Covid-19 detection using CNN

代碼：https://github.com/junaidiqbalsyed/Covid\_detection\_CNN/blob/master/Covid\_detection\_using\_chest\_X\_Ray\_(Day\_2).ipynb

在這個項目中，我們將使用 CNN 進行 Covid-19 檢測，我們將使用的歷史數據是 Covid 胸部 X 片，將使用的預訓練模型是 VGG-16，收到的準確度約為90%

該項目的另一個計劃是可視化 Class Activation maps， Class Activation maps基礎上就是ml算法在最後一個卷積層學到的東西，即如果模型說胸部的圖是COVID-19，使用 GRAD-CAM函數返回熱圖，稍後將其射到輸入的 x 射線上，因此我們可以確定胸部的哪個區域有 COVID-19。

# to ignore the warnings

import warnings

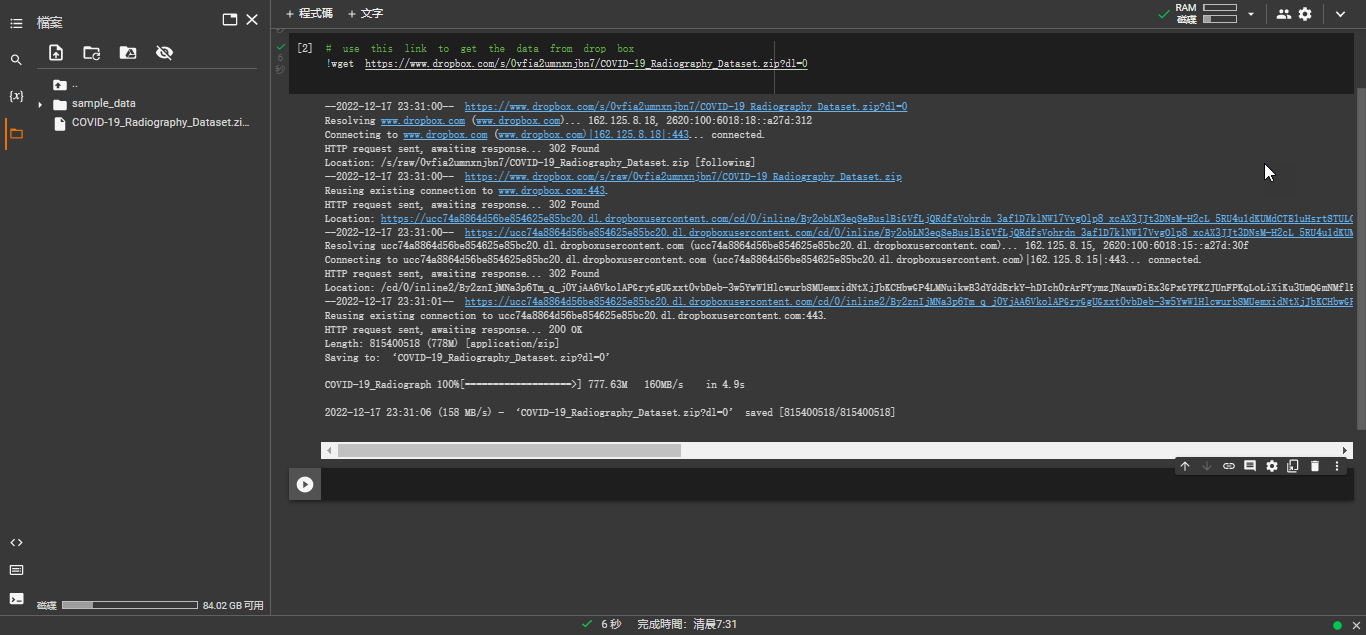
warnings.filterwarnings("ignore")

數据集取得

注意：“https://www.dropbox.com/s/0vfia2umnxnjbn7/COVID-19\_Radiography\_Dataset.zip?dl=0 ”可以使用!wget命令下載數據集。

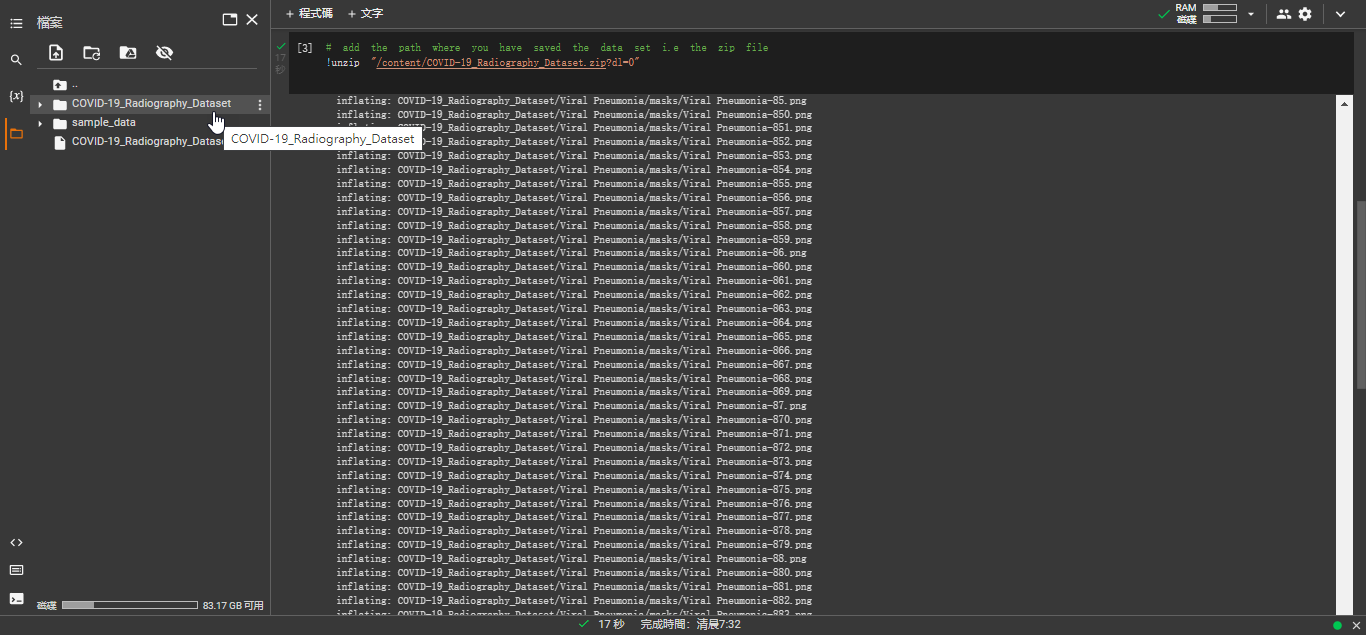
***# use this link to get the data from drop box***

!wget https://www.dropbox.com/s/0vfia2umnxnjbn7/COVID-19\_Radiography\_Dataset.zip?dl=0



***# add the path where you have saved the data set i.e the zip file***

!unzip "/content/COVID-19\_Radiography\_Dataset.zip?dl=0"



匯入使用模組

import pandas as pd

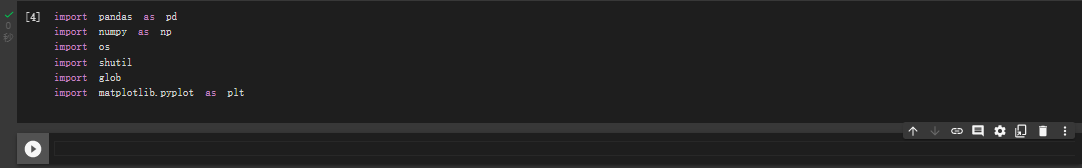
import numpy as np

import os

import shutil

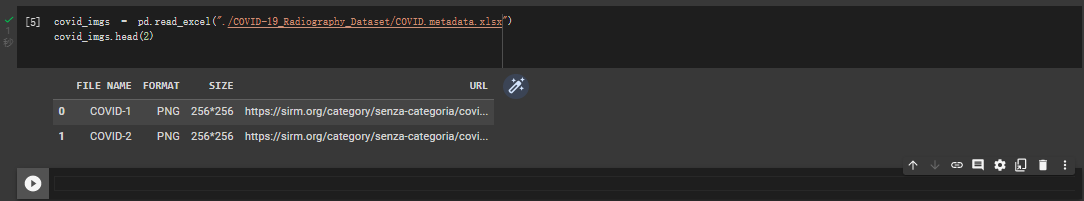
import glob

import matplotlib.pyplot as plt

瀏覽元數據**Going Through Meta Data**

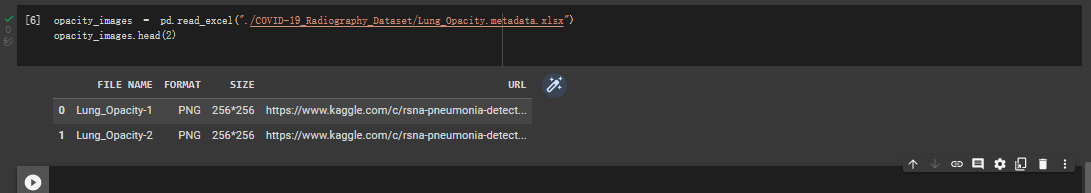
covid\_imgs = pd.read\_excel("./COVID-19\_Radiography\_Dataset/COVID.metadata.xlsx")

covid\_imgs.head(2)



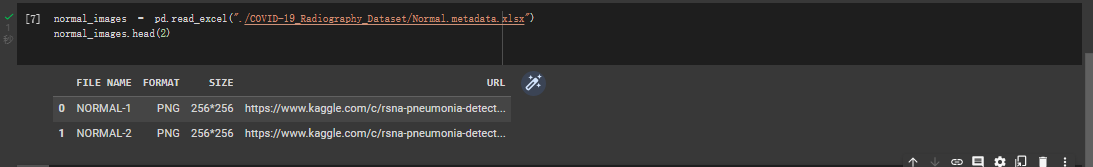
opacity\_images = pd.read\_excel("./COVID-19\_Radiography\_Dataset/Lung\_Opacity.metadata.xlsx")

opacity\_images.head(2)



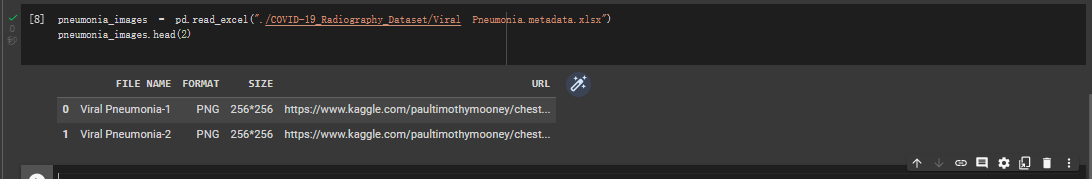
normal\_images = pd.read\_excel("./COVID-19\_Radiography\_Dataset/Normal.metadata.xlsx")

normal\_images.head(2)



pneumonia\_images = pd.read\_excel("./COVID-19\_Radiography\_Dataset/Viral Pneumonia.metadata.xlsx")

pneumonia\_images.head(2)



使用圖像**Working with images**

ROOT\_DIR = "/content/COVID-19\_Radiography\_Dataset/"

imgs = ['COVID','Lung\_Opacity','Normal','Viral Pneumonia']

NEW\_DIR = "/content/all\_images/"

# Copy all my images to a new folder i.e all\_images

if not os.path.exists(NEW\_DIR):

  os.mkdir(NEW\_DIR)

  for i in imgs:

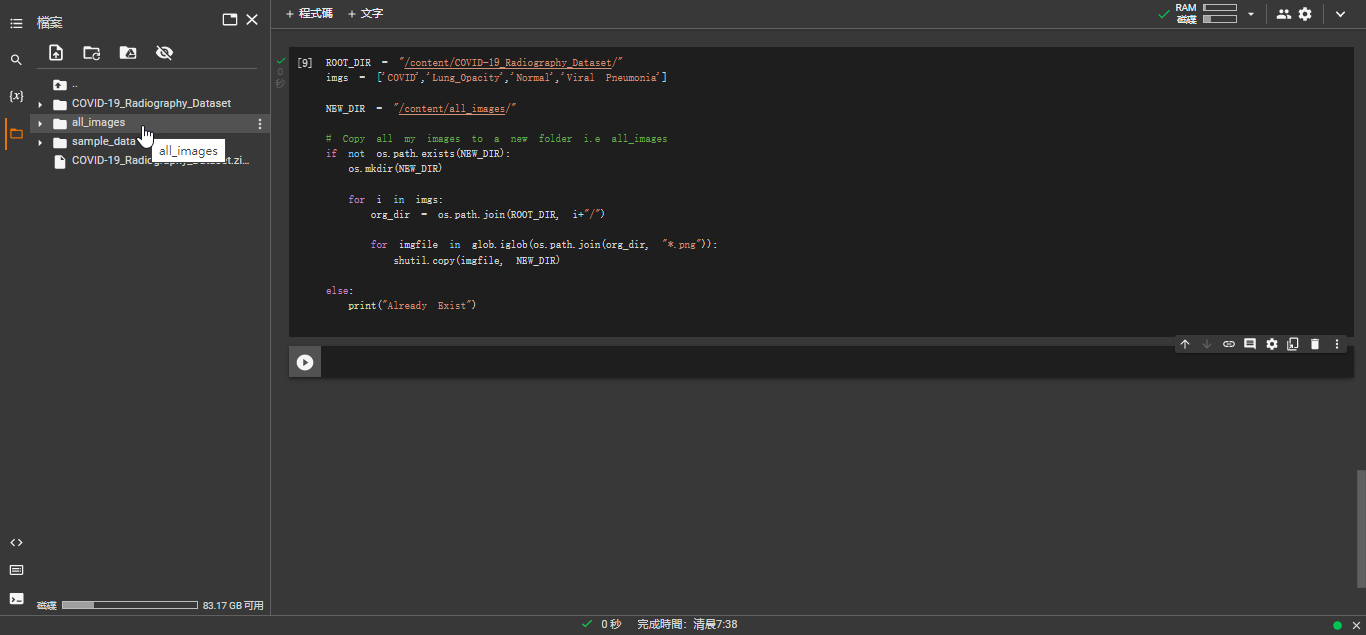
    org\_dir = os.path.join(ROOT\_DIR, i+"/")

    for imgfile in glob.iglob(os.path.join(org\_dir, "\*.png")):

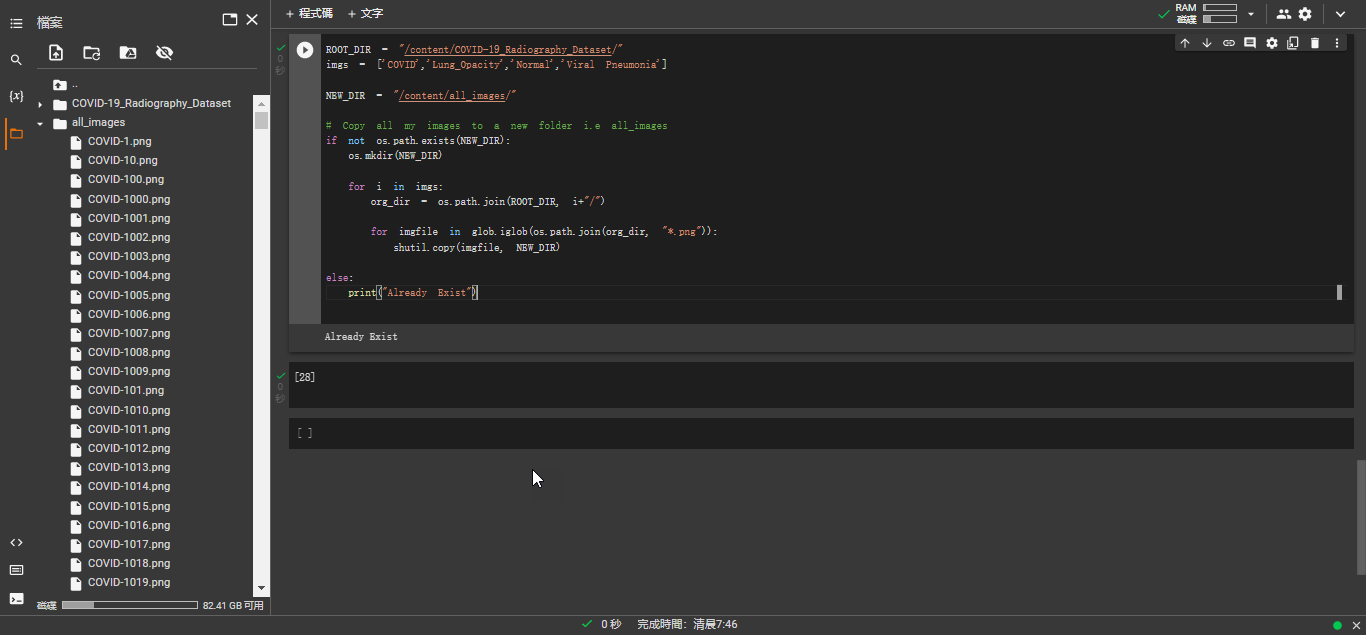
      shutil.copy(imgfile, NEW\_DIR)

else:

  print("Already Exist")



注意：此部份完成后須稍候一段時間等候all\_images目錄下有載入資料始可進行下一步，否則無法取得最新資料，如下所示：



# Now lets visualize the number of Images in each categories

**counter = {'COVID':0,'Lung\_Opacity':0,'Normal':0,'Viral Pneumonia':0}**

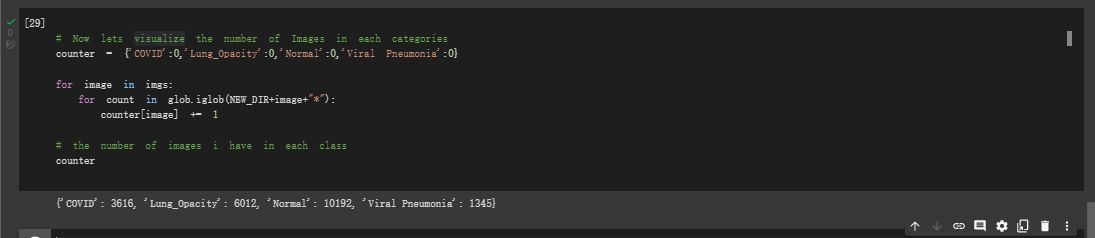
**for image in imgs:**

**for count in glob.iglob(NEW\_DIR+image+"\*"):**

**counter[image] += 1**

***# the number of images i have in each class***

**counter**

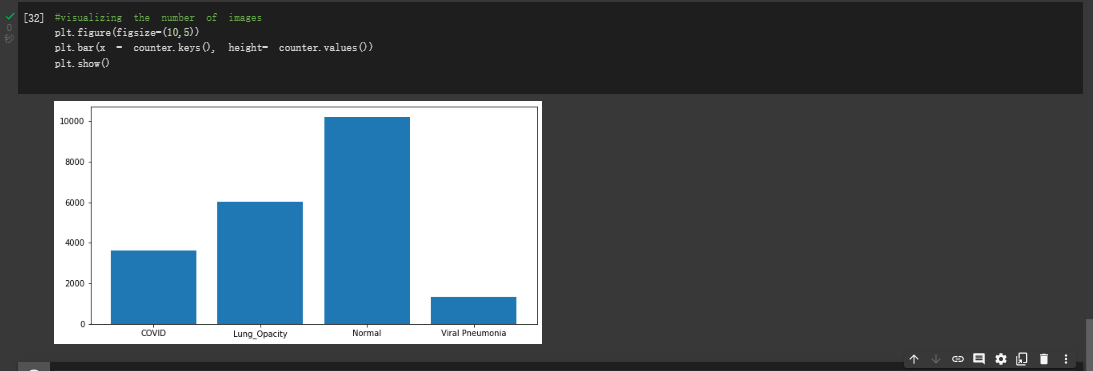


#visualizing the number of images

plt.figure(figsize=(10,5))

plt.bar(x = counter.keys(), height= counter.values())

plt.show()



# First off we need to give equal amout of data for every class.

# we will be Deviding our data between Train, Test, Validation

Train Data:

Normal : 3500

Lung Opacity : 3500

Covid+ Pneumonia : 3000 + 500

Validation Data:

Normal : 1000

Lung Opacity : 1000

Covid + Pneumonia : 500 +500

Test Data:

Normal : 1000

Lung Opacity : 1000

Covid + Pneumonia : 100 +500

代碼如下：

if not os.path.exists(NEW\_DIR+"train\_test\_split/"):

  os.makedirs(NEW\_DIR+"train\_test\_split/")

  os.makedirs(NEW\_DIR+"train\_test\_split/train/Normal")

  os.makedirs(NEW\_DIR+"train\_test\_split/train/Covid")

  os.makedirs(NEW\_DIR+"train\_test\_split/test/Normal")

  os.makedirs(NEW\_DIR+"train\_test\_split/test/Covid")

  os.makedirs(NEW\_DIR+"train\_test\_split/validation/Normal")

  os.makedirs(NEW\_DIR+"train\_test\_split/validation/Covid")

  # Train Data

  for i in np.random.choice(replace= False , size= 3000 , a = glob.glob(NEW\_DIR+imgs[0]+"\*") ):

    shutil.copy(i , NEW\_DIR+"train\_test\_split/train/Covid" )

    os.remove(i)

  for i in np.random.choice(replace= False , size= 3900 , a = glob.glob(NEW\_DIR+imgs[2]+"\*") ):

    shutil.copy(i , NEW\_DIR+"train\_test\_split/train/Normal" )

    os.remove(i)

  for i in np.random.choice(replace= False , size= 900 , a = glob.glob(NEW\_DIR+imgs[3]+"\*") ):

    shutil.copy(i , NEW\_DIR+"train\_test\_split/train/Covid" )

    os.remove(i)

  # Validation Data

  for i in np.random.choice(replace= False , size= 308 , a = glob.glob(NEW\_DIR+imgs[0]+"\*") ):

    shutil.copy(i , NEW\_DIR+"train\_test\_split/validation/Covid" )

    os.remove(i)

  for i in np.random.choice(replace= False , size= 500 , a = glob.glob(NEW\_DIR+imgs[2]+"\*") ):

    shutil.copy(i , NEW\_DIR+"train\_test\_split/validation/Normal" )

    os.remove(i)

  for i in np.random.choice(replace= False , size= 200 , a = glob.glob(NEW\_DIR+imgs[3]+"\*") ):

    shutil.copy(i , NEW\_DIR+"train\_test\_split/validation/Covid" )

    os.remove(i)

  # Test Data

  for i in np.random.choice(replace= False , size= 300 , a = glob.glob(NEW\_DIR+imgs[0]+"\*") ):

    shutil.copy(i , NEW\_DIR+"train\_test\_split/test/Covid" )

    os.remove(i)

  for i in np.random.choice(replace= False , size= 300 , a = glob.glob(NEW\_DIR+imgs[2]+"\*") ):

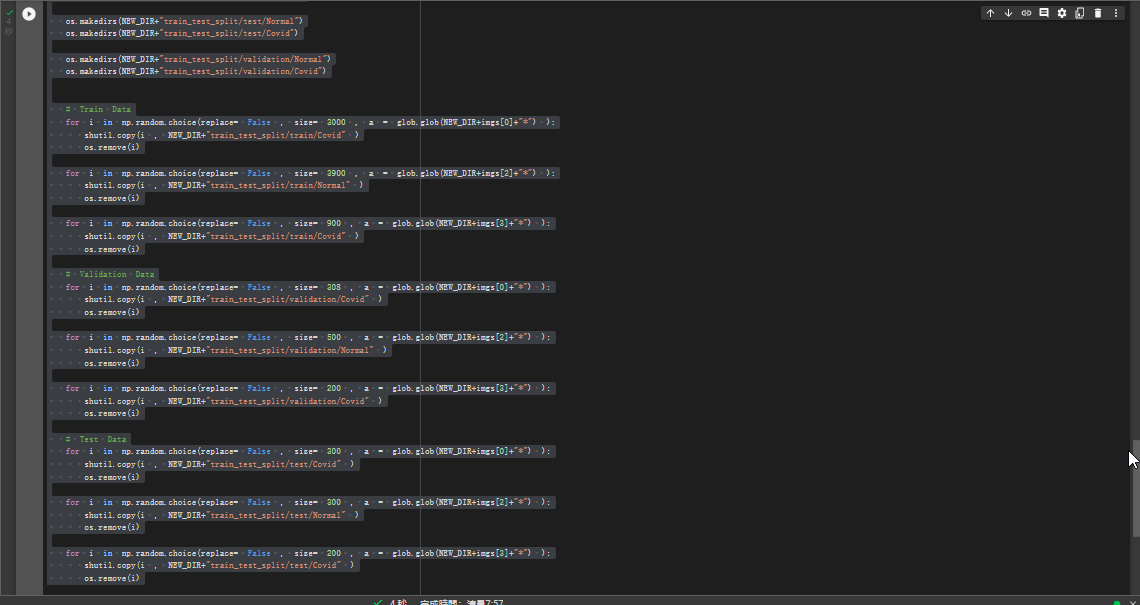
    shutil.copy(i , NEW\_DIR+"train\_test\_split/test/Normal" )

    os.remove(i)

  for i in np.random.choice(replace= False , size= 200 , a = glob.glob(NEW\_DIR+imgs[3]+"\*") ):

    shutil.copy(i , NEW\_DIR+"train\_test\_split/test/Covid" )

    os.remove(i)

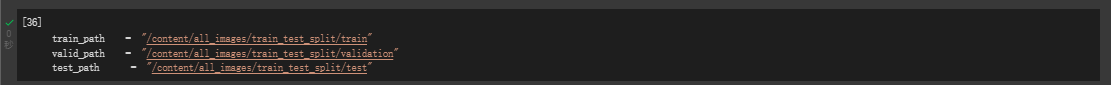


### 为keras生成数据流

train\_path  = "/content/all\_images/train\_test\_split/train"

valid\_path  = "/content/all\_images/train\_test\_split/validation"

test\_path   = "/content/all\_images/train\_test\_split/test"



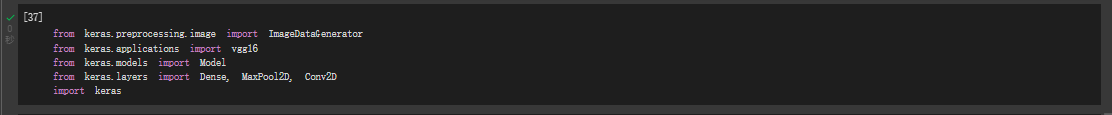
from keras.preprocessing.image import ImageDataGenerator

from keras.applications import vgg16

from keras.models import Model

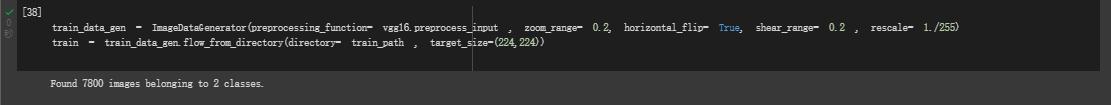
from keras.layers import Dense, MaxPool2D, Conv2D

import keras



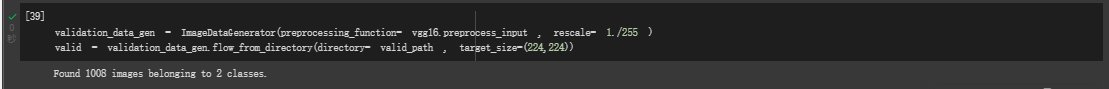
train\_data\_gen = ImageDataGenerator(preprocessing\_function= vgg16.preprocess\_input , zoom\_range= 0.2, horizontal\_flip= True, shear\_range= 0.2 , rescale= 1./255)

train = train\_data\_gen.flow\_from\_directory(directory= train\_path , target\_size=(224,224))



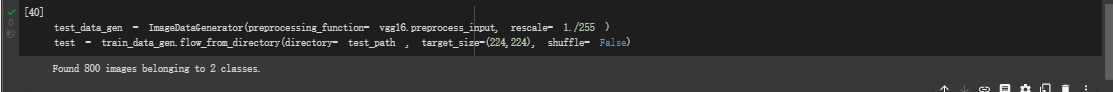
validation\_data\_gen = ImageDataGenerator(preprocessing\_function= vgg16.preprocess\_input , rescale= 1./255 )

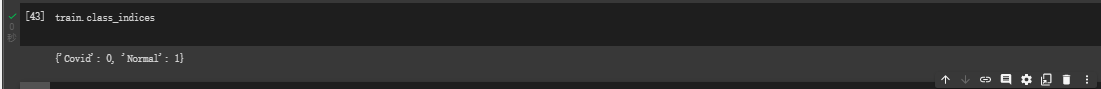
valid = validation\_data\_gen.flow\_from\_directory(directory= valid\_path , target\_size=(224,224))



test\_data\_gen = ImageDataGenerator(preprocessing\_function= vgg16.preprocess\_input, rescale= 1./255 )

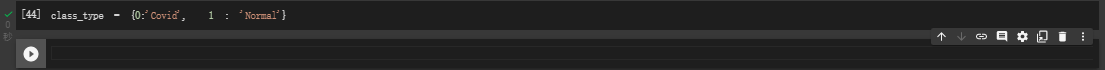
test = train\_data\_gen.flow\_from\_directory(directory= test\_path , target\_size=(224,224), shuffle= False)





***# Covid +ve X-Ray is represented by 0 and Normal is represented by 1***

class\_type = {0:'Covid',  1 : 'Normal'}



***# to visualize the images in the traing data denerator***

t\_img , label = train.next()



# function when called will prot the images

def plotImages(img\_arr, label):

  """

  input  :- images array

  output :- plots the images

  """

  for im, l in zip(img\_arr,label) :

    plt.figure(figsize= (5,5))

    plt.imshow(im, cmap = 'gray')

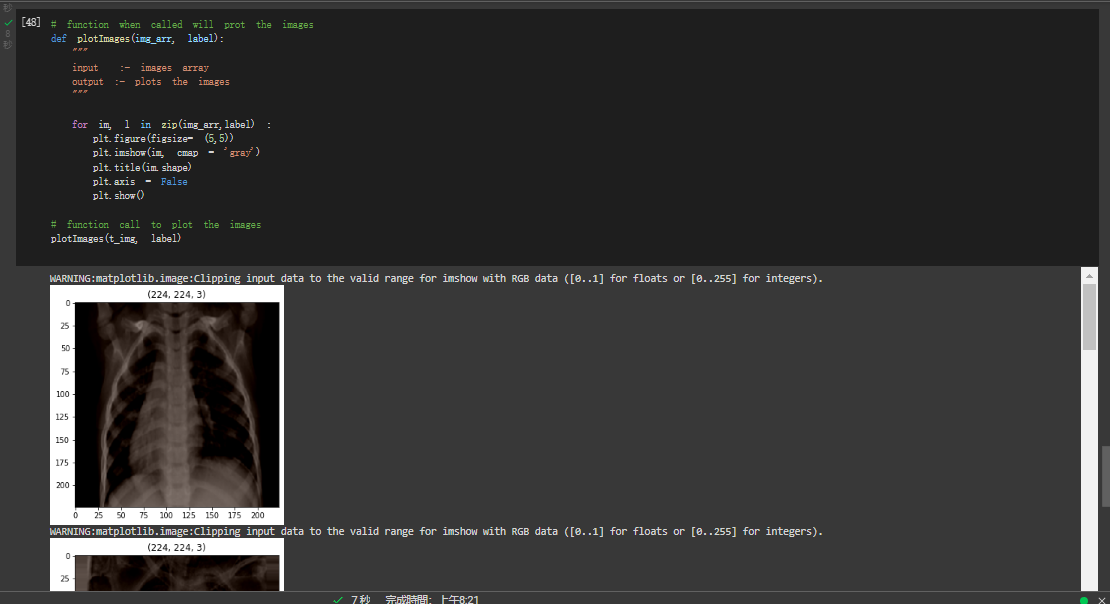
    plt.title(im.shape)

    plt.axis = False

    plt.show()

# function call to plot the images

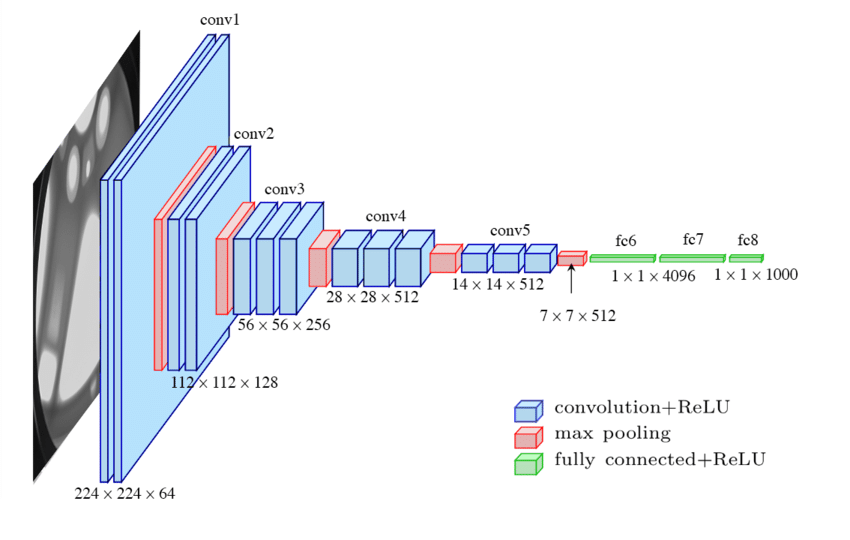
plotImages(t\_img, label)



*##we will be using our model vgg16*

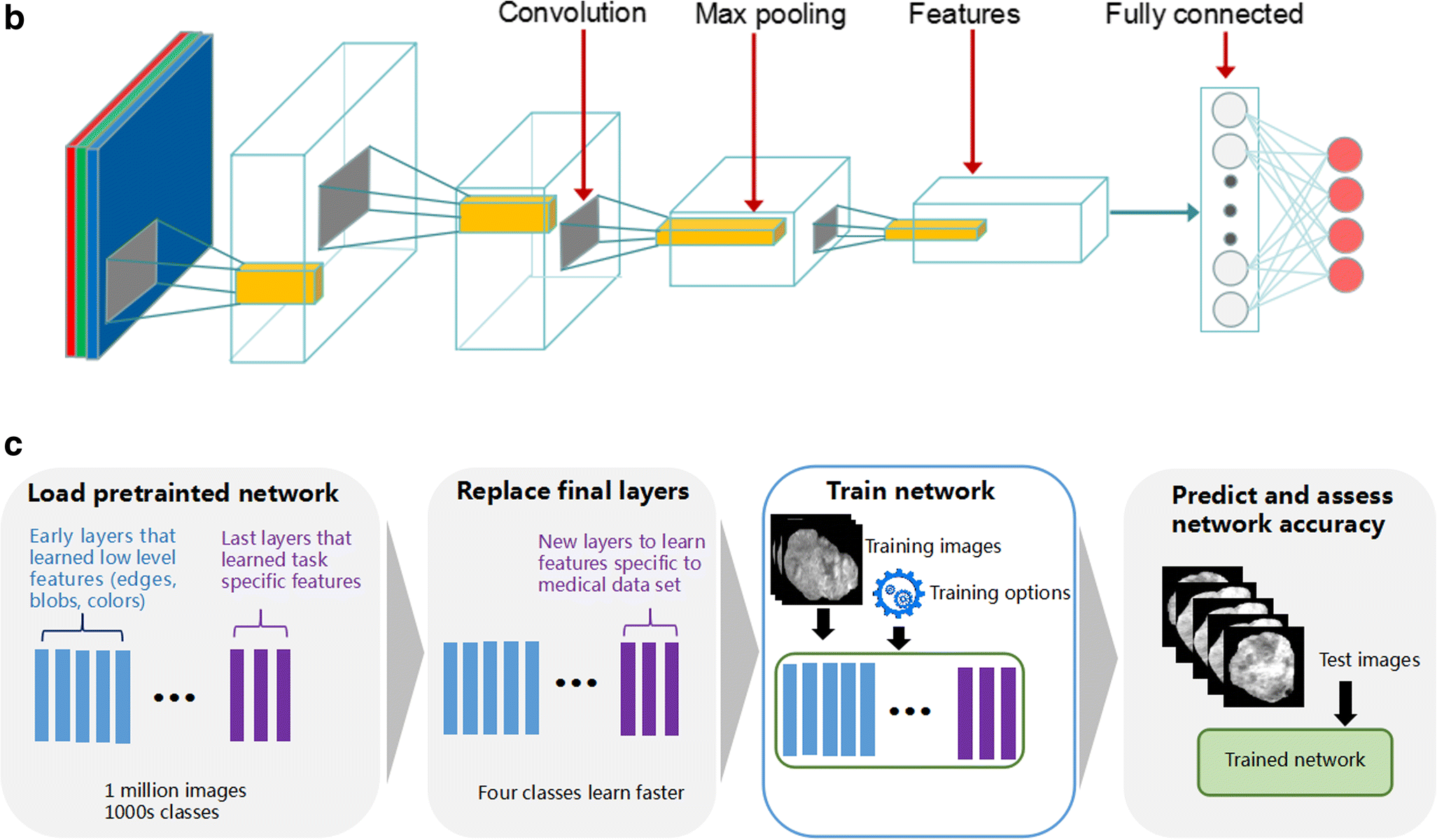
**Model Arechitecture: VGG 16**

**VGG-16 is a convolution neural net (CNN ) architecture which was used to win ILSVR(Imagenet) competition in 2014. It is considered to be one of the excellent vision model architecture till date. ... It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture.**



在這裡使用的模式是 ResNet50

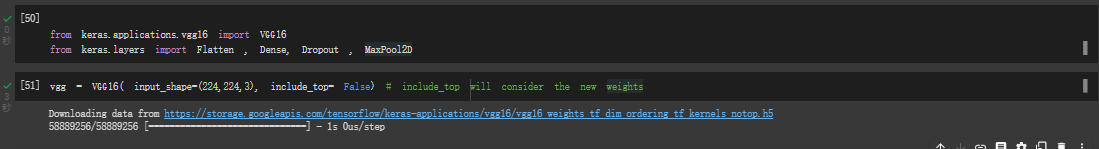
ResNet-50 模型由 5 個階段組成，每個階段都有一個卷積和身份塊。每個卷積塊有 3 個卷積層，每個恆等塊也有 3 個卷積層。 ResNet-50 擁有超過 2300 萬個可訓練參數



from keras.applications.vgg16 import VGG16

from keras.layers import Flatten , Dense, Dropout , MaxPool2D

vgg = VGG16( input\_shape=(224,224,3), include\_top= False) # include\_top will consider the new weights



for layer in vgg.layers:           # Dont Train the parameters again

  layer.trainable = False

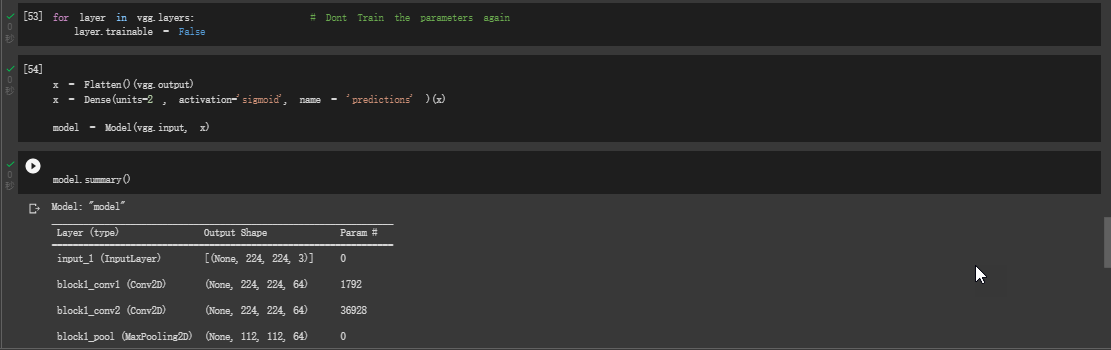
x = Flatten()(vgg.output)

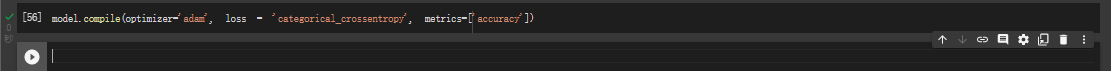
x = Dense(units=2 , activation='sigmoid', name = 'predictions' )(x)

***# creating our model.***

model = Model(vgg.input, x)

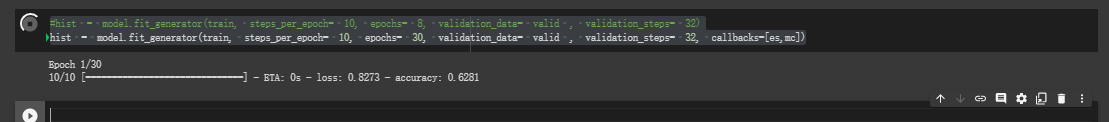
model.summary()





#hist = model.fit\_generator(train, steps\_per\_epoch= 10, epochs= 8, validation\_data= valid , validation\_steps= 32)

hist = model.fit\_generator(train, steps\_per\_epoch= 10, epochs= 30, validation\_data= valid , validation\_steps= 32, callbacks=[es,mc])



注意：此須等候整個完成始可進行下一步！！費時甚久！為節省時間考量可將epoch=10作調整！

Epoch 1/30

10/10 [==============================] - 57s 2s/step - loss: 1.1836 - accuracy: 0.5383 - val\_loss: 0.4807 - val\_accuracy: 0.7867

Epoch 00001: val\_accuracy improved from -inf to 0.78671, saving model to bestmodel.h5

Epoch 2/30

10/10 [==============================] - 19s 2s/step - loss: 0.4780 - accuracy: 0.7896 - val\_loss: 0.4653 - val\_accuracy: 0.7946

Epoch 00002: val\_accuracy improved from 0.78671 to 0.79464, saving model to bestmodel.h5

Epoch 3/30

10/10 [==============================] - 13s 1s/step - loss: 0.4527 - accuracy: 0.8094 - val\_loss: 0.3679 - val\_accuracy: 0.8244

Epoch 00003: val\_accuracy improved from 0.79464 to 0.82440, saving model to bestmodel.h5

Epoch 4/30

10/10 [==============================] - 13s 1s/step - loss: 0.5497 - accuracy: 0.7264 - val\_loss: 0.3696 - val\_accuracy: 0.8492

Epoch 00004: val\_accuracy improved from 0.82440 to 0.84921, saving model to bestmodel.h5

Epoch 5/30

10/10 [==============================] - 13s 1s/step - loss: 0.3894 - accuracy: 0.8267 - val\_loss: 0.3284 - val\_accuracy: 0.8671

Epoch 00005: val\_accuracy improved from 0.84921 to 0.86706, saving model to bestmodel.h5

Epoch 6/30

10/10 [==============================] - 13s 1s/step - loss: 0.3270 - accuracy: 0.8613 - val\_loss: 0.3292 - val\_accuracy: 0.8671

Epoch 00006: val\_accuracy did not improve from 0.86706

Epoch 7/30

10/10 [==============================] - 12s 1s/step - loss: 0.3273 - accuracy: 0.8543 - val\_loss: 0.3554 - val\_accuracy: 0.8601

Epoch 00007: val\_accuracy did not improve from 0.86706

Epoch 8/30

10/10 [==============================] - 13s 1s/step - loss: 0.3367 - accuracy: 0.8548 - val\_loss: 0.3197 - val\_accuracy: 0.8770

Epoch 00008: val\_accuracy improved from 0.86706 to 0.87698, saving model to bestmodel.h5

Epoch 00008: early stopping

**## load only the best model**

from keras.models import load\_model

model = load\_model("bestmodel.h5")

**Seeing how our model has performed**

h = hist.history

h.keys()

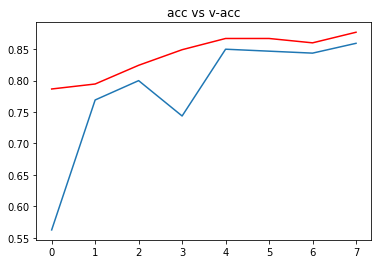
dict\_keys(['loss', 'accuracy', 'val\_loss', 'val\_accuracy'])

plt.plot(h['accuracy'])

plt.plot(h['val\_accuracy'] , c = "red")

plt.title("acc vs v-acc")

plt.show()

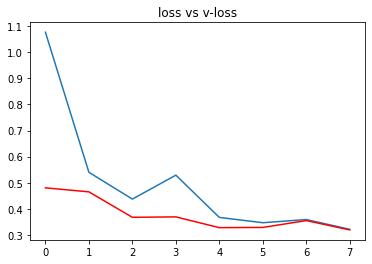
******

plt.plot(h['loss'])

plt.plot(h['val\_loss'] , c = "red")

plt.title("loss vs v-loss")

plt.show()

******

**# checking out the accurscy of our model**

acc = model.evaluate\_generator(generator= test)[1]

print(f"The accuracy of your model is = {acc} %")

The accuracy of your model is = 0.8837500214576721 %

from keras.preprocessing import image

def get\_img\_array(img\_path):

"""

Input : Takes in image path as input

Output : Gives out Pre-Processed image

"""

path = img\_path

img = image.load\_img(path, target\_size=(224,224,3))

img = image.img\_to\_array(img)/255

img = np.expand\_dims(img , axis= 0 )

return img

**# path for that new image. ( you can take it either from google or any other scource)**

path = "/content/all\_images/COVID-1905.png" # you can add any image path

**#predictions: path:- provide any image from google or provide image from all image folder**

img = get\_img\_array(path)

res = class\_type[np.argmax(model.predict(img))]

print(f"The given X-Ray image is of type = {res}")

print()

print(f"The chances of image being Covid is : {model.predict(img)[0][0]\*100} percent")

print()

print(f"The chances of image being Normal is : {model.predict(img)[0][1]\*100} percent")

**# to display the image**

plt.imshow(img[0], cmap = "gray")

plt.title("input image")

plt.show()

The given X-Ray image is of type = Covid

The chances of image being Covid is : 65.1364266872406 percent

The chances of image being Normal is : 11.151573807001114 percent

******

**Grad CAM Visualization**

import tensorflow as tf

**# this function is udes to generate the heat map of aan image**

def make\_gradcam\_heatmap(img\_array, model, last\_conv\_layer\_name, pred\_index=None):

**# First, we create a model that maps the input image to the activations**

**# of the last conv layer as well as the output predictions**

grad\_model = tf.keras.models.Model(

[model.inputs], [model.get\_layer(last\_conv\_layer\_name).output, model.output]

)

**# Then, we compute the gradient of the top predicted class for our input image**

**# with respect to the activations of the last conv layer**

with tf.GradientTape() as tape:

last\_conv\_layer\_output, preds = grad\_model(img\_array)

if pred\_index is None:

pred\_index = tf.argmax(preds[0])

class\_channel = preds[:, pred\_index]

**# This is the gradient of the output neuron (top predicted or chosen)**

**# with regard to the output feature map of the last conv layer**

grads = tape.gradient(class\_channel, last\_conv\_layer\_output)

# This is a vector where each entry is the mean intensity of the gradient

**# over a specific feature map channel**

pooled\_grads = tf.reduce\_mean(grads, axis=(0, 1, 2))

**# We multiply each channel in the feature map array**

**# by "how important this channel is" with regard to the top predicted class**

**# then sum all the channels to obtain the heatmap class activation**

last\_conv\_layer\_output = last\_conv\_layer\_output[0]

heatmap = last\_conv\_layer\_output @ pooled\_grads[..., tf.newaxis]

heatmap = tf.squeeze(heatmap)

**# For visualization purpose, we will also normalize the heatmap between 0 & 1**

heatmap = tf.maximum(heatmap, 0) / tf.math.reduce\_max(heatmap)

return heatmap.numpy()

import matplotlib.cm as cm

from IPython.display import Image, display

**# put the heatmap to our image to understand the area of interest**

def save\_and\_display\_gradcam(img\_path , heatmap, cam\_path="cam.jpg", alpha=0.4):

"""

img input shoud not be expanded

"""

**# Load the original image**

img = keras.preprocessing.image.load\_img(img\_path)

img = keras.preprocessing.image.img\_to\_array(img)

**# Rescale heatmap to a range 0-255**

heatmap = np.uint8(255 \* heatmap)

**# Use jet colormap to colorize heatmap**

jet = cm.get\_cmap("jet")

**# Use RGB values of the colormap**

jet\_colors = jet(np.arange(256))[:, :3]

jet\_heatmap = jet\_colors[heatmap]

**# Create an image with RGB colorized heatmap**

jet\_heatmap = keras.preprocessing.image.array\_to\_img(jet\_heatmap)

jet\_heatmap = jet\_heatmap.resize((img.shape[1], img.shape[0]))

jet\_heatmap = keras.preprocessing.image.img\_to\_array(jet\_heatmap)

**# Superimpose the heatmap on original image**

superimposed\_img = jet\_heatmap \* alpha + img

superimposed\_img = keras.preprocessing.image.array\_to\_img(superimposed\_img)

**# Save the superimposed image**

superimposed\_img.save(cam\_path)

**# Display Grad CAM**

display(Image(cam\_path))

**# function that is used to predict the image type and the ares that are affected by covid**

def image\_prediction\_and\_visualization(path,last\_conv\_layer\_name = "block5\_conv3", model = model):

"""

input: is the image path, name of last convolution layer , model name

output : returs the predictions and the area that is effected

"""

img\_array = get\_img\_array(path)

heatmap = make\_gradcam\_heatmap(img\_array, model, last\_conv\_layer\_name)

img = get\_img\_array(path)

res = class\_type[np.argmax(model.predict(img))]

print(f"The given X-Ray image is of type = {res}")

print()

print(f"The chances of image being Covid is : {model.predict(img)[0][0]\*100} %")

print(f"The chances of image being Normal is : {model.predict(img)[0][1]\*100} %")

print()

print("image with heatmap representing the covid spot")

**# function call**

save\_and\_display\_gradcam(path, heatmap)

print()

print("the original input image")

print()

a = plt.imread(path)

plt.imshow(a, cmap = "gray")

plt.title("Original image")

plt.show()

#predictions

# provide the path of any image from google or any other scource

# the path is already defigned above , but you can also provide the path here to avoid scrolling up

# for covid image :

path = "/content/all\_images/COVID-1873.png"

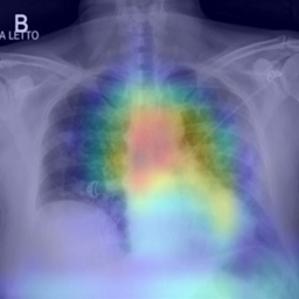
image\_prediction\_and\_visualization(path)

The given X-Ray image is of type = Covid

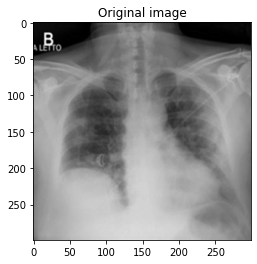
The chances of image being Covid is : 79.94917631149292 %

The chances of image being Normal is : 7.098855823278427 %

image with heatmap representing the covid spot

******

the original input image

******

# for normal image :

path = "/content/all\_images/train\_test\_split/test/Normal/Normal-10095.png"

image\_prediction\_and\_visualization(path)

The given X-Ray image is of type = Normal

The chances of image being Covid is : 5.242202058434486 %

The chances of image being Normal is : 80.79538941383362 %

image with heatmap representing the covid spot

******

the original input image

******