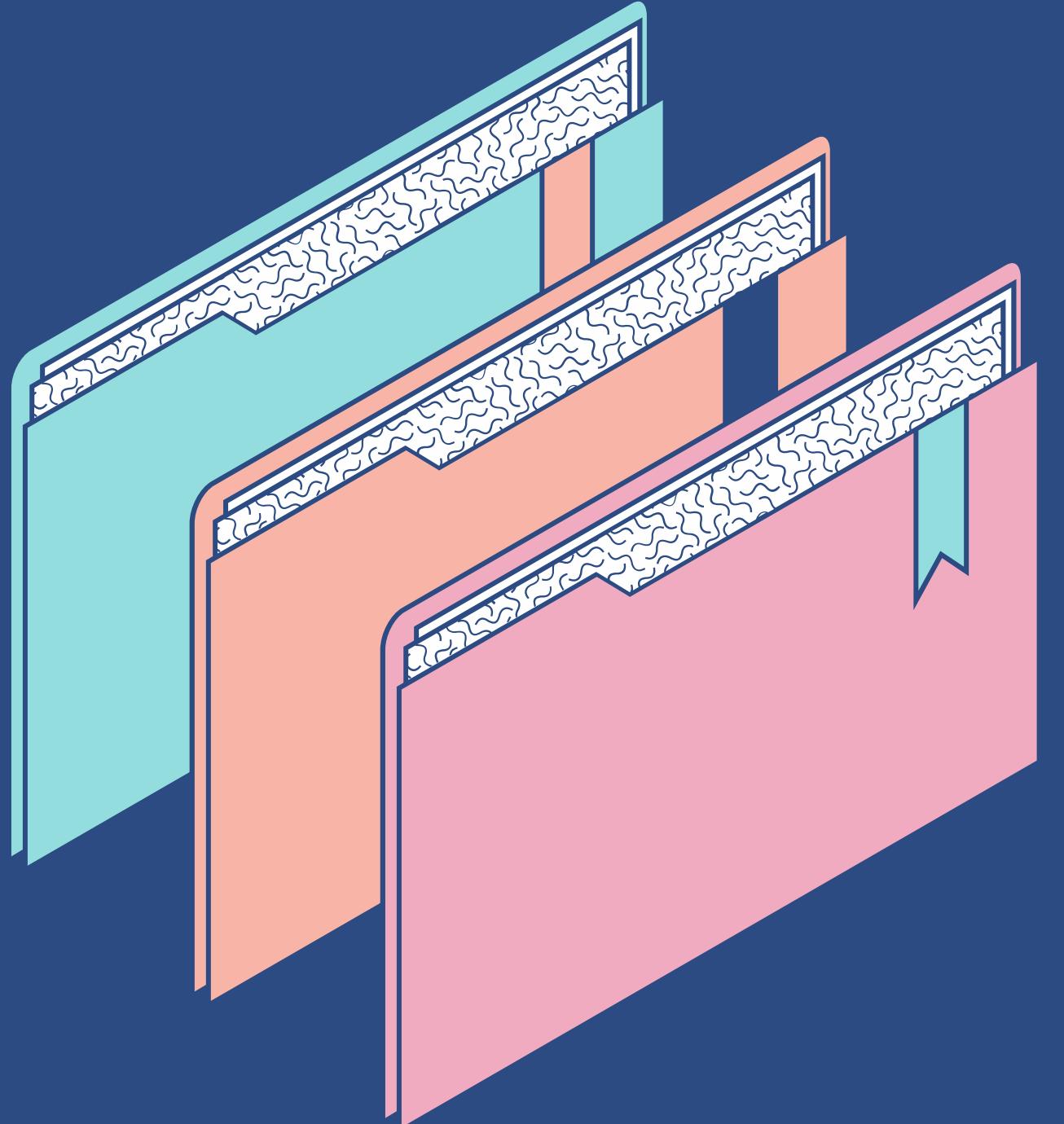


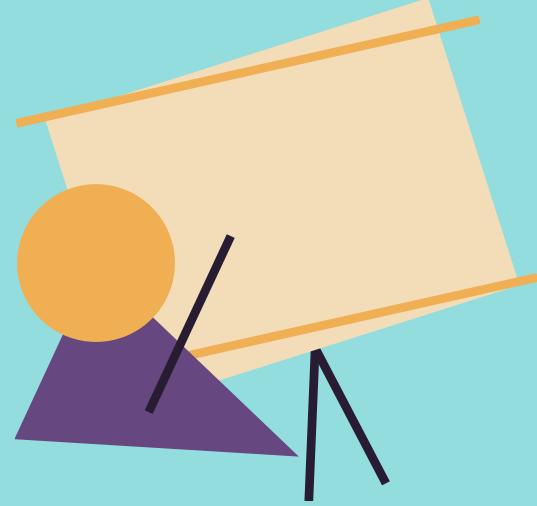
Lab3-ResNet

第四組-四資管AI三A-B11123224-余雨芹

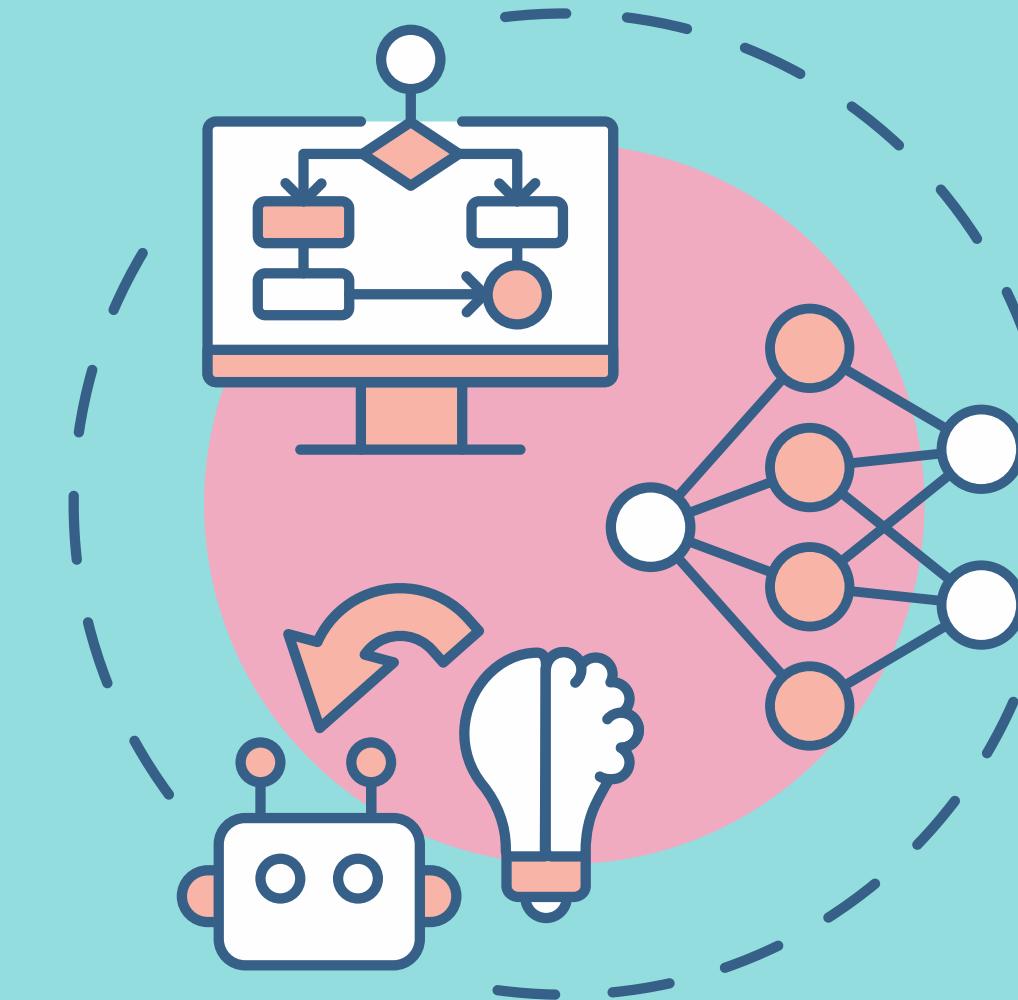
Table of content



01. 模型介紹
02. PC端 模型訓練&推理
03. Edge端 操作流程
04. Edge端 驗證結果
05. 遇到問題與解決
06. 心得



01. 模型介紹





模型介紹：ResNet-18

ResNet-18 是深度殘差網路（Residual Network, ResNet）系列中的輕量級變體，主要設計目的是解決深層神經網路中常見的梯度消失問題，並提升網路的訓練效率與準確性。該模型在影像分類和辨識任務中表現出色，特別適合於需要高效推理的應用場景。

ResNet 的核心理念

1. 殘差學習 (Residual Learning) :

- 深層網路在學習時容易出現退化現象，新增層數不一定能提升準確率。ResNet 通過殘差塊 (Residual Block)，改變網路學習方式，學習殘差 $F(x) = H(x) - x$, $F(x) = H(x) - x$, 而非直接學習目標函數 $H(x)$ 。

2. 跳躍連結 (Skip Connections) :

- 跳躍連結允許輸入訊號直接傳遞到更深層次，保持梯度的流動，解決深層網路中的梯度消失問題。

RESNET-18 的架構設計

ResNet-18 包含 18 層，主要由卷積層和殘差塊組成，具體結構如下：

1. 初始卷積與最大池化層：

- 第一層為一個 $7 \times 77 \times 7$ 的卷積核，步長為 2，接著是一個 $3 \times 33 \times 3$ 的最大池化層。

2. 殘差模組 (Residual Blocks)：

- 包括 4 個殘差模組，每個模組包含 2 個卷積層。輸出通道數隨模組深度逐步增加：[64, 128, 256, 512]。

3. 全域平均池化層 (Global Average Pooling)：

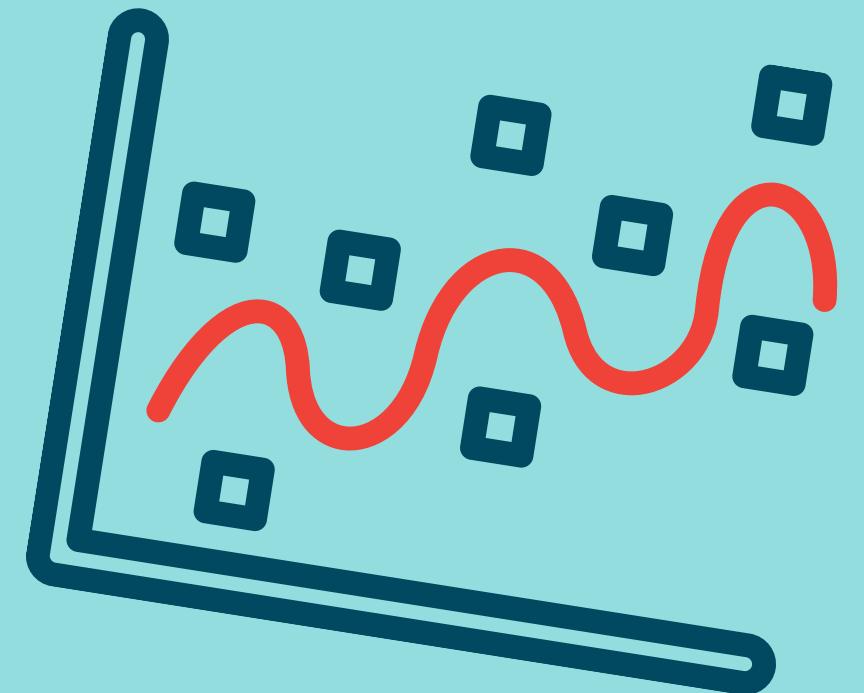
- 將特徵圖的空間維度壓縮為一維特徵向量。

4. 全連接層 (Fully Connected Layer)：

- 將最終特徵向量映射為輸出類別（例如 ImageNet 中為 1000 類別）。



02. PC端 模型訓練&推理



```

Args:
    x: 輸入的張量
Returns:
    x: 經過 ResNet18 處理後的輸出張量
"""
# 初始卷積層與最大池化層
x = Conv2D(filters=64, kernel_size=(7, 7), strides=(2, 2), padding='same')(x)
x = MaxPooling2D()(x)

# 殘差塊的堆疊
x = block(x, out_filters=64, downsample=False)
x = block(x, out_filters=64, downsample=False)
x = block(x, out_filters=128, downsample=True)
x = block(x, out_filters=128, downsample=False)
x = block(x, out_filters=256, downsample=True)
x = block(x, out_filters=256, downsample=False)
x = block(x, out_filters=512, downsample=True)
x = block(x, out_filters=512, downsample=False)

# 全局平均池化與全連接分類層
x = GlobalAveragePooling2D()(x)
x = Dense(5, activation='softmax')(x) # 假設分類為 5 類
return x

```

```

img_input = Input(shape=IMAGE_SIZE+(3,))
output = resnet18(img_input)

model = Model(img_input, output)
print(model.summary())

```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 224, 224, 3)	0	-
conv2d (Conv2D)	(None, 112, 112, 64)	9,472	input_layer[0][0]

- 1. 資料預處理：透過增強技術生成更多樣化的訓練資料。**
- 2. 資料載入：以批次方式將資料送入模型進行訓練與驗證。**
- 3. 類別標籤管理：將資料集中的類別名稱與索引建立對應關係，便於後續使用。**

```

DATASET_PATH = '/content/drive/MyDrive/Colab/lab3/sample'

IMAGE_SIZE = (224, 224)

NUM_CLASSES = 5

BATCH_SIZE = 256

# Epoch 數
NUM_EPOCHS = 100

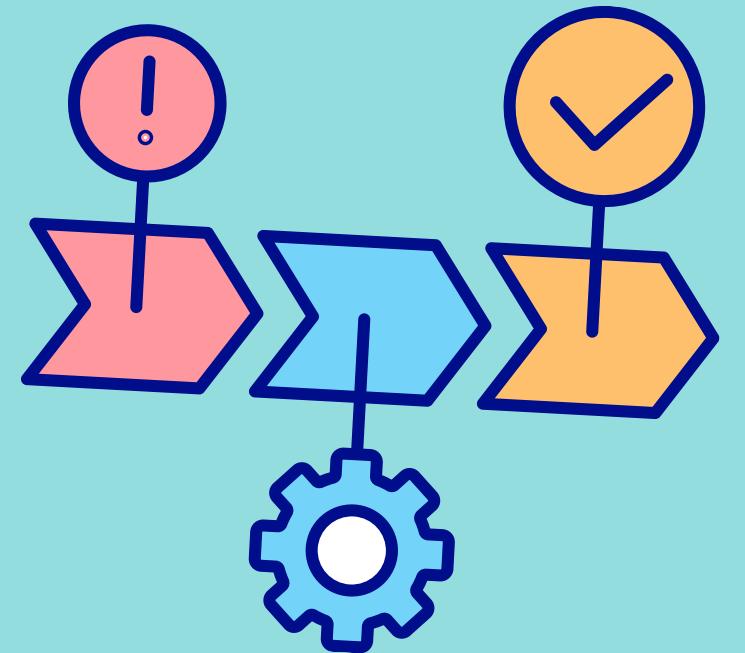
# 模型輸出儲存的檔案
WEIGHTS_FINAL = 'model-resnet18.h5'

# 透過 data augmentation 產生訓練與驗證用的影像資料
train_datagen = ImageDataGenerator(rotation_range=40,
                                    width_shift_range=0.2,
                                    height_shift_range=0.2,
                                    shear_range=0.2,
                                    zoom_range=0.2,
                                    channel_shift_range=10,
                                    horizontal_flip=True,
                                    fill_mode='nearest')
train_batches = train_datagen.flow_from_directory(DATASET_PATH + '/train',
                                                 target_size=IMAGE_SIZE,
                                                 interpolation='bicubic',
                                                 class_mode='categorical',
                                                 shuffle=True,
                                                 batch_size=BATCH_SIZE)

valid_datagen = ImageDataGenerator()
valid_batches = valid_datagen.flow_from_directory(DATASET_PATH + '/valid',
                                                 target_size=IMAGE_SIZE,
                                                 interpolation='bicubic',
                                                 class_mode='categorical',
                                                 shuffle=False,
                                                 batch_size=BATCH_SIZE)

```

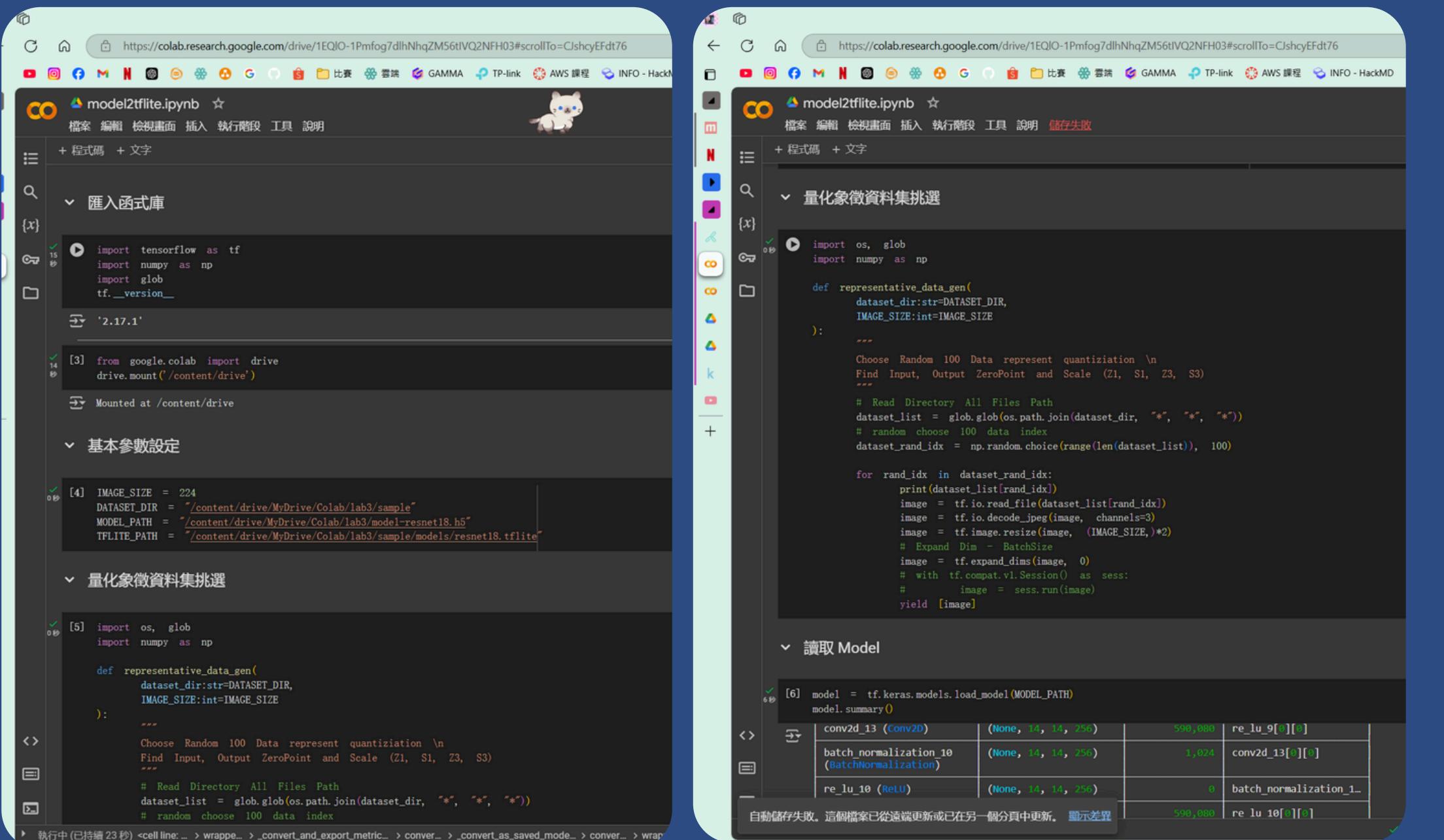
- 1. 定義 ResNet-18 模型的基本架構。**
- 2. 處理輸入影像，提取多層次特徵。**
- 3. 輸出分類結果，適用於多類別分類問題。**
- 4. 整合模型結構：基於 ResNet-18 架構生成完整的模型。**
- 5. 可視化模型：通過 `summary()` 打印模型的層次結構和參數數量，用於檢查模型的設計是否正確。**



03. Edge端 操作流程



1. 量化



```
import tensorflow as tf
import numpy as np
import glob
tf.__version__
'2.17.1'

from google.colab import drive
drive.mount('/content/drive')

IMAGE_SIZE = 224
DATASET_DIR = "/content/drive/MyDrive/Colab/lab3/sample"
MODEL_PATH = "/content/drive/MyDrive/Colab/lab3/model-resnet18.h5"
TFLITE_PATH = "/content/drive/MyDrive/Colab/lab3/sample/models/resnet18.tflite"

import os, glob
import numpy as np

def representative_data_gen(
    dataset_dir:str=DATASET_DIR,
    IMAGE_SIZE:int=IMAGE_SIZE
):
    """
    Choose Random 100 Data represent quantization
    Find Input, Output ZeroPoint and Scale (Z1, S1, Z3, S3)
    """
    # Read Directory All Files Path
    dataset_list = glob.glob(os.path.join(dataset_dir, "*","*","*"))
    # random choose 100 data index
    dataset_rand_idx = np.random.choice(range(len(dataset_list)), 100)

    for rand_idx in dataset_rand_idx:
        print(dataset_list[rand_idx])
        image = tf.io.read_file(dataset_list[rand_idx])
        image = tf.io.decode_jpeg(image, channels=3)
        image = tf.image.resize(image, (IMAGE_SIZE,) * 2)
        # Expand Dim - BatchSize
        image = tf.expand_dims(image, 0)
        # with tf.compat.v1.Session() as sess:
        #     image = sess.run(image)
        yield [image]

model = tf.keras.models.load_model(MODEL_PATH)
model.summary()
```

1. 載入模型：

- 使用 `tf.keras.models.load_model()` 載入一個已經訓練好的 Keras 模型，並使用 `model.summary()` 檢視模型結構和參數數量。

2. 量化模型參數：

- 使用 TensorFlow Lite 轉換器 (`tf.lite.TFLiteConverter`) 將模型量化

3. 量化的過程包括：

- 設定優化選項 (`tf.lite.Optimize.DEFAULT`)。
- 指定目標支援的操作 (INT8 格式)。
- 使用一個代表性資料集來執行量化。

4. 轉換為 TFLite 格式：

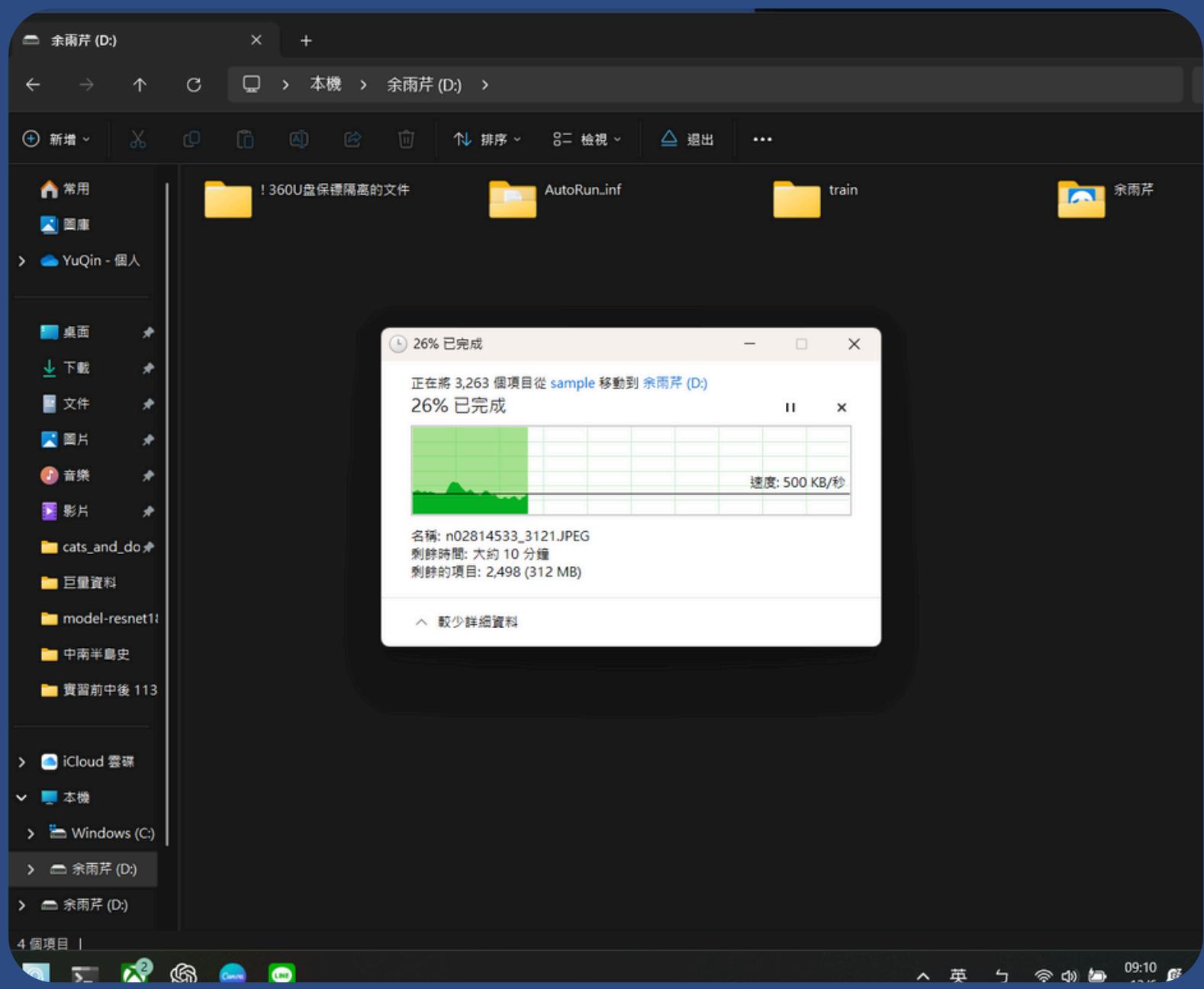
- 呼叫 `converter.convert()` 將模型轉換為 TFLite 格式。

5. 儲存 TFLite 檔案：

- 將轉換後的 TFLite 模型寫入指定路徑。

2.Dev board 執行

1. 將需要的檔案放入USB



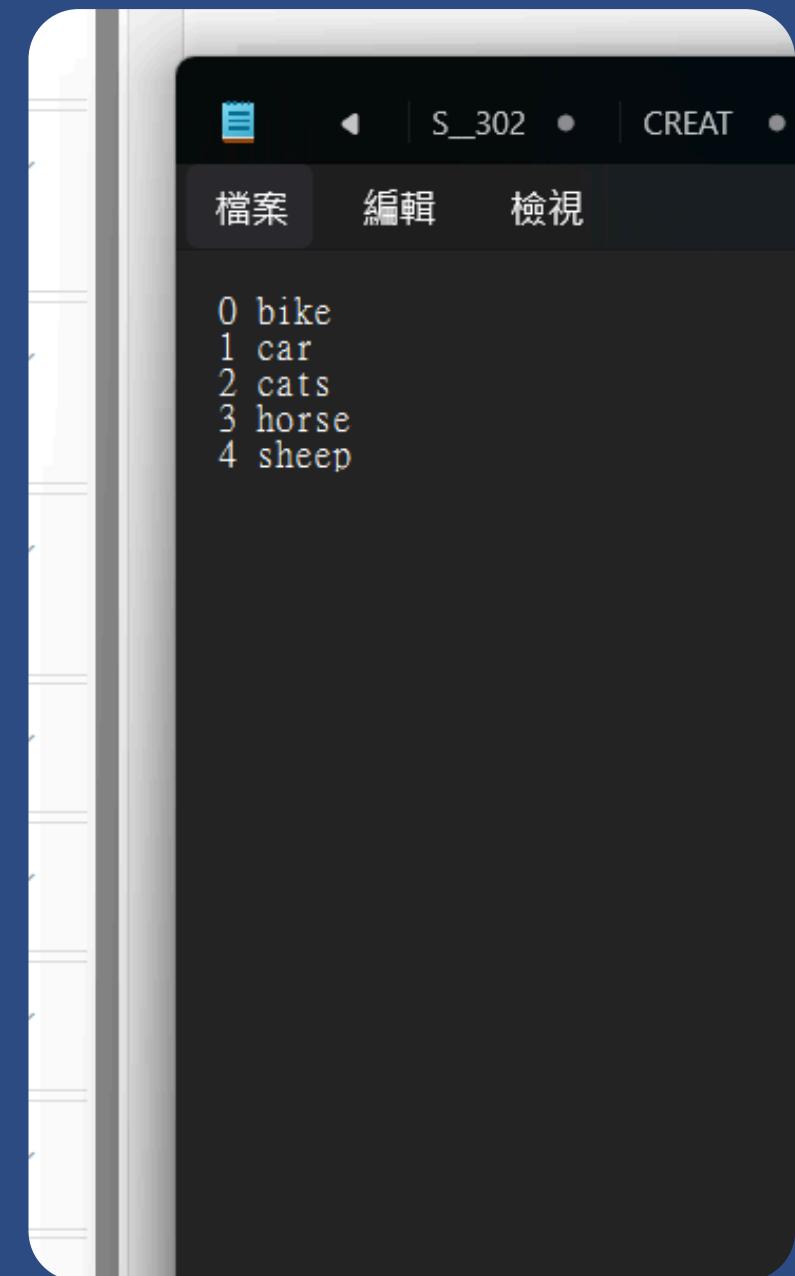
2. 將Dev board接上螢幕、滑鼠和鍵盤



2.Dev board 執行

- 掛載USB
- 輸入指令
- 在Label.txt增加自己的類別標籤

```
● ● ●  
1 sudo mkdir Lab3 #建立資料夾  
2 sudo mount ./dev/sda1 ./Lab3 #掛載  
3 cd Lab3 #進入資料夾  
4 ls #查看資料夾內容
```



2.Dev board 執行

- 連接網路
- 輸入指令
- 更新pip
- 安裝opencv

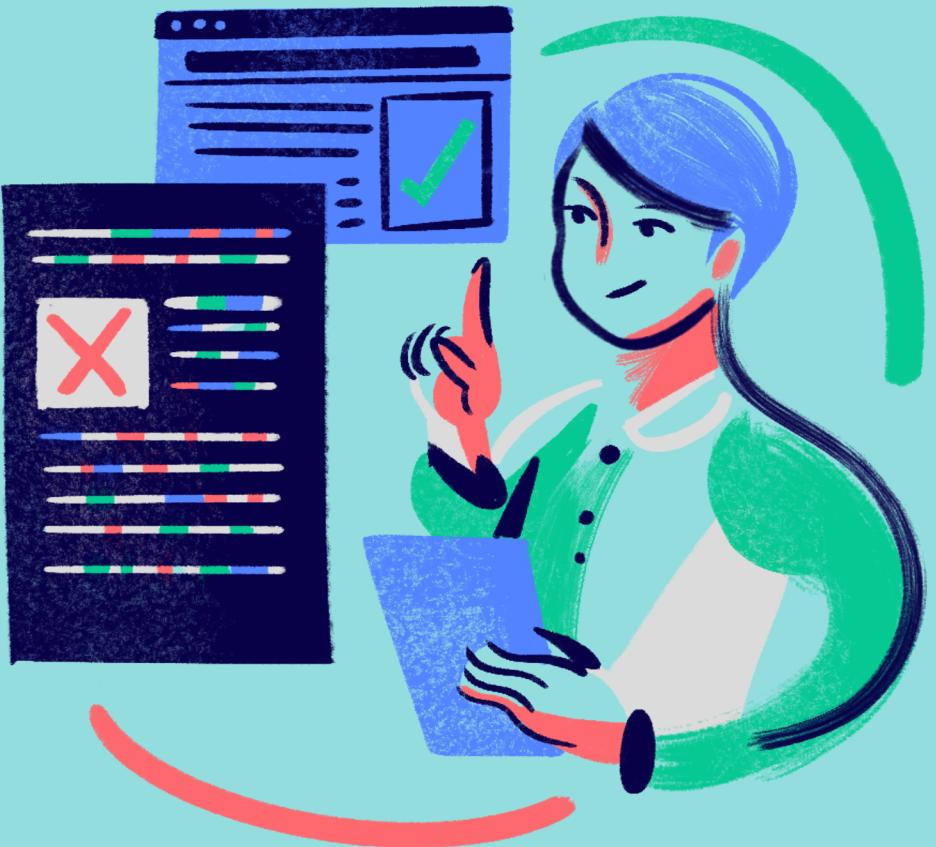
```
● ● ●  
1 &> nmtui #進入網路設定  
2 &> pip3 insrall --upgrade pip #更新pip  
3 &> pip3 install --user opencv-contrib-python #安裝opencv  
4
```

- 連接camera
- 輸入指令

```
● ● ●  
1 &>python3 resnet.py --model resnet18.tflite --labels labels.txt #執行resnet.py
```

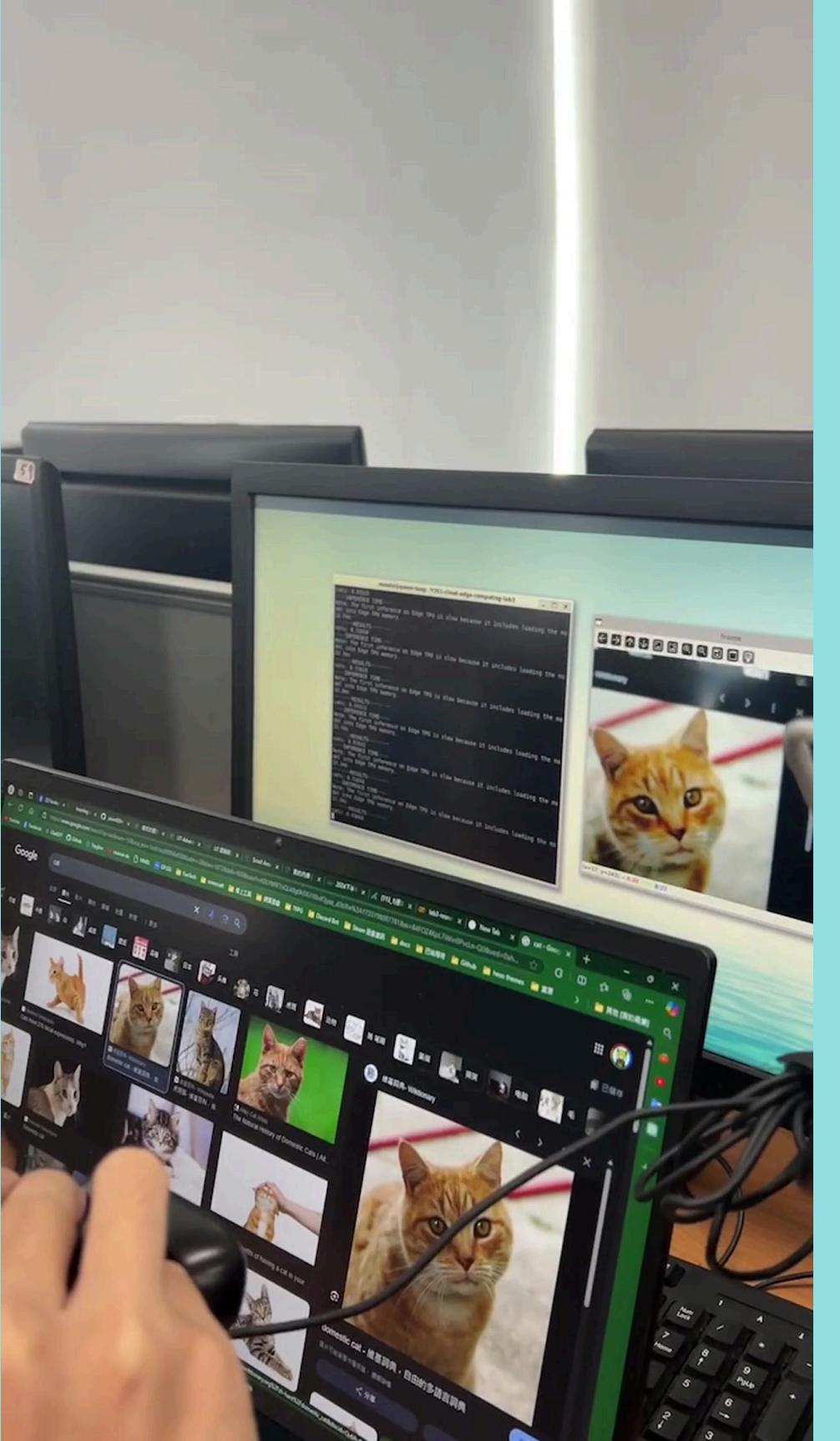


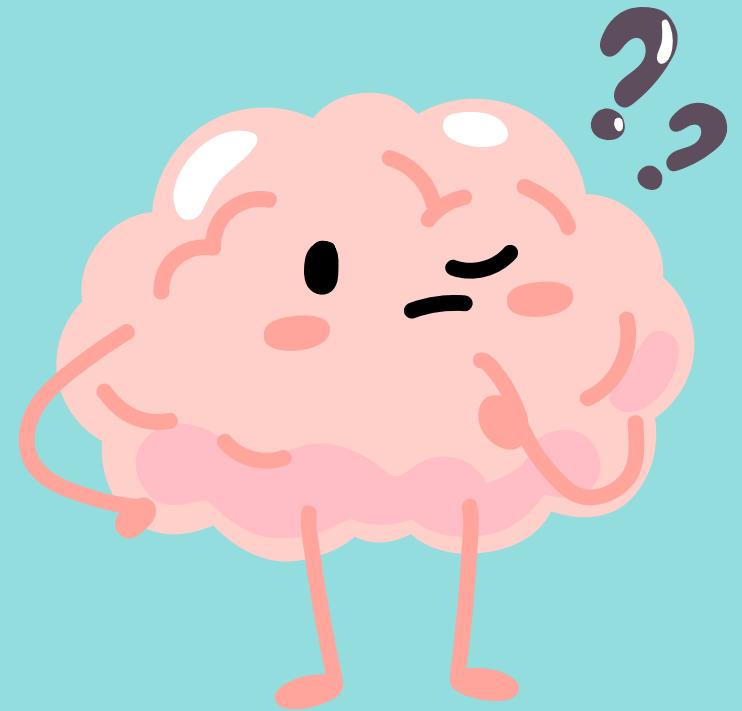
04. Edge端 驗證結果



實際測試影片

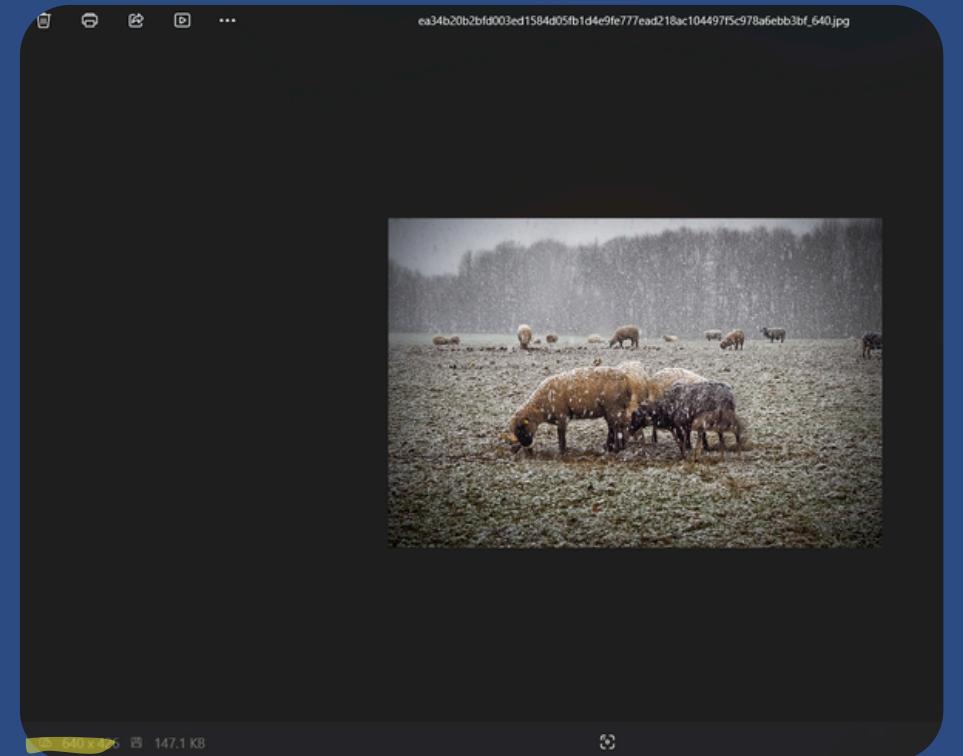
!!點我!!





05. 遇到問題與解決





Q: 在尋找資料集不可以有小於224×224的圖片

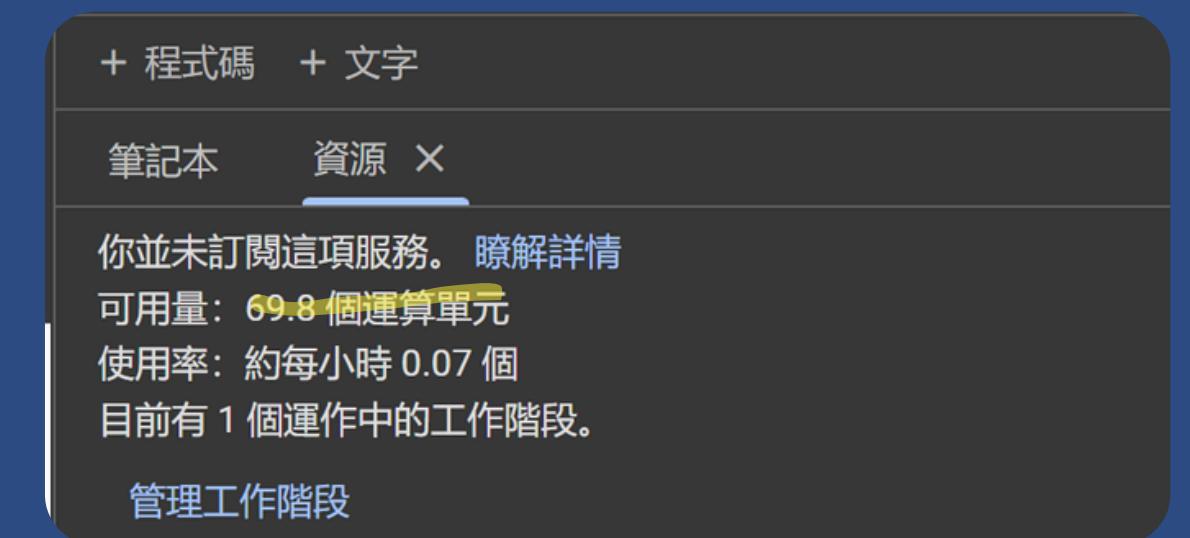
A: 在下載完網路上的資料集之後要每一張檢查是否有符合需求

Q: 在訓練模型時，時間太長colab會中斷連線

A: 購買colab的運算單元，可以節省訓練時間，速度較快

Q: 在量化H5檔案 -> tflite 遇到路徑問題

A: 重新下載h5檔案，再次複製路徑。



```
[4]: IMAGE_SIZE = 224
DATASET_DIR = "/content/drive/MyDrive/Colab/lab3/sample"
MODEL_PATH = "/content/drive/MyDrive/Colab/lab3/model-resnet18.h5"
TFLITE_PATH = "/content/drive/MyDrive/Colab/lab3/sample/models/resnet18.tflite"
```

06.心得



- 
- 透過這次專案實作，從 PC 端模型訓練到 Edge 端部署，我深刻體會到人工智慧在實際應用中的挑戰與細節，特別是在模型量化與效能優化方面學到了許多實用技巧。克服 colab 訓練中斷的問題，讓我更加了解訓練過程中資源管理的重要性，並學會有效利用工具來提升效率。此外，實作過程中雖然遇到 H5 檔案轉換為 tflite 的路徑問題，但透過查詢官方文件與社群討論，成功解決了問題，進一步理解了工具間的相容性與路徑規劃的重要性。同時，在尋找符合要求的資料集（圖片需大於 224×224 ）時，透過資料清理與重新篩選的過程，更深刻認識到資料前處理對於模型訓練的關鍵性。
 - 這次專案也讓我理解到 Edge 端應用的重要性，如何在有限的硬體資源下實現高效能的影像辨識，是未來值得深入探索的課題。而模型的量化過程（H5 到 tflite）則讓我認識到壓縮模型大小對部署的重要性，尤其是在開發嵌入式應用時，如何平衡準確率與效能是一大挑戰。在課堂討論與同學合作中，透過交換心得與解決策略，我對深度學習的實踐有了更全面的理解，每次討論與指導老師的建議，也幫助我重新審視自己的實作方式，避免只專注於技術細節而忽略整體架構的設計。這次經驗讓我對人工智慧的應用有更多期待，也為未來在邊緣運算領域的探索奠定了良好的基礎。

Thank you !

