`

DIGITAL ASSIGNMENT -1
FACE MASK RECOGNITION - SURVEILLANCE

NAME: RUFUS SAM JOHN IMMANVEL .J
REG NO : 19BLC1031

STEPS TO SEE CODE WORKING :

1. **OPEN JUPYTER NOTEBOOK**.
2. RUN THE FIRST THREE BLOCKS OF CODE.
3. THE DATASET WILL BE TRAINED AND VALIDATED.
4. RUN THE LAST BLOCK **, WEBCAM WILL BE OPENED.**
5. THEN IT WILL CHECK IF MASK IS WORN OR NOT.
6. FINALLY, **TO STOP** THE CODE. **PRESS ESC**

THEORY :

Which problem I am trying to solve:

The primary way the coronavirus spreads is from person to person by respiratory droplets produced when an infected person coughs, sneezes or talks. Face masks, however, can block these droplets. They act as a barrier to keep virus-containing particles from escaping an infected individual and landing on another person. Hence for this ,a constant surveillance has to be done by the police dept to ensure that everyone is wearing a mask when they are in public.

Solution:

So the best way to do is by using ***Face Mask Detection*** in the cctv cameras. Therefore it would reduce the risk of getting affected.

How it works:

By **feeding good image data** and training the algorithm to find the diff b/w wearing and not wearing a mask.

By using **opencv** to trace out the faces and informing whether he/she is wearing a mask.

This is a real time application and can be used in a large scale effectively.

Algorithm:

In [1]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense
#Inbuilt Sequential model => uses linear stack of layers for training and predicting.
cnn = Sequential([Conv2D(filters=100, kernel_size=(3,3), activation='relu'),
                  MaxPooling2D(pool_size=(2,2)),
                  Conv2D(filters=100, kernel_size=(3,3), activation='relu'),
                  MaxPooling2D(pool_size=(2,2)),
                  Flatten(),
                  Dropout(0.5),
                  Dense(50),
                  Dense(35),
                  Dense(2)])

cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

In [2]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

training_directory = "training" #training data
testing_directory = "test"  #test data

# ImageDataGenerator() :
#1 Take a batch of images used for training.
#2 Apply random transformations to each image in the batch.
#3 Replacing the original batch of images with a new randomly transformed batch.
#4 Train a Deep Learning model on this transformed batch
training_data_generator = ImageDataGenerator()
testing_data_generator = ImageDataGenerator()
training_generator = training_data_generator.flow_from_directory(training_directory, batch_size=10, target_size=(150, 150))
testing_generator = testing_data_generator.flow_from_directory(testing_directory, batch_size=10, target_size=(150, 150))
```

```
Found 1014 images belonging to 2 classes.
Found 255 images belonging to 2 classes.
```

In [3]:

```
#training the model using training dataset
epochs = cnn.fit(training_generator, epochs=1, validation_data=testing_generator)
```

```
102/102 [==============================] - 37s 361ms/step - loss: 7.9942 - acc: 0.4803 - val_loss: 7.9993 - val_acc: 0.4784
```

In [4]:

```
import cv2
import numpy as np

labels_dict={0:'No mask'} # To print out the labels
color_dict={0:(255,0,0)} # To give color to border
imgsize = 4 #set image resize
camera = cv2.VideoCapture(0) #use camera
#identify front face
classifier = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
while True:
    (rval, im) = camera.read() #get input from front cam
    im=cv2.flip(im,1,1) #mirrow the image
    imgs = cv2.resize(im, (im.shape[1] // imgsize, im.shape[0] // imgsize)) #resize the image by given factor
    face_rec = classifier.detectMultiScale(imgs)#detect faces multi scale
    for i in face_rec:#extract face rectgular
        (x, y, l, w) = [v * imgsize for v in i] #Scale the shapesize backup
        #Save just the rectangle faces in SubRecFaces
        face_img = im[y:y+w, x:x+l]
        #resizing the image by giving height,width=150
        resized=cv2.resize(face_img,(150,150))
        #normalize by a factor 255
        normalized=resized/255.0
        #reshape by also giving x,y co-ordinates => 1,3
        reshaped=np.reshape(normalized,(1,150,150,3))
        #making it a stack using  vstack as cnn accepts a stack to predict
        reshaped = np.vstack([reshaped])
        #using cnn to predict
        result=cnn.predict(reshaped)
        #Calling the 0 index for both label_dict and color_dict
        label=np.argmax(result,axis=1)[0]
        #making a rectangular area for face
        cv2.rectangle(im,(x,y),(x+l,y+w),color_dict[label],1)
        #making a rectangular area for outline border
        cv2.rectangle(im,(x,y-40),(x+l,y),color_dict[label],-1)
        #Giving desired font-style for the label
        cv2.putText(im, labels_dict[label], (x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)
    #Opening webcam
    cv2.imshow('LIVE',im)
    #Accepting key pressed and process in 10 sec
    key = cv2.waitKey(10)
    # stop loop by ESC
    if key == 27: #The Esc key
        break
#webcam stops accepting data
camera.release()
#destroys the webcam window
cv2.destroyAllWindows()
```

Code:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense
#Inbuilt Sequential model => uses linear stack of layers for training and predicting.
cnn = Sequential([Conv2D(filters=100, kernel_size=(3,3), activation='relu'),
                  MaxPooling2D(pool_size=(2,2)),
                  Conv2D(filters=100, kernel_size=(3,3), activation='relu'),
                  MaxPooling2D(pool_size=(2,2)),
                  Flatten(),
                  Dropout(0.5),
                  Dense(50),
                  Dense(35),
                  Dense(2)])

cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

training_directory = "training" #training data
testing_directory = "test"  #test data

# ImageDataGenerator() :
#1 Take a batch of images used for training.
#2 Apply random transformations to each image in the batch.
#3 Replacing the original batch of images with a new randomly transformed batch.
#4 Train a Deep Learning model on this transformed batch
training_data_generator = ImageDataGenerator()
testing_data_generator = ImageDataGenerator()
training_generator = training_data_generator.flow_from_directory(training_directory,
batch_size=10, target_size=(150, 150))
testing_generator = testing_data_generator.flow_from_directory(testing_directory,
batch_size=10, target_size=(150, 150))

#training the model using training dataset
epochs = cnn.fit(training_generator, epochs=1, validation_data=testing_generator)

import cv2
import numpy as np

labels_dict={0:'No mask'} # To print out the labels
color_dict={0:(255,0,0)} # To give color to border
imgsize = 4 #set image resize
camera = cv2.VideoCapture(0) #use camera
#identify front face
classifier = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
while True:
    (rval, im) = camera.read() #get input from front cam
    im=cv2.flip(im,1,1) #mirrow the image
    imgs = cv2.resize(im, (im.shape[1] // imgsize, im.shape[0] // imgsize)) #resize the image by given
factor
    face_rec = classifier.detectMultiScale(imgs)#detect faces multi scale
    for i in face_rec:#extract face rectgular
        (x, y, l, w) = [v * imgsize for v in i] #Scale the shapesize backup
        #Save just the rectangle faces in SubRecFaces
        face_img = im[y:y+w, x:x+l]
        #resizing the image by giving height,width=150
        resized=cv2.resize(face_img,(150,150))
        #normalize by a factor 255
        normalized=resized/255.0
        #reshape by also giving x,y co-ordinates => 1,3
        reshaped=np.reshape(normalized,(1,150,150,3))
        #making it a stack using  vstack as cnn accepts a stack to predict
        reshaped = np.vstack([reshaped])
        #using cnn to predict
        result=cnn.predict(reshaped)
        #Calling the 0 index for both label_dict and color_dict
        label=np.argmax(result,axis=1)[0]
        #making a rectangular area for face
        cv2.rectangle(im,(x,y),(x+l,y+w),color_dict[label],1)
        #making a rectangular area for outline border
```

```
        cv2.rectangle(im,(x,y-40),(x+l,y),color_dict[label],-1)
        #Giving desired font-style for the label
        cv2.putText(im, labels_dict[label], (x, y-
10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)
    #Opening webcam
    cv2.imshow('LIVE',im)
    #Accepting key pressed and process in 10 sec
    key = cv2.waitKey(10)
    # stop loop by ESC
    if key == 27: #The Esc key
        break
#webcam stops accepting data
camera.release()
#destroys the webcam window
cv2.destroyAllWindows()
```

RESULT:

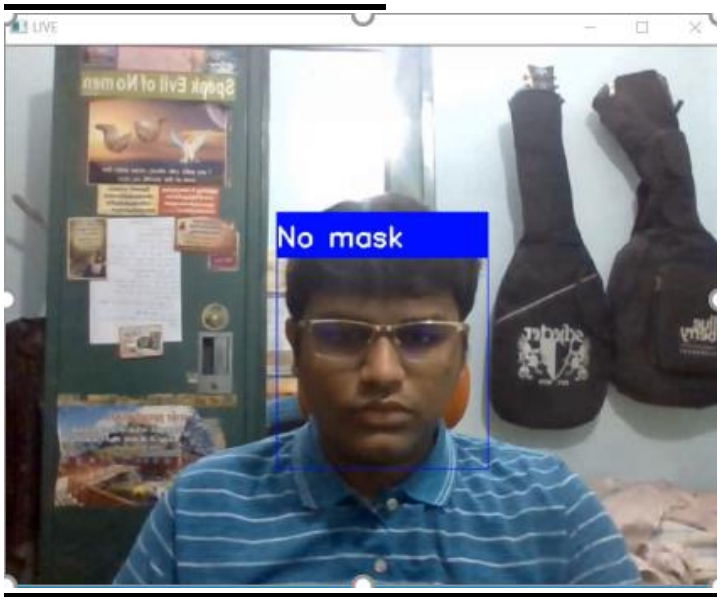
MODEL USED : SEQUENTIAL

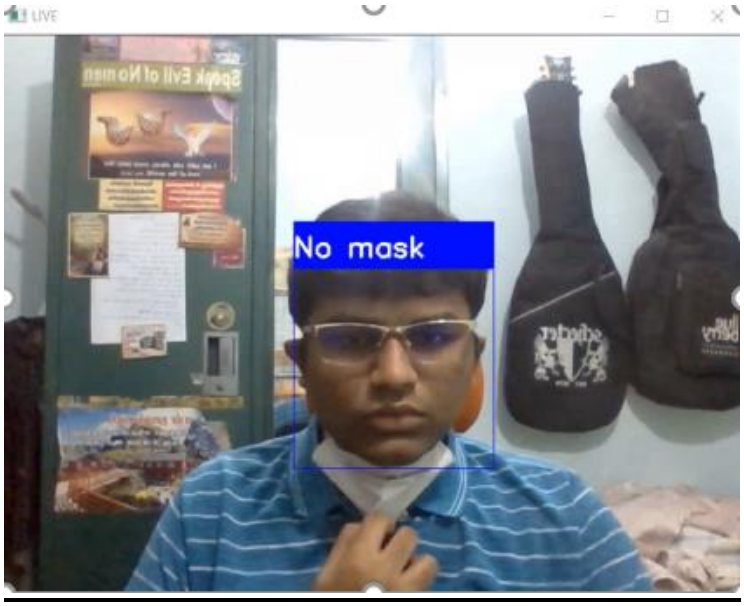
- A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.
- Since one input photo and one output mask recognition is done.
- This one is also simple to use and gives better results when compared with others of the same kind.

INFERENCE:

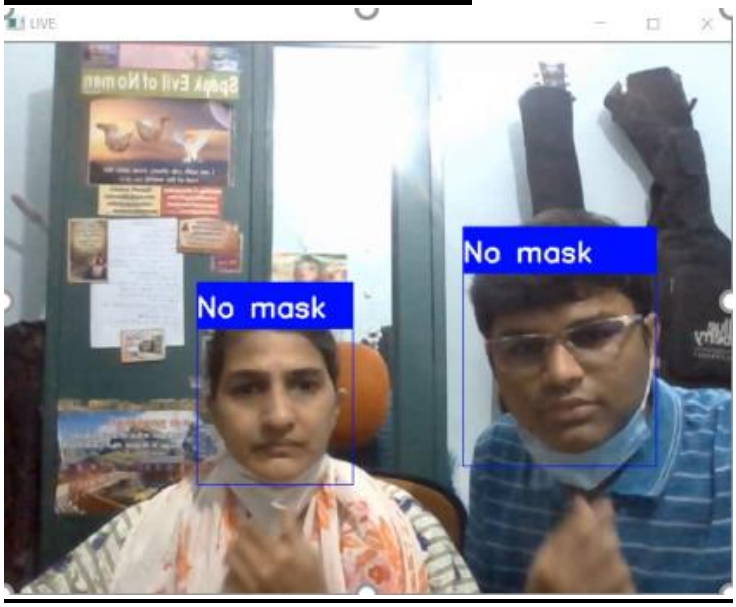
When wearing mask no need to notify.
Only when not wearing mask , the surveillance team must be notified.

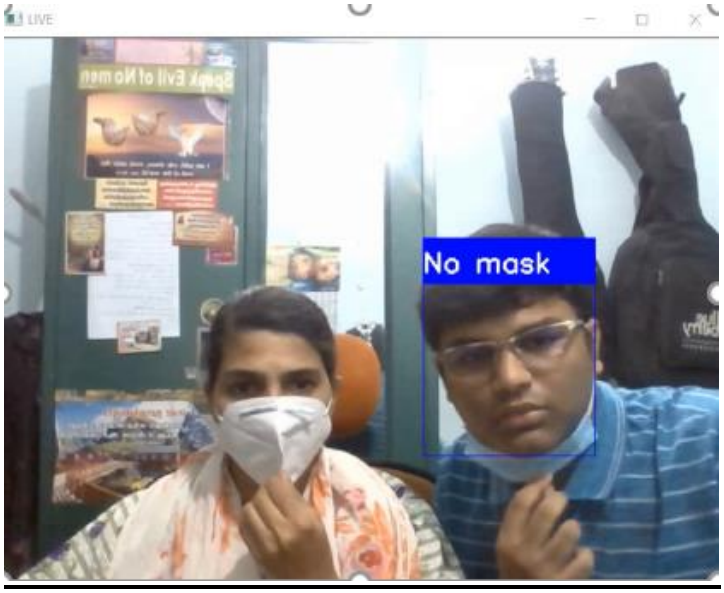
One face check :

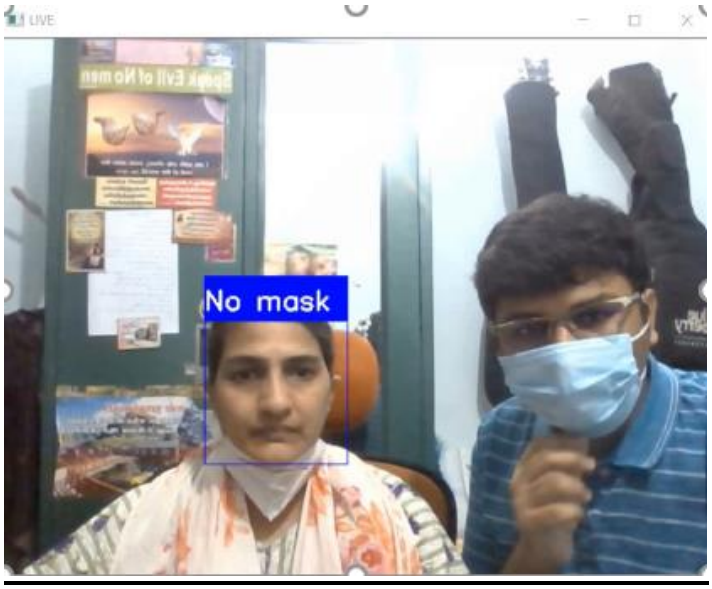




Multiple faces check

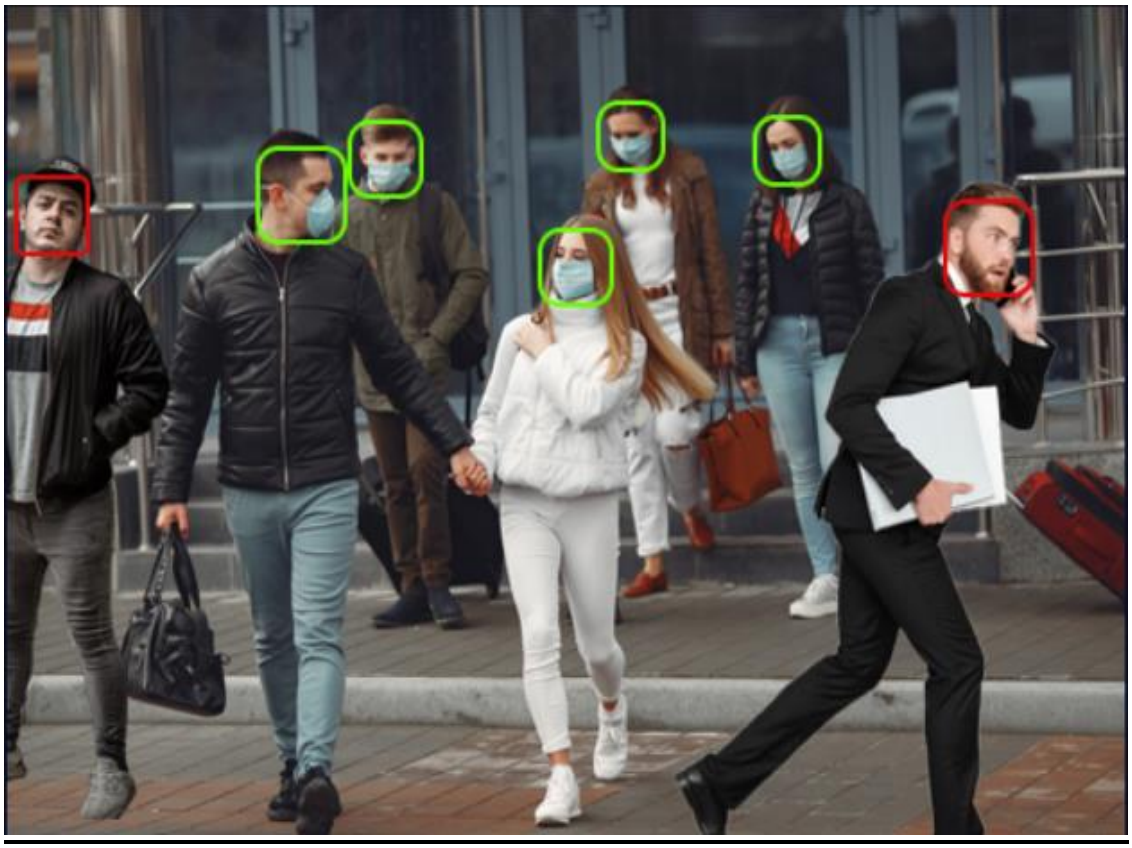


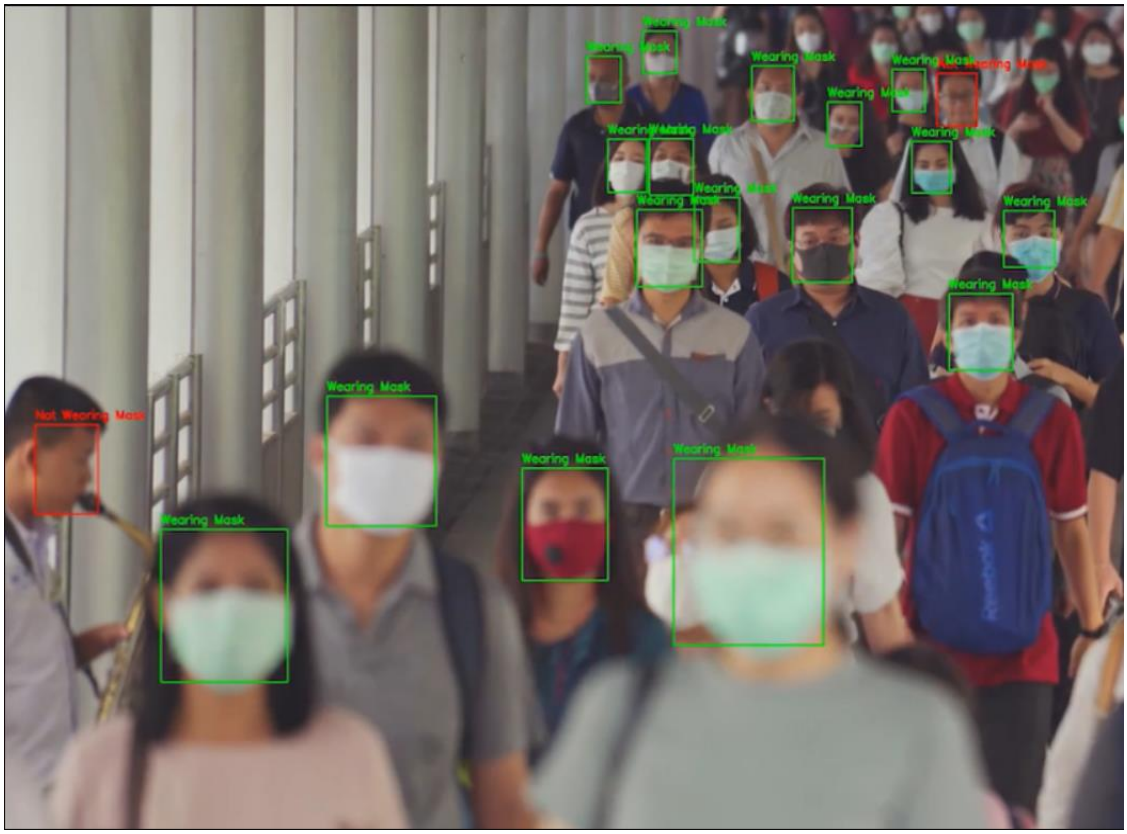




REAL TIME APPLICATION:

- THE PLACE WHERE THIS COULD BE REALLY HELPFUL IS IN THE **SCHOOLS AND COLLEGES** AS THEY ARE OPEN NOW.
- ALSO AS CHILDREN ARE GOING TO SCHOOL, THE SCHOOL IS RESPONSIBLE . THEREFORE THEY CAN MONITOR THE STUDENTS AND HENCE TRY TO REDUCE THE RISK OF ACQUIRING COVID-19.
- ALSO IN **MALLS AND SHOPPING COMPLEXES**




Wearing Mask
Not Wearing Mask
Wearing Mask
Wearing Mask
Wearing Mask
Wearing Mask
Wearing Mask
Wearing Mask
Wearing Mask
Wearing Mask
Wearing Mask
Wearing Mask
Wearing Mask


Input Image Selected Successfully
Mask: 98.74%
Mask: 90.32%