Mask Detection System

Introduction

Now, we can use AI to help improve safety by building a mask detection system which denies entry to a public area if a person does not wear a mask. This system also helps us identify when there are people entering with masks into secured areas. It will help to cause alarm and indicate that one is looking like a threat .We can use it for security purposes and help us in the time of course. For example as :- Due to the COVID pandemic last year, one of the basic methods to ensure safety for a person is wearing a mask. By wearing a mask, a person will be less vulnerable to diseases such as COVID, which spread very fast last year.

Approach

1. Human detection using YOLO

YOLO (You only look once) is a fast, deep learning based object detector. YOLO is trained on the COCO dataset, which contains multiple categories with many images in each category. The category which is useful in this case for our mask detection system is the category of person.
YOLO can be used on a live feed from the webcam or any camera device by individually performing detection on each of the frames. The same process can be performed using faster RCNN also but comparatively YOLO is faster so we can get a much better stream of detection.
We loaded labels and generated colors followed by loading our YOLO model.
We defined a loop and then we grabbed our first frame. We made a check to see if it is the last frame of the video.
Next, we grabbed the frame dimensions and performed a YOLO using our current frame as the input. And finally we computed the bounding boxes.

2. Face detection using VIOLA JONES

We're going to do face detection 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. 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.

One of the popular algorithms that use a feature-based approach is the Viola-Jones algorithm. This algorithm is painfully slow to train but can detect faces in real-time with impressive speed. 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.

As we 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 Viola-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.

First we need to get the frames from the video, we do this using the read() function. We use it in an infinite loop to get all the frames until the time we want to close the stream. The read() function returns: The actual video frame read (one frame on each loop), 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. We need to convert frames into greyscales(because in greyscales we can only use Viola)

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.

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. minNeighbors – Parameter specifying how many neighbours each candidate rectangle should have to retain it. Flags - Mode of operation, minSize – Minimum possible object size.

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.

We will draw a rectangle over it. 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.

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.

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.

3. Mask detection

We are going to make a classifier that can differentiate between faces with masks and without masks.

First we need to read all the images and assign them to some list. We’ll get all the paths associated with these images and then label them accordingly. We’ll load the pre-trained model and customize it according to our problem. So we just remove the top layers of this pre-trained model(MobileNetV2)

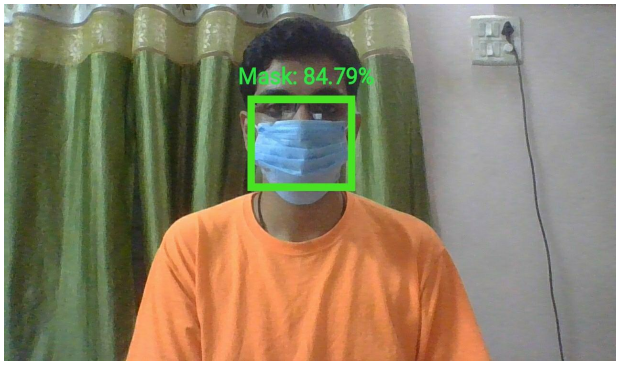
and add a few layers of our own. As you can see the last layer has two nodes as we have only two outputs which is called transfer learning.

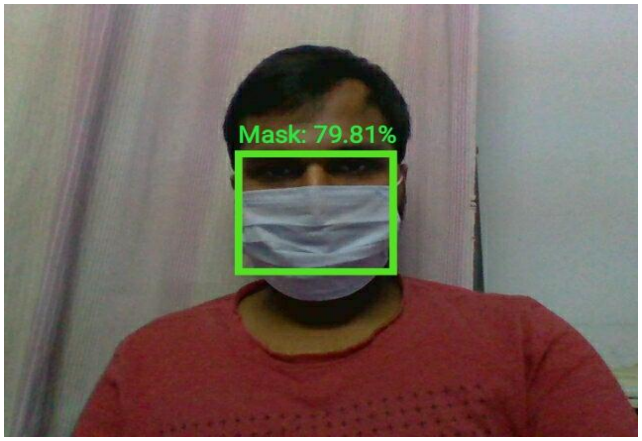
We need to convert the labels into one-hot encoding. After that, we split the data into training and testing sets to evaluate them. Also, the next step is data augmentation which significantly increases the diversity of data available for training models, without actually collecting new data. Data augmentation techniques such as cropping, rotation, shearing and horizontal flipping are commonly used to train large neural networks. Then we'll compile the model and train it on the augmented data.

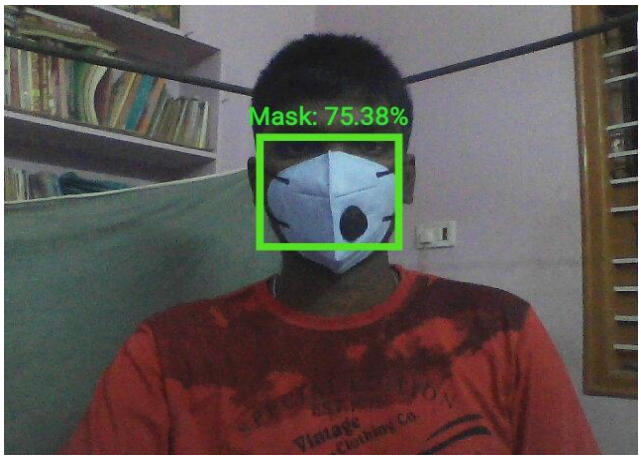
We define some lists. The faces_list contains all the faces that are detected by the faceCascade model and the preds list is used to store the predictions made by the mask detector model.

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 the same that are followed when training the model in the second section. We draw a rectangle over the face and put a label according to the predictions.

The output of the videos are:







Conclusion

Our Model for a Mask Detection System accurately predicts whether a person is wearing a mask or not with a good probability.

Real-Time Usage:- As this model accurately predicts whether a person is wearing a mask or not, it can be used in real-time in the outside world for mask detection as a Vision based automatic door entry system.

Done By:-

- Sairama Sashank Kadiyala - IMT2018064
- M. A. Hadi - IMT2018041
- G.V.Raghava - IMT2018023
- Naveen Kumar - IMT2017029