Mask Detection

Mini-Project

Avik Bhatnagar - IMT2018505
Tanishq Jaswani - IMT2018077
Manan Bansal - IMT2018039

1. Aim:

Build a vision-based automatic door entry system, to allow the door to open only if a human wears a mask.
Modules used here:

- ❏ Human Detection using YOLO or Faster RCNN
- ❏ Face Detection using VIOLA JONES
- ❏ Mask detection using any method

2. Approach:

We run the image with Cascade Classifier, which returns the ROI of the image i.e. the region of interest with the coordinates and dimensions of the same. Then we resize the image to further pass it into a pre-trained convolutional neural network which provides us with the output probabilities of with mask and without a mask.

3. Human Detection:

Introduction:

YOLO (You Only Look Once): The reason we chose this approach was, it being one of the fastest real-time object detection algorithms. It works on the approach of applying a neural network to the entire image to predict bounding boxes and their probabilities.

We'll read a pre-trained model(COCO dataset) and config file, 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 class of people detected is a person and display its image.

Dataset:

COCO is large-scale object detection, segmentation, and captioning dataset. It has several features like Object segmentation, Recognition in context, 1.5 million object instances, 80 object categories, 250,000 people with key points.

Approach:

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 1) coordinates of the bounding box's rectangle and 2) Probability distribution over all classes the system is trained for. Thresholding these confidence scores (probability) eliminates class labels scoring lesser than 30%.

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. Now 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 in a function called detect_objects() that accepts image/frame from 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().

$$filters = 5 * (2 + number\_of\_classes)$$

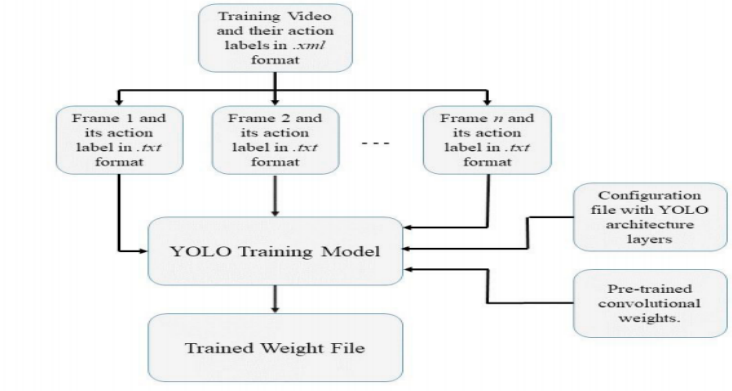
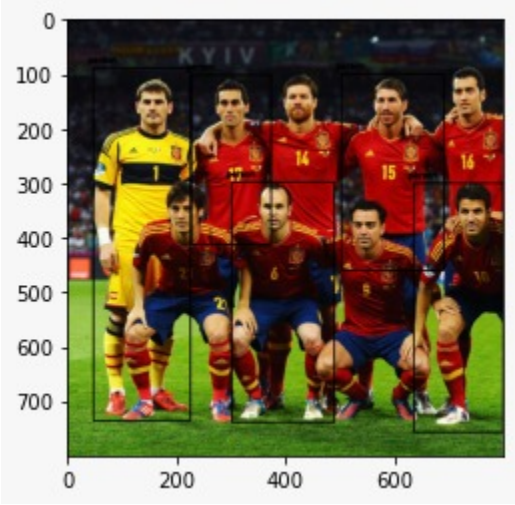


Fig. 3: The Flowchart for training of YOLO

Observation:



In the above image we have successfully detected humans with bounding boxes. We have further used this approach to use in mask detection. The videos used in the later part are attached in the folder but have pasted screenshots of the same along with the report.

Conclusion:

YOLO does not always handle small objects well. It especially does not handle objects grouped close together. This was maybe because it divides an input image into an SxS grid where each cell in the grid predicts only a single object. If there exist multiple, small objects in a single cell then YOLO will be unable to detect them, ultimately leading to missed object detections.

Things we tried that didn't work:
We tried using an anchor box prediction mechanism where we predict the x, y offset as a multiple of the box width or height using a linear activation. We found this formulation decreased model stability and didn't work very well.

4. Face Detection:

Introduction:
Viola Jones is an algorithm which has proved itself to work exceptionally well for real-time face detection despite it taking a long time to train. It takes a grayscale image and looks at smaller subregions to detect faces. It uses Haar-like features to detect faces in the algorithm.

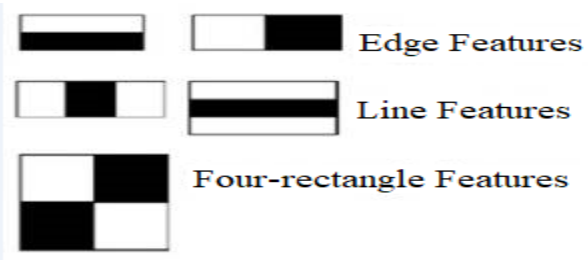
It involves 2 major steps:-

- ❏ Selecting Haar-like features
- ❏ Creating an integral image/ROI

Theory:
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. If one side is lighter than the other, it may be an edge of an eyebrow or sometimes the middle portion may be shinier than the surrounding boxes, which can be interpreted as a nose This can be accomplished using Haar-like features and with the help of them, we can interpret the different parts of a face.

There are 3 types of Haar-like features identified:

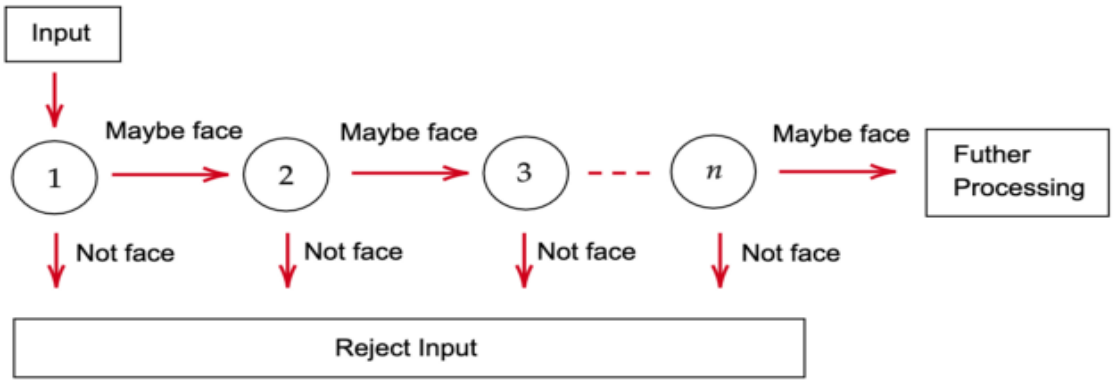
a) Edge-features
b) Line-features
c) Four-sided features: For finding diagonal features.



The value of the feature is calculated as a single number: the sum of pixel values in the black area minus the sum of pixel values in the white area. The value is zero for a plain surface in which all the pixels have the same value, and thus, provide no useful information.

In reality, these calculations can be very intensive since the number of pixels would be much greater when we are dealing with a large feature. The integral image allows us to perform these intensive calculations quickly so we can understand whether a feature of several features fit the criteria. It is the name of both data structure and algorithm. In this, the value of each point is the sum of all pixels above and to the left, including the target pixel.

Using these integral images, we save a lot of time calculating the summation of all the pixels in a rectangle as we only have to perform calculations on four edges of the rectangle. See the example below to understand. No matter how many pixels are in the rectangle box, we will just need to compute on these 4 vertices. To calculate the value of any haar-like feature, we simply have to calculate the difference between the sums of pixel values of two rectangles.



Approach:

Here we have used Haar cascade classifier frontal face file for this project. It is used to detect the face. For this specific classifier to work, we need to convert the frame into grayscale.

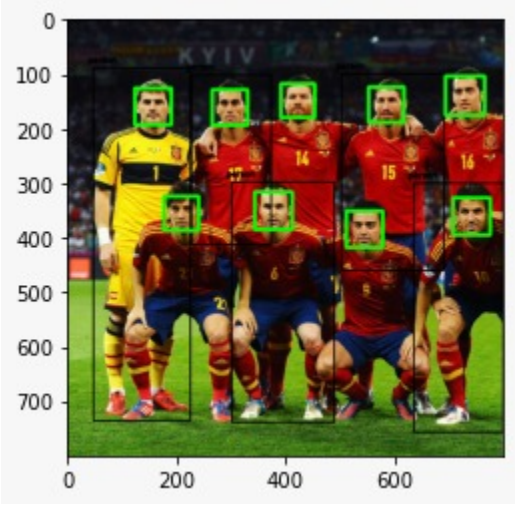
The faceCascade object has a method detectMultiScale(), which receives a frame(image) as an argument and runs the classifier cascade over the image. The term MultiScale indicates that the algorithm looks at subregions of the image in multiple scales, to detect faces of varying sizes.

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

Scale factor - parameter specifying how much the image size is reduced. Reduce the size of the image to increase the chance of matching size with the model for detection.
minNeighbors - parameter specifying how many neighbours each candidate rectangle should have to retain it. It affects the quality of the detected faces.
Flags - mode of operation
minSize - Smaller objects to be 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 the width and height of the rectangle is used to detect the face. This is known as our region of interest.

<u>Observations:</u>

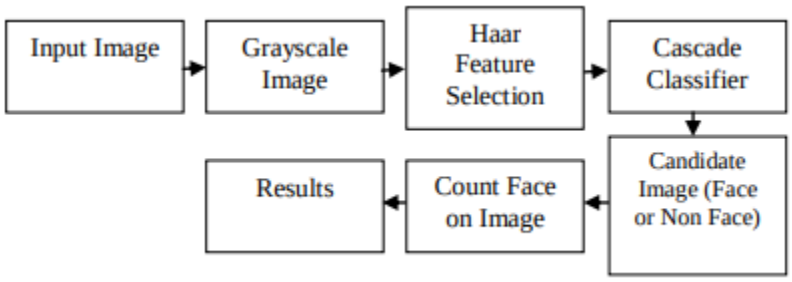


All the faces have been successfully detected and hence we can proceed to the third part of the project to detect masks.

Evaluation:

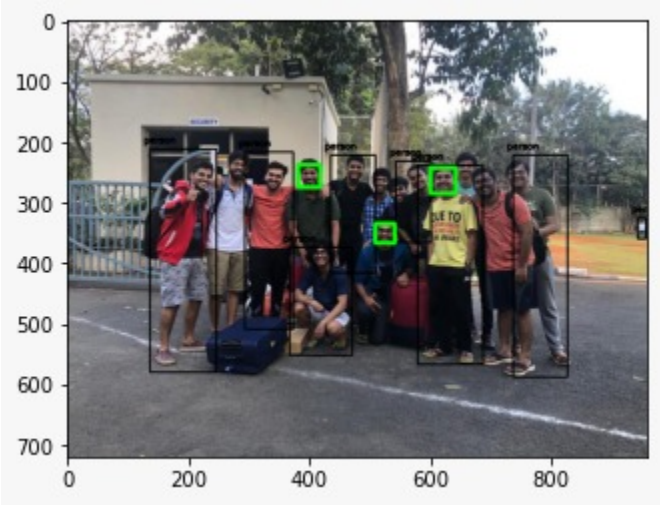
The accuracy may vary with the number of detected faces in the image database. It is calculated as:

Face Detection= Detected Face/ Number of face



Conclusion:

In this work, we use the various techniques to detect and locate faces in images. Faces have been detected through skin segmentation in grey scaled images performed by various Image Processing Techniques. The regions of face were found by analyzing the binary and gray scale variation in different regions of the face. The Viola-Jones method is not functioning properly in images with low light intensity. It especially does not handle objects grouped close together. In the future, this method is expected to detect dark images and function in variable lighting conditions.



5. Mask Detection:

Introduction:

Detect face masks in real-time video streams. We divided this task into two sub-tasks, training and testing. We loaded the given dataset and trained the model on this dataset

using Pytorch. We took 4 different people for our testing who wore masks and did not wear masks. When we run the project on the command lines, the output is to open the door if the person is wearing a mask and not open the door if he is not wearing the masks.

Dataset:

To train a deep learning model to classify whether a person is wearing a mask or not, we need to find a good dataset with a fair amount of images for both classes. The dataset provided to us by sir fulfilled all the requirements needed.

Theory:

a) Data Processing:

Data Visualization -
The total number of images in the dataset is visualized in both categories - ‘with mask’ and ‘without mask’. Each category is mapped to its respective label. The mapping is {‘with mask’ : 0, ‘without mask’ : 1}. 0 returns ‘open the door’ and 1 returns ‘close the door’.

RGB To Grayscale -
We need to convert RGB to grayscale images to avoid non-essential information and hence reduce the size of training data required to achieve good performance. As grayscale rationalizes the algorithm and diminishes the computational requisites, it is utilized for extracting descriptors instead of working on color images instantaneously.

Deep CNN’s require a fixed-size input image. Therefore we need a fixed common size for all the images in the dataset. Using *cv2.resize()* the gray scale image is resized into 100 x 100. The images are normalized to converge the pixel range between 0 and 1.

b) Training of Model:

The architecture we have used in this project is 2 convolutional layers and 2 fully connected layers.
The First Convolution layer is followed by Rectified Linear Unit (ReLU) and MaxPooling layers. The Convolution layer learns from 200 filters. Kernel size is set to 3 x 1 which specifies the height and width of the 2D convolution window. As the model should be aware of the shape of the input expected, the first layer in the model needs to be provided with information about input shape.

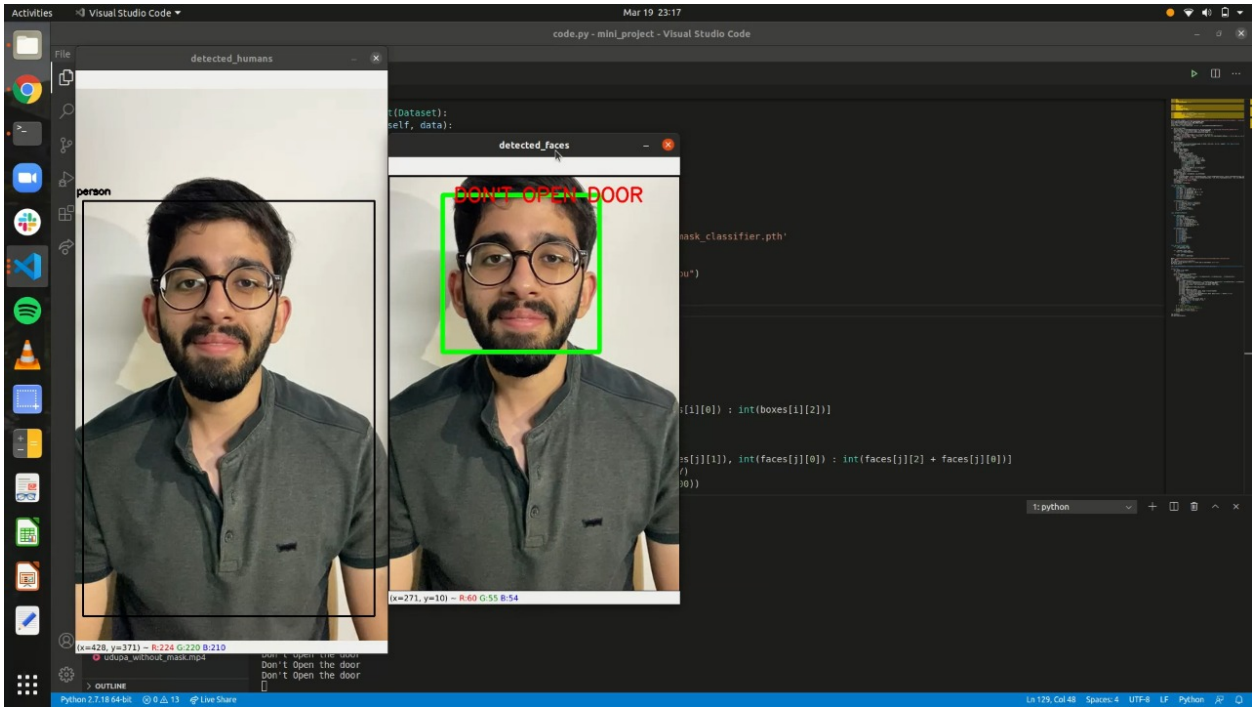
The activation parameter to the Conv2D class is set as “relu”. It represents an approximately linear function that possesses all the assets of linear models that can easily be optimized with gradient-descent methods. Considering the performance and generalization in deep learning, it is better compared to other activation functions]. Max Pooling is used to reduce the spatial dimensions of the output volume. Pool_size is set to 2x 2 and the resulting output has a shape (number of rows or columns) of: shape_of_output = (input_shape – pool_size + 1) / strides), where strides has default value (1,1)

To reduce overfitting a Dropout layer with a 50% chance of setting inputs to zero is added to the model. The final layer (Dense) with two outputs for two categories uses the Softmax activation function.

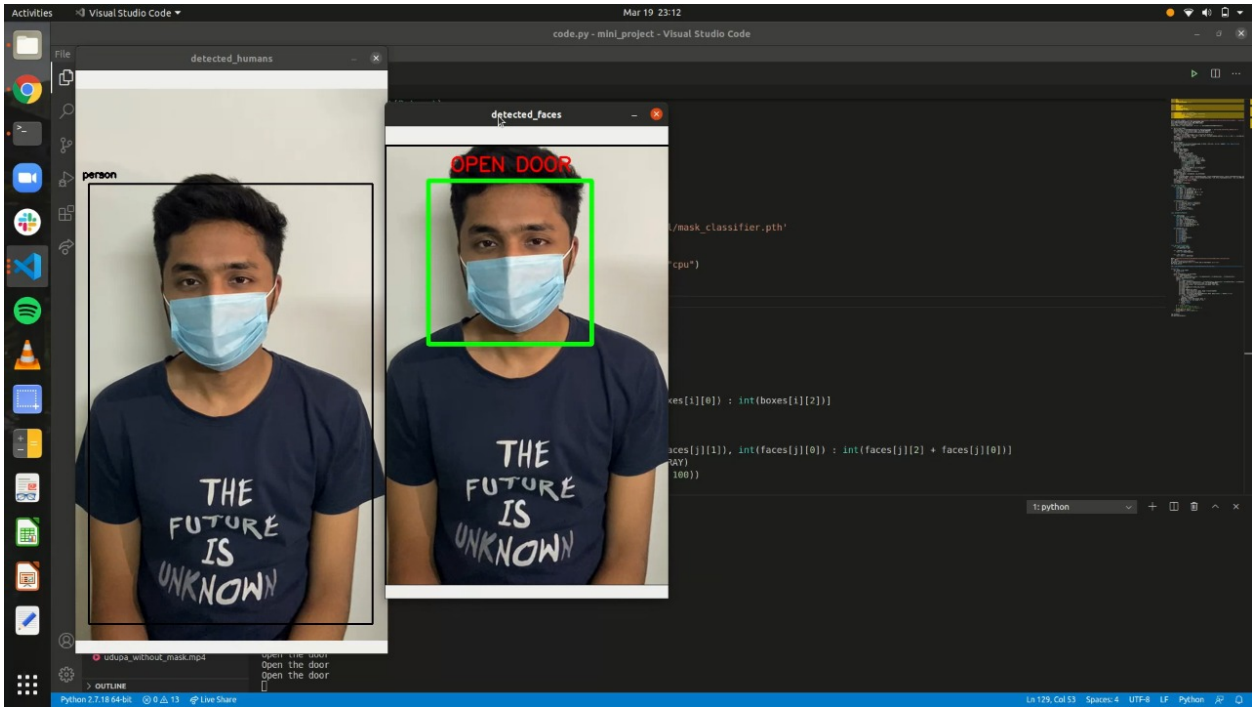
Here the “cross entropy loss” optimizer is used. categorical_crossentropy which is also known as multiclass log loss is used as a loss function (the objective that the model tries to minimize). As the problem is a classification problem, metrics are set to “accuracy”.

Observations:

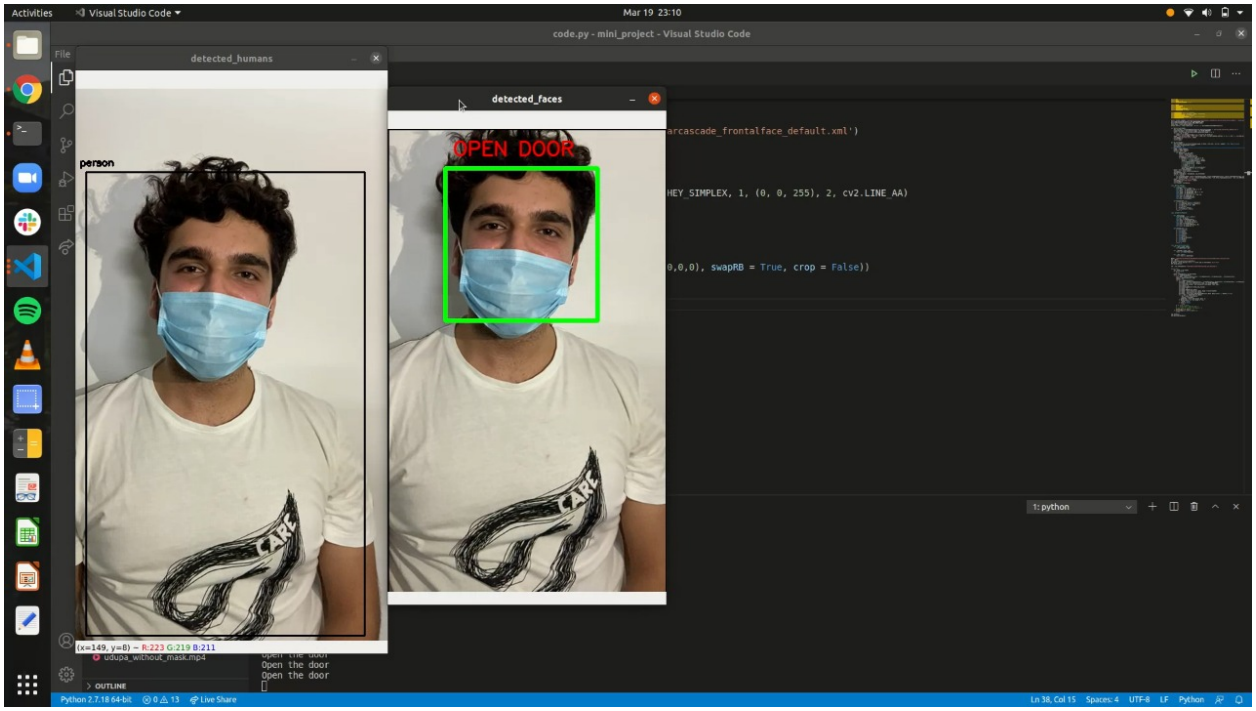
Avik Bhatnagar is not wearing the mask and hence “DOOR NOT OPEN”.



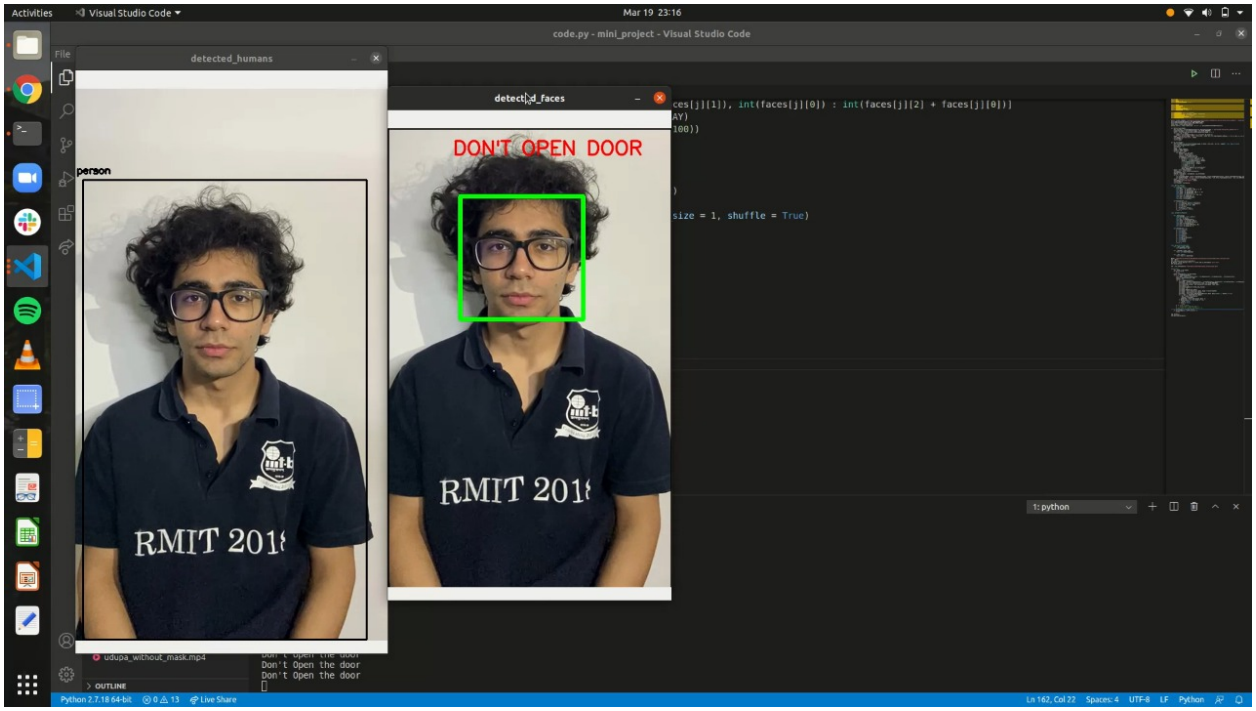
Manan Bansal is wearing the mask and hence our output is “DOOR OPEN”.



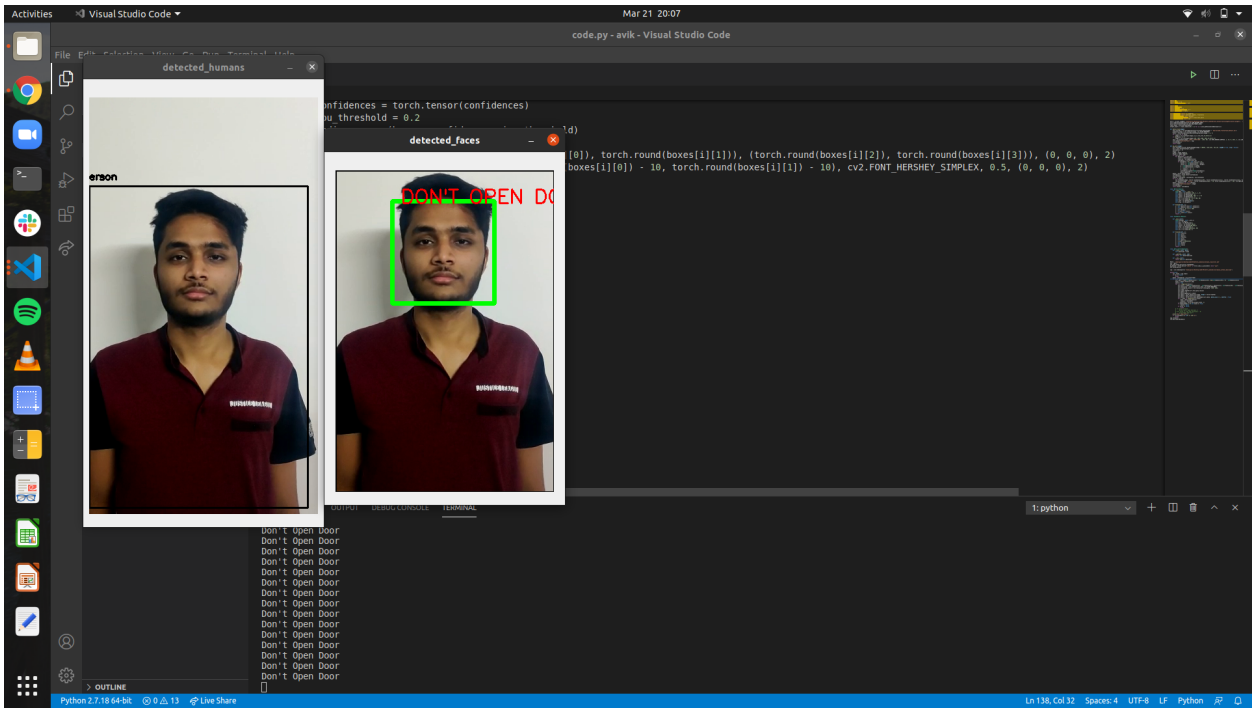
Tanishq Jaswani is wearing a mask, “DOOR OPEN”.



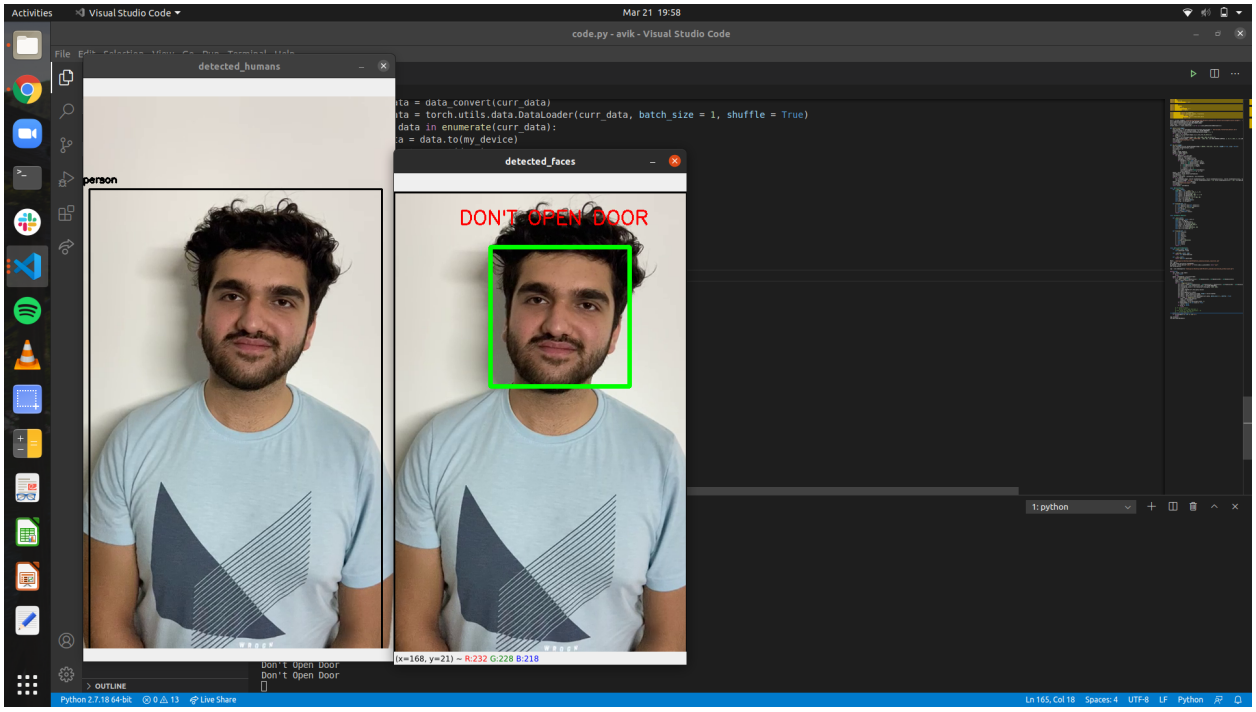
Tanmay Arora is not wearing a mask and hence entry is restricted, “DOOR NOT OPEN”.



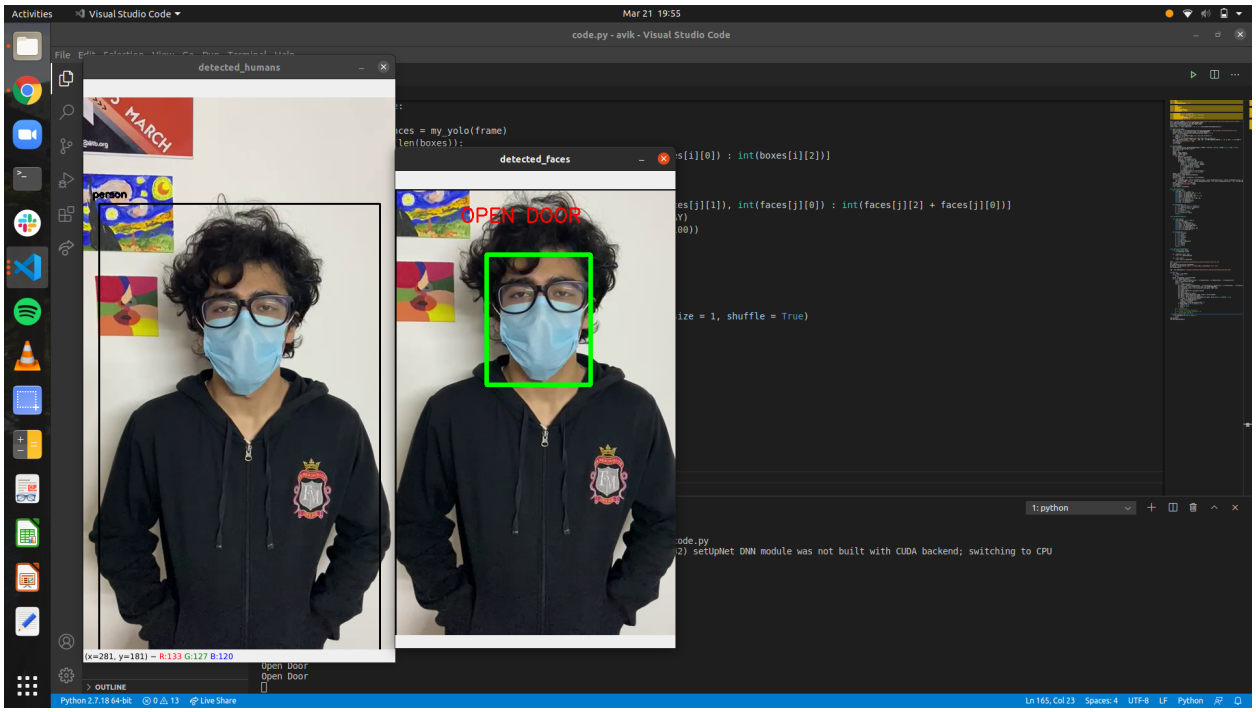
Manan Bansal is not wearing a mask and hence entry is restricted, “DOOR NOT OPEN”.



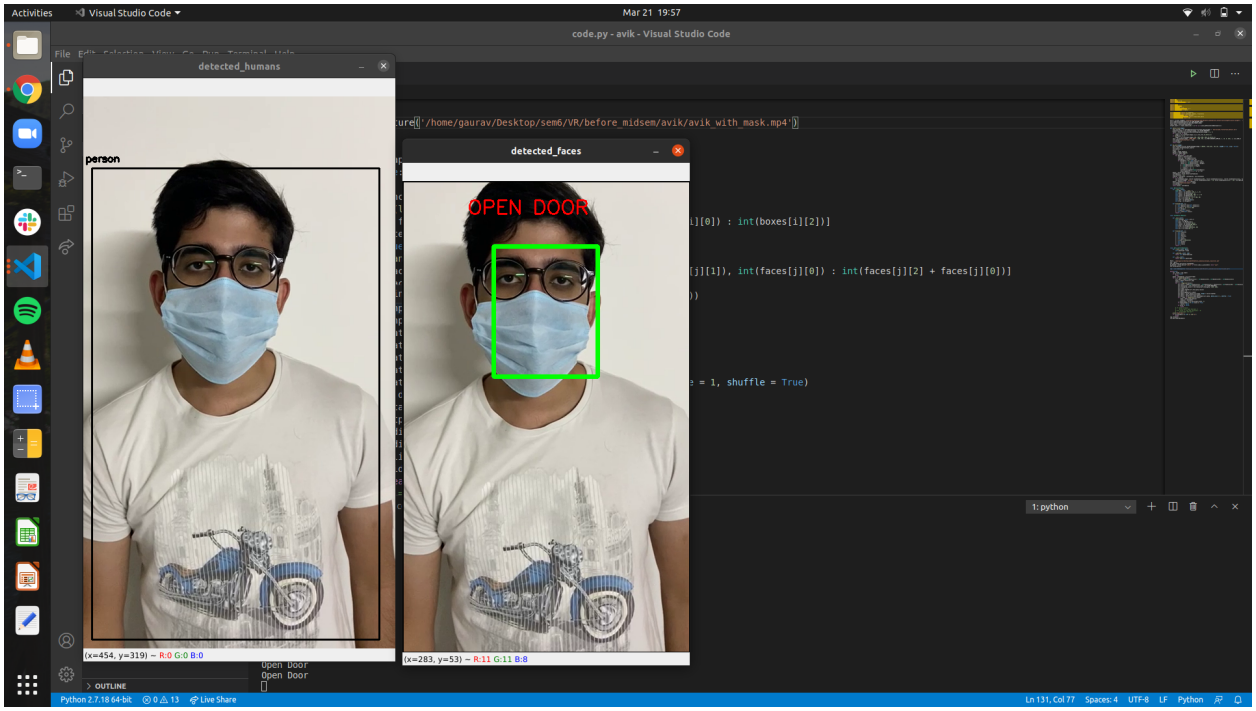
Tanishq Jaswani is not wearing a mask and hence entry is restricted, “DOOR NOT OPEN”.



Tanmay Arora is wearing the mask and hence our output is “DOOR OPEN”.



Avik Bhatnagar is wearing the mask and hence our output is “DOOR OPEN”.



Conclusion:

One of the main reasons behind achieving this accuracy lies in *MaxPooling*. It provides rudimentary translation invariance to the internal representation along with the reduction in the number of parameters the model has to learn. This sample-based discretization process down-samples the input representation consisting of image, by reducing its dimensionality. Number of neurons has an optimized value which is not too high. A much higher number of neurons and filters can lead to worse performance. The optimized filter values and pool_size help to filter out the main portion (face) of the image to detect the existence of the mask correctly without causing over-fitting. Our final accuracy of the above model came out to be 94.6%.

The system can efficiently detect partially occluded faces either with a mask or hair or hand. It considers the occlusion degree of four regions – nose, mouth, chin and eye to differentiate between annotated mask or face covered by hand. Therefore, a mask covering the face fully including nose and chin will only be treated as “with mask” by the model.

The main challenges faced by the method mainly comprise varying angles and lack of clarity. Indistinct moving faces in the video stream make it more difficult. However, following the trajectories of several frames of the video helps to create a better decision

– “with mask” or “without mask”. Size of the image being drastically smaller and much less clear also created problems. Varying angles also was one of the main challenges faced by us. It was also unable to identify different coloured masks.

Scope of Project:

Using basic ML tools and simplified techniques the method has achieved reasonably high accuracy. It can be used for a variety of applications. Wearing a mask may be obligatory in the near future, considering the Covid-19 crisis. Many public service providers will ask the customers to wear masks correctly to avail of their services. The deployed model will contribute immensely to the public health care system. In future it can be extended to detect if a person is wearing the mask properly or not. The model can be further improved to detect if the mask is virus prone or not i.e. the type of the mask is surgical, N95 or not.