國立清華大學

智慧化企業整合
Intelligent Integration of Enterprise

Project 3

基於 CNN 實現邊緣端自動化芒果等級分類

指導教授： 邱銘傳 教授

組別： 第二組
學生： 110034403 祝煜恆

目錄

一、 背景介紹

（一）背景與動機

台灣重要出口農產品之一的愛文芒果於近年銷量持續增長，不僅躍升為三大外銷高經濟生鮮果品之一，更將外銷國拓展至日本、中國、美國以及香港等地。雖然，在各國當地政府的政策配合下，台灣芒果較以往提高了知名度並拓展市佔率，卻還是遭遇其他同為芒果出口國（菲律賓、泰國）的削價競爭，因此諸多品種改良、採收後處理技術以及品牌行銷等提昇產品價值的工作，仍待科技輔助來推進。

而其中亟待改善的是採收後處理技術。愛文芒果採收後依品質篩選為 A、B、C 三等級，依序為出口用、內銷用、加工用。然而愛文芒果依靠人工篩選，除了農村人口流失導致人力短缺，篩果流程也因保鮮期壓縮地極短，導致篩果階段約有 10% 的誤差，若以外銷金額估計，每年恐怕損失 1600 萬台幣。

圖 1　A、B 及 C 等級之愛文芒果圖像

（二）研究目的

由於愛文芒果目前採收方式多數仍然依賴於人力檢視，而這除了需滿足大量的人力需求以外還必須憑藉良好的分揀經驗來進行，眾多的不確定因素使得難以保證愛文芒果在採集後的出產品質。同時由於農產地多數為郊外地區，無法保證網絡的覆蓋程度且基於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，因此希望能夠利用本地愛文芒果資料庫來建立自動化、精準化、同時能夠在邊緣端執行的 AI 影像辨識自動分類系統。

(三)問題描述

表 1 5W1H

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
Why	透過深度學習的模型，建立愛文芒果影像的分類模型，讓廠商在採收後能夠減少分類時間，並降低人工辨識錯誤機會。
Where	農產地、倉庫
When	愛文芒果成熟後並進行採集及分類流程
Who	農民、倉庫員工
How	卷積神經網絡(CNN)模型訓練

（四）資料集介紹

本研究使用 2020 年全國大專生人工智慧競賽之愛文芒果等級分類競賽公開資料集。由 BIIC Lab 與台灣瓦克國際股份有限公司合作，耗時數年收集愛文芒果影像資料包含三種等級類別之愛文芒果圖像，而拍攝背景多數為採收現場以及倉庫等場景。

二、 研究方法

（一）模型選擇

本研究使用 CNN 對愛文芒果圖像進行辨識與分類。現今 CNN 的網路架構眾多且在實務領域中各有各的長處，但由於愛文芒果等級分類未有明確的依據（例如有可能為表皮紋理、斑點、色澤等因素）而不確定適用的網絡架構大小，因此先採用數種在 ImageNet 圖片資料集預訓練過且網絡深度各代表著小中大的 CNN 網路：MobileNetV2、DenseNet201 以及 InceptionResNetV2，在不調整參數的情形下套用在本次研究的圖片資料集中，並選出一個精確度較高的 CNN 網路架構來進行微調並進行參數調整以提升準確度。

(1)MobilenetV2：

MobilenetV2 是由 Google 所提出的一個主要應用在移動端的輕量級新 CNN 網路。它主要強調於設計小模型進行訓練，主要強調使用了深度級可分解卷積操作，並使用 average pooling 將特徵變成 1*1 再接上全連接層進行分類；而 V2 版本引入了殘差結構和 bottleneck 層，這層主要表示利用線性方式進行特徵的激活並採用短路連線變成殘差結構，使得模型在內存的使用上能夠大大的減少，MobilenetV2 網路架構如圖 2：

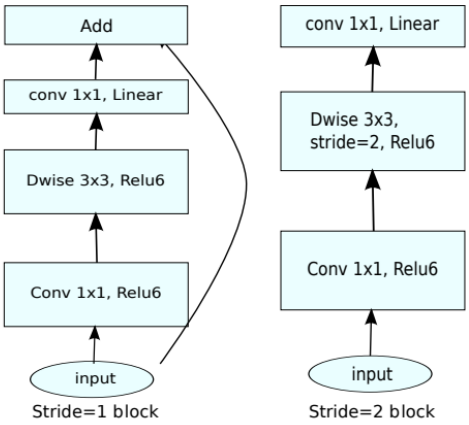

圖 2 MobilenetV2 網路架構

（參考資料：https://medium.com/ai-academy-taiwan/efficient-cnn-%E4%BB%8B%E7%B4%B9-%E4%BA%8C-mobilenetv2-7809721f0bc8）

(2)DenseNet201：

DenseNet 最大的特點在於其區別於一般的卷積神經網絡強調加深或加寬的方式來作為提升效果的方向，而是透過對特徵的極致利用來達到更好的效果。其使用了 dense block 結構，並與一般的 L 層網絡就有 L 個連接不同使用了 L(L+1)/2 個連接， 而基於這種 dense block 設計互相連接所有的層，具體來說就是每個層都會接受其前面所有層作為其額外的輸入。另一大特色是通過特徵在 channel 上的連接來實現特徵重用(feature reuse)因此在每個卷積層的 feature map 數量都較小，而是的特徵和梯度的傳遞更加有效，網絡也更容易訓練。DenseNet201 網路架構如圖 3：

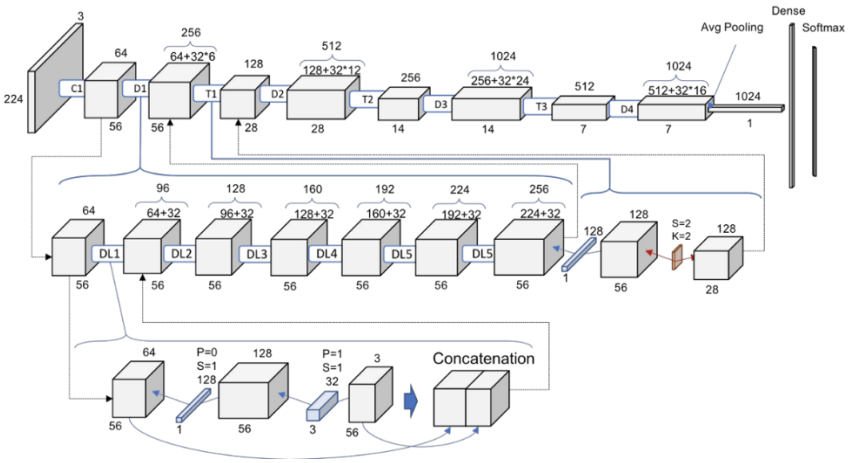

圖 3 DenseNet169 網路架構

（參考資料：https://blog.csdn.net/u014380165/article/details/75142664）

(3)InceptionResnetV2：

由 Google 團隊採用了殘差網絡的概念來進行設計的 CNN 架構，主要強調於殘差連接，即允許模型之中存在 shortcuts 讓研究者能夠訓練更深的神經網絡。其中 Stem 採用了了類似 InceptionV3 前面層的卷積層；Inception-ResNet module 採用 Inception-ResNet-A、Inception-ResNet-B、Inception-ResNet-C，及加入 Reduction-B 來降低 feature map 尺寸，而這樣的設計使得這麼大的模型但計算量大大缺降低而且擁有更快速的收斂速度，InceptionResnetV2 網路架構如圖 4：

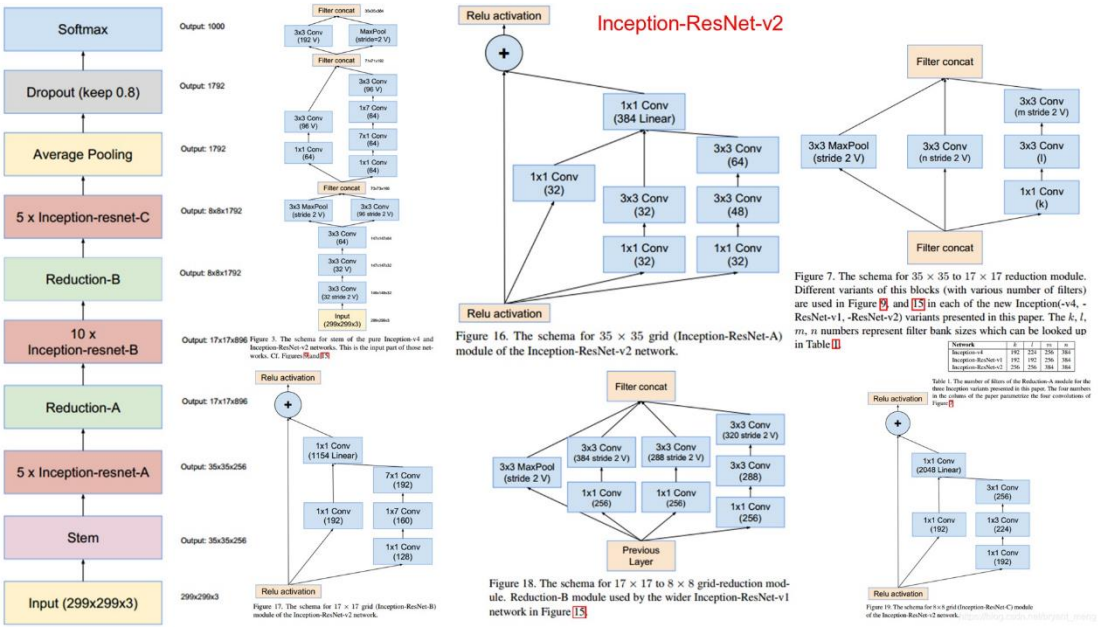

圖 4 InceptionResnetV2 網路架構

（參考資料：https://medium.com/ching-i/inception-%E7%B3%BB%E5%88%97-inceptionv4-inception-resnet-v1-inception-resnet-v2-42be5d23b2ec）

CNN 網路架構初選結果如表 2：

CNN網絡架構名稱	最高accuracy	所需epoch數
MobileNetV2	0.74722	15
DenseNet201	0.7847	13
InceptionResNetV2	0.76389	12

表 2 CNN 網路架構初選結果

因此最終我們選擇 DenseNet201 網路作為本次研究的 CNN 分類網路架構。

（二）、個案研究

(1) 資料介紹

本研究使用 2020 年全國大專生人工智慧競賽之愛文芒果等級分類競賽公開資料集。資料集中包含三種等級類別之愛文芒果圖像，分別為等級 A、B 及 C，拍攝背景多數為採收現場以及倉庫等場景。表 3 為各類別圖片數量表：

表 3 各類別圖片數量表

等級類別	圖片數量
A	2531 張
B	2966 張
C	2502 張

(2) 資料前處理

Step 1：

由於我們非專業人士，無法判別愛文芒果分類是否正確。因此我們人工檢視是否有空白或模糊圖片於其中，最終確定圖片集皆為清晰愛文芒果照片。

Step 2：

由於原本的圖片背景多數為現場拍攝，為了剔除因背景資訊對訓練過程造成的影響，這裡使用了 OpenCV 的 Grabcut 算法進行簡單的前後背景分割，原理為 1、先定義包含前景的物體並進行標註；2、對於被標註的每一個畫素都被看作與周圍畫素相連線；3、當前景與背景的畫素連線超過一定閥值後就會切斷連線並視作非前景的物體：

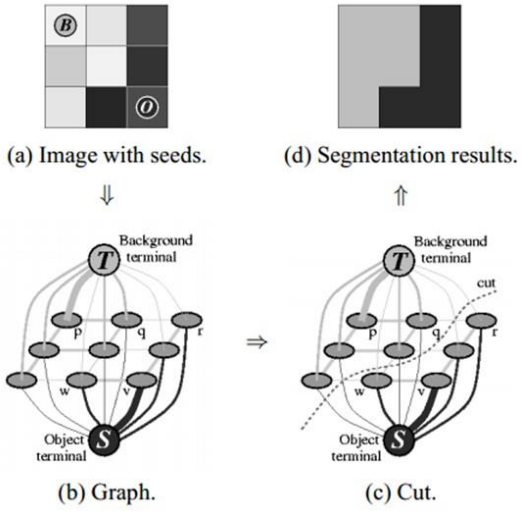

圖 5 Grabcut 演算法示意

```
try:
    img = cv2.resize(img, (width,height), interpolation=cv2.INTER_CUBIC)
    mask = np.zeros(img.shape[:2], np.uint8)# 創建大小相同的掩模
    bgdModel = np.zeros((1,65), np.float64)# 創建背景圖像
    fgdModel = np.zeros((1,65), np.float64)# 創建前景圖像
    rect = (17,20,207,206)
    cv2.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_RECT)
    mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
    img = img*mask2[:,:,np.newaxis]
```

圖 6 Grabcut 程式

圖 7 使用 Grabcut 進行前後景分割

Step 3：

為了減少進行 Grabcut 分割後黑色背景被當作愛文芒果資訊，使用 OpenCV 套件 findContours()函數進行物體的輪廓描繪，並利用最小內接矩形 boundingRect() 框選物體 ROI 並進行黑色背景的去除：

```
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)
```

圖 8 背景去除程式

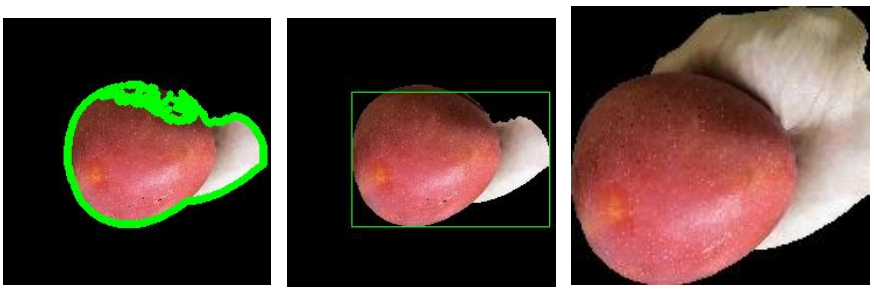

圖 9 物體輪廓描繪、內接矩形、ROI 裁剪

Step 4：

我們沒額外劃分驗證集圖片資料。因此我們先將三種等級之圖像資料進行合併，隨機打亂後再按照訓練與測試集 9:1 的比例重新劃分。如圖 10：

```
[ ]  #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)
```

圖 10 隨機分割訓練與資料集

最後再將訓練集以 9:1 分割為訓練與驗證集。如圖 11：

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

圖 11 分割訓練與驗證集

(三)、模型建立與訓練

本研究利用 TensorFlow 開源軟體庫進行訓練，其提供許多已在 ImageNet 圖片資料集中預訓練過的 CNN 網路架構。因此除了引入網路外，並再最後加入 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，也能避免過擬合；最後再利用 Softmax 函數確保能更有效的分類。模型建立的程式碼如圖 12：

```
[ ]  effnet  =  tf.keras.applications.densenet.DenseNet121(weights='imagenet',include_top=False,input_shape=(image_size,image_size,3))
     model  =  effnet.output
     model  =  tf.keras.layers.GlobalAveragePooling2D()(model)
     model  =  tf.keras.layers.Dropout(rate=0.5)(model)
     model  =  tf.keras.layers.Dense(4,activation='softmax')(model)
     model  =  tf.keras.models.Model(inputs=effnet.input,  outputs  =  model)
```

圖 12 模型建立

除了使用 Optimizer 自動調整學習率與其他相關參數外，為了加快訓練速度，與避免學習率過大導致無法收斂，還應用了 ReduceLROnPlateau 函數。只要在連續幾個 epoch 中，驗證集的 accuracy 皆未提升，當前的學習率便會乘上一個小於 1 的倍數，幫助訓練。相關程式碼如圖 13：

```
[ ] #  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為關值，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)
```

圖 13 學習率調整

但在初步實驗後發現預訓練模型在此資料集上都會有過擬合的現象（圖 14），初步推測是由於預訓練模型所含特徵資訊太多而在新數據集上造成誤差太大，因此用權重衰減的方式加入 L2 正規化讓每層輸出的權重更小一點。相關程式碼如圖 15：

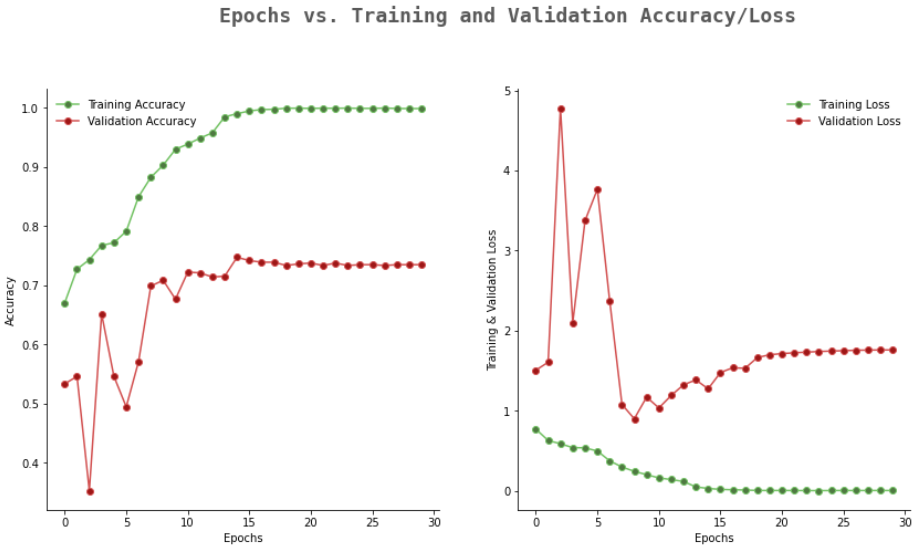

圖 14 過擬合問題

```
l2 = tf.keras.regularizers.l2(1e-4)
for layer in model.layers:
    # if hasattr(layer, 'kernel'):
    # or
    # If you want to apply just on Conv
    if isinstance(layer, tf.keras.layers.Conv2D):
        model.add_loss(lambda layer=layer: l2(layer.kernel))
```

圖 15 加入 L2 正規化

（四）參數優化

三種 CNN 網路架構除了預訓練時所獲得的最佳模型參數外，其餘的參數皆是使用相同的數據，也因此還有超參數優化的空間。其餘參數初始狀態如表 4：

表 4 其餘參數初始數值

參數說明	初始數值
Initial learning rate	0.001
Learning rate change	0.3
Optimizer	Adam
Dropout rate	0.5
Batch size	16

我們也利用了實驗設計中田口方法的方式，有效減少調整參數的總次數，並獲得相同的結果。我們選擇了上述所提到的其中三項參數作為三個因子，並使用兩個水準，應用 L4 直交表來幫助參數優化。直交表詳情如表 5：

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

表 5 L4 直交表

（五）模型壓縮方法

1. 模型剪枝（Model Pruning）

在模型訓練完成後進行推理（inference）的階段（也就是使用新資料測試模型能力）時會產生過參數化的現象，意即在訓練階段所得出的參數在推理階段並不會使用到所有的參數。因此如何在最小化影響準確率的情況下找出冗餘參數並進行裁剪是模型剪枝的核心問題，最簡單的做法就是使用權重的重要性排序並刪減不重要的部分，這裡採用稀疏比 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，而過程中都會重新計算稀疏度，相關程式碼如圖 16：

```
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()
```

圖 16 模型剪枝

（六）模型轉換方法

Tensorflow 套件提供了將模型文件.h5 轉換為讓 Android 上可執行的文件格式.tflite，相關執行程式如圖 17：

```
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)
```

圖 17 模型轉換

將標籤及有無正規化等 metadata 資訊加入 tflite 文件，相關執行程式如圖 17：

```
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)
```

圖 17 加入 metadata

參考官方釋出的簡易 android 腳本，並加入 tflite 文件以及設定相關參數，相關程式碼如圖 18：

```
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))
    }
```

圖 18 Android Studio 相關參數設定

三、 模型績效

利用田口方法之 L4 直交表進行實驗取得之成果如表 6：

實驗	Dropout	Optimizer	Batch Size	Accuracy
1	0.5	Adam	8	0.7667
2	0.5	Adagrad	16	0.79028
3	0.6	Adam	16	0.78333
4	0.6	Adagrad	8	0.7958

其中以 Dropout 設為 0.6、Optimizer 選擇 Adagrad 以及 Batch Size 為 8 之參數設定取得之模型 A 結果最佳。取以上結果進行模型 A 剪枝後，可取得參數量更小的模型 B，結果呈現如圖 19 及圖 20：

```
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,
```

圖 19 剪枝前模型權重

```
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.        ]],
```

圖 20 剪枝後模型權重

模型 A 及 B 準確度分別為 0.7958 及 0.7425 ，模型進行修剪後確實能降低推理速度但同時間推理能力也會對應的稍微下降，因此採用模型 A 轉換為.tflite 文件作為移動端 App 開發模型，相關成果展示如圖 21：

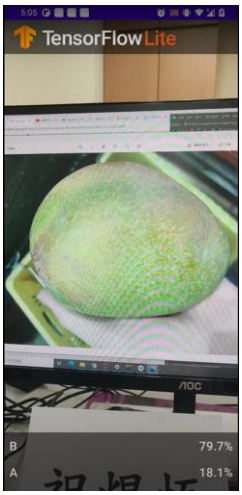

圖 21 軟體成果

四、結果與討論

其中， DenseNet210 模型在 L4 直交表三種因子四種實驗組合下分別在 Dropout/ Optimizer/ Batch Size 得到最佳準確率分別為 0.6/ Adagrad/ 8/。得出模型最終準確率為 0.7958，其中使用模型剪枝後的最終準確率為 0.7425.

總結來說，對於此次研究結果可歸納於以下四點：

（一）貢獻

本次研究中，主要測試了不同大小的 CNN 網路架構在少量類別的表現，並實現了可以將大型網絡修剪並不損失過多的表現能力，並導入到線下邊緣端環境進行測試。

（二）侷限性

因本研究的圖片資料集圖片分類依據不明確，其中 A 與 B 類圖片相似程度很高，造成訓練過程中不斷有過擬合的情況。此若用於測試的圖片有經過擾動，可能會導致分類精確度的下降；且因為時程的關係沒有進行模型剪枝的參數調整。

（三）適用性

由於本研究最終結果尚可，在進行模型的調整之後我認為能應用於現場實施的機率非常大，且壓縮的方式亦可基於不同的硬體條件、環境進行微調再導入自動化，

（四）未來展望

除了現有的愛文芒果圖像，應該可以再進一步利用收集更多具代表性的圖像來增強分類表現；也能轉換方式例如嘗試利用瑕疵檢測作為分類依據

五、參考資料

（一）https://www.kaggle.com/jaykumar1607/brain-tumor-mri-classification-tensorflow-cnn

（二）https://hackmd.io/@computerVision/rkVq2xRWO

(三)https://www.tensorflow.org/model_optimization/guide/pruning/pruning_with_keras

（四）https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android#3