透過胸腔X-光片影像進行肺炎偵測

指導教授：邱銘傳 博士

109034546
吳欣晏

01

研 究 背 景

情境說明

肺炎的初期症狀和一般流感十分相似，因此肺炎經常被誤診為只是一般感冒或流感，但是肺炎症狀通常持續的時間較長且症狀也較為嚴重，可能因錯失治療先機使病人增加死亡風險。醫師通常透過患者之症狀、胸部X光檢查、血液檢查和痰培養去確認患者是否為肺炎，而台灣X光專科醫師人數實在有限，需要判讀的X光影像又非常多，以傳統人力判讀的方式，不僅沒有效率也很容易發生錯誤。因此本研究欲透過胸腔X-光片影像進行肺炎偵測，以協助醫師增加判定的準確性以及加快確診之速度，進而爭取更多時間搶救病患。

問題描述（5W1H）

What

欲解決單獨由醫生判斷肺炎不僅耗時且可能有誤判之問題。

Who

欲透過X光片判定病患是否罹患肺炎之人。

Where

各醫院之胸腔內科。

Why

避免錯誤診斷導致病人看診時間延誤，以及減少檢驗人員之工作量使檢驗人員能更快速地做出診斷。

When

當醫生欲確認並辨別病人是否罹患肺炎時。

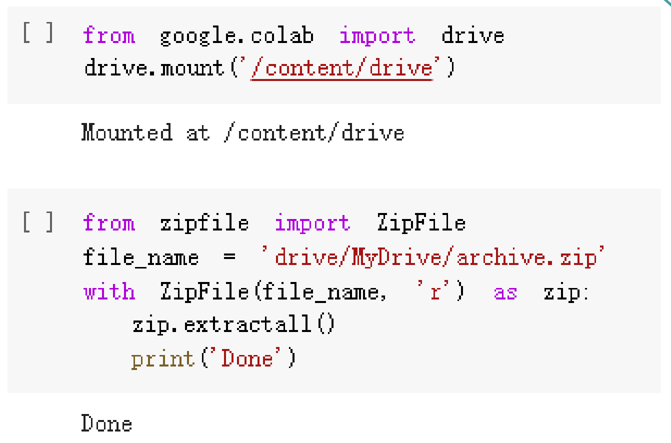
How

利用CNN建構神經網路模型，以協助醫生透過X光片進行肺炎病徵的判斷。

02

資料前處理過程

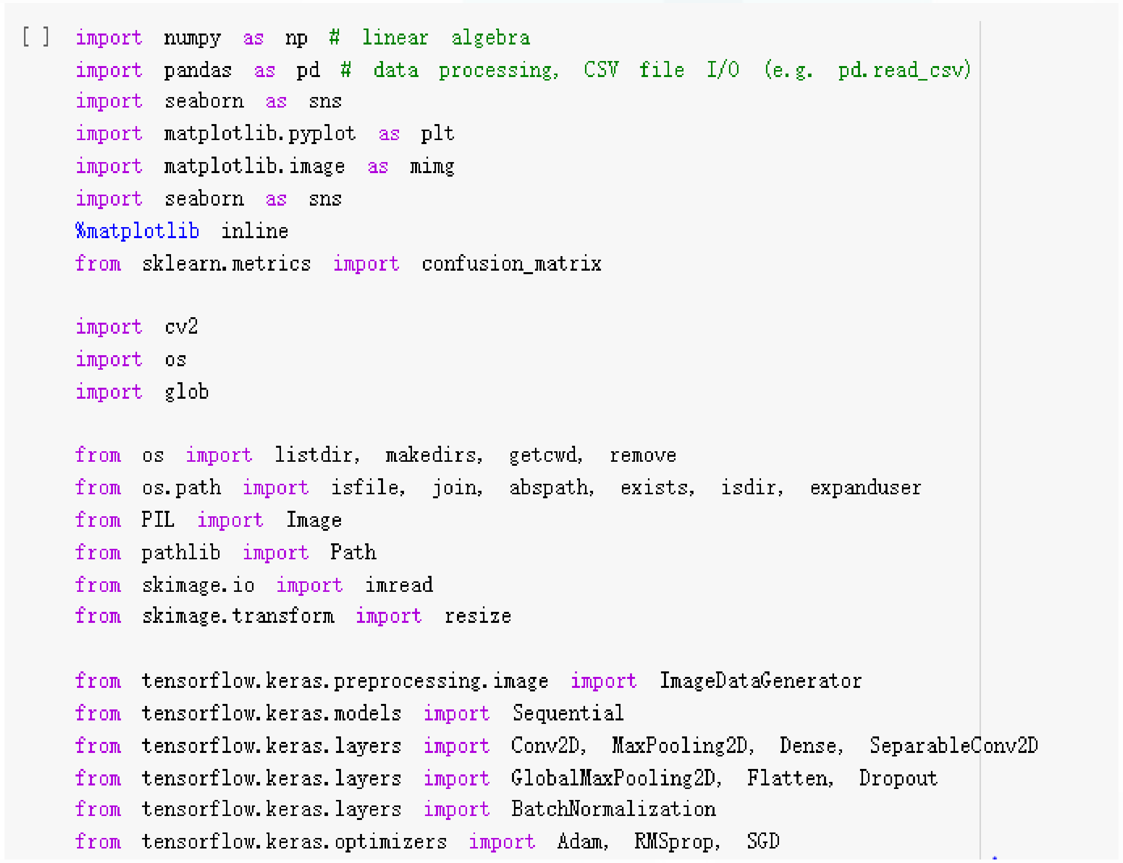
於colab導入資料集



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

    Mounted at /content/drive

[ ] from zipfile import ZipFile
    file_name = 'drive/MyDrive/archive.zip'
    with ZipFile(file_name, 'r') as zip:
        zip.extractall()
        print('Done')

    Done
```

由Kaggle公開數據集中取得胸腔X光片的圖像資料集。

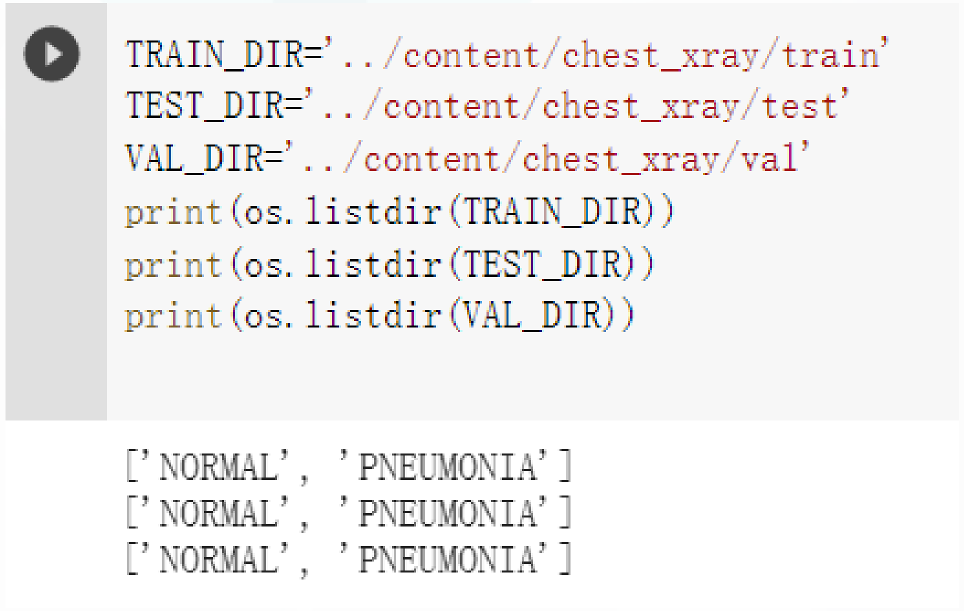
載入會使用到的套件



```
[ ] import numpy as np # linear algebra
    import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
    import seaborn as sns
    import matplotlib.pyplot as plt
    import matplotlib.image as mimg
    import seaborn as sns
    %matplotlib inline
    from sklearn.metrics import confusion_matrix

    import cv2
    import os
    import glob

    from os import listdir, makedirs, getcwd, remove
    from os.path import isfile, join, abspath, exists, isdir, expanduser
    from PIL import Image
    from pathlib import Path
    from skimage.io import imread
    from skimage.transform import resize

    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, SeparableConv2D
    from tensorflow.keras.layers import GlobalMaxPooling2D, Flatten, Dropout
    from tensorflow.keras.layers import BatchNormalization
    from tensorflow.keras.optimizers import Adam, RMSprop, SGD
```

定義X、Y資料集以及

目錄以便導入資料

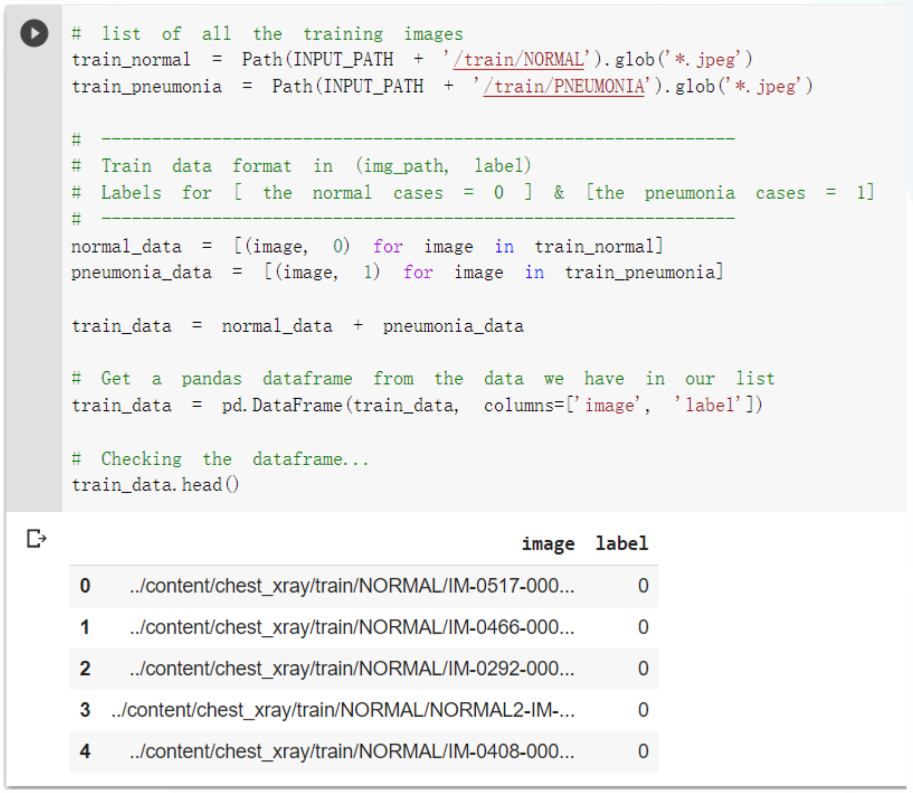


```
TRAIN_DIR='../content/chest_xray/train'
TEST_DIR='../content/chest_xray/test'
VAL_DIR='../content/chest_xray/val'
print(os.listdir(TRAIN_DIR))
print(os.listdir(TEST_DIR))
print(os.listdir(VAL_DIR))
```

```
['NORMAL', 'PNEUMONIA']
['NORMAL', 'PNEUMONIA']
['NORMAL', 'PNEUMONIA']
```

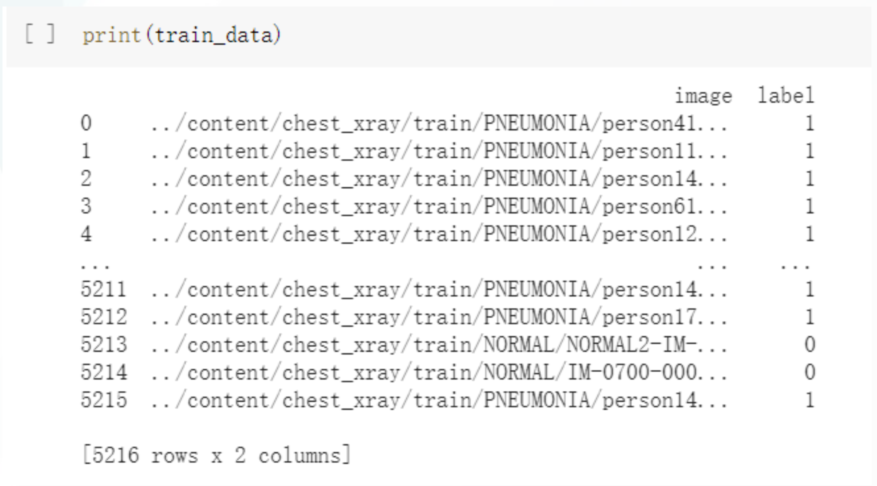
Data 處理

將Normal的圖片資料轉換為Label 0、
Pneumonia的圖片資料轉換為Label 1，
以利後續進行，並將所有的結果列出。



```
# list of all the training images
train_normal = Path(INPUT_PATH + '/train/NORMAL').glob('*.jpeg')
train_pneumonia = Path(INPUT_PATH + '/train/PNEUMONIA').glob('*.jpeg')

# ----------------------------------------------------------------
# Train data format in (img_path, label)
# Labels for [ the normal cases = 0 ] & [the pneumonia cases = 1]
# ----------------------------------------------------------------
normal_data = [(image, 0) for image in train_normal]
pneumonia_data = [(image, 1) for image in train_pneumonia]

train_data = normal_data + pneumonia_data

# Get a pandas dataframe from the data we have in our list
train_data = pd.DataFrame(train_data, columns=['image', 'label'])

# Checking the dataframe...
train_data.head()
```

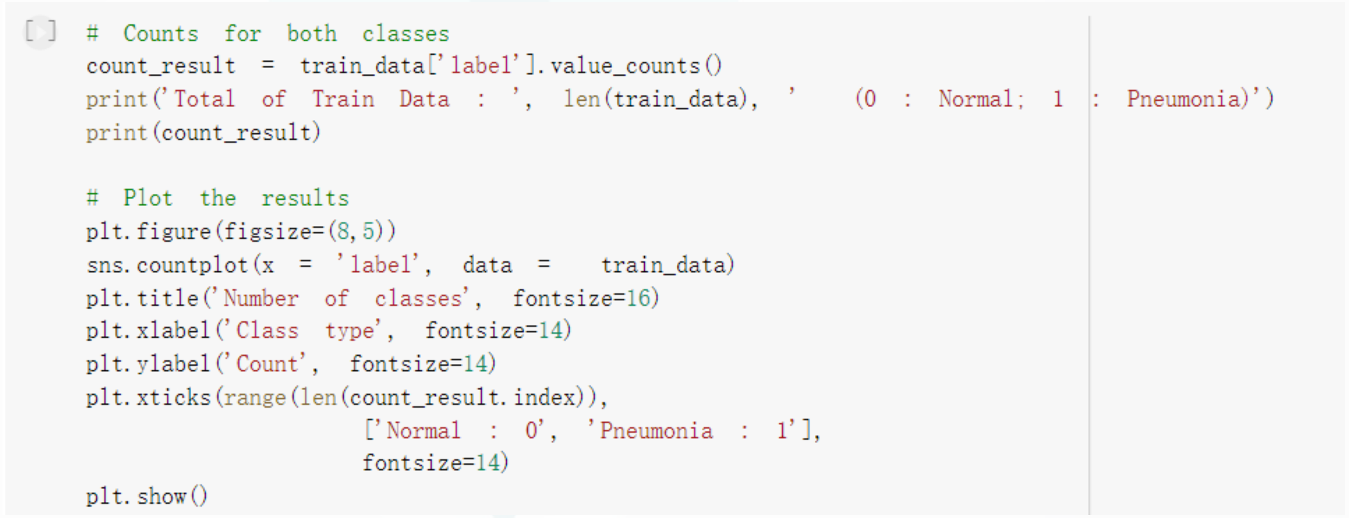
	image	label
0	../content/chest_xray/train/NORMAL/IM-0517-000...	0
1	../content/chest_xray/train/NORMAL/IM-0466-000...	0
2	../content/chest_xray/train/NORMAL/IM-0292-000...	0
3	../content/chest_xray/train/NORMAL/NORMAL2-IM-...	0
4	../content/chest_xray/train/NORMAL/IM-0408-000...	0



```
[ ] print(train_data)
```

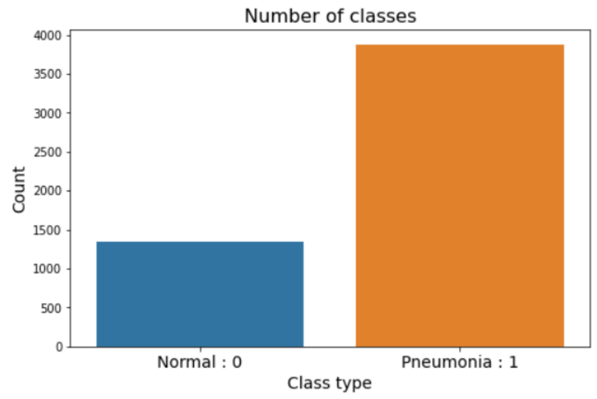
```
                                                  image  label
0     ../content/chest_xray/train/PNEUMONIA/person41...      1
1     ../content/chest_xray/train/PNEUMONIA/person11...      1
2     ../content/chest_xray/train/PNEUMONIA/person14...      1
3     ../content/chest_xray/train/PNEUMONIA/person61...      1
4     ../content/chest_xray/train/PNEUMONIA/person12...      1
...                                                 ...    ...
5211  ../content/chest_xray/train/PNEUMONIA/person14...      1
5212  ../content/chest_xray/train/PNEUMONIA/person17...      1
5213  ../content/chest_xray/train/NORMAL/NORMAL2-IM-...      0
5214  ../content/chest_xray/train/NORMAL/IM-0700-000...      0
5215  ../content/chest_xray/train/PNEUMONIA/person14...      1

[5216 rows x 2 columns]
```

查看所有Training Data 中Normal以及Pneumonia 之數量，並以圖表顯示之。

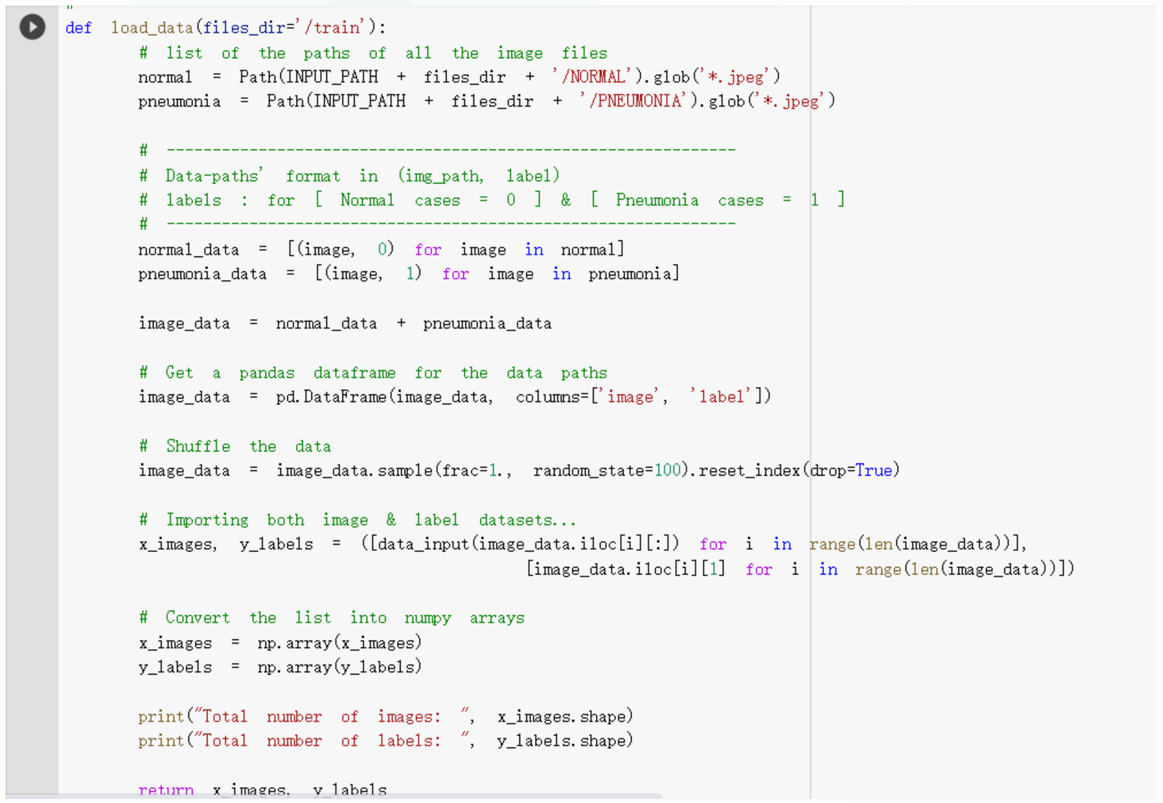


```
# Counts for both classes
count_result = train_data['label'].value_counts()
print('Total of Train Data : ', len(train_data), '    (0 : Normal; 1 : Pneumonia)')
print(count_result)

# Plot the results
plt.figure(figsize=(8,5))
sns.countplot(x = 'label', data =  train_data)
plt.title('Number of classes', fontsize=16)
plt.xlabel('Class type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(range(len(count_result.index)),
           ['Normal : 0', 'Pneumonia : 1'],
           fontsize=14)
plt.show()
```

```
Total of Train Data :  5216   (0 : Normal; 1 : Pneumonia)
1    3875
0    1341
Name: label, dtype: int64
```

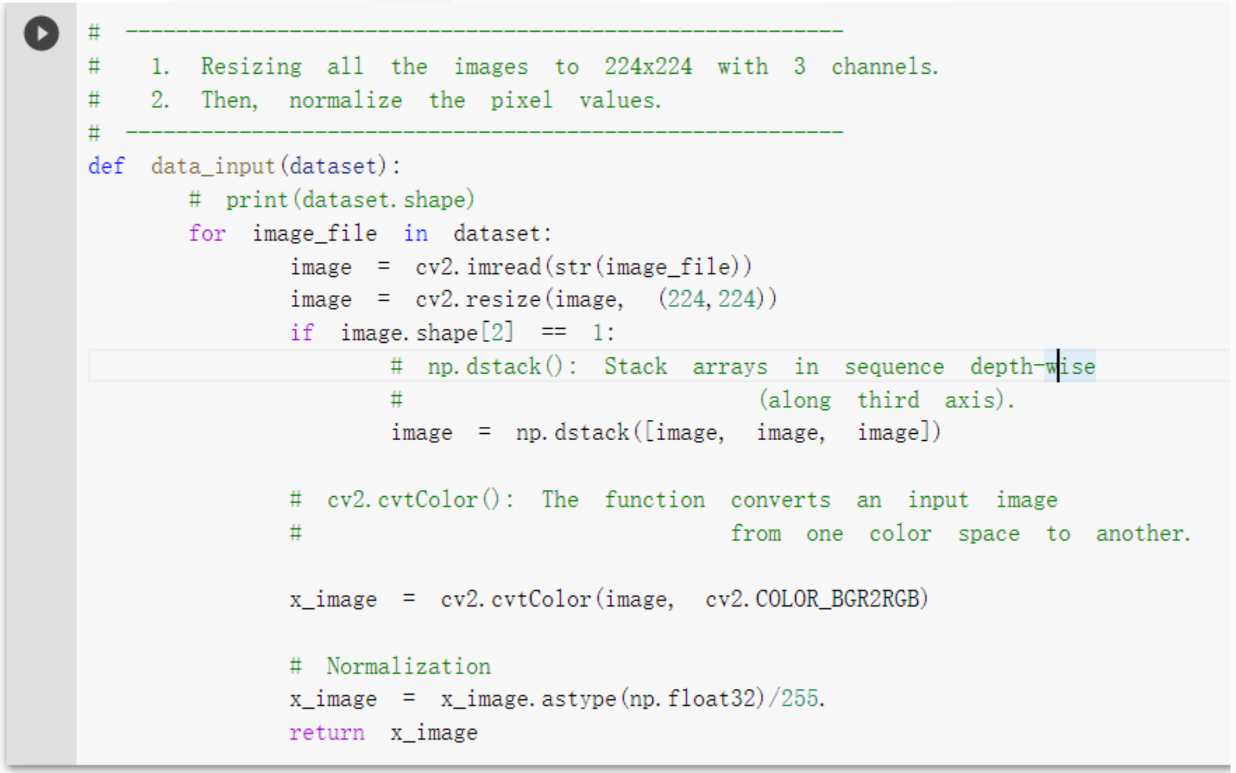


定義load_data函式：以方便將Train、Test、Validation Datasets，以Normal=0、Pneumonia=1之形式匯入。



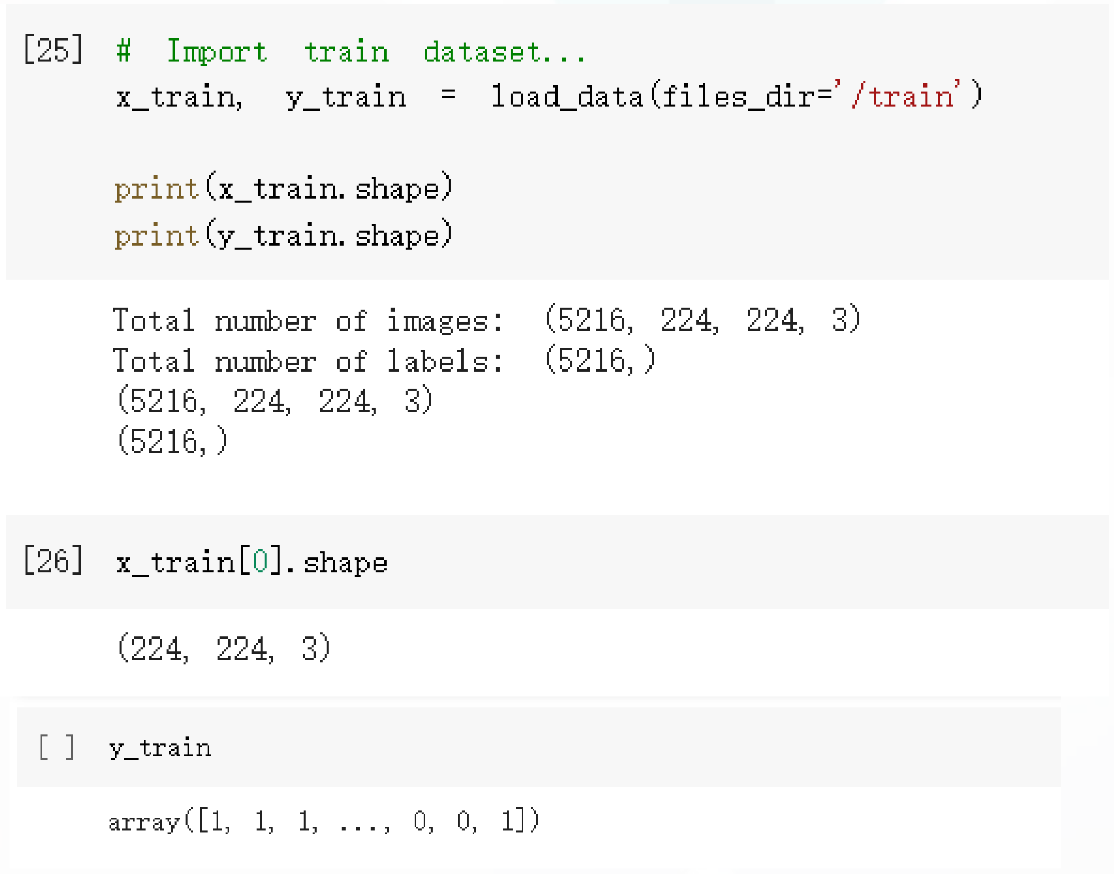
```
def  load_data(files_dir='/train'):
        #  list  of  the  paths  of  all  the  image  files
        normal  =  Path(INPUT_PATH  +  files_dir  +  '/NORMAL').glob('*.jpeg')
        pneumonia  =  Path(INPUT_PATH  +  files_dir  +  '/PNEUMONIA').glob('*.jpeg')

        #  --------------------------------------------------------------
        #  Data-paths'  format  in  (img_path,  label)
        #  labels  :  for  [  Normal  cases  =  0  ]  &  [  Pneumonia  cases  =  1  ]
        #  --------------------------------------------------------------
        normal_data  =  [(image,  0)  for  image  in  normal]
        pneumonia_data  =  [(image,  1)  for  image  in  pneumonia]

        image_data  =  normal_data  +  pneumonia_data

        #  Get  a  pandas  dataframe  for  the  data  paths
        image_data  =  pd.DataFrame(image_data,  columns=['image',  'label'])

        #  Shuffle  the  data
        image_data  =  image_data.sample(frac=1.,  random_state=100).reset_index(drop=True)

        #  Importing  both  image  &  label  datasets...
        x_images,  y_labels  =  ([data_input(image_data.iloc[i][:])  for  i  in  range(len(image_data))],
                                      [image_data.iloc[i][1]  for  i  in  range(len(image_data))])

        #  Convert  the  list  into  numpy  arrays
        x_images  =  np.array(x_images)
        y_labels  =  np.array(y_labels)

        print("Total  number  of  images:  ",  x_images.shape)
        print("Total  number  of  labels:  ",  y_labels.shape)

        return  x_images,  y_labels
```

定義data_input函式：讀入彩色之圖像，並調整圖像的大小（尺寸224x224），且標準化像素。



```
# ------------------------------------------------------------
#   1. Resizing all the images to 224x224 with 3 channels.
#   2. Then, normalize the pixel values.
# ------------------------------------------------------------
def data_input(dataset):
        # print(dataset.shape)
        for image_file in dataset:
                image = cv2.imread(str(image_file))
                image = cv2.resize(image, (224,224))
                if image.shape[2] == 1:
                        # np.dstack(): Stack arrays in sequence depth-wise
                        #                (along third axis).
                        image = np.dstack([image, image, image])

                # cv2.cvtColor(): The function converts an input image
                #                 from one color space to another.

                x_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

                # Normalization
                x_image = x_image.astype(np.float32)/255.
                return x_image
```

將所有Dataset之圖像以標準型式匯入，以Train Data為例：



```
[25] # Import train dataset...
     x_train, y_train = load_data(files_dir='/train')

     print(x_train.shape)
     print(y_train.shape)
```

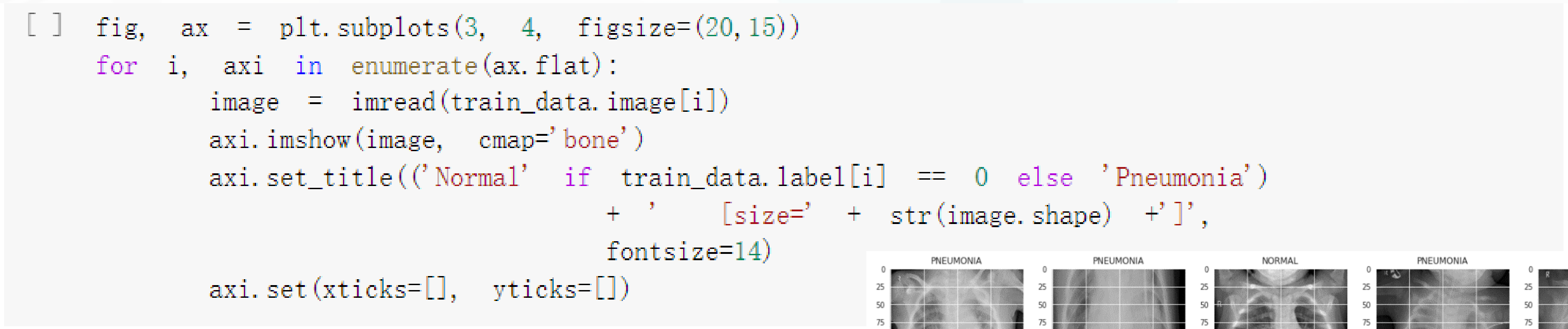
```
     Total number of images:  (5216, 224, 224, 3)
     Total number of labels:  (5216,)
     (5216, 224, 224, 3)
     (5216,)
```

```
[26] x_train[0].shape
```

```
     (224, 224, 3)
```

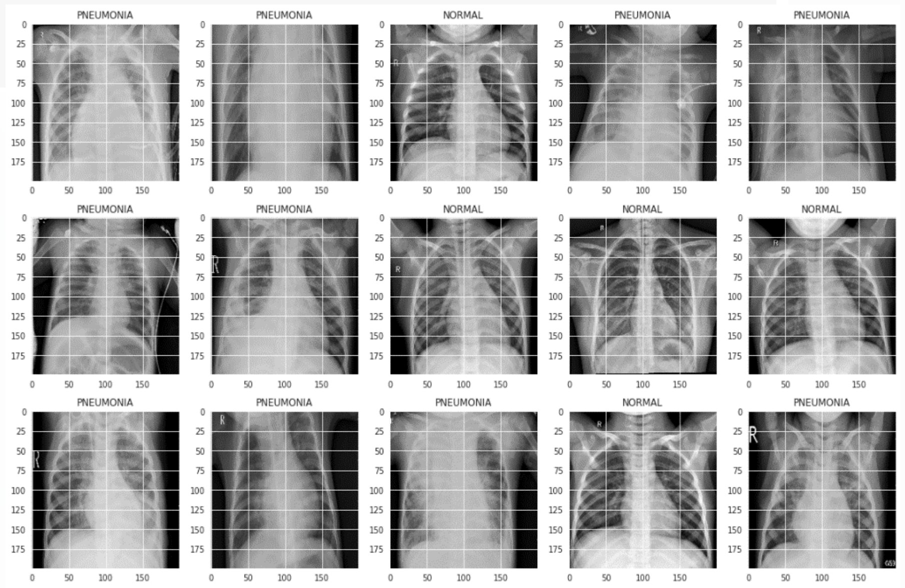
```
[ ] y_train
```

```
     array([1, 1, 1, ..., 0, 0, 1])
```

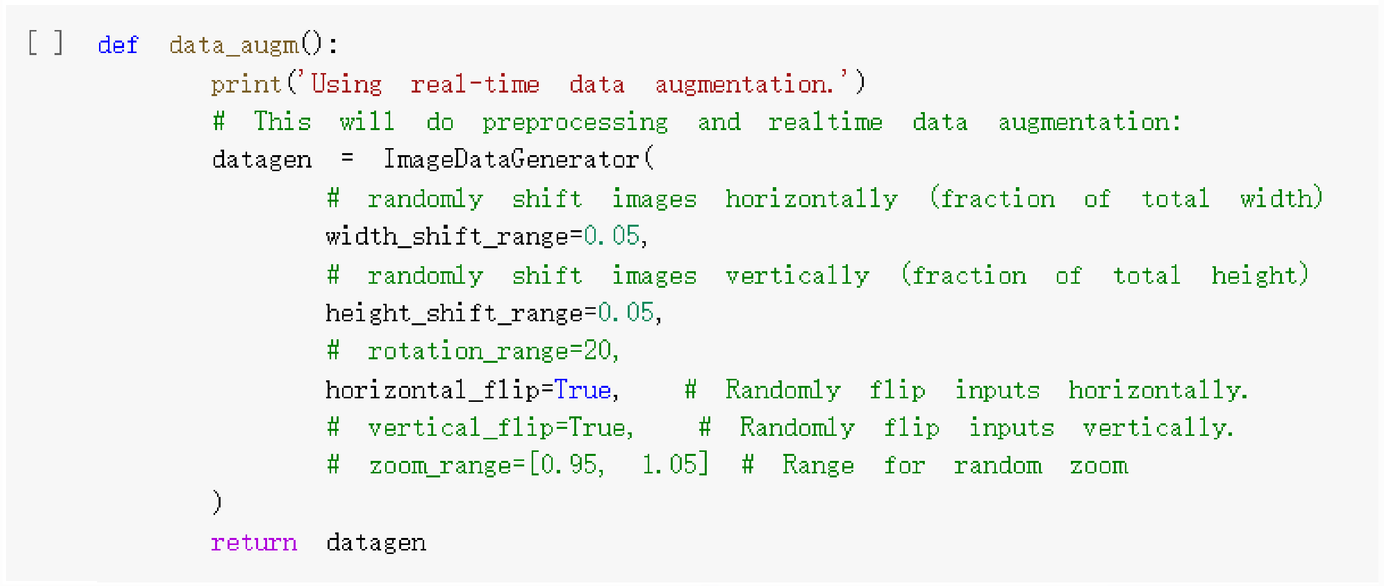


```
[ ]  fig, ax = plt.subplots(3, 4, figsize=(20,15))
     for i, axi in enumerate(ax.flat):
         image = imread(train_data.image[i])
         axi.imshow(image, cmap='bone')
         axi.set_title(('Normal' if train_data.label[i] == 0 else 'Pneumonia')
                       + '  [size=' + str(image.shape) +']',
                       fontsize=14)
         axi.set(xticks=[], yticks=[])
```

以視覺化的方式呈現



將數據擴充以防止過擬合的情況發生。由於 validation dataset 只有 16 筆影像資料，因此，直接將 training datasets (5216 images) 分割出 4200 筆的 training 資料 (80.5% 資料量)，其餘的 1016 張 X-ray 影像資料做為 validation 資料集。

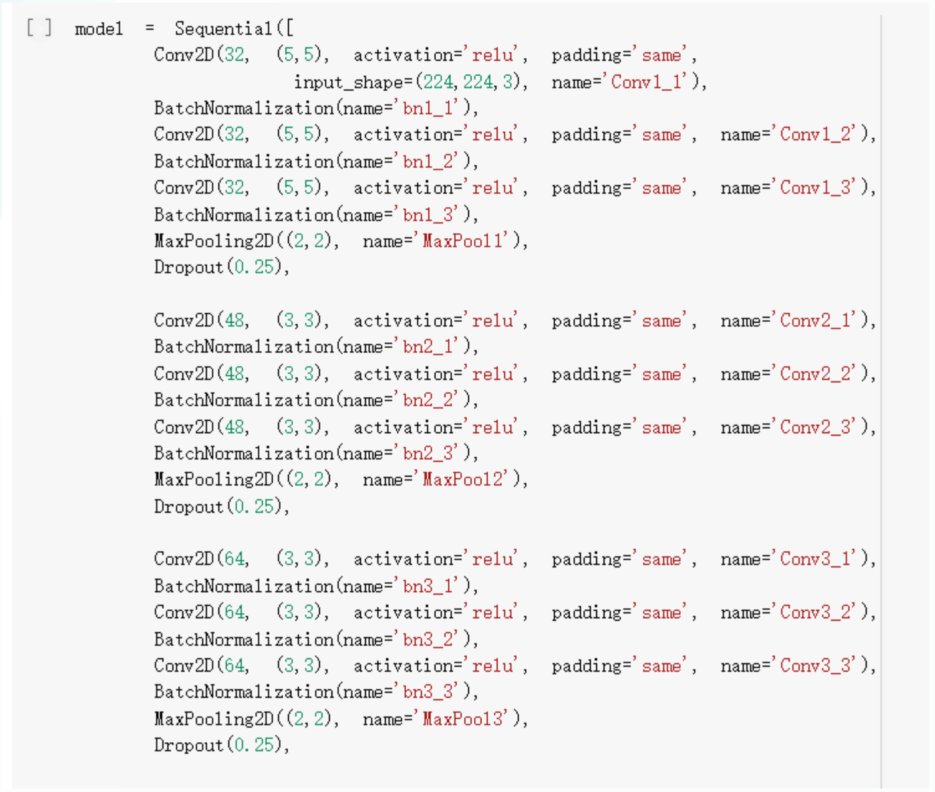


```
[ ]  def data_augm():
         print('Using real-time data augmentation.')
         # This will do preprocessing and realtime data augmentation:
         datagen = ImageDataGenerator(
                 # randomly shift images horizontally (fraction of total width)
                 width_shift_range=0.05,
                 # randomly shift images vertically (fraction of total height)
                 height_shift_range=0.05,
                 # rotation_range=20,
                 horizontal_flip=True,    # Randomly flip inputs horizontally.
                 # vertical_flip=True,    # Randomly flip inputs vertically.
                 # zoom_range=[0.95, 1.05]  # Range for random zoom
         )
         return datagen
```

03

模型建構

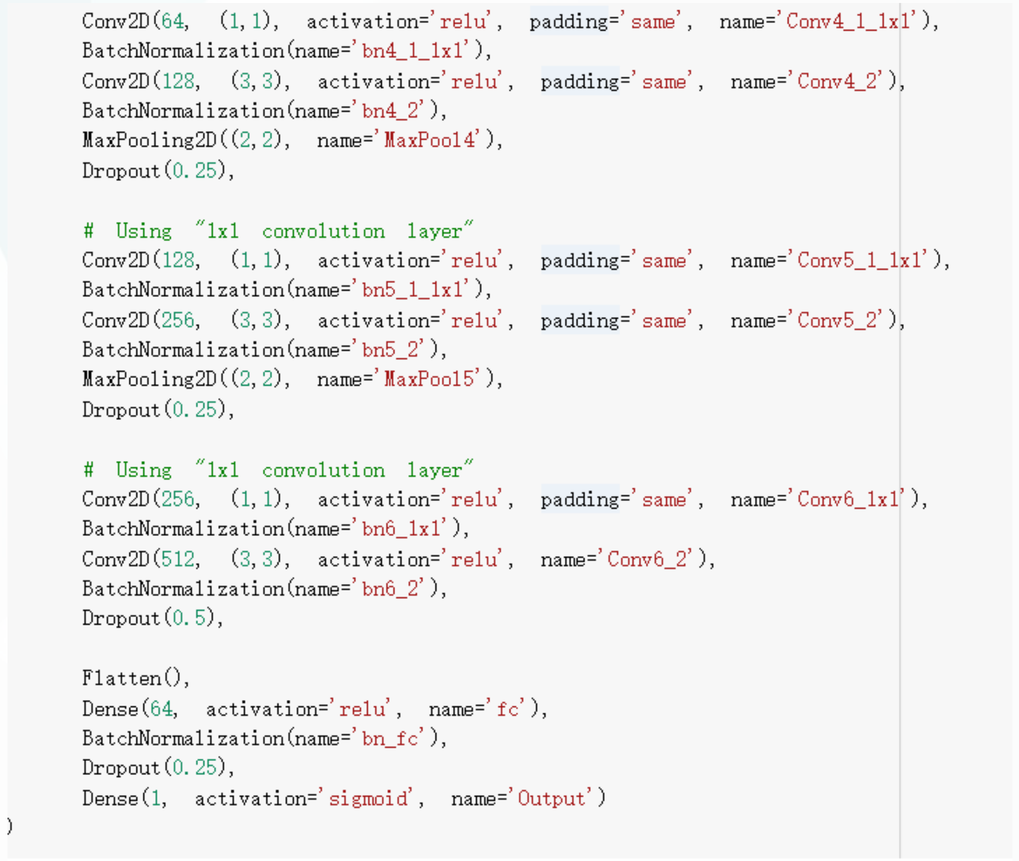
建立簡單的線性執行的模型為 Sequential
建立第一層卷積層filters 為 32，Kernal Size 為5 X 5
第二層卷積層filters 為 48 ，Kernal Size 為3 X 3
第三層卷積層filters 為 64，Kernal Size 為3 X 3
而建立池化層大小皆為 2x2，且Dropout層隨機斷開輸入神經元，用於防止過度擬合。而斷開比例:0.25，activation 皆設為 relu。



```
[ ]  model  =  Sequential([
         Conv2D(32,  (5,5),  activation='relu',  padding='same',
                   input_shape=(224,224,3),  name='Conv1_1'),
         BatchNormalization(name='bn1_1'),
         Conv2D(32,  (5,5),  activation='relu',  padding='same',  name='Conv1_2'),
         BatchNormalization(name='bn1_2'),
         Conv2D(32,  (5,5),  activation='relu',  padding='same',  name='Conv1_3'),
         BatchNormalization(name='bn1_3'),
         MaxPooling2D((2,2),  name='MaxPool1'),
         Dropout(0.25),

         Conv2D(48,  (3,3),  activation='relu',  padding='same',  name='Conv2_1'),
         BatchNormalization(name='bn2_1'),
         Conv2D(48,  (3,3),  activation='relu',  padding='same',  name='Conv2_2'),
         BatchNormalization(name='bn2_2'),
         Conv2D(48,  (3,3),  activation='relu',  padding='same',  name='Conv2_3'),
         BatchNormalization(name='bn2_3'),
         MaxPooling2D((2,2),  name='MaxPool2'),
         Dropout(0.25),

         Conv2D(64,  (3,3),  activation='relu',  padding='same',  name='Conv3_1'),
         BatchNormalization(name='bn3_1'),
         Conv2D(64,  (3,3),  activation='relu',  padding='same',  name='Conv3_2'),
         BatchNormalization(name='bn3_2'),
         Conv2D(64,  (3,3),  activation='relu',  padding='same',  name='Conv3_3'),
         BatchNormalization(name='bn3_3'),
         MaxPooling2D((2,2),  name='MaxPool3'),
         Dropout(0.25),
```

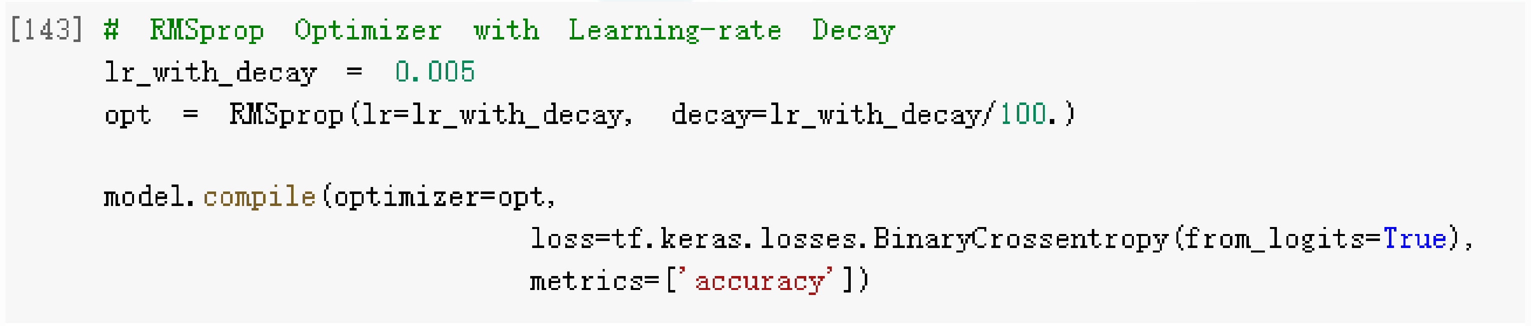
利用1x1 convolution layer 建立第四、五、六階層，1x1 convolution filter 的作用在於降低深度，但不降低原輸入維度情況下，降低計算量。

Flatten層把多維的輸入一維化，常用在從卷積層到全連接層的過渡。而建立池化層大小皆為 2x2，且Dropout層隨機斷開輸入神經元，用於防止過度擬合，斷開比例:0.25， activation 皆設為 relu。



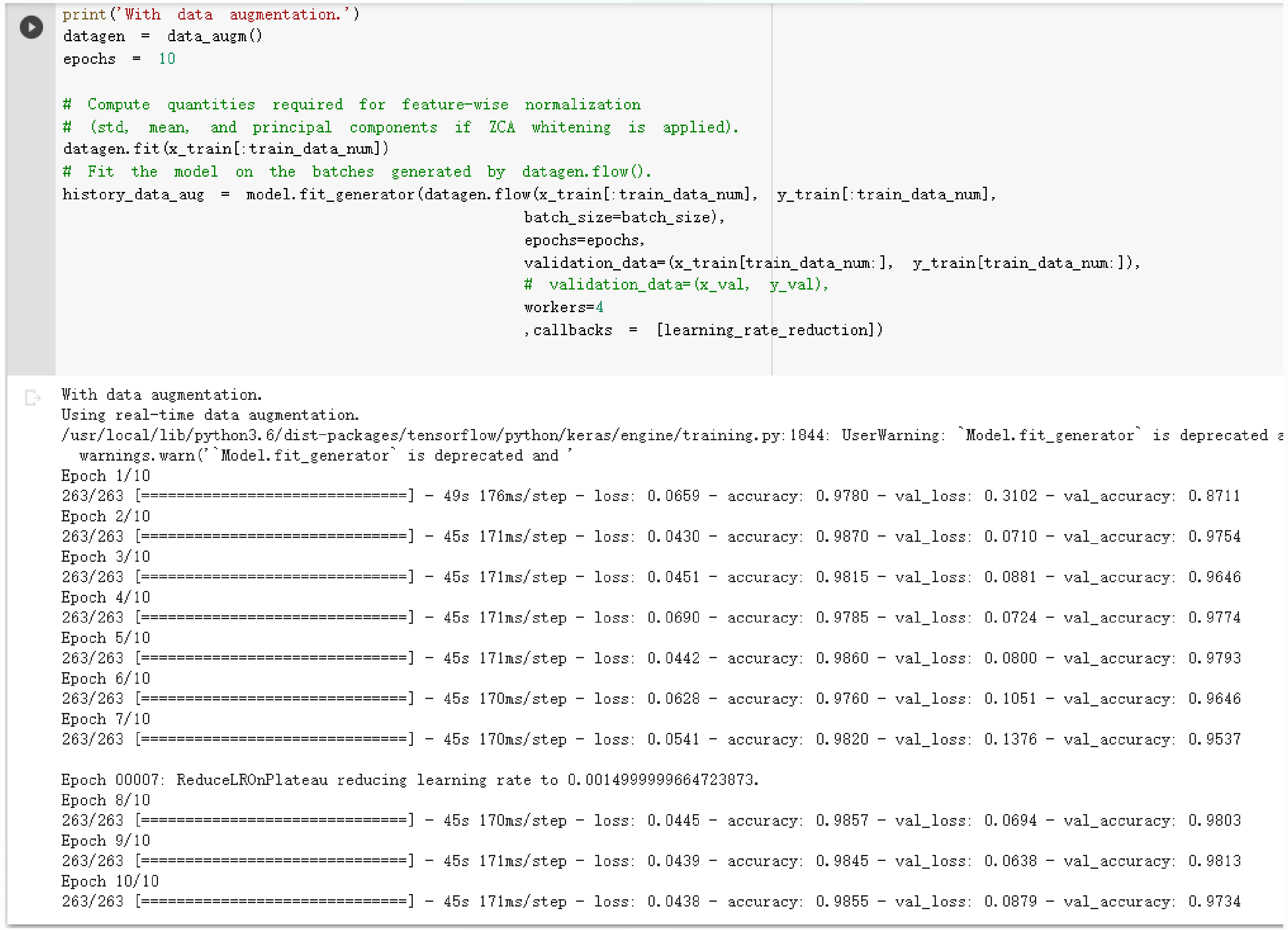
```
    Conv2D(64,  (1,1),  activation='relu',  padding='same',  name='Conv4_1_1x1'),
    BatchNormalization(name='bn4_1_1x1'),
    Conv2D(128,  (3,3),  activation='relu',  padding='same',  name='Conv4_2'),
    BatchNormalization(name='bn4_2'),
    MaxPooling2D((2,2),  name='MaxPool4'),
    Dropout(0.25),

    #  Using  "1x1  convolution  layer"
    Conv2D(128,  (1,1),  activation='relu',  padding='same',  name='Conv5_1_1x1'),
    BatchNormalization(name='bn5_1_1x1'),
    Conv2D(256,  (3,3),  activation='relu',  padding='same',  name='Conv5_2'),
    BatchNormalization(name='bn5_2'),
    MaxPooling2D((2,2),  name='MaxPool5'),
    Dropout(0.25),

    #  Using  "1x1  convolution  layer"
    Conv2D(256,  (1,1),  activation='relu',  padding='same',  name='Conv6_1x1'),
    BatchNormalization(name='bn6_1x1'),
    Conv2D(512,  (3,3),  activation='relu',  name='Conv6_2'),
    BatchNormalization(name='bn6_2'),
    Dropout(0.5),

    Flatten(),
    Dense(64,  activation='relu',  name='fc'),
    BatchNormalization(name='bn_fc'),
    Dropout(0.25),
    Dense(1,  activation='sigmoid',  name='Output')
)
```

設定模型參數，使用的 optimizer 為 RMSprop，學習率為0.005。



```
[143] # RMSprop Optimizer with Learning-rate Decay
      lr_with_decay = 0.005
      opt = RMSprop(lr=lr_with_decay, decay=lr_with_decay/100.)

      model.compile(optimizer=opt,
                                loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                                metrics=['accuracy'])
```

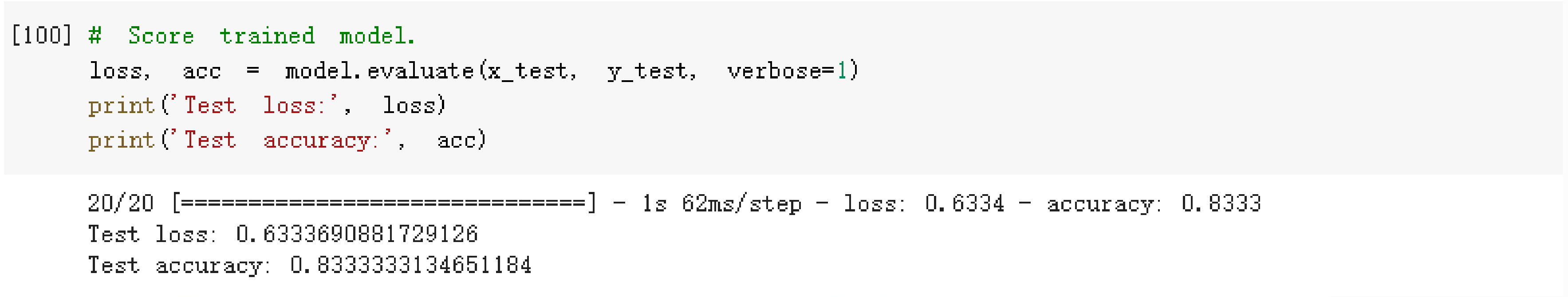
模型擬合訓練數據並驗證測試數據



```
print('With data augmentation.')
datagen = data_augm()
epochs = 10

# Compute quantities required for feature-wise normalization
# (std, mean, and principal components if ZCA whitening is applied).
datagen.fit(x_train[:train_data_num])
# Fit the model on the batches generated by datagen.flow().
history_data_aug = model.fit_generator(datagen.flow(x_train[:train_data_num], y_train[:train_data_num],
                                                    batch_size=batch_size),
                                                    epochs=epochs,
                                                    validation_data=(x_train[train_data_num:], y_train[train_data_num:]),
                                                    # validation_data=(x_val, y_val),
                                                    workers=4
                                                    ,callbacks = [learning_rate_reduction])
```

```
With data augmentation.
Using real-time data augmentation.
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit_generator` is deprecated a
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/10
263/263 [==============================] - 49s 176ms/step - loss: 0.0659 - accuracy: 0.9780 - val_loss: 0.3102 - val_accuracy: 0.8711
Epoch 2/10
263/263 [==============================] - 45s 171ms/step - loss: 0.0430 - accuracy: 0.9870 - val_loss: 0.0710 - val_accuracy: 0.9754
Epoch 3/10
263/263 [==============================] - 45s 171ms/step - loss: 0.0451 - accuracy: 0.9815 - val_loss: 0.0881 - val_accuracy: 0.9646
Epoch 4/10
263/263 [==============================] - 45s 171ms/step - loss: 0.0690 - accuracy: 0.9785 - val_loss: 0.0724 - val_accuracy: 0.9774
Epoch 5/10
263/263 [==============================] - 45s 171ms/step - loss: 0.0442 - accuracy: 0.9860 - val_loss: 0.0800 - val_accuracy: 0.9793
Epoch 6/10
263/263 [==============================] - 45s 170ms/step - loss: 0.0628 - accuracy: 0.9760 - val_loss: 0.1051 - val_accuracy: 0.9646
Epoch 7/10
263/263 [==============================] - 45s 170ms/step - loss: 0.0541 - accuracy: 0.9820 - val_loss: 0.1376 - val_accuracy: 0.9537

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.0014999999664723873.
Epoch 8/10
263/263 [==============================] - 45s 170ms/step - loss: 0.0445 - accuracy: 0.9857 - val_loss: 0.0694 - val_accuracy: 0.9803
Epoch 9/10
263/263 [==============================] - 45s 171ms/step - loss: 0.0439 - accuracy: 0.9845 - val_loss: 0.0638 - val_accuracy: 0.9813
Epoch 10/10
263/263 [==============================] - 45s 171ms/step - loss: 0.0438 - accuracy: 0.9855 - val_loss: 0.0879 - val_accuracy: 0.9734
```

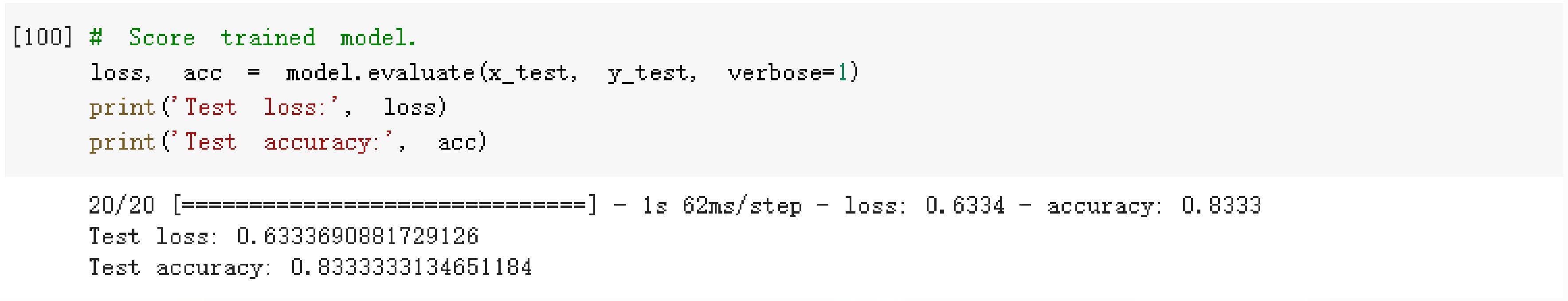
模型校度分析：取得測試集準確率，test accuracy為83.33%



```
[100] # Score trained model.
      loss, acc = model.evaluate(x_test, y_test, verbose=1)
      print('Test loss:', loss)
      print('Test accuracy:', acc)
```

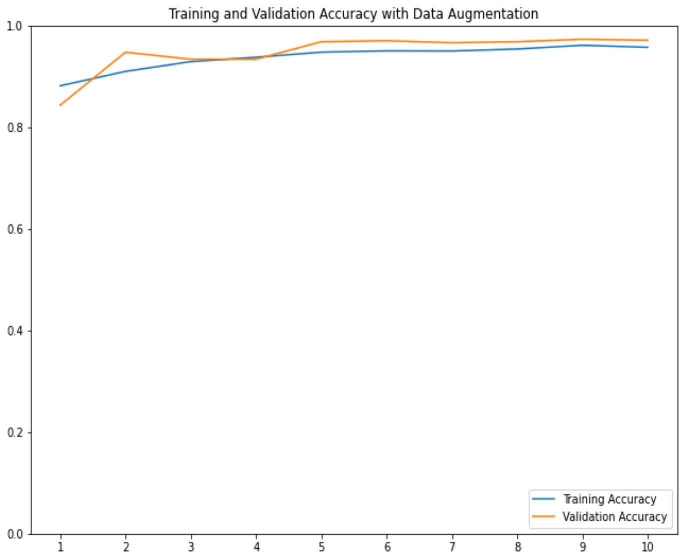
```
20/20 [==============================] - 1s 62ms/step - loss: 0.6334 - accuracy: 0.8333
Test loss: 0.6333690881729126
Test accuracy: 0.8333333134651184
```

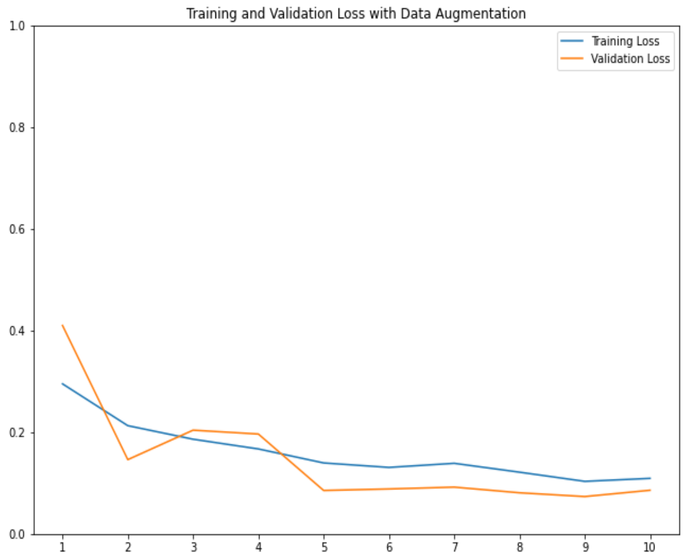
模型校度分析：取得測試集準確率，test accuracy為83.33%



```
[100] # Score trained model.
      loss, acc = model.evaluate(x_test, y_test, verbose=1)
      print('Test loss:', loss)
      print('Test accuracy:', acc)

      20/20 [==============================] - 1s 62ms/step - loss: 0.6334 - accuracy: 0.8333
      Test loss: 0.6333690881729126
      Test accuracy: 0.8333333134651184
```

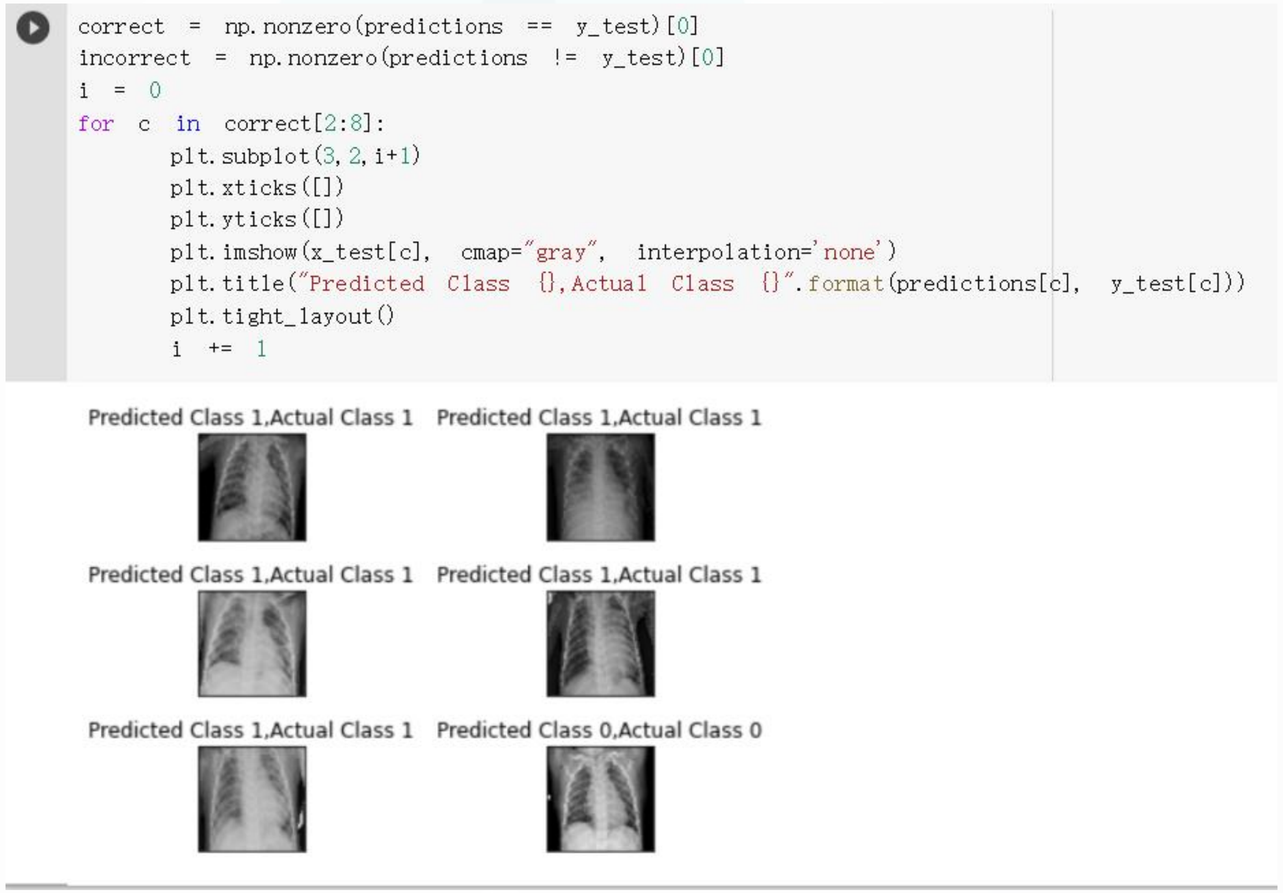
Loss、Accuracy與Epochs之關係圖





視覺化預測結果與實際結果

預測結果與實際結果相同

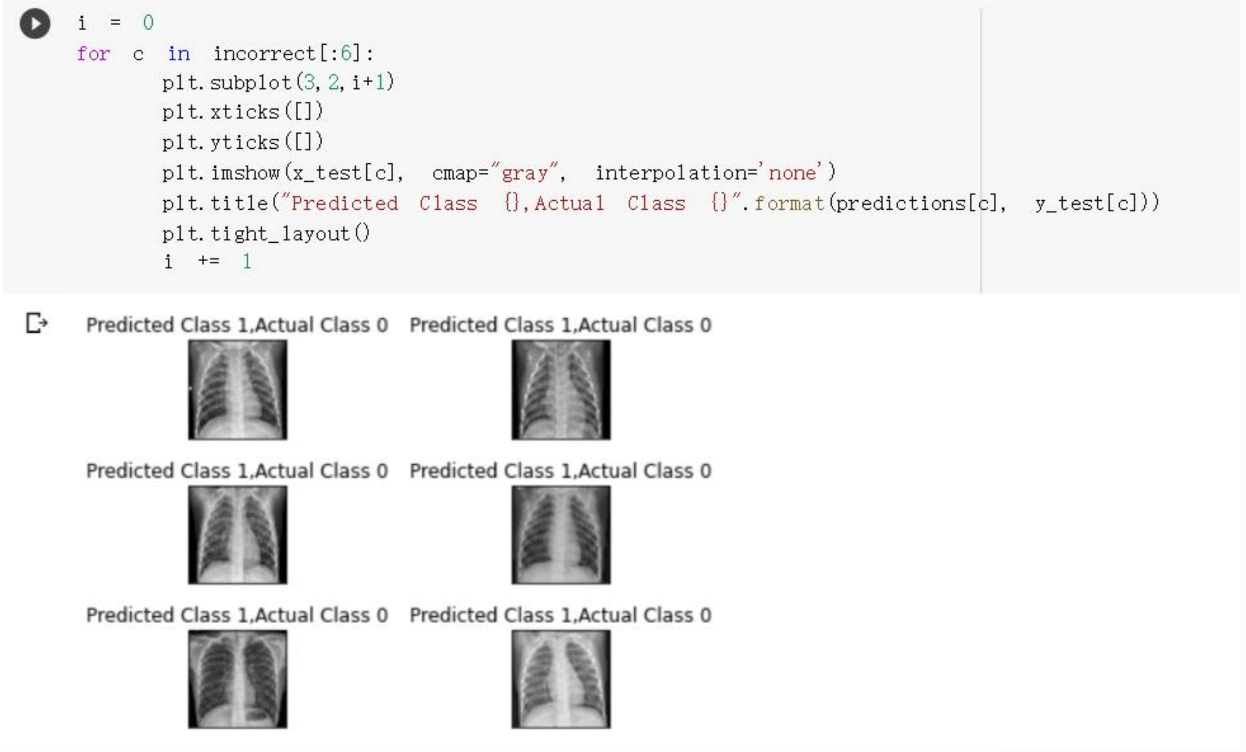


```
correct = np.nonzero(predictions == y_test)[0]
incorrect = np.nonzero(predictions != y_test)[0]
i = 0
for c in correct[2:8]:
    plt.subplot(3,2,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_test[c], cmap="gray", interpolation='none')
    plt.title("Predicted Class {},Actual Class {}".format(predictions[c], y_test[c]))
    plt.tight_layout()
    i += 1
```

```
Predicted Class 1,Actual Class 1    Predicted Class 1,Actual Class 1

Predicted Class 1,Actual Class 1    Predicted Class 1,Actual Class 1

Predicted Class 1,Actual Class 1    Predicted Class 0,Actual Class 0
```

視覺化預測結果與實際結果

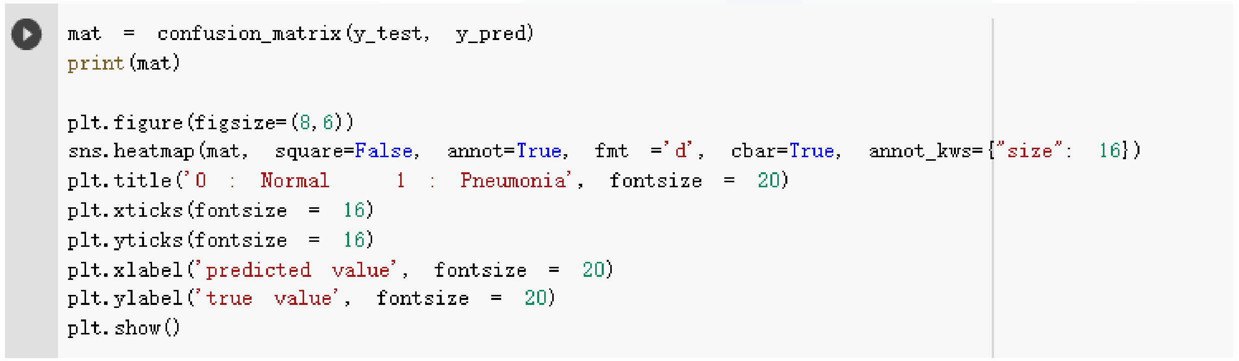
預測結果與實際結果不相同



```
i = 0
for c in incorrect[:6]:
        plt.subplot(3, 2, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(x_test[c], cmap="gray", interpolation='none')
        plt.title("Predicted Class {},Actual Class {}".format(predictions[c], y_test[c]))
        plt.tight_layout()
        i += 1
```

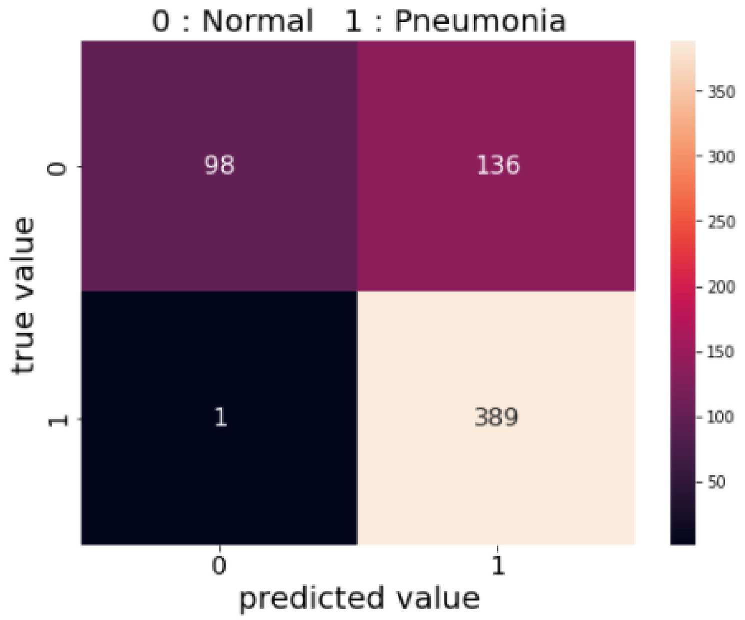
Predicted Class 1,Actual Class 0
Predicted Class 1,Actual Class 0
Predicted Class 1,Actual Class 0
Predicted Class 1,Actual Class 0
Predicted Class 1,Actual Class 0
Predicted Class 1,Actual Class 0

混淆矩陣



```
mat = confusion_matrix(y_test, y_pred)
print(mat)

plt.figure(figsize=(8,6))
sns.heatmap(mat, square=False, annot=True, fmt ='d', cbar=True, annot_kws={"size": 16})
plt.title('0 : Normal    1 : Pneumonia', fontsize = 20)
plt.xticks(fontsize = 16)
plt.yticks(fontsize = 16)
plt.xlabel('predicted value', fontsize = 20)
plt.ylabel('true value', fontsize = 20)
plt.show()
```

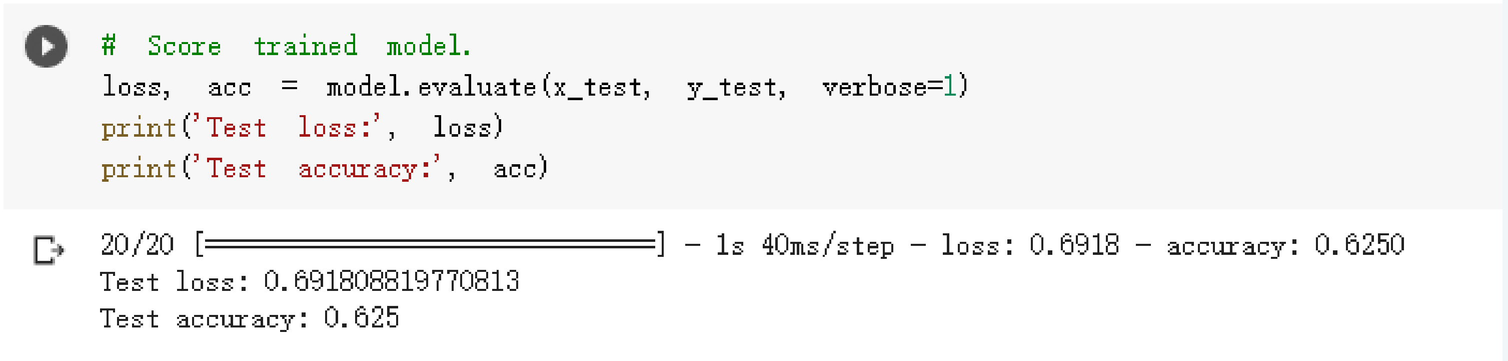
```
[[ 98 136]
 [  1 389]]
```



04

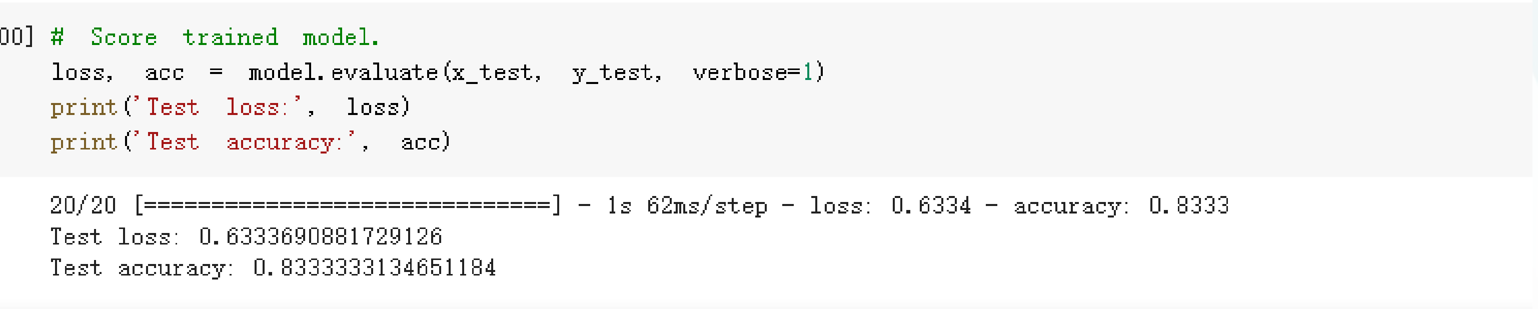
參數調整

調整Activation參數



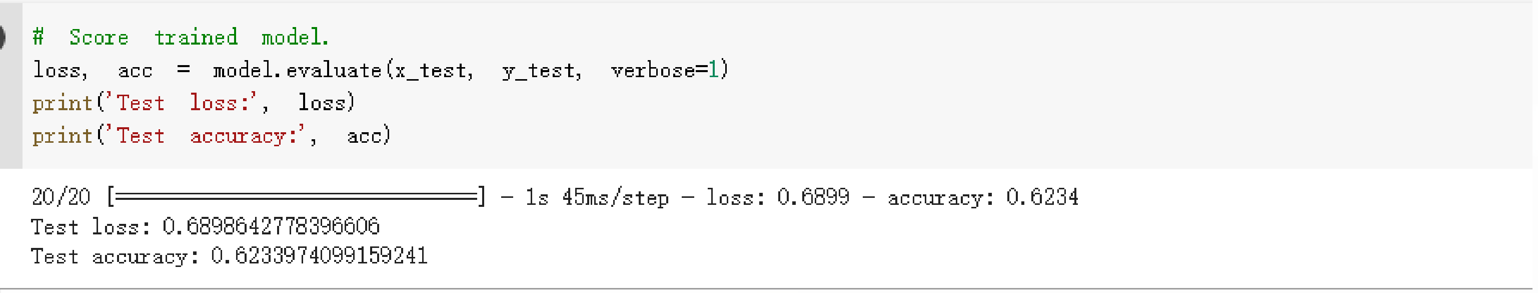
```
# Score trained model.
loss, acc = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', loss)
print('Test accuracy:', acc)
```

```
20/20 [==============================] - 1s 40ms/step - loss: 0.6918 - accuracy: 0.6250
Test loss: 0.6918088197708 13
Test accuracy: 0.625
```



```
00] # Score trained model.
     loss, acc = model.evaluate(x_test, y_test, verbose=1)
     print('Test loss:', loss)
     print('Test accuracy:', acc)
```

```
20/20 [==============================] - 1s 62ms/step - loss: 0.6334 - accuracy: 0.8333
Test loss: 0.6333690881729126
Test accuracy: 0.8333333134651184
```



```
# Score trained model.
loss, acc = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', loss)
print('Test accuracy:', acc)
```

```
20/20 [==============================] - 1s 45ms/step - loss: 0.6899 - accuracy: 0.6234
Test loss: 0.6898642778396606
Test accuracy: 0.6233974099159241
```

設定Activation設為Sigmoid

Sigmoid之test accuracy約為62.5%。

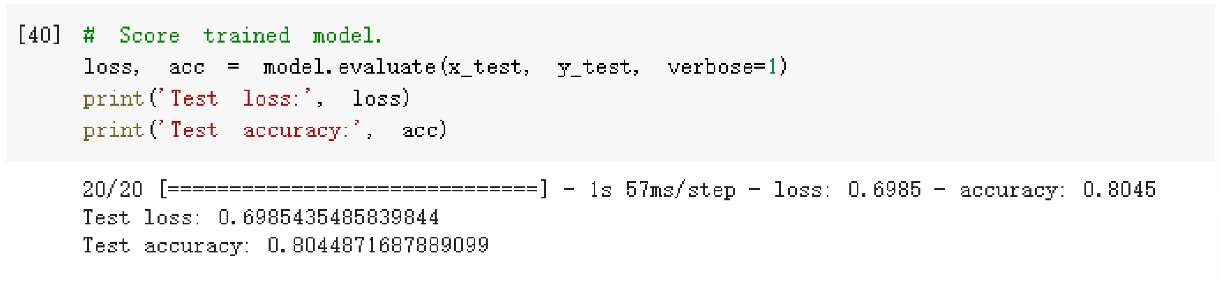
設定Activation設為Relu

Relu之test accuracy約為83.3%。

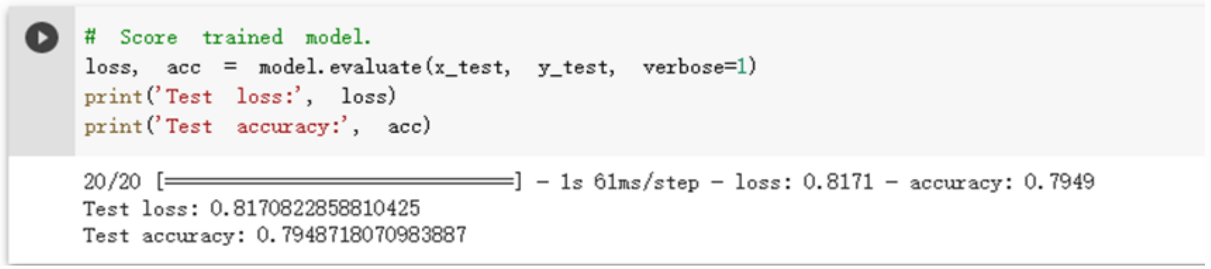
設定Activation設為tanh

tanh之test accuracy約為62.3%。

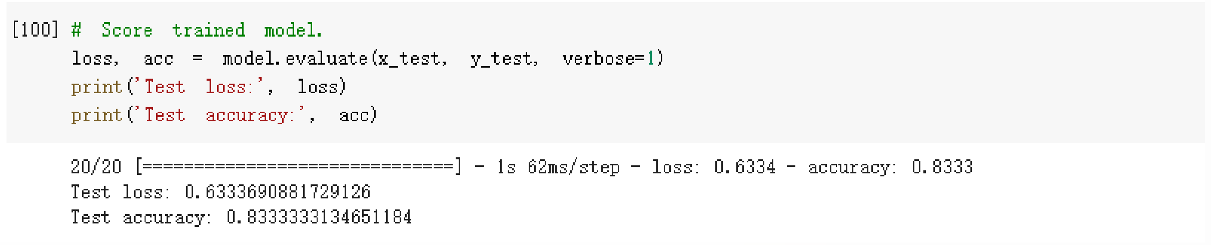
調整Filters參數



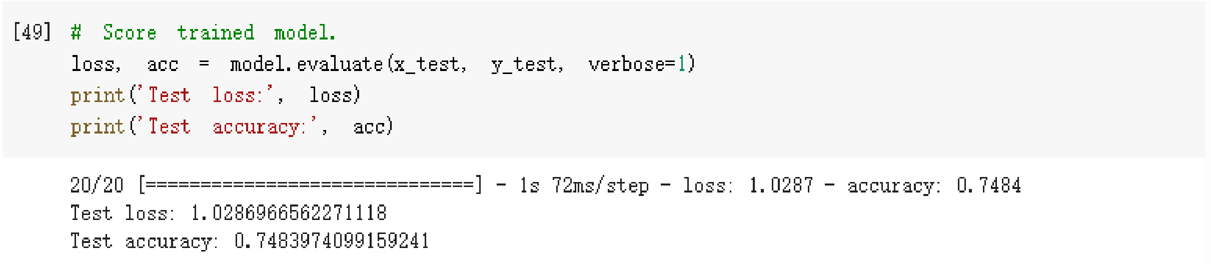
```
[40] # Score trained model.
     loss, acc = model.evaluate(x_test, y_test, verbose=1)
     print('Test loss:', loss)
     print('Test accuracy:', acc)

     20/20 [==============================] - 1s 57ms/step - loss: 0.6985 - accuracy: 0.8045
     Test loss: 0.6985435485839844
     Test accuracy: 0.8044871687889099
```



```
# Score trained model.
loss, acc = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', loss)
print('Test accuracy:', acc)

20/20 [==============================] - 1s 61ms/step - loss: 0.8171 - accuracy: 0.7949
Test loss: 0.8170822858810425
Test accuracy: 0.7948718070983887
```



```
[100] # Score trained model.
      loss, acc = model.evaluate(x_test, y_test, verbose=1)
      print('Test loss:', loss)
      print('Test accuracy:', acc)

      20/20 [==============================] - 1s 62ms/step - loss: 0.6334 - accuracy: 0.8333
      Test loss: 0.6333690881729126
      Test accuracy: 0.8333333134651184
```



```
[49] # Score trained model.
     loss, acc = model.evaluate(x_test, y_test, verbose=1)
     print('Test loss:', loss)
     print('Test accuracy:', acc)

     20/20 [==============================] - 1s 72ms/step - loss: 1.0287 - accuracy: 0.7484
     Test loss: 1.0286966562271118
     Test accuracy: 0.7483974099159241
```

設定Filter設為8

test accuracy約為80.4%。

設定Filter設為16

test accuracy約為79.4%。

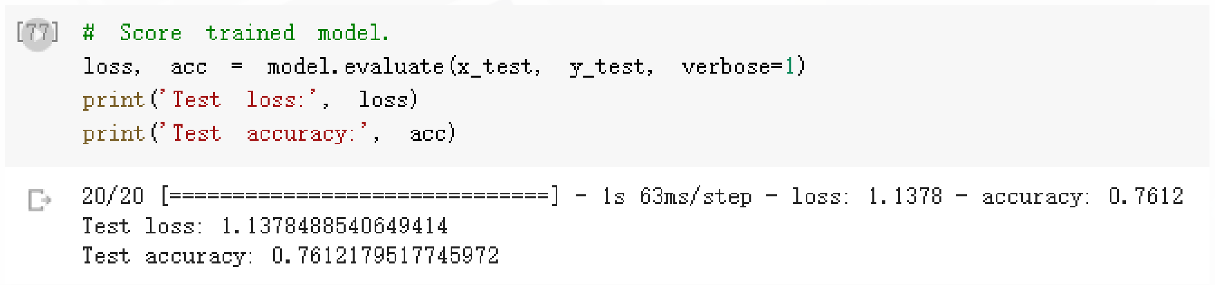
設定Filter設為32

test accuracy約為83.3%。

設定Filter設為64

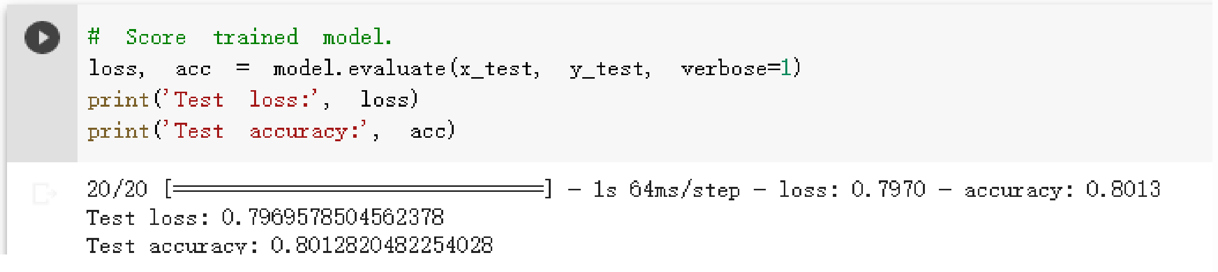
test accuracy約為74.8%。

調整Learning Rate參數



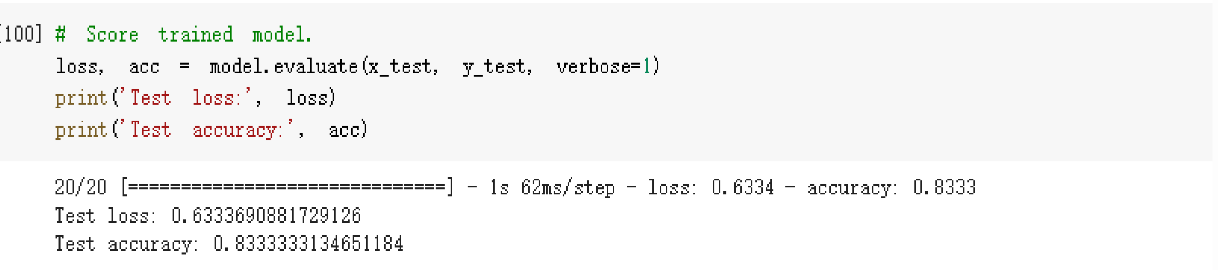
```
[77] # Score trained model.
     loss, acc = model.evaluate(x_test, y_test, verbose=1)
     print('Test loss:', loss)
     print('Test accuracy:', acc)
```

```
20/20 [==============================] - 1s 63ms/step - loss: 1.1378 - accuracy: 0.7612
Test loss: 1.1378488540649414
Test accuracy: 0.7612179517745972
```



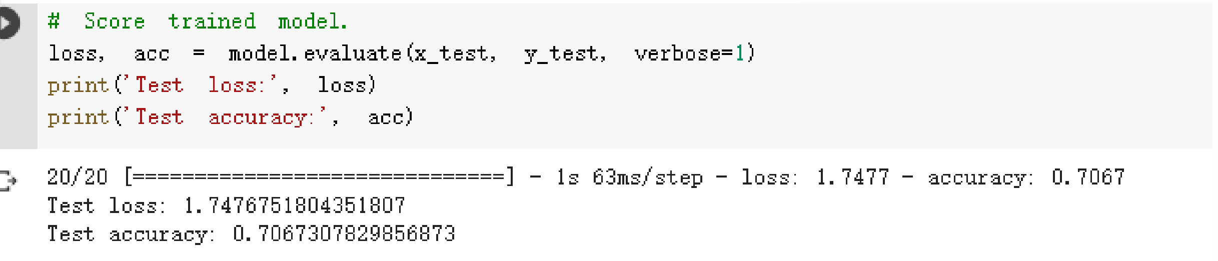
```
# Score trained model.
loss, acc = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', loss)
print('Test accuracy:', acc)
```

```
20/20 [==============================] - 1s 64ms/step - loss: 0.7970 - accuracy: 0.8013
Test loss: 0.7969578504562378
Test accuracy: 0.8012820482254028
```



```
[100] # Score trained model.
      loss, acc = model.evaluate(x_test, y_test, verbose=1)
      print('Test loss:', loss)
      print('Test accuracy:', acc)
```

```
20/20 [==============================] - 1s 62ms/step - loss: 0.6334 - accuracy: 0.8333
Test loss: 0.6333690881729126
Test accuracy: 0.8333333134651184
```



```
# Score trained model.
loss, acc = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', loss)
print('Test accuracy:', acc)
```

```
20/20 [==============================] - 1s 63ms/step - loss: 1.7477 - accuracy: 0.7067
Test loss: 1.7476751804351807
Test accuracy: 0.7067307829856873
```

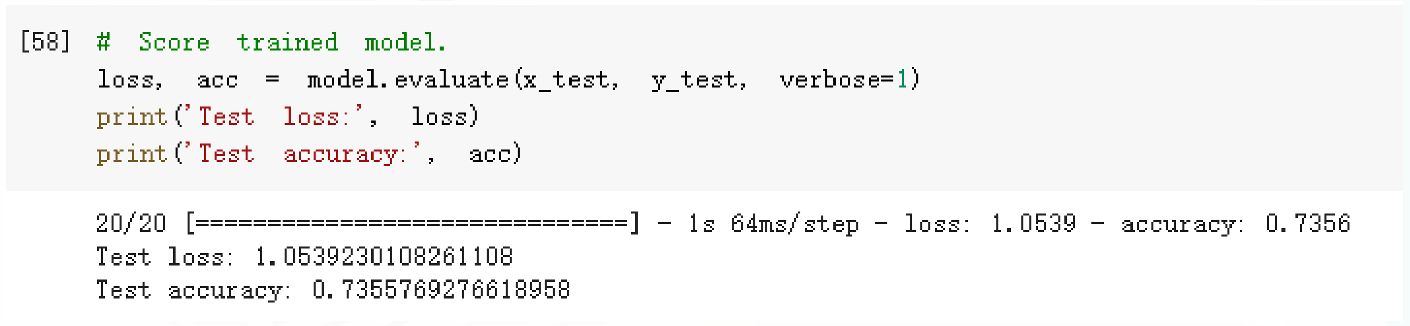
設定Learning Rate設為0.001
test accuracy約為76.1%。

設定Learning Rate設為0.003
test accuracy約為80.1%。

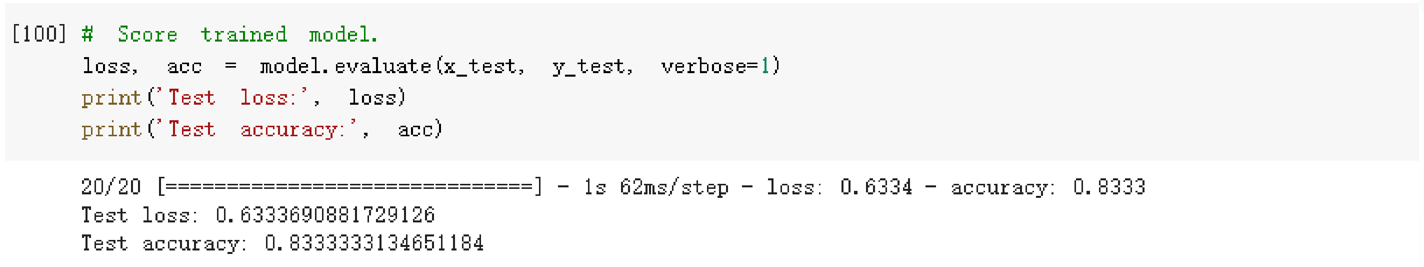
設定Learning Rate設為0.005
test accuracy約為83.3%。

設定Learning Rate設為0.008
test accuracy約為70.6%。

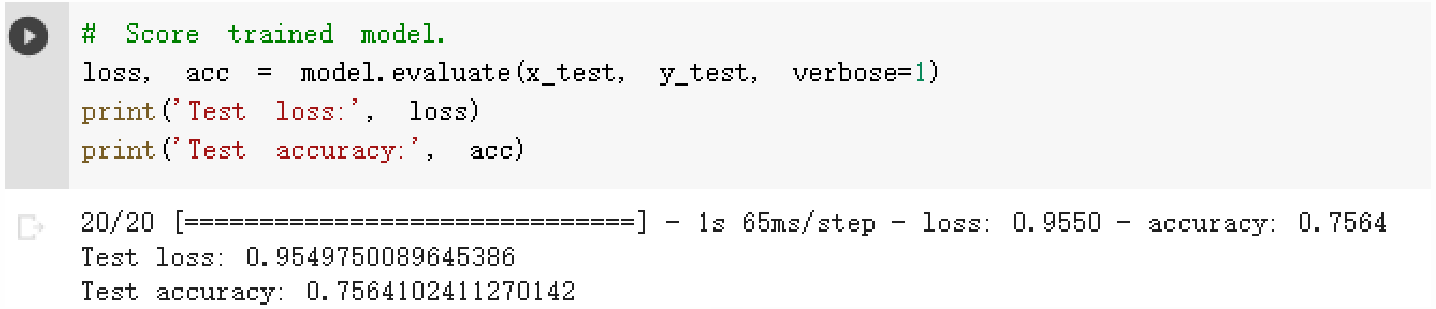
調整Optimizer參數



```
[58] # Score trained model.
     loss, acc = model.evaluate(x_test, y_test, verbose=1)
     print('Test loss:', loss)
     print('Test accuracy:', acc)

     20/20 [==============================] - 1s 64ms/step - loss: 1.0539 - accuracy: 0.7356
     Test loss: 1.0539230108261108
     Test accuracy: 0.7355769276618958
```



```
[100] # Score trained model.
      loss, acc = model.evaluate(x_test, y_test, verbose=1)
      print('Test loss:', loss)
      print('Test accuracy:', acc)

      20/20 [==============================] - 1s 62ms/step - loss: 0.6334 - accuracy: 0.8333
      Test loss: 0.6333690881729126
      Test accuracy: 0.8333333134651184
```



```
# Score trained model.
loss, acc = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', loss)
print('Test accuracy:', acc)

20/20 [==============================] - 1s 65ms/step - loss: 0.9550 - accuracy: 0.7564
Test loss: 0.9549750089645386
Test accuracy: 0.7564102411270142
```

設定 Optimizer 設為 Adam

test accuracy約為 73.6%。

設定 Optimizer 設為 RMSprop

test accurac約為 83.3%。

設定 Optimizer 設為 SGD

test accurac約為 75.6%。

各參數調整彙整表

Activation		Filters		Learning Rate		Optimizer	
參數	Test accuracy	參數	Test accuracy	參數	Test accuracy	參數	Test accuracy
Sigmoid	62.5%	8	80.4%	0.001	64%	Adam	73.5%
Relu	83.3%	16	76%	0.003	80%	RMSprop	83.3%
Tanh	62.4%	32	83.3%	0.005	83.3%	SGD	75.6%
		64	74.8%	0.008	75%		

05

結論與未來展望

結論與未來展望

透過混淆矩陣所得知的結果可知，未得病之病患X光片被判讀成有肺炎之案例較多，因以此方向進行改善，避免病患恐慌。

結論與未來展望

因兩種 X 光片的照片張數較少且不平均，而這樣會導致病徵模型學習的次數較少，導致分類預測不準確。

訓練模型的過程，每次修改只能變動一個指標，因此沒辦法將所有可能優化模型的狀況皆跑過一 次，因此這次所得到的結論只是根據所有試過的模型當中，所推得的結果，並不是所有可能組合當中最好的結果。

未來發展

因資料集有限，因此我們可透過DCGAN，DCGAN主要是利用反卷積網(Deconvolution network)反覆生成圖像，再將生成圖像放入 GAN 模型中執行，不斷訓練，最後產出多個圖像以增加資料集。

增加圖片張數較少的 X 光片數量，讓兩種分類結果都有被完整訓練

參考資料

https://docs.scipy.org/doc/numpy/reference/generated/numpy.dstack.html

https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html

https://www.kaggle.com/calexhu/data-preprocessing-for-pneumonia-detection

https://www.kaggle.com/madz2000/pneumonia-detection-using-cnn-92-6-accuracy

The End

謝謝聆聽