基於CNN實現邊緣端自動化芒果等級分類

報告人：Group2 - 110034403 祝煜恆

CONTENTS

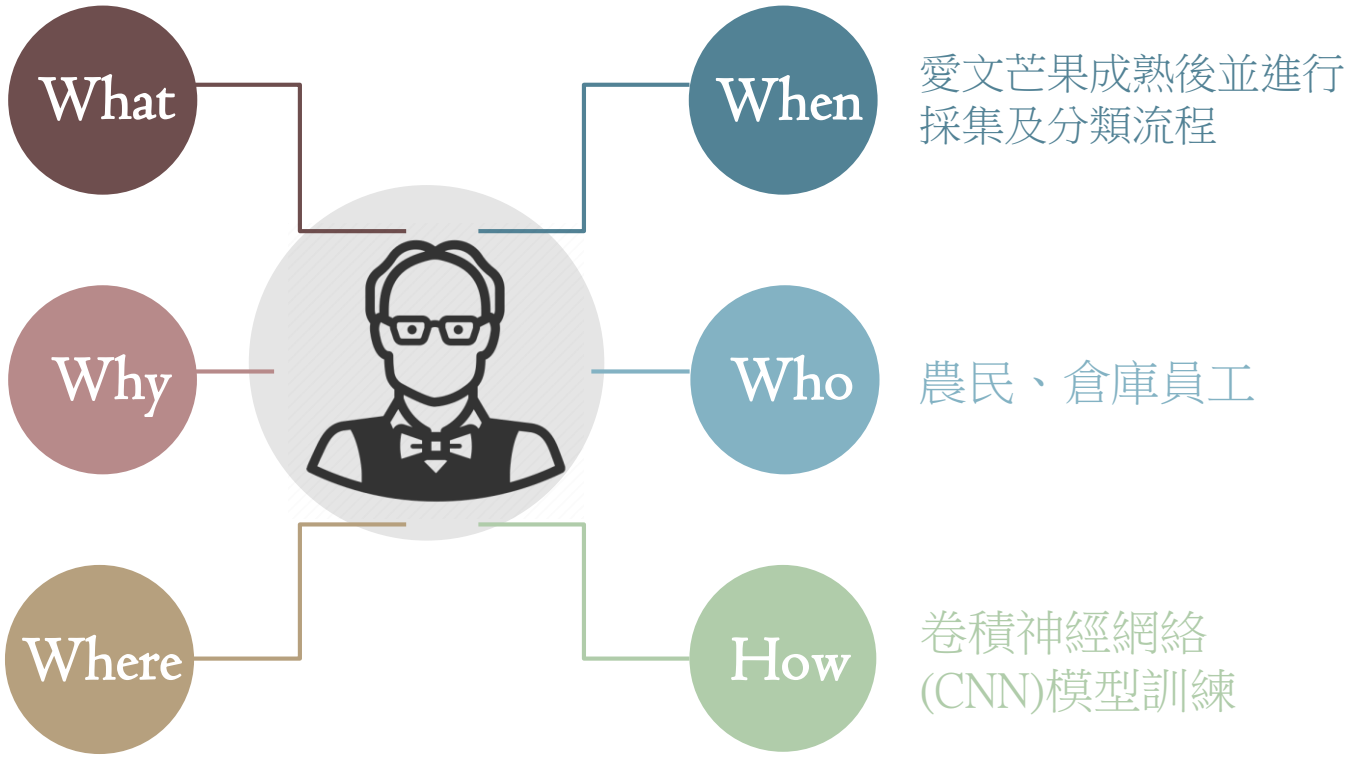
背景介紹

- ◆ 愛文芒果於近年銷量持續增長，三大外銷高經濟生鮮果品之一
- ◆ 依靠人工篩選：
  - 農村人口流失導致人力短缺
  - 篩果流程也導致保鮮期壓縮
- ◆ 場地因素無法保證能夠使用主機、相機等傳統影像辨識系統

研究目的

建立自動化、輕量化、同時能夠在邊緣端執行的AI影像辨識自動分類系統

問題描述

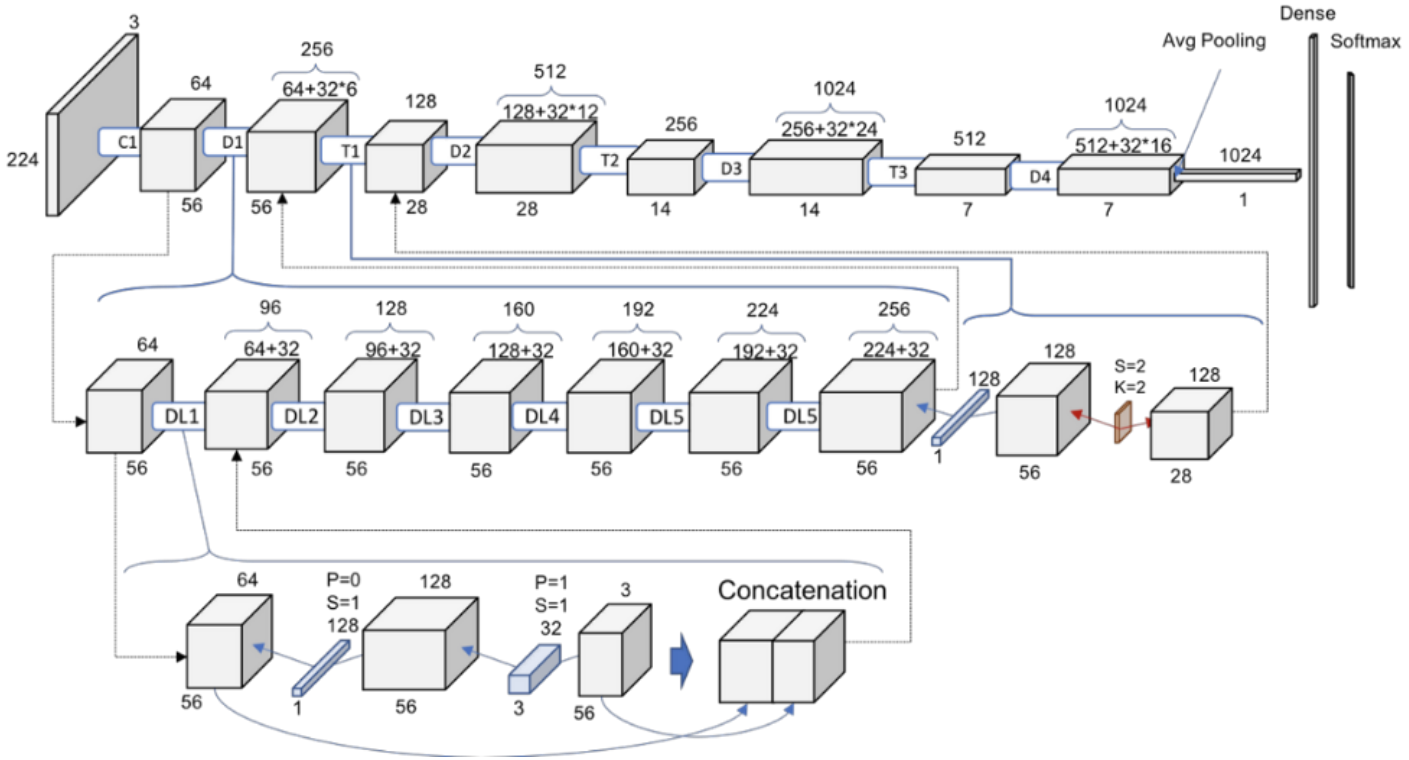

What
愛文芒果人力檢視分類耗時，需依賴員工的經驗，有機會因經驗不足而辨識錯誤
When
愛文芒果成熟後並進行採集及分類流程
Why
透過深度學習模型，讓廠商在採收後能夠減少分類時間
Who
農民、倉庫員工
Where
農產地、倉庫
How
卷積神經網絡(CNN)模型訓練

研究方法

CNN 網路架構名稱	Parameters
MobilenetV2	3,538,984
DenseNet201	20,242,984
InceptionResnetV2	55,873,736

- 採用數種在ImageNet圖片資料集的pre-trained model
- 選擇各代表著輕、中、重量級的網絡以不調整參數的情況進行初始訓練查看精準度
- 最後以DenseNet201準確度78.47%作為初選模型

DenseNet201



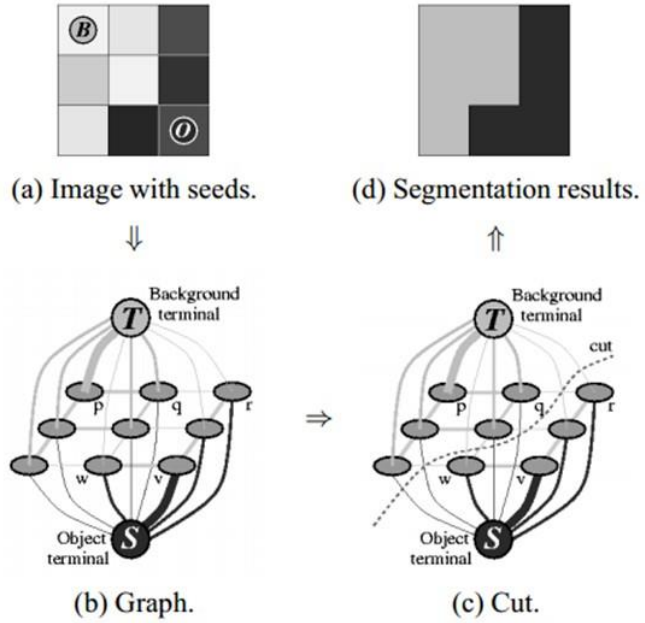
- 使用了dense block結構
- L層網絡用了L(L+1)/2個連接
- 實現特徵重用
- 對特徵的極致利用來達到更好的效果

資料集介紹

- 2020年全國大專生人工智慧競賽 - 愛文芒果等級分類競賽
- 拍攝背景多數為採收現場以及倉庫等場景
- 分為A、B、C三種等級，總數8002張

資料前處理（1/3）

- 使用了OpenCV的Grabcut算法進行簡單的前後背景分割
- 原理：
  1、定義包含前景的物體並進行標註
  2、每一個畫素都被看作與周圍畫素相連線
  3、超過一定閥值後就會切斷連線



資料前處理（2/3）

◆ 為了減少黑色背景被當作愛文芒果資訊，進行黑色背景的去除

◆ 做法：

1、findContours()輪廓描繪

2、最小內接矩形boundingRect()

3、黑色背景的去除

```
img2=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(img2,50,255,cv2.THRESH_BINARY)
cnts,hier = cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
dot = []
if cnts is not None:
    for c in cnts:
        min_list = []
        x,y,w,h=cv2.boundingRect(c)
        min_list.append(x)
        min_list.append(y)
        min_list.append(w)
        min_list.append(h)
        min_list.append(w*h)
        dot.append(min_list)
max_area=dot[0][4]
for inlist in dot:
    area=inlist[4]
    if area >= max_area:
        x=inlist[0]
        y=inlist[1]
        w=inlist[2]
        h=inlist[3]
        max_area=area
img=img[y:y+h, x:x+w]
cv2.imwrite(os.path.join(outdir,os.path.basename(jpgfile)), img)
```

資料前處理（3/3）

資料混和+重新分割

將訓練與測試集混和再以9:1重新分割訓練與測試集

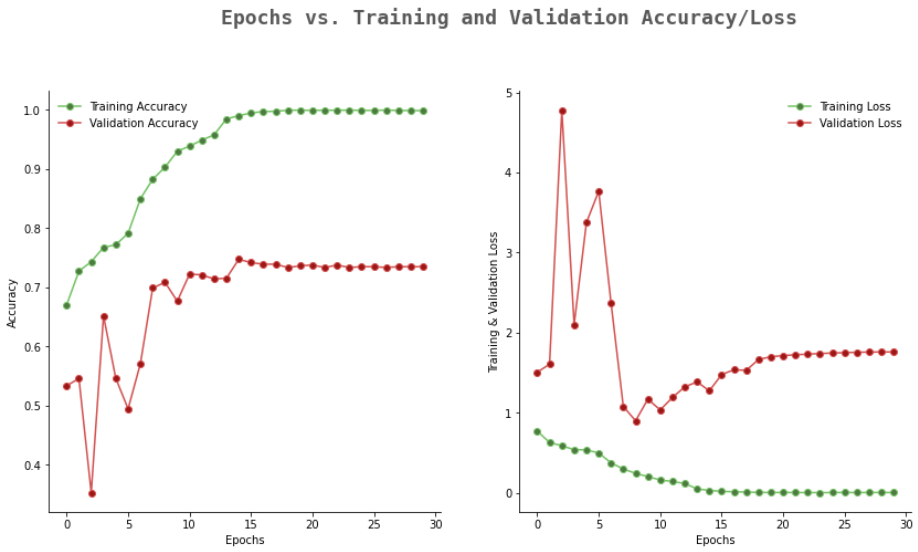
```
[ ] #shuffle是將圖片順序隨機打亂
    X_train,  y_train  =  shuffle(X_train,y_train,  random_state=101)

    #  train_test_split為交叉驗證的函數，函數中X_train代表所要劃分的樣本特徵集，y_train代表所要劃分的樣本結果，test_size代表樣本占比
    X_train,X_test,y_train,y_test  =  train_test_split(X_train,y_train,  test_size=0.1,random_state=101)
```

資料分割 (訓練與驗證集)

以9:1將訓練集分割成訓練與驗證集

```
[ ] history  =  model.fit(X_train,y_train,validation_split=0.1,  epochs  =50,  verbose=1,  batch_size=28,
                          callbacks=[tensorboard,checkpoint,reduce_lr])
```

模型建立與訓練（1/2）



利用TensorFlow開源軟體庫進行訓練，使用其中預訓練過的CNN網路架構

```
basemodel = DenseNet201(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
```

額外加入2D全局平均池化層減少模型參數，避免過擬合; 以及加入Dropout層減少訓練的時間，最後再利用Softmax函數確保能更有效的分類

```
basemodel = DenseNet201(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
model = basemodel.output
model = tf.keras.layers.GlobalAveragePooling2D()(model)
model = tf.keras.layers.Dropout(rate=0.5)(model)
model = tf.keras.layers.Dense(3, activation='softmax')(model)
```

添加L2正規化（Reguralization）減少過擬合

```
if isinstance(layer, tf.keras.layers.Conv2D):
        model.add_loss(lambda layer=layer: l2(layer.kernel))
```

模型建立與訓練（2/2）

自動調整學習率，避免學習率過大導致無法收斂。

```
[ ] # ReduceLROnPlateau 可以在訓練過程中優化學習率。val_accuracy表示要監控的是accuracy,只要它一直沒提升就會調整學習率；factor表示要改變學習率時所改變的幅度
    # patience表示每幾個epoch沒改變就會去調整學習率；min_delta為閾值，只有改變量超過這個數字才會被採用
    reduce_lr = ReduceLROnPlateau(monitor = 'val_accuracy', factor = 0.3, patience = 2, min_delta = 0.001,
                                  mode='auto',verbose=1)
```

```
conv5_block32_2_conv (Conv2D)   (None, 7, 7, 32)     36864       conv5_block32_1_relu[0][0]
__________________________________________________________________________________________________
conv5_block32_concat (Concatena (None, 7, 7, 1920)   0           conv5_block31_concat[0][0]
                                                                 conv5_block32_2_conv[0][0]
__________________________________________________________________________________________________
bn (BatchNormalization)         (None, 7, 7, 1920)   7680        conv5_block32_concat[0][0]
__________________________________________________________________________________________________
relu (Activation)               (None, 7, 7, 1920)   0           bn[0][0]
__________________________________________________________________________________________________
global_average_pooling2d_1 (Glo (None, 1920)         0           relu[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 1920)         0           global_average_pooling2d_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 3)            5763        dropout_1[0][0]
==================================================================================================
Total params: 18,327,747
Trainable params: 18,098,691
Non-trainable params: 229,056
__________________________________________________________________________________________________
```

參數優化

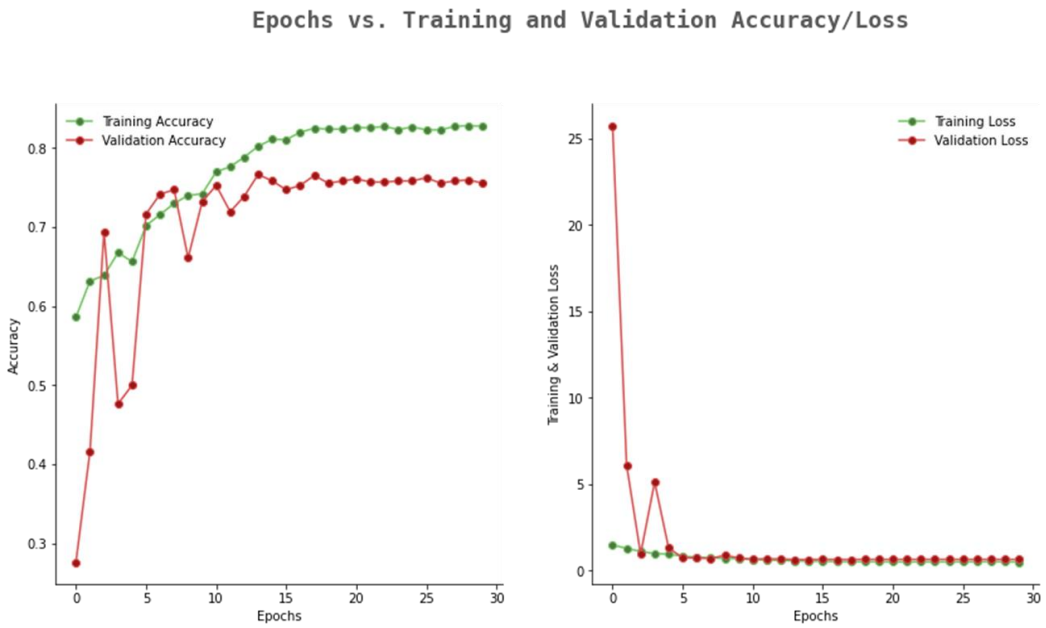
因子	說明	水準1	水準2
A	Dropout	0.5	0.6
B	Optimizer	Adam	Adagrad
C	Batch Size	8	16

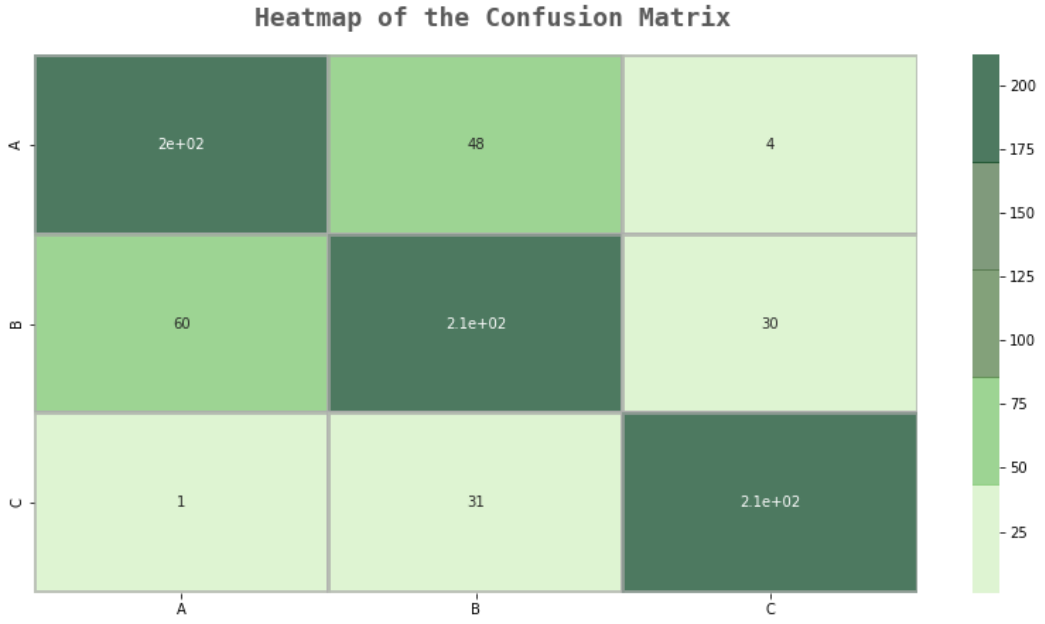
實驗	Dropout	Optimizer	Batch Size
1	0.5	Adam	8
2	0.5	Adagrad	16
3	0.6	Adam	16
4	0.6	Adagrad	8

- 利用田口方法做三個因子各兩水準，應用L4直交表幫助參數優化。
- 每個實驗皆進行30epoch

最佳模型參數結果

實驗	Dropout	Optimizer	Batch Size	Test Accuracy
4	0.6	Adagrad	8	0.7958

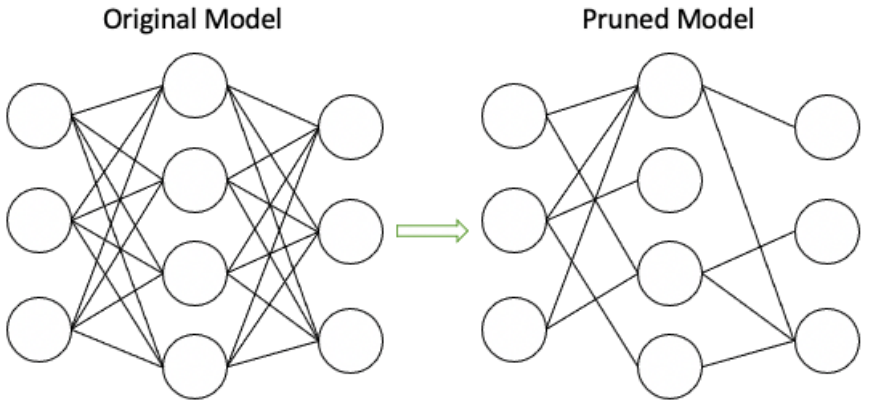




模型壓縮（1/2）

模型剪枝 Model-Pruning

- 推理階段並不會使用到所有的參數
- 使用權重的重要性排序並刪減不重要的部分
- 採用稀疏比Sparsity Ratio的目標（0參數所佔比例）進行迭代修剪直至達到當前的稀疏度目標




```
import tensorflow_model_optimization as tfmot
prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude
# Compute end step to finish pruning after 2 epochs.
batch_size = 16
epochs = 2
validation_split = 0.1
num_images = X_train.shape[0] * (1 - validation_split)
end_step = np.ceil(num_images / batch_size).astype(np.int32) * epochs

# Define model for pruning.
pruning_params = {
      'pruning_schedule': tfmot.sparsity.keras.PolynomialDecay(initial_sparsity=0.50,
                                                               final_sparsity=0.80,
                                                               begin_step=0,
                                                               end_step=end_step)
}

model_for_pruning = prune_low_magnitude(model, **pruning_params)

# `prune_low_magnitude` requires a recompile.
model_for_pruning.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model_for_pruning.summary()
```

模型壓縮（2/2）

剪枝前模型權重

```
1 model.weights[1]
```

```
<tf.Variable 'conv1/bn/gamma:0' shape=(64,) dtype=float32, numpy=
array([ 8.06224421e-02,  1.52602587e-02, -1.17970752e-02,  3.39749083e-02,
        6.58297241e-02,  3.57881188e-02, -4.68702689e-02,  2.37534679e-02,
        4.36919145e-02,  5.26746660e-02, -2.00182242e-07,  6.69109300e-02,
        4.44241688e-02,  1.32779125e-02, -1.31027311e-01,  3.68724056e-02,
        1.02752388e-01, -1.34629637e-01,  4.02679779e-02, -4.25500385e-02,
       -1.89552903e-02, -2.15617735e-02,  7.83290118e-02,  1.52008086e-01,
```

準確度：0.7958

模型大小：73MB

剪枝後模型權重

```
1 model_for_pruning.weights[1]
```

```
<tf.Variable 'conv1/conv/kernel:0' shape=(7, 7, 3, 64) dtype=float32, numpy=
array([[[[-0.        ,  0.        ,  0.        , ..., -0.        ,
           0.        ,  0.        ],
         [ 0.        ,  0.        ,  0.        , ...,  0.        ,
           0.        ,  0.        ],
         [ 0.        , -0.        ,  0.        , ...,  0.        ,
           0.        ,  0.        ]],
```

準確度：0.7425

模型大小：70MB

模型轉換（1/2）

Tensorflow Lite

- 提供了將模型文件.h5轉換為讓Android上可執行的文件格式.tflite

- 將標籤及有無正規化等metadata資訊加入tflite文件

```
converter = tf.lite.TFLiteConverter.from_keras_model(model_for_export)
pruned_tflite_model = converter.convert()
with open('.\pruned_model_exp4.tflite', 'wb') as f:
    f.write(pruned_tflite_model)
```

```
from tflite_support.metadata_writers import writer_utils
from tflite_support.metadata_writers import image_classifier
ImageClassifierWriter = image_classifier.MetadataWriter
_MODEL_PATH = "pruned_model_exp4.tflite"
_LABEL_FILE = "label.txt"
_SAVE_TO_PATH = "pruned_model_exp4_meta.tflite"
_INPUT_NORM_MEAN = 0
_INPUT_NORM_STD = 1
writer = ImageClassifierWriter.create_for_inference(
    writer_utils.load_file(_MODEL_PATH), [_INPUT_NORM_MEAN], [_INPUT_NORM_STD],
    [_LABEL_FILE])
# Verify the metadata generated by metadata writer.
print(writer.get_metadata_json())
# Populate the metadata into the model.
writer_utils.save_file(writer.populate(), _SAVE_TO_PATH)
```

模型轉換（2/2）

Android Studio

◆ 利用官方的簡易android腳本

◆ 加入tflite文件以及設定相關參數

（讀取相機資訊、顯示順序）

```
private val fruitModel = FruitModel.newInstance(ctx)
// TODO 6. Optional GPU acceleration


override fun analyze(imageProxy: ImageProxy) {

    val items = mutableListOf<Recognition>()

    // TODO 2: Convert Image to Bitmap then to TensorImage
    val tfImage = TensorImage.fromBitmap(toBitmap(imageProxy))
    // TODO 3: Process the image using the trained model, sort and pick out the top results
    val outputs = fruitModel.process(tfImage)
        .probabilityAsCategoryList.apply { this: (Mutable)List<Category!>
            sortByDescending { it.score } // Sort with highest confidence first
        }.take(MAX_RESULT_DISPLAY) // take the top results

    // TODO 4: Converting the top probability items into a list of recognitions
    for (output in outputs){
        items.add(Recognition(output.label, output.score))
    }
```

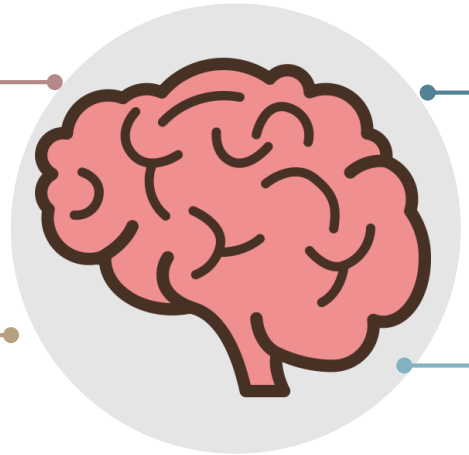
結果討論

貢獻

- 測試了不同大小網絡在少量類別的表現
- 實現可以將大型網絡修剪並不損失過多的表現能力
- 導入到線下邊緣端環境進行測試

適用性

- 精確度結果尚可
- 可基於不同的硬體條件、環境導入自動化



侷限性

- 資料集圖片分類依據不明確，A與B類圖片相似程度很高
- 時程關係沒有進行模型剪枝的參數調整。

未來展望

- 利用收集更多具代表性的圖像來增強分類表現
- 嘗試利用瑕疵檢測作為分類依據