VISUAL RECOGNITION MINI PROJECT

AUTOMATIC DOOR ENTRY SYSTEM

AKSHATH (IMT2018501)
HRUTHIK (IMT2018507)
SHANTHAN (IMT2018522)
MADHAV (IMT2018524)

Abstract

In this project, we are designing a system that can detect if a mask is present on the face of a person in real time. We have used *YOLOv3* for human detection, *Viola -Jones* algorithm for face detection and a CNN model for mask detection. This system can be very useful for building automatic doors that can only allow people wearing masks.

Introduction

With the rise of COVID-19 pandemic, face mask detection has seen significant progress in the domain of Computer Vision. Since the most simple and effective way of suppressing the transmission of COVID-19 is by wearing masks, we were motivated to work on this face mask detection technology. As many public buildings do not have strict mask monitoring systems, our mask detection system can be helpful to automate such processes.

Human Detection: YOLOv3

YOLO (You Only Look Once) is a single stage object detection algorithm. YOLO is one of the most efficient choices to deploy on a real time object detection.YOLO has released various versions and variants over the period of time. For the project we have chosen to implement the YOLOv3 model as this provides good accuracy.This YOLOv3 was trained on COCO dataset which consists of 80 classes.

YOLO uses a single deep convolutional network that predicts multiple bounding boxes and class probabilities for the boxes. Unlike the brute force method or the region proposal method YOLO sees the entire image during the training and detecting phase. YOLO splits the given image into SxS grids. Then YOLO makes an assumption that every grid cell has B objects. Then YOLO uses the Darknet that helps in predicting what can be the B objects that can be present in that cell with their bounding boxes and confidence score. It also predicts the probability of the object being a particular class. In the next step YOLO applies the non maximum suppression to get rid of the duplicates and the low confidence boxes. By the end of this process YOLO has nicely predicted the object(s).

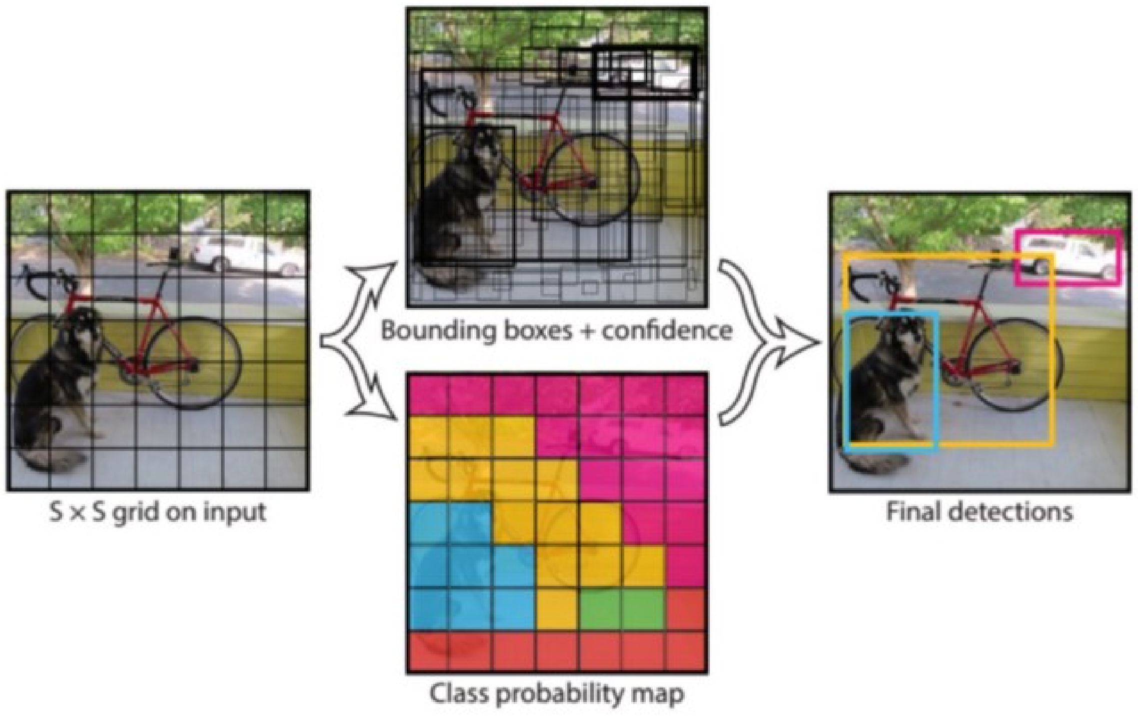


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

<u>Code:</u> After importing the required packages, in the first step we have created a list of all the names of the classes classified in the coco dataset. Next we will read the .cfg and weights of the YOLOv3 model, create and set input blob for the network. Finally we get the output predictions from the output layers. For each detection we will be creating bounding boxes ignoring the confidences below the threshold. Now we will be drawing the bounding boxes for the remaining predictions. If a person is detected then the image will be sent to the next stage.

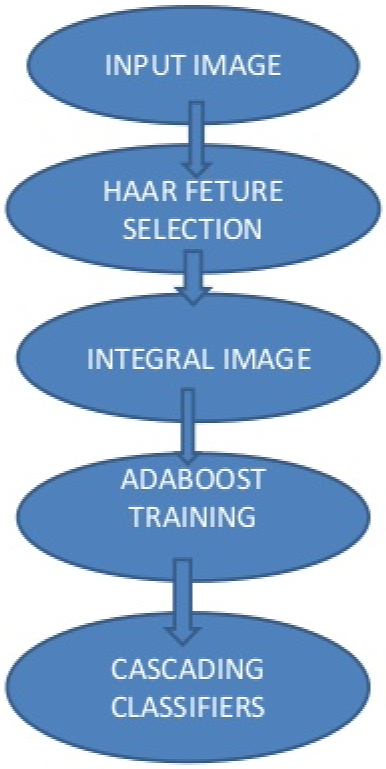
Face Detection: Viola-Jones

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.

This algorithm works only on a grayscale image. So it is necessary to convert the incoming video frame (or an input image) to grayscale. Given an 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.

Here are the four main steps that are implemented in Viola-Jones:

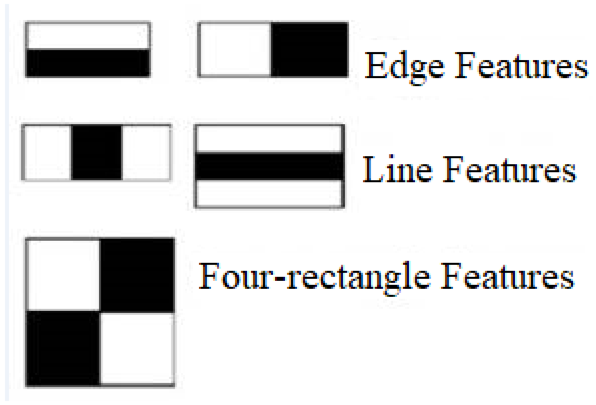
1. Selecting Haar-like features
2. Creating an internal image
3. Running adaboost training
4. Creating classifier cascades



Workflow of Viola-Jones algorithm

Haar-like features:

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.



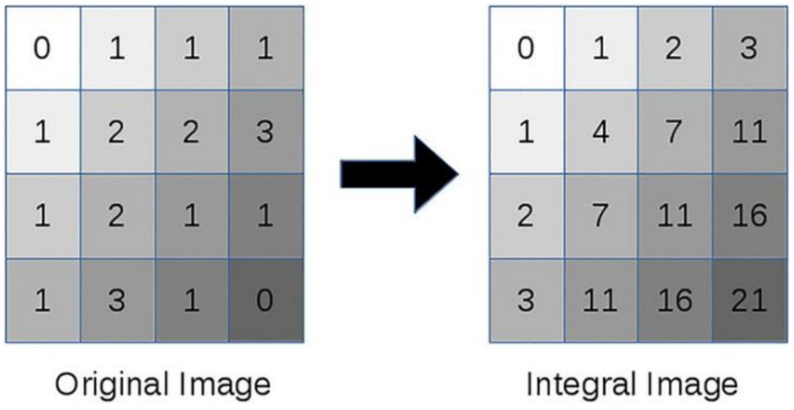
The above figure shows the three main Haar-like features that were identified by Viola and Jones in their research.

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.

Since our faces are of complex shapes with darker and brighter spots, a Haar-like feature gives you a large number when the areas in the black and white rectangles are very different. Using this value, we get a piece of valid information out of the image.

<u>Integral images:</u>

In an integral image, 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. Now to calculate the value of any haar-like feature, you have a simple way to calculate the difference between the sums of pixel values of two rectangles.

<u>Adaboost training:</u>

Adaboost is used for applying machine learning on the obtained integral image.

The number of features that are present in the 24×24 detector window is nearly 160,000, but only a few of these features are important to identify a face. So we use the AdaBoost algorithm to identify the best features in the 160,000 features.

In the Viola-Jones algorithm, each Haar-like feature represents a weak learner. To decide the type and size of a feature that goes into the final classifier, AdaBoost checks the performance of all classifiers that you supply to it.

To calculate the performance of a classifier, we evaluate it on all subregions of all the images used for training. Some subregions will produce a strong response in the classifier. Those will be classified as positives, meaning the classifier thinks it contains a human face. Subregions that don't provide a strong response don't contain a human face, in the classifiers opinion. They will be classified as negatives. The classifiers that performed well are given higher importance or weight. The final result is a strong classifier, also called a boosted classifier, that contains the best performing weak classifiers.

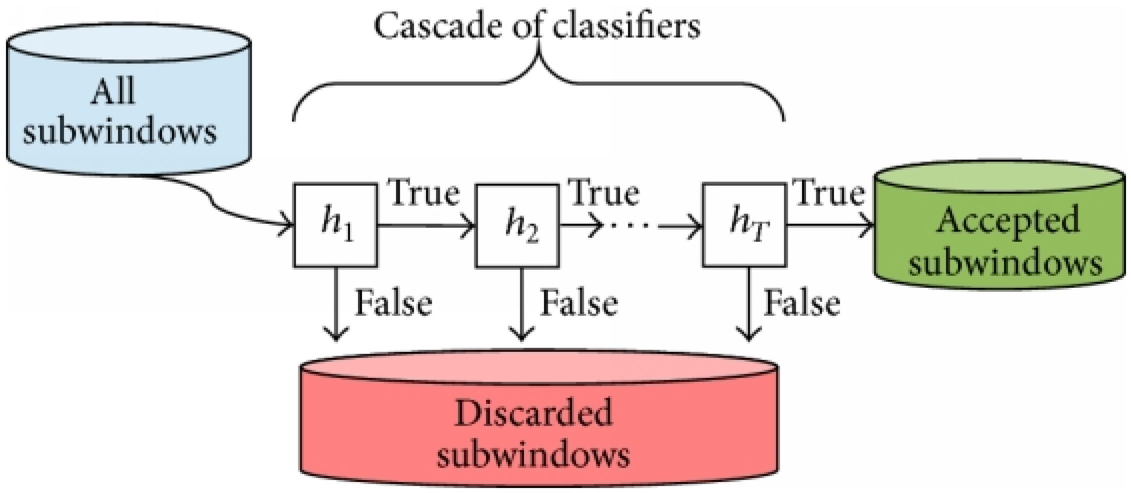
So while training the AdaBoost to identify important features, we're feeding it information in the form of training data and subsequently training it to learn from the information to predict. So ultimately, the algorithm is setting a minimum threshold to determine whether something can be classified as a useful feature or not.

<u>Cascading Classifiers:</u>

Now, out of the 160,000+ features, Adaboost chooses some and discards the other. Yet, it will still be a time consuming process to calculate these features for each region in the image. Since we have a 24x24 window, that slides over the image and classifies whether it is a facial region or not, we basically pipeline this process of classifying regions.

We set up a cascaded system in which we divide the process of identifying a face into multiple stages. In the first stage, we have a classifier which is made up of the best

features, ie. in the first stage, the subregion passes through the best features such as the feature which identifies the nose bridge or the one that identifies the eyes. In the next stages, we have all the remaining features.
When a subregion is processed by the first stage, and it is classified as negative, it is not passed to the further stages. Thus, saving precious time and making real-time face detection possible in a live video feed.



<u>The code:</u>
Firstly, the incoming frame from the webcam was converted into grayscale. After that, it was passed to the haarcascade_frontalface_alt2.xml pretrained model available in opencv. The scale factor was kept as 1.1 with 5 minimum neighbours and the minimum size was kept at (60,60) for clarity and proper detection of mask (if present).
If the face is present, a box is drawn around the face and the frame is returned.

Mask Detection

Dataset:

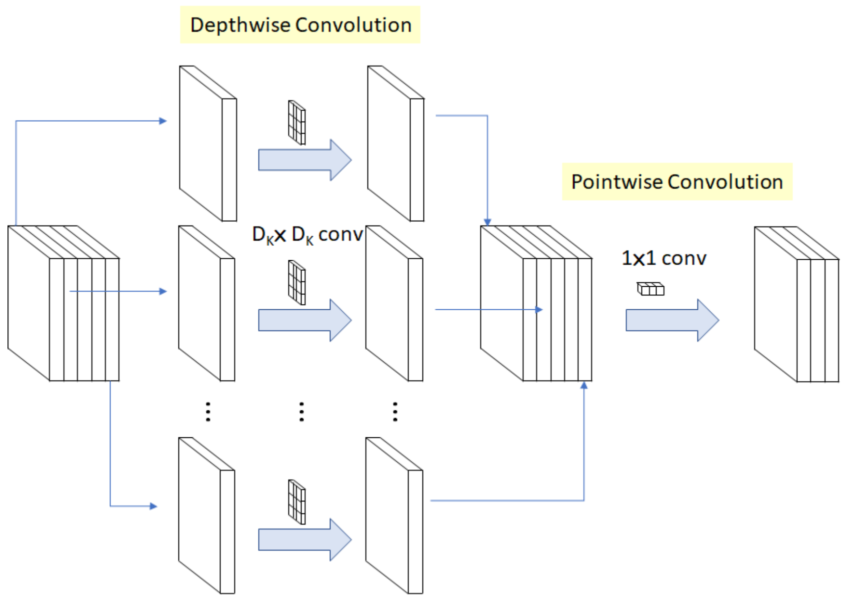
For building the mask detection model, we used this dataset which was provided by *Prajna Bhandary*. It has about 1376 images out of which 690 are images of people with masks and the rest 686 without masks kept in 2 folders named ‘with_mask’ and ‘without_mask’.

MobileNet:

CNN is a class of deep learning models which became popular in various computer vision tasks and it is also becoming handy in many other domains.CNN’s are made up of multiple building blocks such as convolutional layers,fully connected layers etc.. and is designed to learn automatically and adaptively the spatial hierarchies of the features.However, in order to achieve a high performance, modern CNNs are becoming deeper and increasingly complex. Such networks cannot be used in real applications like robots and self driving cars.

MobileNet is an efficient and portable CNN architecture that is used widely in real life. It is an architecture that uses depth-wise separable convolutions to build light networks. Each depth-wise separable convolution layer consists of a depthwise convolution and a pointwise convolution.Counting depthwise and pointwise convolutions as seperate layers.

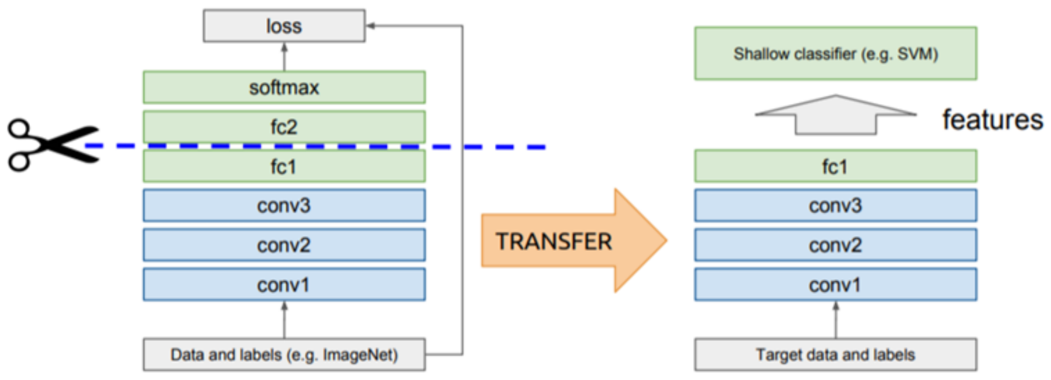
This architecture consists of 28 layer in total. It has approximately 4M parameters(4253864) which is light compared to some other deep models.The below image is clearly depicting the depth wise convolutions and point wise convolution.



In the standard convolution, the entire depth of the input layer undergoes convolution but here each channel(depth) convolves with the filters as shown above.

Transfer learning:

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. There are many famous CNN architectures like AlexNet, ResNet etc.. which are trained on large datasets like ImageNet which have learnt a lot of information . Now we can use these pre-trained models to any other task which saves a lot of training time. The below is an example of transfer learning:-



Training the model:

We have used MobileNet model as our base model and then a flattening layer and dense layer has been added on top of this base model . The input for the model is 224 x 224. So all the images are initially resized. Then they were fed for training the model. Now all the data along with the corresponding labels(with mask: 0 , without mask: 1) are converted into a numpy array. These arrays are used as the input for our Sequential CNN model. The architecture of our CNN model consists of various layers such as *Conv2D, MaxPooling2D, Flatten, dropout* and *dense.* The activation function used all along is *ReLU* except for the last dense layer. For this layer, the *softmax* function is used to output a vector which specifies the probability of our two classes. For our model, we used the *adam* optimizer and *binary_crossentropy* loss function. On the test data, every test image needs to be resized . This mask detection model classifies the images with an accuracy of 99%.

Each group member has recorded a video to test the system and the videos are uploaded along with this report. An indication mark has been set at the top left corner of the live video or image to indicate whether the person is allowed(wearing mask) or not(Not wearing mask). The detected person and the face are put in the rectangular boxes.

Note: In order to speed up the detection in real time, we have pipelined the three models by using multi threading. This gave a boost to the obtained FPS (from 0.3 to about 7).

References:

- YOLO — You only look once, real time object detection explained https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006
- Detecting people with YOLO and OpenCV. https://medium.com/@luanaebio/detecting-people-with-yolo-and-opencv-5c1f9bc6a810
- “Viola-Jones object detection framework” *Wikipedia*, 25 Oct. 2020, https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework
- Breaking Down Facial Recognition: The Viola-Jones Algorithm https://towardsdatascience.com/the-intuition-behind-facial-detection-the-viola-jones-algorithm-29d9106b6999
- Face Detection using Viola Jones Algorithm https://www.mygreatlearning.com/blog/viola-jones-algorithm
- Traditional face detection with python https://realpython.com/traditional-face-detection-python
- Face Mask Detection using Convolutional Neural Networks - Python | Keras | Tensorflow | OpenCV https://www.youtube.com/watch?v=d3DJqucOq4g&ab_channel=AIwithThakshila
- MobileNet Architecture https://iq.opengenus.org/mobilenet-v1-architecture/