Face Mask Detection

Importing the necessary libraries and functions

```
import pandas as pd
import cv2
import numpy as np
import os
import tensorflow as tf
import matplotlib.pyplot as plt
import pathlib
import matplotlib.image as mpimg
from keras.preprocessing import image
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
```

Image Datasets and Preprocessing

Using a dataset merged from some datasets on kaggle.

The vgg16 preprocessing function is used for image preprocessing.

The dataset consists of 3 directories named Train, Validation, Test and each of these has 2 sub-directories named Mask and Non Mask.

```
train_path="../input/self2/archive (1)/New Masks Dataset/Train"
train_data=ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(directory=train_path, target_size=(224,224),
                         classes=['Mask', 'Non Mask'], batch_size=32, shuffle= True)

val_path="../input/self2/archive (1)/New Masks Dataset/Validation"
val_data=ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(directory=val_path, target_size=(224,224),
                         classes=['Mask', 'Non Mask'], batch_size=32, shuffle = True)

test_path="../input/self2/archive (1)/New Masks Dataset/Test"
test_data=ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(directory=test_path, target_size=(224,224),
                         classes=['Mask', 'Non Mask'], batch_size=32, shuffle = False)
```

```
Found 1630 images belonging to 2 classes.
Found 244 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
```

The images are equally divided between the two classes

The classes are represented by 0 and 1

```
train_data.class_indices
```

```
{'Mask': 0, 'Non Mask': 1}
```

Convolutional Neural Network (CNN)

We classify the images with the help of a CNN.

Keras makes it pretty simple to build our model so we use Sequential to build our model (layer by layer).

The model looks like

```
num_classes = 2

model = tf.keras.Sequential([
  tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(224,224,3)),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(64, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(128, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(256, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(512, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(256, activation="relu"),
  tf.keras.layers.Dense(128, activation="relu"),
  tf.keras.layers.Dense(num_classes, activation="softmax")
])
```

Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 52, 52, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 26, 26, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 512)       1180160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 12800)             0
_________________________________________________________________
dense (Dense)                (None, 256)               3277056
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 4,878,786
Trainable params: 4,878,786
Non-trainable params: 0
```

model.compile

```
model.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=['accuracy'])
```

model.fit

```
model.fit(
  train_data,
  validation_data=val_data,
  epochs=15
)
```

Training the model

```
Epoch 1/15
51/51 [==============================] - 22s 428ms/step - loss: 10.6165 - accuracy: 0.6796 - val_loss: 0.3780 - val_accuracy: 0.8730
Epoch 2/15
51/51 [==============================] - 23s 446ms/step - loss: 0.2570 - accuracy: 0.9001 - val_loss: 0.2549 - val_accuracy: 0.9016
Epoch 3/15
51/51 [==============================] - 22s 427ms/step - loss: 0.2323 - accuracy: 0.9270 - val_loss: 0.1899 - val_accuracy: 0.9426
Epoch 4/15
51/51 [==============================] - 22s 440ms/step - loss: 0.2065 - accuracy: 0.9185 - val_loss: 0.1869 - val_accuracy: 0.9262
Epoch 5/15
51/51 [==============================] - 21s 416ms/step - loss: 0.1540 - accuracy: 0.9487 - val_loss: 0.1843 - val_accuracy: 0.9426
Epoch 6/15
51/51 [==============================] - 21s 418ms/step - loss: 0.1178 - accuracy: 0.9613 - val_loss: 0.4138 - val_accuracy: 0.8893
Epoch 7/15
51/51 [==============================] - 22s 436ms/step - loss: 0.1808 - accuracy: 0.9410 - val_loss: 0.1622 - val_accuracy: 0.9508
Epoch 8/15
51/51 [==============================] - 21s 417ms/step - loss: 0.1241 - accuracy: 0.9623 - val_loss: 0.2416 - val_accuracy: 0.9180
Epoch 9/15
51/51 [==============================] - 21s 408ms/step - loss: 0.1305 - accuracy: 0.9548 - val_loss: 0.1551 - val_accuracy: 0.9426
Epoch 10/15
51/51 [==============================] - 23s 441ms/step - loss: 0.0968 - accuracy: 0.9554 - val_loss: 0.2124 - val_accuracy: 0.9426
Epoch 11/15
51/51 [==============================] - 21s 416ms/step - loss: 0.0737 - accuracy: 0.9746 - val_loss: 0.1409 - val_accuracy: 0.9631
Epoch 12/15
51/51 [==============================] - 21s 411ms/step - loss: 0.0661 - accuracy: 0.9809 - val_loss: 0.1379 - val_accuracy: 0.9590
Epoch 13/15
51/51 [==============================] - 23s 439ms/step - loss: 0.0596 - accuracy: 0.9728 - val_loss: 0.3890 - val_accuracy: 0.8361
Epoch 14/15
51/51 [==============================] - 21s 419ms/step - loss: 0.0693 - accuracy: 0.9717 - val_loss: 0.1897 - val_accuracy: 0.9385
Epoch 15/15
51/51 [==============================] - 21s 413ms/step - loss: 0.0864 - accuracy: 0.9637 - val_loss: 0.1094 - val_accuracy: 0.9631
<tensorflow.python.keras.callbacks.History at 0x7f838037f2d0>
```

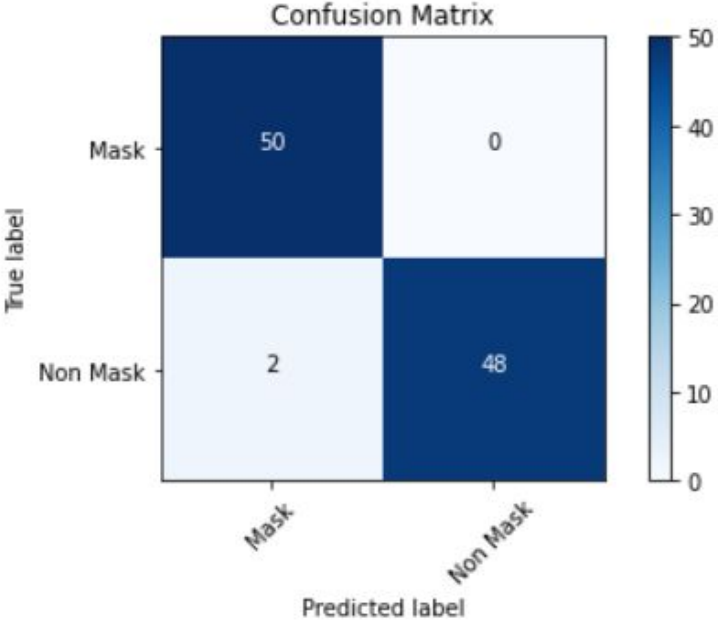
Evaluating the test data

```
model.evaluate(test_data)
```

```
4/4 [==============================] - 1s 259ms/step - loss: 0.0521 - accuracy: 0.9800
[0.05208831652998924, 0.9800000190734863]
```

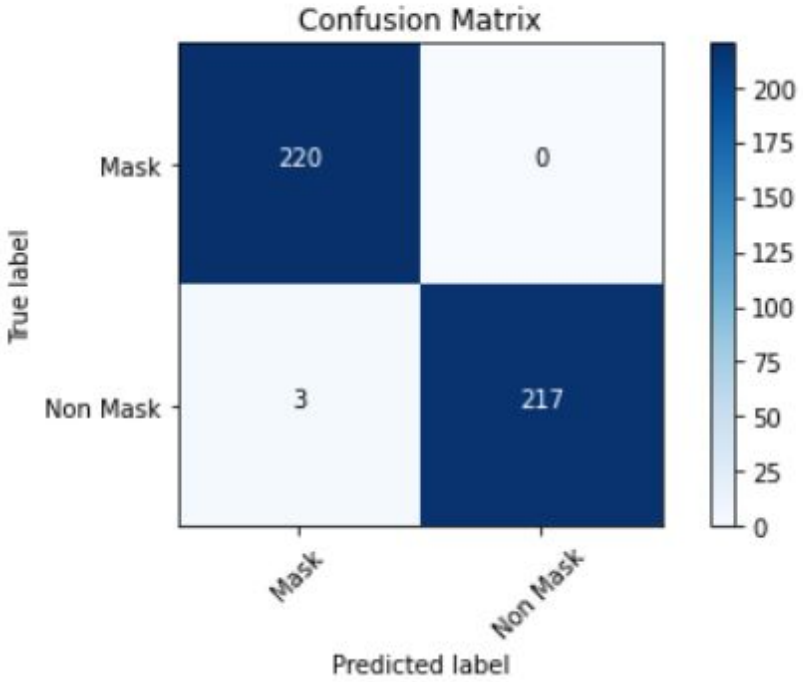
Using a confusion matrix for summary of the predictions

```
Confusion matrix, without normalization
[[50  0]
 [ 2 48]]
```

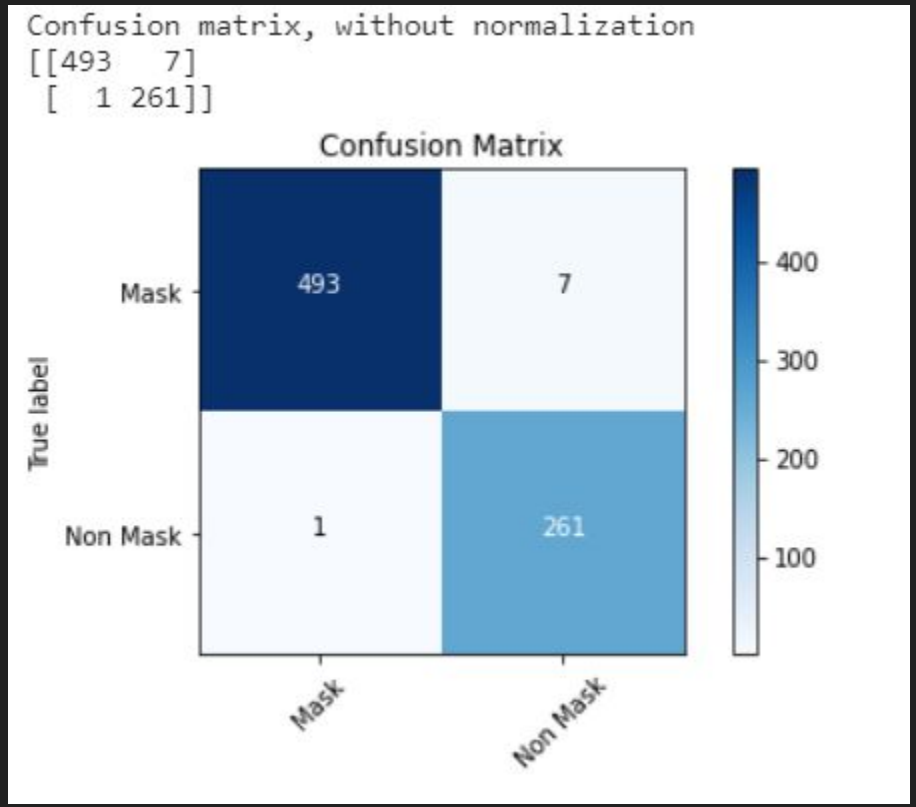


Confusion matrices for other test datasets

```
Confusion matrix, without normalization
[[220   0]
 [  3 217]]
```




```
Confusion matrix, without normalization
[[493   7]
 [  1 261]]
```



Predicting an image

We first load the image and resize it to the input size(224x224 in this case).

We then create a batch of one image as the model expects batch of images as input.

Next we process the image using the same preprocessing function before passing it on to the model.

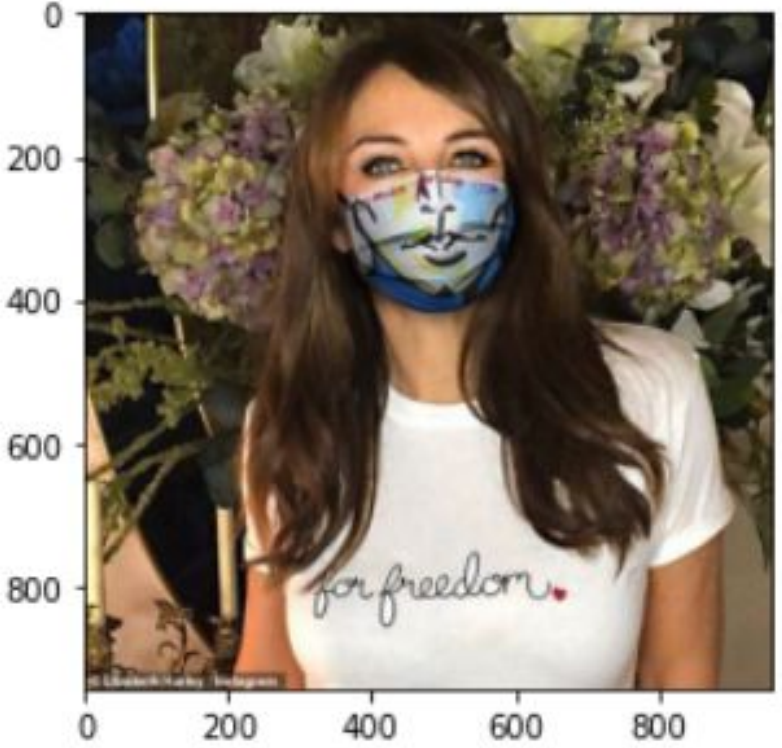
Predicting an image

```
path='../input/face-mask-detection/dataset/with_mask/image_10.png'
im=mpimg.imread(path)
plt.imshow(im)
img = image.load_img(path,target_size=(224,224))
img = image.img_to_array(img)
img = np.expand_dims(img,axis=0)
img_preprocessed = tf.keras.applications.vgg16.preprocess_input(img)
prediction = model.predict(img_preprocessed)
print (prediction)
if prediction[0][1] < 0.5:
    print("With Mask")

else:
    print("Without Mask")
```

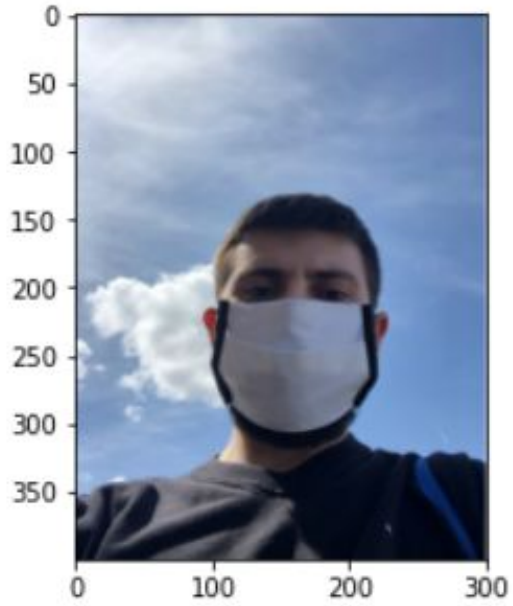
Output:

```
[[9.9994802e-01 5.1983934e-05]]
With Mask
```

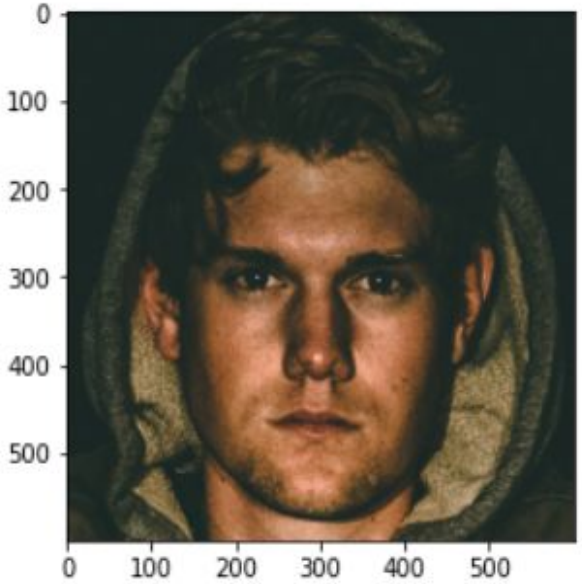


Some more predictions:

```
[[9.9917406e-01 8.2591991e-04]]
With Mask
```




```
[[5.463338e-04 9.994537e-01]]
Without Mask
```




```
[[3.5581117e-05 9.9996448e-01]]
Without Mask
```

The dataset used for training:

www.kaggle.com/dataset/4433c9cbc3d663ae79f77497349d64058437949da2611edacbc0dc5264722f0d

The datasets used for testing are

https://www.kaggle.com/dhruvmak/face-mask-detection

https://www.kaggle.com/shrirajchauhan/face-mask-detection-medical-and-non medical-masks

THANK YOU