Name – Md Ferhan Khan **Registration Number – 11810892**

Roll Number – B67 **Section – KM073**

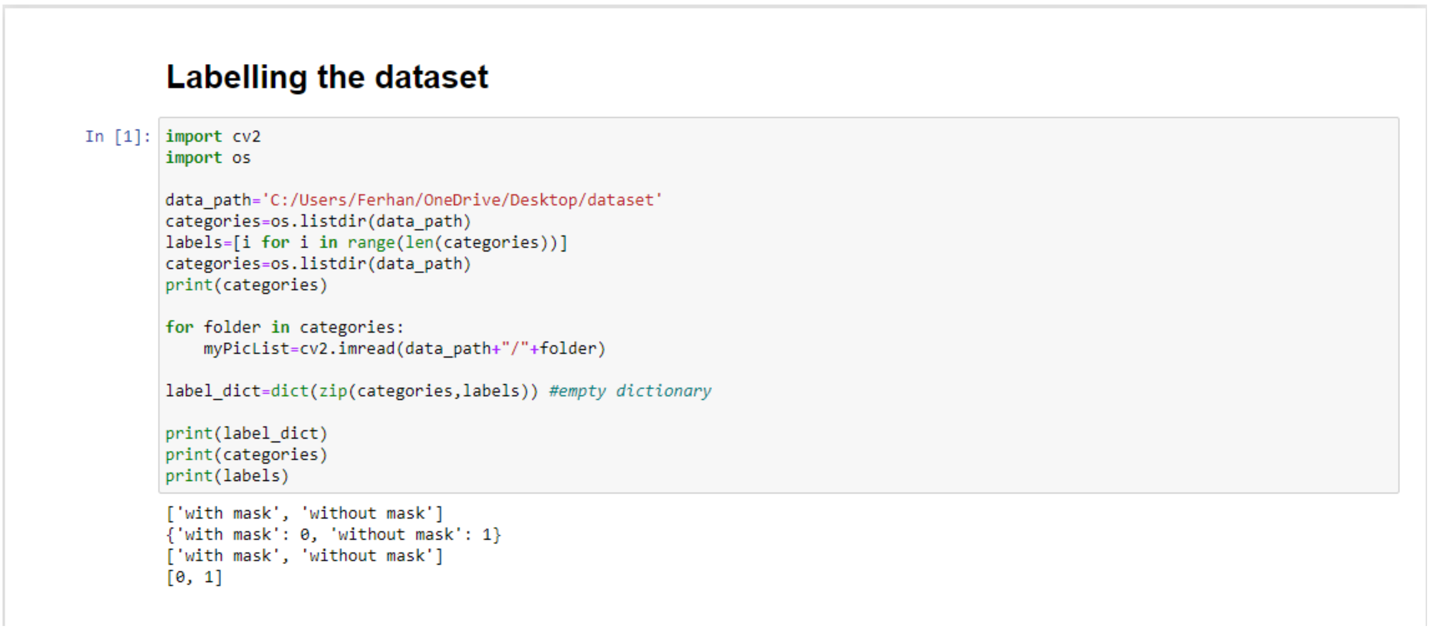
Project Description:

Face Mask Detection using Machine Learning. A system could be made to detect masks automatically. It could be installed on the gateway of restaurants, Colleges, Offices etc. so that people without wearing masks are not allowed to enter.

Libraries used:

- OpenCV – for collecting data from webcam
- Os – for joining paths
- Numpy – for various mathematical operations like normalisation, reshaping etc.
- Scikit – for splitting the testing and training dataset
- Keras – for specifying the architecture
- Matplotlib – for plotting the dataset images.

Code Description

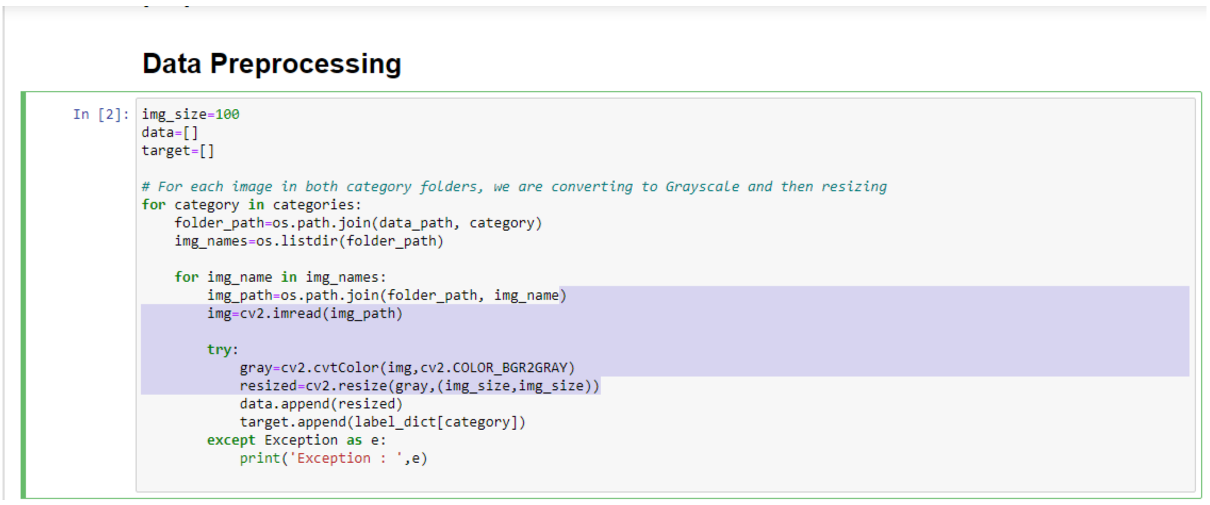


Labelling the dataset

In [1]:
```
import cv2
import os

data_path='C:/Users/Ferhan/OneDrive/Desktop/dataset'
categories=os.listdir(data_path)
labels=[i for i in range(len(categories))]
categories=os.listdir(data_path)
print(categories)

for folder in categories:
    myPicList=cv2.imread(data_path+"/"+folder)

label_dict=dict(zip(categories,labels)) #empty dictionary

print(label_dict)
print(categories)
print(labels)
```

```
['with mask', 'without mask']
{'with mask': 0, 'without mask': 1}
['with mask', 'without mask']
[0, 1]
```

- Importing opencv and os library.
- Giving the path of the dataset. Then the folder names will be loaded.
- Then labels are created 0 and 1 because there are two folders in the dataset, with_mask and without_mask.
- Label dictionary is created with keys as without_mask and with_mask and values 0 and 1 respectively.



Data Preprocessing

```
In [2]: img_size=100
        data=[]
        target=[]

        # For each image in both category folders, we are converting to Grayscale and then resizing
        for category in categories:
            folder_path=os.path.join(data_path, category)
            img_names=os.listdir(folder_path)

            for img_name in img_names:
                img_path=os.path.join(folder_path, img_name)
                img=cv2.imread(img_path)

                try:
                    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
                    resized=cv2.resize(gray,(img_size,img_size))
                    data.append(resized)
                    target.append(label_dict[category])
                except Exception as e:
                    print('Exception : ',e)
```

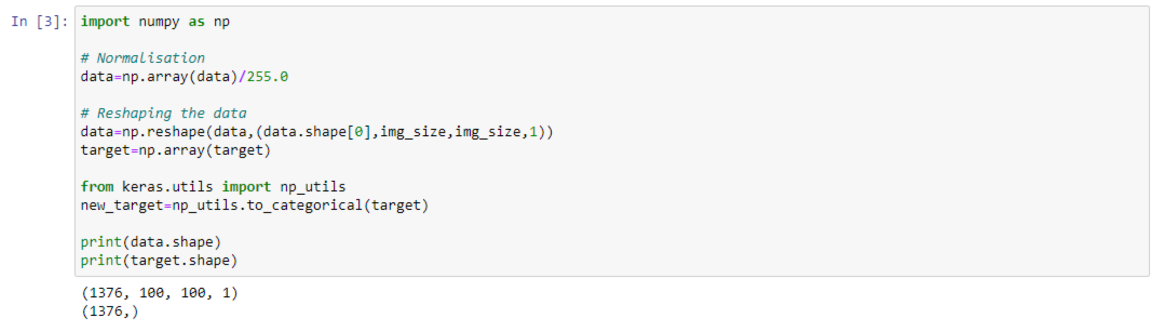
- Declaring the size of the image as 100 by 100.
- Two empty lists to save the images and labels.
- Loading all the images from the

two folders in our dataset. In the try

block:

  - At first the images are converted into the grayscale.
  - Then resized to 100 X 100.
  - Then the resized image is appended to data.
  - And also target is also appended.

If sometimes images is not available then it is handled using the exception .



```
In [3]: import numpy as np

        # Normalisation
        data=np.array(data)/255.0

        # Reshaping the data
        data=np.reshape(data,(data.shape[0],img_size,img_size,1))
        target=np.array(target)

        from keras.utils import np_utils
        new_target=np_utils.to_categorical(target)

        print(data.shape)
        print(target.shape)

        (1376, 100, 100, 1)
        (1376,)
```

  - Normalization is done (dividing by 255 which converts the pixel range between 0 and 1).
  - The images are reshaped into 4 dimensional arrays as per the requirement of CNN.
  - Target is also converted into numpy array.
  - Since, our last layer of our neural network will have two neurons, for without mask and mask so we need to convert the target to categorical representation.

Saving the data and target

```
In [4]: import numpy as np
        np.save('data',data)
        np.save('target',new_target)
```

```
In [5]: import numpy as np

        data=np.load('data.npy')
        target=np.load('target.npy')
```

The data containing images and the target containing whether it is 'mask' or 'without mask' is saved and loaded for further processing.

Splitting the dataset

```
In [6]: from sklearn.model_selection import train_test_split
        train_data, test_data, train_target, test_target=train_test_split(data,target, test_size=0.1)
```

```
In [7]: print(train_data.shape)
        print(test_data.shape)
        print(train_target.shape)
        print(test_target.shape)

        (1238, 100, 100, 1)
        (138, 100, 100, 1)
        (1238, 2)
        (138, 2)
```

The dataset is splitted, 90% for training and 10% for testing.

Generating Images

```
In [8]: from keras.preprocessing.image import ImageDataGenerator
```

```
In [9]: dataGen=ImageDataGenerator(width_shift_range=0.1,height_shift_range=0.1,zoom_range=0.2,shear_range=0.1,rotation_range=10)
```

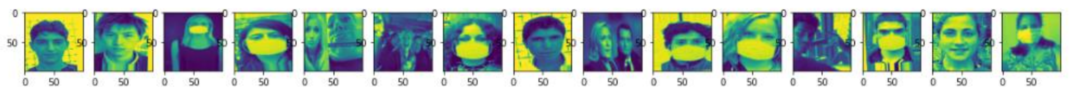
```
In [10]: dataGen.fit(train_data)
```

```
In [11]: batches=dataGen.flow(train_data, train_target,batch_size=20)
         x_batch,y_batch=next(batches)
```

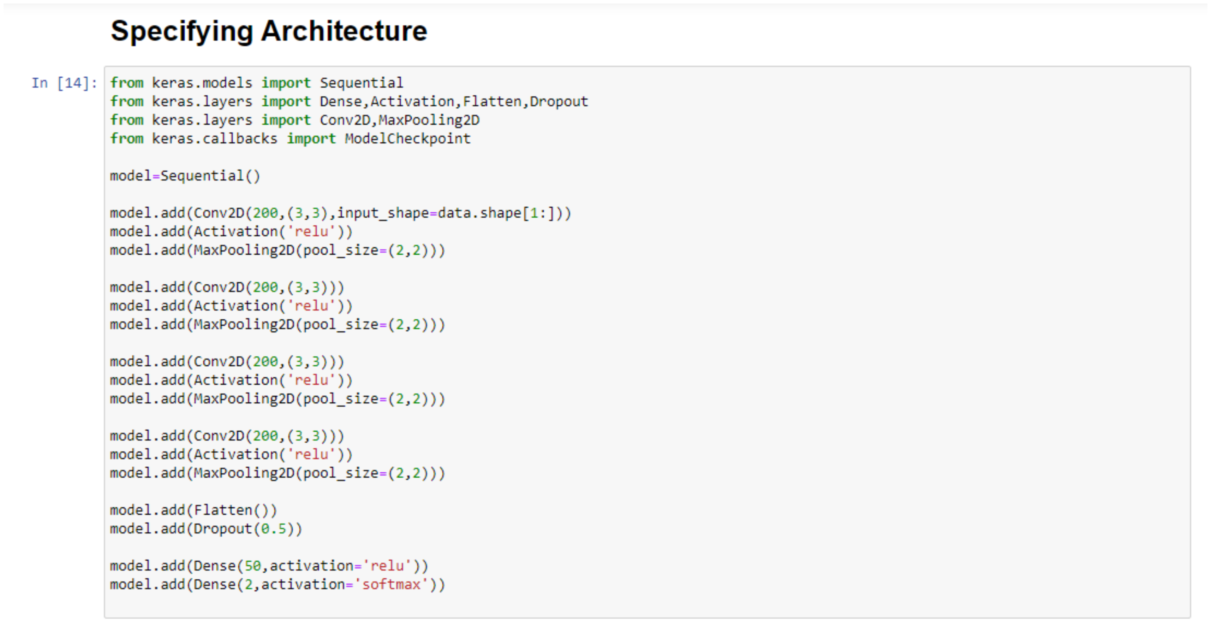
```
In [12]: x_batch.shape
Out[12]: (20, 100, 100, 1)
```

By the help of ImageDataGenerator from the keras we are generating the images so that our model would then also when the face from live feed is not at the centre or is slightly rotated and so.

```
In [13]: import matplotlib.pyplot as plt

         fig,ax=plt.subplots(1,15,figsize=(20,5))
         for i in range(15):
             ax[i].imshow(x_batch[i])
         plt.show()
```



Here the generated image is plotted using matplotlib.

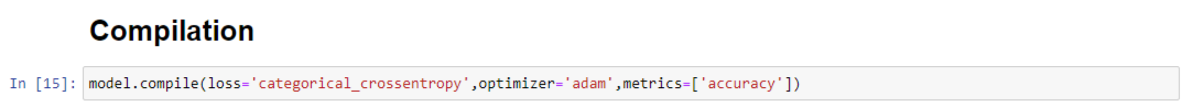


Specifying Architecture

```
In [14]: from keras.models import Sequential
         from keras.layers import Dense,Activation,Flatten,Dropout
         from keras.layers import Conv2D,MaxPooling2D
         from keras.callbacks import ModelCheckpoint

         model=Sequential()

         model.add(Conv2D(200,(3,3),input_shape=data.shape[1:]))
         model.add(Activation('relu'))
         model.add(MaxPooling2D(pool_size=(2,2)))

         model.add(Conv2D(200,(3,3)))
         model.add(Activation('relu'))
         model.add(MaxPooling2D(pool_size=(2,2)))

         model.add(Conv2D(200,(3,3)))
         model.add(Activation('relu'))
         model.add(MaxPooling2D(pool_size=(2,2)))

         model.add(Conv2D(200,(3,3)))
         model.add(Activation('relu'))
         model.add(MaxPooling2D(pool_size=(2,2)))

         model.add(Flatten())
         model.add(Dropout(0.5))

         model.add(Dense(50,activation='relu'))
         model.add(Dense(2,activation='softmax'))
```

The Convolutional Neural Network Architecture:

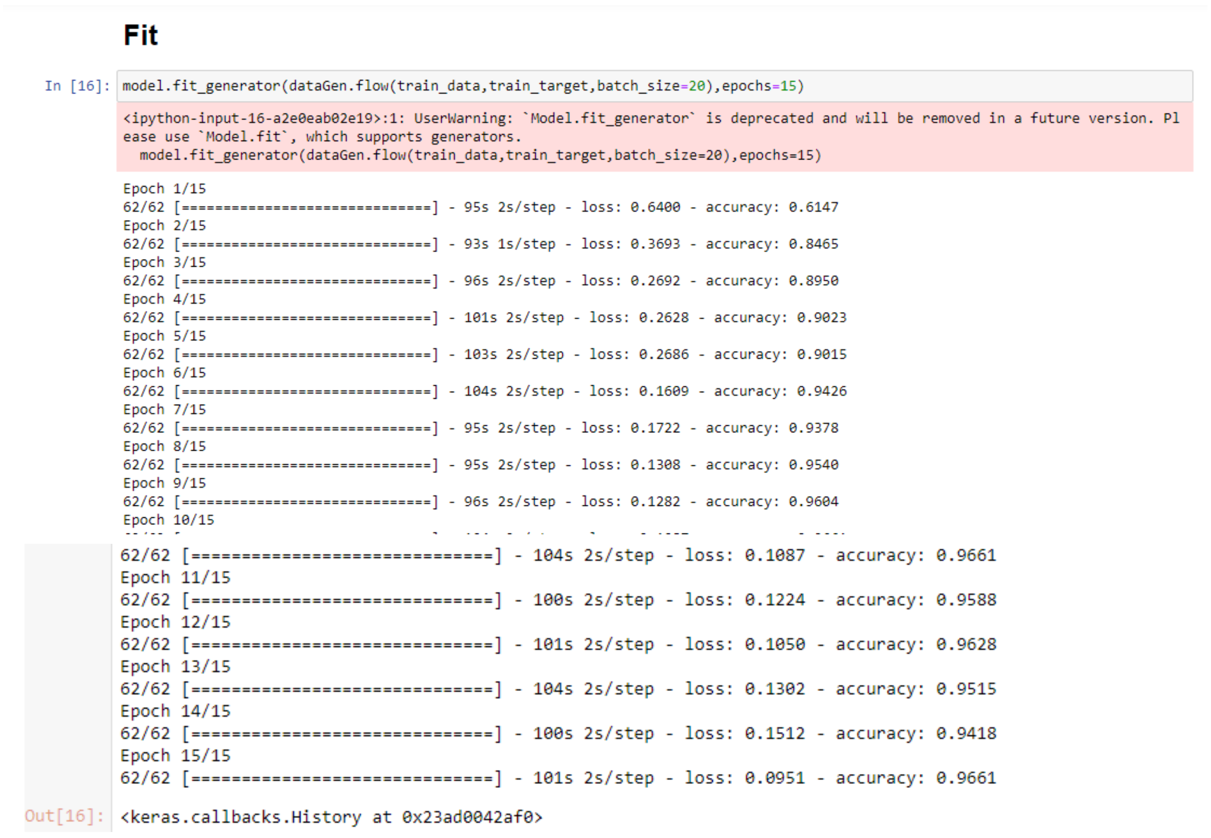
- Two convolutional layers, input as 100x100 image.
- The first CNN has 200 kernels in it and the second CNN has 100.
- Then the output from the two CNN is flatten and then connected to a dense layer of 50 neurons.
- And the last layer will have two neurons, with mask and without mask.
- Dropout is used to get rid of overfitting.



Compilation

```
In [15]: model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

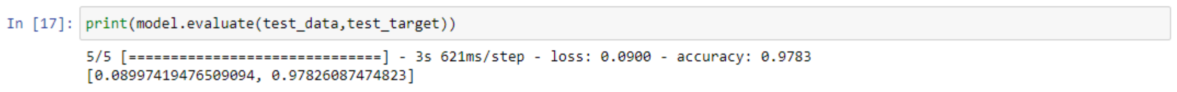
- 'Categorical Crossentropy' is used in the compilation because we have two categories : mask and without mask.
- Adam optimizer is used and the accuracy is to be printed in each epoch.



Fit

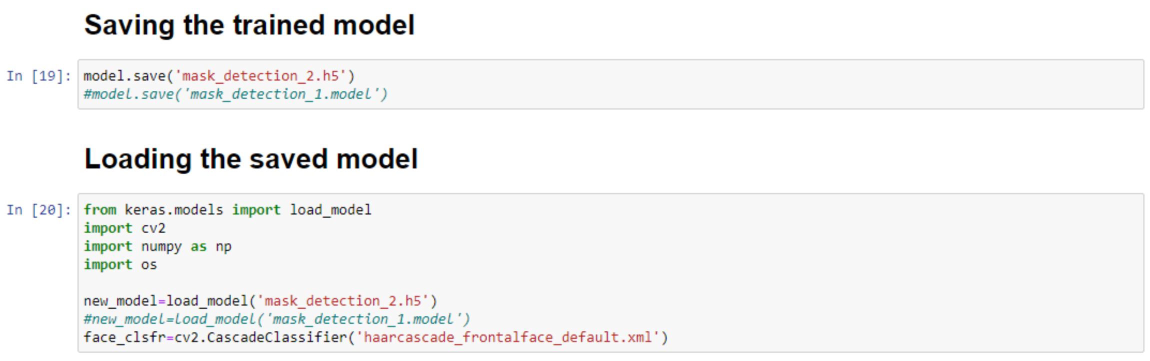
```
In [16]: model.fit_generator(dataGen.flow(train_data,train_target,batch_size=20),epochs=15)

<ipython-input-16-a2e0eab02e19>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Pl
ease use `Model.fit`, which supports generators.
  model.fit_generator(dataGen.flow(train_data,train_target,batch_size=20),epochs=15)

Epoch 1/15
62/62 [==============================] - 95s 2s/step - loss: 0.6400 - accuracy: 0.6147
Epoch 2/15
62/62 [==============================] - 93s 1s/step - loss: 0.3693 - accuracy: 0.8465
Epoch 3/15
62/62 [==============================] - 96s 2s/step - loss: 0.2692 - accuracy: 0.8950
Epoch 4/15
62/62 [==============================] - 101s 2s/step - loss: 0.2628 - accuracy: 0.9023
Epoch 5/15
62/62 [==============================] - 103s 2s/step - loss: 0.2686 - accuracy: 0.9015
Epoch 6/15
62/62 [==============================] - 104s 2s/step - loss: 0.1609 - accuracy: 0.9426
Epoch 7/15
62/62 [==============================] - 95s 2s/step - loss: 0.1722 - accuracy: 0.9378
Epoch 8/15
62/62 [==============================] - 95s 2s/step - loss: 0.1308 - accuracy: 0.9540
Epoch 9/15
62/62 [==============================] - 96s 2s/step - loss: 0.1282 - accuracy: 0.9604
Epoch 10/15
62/62 [==============================] - 104s 2s/step - loss: 0.1087 - accuracy: 0.9661
Epoch 11/15
62/62 [==============================] - 100s 2s/step - loss: 0.1224 - accuracy: 0.9588
Epoch 12/15
62/62 [==============================] - 101s 2s/step - loss: 0.1050 - accuracy: 0.9628
Epoch 13/15
62/62 [==============================] - 104s 2s/step - loss: 0.1302 - accuracy: 0.9515
Epoch 14/15
62/62 [==============================] - 100s 2s/step - loss: 0.1512 - accuracy: 0.9418
Epoch 15/15
62/62 [==============================] - 101s 2s/step - loss: 0.0951 - accuracy: 0.9661

Out[16]: <keras.callbacks.History at 0x23ad0042af0>
```

The model is fit using 15 epochs with accuracy of approximate 96%.



```
In [17]: print(model.evaluate(test_data,test_target))

5/5 [==============================] - 3s 621ms/step - loss: 0.0900 - accuracy: 0.9783
[0.08997419476509094, 0.9782608747482 3]
```

Here the model is evaluated with testing accuracy of 97%.



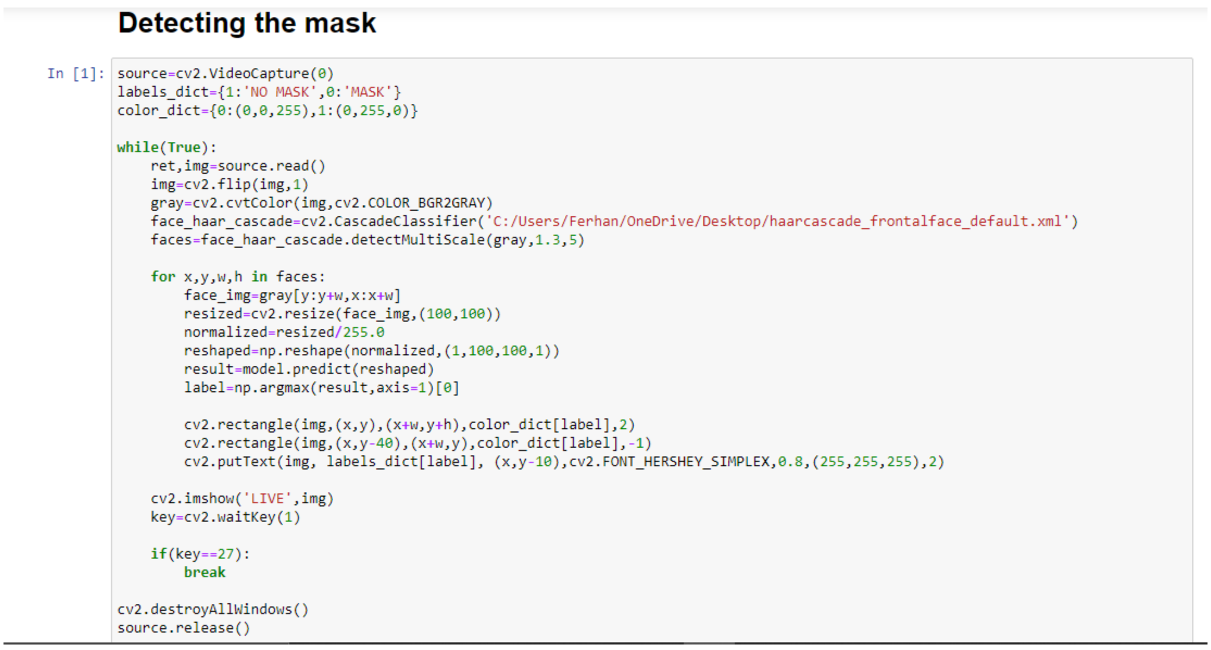
Saving the trained model

```
In [19]: model.save('mask_detection_2.h5')
         #model.save('mask_detection_1.model')
```

Loading the saved model

```
In [20]: from keras.models import load_model
         import cv2
         import numpy as np
         import os

         new_model=load_model('mask_detection_2.h5')
         #new_model=load_model('mask_detection_1.model')
         face_clsfr=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

The model is saved and loaded also the haarcascade classifier is brought into use which will help in detecting faces.



Detecting the mask

In [1]:

```
source=cv2.VideoCapture(0)
labels_dict={1:'NO MASK',0:'MASK'}
color_dict={0:(0,0,255),1:(0,255,0)}

while(True):
    ret,img=source.read()
    img=cv2.flip(img,1)
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    face_haar_cascade=cv2.CascadeClassifier('C:/Users/Ferhan/OneDrive/Desktop/haarcascade_frontalface_default.xml')
    faces=face_haar_cascade.detectMultiScale(gray,1.3,5)

    for x,y,w,h in faces:
        face_img=gray[y:y+w,x:x+w]
        resized=cv2.resize(face_img,(100,100))
        normalized=resized/255.0
        reshaped=np.reshape(normalized,(1,100,100,1))
        result=model.predict(reshaped)
        label=np.argmax(result,axis=1)[0]

        cv2.rectangle(img,(x,y),(x+w,y+h),color_dict[label],2)
        cv2.rectangle(img,(x,y-40),(x+w,y),color_dict[label],-1)
        cv2.putText(img, labels_dict[label], (x,y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)

    cv2.imshow('LIVE',img)
    key=cv2.waitKey(1)

    if(key==27):
        break

cv2.destroyAllWindows()
source.release()
```

- The video capture zero is used (default webcam).
- From the array ‘labels_dict’ the neural network will give 0 if the face is with mask and 1 if the face is without mask in the detected image.
- The rectangle bounding masked image will have green colour and the rectangle bounding the without mask image will have red colour.

In the while loop:

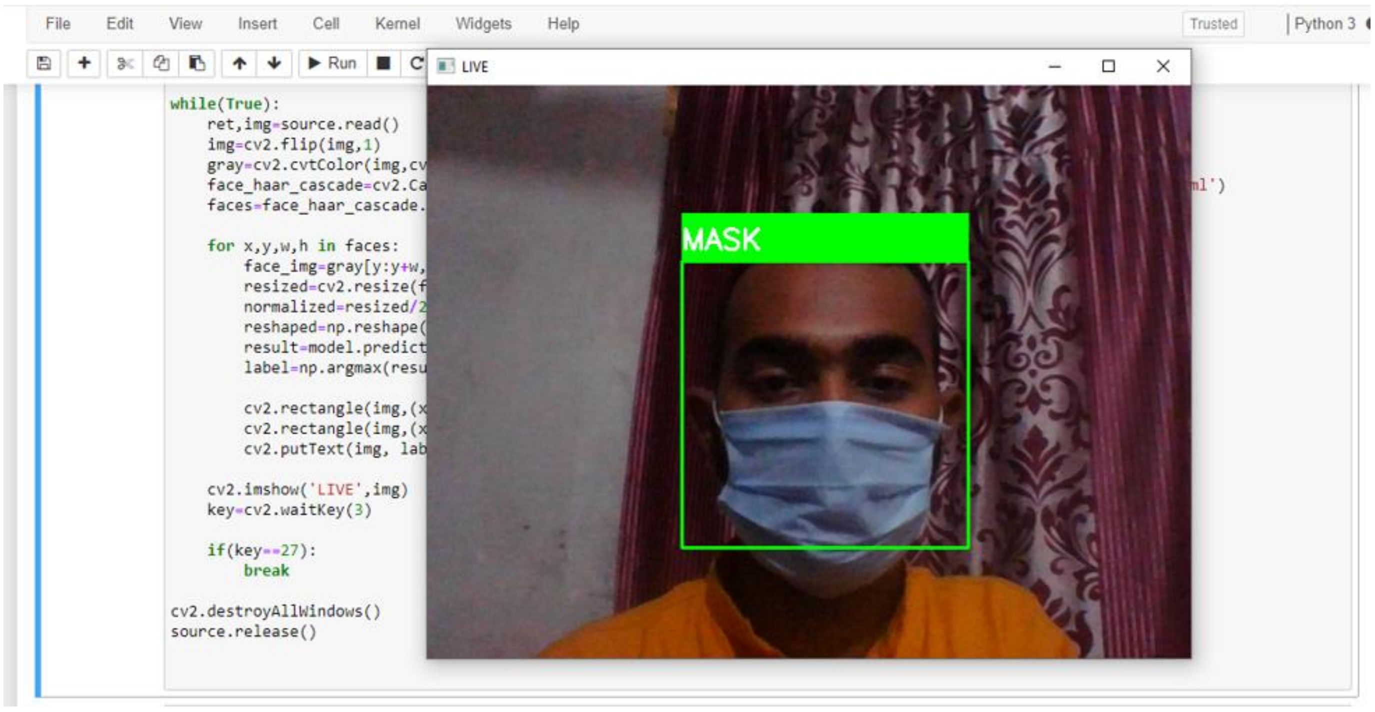
- reading the data frame by frame from the
- camera converting into grey scale detecting

faces In. the for loop

- for each and every face we are cropping the region of interest.
- Saving to resized, normalization, reshaping to 4-d.
- The trained model is used for prediction by passing ‘reshaped’ as argument and the output is saved
  to the result.
- The result will be actually giving probabilities, [[P1][P2]], It will be having 2 colums and 1 row.
- We are using argmax function with axis as 1, so that it will return the column indexes. 0 for with mask and 1 for without mask.
- Then a bounding rectangle is created bounding the face.
- Also a closed rectangle is created at the top of the bounding rectangle.
- Using the putText() we will be displaying ‘MASK’ and ‘NO MASK’ from the labels_dict on the video feed.

Outside the for loop, image is shown and waited for 1 second and if the user is pressing the ESC key then the while loop terminates.

Output:



```
while(True):
    ret,img=source.read()
    img=cv2.flip(img,1)
    gray=cv2.cvtColor(img,cv
    face_haar_cascade=cv2.Ca                                                              ml')
    faces=face_haar_cascade.

    for x,y,w,h in faces:
        face_img=gray[y:y+w,
        resized=cv2.resize(f
        normalized=resized/2
        reshaped=np.reshape(
        result=model.predict
        label=np.argmax(resu

        cv2.rectangle(img,(x
        cv2.rectangle(img,(x
        cv2.putText(img, lab

    cv2.imshow('LIVE',img)
    key=cv2.waitKey(3)

    if(key==27):
        break

cv2.destroyAllWindows()
source.release()
```

Github Link - https://github.com/mdferhankhan/INT248-PROJECT-CA1