

**National Sun Yat-sen University**  
**Department of Electrical Engineering**

**112-2 機器學習系統設計實務與應用**

**HW3 Image Classification基礎**

**組別：你說的隊**

**組員：B103012034 黃麗穎**

**B103012041 黃詣辰**

**B103015006 胡庭翊**

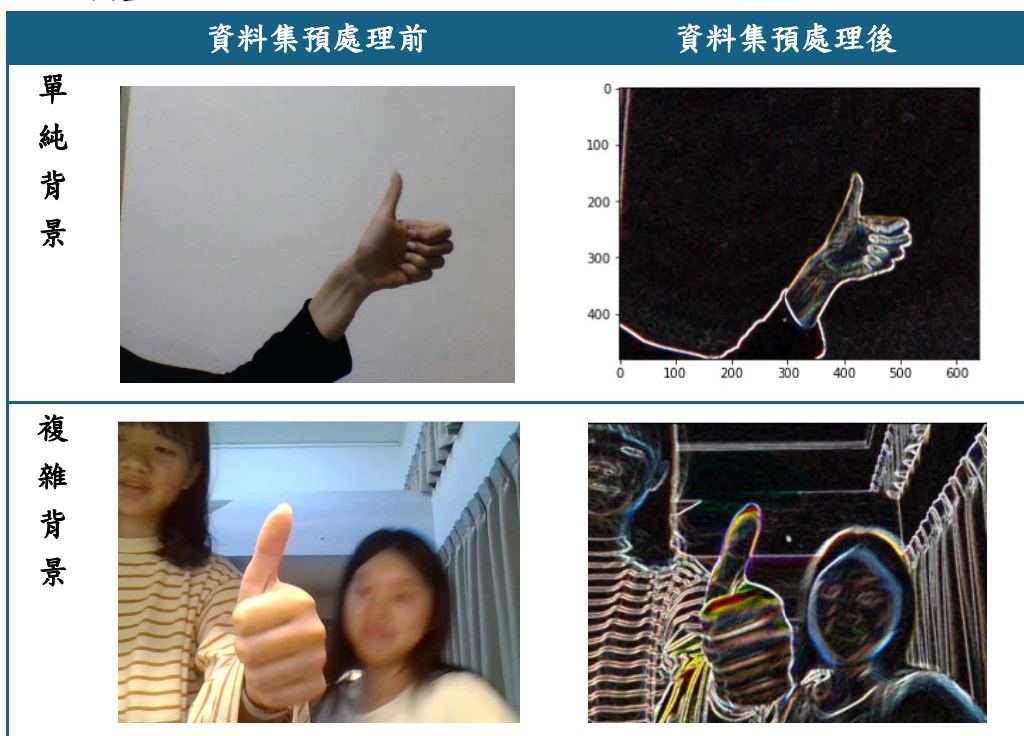
**B103015001 林佳明**

## 一、 資料蒐集時遇到的問題與解決方法


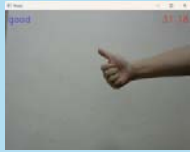




為了比較資料集的效果，我們將模型內的參數設為:weight=None，優化器統一使用 adam，損失函數統一使用 categorical\_crossentropy 做 batch 等於 16，epoch15 次的訓練。

### I. 資料預處理

1. 測試物件:剪刀、石頭、布、好棒手勢、其他。
2. 資料數:1500
3. 模型:mobilenet



### 4. 預測結果

類別	剪刀	石頭	布	好棒手勢	其他
預處理前					
預測	剪刀	好棒(錯誤)	布	好棒	好棒(錯誤)
預處理後					
預測	剪刀	剪刀(錯誤)	布	其他(錯誤)	其他

## 5. 討論

針對剪刀、石頭、布資料集預測，我們在嘗試多種模型預測後，發現準確率都偏低，所以我們參考了 HW1 所學的，在存入圖片轉成 mat 檔之前，先把照片去除雜訊、邊緣偵測預處理後保留重要特徵，再存入 data.mat 檔中。我們預期上是能將圖像中不重要的資訊去除，保留重點資訊，例如在這裡圖像顏色對偵測效果而言是多餘資訊，而手的邊緣線條、形狀則是需要加強的資訊。

經過預處理，我們預期能提高手勢分辨的正確率。但實際測試後準確率仍然偏低。我們認為可能的原因在於此資料集的物件相似性其實蠻高的，拍攝時手部旋轉角度或比的手勢都不太一致，造成模型後續擷取特徵有一定的難度。因此，後續的比較與實作我們選擇較不容易在拍攝時有變化性的錢幣作為我們的資料集實作。

## II. 物件類別

### 1. 資料集(一)

類別:10 硬幣、100 元鈔票、1000 元鈔票

資料數:1500

### 2. 資料集(二)


類別:10 硬幣、100 元鈔票、500 元鈔票、1000 元鈔票

資料數:2144





### 3. 模型:mobilenet。

### 4. 預測結果:

資料集(一)

類別	10	100	1000
輸出			
預測	10	100	1000

資料集(二)

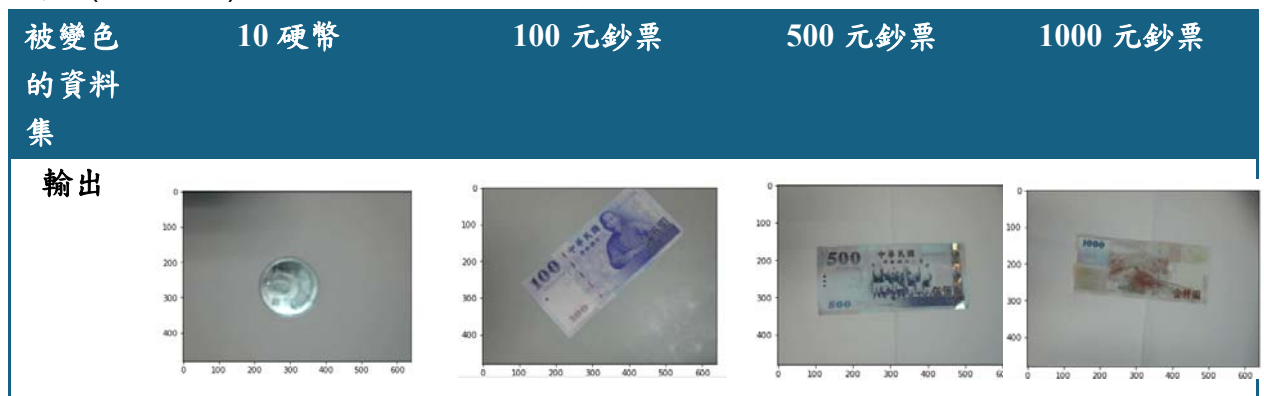
類別	10	100	500	1000
輸出				
預測	100(錯誤)	100	1000(錯誤)	1000

## 5. 討論

針對不同類別數量探討，在資料集(一)當中，類別數量只有 3 種，預測時準確度非常高，不管如何調整攝像頭的擷取角度，或把鈔票反轉，都能準確把 100 元與 1000 元區辨識出來。但當換成使用資料集(二)訓練，只是多加上一個類別 500 元鈔票，整個預測準確度下降非常多，10 元被辨識為 100 元，而 500 元被辨識成 1000 元。

我們認為 500 元被誤判的原因可能在於相較 100 元與 1000 元，500 元的顏色對比度較不明顯，若在特定光線下，500 元的顏色可能會與 1000 元相仿。另外，我們也列印出做為資料集的圖片觀察，我們發現轉換成 mat 檔的資料集有經過變色處理，而處理過後的 1000 元鈔票與 500 元幾乎相同。

至於 10 元硬幣的誤判，我們推測是新收集的資料集中的圖片中含有過多的背景(無關資訊)，物件主體占比太小，導致模型錯判。



## III. 物件數量

1. 類別: 10 硬幣、100 元鈔票、500 元鈔票、1000 元鈔票。
2. 模型: mobilenet
3. 資料數: 1500
4. 預測結果:

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
輸出				
預測	10	100	1000(錯誤)	1000




## 5. 討論:

資料集為多物件時，我們將鏡頭中原先只有一個樣本(例:一張 100 元鈔票)，改為有兩到三個樣本(例:3 張 100 元鈔票)去做訓練，而結果顯示大部分還是都可以做辨識，只有 500 塊無法辨識成功。推測可能是 500 跟 1000 在多物件訓練時，因為畫面裡比較多張鈔票，導致每張鈔票大小占比變小，而且 500 和 1000

塊的顏色在某些光線下顏色相似度極高，才導致模型無法正確判讀 500 與 1000。

#### IV. 資料集背景複雜度

1. 測試物件: 10 硬幣、500 元鈔票、1000 元鈔票。
2. 模型: mobilenet
3. 資料數: 1500
4. 預測結果:  
無背景

類別	10	500	1000
輸出			
預測	10	500	1000

有背景

類別	10	500	1000
輸出			
預測	100	1000(錯誤)	1000

#### 5. 討論:

無背景的照片準確度確實比較高，這是因為在無背景的照片中，只有偵測的物件，模型集中在該物件上並準確的抓取到主要特徵。有複雜背景的照片會導致模型受到背景干擾，模型可能會錯誤將背景中的特徵，辨識為的該物件的特徵，導致錯誤的分類或檢測。像是有背景的可能會把拿鈔票的方式當作一個判讀點，但換一個人測試就無法預測成功。

#### V. 空物件

1. 測試物件: 10 硬幣、100 元鈔票、500 元鈔票、1000 元鈔票。
2. 模型: mobilenet
3. 資料數: 1500
4. 預測結果:



## 無空物件

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
輸出				
預測	10	100	500	1000

## 有空物件

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
輸出				
預測	1000(錯誤)	100	1000(錯誤)	1000

## 5. 討論

我們在資料集中將 1/8 的資料替換為無任何物件，只有背景的图片作為測試。相較於無空物件預測結果無誤判的模型，增加了空物件的模型雖然仍能偵測成功一定比例的資料，但其正確性明顯降低許多。

## VI. 小結

由以上結果可得知，資料集的建立會大幅影響模型的偵測結果，良善的資料集會增加判斷的正確性，而干擾因素多的資料集則會增加誤判的可能性。要提高我們模型的可用性，資料集內最好是背景單純、無重複物件、無空物件，並且類別區分明顯的資料。

## 二、 神經網路架構探討

為了比較各個神經網路的差異，我們將參數設為:weight=None，優化器統一使用 adam，損失函數統一使用 categorical\_crossentropy 做 batch 等於 16，epoch15 次的訓練，並使用同一台電腦比較各個模型的運算速度與記錄各個網路的層數、大小等資訊。

測試物件: 10 硬幣、100 元鈔票、500 元鈔票、1000 元鈔票。

總資料數:2000

以下為我們計算層數的方法:

```
print("Number of layers in the base model: ", len(model.layers))
```

Number of layers in the base model: 92

運算速度則是觀察模型完成一個 epoch 所需的時間:

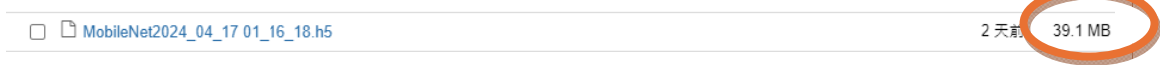
```
complete_time=time.strftime("%Y_%m_%d %H_%M_%S", time.localtime())
model.save('InceptionResNetV2'+str(complete_time)+'.h5')
```

```
WARNING:tensorflow:From C:\Users\User\anaconda3\envs\MNIST\lib\site-packages\tensorflow\python\ops\resource_variable_ops.py:43
5: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
Train on 2144 samples, validate on 1056 samples
WARNING:tensorflow:From C:\Users\User\anaconda3\envs\MNIST\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32
(from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/15
2144/2144 [=====] - 2261s 1s/sample - loss: 0.2214 - acc: 0.9366 - val_loss: 6.0246 - val_acc: 0.0606
Epoch 2/15
2144/2144 [=====] - 2131s 994ms/sample - loss: 0.0303 - acc: 0.9907 - val_loss: 7.5843 - val_acc: 0.23
86
Epoch 3/15
2144/2144 [=====] - 2120s 989ms/sample - loss: 0.0693 - acc: 0.9804 - val_loss: 13.3890 - val_acc: 0.0
085
Epoch 4/15
2144/2144 [=====] - 2114s 986ms/sample - loss: 0.0504 - acc: 0.9860 - val_loss: 9.3616 - val_acc: 0.19
70
Epoch 5/15
2144/2144 [=====] - 2109s 984ms/sample - loss: 0.0171 - acc: 0.9949 - val_loss: 10.0840 - val_acc: 0.1
165
Epoch 6/15
2144/2144 [=====] - 2118s 988ms/sample - loss: 0.0626 - acc: 0.9860 - val_loss: 12.3671 - val_acc: 0.0
208
Epoch 7/15
2144/2144 [=====] - 1540s 718ms/sample - loss: 0.0187 - acc: 0.9944 - val_loss: 8.7299 - val_acc: 0.13
16
Epoch 8/15
2144/2144 [=====] - 1071s 500ms/sample - loss: 0.0199 - acc: 0.9935 - val_loss: 10.0806 - val_acc: 0.2
197
Epoch 9/15
2144/2144 [=====] - 1071s 499ms/sample - loss: 0.0375 - acc: 0.9893 - val_loss: 11.5887 - val_acc: 0.0
227
Epoch 10/15
2144/2144 [=====] - 1072s 500ms/sample - loss: 0.0035 - acc: 0.9991 - val_loss: 11.5553 - val_acc: 0.0
218
Epoch 11/15
2144/2144 [=====] - 1072s 500ms/sample - loss: 0.0397 - acc: 0.9916 - val_loss: 13.6599 - val_acc: 0.0
000e+00
Epoch 12/15
2144/2144 [=====] - 1071s 500ms/sample - loss: 0.0244 - acc: 0.9930 - val_loss: 11.1137 - val_acc: 0.2
405
Epoch 13/15
2144/2144 [=====] - 1072s 500ms/sample - loss: 0.1031 - acc: 0.9762 - val_loss: 8.1140 - val_acc: 0.18
66
Epoch 14/15
2144/2144 [=====] - 1161s 541ms/sample - loss: 0.0491 - acc: 0.9902 - val_loss: 7.9368 - val_acc: 0.17
99
Epoch 15/15
2144/2144 [=====] - 1177s 549ms/sample - loss: 0.0210 - acc: 0.9939 - val_loss: 9.4078 - val_acc: 0.02
84
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

模型的偵測速度是紀錄右上角的 fps (每秒可以處理的圖像數量):



而模型大小則是從 h5 檔的檔案大小得知:

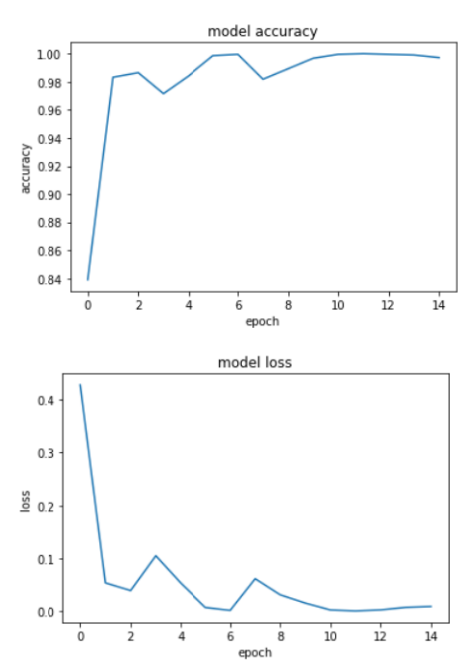


I. Mobilenet

1. 預測結果

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
輸出				
預測	10	100	500	1000

2. 訓練準確度圖





### 3. 網路大小

網路層數：92

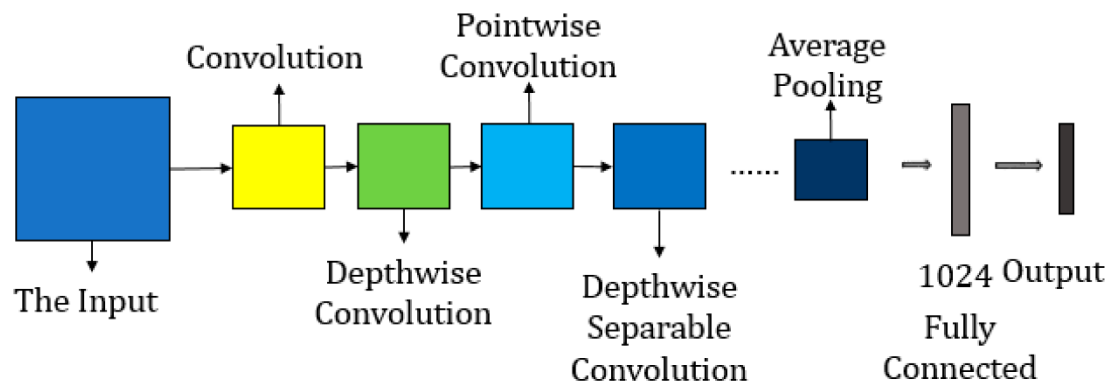
大小: 39.1MB

### 4. 運算速度

訓練運算速度: Epoch: 827s

預測運算速度: 12 fps

5. 分析: 我們用 Mobilenet 去做訓練後，電腦預測之後發現，10、100、500、1000 都能被清楚辨識，同時反應也很快，在切換鈔票的當下，馬上就能反應過來，輸出正確的預測結果。Mobilenet 是一個輕量級的深度神經網路架構，雖然他在計算上的精度不那麼高，但經過層層的堆疊之後，仍然可以保有他應有的準確度，所以這種架構比較適合用在我們如果今天在計算面積上有所限制時，Mobilenet 就會是一個能夠滿足限制，而且達到一定水準的準確度。其架構如下



輸入影像後，會先進入深度卷基層，接著到達逐點卷基層，做特徵提取的處理，再經過標準化後進入平均池化層將圖片轉為向量形式，方便我們後續做訓練，池化完成後進入全連接層，最後便可以執行分類。(圖摘自[8])

其中，我們可以透過調整網路架構中的參數 alpha 來控制網路的寬度

< 1.0 按比例減少每一層中的濾波器數量

= 1 默認的過濾器數量

> 1.0 按比例增加每一層中的濾波器數量

實際操作比較 alpha=1 及 0.75 發現 alpha=0.75 時明顯速度變快

優點	缺點
高效率	靈活性有限
快速	精準較低
記憶體占用小	不適用於大型數據集
	複雜性有限

MobileNet 為了達到快速與高效率的目的會產生犧牲部分精度的 trade off，但因為他的高效率使其可以在計算資源有限的移動和嵌入式設備上運行。MobileNet

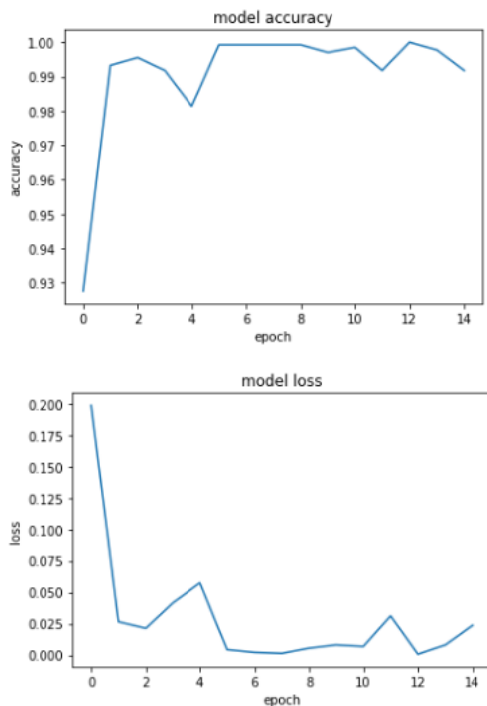
的應用範疇有目標檢測、圖像分類、人臉辨識、風格轉移、影片分析。

## II. Xception

### 1. 預測結果

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
輸出				
預測	10	100	1000(錯誤)	1000

### 2. 訓練準確度圖



### 3. 網路大小

網路層數：134

大小: 251MB

### 4. 運算速度

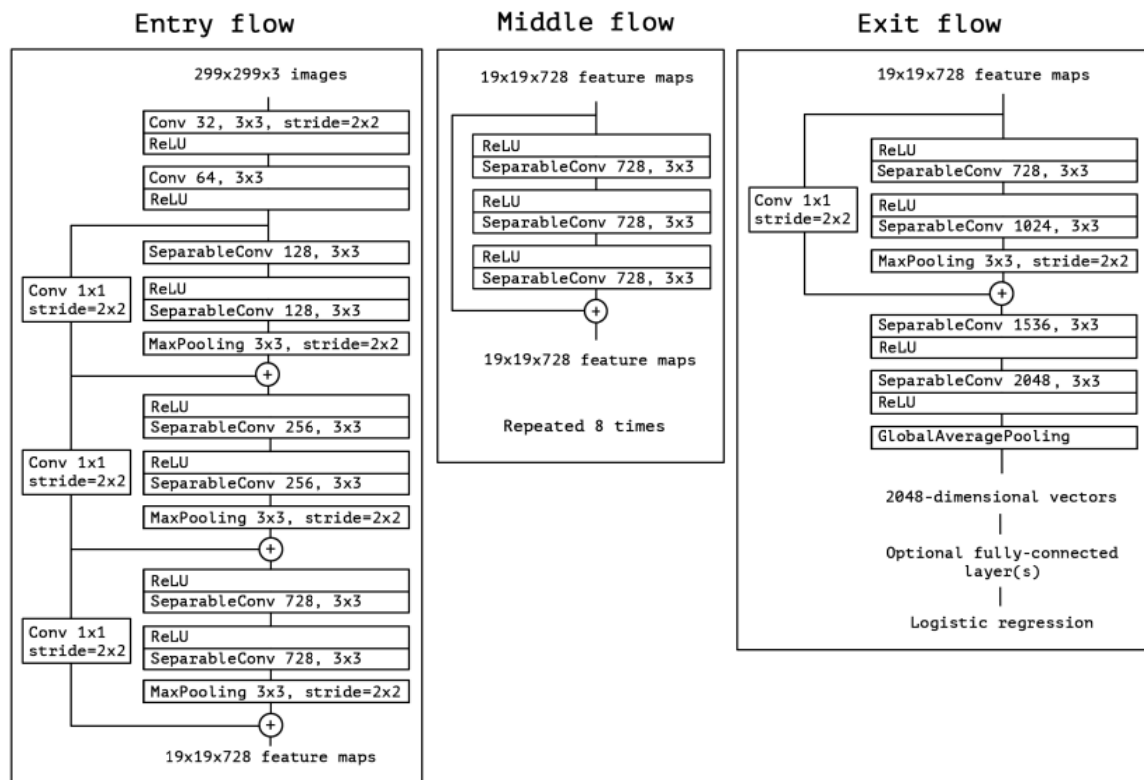
訓練運算速度:Epoch:45min

預測運算速度(平均): 6 fps

### 5. 分析:

我們用 Xception 做完訓練後，預測的結果只有 500 無法辨識，其他都可以清楚的做辨識，但 Xception 的計算精度比 Mobilenet 多了許多，預測準確度卻沒有像 Mobilenet 一樣好，所以這邊可能是已經有點 overfit 了，才會導致用了更複雜的網路架構反而沒有達到更好的結果。

Xception 架構是一系列深度可分離卷積層的線性堆疊，帶有殘差連接。這使得架構非常容易定義和修改，其網路架構發想於 inception 這個架構，不同的地方是他把 inception 原先卷基層的地方做了改進，從標準卷基層變成深度可分離層，如此一來可以降低參數的數量還有計算上的成本，雖然降低了成本，但因為計算精度較高，所以跟剛剛提到的 Mobilenet 相比起來，Xception 的計算成本還是高上許多，比較適合拿來訓練需要高精確度的資料集。







註:Xception 架構圖 (摘自[14])

### III. VGG

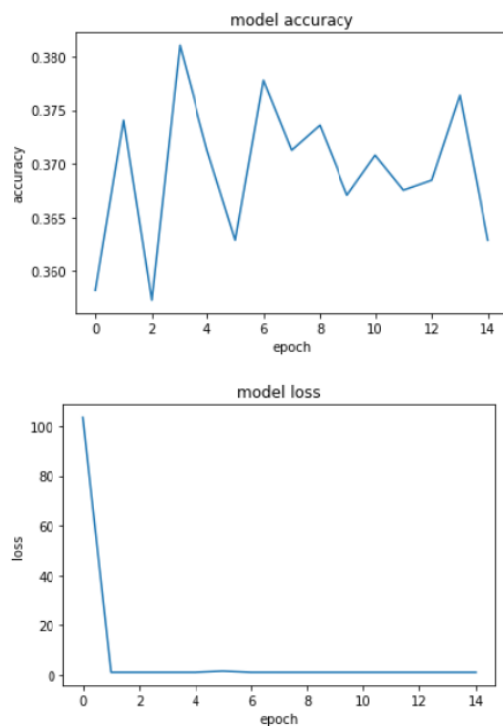
測試物件: 10 硬幣、100 元鈔票、500 元鈔票、1000 元鈔票。

#### ● VGG16

##### 1. 預測結果

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
輸出				
預測	10	10(錯誤)	10(錯誤)	10(錯誤)

##### 2. 訓練準確度圖



##### 3. 網路大小

網路層數 : 23

大小: 1.61GB

##### 4. 運算速度

訓練運算速度: Epoch: 60min

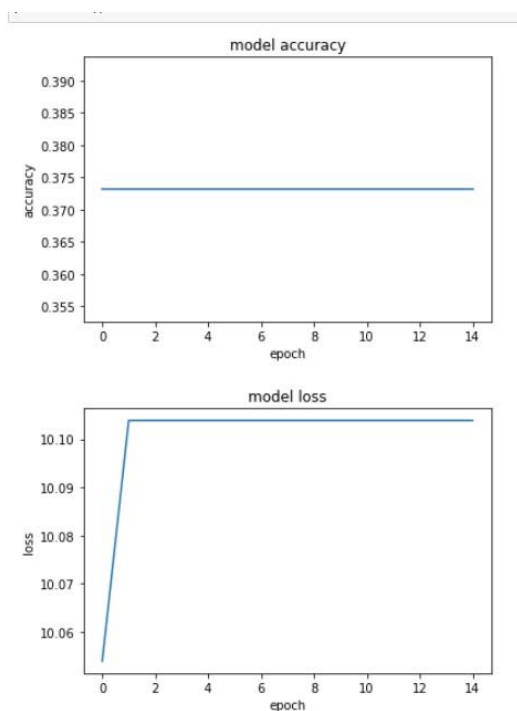
預測運算速度(平均): 6 fps

## ● VGG19

### 1. 預測結果

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
輸出				
預測	10	10(錯誤)	10(錯誤)	10(錯誤)

### 2. 訓練準確度圖



### 3. 網路大小

網路層數 : 26

大小: 1.68GB

### 4. 運算速度

訓練運算速度: Epoch: 32 分鐘

預測運算速度(平均): 3 fps

## ● VGG 分析:

我們用 VGG16 這個網路架構做完訓練後，只能預測出 10 元，而且其他鈔票也都顯示 10 元，預測準確度大不如前。

查了資料發現 VGG16 這個網路的架構十分簡單，同時也因為包含很多卷基層及全連接層所以導致他的架構很深層（可以提取到更深層的特徵值），也因為有很多的卷基層，導致計算量大量增加，這點我們自己在訓練時特別感同身受，

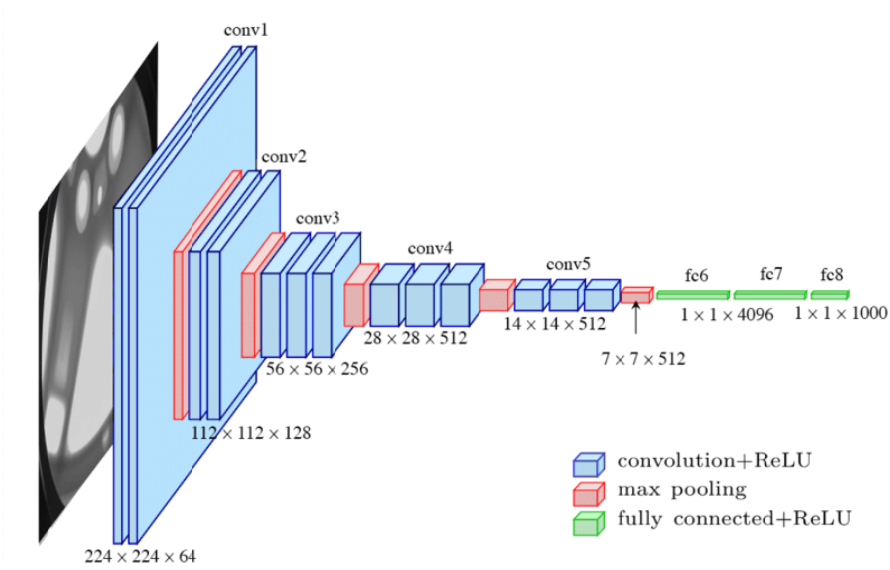


因為 VGG 每次訓練都是 10 小時起跳的，但 VGG 不適合用於訓練數據集小的模型，會很容易產生過擬合，為了解決這個問題，可以適當的 dropout，來減少 overfit 的發生。

用 VGG19 這個架構做完訓練後，發現跟 VGG16 一樣，所有鈔票都會顯示 10 元，不會出現其他的預測。

VGG19 是 VGG16 的延伸，只是多了更多的卷基層，所有計算又更加繁瑣複雜了，可以處理非常複雜的資料集，但同時對於我們小資料集來說，會產生嚴重過擬合的現象。





我們推測是因為我們的資料集太小，才不適合用 VGG 這個網路架構來做預測。下圖是 VGG 的架構



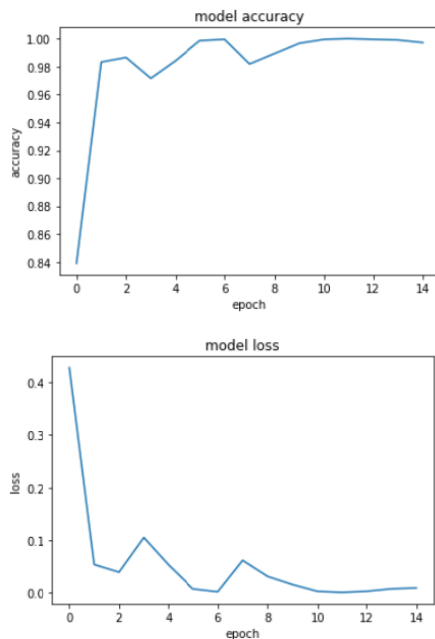
(圖摘自[9])

#### IV. Resnet50

##### 1. 預測結果

類別	10	100	500	1000
輸出				
預測	10	100	1000(錯誤)	1000

## 2. 訓練準確度圖:



## 3. 網路大小

網路層數 : 177

大小: 259MB

## 4. 運算速度

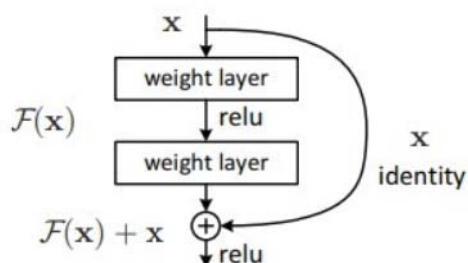
訓練運算速度: Epoch: 55min

預測運算速度(平均): 8fps

## 5. 分析:

Resnet50 的理論層數有 168 層，在課堂上老師有提到說，當 CNN 層數越多其實準確度不會一直上升，隨著層數的增加，梯度有可能會越來越小或越大，導致模型無法有效地學習。Resnet50 通過在每一層中引入一個 residual block 來解決梯度爆炸消失。

也就是將前面層的某一層數據輸出直接跳過多層引入到後面數據層的輸入部分，避免過度擬合。在實際測試中，準確度的部分只有 500 元辨識不出來。對比 VGG 提升很多。但訓練時間上比較久大概 7 小時。



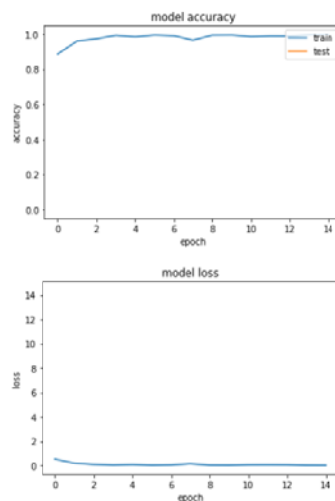
(摘自[10])

## V. InceptionV3

### 1. 預測結果

類別	10	100	500	1000
輸出				
預測	10	100	1000(錯誤)	1000

### 2. 訓練準確度圖：



### 3. 網路大小

網路層數：313

大小：263MB

### 4. 運算速度

訓練運算速度:Epoch:24min

預測運算速度(平均): 3 fps

5. 分析: InceptionV3 的理論層數有 159 層，此模型目的在於用相同層數但減少參數，減少訓練執行時間，所以與 Resnet50 想解決的方向不同，InceptionV3 作法是將卷積分解為不對稱卷積。把原本的  $3 \times 3$  卷積透過  $3 \times 1$  卷積後跟  $1 \times 3$  卷積取代。使用  $3 \times 3$  filter，參數的數量 =  $3 \times 3 = 9$ ，但是用  $3 \times 1$  and  $1 \times 3$  filters 取代，參數的數量 =  $3 \times 1 + 1 \times 3 = 6$ ，參數數量減少 33%。在實測上，InceptionV3 的準確度同樣只辨識不出 500 元，但感覺執行時間有快一些。雖然整體還是要跑 7 小時。

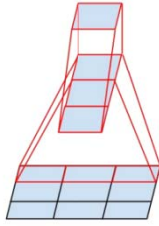


Figure 3. Mini-network replacing the  $3 \times 3$  convolutions. The lower layer of this network consists of a  $3 \times 1$  convolution with 3 output units.

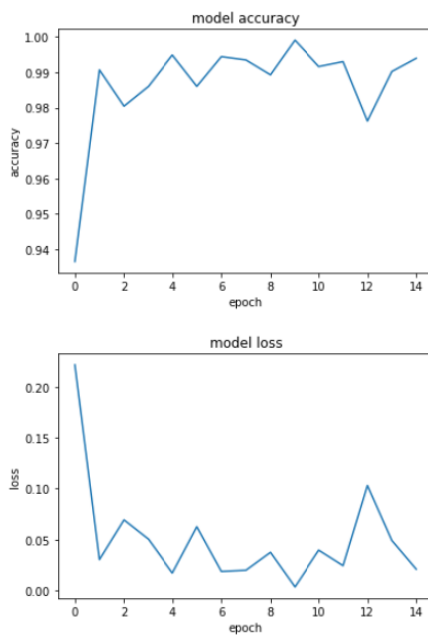
(摘自[11])

## VI. InceptionResnetV2

### 1. 預測結果

類別	10	100	500	1000
輸出				
預測	10	100	10(錯誤)	10(錯誤)

### 2. 訓練準確度圖



### 3. 網路大小

網路層數 : 782

大小: 263MB

### 4. 運算速度

訓練運算速度: Epoch: 47min

預測運算速度(平均): 2 fps

5. 分析:InceptionResnetV2 就是結合上面提到的 Inception 模型使用不對稱卷積層結合 Resnet 中 residual block 技術,InceptionResNetV2 使用極深的網路架構時,模型更容易收斂。但也因為 InceptionResnetV2 有 159 層,所以訓練時間極長,雖然理論上準確度希望可以比 Inception 和 Resnet 高,但實際預測發現 500 和 1000 元都沒變法預測成功。推測是因為我們的訓練資料集只 2000 張,所以應用在 572 層的模型中,有過擬合發生。所以針對此次作業,太複雜的模型可能不適合預測。

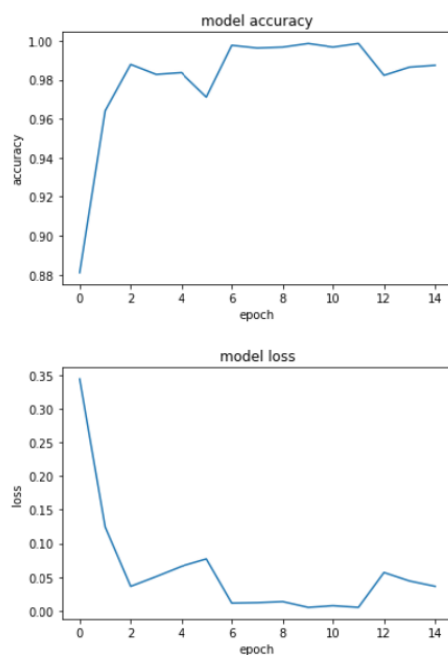
## VII. Desnet

### ● Desnet121

#### 1. 預測結果

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
輸出				
預測	10	100	10(錯誤)	1000

#### 2. 訓練準確度圖



#### 3. 網路大小

網路層數 : 429

大小: 84MB

#### 4. 運算速度




訓練運算速度:Epoch:90min

預測運算速度(平均): 4fps

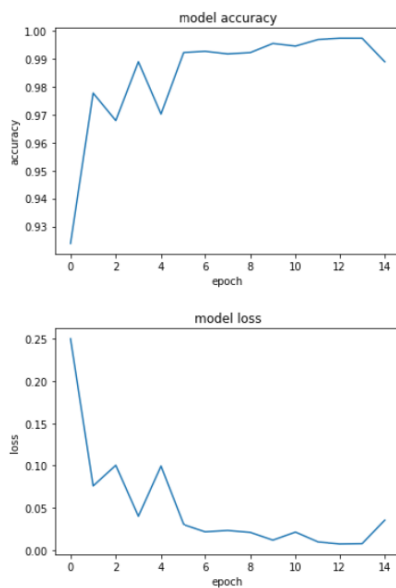


## ● Desnet169

### 1. 預測結果

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
輸出				
預測	10	100	100(錯誤)	1000

### 2. 訓練準確度圖



### 3. 網路大小

網路層數 :597

大小:149MB





### 4. 運算速度

訓練運算速度:Epoch:41min

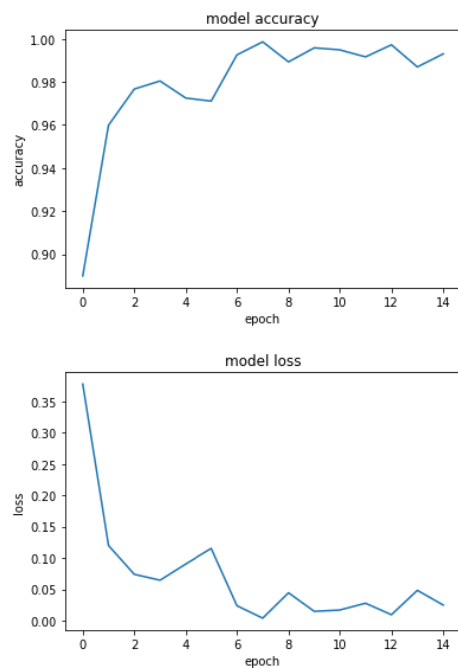
預測運算速度(平均):4.5fps

## ● Desnet201

### 1.預測結果

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
輸出				
預測	100(錯誤)	100	100(錯誤)	100(錯誤)

## 2. 訓練準確度圖



## 3. 網路大小

網路層數 :709

大小:222MB

## 4. 運算速度

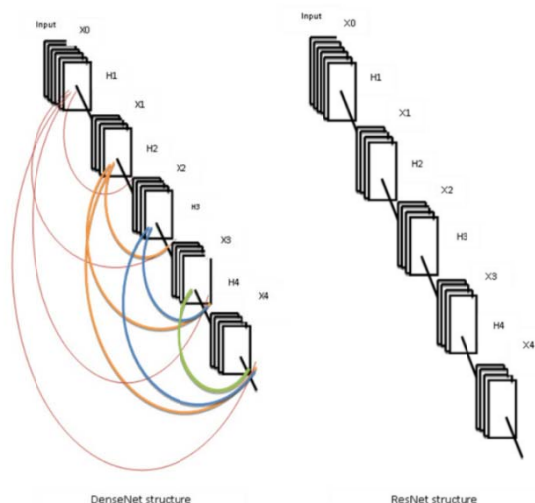
訓練運算速度:Epoch:76min

預測運算速度(平均): 7fps

## ● DenseNet 分析:

DenseNet(Densely Connected Convolutional Network) 架構分析:

ResNet 和 DenseNet 的設計是要減緩為了提高準確度使用更深的模型而導致梯度消失更加嚴重的問題，他們的作法是在卷積神經網路中加入短連接，使輸入和輸出之間的距離更短，使模型更有效率。DenseNet 為 ResNet 的改良版，對 ResNet 而言，用來穩定訓練模型的 identity shortcut 可能會限制他的表現能力，而 DenseNet 藉由 multi-layer feature concatenation 來緩解此問題，卻也使得他需要更高的 GPU 內存及更長的訓練時間。



圖一、ResNet&DenseNet 結構圖(摘自:[3])

DenseNet 因密集連接卷積網路而得名，為了在結構中既進行降採樣又進行特徵串接，神經網路被分為多個密集連接的 dense block。在 dense block 內部，特徵圖的大小保持不變。在密集塊外部的卷積 + 及池化操作可以進行降採樣操作，在密集塊內部，我們可以確保特徵圖的大小相同，以便進行特徵串接。在 DenseNet 中，bottle neck layer 旨計算和記憶體需求，減少輸入特徵圖的通道數，從而降低了計算成本，此設計可以提高模型的效率，同時保持較高的準確性。

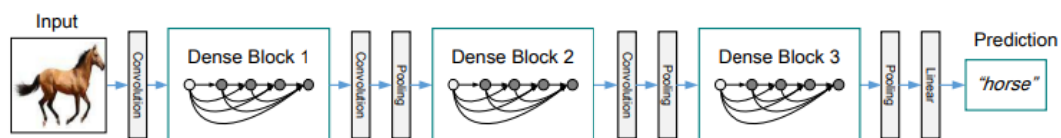


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

圖二、一個具有三個密集區塊的深度 DenseNet。兩個相鄰區塊之間的層被稱為轉換層，通過卷積和池化改變特徵映射的大小(摘自[6])

### DenseNet 的優缺點

優點	缺點
減緩梯度消失/反向傳播容易	計算複雜
特徵重用	消耗大量記憶體
減少參數	

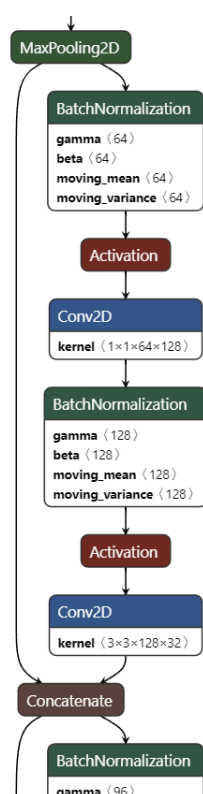
DenseNet121、DenseNet169 和 DenseNet201 的主要區別在於它們的層數

	層數	使用時機
DenseNet121	121	計算資源有限的環境
DenseNet169	169	需要較高準確性但計算要求不太高的應用
DenseNet201	201	需要極高準確性且計算資源較豐富的場合。

因為 DenseNet121 層數最少，所以計算效率最高，也較適合使用在移動或嵌入式設備。

## VIII. 小結

實際上使用 `len(model.layers)` 測出來的模型層數會跟理論的模型層數有差異，我們認為這是因為有些模型加入了短連接，而在實測中我們只能做到計算深度，無法比較出實際使用時的網路大小。下圖為 densenet201 的短連接示意圖：



此外，透過實作我們也發現並不是越深越大型的網路就能做到精準預測，反而太大型的網路容易出現 overfitting 的狀況。我們認為，若要減少 overfitting 的發生，減少樣本數或是降低 epoch 數會是可以嘗試的方法。

另外，除了我們實際上測試取得的數據，keras 官網也有提供各個模型的大小、深度、準確率以及參數供參考。這些都有助於我們訓練模型時挑選最合適的網路模型。

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88
DenseNet121	33 MB	0.745	0.918	8,062,504	121
DenseNet169	57 MB	0.759	0.928	14,307,880	169
DenseNet201	80 MB	0.770	0.933	20,242,984	201

### 三、 樹莓派實作成果

#### ● 資料集類別數量

1.

資料集(一):10 硬幣、500 元鈔票、1000 元鈔票。

數量:1500







資料集(二):10 硬幣、100 元鈔票、500 元鈔票、1000 元鈔票。

數量:2000

2. 模型:mobilenet。









3. 預測結果:

資料集(一):

類別	10	100	1000
PC			
預測	10	500	1000
樹梅派			
預測	10	500	1000



資料集(二):

類別	10	100	500	1000
PC				
預測	100	100	100	100
樹梅派				
預測	10	100	1000	1000

● 資料集物件數量(同張照片出現複數相同物件)

1.測試物件: 10 硬幣、100 元鈔票、500 元鈔票、1000 元鈔票。

數量:2000

2.模型:mobilenet

3.預測結果:

類別	10 硬幣	100 元鈔票	500 元鈔票	1000 元鈔票
PC				
預測	10	100	1000(錯誤)	1000
樹梅派				
預測	10	100	10(錯誤)	1000

● 樹莓派與電腦預測的結果比較：

從上面的表格我們可以發現，用樹莓派和用電腦預測的結果差異並不大，值得一提的是，原本用電腦跑預測時的 fps（每秒幀數）都能到大約 30 上下，但跑進樹莓派之後，fps 都變到只有個位數之小，所以我們在用樹莓派預測時，畫面

都很卡，要過好幾秒之後才會反應過來，對預測的過程增添了不少困難。

另外一點是，我們在下樹莓派時，跑 MobileNet 這個神經網路架構時可以順利載入權重檔，但再試其他模型時，樹莓派都會警告記憶體容量不足，有時候甚至會直接死當，沒辦法正常使用，所以對需要較大記憶體儲存空間的模型，我們都無法預測，因為權重檔載入不了。

```
0 gvfsd-trash 15236 20 34816 181
2079.537753] [ 939] 1000 939 7781 77 26624 47
0 gvfsd-metadata
2079.537761] [ 981] 1000 981 58996 2877 157696 6
0 lxterminal
2079.537772] [ 995] 1000 995 2128 223 16384 1
0 bash
2079.537782] [ 2239] 1000 2239 971 1 12288 52
0 xsel
2079.537792] [ 2323] 1000 2323 20934 971 67584 3875
0 thonny
2079.537802] [ 2329] 1000 2329 8571 1809 32768 5
0 python3
2079.537812] [ 2339] 1000 2339 320433 190497 1157120 2
0 python3
2079.537820] oom-kill:constraint=CONSTRAINT_NONE,nodemask=(null),cpuset=/,mems
allowed=0,global_oom,task_memcg=/,task=python3,pid=2339,uid=1000
2079.537902] Out of memory: Killed process 2339 (python3) total-vm:1281732kB,
anon-rss:761988kB, file-rss:0kB, shmem-rss:0kB, UID:1000 pgtables:1130kB oom_sco
re_adj:0
2079.723954] oom_reaper: reaped process 2339 (python3), now anon-rss:16kB, fil
e-rss:0kB, shmem-rss:0kB
pi@raspberrypi:~/Downloads/test $
```

由這次作業我們可以知道，雖然後面幾個網路架構能夠計算的更加精準，但也因為計算量龐大，需要更多的記憶體空間來儲存，導致根本無法下樹莓派做預測，也提醒了我們，在以後做相關研究時，不只要考慮準確度，更要考慮到在硬體上的限制，才不會導致訓練時結果很好，但下硬體之後根本無法預測的狀況發生。

#### 四、 分工比例表

組員	黃麗穎	黃詣辰	胡庭翊	林佳明
工作比例	25%	25%	25%	25%
簽名	黃麗穎	黃詣辰	胡庭翊	林佳明

## 五、 參考資料

- 1.Xinkai Yuan<sup>1</sup>, Lanrui Zhang<sup>2</sup>, Shuming Zhao<sup>3</sup>. DenseNet Convolutional Neural Network for Breast Cancer Diagnosis. : IC-ICAIE 2022, AHCS 9, pp. 197-202, 2023.
- 2.Chaoning Zhang, Philipp Benz, Dawit Mureja Argaw, Seokju Lee, Junsik Kim, Francois Rameau, Jean-Charles Bazin, In So Kweon. ResNet or DenseNet? Introducing Dense Shortcuts to ResNet. 2021 IEEE Winter Conference on Applications of Computer Vision (WACV)
- 3.N NandaPrakash, V Rajesh, Dumisani LicksonNamakhwa, Sandeep Dwarkanath Pande, SkHasane Ahammad. A DenseNet CNN-based liver lesion prediction and classification for future medical diagnosis. Scientific African Volume 20, July 2023, e01629
- 4.IT 邦幫忙.AI Facial Expression Recognition: Data, Model, Application 系列第 11 篇. [Day 11] 從零開始的 DenseNet 生活. 佑佑來了.2021-09-26
- 5.Aman Arora. DenseNet Architecture Explained with PyTorch Implementation from TorchVision. (GitHub blog post) Densely Connected Convolutional Networks August 2, 2020
- 6.Gao Huang, Zhuang Liu, Laurens van der Maaten. Densely Connected Convolutional Networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2261-2269
- 7.Medium. Cinnamon AI TaiwanCNN 模型-ResNet、MobileNet、DenseNet、ShuffleNet、EfficientNet. Jul 5, 2019  
(<https://cinnamonaitaiwan.medium.com/cnn%E6%A8%A1%E5%9E%8B-resnet-mobile-net-densenet-shufflenet-efficientnet-5eba5c8df7e4>)
- 8.圖 <https://www.mdpi.com/2073-8994/14/10/2055>
- 9.圖  
<https://gowrishankar.info/blog/gradcam-model-interpretability-vgg16-xception-networks/>
- 10.圖片出處: Kaiming He. Deep Residual Learning for Image Recognition
- 11.圖片出處: Rethinking the Inception Architecture for Computer Vision