from keras.layers import Input, Dense, Flatten

from keras.models import Model

from keras.applications.vgg16 import VGG16

from keras.applications.vgg16 import preprocess\_input

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

import numpy as np

from keras.applications.vgg16 import preprocess\_input, decode\_predictions

from glob import glob

import tensorflow as tf

import matplotlib.pyplot as plt

***These are the libraries and dependencies required for our Task.***

* **Now lets do some pre variable creations of Image shapes and file paths of our test and train sets.**

IMAGE\_SIZE = [224, 224]

train\_path = 'C:/Users/lenovo/Desktop/mask\_face/Train\_set/'

valid\_path = 'C:/Users/lenovo/Desktop/mask\_face/Test\_set/'

folders = glob(train\_path)

**Assigning this Image Size because VGG16 uses the same image size in its input layer.**

* **Then comes the downloading the weights of Our Pre Trained model "VGG16" from the Internet**

rom keras.layers import Input, Dense, Flatten

from keras.models import Model

from keras.applications.vgg16 import VGG16

from keras.applications.vgg16 import preprocess\_input

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

import numpy as np

from keras.applications.vgg16 import preprocess\_input, decode\_predictions

from glob import glob

import tensorflow as tf

import matplotlib.pyplot as plt

***These are the libraries and dependencies required for our Task.***

* **Now lets do some pre variable creations of Image shapes and file paths of our test and train sets.**

IMAGE\_SIZE = [224, 224]

train\_path = 'C:/Users/lenovo/Desktop/mask\_face/Train\_set/'

valid\_path = 'C:/Users/lenovo/Desktop/mask\_face/Test\_set/'

folders = glob(train\_path)

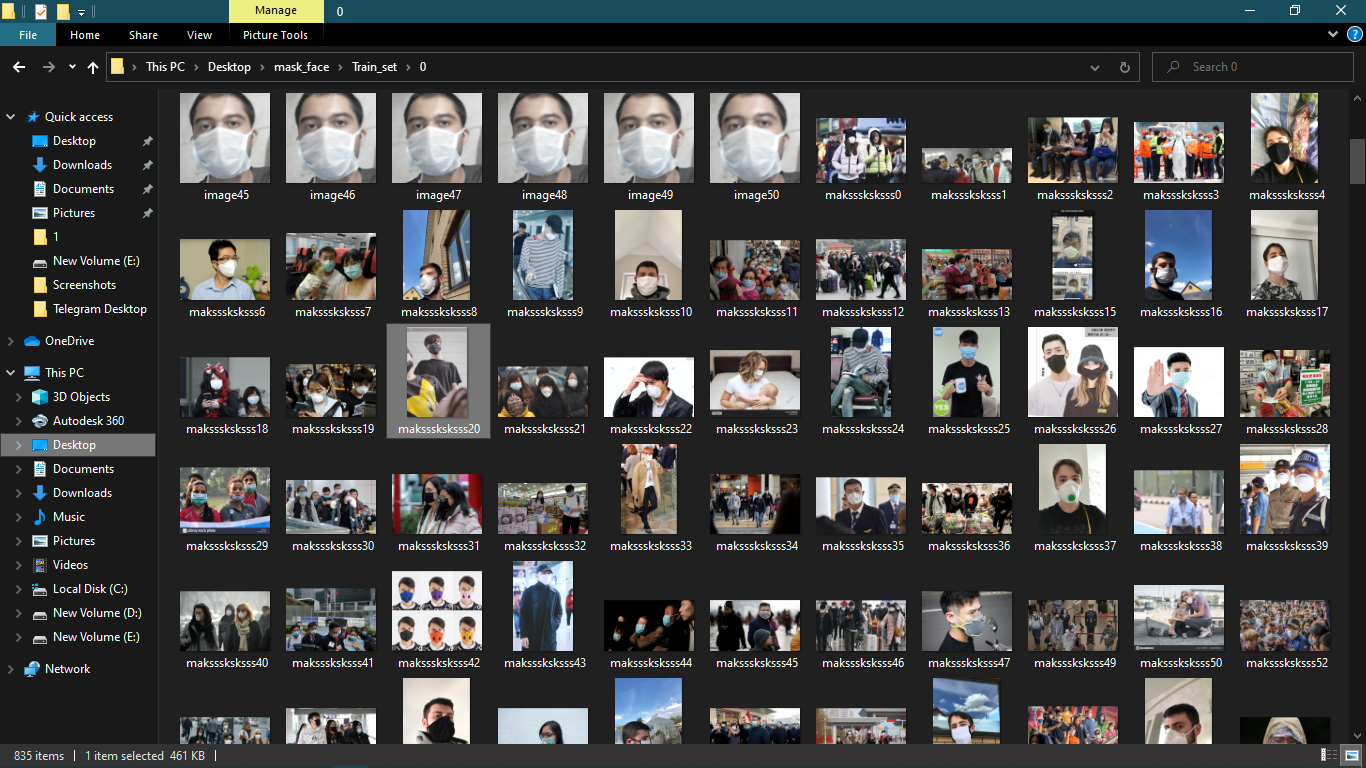
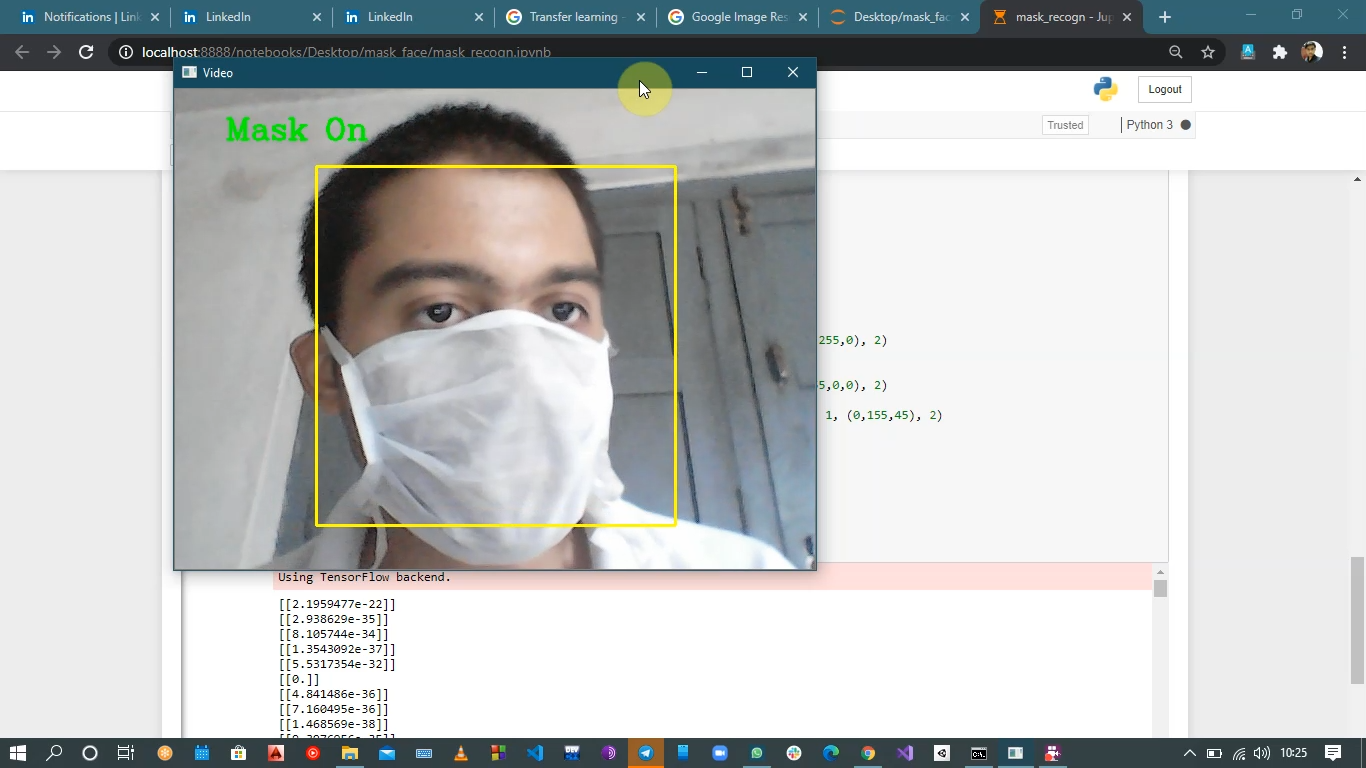
**Assigning this Image Size because VGG16 uses the same image size in its input layer.**

* **Then comes the downloading the weights of Our Pre Trained model "VGG16" from the Internet**

vgg = VGG16(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)

**I have downloaded the weights of model in the variable "vgg" with the image size as assigned before.**

* **Moving ahead.. as the model VGG16 has its own 16 layers, we have to make them not trainable as in transfer learning we just need experience (weights) of the model to be passed not to train the whole model from Zero level through all those layers...It will become a hectic process and also of no use.**

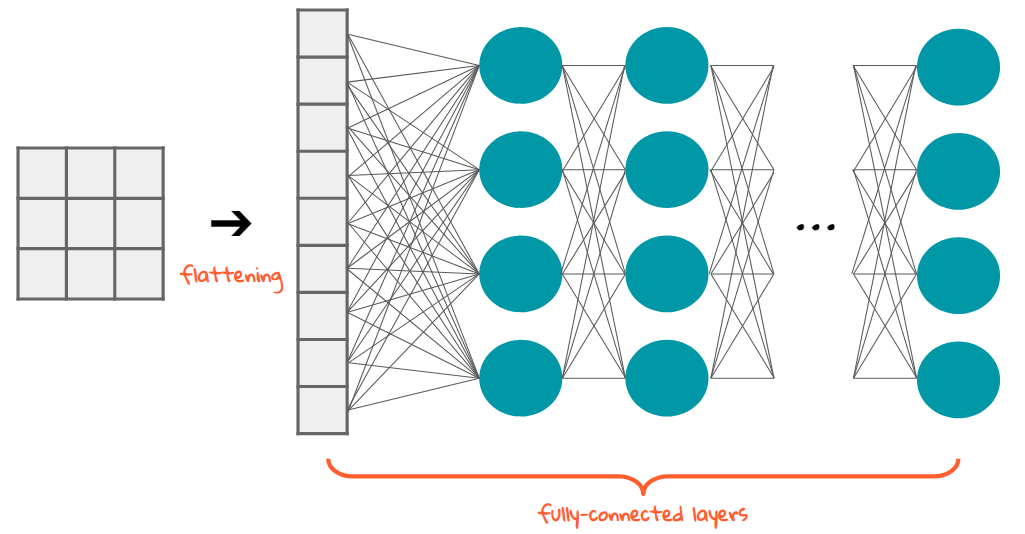


for layer in vgg.layers:

  layer.trainable = False

* **Now lets flatten the last layer of the model as Fully connected layer (which we will create further) takes flattened (1D / vector) as input**

x = Flatten()(vgg.output)



* **Lets create our fully connected layer and our Model**

prediction = Dense(len(folders), activation='sigmoid')(x)

model = Model(inputs=vgg.input, outputs=prediction)

model.summary()

model.compile(

  loss= 'binary\_crossentropy',

  optimizer='adam',

  metrics=['accuracy']

)

**The loss function here is *binary\_crossentropy as our model only have 2 classes.***

* **Now I am doing some Augmentation with my dataset of Images for training the model more efficiently.**

from keras.preprocessing.image import ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale = 1./255,

                                   shear\_range = 0.2,

                                   zoom\_range = 0.2,

                                   horizontal\_flip = True)

test\_datagen = ImageDataGenerator(rescale = 1./255)

training\_set = train\_datagen.flow\_from\_directory(train\_path,

                                                 target\_size = (224, 224),

                                                 batch\_size = 32,

                                                 class\_mode = 'binary')

test\_set = test\_datagen.flow\_from\_directory(valid\_path,

                                            target\_size = (224, 224),

                                            batch\_size = 32,

                                            class\_mode = 'binary')

r = model.fit\_generator(

  training\_set,

  validation\_data=test\_set,

  epochs=7,

  steps\_per\_epoch=len(training\_set),

  validation\_steps=len(test\_set)

)

* **Its obvious class mode will be *binary.***
* **Lets plot the loss and accuracies which we will encounter during the epochs..:-**

# loss

plt.plot(r.history['loss'], label='train loss')

plt.plot(r.history['val\_loss'], label='val loss')

plt.legend()

plt.show()

plt.savefig('LossVal\_loss')

# accuracies

plt.plot(r.history['accuracy'], label='train acc')

plt.plot(r.history['val\_accuracy'], label='val acc')

plt.legend()

plt.show()

plt.savefig('AccVal\_acc')

#saving the model

model.save('C:/Users/lenovo/Desktop/mask\_face/mask\_detection\_model.h5'

***After the model is trained it will be saved with your desired name.***

* **Also you can tune the hyperparameters accordingly to increase the accuracy rate.**
* ***To automate the tuning of Hyper-parameters check out my blog on Integration of***[***Machine Learning with DevOps.***](https://www.linkedin.com/pulse/machine-learning-devops-abhishek-mishra/)***(click on the hyperlink)***

After Training the model You can run it with live webcam feed or any alternative you like for prediction , here is my way of predicting the model by OpenCV live cam feed.

from PIL import Image

from keras.applications.vgg19 import preprocess\_input

import base64

from io import BytesIO

import json

import random

import cv2

from keras.models import load\_model

import numpy as np

from keras.preprocessing import image

model = load\_model('C:/Users/lenovo/Desktop/mask\_face/maskmodel.h5')

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

def face\_extractor(img):

    faces=face\_cascade.detectMultiScale(img, scaleFactor=1.3, minNeighbors=5)

    if faces is ():

        return None

    for (x,y,w,h) in faces:

        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),2)

        cropped\_face = img[y:y+h, x:x+w]

    return cropped\_face

video\_capture = cv2.VideoCapture(0)

while True:

    \_, frame = video\_capture.read()

    face=face\_extractor(frame)

    if type(face) is np.ndarray:

        face = cv2.resize(face, (224, 224))

        im = Image.fromarray(face, 'RGB')

        img\_array = np.array(im)

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

        pred = model.predict(img\_array)

        print(pred)

        name="None matching"

        if(pred[0][0]==0.):

            name='Mask On'

            cv2.putText(frame,name, (50, 50), cv2.FONT\_HERSHEY\_COMPLEX, 1, (0,255,0), 2)

        else:

            name='Mask Not On'

            cv2.putText(frame,name, (50, 50), cv2.FONT\_HERSHEY\_COMPLEX, 1, (255,0,0), 2)

    else:

        cv2.putText(frame,"No face found", (50, 50), cv2.FONT\_HERSHEY\_COMPLEX, 1, (0,155,45), 2)

    cv2.imshow('Video', frame)

    if cv2.waitKey(2)==27:

                break

video\_capture.release()

cv2.destroyAllWindows()