Face Mask Based Door Automation System

Khaveesh Nagappan
IMT2018036
khaveesh.nagappan@iiitb.org

Sai Rithwik M
IMT2018061
sai.rithwik@iiitb.org

Sriram G
IMT2018526
sriram.g@iiitb.org

Utkarsh Agarwal
IMT2018082
utkarsh.agarwal@iiitb.org

***Abstract*— Through this report, we have described a process to make an automatic door entry system using a combination of various modules, to check if a person is wearing a mask or not, and further allow their entry by passing this information to the door.**

***Index Terms*— YOLO, Faster RCNN, Viola-Jones, human detection, face detection, mask detection, Haar feature, Integral Image, Adaboost, ResNet-34**

Problem Statement

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

Modules:

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

Dataset

For dataset training we had used a dataset of 687 images with and without mask respectively. A validation set of similar size was used. The trained model was then tested on a real time capture from camera. The dataset can be found from here[1]. We had also trained on another dataset which was procured from external resources, which will be mentioned in the further parts of the report.

Introduction

The world is recovering from the CoVID-19 pandemic. Many essential measures are required to fight the virus, and one of them is wearing masks to prevent the spread of the virus. With the opening of various places like restaurants, movie halls etc. it has become essential to make sure masks must be worn to prevent the spread of the virus. An automatic door based system must be implemented for the same too so that people wearing masks are only allowed entry. This reduces a lot of manpower. This feature when merged with various other features like temperature sensor etc. can make sure there is a very less chance of the spread of virus.

This project is divided into three modules, where, in each module we detect the bounding boxes for each object being detected in that specific module and pass it to the next module. The first module detects humans in the image and then passes it to the second module where the face is detected, and it is then passed to the third module where a classifier detects whether the mask is present or not.

There has been a lot of research in face mask detection lately. Though there are many implementations of face mask detection models using CNNs, the focus has shifted to provision of these services for devices with lesser capabilities, like embedded devices, CCTV's, mobile phones etc.

The following document is divided into 6 sections. **Sec. 1** talks about the preprocessing phase of the project, on how data was passed before any operations could be performed on it. **Sec. 2** speaks about our architecture where we discuss the modules in detail, and the parameters used. **Sec. 3** and **4** deal with discussion of demo videos and their respective results. **Sec. 5** and **6** discusses the challenges faced while designing the architecture and the conclusion.

Preprocessing

There has been a minimal amount of preprocessing as the data that we received was preprocessed data. Initially, when we tried to make Viola-Jones algorithm from scratch there was a considerable amount of preprocessing involved. For that we had to first convert the image into grayscaled image and then we had to rescale the image to 25 x 25 dimension to reduce the amount of computations. In almost every other case the data which was given to us was preprocessed.

Architecture

A. Human Detection Module

We made 2 separate architectures for human detection. One using Faster RCNN and one using YOLO. There was a huge difference in the time that took for the model to predict humans among these cases.

Object detection is a task in computer vision that involves identifying the presence, location, and type of one or more objects in a given photograph. It is a challenging problem that involves building upon methods for object recognition (e.g. where are they), object localization (e.g. what are their extent), and object classification (e.g. what are they).

B. YOLO

In recent years, deep learning techniques are achieving state-of-the-art results for object detection, such as on standard benchmark datasets and in computer vision competitions. Notable is the "You Only Look Once," or YOLO, a family of Convolutional Neural Networks that achieve near

state-of-the-art results with a single end-to-end model that can perform object detection in real-time.

The approach involves a single deep convolutional neural network (originally a version of GoogleNet, later updated and called DarkNet based on VGG) that splits the input into a grid of cells and each cell directly predicts a bounding box and object classification. A result is a large number of candidate bounding boxes that are consolidated into a final prediction by a post-processing step.

Although the accuracy of the models is close but not as good as Region-Based Convolutional Neural Networks (R-CNNs), they are popular for object detection because of their detection speed, often demonstrated in real-time on video or with camera feed input.

A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

C. Faster RCNN

Faster R-CNN is an object detection model that improves on Fast R-CNN by utilising a region proposal network (RPN) with the CNN model. The RPN shares full-image convolutional features with the detection network, enabling nearly cost-free region proposals. It is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. RPN and Fast R-CNN are merged into a single network by sharing their convolutional features: the RPN component tells the unified network where to look.

As a whole, Faster R-CNN consists of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector that uses the proposed regions.

- Pre-train a CNN network on image classification tasks.
- Fine-tune the RPN (region proposal network) end-to-end for the region proposal task, which is initialized by the pre-train image classifier. Positive samples have IoU (intersection-over-union) > 0.7, while negative samples have IoU < 0.3.
- Train a Fast R-CNN object detection model using the proposals generated by the current RPN
- Then use the Fast R-CNN network to initialize RPN training. While keeping the shared convolutional layers, only fine-tune the RPN-specific layers. At this stage, RPN and the detection network have shared convolutional layers!
- Finally fine-tune the unique layers of Fast R-CNN
- Step 4-5 can be repeated to train RPN and Fast R-CNN alternatively if needed.

Models in the R-CNN family are all region-based. The detection happens in two stages: (1) First, the model proposes a set of regions of interest by select search or regional proposal network. The proposed regions are sparse as the potential bounding box candidates can be infinite. (2) Then a classifier only processes the region candidates.

The other different approach skips the region proposal stage and runs detection directly over a dense sampling of possible locations. This is how a one-stage object detection algorithm like YOLO works. This is faster and simpler but might potentially drag down the performance a bit.

YOLO makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image.

In our architecture we had tried out both YOLO and Faster RCNN, and from our experiments we found out that for a real time architecture, it would be efficient if there is low computational power, and hence a model like YOLO would be beneficial as it is compuationally way more efficient than Faster RCNN.

D. Face Detection Module

Once the humans have been detected, the cropped images are then sent to the face detection module, where we have used the Viola-Jones algorithm to detect faces. Viola-Jones algorithm consists of 5 steps to detect the bounding box for the face.

- **Defining Haar-like features:** According to the paper, there were 4 default Haar-like features having patterns of black and white rectangles combined together. Using these patterns, a combined sum of intensities of light regions are subtracted with the sum of intensities of dark regions to produce a singular value. Using these Haar features, we could track the useful data regarding images mostly regarding their features such as edges, straight lines or edges.
- **Integral Images:** One of the biggest issues with the calculation of intensities is to calculate the sum of intensities and then subtract the difference of them we might need to loop around again and again on the pixels. One of the ways to reduce this complexity is to find the integral values for the image and then in a constant amount of time we can operate the values of the intensities. In this, we utilise the advantage of the fact that each feature's rectangular area is always adjacent to at least one other rectangle. Building upon this inference we can conclude that we can build integral images for a box size of any width and height.
- **Classifier:** Once we have set up the parameters for detection, we need our architecture to predict the images given as input. To do this we use Adaboost to classify, whether a given set of features can be classified as a given object or not.
- **Cascading:** Cascading is a method to improve the speed of prediction by checking if the most important feature of the object is present in the given window. If it is not present in that given window no other features are looked at in that window. Due to this, the prediction time decreases by a lot.

Of the mentioned 4 steps, the first three steps had already been performed and given in the form of XML file and the cascading can be done using an OpenCV function.

E. Mask Detection Module

Mask detection is done using the CNN binary classification. We used pre-trained ResNet-34 from `torchvision.models` as the trunk of the model to exploit the good use of residual connections.

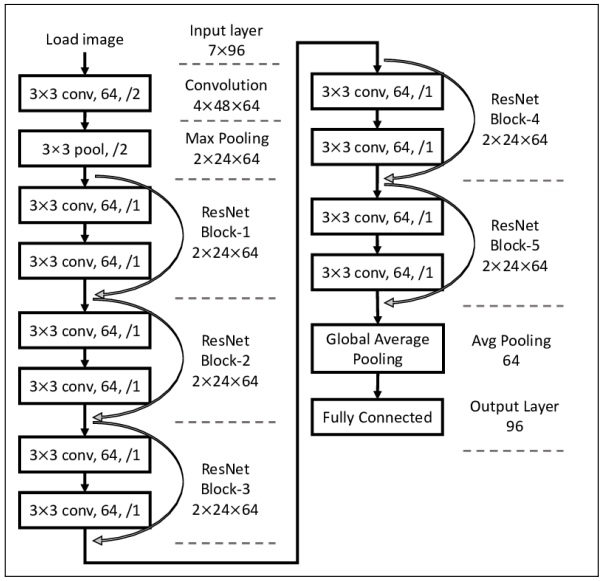


Fig. 1. ResNet Architecture[2]

ResNet is a really deep and one of the SoTA architectures in image classification. The brilliant idea of residual skip connections was much better than auxiliary losses (in InceptionNet) in order to solve the vanishing gradients problem. Unlike other models, this uses a uniform kernel size of 3x3 consistently. Smaller kernel sizes mean it has a good chance of predicting well with small objects.

The model was trained on link[3] and we obtained a test accuracy of 99.60%. This dataset was really good with a lot of augmented images and hence making the model even more powerful. It had a collection of different types and colours of masks to make detection robust.

The CNN weights were frozen and a simple classifier was trained on top of the features. The pretrained model was trained on ImageNet. So we resized all input images to 224x224 which is usually used with ImageNet. Since ImageNet is a large image classification dataset, the transfer learning worked seamlessly and the classifier was learnt in just a few epochs. Fine-tuning was not attempted since the weights learnt from ImageNet were good and powerful enough.

The final pipeline architecture is as follows:

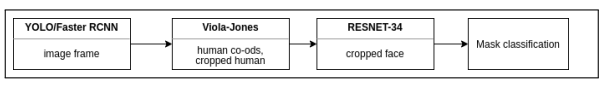


Fig. 2. Pipeline Architecture

Demo Images and Videos:

We had performed the tests on various people, and the results are stored in the drive, whose link is shared here[4]

Results

For the human and face detection modules, we had used pre-trained models and ran the frames through them. The mask detection module was built using ResNet-34 which had a training accuracy of 99.56% and a test accuracy of 99.60%. Further details on the epochs are mentioned in the graphs below.

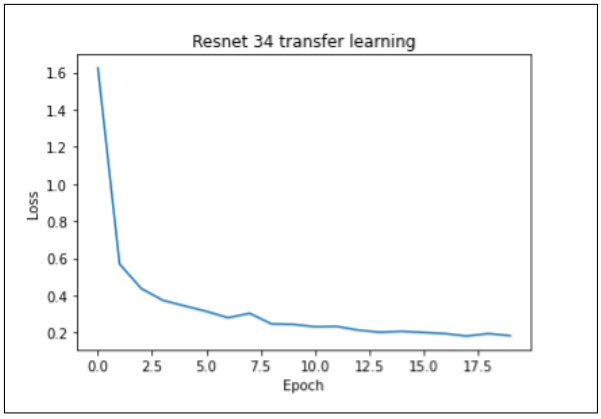


Fig. 3. Epoch vs Loss for ResNet

A detailed table on training for ResNet is as follows

Epoch	Loss Value	Time to run(s)
1	1.47296	27
2	0.59786	40
3	0.46067	40
4	0.36428	41
5	0.33672	40
6	0.29789	40
7	0.27779	40
8	0.26387	40
9	0.23533	41
10	0.24773	41
11	0.22214	41
12	0.23480	41
13	0.21025	40
14	0.24426	41
15	0.19942	40
16	0.19674	41
17	0.18832	41
18	0.20278	40
19	0.18886	41
20	0.18651	40

TABLE I

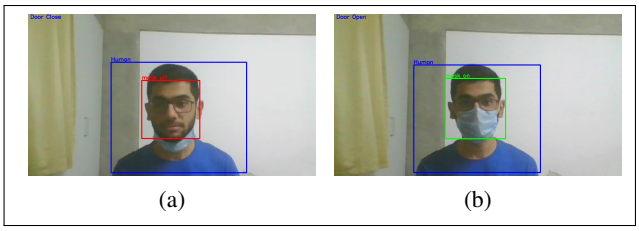
Epochs, Loss values vs Time to run

Epoch Batch	Test %	Train %
Batch 1	99.09	99.26
Batch 2	99.29	99.42
Batch 3	99.60	99.56
Batch 4	99.29	99.60

TABLE II

Epoch batches, Train % and Test %

We can clearly see that the given pipeline is a robust one and can clearly be used in real time scenarios.



(a) (b)

Fig. 4. Result on passing through pipeline

CHALLENGES AND SCOPE

One of the most important challenges we faced was the limited computation power that we had. Even though we made sure we had pre-trained models and done the majority of the training on servers, the human detection module still was computationally intensive for a CPU based system. Even though this pipeline can also accommodate multiple people in a single image, there is a lot of scope of improvement for the given project. This project can then be merged with a temperature sensor and a decision manager can be used to check if a person can come inside a given room or not.

CONCLUSION

Overall this is a production-ready model with very high classification accuracy. It can detect the presence or absence of a mask among multiple people. Using this tool would be very helpful to stop the rapid spread of the virus.

ACKNOWLEDGMENT

Firstly we would like to thank Prof. Dinesh Babu Jayagopi for teaching such a wonderful course. Without the concepts taught by them, we believe it would have been much harder to understand what we were doing. Thanks to Soham Joshi and Sowmith Nandan, who helped us with the videos. Special Thanks to Saksham Agarwal, Kartik Udupa with whom we had very fruitful discussions on how to approach this problem, and for sharing tips.

Overall this project has been a great learning experience for the entire team and we hope to implement the concepts we learnt here in further courses and competitions.

REFERENCES

[1] Dataset of people with and without masks
[2] ResNet Architecture Diagram
[3] Kaggle 12k image dataset
[4] Drive Link of the code and models
[5] YOLO - Code implementation in pytorch Darknet