**VIVEKANAND EDUCATION SOCIETY’S**

**INSTITUTE OF TECHNOLOGY**

  
  
**Department of Computer Engineering**

**DIGITAL SIGNAL IMAGE PROCESSING**

**Mini Project Report on**

**FACE MASK DETECTION**

**Submitted in partial fulfilment of the requirements**

**of FOURTH Year Computer Engineering**

**By**

**Rahul Koli (D17B-31)**

**Sahil Lotya (D17B-33)**

**Karan Sachdev (D17B-50)**

**Supervisor: Mrs. Indu Dokare**

**DEPARTMENT OF COMPUTER ENGINEERING**

**V.E.S INSTITUTE OF TECHNOLOGY**

**2021-22**

**FACE MASK DETECTION**

**INTRODUCTION :**

Face detection -- also called facial detection -- is an [artificial intelligence](https://searchenterpriseai.techtarget.com/definition/AI-Artificial-Intelligence) (AI) based computer technology used to find and identify human faces in digital images. Face detection technology can be applied to various fields -- including security, [biometrics](https://searchsecurity.techtarget.com/definition/biometrics), law enforcement, entertainment and personal safety -- to provide surveillance and tracking of people in real time.

Face detection has progressed from rudimentary [computer vision](https://searchenterpriseai.techtarget.com/definition/machine-vision-computer-vision) techniques to advances in machine learning ([ML](https://searchenterpriseai.techtarget.com/definition/machine-learning-ML)) to increasingly sophisticated artificial neural networks ([ANN](https://searchenterpriseai.techtarget.com/definition/neural-network)) and related technologies; the result has been continuous performance improvements. It now plays an important role as the first step in many key applications -- including face tracking, face analysis and [facial recognition](https://searchenterpriseai.techtarget.com/definition/facial-recognition). Face detection has a significant effect on how sequential operations will perform in the application.

Face mask detection refers to detect whether a person is wearing a mask or not. In fact, the problem is reverse engineering of face detection where the face is detected using different machine learning algorithms for the purpose of security, authentication and surveillance. Face detection is a key area in the field of Computer Vision and Pattern Recognition. In recent years, face detection methods based on deep convolutional neural networks (CNN) have been widely developed to improve detection performance.

**Tools Used:**

* Face Net Model : for detection of faces ROI(Region of Interest)
* OpenCV/CV2: To display the label and bounding box rectangle on the output.
* Python Library Stack
  + Tensorflow.keras
  + Sklearn
  + Matplotlib
  + Numpy
  + CV2

**Methodology**:

* Data Preprocessing : Load Face Mask Dataset and converting each image size into 224,224 and then converting them into arrays. Converting Labels into categorical binary variables (0 referring to no mask and 1 referring to mask).
* Training And Testing Split:Splitting the dataset into training and testing at 80% and 20 % rate.
* Data Augmentation: using image Data Generator.
* After preprocessing the data will be passed on to base model :MobileNetV2.Connecting other layers(AveragePooling, Flatten, Dense) and Activation function used is softmax due to binary values.
* Training the model on 1100 images and augmented images for 20 epochs to get desired accuracy.

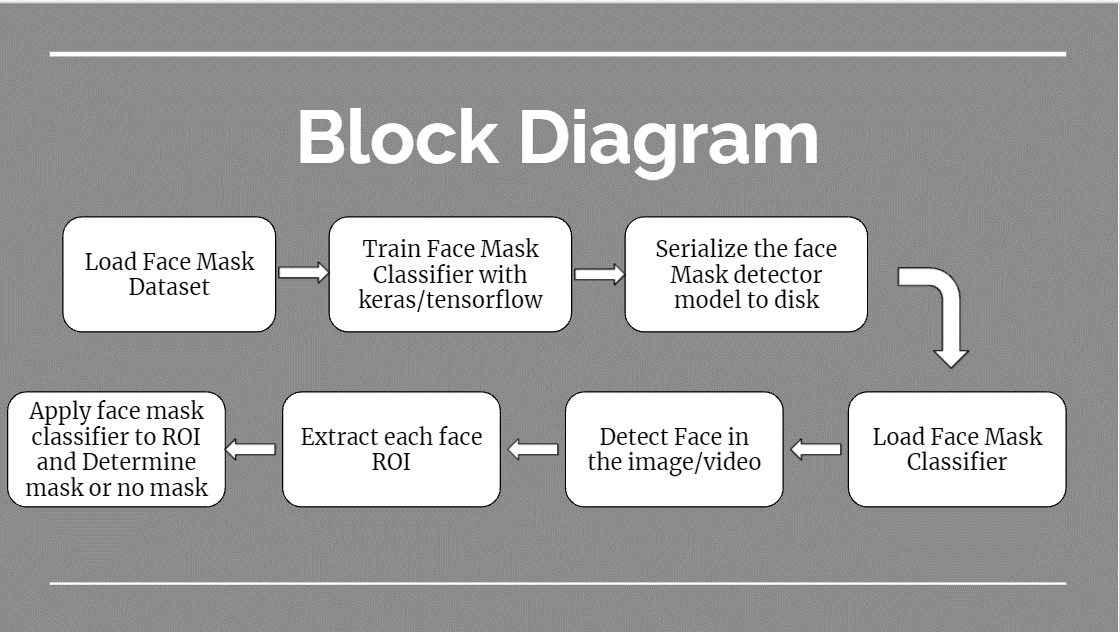


Fig 1.Block Diagram of Face Mask detection

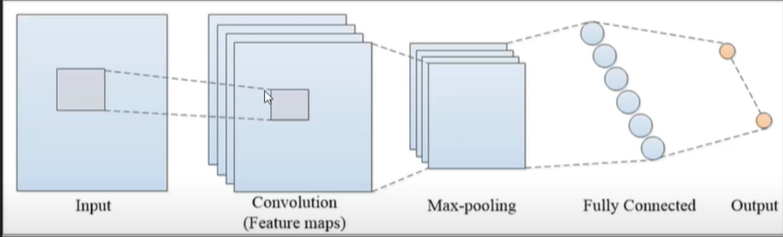


Fig 2. Model structure

**RESULTS :**

Face Mask Detector Model acquired 98% accuracy in validation set and maximum loss of 2 %.



Fig 3. Accuracy and Loss Graph

**OUTPUT:**

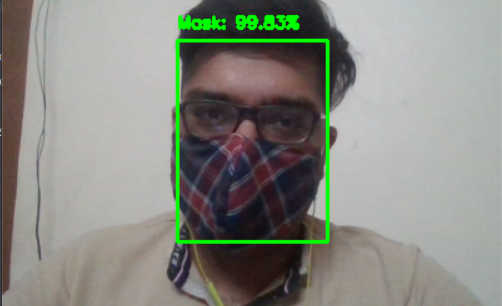
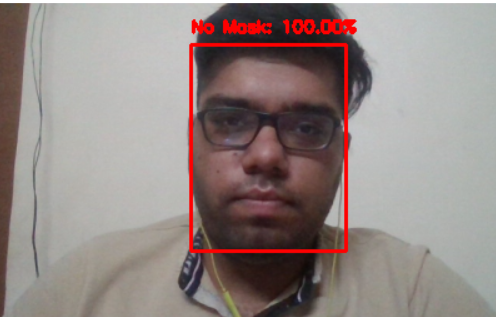


Fig 4. No Mask Output Fig 5. Mask Output

[**CODE (Python)**](https://drive.google.com/file/d/1hTabMz-G-pE8LaBhDDDNCpB3Ac0Df86v/view?usp=sharing)**:**

# import the necessary packages

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2

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

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.applications.mobilenet\_v2 import preprocess\_input

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 imutils import paths

import matplotlib.pyplot as plt

import numpy as np

import os

# initialize the initial learning rate, number of epochs to train for,

# and batch size

INIT\_LR = 1e-4

EPOCHS = 20

BS = 32

DIRECTORY = r"D:\PycharmProjects\DSIPproject\dataset"

CATEGORIES = ["with\_mask", "without\_mask"]

# grab the list of images in our dataset directory, then initialize

# the list of data (i.e., images) and class images

print("[INFO] loading images...")

data = []

labels = []

for category in CATEGORIES:

path = os.path.join(DIRECTORY, category)

for img in os.listdir(path):

img\_path = os.path.join(path, img)

image = load\_img(img\_path, target\_size=(224, 224))

image = img\_to\_array(image)

image = preprocess\_input(image)

data.append(image)

labels.append(category)

# perform one-hot encoding on the labels

lb = LabelBinarizer()

labels = lb.fit\_transform(labels)

labels = to\_categorical(labels)

data = np.array(data, dtype="float32")

labels = np.array(labels)

(trainX, testX, trainY, testY) = train\_test\_split(data, labels,

test\_size=0.20, stratify=labels, random\_state=42)

# construct the training image generator for data augmentation

aug = ImageDataGenerator(

rotation\_range=20,

zoom\_range=0.15,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.15,

horizontal\_flip=True,

fill\_mode="nearest")

# load the MobileNetV2 network, ensuring the head FC layer sets are

# left off

baseModel = MobileNetV2(weights="imagenet", include\_top=False,

input\_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the

# the base model

headModel = baseModel.output

headModel = AveragePooling2D(pool\_size=(7, 7))(headModel)

headModel = Flatten(name="flatten")(headModel)

headModel = Dense(128, activation="relu")(headModel)

headModel = Dropout(0.5)(headModel)

headModel = Dense(2, activation="softmax")(headModel)

# place the head FC model on top of the base model (this will become

# the actual model we will train)

model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so they will

# \*not\* be updated during the first training process

for layer in baseModel.layers:

layer.trainable = False

# compile our model

print("[INFO] compiling model...")

opt = Adam(lr=INIT\_LR, decay=INIT\_LR / EPOCHS)

model.compile(loss="binary\_crossentropy", optimizer=opt,

metrics=["accuracy"])

# train the head of the network

print("[INFO] training head...")

H = model.fit(

aug.flow(trainX, trainY, batch\_size=BS),

steps\_per\_epoch=len(trainX) // BS,

validation\_data=(testX, testY),

validation\_steps=len(testX) // BS,

epochs=EPOCHS)

# make predictions on the testing set

print("[INFO] evaluating network...")

predIdxs = model.predict(testX, batch\_size=BS)

# for each image in the testing set we need to find the index of the

# label with corresponding largest predicted probability

predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report

print(classification\_report(testY.argmax(axis=1), predIdxs,

target\_names=lb.classes\_))

# serialize the model to disk

print("[INFO] saving mask detector model...")

model.save("mask\_detector.model", save\_format="h5")

# plot the training loss and accuracy

N = EPOCHS

plt.style.use("ggplot")

plt.figure()

plt.plot(np.arange(0, N), H.history["loss"], label="train\_loss")

plt.plot(np.arange(0, N), H.history["val\_loss"], label="val\_loss")

plt.plot(np.arange(0, N), H.history["accuracy"], label="train\_acc")

plt.plot(np.arange(0, N), H.history["val\_accuracy"], label="val\_acc")

plt.title("Training Loss and Accuracy")

plt.xlabel("Epoch #")

plt.ylabel("Loss/Accuracy")

plt.legend(loc="lower left")

plt.savefig("plot.png")

# import the necessary packages

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

from imutils.video import VideoStream

import numpy as np

import imutils

import time

import cv2

import os

def detect\_and\_predict\_mask(frame, faceNet, maskNet):

# grab the dimensions of the frame and then construct a blob

# from it

(h, w) = frame.shape[:2]

blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),

(104.0, 177.0, 123.0))

# pass the blob through the network and obtain the face detections

faceNet.setInput(blob)

detections = faceNet.forward()

print(detections.shape)

# initialize our list of faces, their corresponding locations,

# and the list of predictions from our face mask network

faces = []

locs = []

preds = []

# loop over the detections

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

# extract the confidence (i.e., probability) associated with

# the detection

confidence = detections[0, 0, i, 2]

# filter out weak detections by ensuring the confidence is

# greater than the minimum confidence

if confidence > 0.5:

# compute the (x, y)-coordinates of the bounding box for

# the object

box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])

(startX, startY, endX, endY) = box.astype("int")

# ensure the bounding boxes fall within the dimensions of

# the frame

(startX, startY) = (max(0, startX), max(0, startY))

(endX, endY) = (min(w - 1, endX), min(h - 1, endY))

# extract the face ROI, convert it from BGR to RGB channel

# ordering, resize it to 224x224, and preprocess it

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)

# add the face and bounding boxes to their respective

# lists

faces.append(face)

locs.append((startX, startY, endX, endY))

# only make a predictions if at least one face was detected

if len(faces) > 0:

# for faster inference we'll make batch predictions on \*all\*

# faces at the same time rather than one-by-one predictions

# in the above `for` loop

faces = np.array(faces, dtype="float32")

preds = maskNet.predict(faces, batch\_size=32)

# return a 2-tuple of the face locations and their corresponding

# locations

return (locs, preds)

# load our serialized face detector model from disk

prototxtPath = r"face\_detector\deploy.prototxt"

weightsPath = r"face\_detector\res10\_300x300\_ssd\_iter\_140000.caffemodel"

faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk

maskNet = load\_model("mask\_detector.model")

# initialize the video stream

print("[INFO] starting video stream...")

vs = VideoStream(src=0).start()

# loop over the frames from the video stream

while True:

# grab the frame from the threaded video stream and resize it

# to have a maximum width of 400 pixels

frame = vs.read()

frame = imutils.resize(frame, width=400)

# detect faces in the frame and determine if they are wearing a

# face mask or not

(locs, preds) = detect\_and\_predict\_mask(frame, faceNet, maskNet)

# loop over the detected face locations and their corresponding

# locations

for (box, pred) in zip(locs, preds):

# unpack the bounding box and predictions

(startX, startY, endX, endY) = box

(mask, withoutMask) = pred

# determine the class label and color we'll use to draw

# the bounding box and text

label = "Mask" if mask > withoutMask else "No Mask"

color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

# include the probability in the label

label = "{}: {:.2f}%".format(label, max(mask, withoutMask) \* 100)

# display the label and bounding box rectangle on the output

# frame

cv2.putText(frame, label, (startX, startY - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, color, 2)

cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

# show the output frame

cv2.imshow("Frame", frame)

key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from the loop

if key == ord("q"):

break

# do a bit of cleanup

cv2.destroyAllWindows()

vs.stop()

**CONCLUSION :**

Face Detection is the base for other applications like facial expression recognition, face recognition, face mask detection, determining age from image, determining gender from the image etc.

**Some of possible difficulties:**

1. The orientation of the image

2. Blurred image

3. Lighting conditions

4. Entire face should be present in the image, partial faces aren’t detected.

**REFERENCES:**

1. A. Das, M. Wasif Ansari and R. Basak, "Covid-19 Face Mask Detection Using TensorFlow, Keras and OpenCV," 2020 IEEE 17th India Council International Conference (INDICON), 2020, pp. 1-5, doi: 10.1109/INDICON49873.2020.9342585.

2. Shilpa Sethi, Mamta Kathuria, Trilok Kaushik,Face mask detection using deep learning: An approach to reduce risk of Coronavirus spread,Journal of Biomedical Informatics,Volume 120,2021,103848,ISSN 1532-0464, <https://doi.org/10.1016/j.jbi.2021.103848> .

3. S. Susanto, F. A. Putra, R. Analia and I. K. L. N. Suciningtyas, "The Face Mask Detection For Preventing the Spread of COVID-19 at Politeknik Negeri Batam," 2020 3rd International Conference on Applied Engineering (ICAE), 2020, pp. 1-5, doi: 10.1109/ICAE50557.2020.9350556.

4. Steffen E. Eikenberry, Marina Mancuso, Enahoro Iboi, Tin Phan, Keenan Eikenberry, Yang Kuang, Eric Kostelich, Abba B. Gumel, To mask or not to mask: Modeling the potential for face mask use by the general public to curtail the COVID-19 pandemic, Infectious Disease Modelling, Volume 5, 2020, Pages 293-308, ISSN 2468-0427, https://doi.org/10.1016/j.idm.2020.04.001.

# 5. Safa Teboulbi, Seifeddine Messaoud, Mohamed Ali Hajjaji, Abdellatif Mtibaa, "Real-Time Implementation of AI-Based Face Mask Detection and Social Distancing Measuring System for COVID-19 Prevention", Scientific Programming, vol. 2021, Article ID 8340779, 21 pages, 2021. https://doi.org/10.1155/2021/8340779