1 Human Detection

The objective of Object Detection in an image is the following,

1. Identify the objects of interests present in the images along with their locations.
2. Classify and filter out objects that are of interest.

1.1 Object Detection Algorithms

There are many techniques and algorithms under Object Detection.
The initial methods were using divide and conquer to perform structured divisions of the images. The divisions were passed through classifiers to obtain class-wise classification results.
This was followed by deep learning based methods. R-CNN, Fast R-CNN, Faster R-CNN, YOLO are some of the popular object detection algorithms.

1.2 YOLO

You Only Look Once (YOLO) framework is an effective object detection algorithm. It is fast because it computes the bounding boxes and the class probabilities at the same time. Darknet-53 is a convolutional neural network that acts as the backbone architecture of YOLO detection algorithm.
Darknet-53 includes additional layers and residual connections as compared to Darknet-19. YOLO is trained on MS-COCO dataset.
We have selected YOLO as the person detection algorithm due its speed and effectiveness in real time detection.

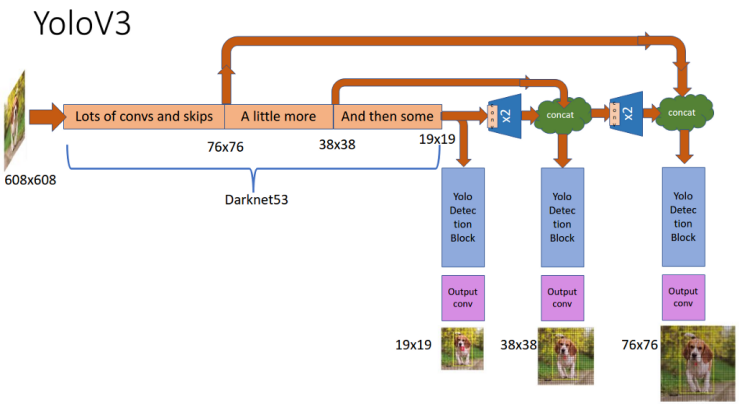


Figure 1: YOLOV3 Architecture

YOLO Algorithm
It works on the principle of regression and predicts classes and bounding boxes for the whole image in one shot instead of selecting region of proposals(used in R-CNN based networks).
Each bounding box can be described using four descriptors:

1. Center of the box
2. Width
3. Height
4. Class of the object

During the one pass, YOLO determines the probability that the cell contains an object belonging to a certain class.
Class labels with confidence score is obtained. The class with the maximum probability is chosen and assigned to that particular grid cell. Similar process is repeated for all the grid cells present in the image.
After predicting the class probabilities, the next step is Non-max suppression, it helps the algorithm to get rid of the unnecessary anchor boxes.
Non-max suppression eliminates the bounding boxes that are very close by preforming the IOU with the one having the highest class probability among them.

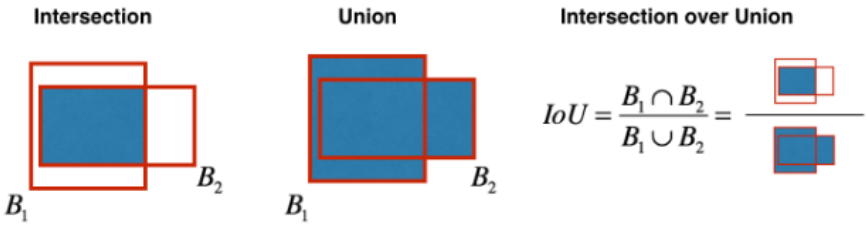


Figure 2: IOU : Bounding box Elimination

It calculates the value of IOU for all the bounding boxes respective to the one having the highest class probability and then it rejects the bounding boxes whose value of IOU is greater than a threshold.
It signifies that those two bounding boxes are covering the same object but the other one has a low probabilty for the same, thus it is eliminated.
Finally, the bounding box is obtained as a vector.

1.3 YOLO version4

YOLO v4 is developed by three developers Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao.
YOLOv4's architecture is composed of CSPDarknet53 as a backbone, spatial pyramid pooling additional module, PANet path-aggregation neck and YOLOv3 head.

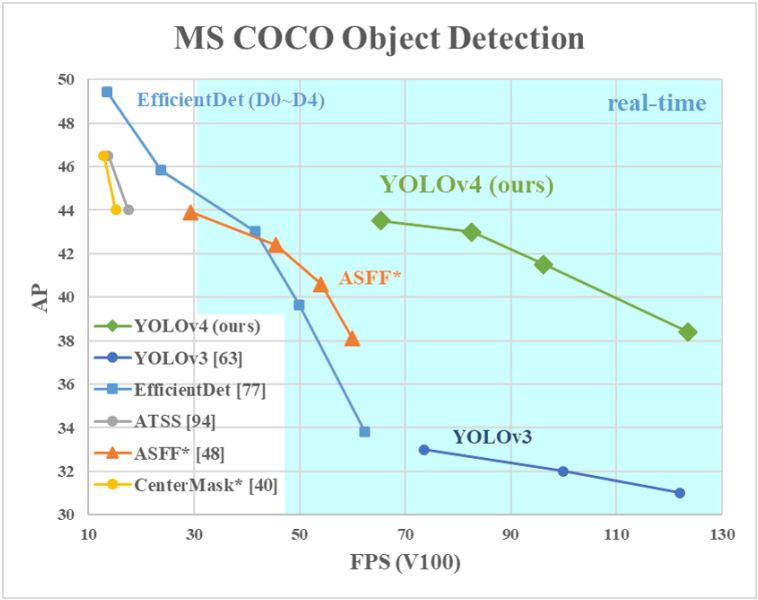


Figure 3: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors.

Average precision improves by 10% and Frames per Second improves by 12% in YOLOv4 compared to YOLOv3.

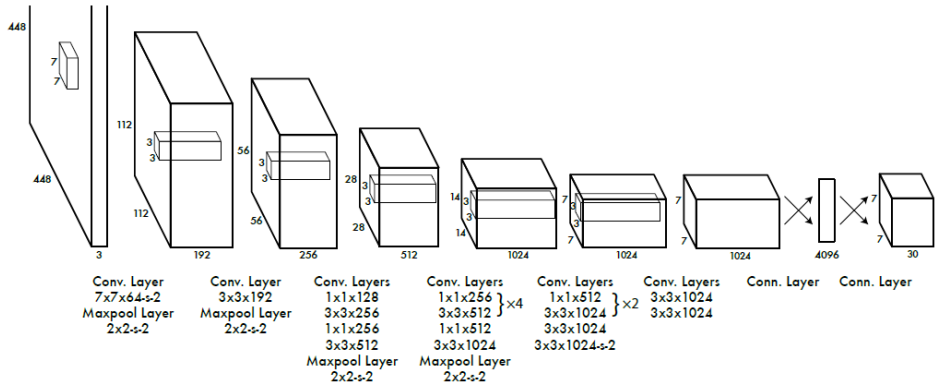


Figure 4: YOLOV4 Architecture

2 Face Detection

2.1 Viola Jones Algorithm

Viola-Jones algorithm is an object-recognition framework that allows the detection of image features in real-time. Viola-Jones is quite powerful and its application has proven to be exceptionally notable in real-time face detection.

There are 2 stages in the Viola-Jones Algorithm:

1) Training
2) Detection

Before detecting a face, the image is converted into grayscale, since it is easier to work with and there's lesser data to process. The Viola-Jones algorithm first detects the face on the grayscale image and then finds the location on the colored image.

Viola-Jones outlines a box and then searches for a face within the box. It is essentially searching for haar-like features.

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

1. Edge features
2. Line-features
3. Four-sided features

The Haar like features get converted into a grid where each square represents a pixel and the numbers stored in the grid are the darkness values.

Using these values we get an idea about the edges on moving from bright pixel to dark pixel, where the dark pixel represent the edges which help in detection.

The algorithm learns from the images we supply it and is able to determine the false positives and true negatives in the data, thus becoming accurate.

2.2 Issues Faced

2.2.1 haarcascade_frontalface_defualt

Initially, we were performing face detection by using the haarcascade_frontalface_default.xml present in the data folder of opencv.

However, we could notice that the detection was flawed since:

1. There were multiple bounding boxes where there must have been only 1

2. Too many false positives were occurring

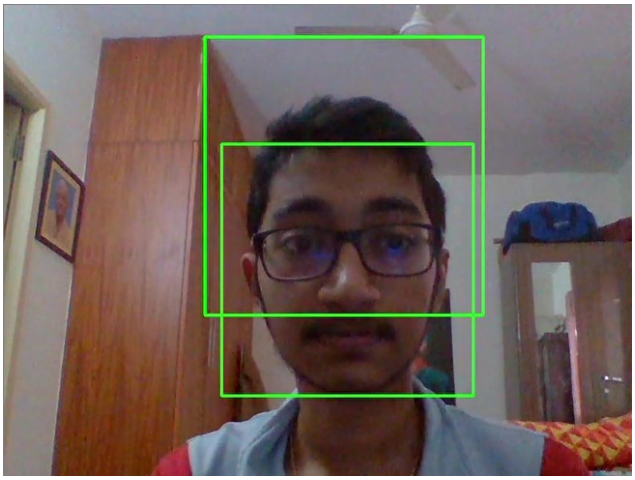


Figure 5: haarcascade_frontalface_defualt.xml

2.2.2 haarcascade_frontalface_alt2

Seeing as to how haarcascade_frontalface_default.xml wasn't working out for us, we shifted to another xml present in the data folder of opencv haarcascade_frontalface_alt2.xml.
Using this gave us much better results than previously since there almost most of the major issues we came across before were now solved.

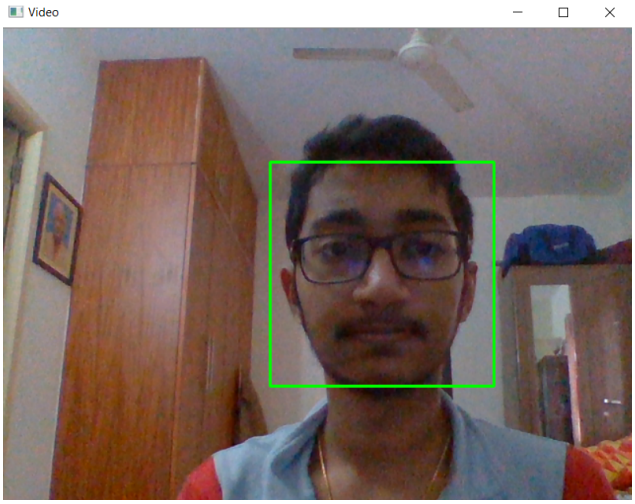


Figure 6: haarcascade_frontalface_alt2.xml

However, there were still some detection issues we were facing:

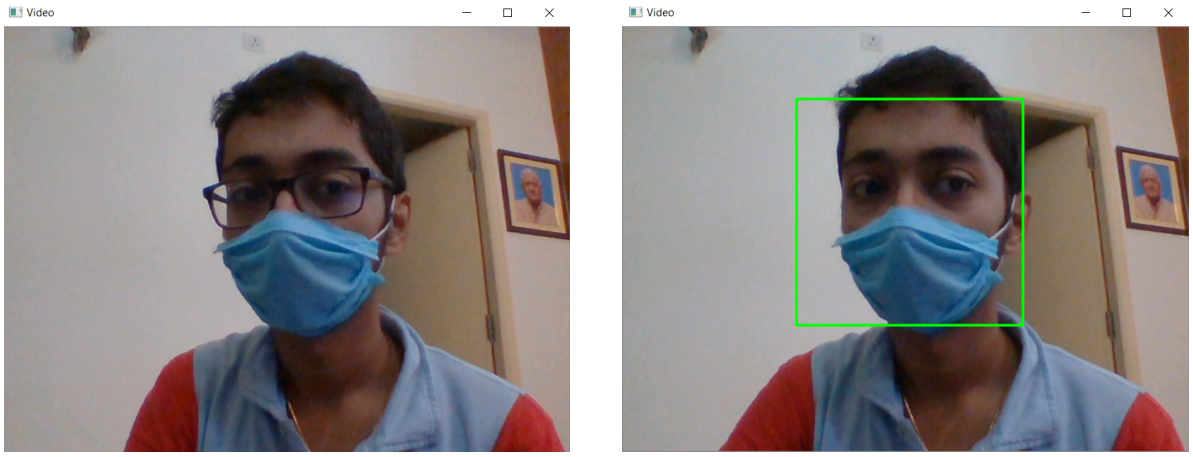
1. Glare from the spectacles worn on the face sometimes confused the model
2. It was not trained on masked faces

2.3 Haar Training

2.3.1 Problem

Though false negatives, such as the ones shown/mentioned above, were extremely rare, we felt this will have a non-negligible impact since nearly 20-30% of the Indian population (65% of the global population) wears glasses.

Thus we felt the need to train a new model, on Haar features, which would perform well on all kinds of faces, be it masked faces, spectacles wearing faces and even masked faces wearing spectacles.



(a) Glasses & Mask problem (b) Only Mask

Figure 7: Bounding box prediction with/without spectacles

2.3.2 Procedure

1. We took 10-20 photographs of our faces (with and without masks, spectacles) and cropped each one of them to get rid of background using Image Clipper.
2. We then downloaded the necessary linux executables for the Haar Training.
3. The positive images were images where we had our face cropped.
4. For the negative images, we took random images without faces.
5. Then using the linux executables, we created a haarcascade xml file containing the features.

However, the result, contrary to what we were expecting, was a huge dip in accuracy compared to before.

2.3.3 Conclusion

- Since the number of positive images we used were very less compared to what is generally expected to be used for training and also due to the dataset used by us for the negative images were random, we got much lesser accuracy in face detection.
- Generally a minimum of 5000-10000 positive images (and many more negative images) are required to create a robust classifier.
- Training such large models on our laptops (either physically or on sites such as google colab and Kaggle) is definitely not efficient, since it requires either enormous amount of resources and time.

Thus we decided against implementing our custom Haar trained classifier file and instead opted to implement our previously used Haar cascade classifier file (haarcascade_frontalface_alt2.xml).

3 Mask Detection

3.1 Dataset

As we implemented face detection, our next task was to build a classifier that can predict whether or not a person is wearing a mask. We built a basic CNN model and tried on a dataset that consisted of 1376 images, 690 face images with masks, and 686 without masks and are available here. Shown below are the samples from the dataset.

Figure 8: Old dataset

The images in the dataset contained a single mask with different orientations. Hence, the model was unable to detect masks of a different orientation, color and sizes. Therefore, to build a robust model, we used a dataset that is available here. Shown below are the samples from the dataset.

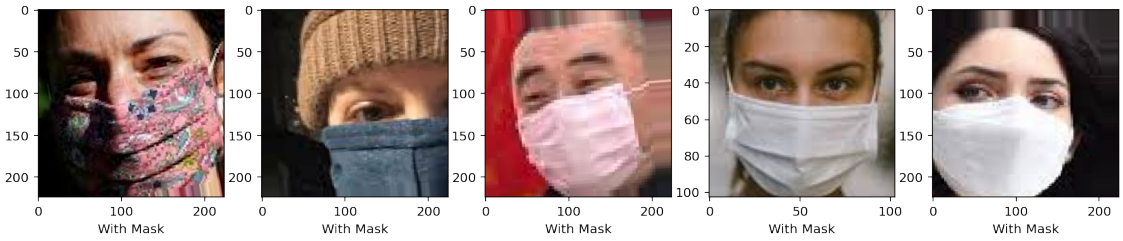


Figure 9: Images with mask

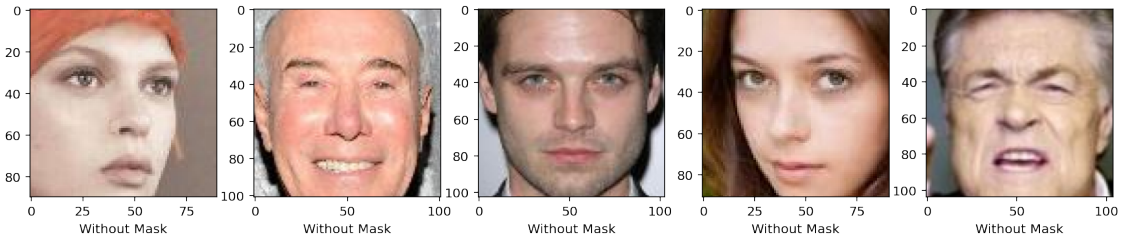


Figure 10: Images without mask

This dataset contains authentic images, and hence we got better accuracy when we trained our model on this dataset. It contains 10,000 images in its train section and around 1,000 images in its test section. For our final mask detection, we used a model trained on this dataset.

Finally, we created a transform function that resized all the images present in the dataset and converted them to tensors used for training and testing our classifier model. We also applied this transform function on our frames of the video before passing them to our model.

3.2 Classifier Architecture

Now for classifying the images, we used pre-trained models such as alexnet and resnet.

The final layer of a CNN model, which is often an FC layer, has the same number of nodes as the number of output classes in the dataset. Since all the models have been trained on Imagenet, they all have output layers of 1000 classes, one node for each class.

Hence, we reshaped the last layer to have the same number of inputs as before while having the same number of outputs as our dataset's number of classes.

For this, we tried two different approaches mentioned below.

3.2.1 Finetuning Resnet

Several variants of Resnet of different sizes, including Resnet18, Resnet34, Resnet50, Resnet101, and Resnet152, exist. We used pre-trained Resnet18, as our dataset is small and only has two classes.

Then we finetuned resnet18 for our dataset, which will have the number of outputs as 2. We used SGD optimizer with a learning rate = 0.001 and with a momentum of 0.9. Further, we used cross-entropy for calculating loss at each epoch and training our model.

We achieved an accuracy of approximately **99%** on our test dataset with our finetuned Resnet model.

Below shown is our loss curve and accuracy curve for resnet.

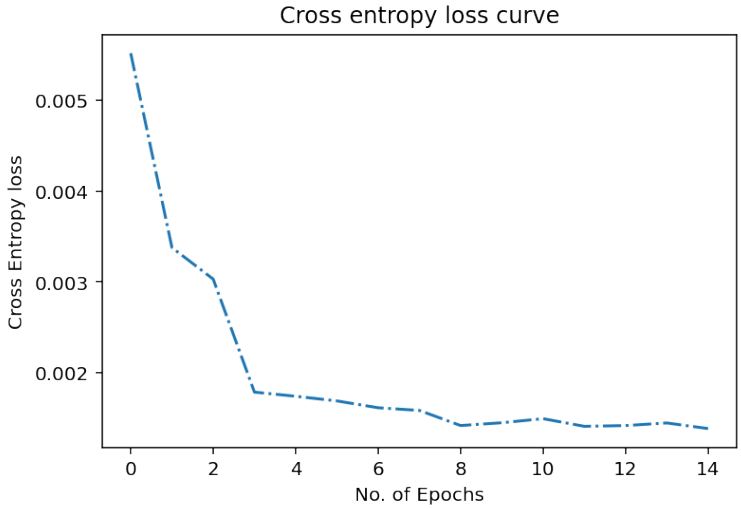


Figure 11: Cross entropy loss curve

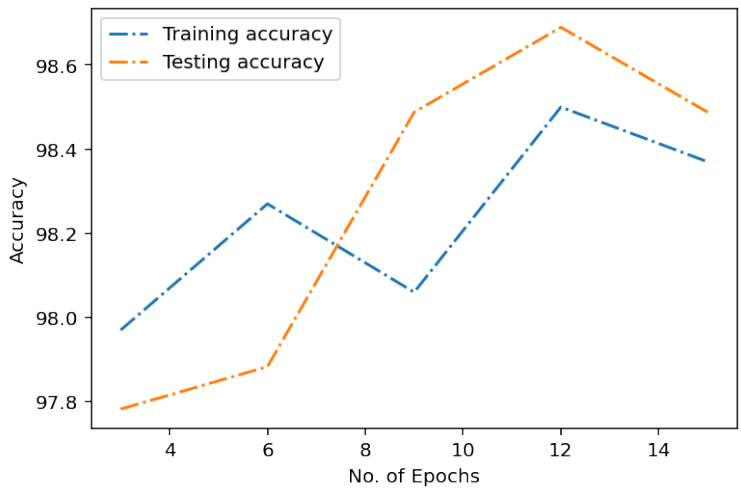


Figure 12: Resnet18 accuracy curve

3.2.2 Alexnet and Logistic Regression

Another approach we tried is to use pre-trained Alexnet for extracting features from the train and test dataset and then using a classifier like logistic regression on the extracted features.

We got a total of **9216** features from alexnet on which we trained Logistic regression, and we got an accuracy of **97%** for this model.
The following is the evaluation metrics for the final Logistic Regression model.

	precision	recall	f1-score	support
0	0.97	0.96	0.97	483
1	0.97	0.97	0.97	509
accuracy			0.97	992
macro avg	0.97	0.97	0.97	992
weighted avg	0.97	0.97	0.97	992

Figure 13: Metrics of Logistic Regression

4 Nose Detection

Wearing one's mask incorrectly is a huge risk both to oneself and those around you. The dataset we used and most of other datasets available focus mainly on whether mask is worn or not. Hence the models trained on such datasets would not be able to detect incorrectly worn mask accurately every time since it share features common to both classes.
Hence we came up with an idea to also detect one's nose after mask detection is done, to see whether or not the person is wearing the mask properly.
We achieved this by following the Viola Jones algorithm and detect the nose, due to it's Haar like features, with the help of cv2.CascadeClassifier.
Currently we are using a xml file containing the necessary features, obtained from a public github repository[Cas+07], to detect the nose. However, this can be further trained upon to achieve better results; if needed.

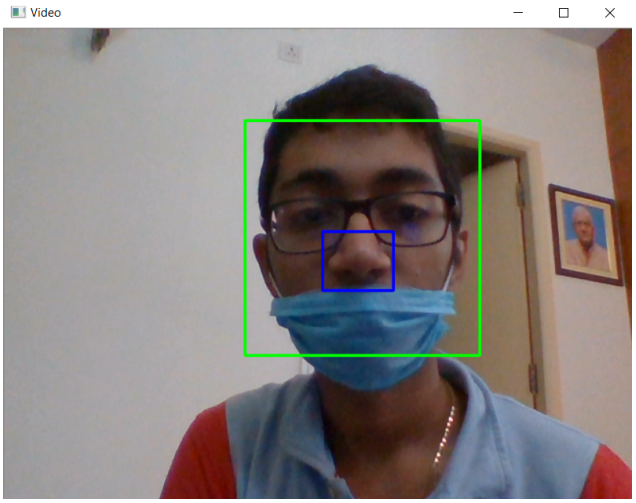


Figure 14: Nose Detection

5 Experiments

Our pipeline for the mask detection is as follows,
1. Human detection using YOLO.
2. Face detection using Viola Jones.
3. The cropped image is then passed to our classifier.
4. Nose detection is performed to check if mask is worn correctly.
Following are the snapshots illustrating the working of the program.

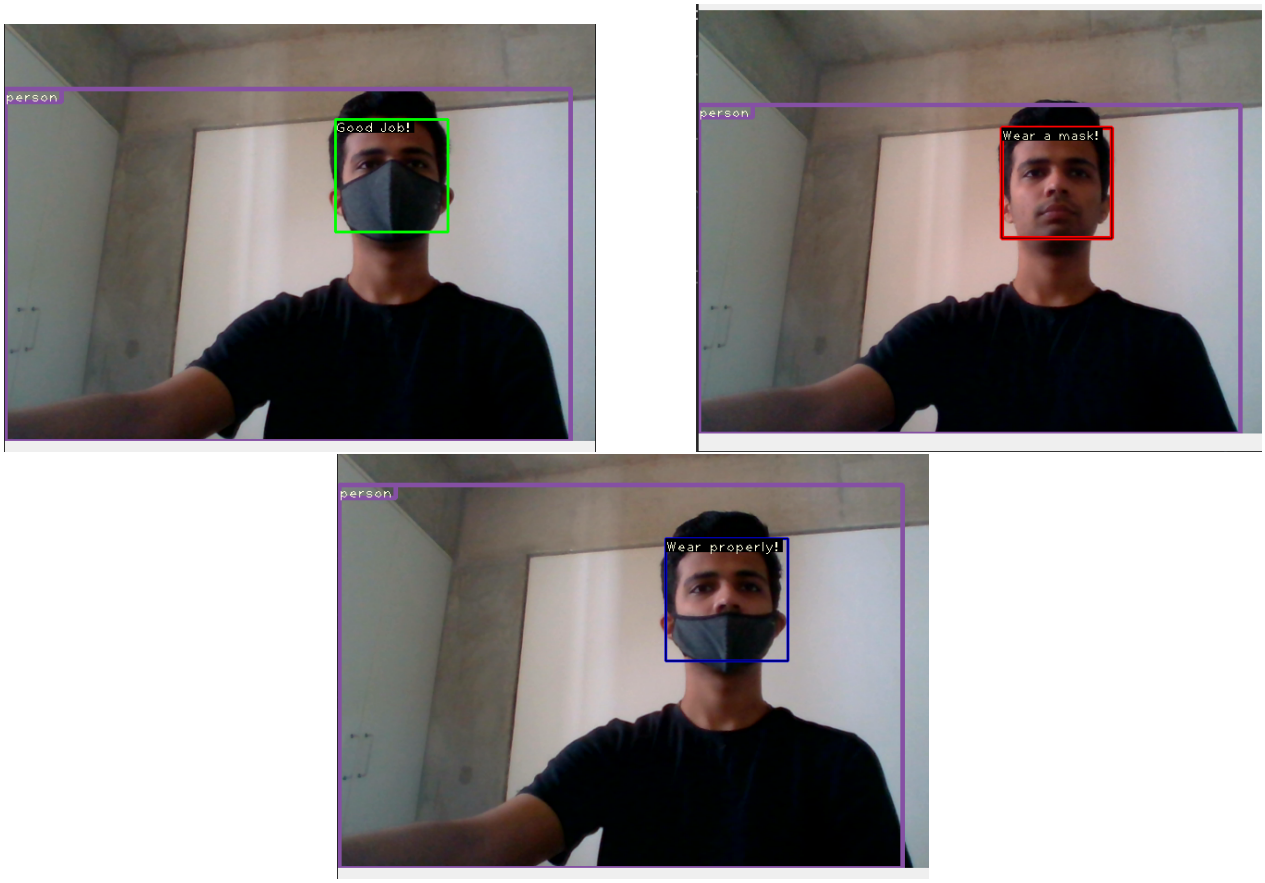


Figure 15: Single user : Mask, No Mask and Incorrect Mask

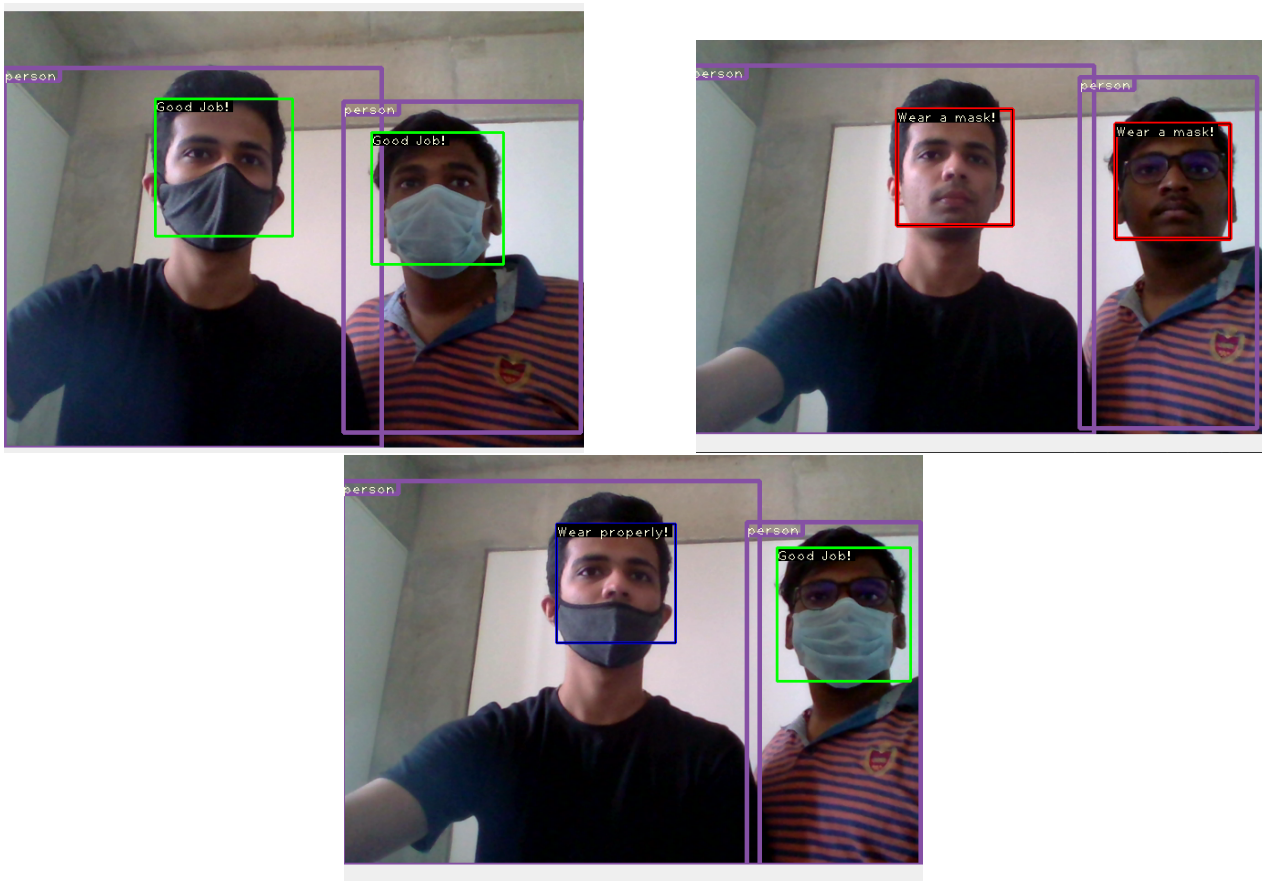


Figure 16: Multi user : Both Mask, Both No Mask, Mask and Partial Mask

6 Conclusion

Our system works considerably well in real time mask detection. As our YOLO is running on GPU, we experience no lag with only 1 or 2 people present in the frame.
Nevertheless, A minute lag can be noticed when there are many people in the frame, since the computation need increases, due to the Human, Face, Nose and Classification models running sequentially on multiple people.
This can be solved by making use of a better GPU allowing for faster computations.

The real time video demonstrations can be found in the following link.
Drive Link of demo videos

References

YOLO V3 PyTorch (Github)
YOLO Architecture (Medium)
YOLO Working
Comparison of Object localization models
Face Detection
Data loading in pytorch
Face mask detection
Finetuning pytorch models
CNN transfer learning and finetuning

Bibliography

[Cas+07] M. Castrillón Santana et al. “ENCARA2: Real-time Detection of Multiple Faces at Different Resolutions in Video Streams”. In: *Journal of Visual Communication and Image Representation* (2 Apr. 2007), pp. 130–140.