A PROJECT ON

"FACE MASK DETECTION USING MobileNetV3Small"

Submitted to:

KIIT Deemed to be University , Bhubaneswar

In Partial Fulfilment of the Requirement for the

Award of BACHELOR'S DEGREE IN
INFORMATION TECHNOLOGY

INPUT : CODE
LANGUAGE : PYTHON
VERSION : Python 3.9.12
IDE USED : JUPYTER

SUBMITTED BY:

NAME : RUDRASISH MISHRA
ROLL NO : 1906649
SECTION : IT - 8

UNDER THE GUIDANCE OF : DR. DIPAK KUMAR MOHANTY

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY BHUBANESWAR, ODISHA - 751024,May 2020

PROBLEM STATEMENT:

The outspread of the corona virus types has caused pandemic across the globe.The virus is known for severe acute respiratory issues. The ailment it causes is called coronavirus disease 2019 or (COVID-19) affecting our day-to-day life disrupting world trade and movements. Wearing a mask is one of the effective methods to prevent us from being affected by the vicious COVID virus. Due to carelessness and unawareness of the local public towards this issue a facemask detection system can be used by the authorities to keep a check on them and create awareness about the same. This project titled Face Mask Detection has been developed using a machine learning technique known as transfer learning and by using image classification method of MobileNetV3small .Steps followed while building the model are data collection, data preprocessing , data splitting , model-rectification ,model testing and model implementation. Model is capable of predicting if individuals are wearing or not wearing masks at an accuracy of 99.81 percent. We also explored using other image classification models like VGG16 with an accuracy of 96 percent, MobileNetV2 with an accuracy of 97.21 percent and ResNet101 with an accuracy of 97.79 percent to detect masks. We have implemented the Transfer learning method that uses already acquired knowledge by avoiding overfitting of the models.

INPUT CODE:

```
import numpy as np
import os
import matplotlib.pyplot as plt
from imutils import paths
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from          tensorflow.keras.preprocessing.image          import
ImageDataGenerator
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.applications import MobileNetV3Small
From tensorflow.keras.applications.mobilenet_v3 import
preprocess_input
```

```
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model

dataset=r'C:\Users\KIIT\Videos\MobileNetV3\dataset'
image_site=list(paths.list_images(dataset))
image_site

images=[]
category=[]
for a in image_site:
    label=a.split(os.path.sep)[-2]
    category.append(label)
    image=load_img(a,target_size=(224,224))
    image=img_to_array(image)
    image=preprocess_input(image)
    images.append(image)

images=np.array(images,dtype='float32')
category=np.array(category)

Images.shape
category.shape

lb=LabelBinarizer()
category=lb.fit_transform(category)
category=to_categorical(category)
Category

train_X,test_X,train_Y,test_Y=train_test_split(images,category,t
est_size=0.25,stratify=category,random_state=10)

train_X.shape
test_X.shape
train_Y.shape
test_Y.shape

aug=ImageDataGenerator(rotation_range=20,
zoom_range=0.15,
width_shift_range=0.2,
shear_range=0.15,
horizontal_flip=True,
vertical_flip=True)
```

```
baseModel=MobileNetV3Small(
    input_tensor=Input(shape=(224,224,3)),
    alpha=1.0,
    minimalistic=False,
    include_top=False,
    weights="imagenet",
    classes=1000,
    pooling=None,
    include_preprocessing=True,
)

def __init__(mdl):
        super(MobileNetV3Small, mdl).__init__()
        mdl.conv1 = tf.keras.layers.Conv2D(filters=16,
                                           kernel_size=(3, 3),
                                           strides=2,
                                           padding="same")
        mdl.bn1 = tf.keras.layers.BatchNormalization()
              mdl.bneck1  =  BottleNeck(in_size=16,  exp_size=16,
out_size=16, s=2, is_se_existing=True, NL="RE", k=3)
              mdl.bneck2  =  BottleNeck(in_size=16,  exp_size=72,
out_size=24, s=2, is_se_existing=False, NL="RE", k=3)
              mdl.bneck3  =  BottleNeck(in_size=24,  exp_size=88,
out_size=24, s=1, is_se_existing=False, NL="RE", k=3)
              mdl.bneck4  =  BottleNeck(in_size=24,  exp_size=96,
out_size=40, s=2, is_se_existing=True, NL="HS", k=5)
             mdl.bneck5  =  BottleNeck(in_size=40,  exp_size=240,
out_size=40, s=1, is_se_existing=True, NL="HS", k=5)
             mdl.bneck6  =  BottleNeck(in_size=40,  exp_size=240,
out_size=40, s=1, is_se_existing=True, NL="HS", k=5)
             mdl.bneck7  =  BottleNeck(in_size=40,  exp_size=120,
out_size=48, s=1, is_se_existing=True, NL="HS", k=5)
             mdl.bneck8  =  BottleNeck(in_size=48,  exp_size=144,
out_size=48, s=1, is_se_existing=True, NL="HS", k=5)
             mdl.bneck9  =  BottleNeck(in_size=48,  exp_size=288,
out_size=96, s=2, is_se_existing=True, NL="HS", k=5)
            mdl.bneck10  =  BottleNeck(in_size=96,  exp_size=576,
out_size=96, s=1, is_se_existing=True, NL="HS", k=5)
            mdl.bneck11  =  BottleNeck(in_size=96,  exp_size=576,
out_size=96, s=1, is_se_existing=True, NL="HS", k=5)

        mdl.conv2 = tf.keras.layers.Conv2D(filters=576,
                                           kernel_size=(1, 1),
                                           strides=1,
                                           padding="same")
        mdl.bn2 = tf.keras.layers.BatchNormalization()
```

```
                                                mdl.avgpool      =
tf.keras.layers.AveragePooling2D(pool_size=(7, 7),

strides=1)
        mdl.conv3 = tf.keras.layers.Conv2D(filters=1280,
                                           kernel_size=(1, 1),
                                           strides=1,
                                           padding="same")
        mdl.conv4 = tf.keras.layers.Conv2D(filters=NUM_CLASSES,
                                           kernel_size=(1, 1),
                                           strides=1,
                                           padding="same",

activation=tf.keras.activations.softmax)



def call(mdl, inputs, training=None, mask=None):
        x = mdl.conv1(inputs)
        x = mdl.bn1(x, training=training)
        x = h_swish(x)

        x = mdl.bneck1(x, training=training)
        x = mdl.bneck2(x, training=training)
        x = mdl.bneck3(x, training=training)
        x = mdl.bneck4(x, training=training)
        x = mdl.bneck5(x, training=training)
        x = mdl.bneck6(x, training=training)
        x = mdl.bneck7(x, training=training)
        x = mdl.bneck8(x, training=training)
        x = mdl.bneck9(x, training=training)
        x = mdl.bneck10(x, training=training)
        x = mdl.bneck11(x, training=training)

        x = mdl.conv2(x)
        x = mdl.bn2(x, training=training)
        x = h_swish(x)
        x = mdl.avgpool(x)
        x = mdl.conv3(x)
        x = h_swish(x)
        x = mdl.conv4(x)

        return x


baseModel.summary()
```

```
headModel=baseModel.output
headModel=AveragePooling2D(pool_size=(7,7))(headModel)
headModel=Flatten(name='Flatten')(headModel)
headModel=Dense(1280,activation='swish')(headModel)
headModel=Dropout(0.5)(headModel)
headModel=Dense(2, activation='softmax')(headModel)
model=Model(inputs=baseModel.input,outputs=headModel)
model.summary()

learning_rate=0.001
Epochs=20
BS=12

opt=Adam(lr=learning_rate,decay=learning_rate/Epochs)
model.compile(loss='binary_crossentropy',optimizer=opt,metrics=[
'accuracy'])
history=model.fit(

    aug.flow(train_X,train_Y,batch_size=BS),
    steps_per_epoch=len(train_X)//BS,
    validation_data=(test_X,test_Y),
    validation_steps=len(test_X)//BS,
    epochs=Epochs
)


model.save(r'C:\Users\KIIT\Videos\MobileNetV3\mobilenetv3Small.m
odel')

predict=model.predict(test_X,batch_size=BS)
predict=np.argmax(predict,axis=1)
print("CLASSIFICATION REPORT :\n")
print(classification_report(test_Y.argmax(axis=1),predict,target
_names=lb.classes_))

history.history.keys()

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.title('model loss')
```

```
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()

train_datagen = ImageDataGenerator(zoom_range=0.5)
train_generator = train_datagen.flow_from_directory(dataset,
target_size=(224, 224),batch_size=BS, class_mode='categorical',
shuffle=True)

valid_datagen = ImageDataGenerator()
valid_generator=valid_datagen.flow_from_directory(dataset,target
_size=(224,224),batch_size=BS,class_mode='categorical',
shuffle=True)

test_datagen = ImageDataGenerator()

test_generator = test_datagen.flow_from_directory(dataset,
target_size=(224, 224),batch_size=BS,class_mode='categorical',
shuffle=False)

import itertools

def    plot_confusion_matrix(cm,    classes,    normalize=True,
title='Confusion matrix', cmap=plt.cm.Blues):

    plt.figure(figsize=(10,10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)

    plt.title(title)

    plt.colorbar()



    tick_marks = np.arange(len(classes))

    plt.xticks(tick_marks, classes, rotation=45)

    plt.yticks(tick_marks, classes)

    if normalize:

        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

        cm = np.around(cm, decimals=2)
```

```
        cm[np.isnan(cm)] = 0.0

        print("Normalized confusion matrix")

    else:

        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.

fori,      j      in      itertools.product(range(cm.shape[0]),
range(cm.shape[1])):

        plt.text(j, i, cm[i, j],

                 horizontalalignment="center",

                      color="white" if cm[i, j] > thresh else
"black")

    plt.tight_layout()

    plt.ylabel('True label')

    plt.xlabel('Predicted label')

target_names = []

for key in train_generator.class_indices:

    target_names.append(key)


from sklearn.metrics import confusion_matrix

Y_pred = model.predict_generator(test_generator)

y_pred = np.argmax(Y_pred, axis=1)

print('Confusion Matrix')

cm = confusion_matrix(test_generator.classes, y_pred)
plt.figure(figsize=(20, 20))
plot_confusion_matrix(cm,     target_names,     title='Confusion
Matrix')
plt.title('Confusion matrix')
plt.show()
```

MODEL TESTING:

```
from      tensorflow.keras.applications.mobilenet_v2      import
preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import numpy as np
import cv2
import os
from imutils.video import VideoStream
import imutils

def detect_and_predict_mask(frame,faceNet,maskNet):
    (h,w)=frame.shape[:2]

blob=cv2.dnn.blobFromImage(frame,1.0,(300,300),(104.0,177.0,123.
0))

    faceNet.setInput(blob)
    detections=faceNet.forward()

    faces=[]
    locs=[]
    preds=[]

    for i in range(0,detections.shape[2]):
        confidence=detections[0,0,i,2]


        if confidence>0.5:
        #we need the X,Y coordinates
            box=detections[0,0,i,3:7]*np.array([w,h,w,h])
            (startX,startY,endX,endY)=box.astype('int')

            (startX,startY)=(max(0,startX),max(0,startY))
            (endX,endY)=(min(w-1,endX), min(h-1,endY))

            face=frame[startY:endY, startX:endX]
            face=cv2.cvtColor(face,cv2.COLOR_BGR2RGB)
            face=cv2.resize(face,(224,224))
            face=img_to_array(face)
            face=preprocess_input(face)

            faces.append(face)
            locs.append((startX,startY,endX,endY))
        if len(faces)>0:
```

```
            faces=np.array(faces,dtype='float32')
            preds=maskNet.predict(faces,batch_size=12)

        return (locs,preds)

maskNet=load_model(r'C:\Users\KIIT\Videos\Face_Mask_Detection\mo
bilenet_v3.model')
prototxtPath=os.path.sep.join([r'C:\Users\KIIT\Videos\Face_Mask_
Detection\face_detector','deploy.prototxt'])
weightsPath=os.path.sep.join([r'C:\Users\KIIT\Videos\Face_Mask_D
etection\face_detector','res10_300x300_ssd_iter_140000.caffemode
l'])
faceNet=cv2.dnn.readNet(prototxtPath,weightsPath)
vs=VideoStream(src=0).start()

while True:
    frame=vs.read()
    frame=imutils.resize(frame,width=400)

    (locs,preds)=detect_and_predict_mask(frame,faceNet,maskNet)

    for (box,pred) in zip(locs,preds):
        (startX,startY,endX,endY)=box
        (mask,withoutMask)=pred

        label='Mask' if mask>withoutMask else 'No Mask'
        color=(0,255,0) if label=='Mask' else (0,0,255)

cv2.putText(frame,label,(startX,startY-10),cv2.FONT_HERSHEY_SIMP
LEX,0.45,color,2)
        rY=(endY - startY)//2

            cv2.circle(frame,((endX  -  startX)//2+startX,(endY  -
startY)//2+startY),rY,color,3)

    cv2.imshow("Face Mask Detector",frame)
    key=cv2.waitKey(1) & 0xFF

    if key==ord('o'):
        break

cv2.destroyAllWindows()
vs.stop()
#
```

{NAME:RUDRASISH MISHRA} | {SECTION:IT-8} | {ROLL NO:1906649}