VR - Mini Project

Aarushi Goenka - IMT2018001
Ameya Kurme - IMT2018007
Ayush Mishra - IMT2018013
Soham Kolhe - IMT2018073

Introduction:

Since the spread of coronavirus, masks have now become a part of our daily life. Wearing face masks is one of the most effective ways to reduce the spread of the virus from one person to another. All public places have made it compulsory to wear masks, even in private places and homes, people are expected to wear masks all the time.

Our aim is to build a vision based automatic door entry system. Such a system would open the gate only if the person is wearing a mask. Such a system would help tremendously in making the place safer.

Abstract:

Given a video, we first detect humans using YOLO, then after slicing only the human part from the frames, face detection is done using VIOLA JONES and finally mask detection is done using a convolutional neural network.

Human Detection Module:

Object detection deals with detecting instances of semantic objects of a certain class in digital images and videos. YOLO algorithm is used for Human Detection.

The YOLO algorithm is one of the fastest real time object detection algorithms. It applies a neural network to the entire image to predict bounding boxes and their probabilities. The YOLO algorithm on running returns the coordinates of the bounding box of the detected human , i.e. the top-left coordinate of the box and also the width and height of our box, using this information we can create the bounding box.

After we have all the four coordinates we slice our frames in the range of these coordinates so that we have only the detected human in each frame and the rest is discarded , we store these frames in a list and use this list in the next part of our code that is Face Detection.

Approach :

We will create a human detector using a pre-trained model. Our model was trained on the COCO dataset which is a large-scale object detection, segmentation, and captioning dataset. We will download pre-trained weights of YOLOV3, the configuration file and the names file.

We will then create and set an input blob for the network, run inference through the network, and gather predictions from output layers. After creating bounding boxes, we'll check if the object detected is a person and display its image.

YOLO takes an input image and resizes it to 448×448 pixels. The image further goes through the convolutional network and gives output in the form of a 7×7×30 tensor. Tensor gives the information about coordinates of the bounding box's rectangle and probability distribution over all classes the system is trained for. We create a threshold of 10% and eliminate labels scoring lesser than that.

The coco.names file contains the names of the different objects that our model has been trained to identify. We store them in a list called classes. To run a forward pass using the cv2.dnn module, we need to pass in the names of layers for which the output is to be computed. net.getUnconnectedOutLayers() returns the indices of the output layers of the network.

For accepting image files, we used another function called load_image() which accepts an image path as a parameter, reads the image, resizes it and returns it.

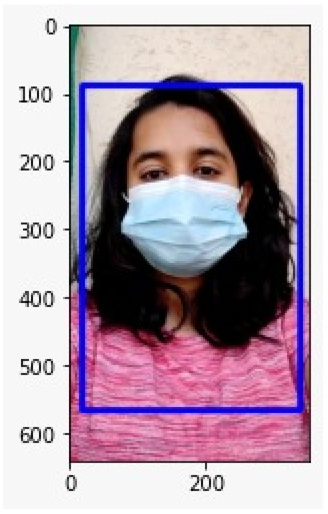
We used blobFromImage that accepts an image/frame from a video or webcam stream, model and output layers as parameters.

The forward() function of cv2.dnn module returns a nested list containing information about all the detected objects which includes the x and y coordinates of the centre of the object detected, height and width of the bounding box, confidence and scores for all the classes of objects listed in coco.names. The class with the highest score is considered to be the predicted class.

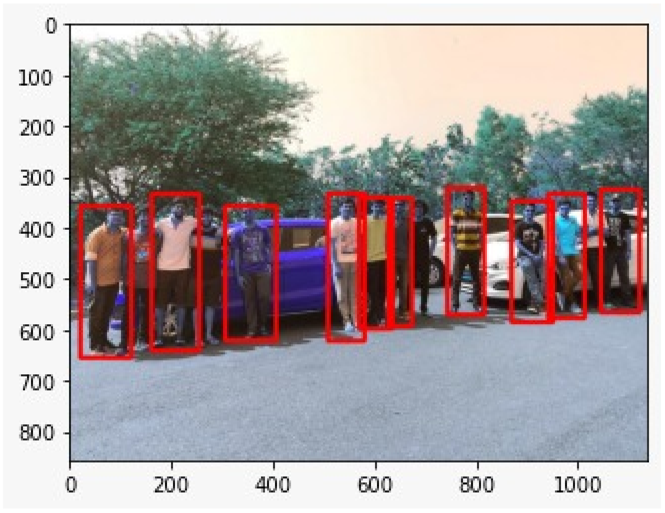
In get_box_dimensions() function, a list called scores is created which stores the confidence corresponding to each object. We then identify the index of class with highest confidence/score using np.argmax(). We can get the name of the class corresponding to the index from the classes list we created in load_yolo().

In some images, we were getting duplicate detections around an object. To solve that we used Non-Maximum Suppression (NMS) to pass in confidence threshold value and nms threshold value as parameters to select one bounding box.

Observations :



Here we can see that the complete person is getting detected by the YOLO Algorithm.



In this case we have a lot of people and therefore it is difficult for the algorithm to differentiate between people but even then it has managed to find a lot of people.

Conclusions :

When we tried to use group photos or images with multiple objects we found that YOLO does not detect small objects or objects which are grouped close together. Since YOLO detects an object by dividing the image into a SxS grid, it predicts only a single object in each grid. Therefore if there exists multiple small or closely grouped objects in a single cell, then YOLO might not be able to detect it properly.

Face Detection Module:

For face detection, we used the Viola Jones algorithm. Viola-Jones was designed for frontal faces, so it is able to detect frontal the best rather than faces looking sideways, upwards or downwards. First we convert the image to grayscale since it is easier to work with and lesser data to process. Viola-Jones outlines a box and searches for a face within the box. It searches for haar-like features. The box moves a step to the right after going through every tile in the picture.

Haar-like features are digital image features used in object recognition. All human faces share some universal properties of the human face like the eyes region is darker than its neighbour pixels, and the nose region is brighter than the eye region. A simple way to find out which region is lighter or darker is to sum up the pixel values of both regions and compare them. The sum of pixel values in the darker region will be smaller than the sum of pixels in the lighter region. By identifying the dark and light areas, we can interpret different parts of a face.

There are 3 types of Haar-like features that Viola and Jones identified in their research:

- Edge features
- Line-features
- Four-sided features

Approach :

We use a trained Haar Cascade classifier model. Then we convert the image to grayscale.

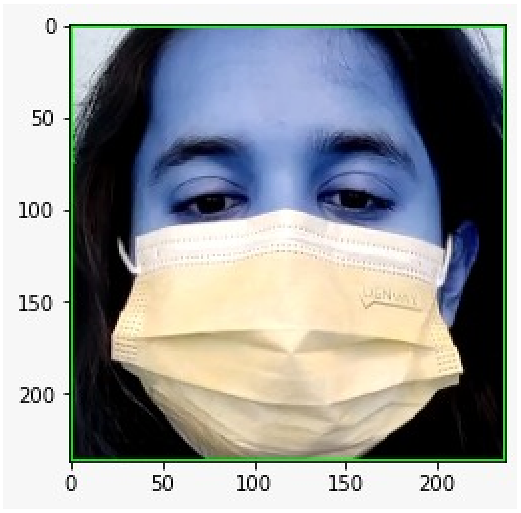
The faceCascade object has a method detectMultiScale(), which receives a frame(image) as an argument and runs the classifier cascade over the image. The term MultiScale indicates that the algorithm looks at subregions of the image in multiple scales, to detect faces of varying sizes.

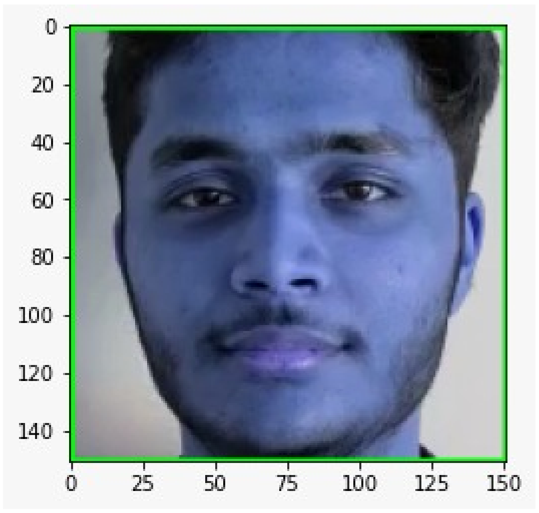
This function takes the following arguments:

- scaleFactor to specify how much to reduce the image size
- minNeighbors to specify how many neighbors each candidate rectangle should have to retain it
- Flags to give the mode of operation
- minSize to specify the minimum object size.

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. This is our region of interest.

Observations :





Here we can clearly see the face along with the mask getting carved out from the complete photo and also in the 2nd photo we see that even without the mask it works perfectly.

Conclusions :

The algorithm finds it difficult to detect faces of people with dark beards. Therefore the model was more accurate for beardless males and females.

Mask Detection Module:

To train a deep learning model to detect whether a person is wearing a mask or not, we used the dataset prepared by Prajna Bhandary. This dataset contains images of people wearing masks and of people without masks.

Approach:

First the data is preprocessed. Dataset is loaded and the model is trained on this dataset using Pytorch. We first converted the image to grayscale and resized it into 100 x 100. The images are then normalized.

We have then used an architecture of 2 convolutional layers and 2 fully connected layers. Both the first and the second convolution layers are followed by a Rectified Linear Unit (ReLU) layer and a MaxPooling layer. After that we have a flatten layer, followed by a dropout layer to reduce overfitting. Then there are two fully connected layers.

The loss is cross entropy loss and the optimiser used is Stochastic Gradient Descent(SGD). Then the model is trained and the hyperparameters are calculated. Then this model is run on our demo videos to detect masks. The loss goes on decreasing and the hyperparameters are set in accordance with least loss. We avoided splitting the data because the size was small.

Observations:

Epoch Number	Total Training Loss	Average Loss
1	91.45051881670952	0.6626849189616631
2	79.48498103022575	0.575978123407433
3	71.50265270471573	0.5181351645269255
4	67.88327871263027	0.4919078167581904
5	63.91326604783535	0.4631396090422851

Using the metric “accuracy”, the value for the model is 0.93.

The time taken for processing each frame was generally very low ,but sometimes the previous frame flickers onto the next scene which creates a hint of lag.

Conclusions:

As the dataset includes masks with only one color, i.e. blue, therefore the model found it difficult to detect masks of any other colour.

References:

1) https://www.mygreatlearning.com/blog/viola-jones-algorithm/

2) https://github.com/prajnasb/observations/tree/master/experiements/data

3) https://towardsdatascience.com/object-detection-using-yolov3-and-opencv-19ee0792a420

4) https://www.youtube.com/watch?v=d3DJqucOq4g

5) https://github.com/aieml/face-mask-detection-keras

6) https://drive.google.com/drive/folders/1e2-Ms1e9ar79MMl7gtEzQ8fXKad5yFP0

7) https://towardsdatascience.com/the-intuition-behind-facial-detection-the-viola-jones-algorithm-29d9106b6999

8) https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/