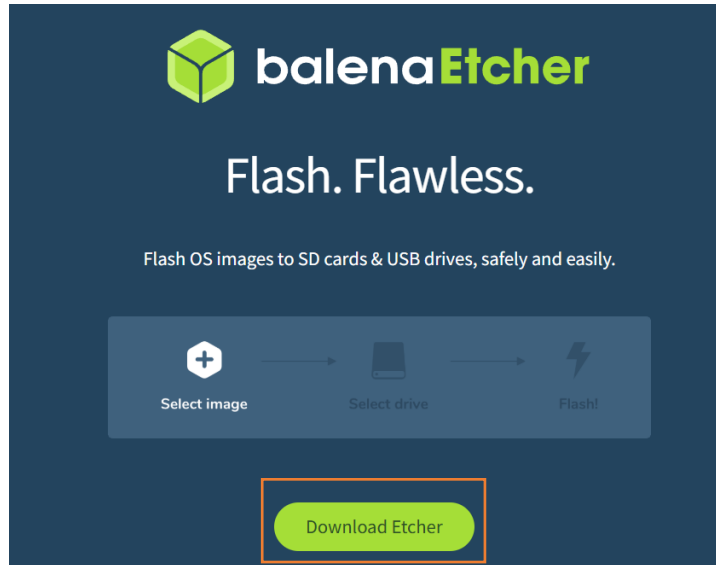
```
C: > Users > user > Downloads > Lab2-LeNet Inference > h5_to_tflite.py > ...
9
10 import tensorflow as tf
11 import numpy as np
12 import glob
13 tf.__version__
14
15 IMAGE_SIZE=28
16
17 def color_preprocessing(x_train,x_test):
18     x_train=x_train.astype('float32')
19     x_test=x_test.astype('float32')
20     x_train=x_train/255.
21     x_test=x_test/255.
22     return x_train,x_test
23
24 (x_train,y_train),(x_test,y_test)=tf.keras.datasets.mnist.load_data()
25 y_train=tf.keras.utils.to_categorical(y_train,10)
26 y_test=tf.keras.utils.to_categorical(y_test,10)
27 x_train,x_test=color_preprocessing(x_train,x_test)
28 x_train=np.reshape(x_train,(x_train.shape[0],x_train.shape[1],x_train.shape[2],1))
29
30 def representative_data_gen():
31     dataset_list_index=np.random.choice(range(50000),100)
32     for i in range(100):
33         image=x_train[dataset_list_index[i]]
34         image=tf.image.resize(image,[IMAGE_SIZE,IMAGE_SIZE])
35         image=tf.cast(image,tf.float32)
36         image=tf.expand_dims(image,0)
37         #with tf.Session() as sess:
38         #image=sess.run(image)
39         yield [image]
40
41 model=tf.keras.models.load_model('C:/model-resnet18.h5')
42 converter = tf.lite.TFLiteConverter.from_keras_model(model)
43
44
45 converter=tf.lite.TFLiteConverter.from_keras_model_file('C:/model-resnet18.h5')
46 converter.optimizations=[tf.lite.Optimize.DEFAULT]
47 converter.target_spec.supported_ops=[tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
48 converter.representative_dataset=representative_data_gen
49 converter.inference_input_type=tf.uint8
50 converter.inference_output_type=tf.uint8
51 tflite_model=converter.convert()
52
53 with open('models/mnist_lenet.tflite','wb') as f:
54     f.write(tflite_model)
```



參、 Edge 端 操作流程-

DevBoard 燒機：

1. 下載 balenaEtcher



下載 Etcher for Windows (x86|x64) (Installer)

DOWNLOAD

Download Etcher

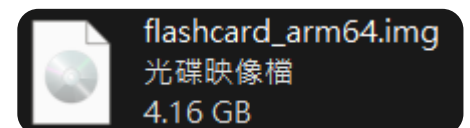
ASSET	OS	ARCH	
ETCHER FOR WINDOWS (X86 X64) (INSTALLER)	WINDOWS	X86 X64	Download
ETCHER FOR MACOS	MACOS	X64	Download
ETCHER FOR MACOS (ARM64)	MACOS	ARM64	Download
ETCHER FOR LINUX X64 (64-BIT) (ZIP)	LINUX	X64	Download
ETCHER FOR LINUX (LEGACY 32 BIT) (APPIMAGE)	LINUX	X86	Download

1. 下載並解壓縮SD卡鏡像：[enterprise-eagle-flashcard-20211117215217.zip](#)

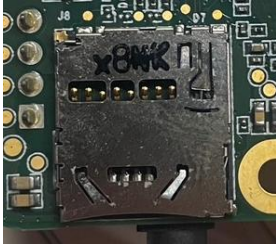
ZIP 包含一個名為 `flashcard_arm64.img`。

2. 使用**balenaEtcher**等程式將 `flashcard_arm64.img` 檔案閃存到您的 microSD 卡上。

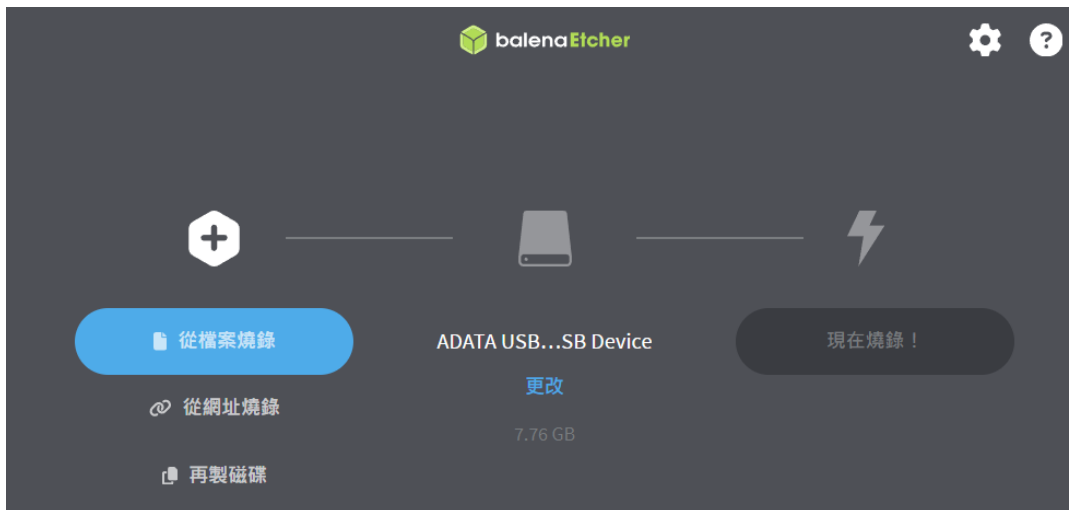
這需要 5-10 分鐘，具體取決於您的 microSD 卡和適配器的速度。



2. 調整啟動模式、插入 SD 卡



3. 燒入映像檔到 SD 卡



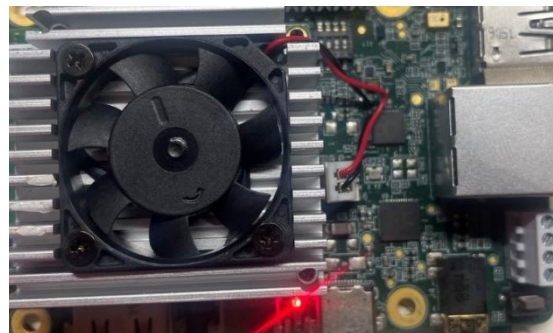
4. 調整開關改為 SD 卡啟動



啟動模式	開關1	開關2	開關3	開關4
SD卡	在	離開	在	在

5. 進行刷機

插上電源前先插入 SD 卡
並根據前一步驟調整開機模式
插入 SD 卡並通電後
等待電路板關閉並紅色 LED 燈熄滅
即完成燒機



6. 斷電並整開機模式重新開機

- 當紅色 LED 熄滅時，拔下電源並取出 microSD 卡。
- 將啟動模式開關變更為 eMMC 模式，如圖4所示：

啟動模式	開關1	開關2	開關3	開關4
多媒體卡	在	離開	離開	離開

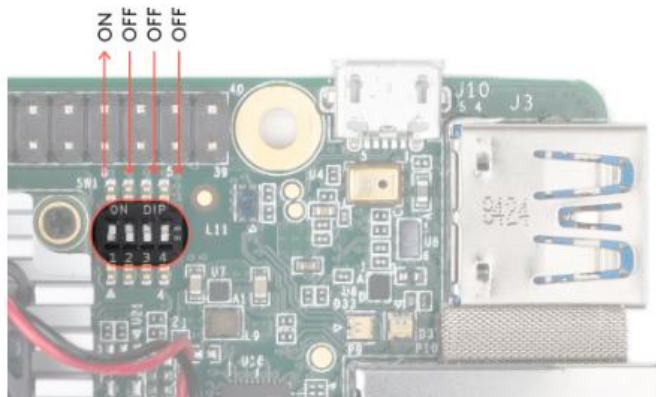


圖 4.啟動開關設定為 eMMC 模式

- 將開發板連接到電源，它現在應該啟動 Mendel Linux。
- 刷機後首次啟動大約需要 3 分鐘（後續啟動時間快得多）。

7. 按照步驟使用 Putty 連接 Windows

與 Windows 連線

您可以從 Windows 10 連接到開發板的序列控制台，如下所示：

1. 使用 micro-B USB 線將電腦連接到開發板，然後將開發板連接到電源，如圖 1 所示。
2. 在 Windows 電腦上，開啟裝置管理員並找到開發板的 COM 連接埠。

連接 USB 連接線後一分鐘內，Windows 應自動安裝必要的驅動程式。因此，如果您展開連接埠（COM 和 LPT），您應該會看到兩個名為「Silicon Labs Dual CP2105 USB to UART Bridge」的裝置。

記下名為「增強型 COM 連接埠」的裝置的 COM 連接埠（例如「COM3」）。您將在下一步中使用它。

如果 Windows 無法識別該設備，則應將其列在「其他設備」下。右鍵單擊增強型 Com 連接埠並選擇更新驅動程式以查找適當的裝置驅動程式。

3. 開啟 PuTTY 或其他串行控制台應用程式，並使用波特率與上述 COM 連接埠啟動串行控制台連接 115200。例如，如果使用 PuTTY：
 1. 在左側窗格中選擇會話。
 2. 對於連接類型，選擇串行。
 3. 在「序列線路」中輸入 COM 連接埠（「COM3」），在「速度」中輸入「115200」。
 4. 然後點選「開啟」。
4. 當螢幕終端打開時，它可能是空白的。按 Enter 鍵，系統完成啟動後您應該會看到登入提示。
預設使用者名稱和密碼均為「mendel」。

插好連接到電腦的傳輸線和電源線

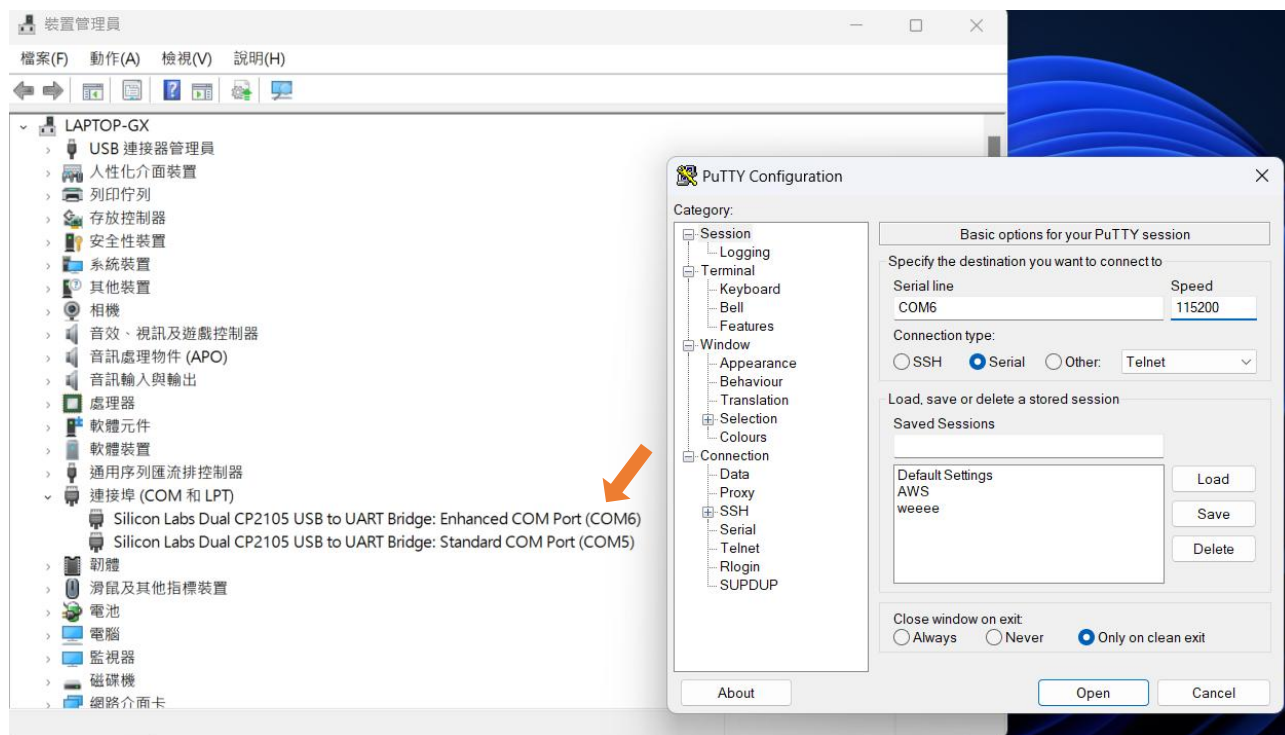


先到裝置管理員尋找

Silicon Labs Dual CP2105 USB to UART Bridge: Enhanced COM Port

PuTTY Configuration:

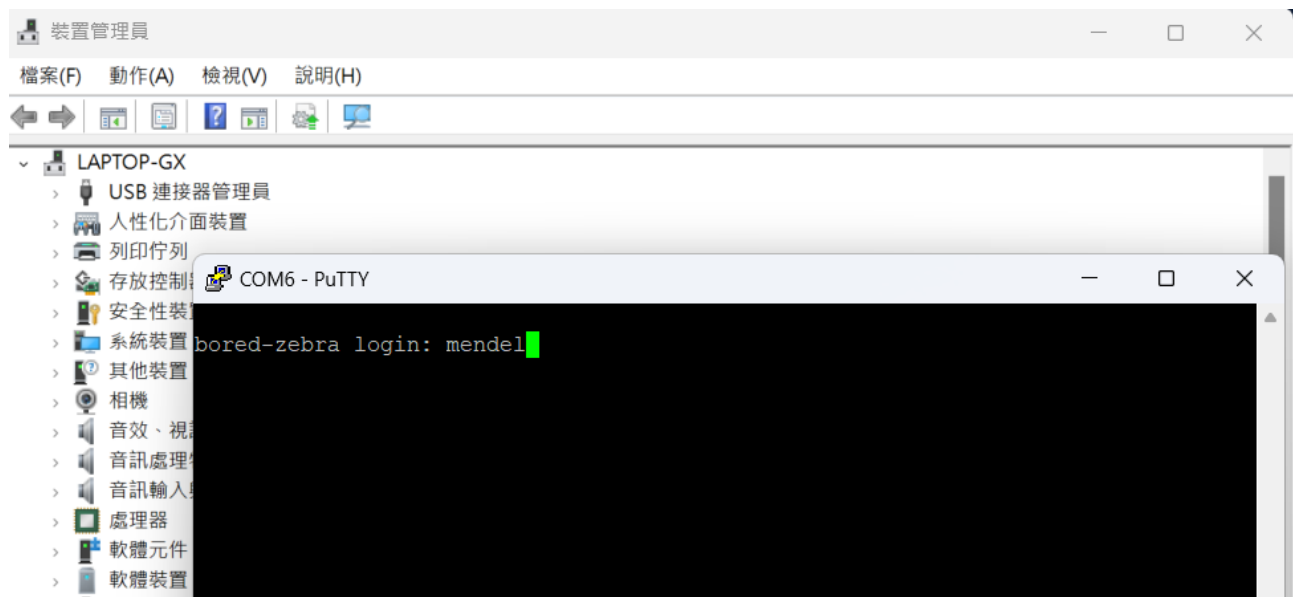
調整連接類型為 Serial、連接埠為 COM6、speed 為 115200



8. 進入終端機

如果 screen 後什麼都沒有記得按 Enter

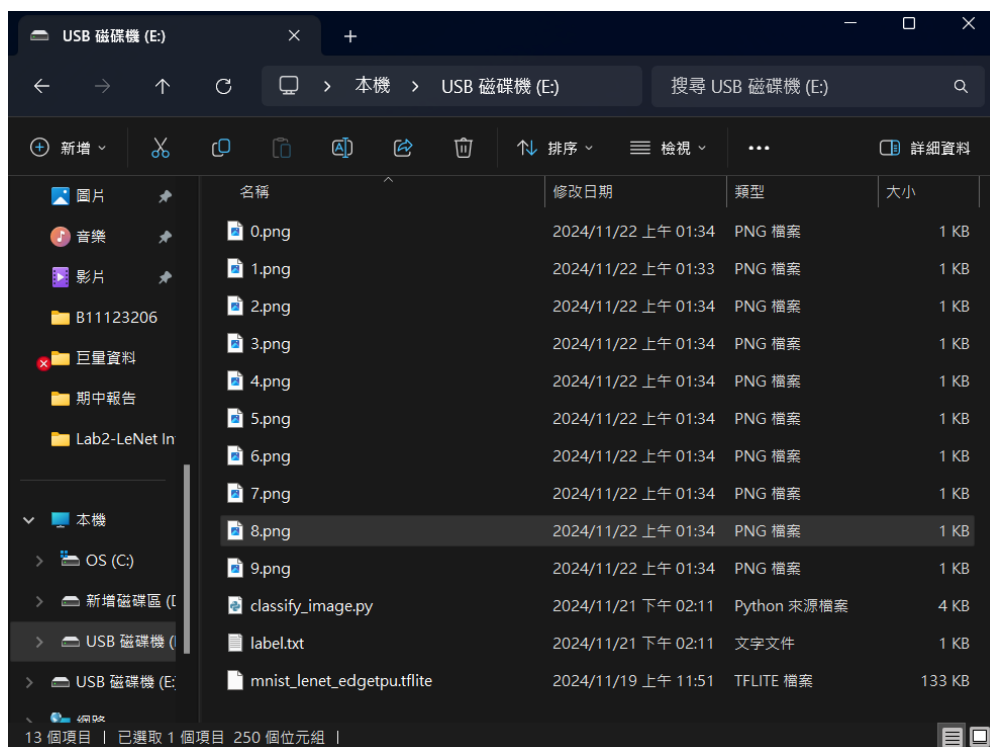
Enter 後，第一次 screen 需要登入，輸入帳號密碼都是 mendel



9. 將所需的檔案放入 USB

需要的檔案有

1. tflite
2. Label
3. Py 檔
4. 要預測的圖片



10. 插上 USB 到 DevBoard

```
mendel@bored-zebra:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda          8:0      1   7.5G  0 disk
└─sda1       8:1      1   7.5G  0 part
mmcblk0     179:0    0    7.3G  0 disk
├─mmcblk0p1 179:1    0   128M  0 part /boot
├─mmcblk0p2 179:2    0     1M  0 part
├─mmcblk0p3 179:3    0     2G  0 part /home
├─mmcblk0p4 179:4    0    5.1G  0 part /
├─mmcblk0boot0 179:32   0     4M  1 disk
├─mmcblk0boot1 179:64   0     4M  1 disk
└─mmcblk0rpb 179:96   0     4M  0 disk
```

USB 在這裡

11. 回到終端機尋找有 dev 資料夾的地方，輸入 `sudo mkdir test` 創建一個掛載 USB 的資料夾 test

```
mendel@coy-tang:/$ sudo mkdir test
mendel@coy-tang:/$ ls
bin    home      lib64      opt        sbin      tmp        vmlinuz.old
boot   initrd.img lost+found  proc       srv        usr
dev    initrd.img.old media       root       sys        var
etc    lib        mnt        run        test       vmlinuz
```

12. 進入 dev 尋找 sda1

```
COM6 - PuTTY
mendel@bored-zebra:~$ mendel
-bash: mendel: command not found
mendel@bored-zebra:~$ ls
mendel@bored-zebra:~$ mkdir test
mendel@bored-zebra:~$ ls
test
mendel@bored-zebra:~$ ls /dev
apex_0      loop2      ptyp6      tty2       tty51      ttye
autofs      loop3      ptyp7      tty20      tty52      ttypf
block       loop4      ptyp8      tty21      tty53      ttyS0
bsg         loop5      ptyp9      tty22      tty54      ttyS1
btrfs-control loop6      ptypa      tty23      tty55      ttyS2
bus         loop7      ptypb      tty24      tty56      ttyS3
cec0        loop-control ptypc      tty25      tty57      ubi_ctrl
char        mapper     ptypd      tty26      tty58      urandom
console     mem        ptype      tty27      tty59      v4l
cpu_dma_latency memory_bandwidth ptypf      tty28      tty6       vcs
cuse        mmcblk0    random     tty29      tty60      vcs1
disk        mmcblk0boot0 rtc        tty3       tty61      vcs2
dri         mmcblk0boot1 rtc0       tty30      tty62      vcs3
fd          mmcblk0p1 sda        tty31      tty63      vcs4
full        mmcblk0p2 sda1       tty32      tty7       vcs5
fuse        mmcblk0p3 shm        tty33      tty8       vcs6
galcore     mmcblk0p4 snd        tty34      tty9       vcs7
gpiochip0   mmcblk0rpmb spidev0.0  tty35      ttyGS0     vcsa
gpiochip1   mqeuue     spidev0.1  tty36      ttymx0     vcsa1
gpiochip2   mxc_hantro stderr     tty37      ttymx1     vcsa2
gpiochip3   net        stdin      tty38      ttymx2     vcsa3
gpiochip4   network_latency stdout     tty39      ttyv0      vcsa4
```

13. 確認完名稱就可以退出 dev 將 USB 掛載到 test

指令為 `sudo mount ./dev/sda1 ./test`

```
mendel@coy-tang:/dev$ cd ..
mendel@coy-tang:/$ sudo mount ./dev/sda1 ./test
mendel@coy-tang:/$ ls
bin    home      lib64      opt        sbin      tmp        vmlinuz.old
boot   initrd.img lost+found  proc       srv        usr
dev    initrd.img.old media       root       sys        var
etc    lib        mnt        run        test       vmlinuz
mendel@coy-tang:/$ cd test
mendel@coy-tang:/test$ ls
0.png  3.png  6.png  9.png  mnist_lenet_edgetpu.tflite
1.png  4.png  7.png  classify_image.py 'System Volume Information'
2.png  5.png  8.png  label.txt
mendel@coy-tang:/test$
```

就可以看到 test 裡有 USB 的檔案了

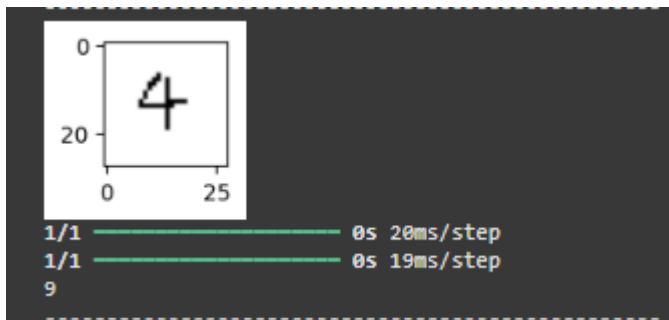
肆、 Edge 端 驗證結果

輸入指令執行

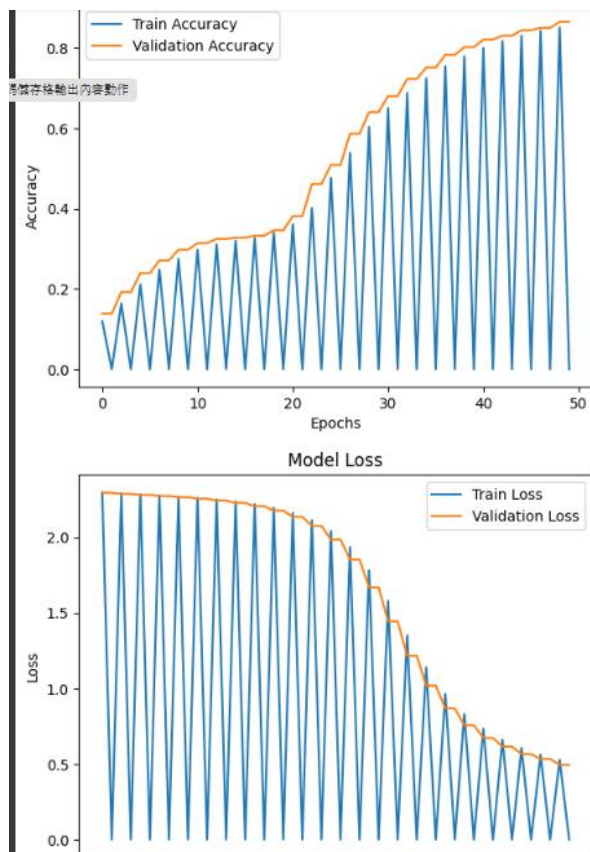
```
python3 classify_image.py -m mnist_lenet_edgetpu.tflite -l label.txt -i 6.png
```

```
mendel@bored-zebra:~$ python3 ~/test/classify_image.py -m ~/test/mnist_lenet_edgetpu.tflite -i ~/test/1.png
----INFERENCE TIME----
Note: The first inference on Edge TPU is slow because it includes loading the model into Edge TPU memory.
1.7ms
0.4ms
0.5ms
0.3ms
0.3ms
-----RESULTS-----
1: 0.75391
mendel@bored-zebra:~$ python3 ~/test/classify_image.py -m ~/test/mnist_lenet_edgetpu.tflite -i ~/test/2.png
----INFERENCE TIME----
Note: The first inference on Edge TPU is slow because it includes loading the model into Edge TPU memory.
2.0ms
1.3ms
1.3ms
1.3ms
0.3ms
-----RESULTS-----
2: 0.76172
mendel@bored-zebra:~$ python3 ~/test/classify_image.py -m ~/test/mnist_lenet_edgetpu.tflite -i ~/test/3.png
----INFERENCE TIME----
Note: The first inference on Edge TPU is slow because it includes loading the model into Edge TPU memory.
1.5ms
0.4ms
0.9ms
0.9ms
0.9ms
-----RESULTS-----
3: 0.43359
mendel@bored-zebra:~$ python3 ~/test/classify_image.py -m ~/test/mnist_lenet_edgetpu.tflite -i ~/test/4.png
----INFERENCE TIME----
Note: The first inference on Edge TPU is slow because it includes loading the model into Edge TPU memory.
1.8ms
0.4ms
0.3ms
0.3ms
0.3ms
-----RESULTS-----
4: 0.46484
```


伍、 遇到問題與解決



模型訓練時，有時候會遇到模型便是錯誤的問題、
不過在調整參數並重新訓練之後酒可以正常辨識了。



Acc 和 loss 忽高忽低的問題:推測是 batch_size
過大造成的，在調整 batch_size 之後問題就解決了

陸、心得

透過這次實驗，我深入了解了 LeNet 模型的架構及其在圖像辨識上的應用。從模型的訓練到在 Edge 端進行推論，整個過程不僅加深了我對深度學習的理論理解，也讓我體會到實際操作中的挑戰與收穫。

在模型訓練的過程中，我遇到了參數設定不當導致模型表現不穩定的情況。經過多次嘗試與調整，如修改 `batch_size` 和學習率等參數，最終解決了模型準確率波動過大的問題。這也讓我明白，實驗的關鍵不僅在於程式的執行，更在於對問題的深入剖析與解決方法的探索。

在 Edge 端實現推論時，將模型從 PC 端轉移並執行於開發板的整個過程讓我收穫頗多。從燒機到連接開發板，雖然過程中需要克服硬體設定的挑戰，但成功將模型部署並進行推論的成就感無法言喻。

此次實驗使我體認到理論與實踐相結合的重要性，也提升了我的問題解決能力與動手能力。我將持續探索人工智慧與邊緣運算的應用，期待能將這些技術運用於更具創新性的實際案例中。