Real time Face Mask Detection

Blog by Yash R. Shiyani, Intern at MSRF

In this article, we are going to find out how to detect facemasks in real-time .After detecting the face from the webcam stream, we are going to pass these frames (real time) to our mask detector classifier to find out if the person is wearing a mask or not.

Here is the list of subtopics we are going to cover:

1. What is Face Detection?
2. Face Detection Methods
3. Face Detection Algorithm
4. Face recognition
5. Face Detection Using Python
6. Face Detection Using OpenCV
7. Create a model to recognise faces wearing a mask
8. How to do Real-time Mask Detection

What is Face Detection?

The goal of face detection is to determine if there are any faces in the image or video. If multiple faces are present, each face is enclosed by a bounding box and thus we know the location of the faces

Human faces are difficult to model as there are many variables that can change for example facial expression, orientation, lighting conditions and partial occlusions such as sunglasses, scarf, mask etc. The result of the detection gives the face location parameters and it could be required in various forms, for instance, a rectangle covering the central part of the face, eye centers or landmarks including eyes, nose and mouth corners, eyebrows, nostrils, etc.

Face Detection Methods

There are two main approaches for Face Detection:

1. Feature Base Approach
2. Image Base Approach

Feature Base Approach

Objects are usually recognized by their unique features. There are many features in a human face, which can be recognized between a face and many other objects. It locates faces by extracting structural features like eyes, nose, mouth etc. and then uses them to detect a face. Typically, some sort of statistical classifier qualified then helpful to separate between facial and non-facial regions. In addition, human faces have particular textures which can be used to differentiate between a face and other objects. Moreover, the edge of features can help to detect the objects from the face. In the coming section, we will implement a feature-based approach by using OpenCV.

Image Base Approach

In general, Image-based methods rely on techniques from statistical analysis and machine learning to find the relevant characteristics of face and non-face images. The learned characteristics are in the form of distribution models or discriminant functions that is consequently used for face detection. In this method, we use different algorithms such as Neural-networks, HMM, SVM, AdaBoost learning. In the coming section, we will see how we can detect faces with MTCNN or Multi-Task Cascaded Convolutional Neural Network, which is an Image-based approach of face detection.

Face detection algorithm

One of the popular algorithms that use a feature-based approach is the Viola-Jones algorithm and here I am briefly going to discuss it. If you want to know about it in detail, I would suggest going through this article, Face Detection Using Viola Jones Algorithm.

Viola-Jones algorithm is named after two computer vision researchers who proposed the method in 2001, Paul **Viola** and Michael **Jones** in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features”. Despite being an outdated framework, Viola-Jones is quite powerful, and its application has proven to be exceptionally notable in real-time face

detection. This algorithm is painfully slow to train but can detect faces in real-time with impressive speed.

Given an image (this algorithm works on grayscale image), the algorithm looks at many smaller subregions and tries to find a face by looking for specific features in each subregion. It needs to check many different positions and scales because an image can contain many faces of various sizes. Viola and Jones used Haar-like features to detect faces in this algorithm.

Face Recognition

Face detection and Face Recognition are often used interchangeably but these are quite different. In fact, Face detection is just part of Face Recognition.

Face recognition is a method of identifying or verifying the identity of an individual using their face. There are various algorithms that can do face recognition but their accuracy might vary. Here I am going to describe how we do face recognition using deep learning.

Face Detection using Python

As mentioned before, here we are going to see how we can detect faces by using an Image-based approach. MTCNN or Multi-Task Cascaded Convolutional Neural Network is unquestionably one of the most popular and most accurate face detection tools that work this principle. As such, it is based on a deep learning architecture, it specifically consists of 3 neural networks (P-Net, R-Net, and O-Net) connected in a cascade.

So, let's see how we can use this algorithm in Python to detect faces in real-time. First, you need to install MTCNN library which contains a trained model that can detect faces.

```
pip install mtcnn
```

Now let us see how to use MTCNN:

```
1	from mtcnn import MTCNN
2	import cv2
3	detector = MTCNN()
```

```
#Load a videopip TensorFlow
video_capture = cv2.VideoCapture(0)

while (True):
    ret, frame = video_capture.read()
    frame = cv2.resize(frame, (600, 400))
    boxes = detector.detect_faces(frame)
    if boxes:

        box = boxes[0]['box']
        conf = boxes[0]['confidence']
        x, y, w, h = box[0], box[1], box[2], box[3]

        if conf > 0.5:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 255), 1)

    cv2.imshow("Frame", frame)
    if cv2.waitKey(25) &amp;amp; 0xFF == ord('q'):
        break

video_capture.release()
cv2.destroyAllWindows()
```

Face Detection using OpenCV

In this section, we are going to use OpenCV to do real-time face detection from a live stream via our webcam.

As you know videos are basically made up of frames, which are still images. We perform the face detection for each frame in a video. So when it comes to detecting a face in still image and detecting a face in a real-time video stream, there is not much difference between them.

We will be using Haar Cascade algorithm, also known as Voila-Jones algorithm to detect faces. It is basically a machine learning object detection algorithm which is used to identify objects in an image or video. In OpenCV, we have several trained Haar Cascade models which are saved as XML files. Instead of creating and training the model from scratch, we use this file. We are going to use “haarcascade_frontalface_alt2.xml” file in this project. Now let us start coding this up.

The first step is to find the path to the “haarcascade_frontalface_alt2.xml” file. We do this by using the os module of Python language.

```
import os

cascPath = os.path.dirname(

    cv2.__file__) + "/data/haarcascade_frontalface_alt2.xml"
```

The next step is to load our classifier. The path to the above XML file goes as an argument to CascadeClassifier() method of OpenCV.

```
faceCascade = cv2.CascadeClassifier(cascPath)
```

After loading the classifier, let us open the webcam using this simple OpenCV one-liner code

```
video_capture = cv2.VideoCapture(0)
```

Next, we need to get the frames from the webcam stream, we do this using the read() function. We use it in infinite loop to get all the frames until the time we want to close the stream.

```
while True:

    # Capture frame-by-frame

    ret, frame = video_capture.read()
```

The read() function returns:

1. The actual video frame read (one frame on each loop)
2. A return code

The return code tells us if we have run out of frames, which will happen if we are reading from a file. This doesn't matter when reading from the webcam since we can record forever, so we will ignore it.

For this specific classifier to work, we need to convert the frame into greyscale.

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

The faceCascade object has a method detectMultiScale(), which receives a frame(image) as an argument and runs the classifier cascade over the image. The term MultiScale indicates that the algorithm looks at subregions of the image in multiple scales, to detect faces of varying sizes.

```
faces = faceCascade.detectMultiScale(gray,
                                     scaleFactor=1.1,
                                     minNeighbors=5,
                                     minSize=(60, 60),
                                     flags=cv2.CASCADE_SCALE_IMAGE)
```

Let us go through these arguments of this function:

- scaleFactor – Parameter specifying how much the image size is reduced at each image scale. By rescaling the input image, you can resize a larger face to a smaller one, making it detectable by the algorithm. 1.05 is a good possible value for this, which means you use a small step for resizing, i.e. reduce the size by 5%, you increase the chance of a matching size with the model for detection is found.
- minNeighbors – Parameter specifying how many neighbours each candidate rectangle should have to retain it. This parameter will affect the quality of the detected faces. Higher value results in fewer detections but with higher quality. 3~6 is a good value for it.
- flags –Mode of operation
- minSize – Minimum possible object size. Objects smaller than that are ignored.

The variable faces now contain all the detections for the target image. Detections are saved as pixel coordinates. Each detection is defined by its top-left corner coordinates and width and height of the rectangle that encompasses the detected face.

To show the detected face, we will draw a rectangle over it.OpenCV's rectangle() draws rectangles over images, and it needs to know the pixel coordinates of the top-left and bottom-right corner. The coordinates indicate the row and column of pixels in the image. We can easily get these coordinates from the variable face.

```
for (x,y,w,h) in faces:
    cv2.rectangle(frame, (x, y), (x + w, y + h),(0,255,0), 2)
```

rectangle() accepts the following arguments:

- The original image
- The coordinates of the top-left point of the detection

- The coordinates of the bottom-right point of the detection
- The colour of the rectangle (a tuple that defines the amount of red, green, and blue (0-255)).In our case, we set as green just keeping the green component as 255 and rest as zero.
- The thickness of the rectangle lines

Next, we just display the resulting frame and also set a way to exit this infinite loop and close the video feed. By pressing the ‘q’ key, we can exit the script here

```
cv2.imshow('Video', frame)

  if cv2.waitKey(1) & 0xFF == ord('q'):

     break
```

The next two lines are just to clean up and release the picture.

```
video_capture.release()

cv2.destroyAllWindows()
```

Here is the full code.

```
import cv2
import os
cascPath = os.path.dirname(
   cv2.__file__) + "/data/haarcascade_frontalface_alt2.xml"
faceCascade = cv2.CascadeClassifier(cascPath)
video_capture = cv2.VideoCapture(0)
while True:
   # Capture frame-by-frame
   ret, frame = video_capture.read()
   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
   faces = faceCascade.detectMultiScale(gray,
                                        scaleFactor=1.1,
                                        minNeighbors=5,
                                        minSize=(60, 60),
                                        flags=cv2.CASCADE_SCALE_IMAGE)
   for (x,y,w,h) in faces:
      cv2.rectangle(frame, (x, y), (x + w, y + h),(0,255,0), 2)
      # Display the resulting frame
   cv2.imshow('Video', frame)
   if cv2.waitKey(1) &amp;amp; 0xFF == ord('q'):
      break
video_capture.release()
cv2.destroyAllWindows()
```

Create a model to recognize faces wearing a mask

In this section, we are going to make a classifier that can differentiate between faces with masks and without masks.

So for creating this classifier, we need data in the form of Images. Luckily we have a dataset containing images faces with mask and without a mask. Since these images are very less in number, we cannot train a neural network from scratch. Instead, we finetune a pre-trained network called MobileNetV2 which is trained on the Imagenet dataset.

Let us first import all the necessary libraries we are going to need.

```
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 imutils import paths

import matplotlib.pyplot as plt

import numpy as np

import os
```

The next step is to read all the images and assign them to some list. Here we get all the paths associated with these images and then label them accordingly. Remember our dataset is contained in two folders viz- with_masks and without_masks. So we can easily get the labels by extracting the folder name from the path. Also, we preprocess the image and resize it to 224x 224 dimensions.

```
imagePaths = list(paths.list_images('/content/drive/My Drive/dataset'))

data = []

labels = []

# loop over the image paths

for imagePath in imagePaths:

        # extract the class label from the filename

        label = imagePath.split(os.path.sep)[-2]

        # load the input image (224x224) and preprocess it

        image = load_img(imagePath, target_size=(224, 224))

        image = img_to_array(image)

        image = preprocess_input(image)

        # update the data and labels lists, respectively

        data.append(image)

        labels.append(label)

# convert the data and labels to NumPy arrays
```

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

labels = np.array(labels)
```

The next step is to load the pre-trained model and customize it according to our problem. So we just remove the top layers of this pre-trained model and add few layers of our own. As you can see the last layer has two nodes as we have only two outputs. This is called transfer learning.

```
baseModel = MobileNetV2(weights="imagenet", include_top=False,

        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
```

Now we need to convert the labels into one-hot encoding. After that, we split the data into training and testing sets to evaluate them. Also, the next step is data augmentation which

significantly increases the diversity of data available for training models, without actually collecting new data. Data augmentation techniques such as cropping, rotation, shearing and horizontal flipping are commonly used to train large neural networks.

```
lb = LabelBinarizer()

labels = lb.fit_transform(labels)

labels = to_categorical(labels)

# partition the data into training and testing splits using 80% of

# the data for training and the remaining 20% for testing

(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")
```

The next step is to compile the model and train it on the augmented data.

```
INIT_LR = 1e-4

EPOCHS = 20

BS = 32

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

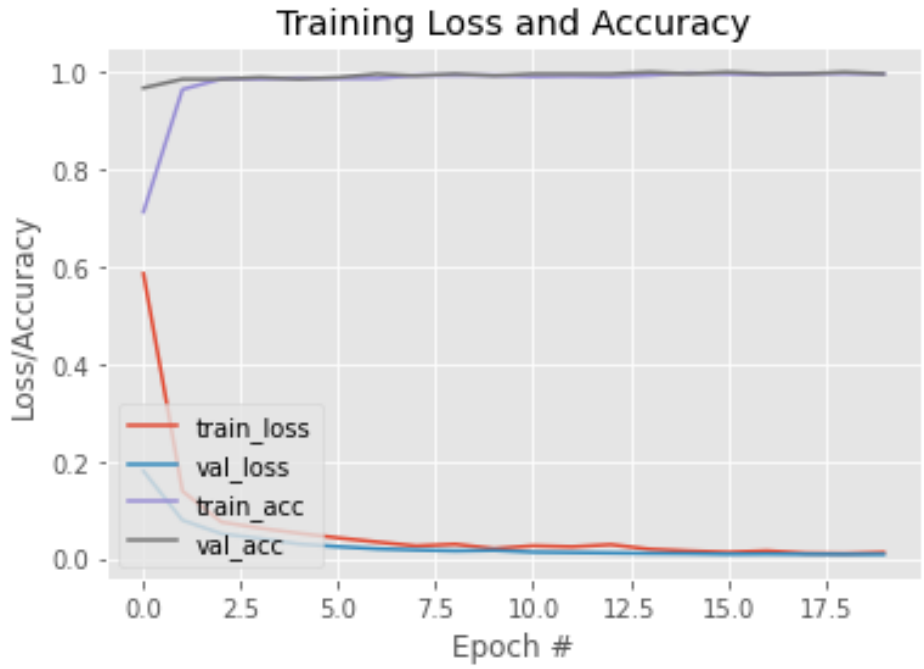
```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
	metrics=["accuracy"])
# train the head of the network
print("[INFO] training head...")
H = model.fit(
	aug.flow(trainX, trainY, batch_size=BS),
	steps_per_epoch=len(trainX) // BS,
	validation_data=(testX, testY),
	validation_steps=len(testX) // BS,
	epochs=EPOCHS)
```

Now that our model is trained, let us plot a graph to see its learning curve. Also, we save the model for later use.

```
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")
```

Output:




```
#To save the trained model

model.save('mask_recog_ver2.h5')
```

How to do Real-time Mask detection

Now that our model is trained, we can modify the code in the first section so that it can detect faces and also tell us if the person is wearing a mask or not.

In order for our mask detector model to work, it needs images of faces. For this, we will detect the frames with faces using the methods as shown in the first section and then pass them to our model after preprocessing them. So let us first import all the libraries we need.

```
import cv2

import os

from tensorflow.keras.preprocessing.image import img_to_array

from tensorflow.keras.models import load_model
```

```
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input

import numpy as np
```

The first few lines are exactly the same as the first section. The only thing that is different is that we have assigned our pre-trained mask detector model to the variable model.

```
ascPath = os.path.dirname(

    cv2.__file__) + "/data/haarcascade_frontalface_alt2.xml"

faceCascade = cv2.CascadeClassifier(cascPath)

model = load_model("mask_recog1.h5")



video_capture = cv2.VideoCapture(0)

while True:

    # Capture frame-by-frame

    ret, frame = video_capture.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(gray,

                                         scaleFactor=1.1,

                                         minNeighbors=5,

                                         minSize=(60, 60),

                                         flags=cv2.CASCADE_SCALE_IMAGE)
```

Next, we define some lists. The faces_list contains all the faces that are detected by the faceCascade model and the preds list is used to store the predictions made by the mask detector model.

```
faces_list=[]

preds=[]
```

Also since the faces variable contains the top-left corner coordinates, height and width of the rectangle encompassing the faces, we can use that to get a frame of the face and then preprocess that frame so that it can be fed into the model for prediction. The preprocessing steps are same that are followed when training the model in the second section. For example, the model is trained on RGB images so we convert the image into RGB here

```
for (x, y, w, h) in faces:
    face_frame = frame[y:y+h,x:x+w]
    face_frame = cv2.cvtColor(face_frame, cv2.COLOR_BGR2RGB)
    face_frame = cv2.resize(face_frame, (224, 224))
    face_frame = img_to_array(face_frame)
    face_frame = np.expand_dims(face_frame, axis=0)
    face_frame =  preprocess_input(face_frame)
    faces_list.append(face_frame)
    if len(faces_list)>0:
        preds = model.predict(faces_list)
    for pred in preds:
    #mask contain probabily of wearing a mask and vice versa
        (mask, withoutMask) = pred
```

After getting the predictions, we draw a rectangle over the face and put a label according to the predictions.

```
label = "Mask" if mask > withoutMask else "No Mask"
    color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
    label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
    cv2.putText(frame, label, (x, y- 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
```

```
    cv2.rectangle(frame, (x, y), (x + w, y + h),color, 2)
```

The rest of the steps are the same as the first section.

```
cv2.imshow('Video', frame)

  if cv2.waitKey(1) & 0xFF == ord('q'):

    break

video_capture.release()

cv2.destroyAllWindows()
```

Here is the complete code:

```
1   import cv2
2   import os
3   from tensorflow.keras.preprocessing.image import img_to_array
4   from tensorflow.keras.models import load_model
5   from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
6   import numpy as np
7
8   cascPath = os.path.dirname(
9      cv2.__file__) + "/data/haarcascade_frontalface_alt2.xml"
10  faceCascade = cv2.CascadeClassifier(cascPath)
11  model = load_model("mask_recog1.h5")
12
13  video_capture = cv2.VideoCapture(0)
14  while True:
15     # Capture frame-by-frame
16     ret, frame = video_capture.read()
17     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
18     faces = faceCascade.detectMultiScale(gray,
19                                          scaleFactor=1.1,
20                                          minNeighbors=5,
21                                          minSize=(60, 60),
22                                          flags=cv2.CASCADE_SCALE_IMAGE)
23     faces_list=[]
24     preds=[]
25     for (x, y, w, h) in faces:
26        face_frame = frame[y:y+h,x:x+w]
27        face_frame = cv2.cvtColor(face_frame, cv2.COLOR_BGR2RGB)
28        face_frame = cv2.resize(face_frame, (224, 224))
29        face_frame = img_to_array(face_frame)
30        face_frame = np.expand_dims(face_frame, axis=0)
31        face_frame =  preprocess_input(face_frame)
32        faces_list.append(face_frame)
33        if len(faces_list)>0:
34           preds = model.predict(faces_list)
35        for pred in preds:
```

```
            label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
            cv2.putText(frame, label, (x, y- 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)

            cv2.rectangle(frame, (x, y), (x + w, y + h),color, 2)
            # Display the resulting frame
        cv2.imshow('Video', frame)
        if cv2.waitKey(1) &amp;amp; 0xFF == ord('q'):
            break
    video_capture.release()
    cv2.destroyAllWindows()
```

This brings us to the end of this article where we learned how to detect faces in real-time and also can detect faces with masks. Using this model we were able to modify the face detector to mask detector.

Github repository for Face Mask Detection :

https://github.com/YashShiyani/Real_time_Face_Mask_Detection

References :

https://searchenterpriseai.techtarget.com/definition/face-detection

https://towardsdatascience.com/face-detection-for-beginners-e58e8f21aad9

https://www.upgrad.com/blog/face-detection-project-in-python/

https://www.datacamp.com/community/tutorials/face-detection-python-opencv

https://www.youtube.com/watch?v=Ax6P93r32KU