水果辨識 CNN(HW4)

姓名:楊明哲
學號:00757143
日期:2021/06/12

方法

第一題:　套用的 pretrained model 是 VGG16
資料的取得和範例程式相同，在 generator 的地方加入 VGG16 的 preprocessing function

```
erator(horizontal_flip=True, preprocessing_function=tf.keras.applications.vgg16.preprocess_input, validation_split=0.1)

rator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input)
```

在載入 pretrained model 時，將 pooling 的參數寫入 'avg' 。 代表在 include Top = False 的情況下會在最後一層的 ouput 出來後先套一個 GlobalAveragePooling 再接到我們的 layer。

VGG16 後再接一個 filter 為 256 的 Dense Layer 和一個決定 class 的 Dense Layer 作結束

```
def fruit_model_pretrained(class_num, height, width, channel):
    model = tf.keras.models.Sequential(name="fruit_model_pretrained")
    pretrained = tf.keras.applications.VGG16(include_top=False, input_shape=(height, width, channel), pooling='avg', weights='imagenet')
    model.add(pretrained)
    model.add(tf.keras.layers.Dense(256, activation='relu'))
    model.add(tf.keras.layers.Dense(class_num, activation='softmax'))
    pretrained.trainable = False
    model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(), optimizer='adam', metrics=['accuracy'])
    return model
```

Summary 如下圖

```
fruit_model.summary()

Model: "fruit_model_pretrained"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 512)               14714688
_________________________________________________________________
dense (Dense)                (None, 256)               131328
_________________________________________________________________
dense_1 (Dense)              (None, 131)               33667
=================================================================
Total params: 14,879,683
Trainable params: 164,995
Non-trainable params: 14,714,688
_________________________________________________________________
```

Epoch 數設定為 20 下去訓練

```
history = fruit_model.fit(train_generator,epochs=20,validation_data=valid_generator)
```

```
Epoch 1/20
1905/1905 [==============================] - 105s 39ms/step - loss: 0.2998 - accuracy: 0.9471 - val_loss: 0.3095 - val_accuracy: 0.9356
Epoch 2/20
1905/1905 [==============================] - 72s 38ms/step - loss: 0.0411 - accuracy: 0.9902 - val_loss: 0.0853 - val_accuracy: 0.9746
Epoch 3/20
1905/1905 [==============================] - 73s 38ms/step - loss: 0.0366 - accuracy: 0.9915 - val_loss: 0.0917 - val_accuracy: 0.9803
Epoch 4/20
1905/1905 [==============================] - 73s 38ms/step - loss: 0.0275 - accuracy: 0.9944 - val_loss: 0.0929 - val_accuracy: 0.9801
```

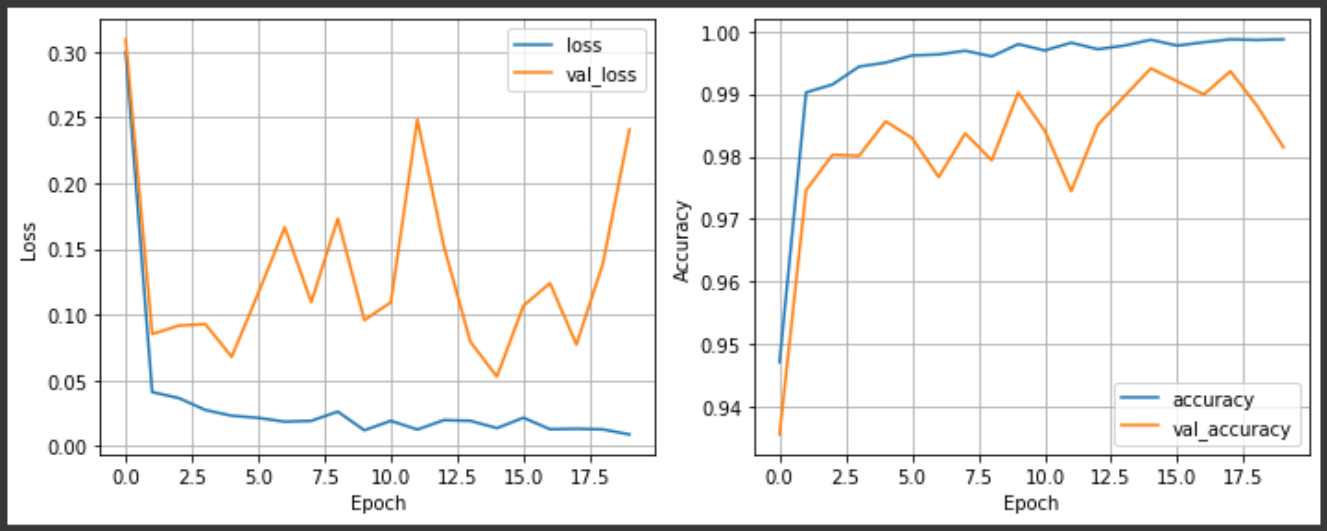
每次大概都跑 70 幾秒 精準度最後都在 0.9 以上

```
Epoch 17/20
1905/1905 [==============================] - 73s 38ms/step - loss: 0.0129 - accuracy: 0.9982 - val_loss: 0.1240 - val_accuracy: 0.9899
Epoch 18/20
1905/1905 [==============================] - 73s 38ms/step - loss: 0.0132 - accuracy: 0.9987 - val_loss: 0.0773 - val_accuracy: 0.9936
Epoch 19/20
1905/1905 [==============================] - 73s 38ms/step - loss: 0.0127 - accuracy: 0.9986 - val_loss: 0.1392 - val_accuracy: 0.9881
Epoch 20/20
1905/1905 [==============================] - 73s 38ms/step - loss: 0.0089 - accuracy: 0.9987 - val_loss: 0.2411 - val_accuracy: 0.9814
```

執行結果:

Training and Validation loss / accuracy

```
plt.tight_layout()
plt.show()
```



Evaluation

```
test_generator.reset()
loss,acc = fruit_model.evaluate(test_generator,verbose=0)
print('loss : {:.4f} , acuracy : {:.4f}'.format(loss,acc))
```

```
loss : 1.3954 , acuracy : 0.9499
```

第二題:

資料獲取如同範例程式，前處理也相同

```
test_path          =  os.path.sep.join(file.split(os.path.sep)[0:-1]+['Fruit-Images-Dataset-master','Test'])

#  定義訓練影像資料產生器
train_datagen  =  tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255,validation_split=0.1)
#  定義測試影像資料產生器
test_datagen  =  tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

#  訓練資料產生器
train_generator  =  train_datagen.flow_from_directory(training_path,target_size=(100,100),batch_size=32,class_mode='sparse',subset='training')
#  驗證資料產生器
valid_generator  =  train_datagen.flow_from_directory(training_path,target_size=(100,100),batch_size=32,class_mode='sparse',subset='validation')
```

資料進來後先進入一個 conv2D 的 layer

```
def  fruit_model(height,width,channel,classes,name):
    input  =  tf.keras.layers.Input((height,width,channel))

    conv1  =  tf.keras.layers.Conv2D(16,(3,3),padding='valid',activation='relu')(input)

    x  =  tf.keras.layers.BatchNormalization()(conv1)
    x  =  tf.keras.layers.ReLU()(x)
    x  =  tf.keras.layers.Conv2D(16,(3,3),padding  =  'same',activation='relu')(x)
```

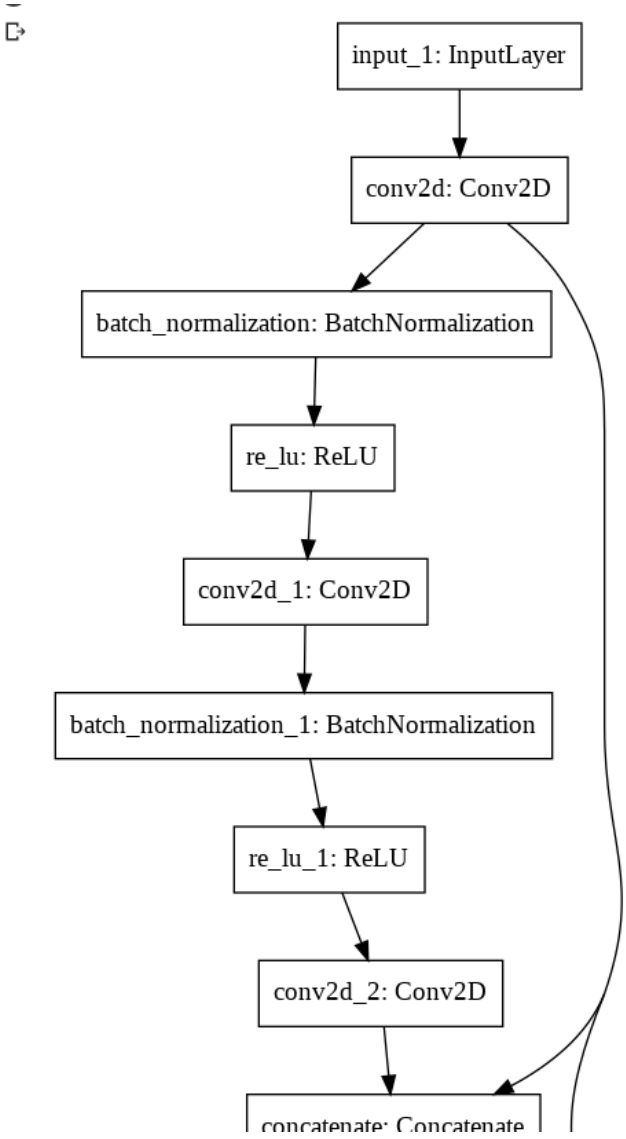
後面接一個 2-layer 的 Dense block (BN->Relu->conv->BN->relu->conv)

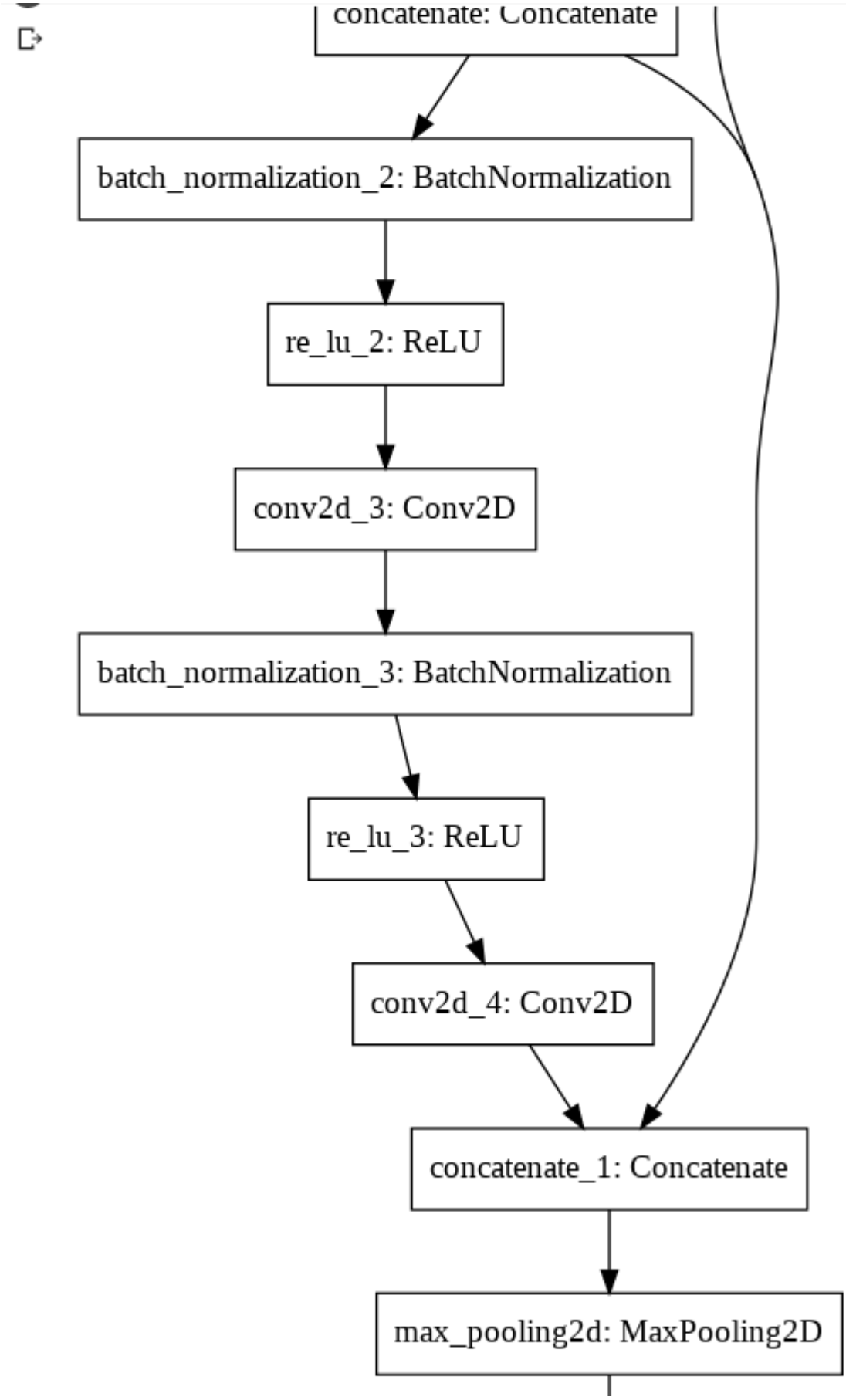
```
    x  =  tf.keras.layers.BatchNormalization()(conv1)
    x  =  tf.keras.layers.ReLU()(x)
    x  =  tf.keras.layers.Conv2D(16,(3,3),padding  =  'same',activation='relu')(x)
    x  =  tf.keras.layers.BatchNormalization()(x)
    x  =  tf.keras.layers.ReLU()(x)
    x  =  tf.keras.layers.Conv2D(16,(3,3),padding  =  'same',activation='relu')(x)
    out  =  tf.keras.layers.Concatenate()([x,conv1])


    z  =  tf.keras.layers.BatchNormalization()(out)
    z  =  tf.keras.layers.ReLU()(z)
    z  =  tf.keras.layers.Conv2D(16,(3,3),padding  =  'same',activation='relu')(z)
    z  =  tf.keras.layers.BatchNormalization()(z)
    z  =  tf.keras.layers.ReLU()(z)
    z  =  tf.keras.layers.Conv2D(16,(3,3),padding  =  'same',activation='relu')(z)
    out1  =  tf.keras.layers.Concatenate()([z,out,conv1])

    pool  =  tf.keras.layers.MaxPooling2D((3,3))(out1)
    conv2  =  tf.keras.layers.Conv2D(32,(3,3),padding='valid',activation='relu')(pool)
```

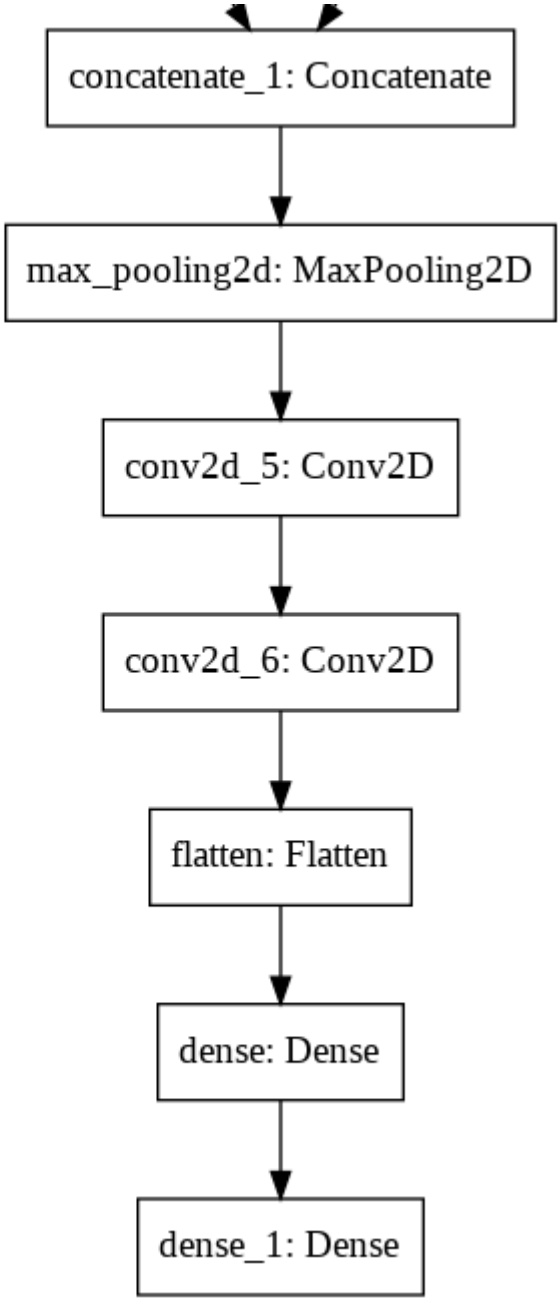
離開 Dense block 後做一個 maxpooling 接兩層 conv2D 然後 flatten
之後經過 Relu 的 Dense layer 然後 分類出 class

```
        pool  =  tf.keras.layers.MaxPooling2D((3,3))(out1)
        conv2  =  tf.keras.layers.Conv2D(32,(3,3),padding='valid',activation='relu')(pool)
        conv3  =  tf.keras.layers.Conv2D(32,(3,3),padding='valid',activation='relu')(conv2)

        flat  =  tf.keras.layers.Flatten()(conv3)
        dense  =  tf.keras.layers.Dense(classes,activation='relu')(flat)
        dense2  =  tf.keras.layers.Dense(classes,activation='softmax')(dense)

        model  =  tf.keras.Model(inputs=input,outputs=dense2,name=name)
        model.summary()
        model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

        return  model
    mod  =  fruit_model(100,100,3,train_generator.num_classes,"fruit_model")
```

Plot 出來如下圖 (Summary 有點亂就不放了)




concatenate: Concatenate
batch_normalization_2: BatchNormalization
re_lu_2: ReLU
conv2d_3: Conv2D
batch_normalization_3: BatchNormalization
re_lu_3: ReLU
conv2d_4: Conv2D
concatenate_1: Concatenate
max_pooling2d: MaxPooling2D



訓練如下圖 10 個 epoch

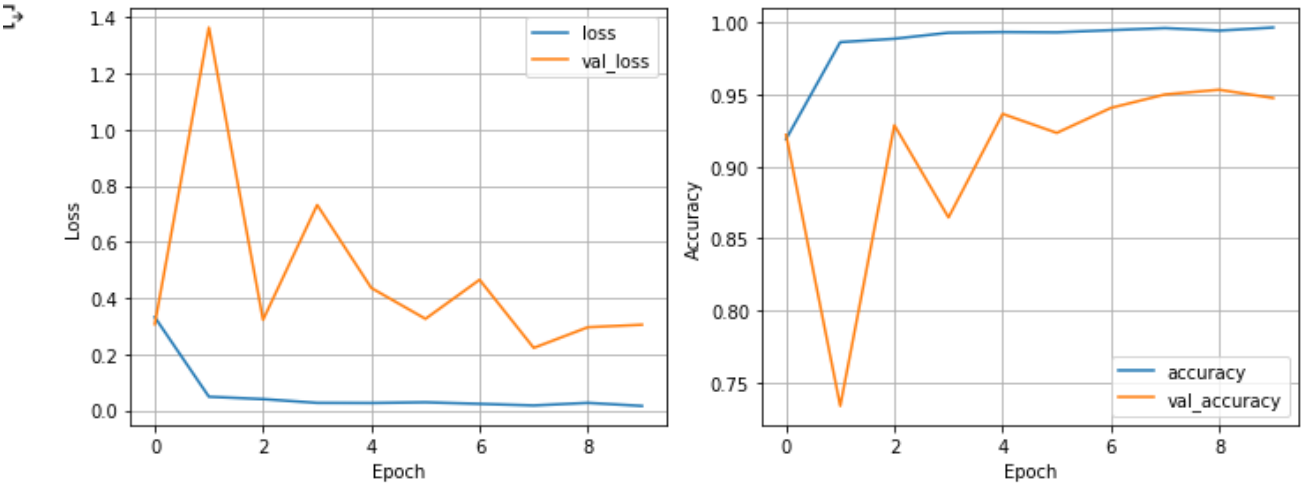
```
history = mod.fit(train_generator,epochs=10,validation_data=valid_generator)
```

```
Epoch 1/10
1905/1905 [==============================] - 113s 42ms/step - loss: 0.3320 - accuracy: 0.9191 - val_loss: 0.3072 - val_accuracy: 0.9221
Epoch 2/10
1905/1905 [==============================] - 81s 42ms/step - loss: 0.0488 - accuracy: 0.9865 - val_loss: 1.3627 - val_accuracy: 0.7337
Epoch 3/10
1905/1905 [==============================] - 81s 42ms/step - loss: 0.0404 - accuracy: 0.9888 - val_loss: 0.3224 - val_accuracy: 0.9288
Epoch 4/10
1905/1905 [==============================] - 80s 42ms/step - loss: 0.0270 - accuracy: 0.9930 - val_loss: 0.7314 - val_accuracy: 0.8646
Epoch 5/10
1905/1905 [==============================] - 80s 42ms/step - loss: 0.0265 - accuracy: 0.9935 - val_loss: 0.4359 - val_accuracy: 0.9366
Epoch 6/10
1905/1905 [==============================] - 80s 42ms/step - loss: 0.0290 - accuracy: 0.9933 - val_loss: 0.3260 - val_accuracy: 0.9234
Epoch 7/10
1905/1905 [==============================] - 80s 42ms/step - loss: 0.0234 - accuracy: 0.9948 - val_loss: 0.4651 - val_accuracy: 0.9408
Epoch 8/10
1905/1905 [==============================] - 80s 42ms/step - loss: 0.0178 - accuracy: 0.9962 - val_loss: 0.2226 - val_accuracy: 0.9501
Epoch 9/10
1905/1905 [==============================] - 80s 42ms/step - loss: 0.0269 - accuracy: 0.9945 - val_loss: 0.2962 - val_accuracy: 0.9534
Epoch 10/10
1905/1905 [==============================] - 80s 42ms/step - loss: 0.0161 - accuracy: 0.9966 - val_loss: 0.3051 - val_accuracy: 0.9476
```

執行結果:

Training and Validation loss / accuracy

```
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



Evaluation

```
test_generator.reset()
loss,acc = mod.evaluate(test_generator,verbose=0)
print('loss : {:.4f} , acuracy : {:.4f}'.format(loss,acc))
```

```
loss : 0.5867 , acuracy : 0.9258
```

結論

寫到 GPU 被強制關起來只好換一個帳號跑，不然一個 epoch 要跑 1 個小時多。做 CNN 時還算蠻幸運的，沒碰到什麼問題。一開始抓 test generator 的時候一直 find 0 photos 最後發現少下一個參數，折騰了好久。

參考文獻

作業範例程式
上課簡報