Face-Mask Detection

Siddartha Veluvolu
IMT2018071
siddhartha.veluvolu@iiitb.org

Akash Chunduri
IMT2018005
akash.chunduri@iiitb.org

Ellanti Bharath Sai
IMT2018022
bharath.sai@iiitb.org

Kousthubh Tadanki
IMT2018048
Tadanki.048@iiitb.org

***Abstract*—To build a vision based door entry system that opens door only when a person wears a mask.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

In this project we are trying to build a vision based door entry system. We need to open door only when a person wears a mask. This should be a real time working model. We divide our model into three different models.
1. Human detection.
2. Face detection.
3. Mask detection.

We send the live video to human detection model. this model identifies the people in the video and draw a bounding box around people. We then only send these bounding box of people into face detection model which tries to detect the face of these people and draw another bounding box around faces of people in video. We finally send this output faces to the mask detection system and predict whether the person wears the mask or not.

II. Human Detection system

The task here is to detect people in the live video and get only the part bounding box that encloses person as output. We have several methods to do this job.

We basically use a algorithm that generally detects all the different objects present in a image. There are several algorithms for object detection. Some of them would be

- R-CNN, Fast R-CNN, Faster R-CNN.
- Single Shot detector(SSD's).
- YOLO.

A. R-CNN

R-CNN is one of the first object detector based on deep learning. The first variant of R-CNN is based on the algorithm like selective search to find bounding boxes for objects. Then these bounding boxes are sent to CNN for classification of objects. Hence this makes the detector very slow.
Then other variants of R-CNN like Fast R-CNN, Faster R-CNN are released. Fast R-CNN is nothing but improvement of accuracy and training time of R-CNN. Whereas Faster R-CNN completely removed the idea of using selective search for determining boundary box of objects and relied on Regional Proposed Network (RPN) that is fully convolutional and can predict bounding box of object along with objectness score that explains how likely the bounding box contains the object.
R-CNN is a two stage detector and is very slow compared to single stage detectors like YOLO, SSD's. Hence I used YOLO for this human detection as it is much faster than Faster R-CNN.

B. YOLO- You only look Once

Unlike R-CNN family which primarily uses region to localize bounding box. This region based aproach makes possible to vissit some region many times YOLO simply takes complete image and divides the image into grids say mxn grid and applies image classification and localization to each grid. Thus it tries to predict the bounding boxes of different objects in an image. In this we are visisting to one region only once. This is much faster than R-CNN family and comparatively gives similar accuracy of R-CNN. Accuracy of YOLO model is the mean average position. Accuracy of YOLO detector would be around 50-60 percent.

We used a pretrained YOLOv3 model that is trained on coco dataset which has 12 different type of classes including people.

III. Face Detection

The aim of this module is to detect the actual face in the region sent by the human detection module. To do this we used Viola Jones object detection framework.
The characteristics of Viola–Jones algorithm which make it a good detection algorithm are:

- Robust – very high detection rate (true-positive rate) very low false-positive rate always.
- Real time
- Face detection - The goal is to distinguish faces from non-faces.

Viola Jones algorithm has four main steps:

- Haar Feature Selection
- Creating an integral image
- AdaBoost training
- Creating classifier cascades

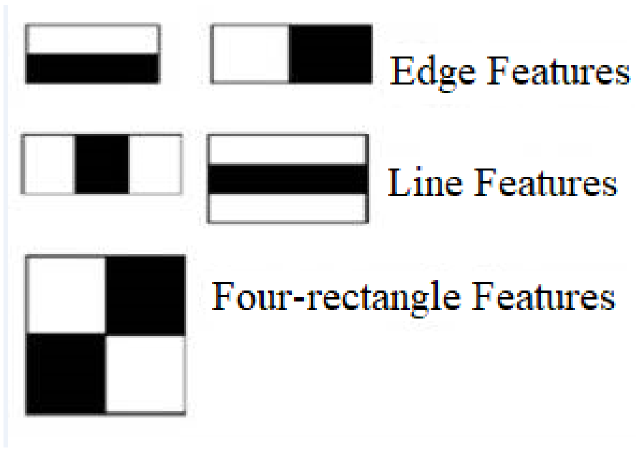

Fig. 1. Haar-like features

A. Haar Feature Selection

Faces have few unique features like eyes region is darker that it's surrounding regions. These can be identified by extracting Haar-like features. There are 3 types of Haar-like features that Viola and Jones identified in their research:

- Edge features
- Line features
- Four-sided features

Each feature is simply calculated by subtracting sum of pixels in white rectangle and sum of pixels in black rectangle.

B. Integral Image

For large images, calculating Haar-like features can be intensive. So we find the integral image, which is basically a summed-table. In integral image each point has a value of sum of all the pixels to the left and above that point including the target point. This can be used to calculate Haar-like features, which are rectangle, by adding the diagonal elements and subtracting them. This algorithm is so efficient.

C. Adaboost Training

Now that we have extracted large number of features, we need to identify best features useful for detecting a face. This is done by using AdaBoost.

D. Creating classifier cascades

This is another step to reduce the computing complexity further. We need to check all the sub-regions of an image for a face by calculating features for that sub-region. The job of cascade is to check for a sub-region inside that sub-region for important features. Detecting a face is now divided in stages and in the initial stages the sub-regions are checked for best features like eyes or nose and if these best features are not present they are discarded and the algorithm is proceeded to next stages.

IV. Mask Detection system

In this module we train models on existing dataset to predict if the detected face in the module is wearing a mask or not. We first trained our CNN models on the dataset given to us in the assignment references. But this dataset has only 1000 images and so the accuracy of real time mask prediction was quite low.

So, we used a dataset from kaggle (linked in references) which had 12k images to work with. We tested with three existing CNN models (alexnet, resnet, and vgg19) with the help of tensorflow and Keras libraries. To understand these models, we need to have a basic understanding of CNNs. There are three main layer categories in any CNN model, convolution, pooling and fully connected layers.

Convolution Layer: In each convolution layer, a kernel strides through the input matrix and performs dot product. The size of the kernel and the stride length can be modeled using hyperparameters. Additionally, we can also define the number of such kernels to use on the input image. These are called filters and the number of filters to use is also passed as a hyperparameter. Convolution is accompanied by Activation functions. There are diff activation functions, the most popular one used here is Relu. There can be many such convolution layers.

Batch Normalisation and Maxpooling Layers: After convolution "Batch Normalization" and "Maxpooling" are usually applied. Batch Normalisation standardize and normalize the inputs. In the Max pooling layer, we select a window and slide it across the input matrix. For example, if the window size is 2x2, it slides the input matrix outputting the maximum value found in that window. MAx pooling helps in reducing noise and preserves significant features. The size of the matrix is also considerably decreased after pooling.

Flattening layer: After multiple convolutions layers are used, we flatten the matrix into one dimensional array.

Fully Connected layers(FC): Fully connected layer is a feed forward neural network. This is the final step in CNN. The flattened matrix is sent as an input to this layer. In a single FC layer, the input matrix is multiplied with a weight matrix to give a 1d array of size = The number neurons in the present layer. Bias is usually added to this matrix and is sent into an activation function. This process is repeated for each layer. The final FC layer will output the required classification. In our case, a binary classification to see if the faces are wearing a mask or not.

Activation Functions: Activation functions decide if it should be activated or not. It takes the weighted sum with added bias as input. The two popular activation functions which are used in our models are ReLu and Sigmoid. Relu is

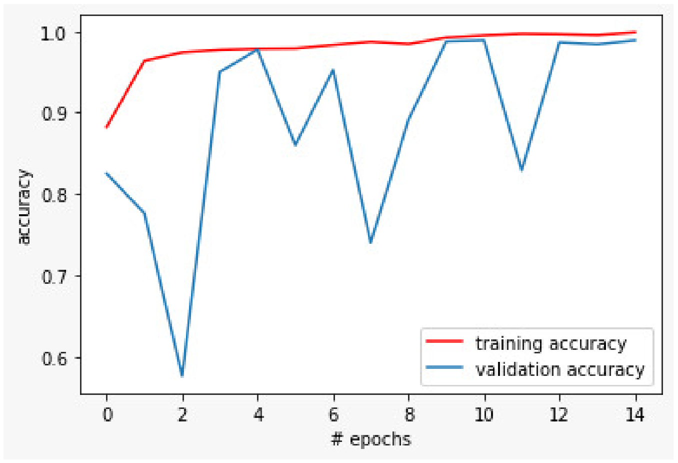

Fig. 2. Alexnet

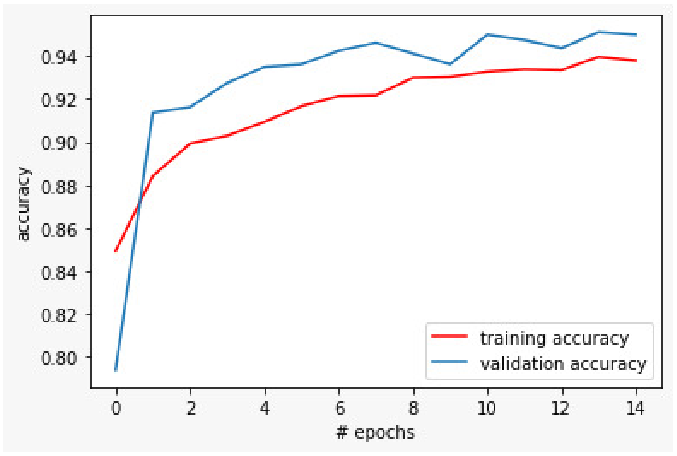

Fig. 3. Resnet

short for rectified linear. This function will output the input if it is positive or 0 if the input is negative. Sigmoid function takes an input and converts it into a value between 0 and 1. This is usually used in the final FC layer when the output is binary to get the probability of the outcomes.

A. *AlexNet:*

ALexnet has 5 convolution layers and 3 fully connected layers. The hyperparameters of these layers are given in Alexnet. The last layer of Alexnet is modified for binary classification. Alexnet is one of the simplest CNNs to implement. Face detected is grayscaled and reshaped to a 128x128 matrix. The model is then compiled. Binary_crossentropy for loss and accuracy metric are given as hyper parameters during compiling. The model is then fit on traindata (10k images) and a validation set of 1k images is used. The model is fit in 15 rounds (15 epochs) for better accuracy and callbacks are used. After each epoch callback is called to correct the model. In this model we monitored the loss on validation set and if the loss doesn't change for 5 epochs, we half the learning rate for a better fit. For this "ReduceLROnPlateau" from keras utilities is used.

This model gave 99.88 percent accuracy on train data and 98.8 percent accuracy on the validation data. This model is then tested on 1k images. We got an accuracy of **99.39** percent.

B. *Resnet50:*

Resnet50 has 50 different layers. The number of convolution layers and the hyper parameters are all pregiven. We define a model which applies resnet on the input image then performs pooling, Batch Normalisation in sequence. We then get a 1d matrix. We randomly drop 30 percent of the values. Dropout prevents overfitting of the model. Then we apply two dense layers (FC layers) to predict the label of an image. ReLu and Sigmoid are used as activation functions in each of these respectively. This model is composed and fit similar to ALexnet using the same callbacks.

This model gave 94.0 percent accuracy on train data and 95. percent accuracy on the validation data. This model is then tested on 1k images. We got an accuracy of **94.6** percent.

These models are trained in kaggle and are then downloaded to local systems. These models are loaded to our main final.py python file. The face detected in the above modules is then predicted by the loaded CNN models. Based on the prediction of these models, the frames are labelled Mask or No Mask.

V. Conclusion

To check this model performance we created demo videos with all our team members where each member provides his live cam feed to model and in output we get labels whether the person is wearing mask or not and a bounding box around his body. In all four cases we got almost expected output. While detecting face when wearing mask it requires good lighting on face otherwise it is not detecting the face. As we are running this model on our local laptops which have less processing power we experienced a lag in output video. We got better accurate results when we use alexnet model. The accuracy of this model is 99.39 percent. Using YOLO we got around 50-60 percent accuracy.

Acknowledgment

We would like to thank Prof Dinesh Babu for giving us an opportunity to work on this amazing project. It was a great learning experience while working on the project and learning new practical things was also very enjoyable.

References

[1] https://medium.com/@luanaebio/detecting-people-with-yolo-and-opencv-5c1f9bc6a810
[2] https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/: :text=YOLO
[3] https://www.mygreatlearning.com/blog/viola-jones-algorithm/
[4] https://towardsdatascience.com/the-intuition-behind-facial-detection-the-viola-jones-algorithm-29d9106b6999
[5] https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework
[6] https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset