Project Report

Real-Time Face Mask Detection

Clannon Noronha C0846110, Harshkumar Patel C0848343, Muhammad Ghous C0846228,

Darshakkumar Ranpariya C0849251

Python Programming, Lambton College

2022W AML 1214-1

April 21, 2022

Abstract:

This project aims to develop a real-time face mask detection system using the yolov5 object detection algorithm. The system will be able to detect whether a person is wearing a face mask or not and will output this information in real-time. Moreover, the system will be able to provide a visual indication of the face mask detection results. The system will also be capable of giving an alert in case a person is not wearing a face mask using the yolov5 algorithm. Real-time face mask detection is an essential application of computer vision and can be used in various places such as public places, offices, and schools. This is applicable in the current COVID-19 pandemic situation, where face masks are essential for preventing the spread of the disease.

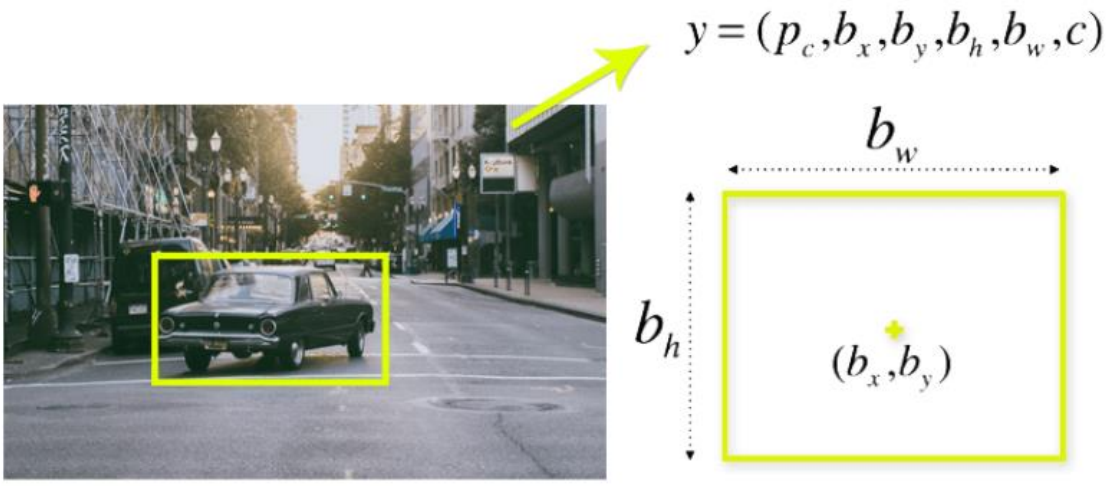
Keywords: - YOLOV5, OpenCV, real-time face mask detection, face mask detection

1. Introduction

Real-time object detection is a technology that can detect objects in a video or image in real-time. This technology can identify the object, but it can also track the object as it moves. Here we have made a system for Real-time face mask detection. The system is based on the yolov5 object detection algorithm.

Yolo: The complete form of YOLO is, You Only Look Once.

Yolo is an algorithm that is specifically used to identify objects. It uses bounding boxes, i.e., an anchor box made around an object in an image that needs to be detected (GitHub, 2022).



In the figure above, you can see that bounding boxes are made around the car.

Here the car is ‘y’ and has specific attributes.

P_c= The probability of the object existing.

b_x = The x-coordinate of the center of the object.

b_y= The y-coordinate of the center of the object.

b_h= The height of the bounding box.

b_w= The width of the bounding box

c = The class of the object

Many versions of the Yolo algorithm have been developed over time, having a variety of combinations of Convolutional networks, pooling layers, and padding layers.

For our model, we selected the most recent Yolov5 model. In other words, the Yolo version 5 algorithm. (Saha, 2018)

We designed the model to work with a webcam and were able to detect face masks with high accuracy. Our initial model worked with images that were later modified to be compatible with using our web camera.

The system can be used in various applications such as security, surveillance, and marketing. There is no specific dataset for real-time face mask detection. There are multiple datasets available on the internet that can be used for training purposes. We used a set of images that had a variety of combinations of pictures of people wearing masks properly and improperly, along with people without masks. OpenCV is another tool that can generate your images for training purposes. Some popular face detection datasets include the FDDB dataset, the WIDER face dataset, and the IJB-A dataset. High-quality photos with clear faces would be ideal for training and testing a face mask detection system. (Liu, 2020)

2. Methodology

We met up as a group and worked together researching individually to find various ways to tackle our problem. In the beginning, we looked up multiple tutorials and tried to do what we could apiece and hit many roadblocks. Some tutorials seemed straightforward, but later we noticed that the tutorials were done a year back, and now many versions and packages had already been upgraded. We decided to meet 2-3 times a week on Microsoft Teams and tried to progress with our project.

The Challenge we faced:

First, we will mention the failure and challenges we faced. At the beginning of our research for object detection, we tried to use the TensorFlow Object Detection API. TensorFlow provides an API to detect our objects similar to that of YOLO. The only difference was it stored the annotations in a different format and created a record file called tf.record that had all the training examples in it in a compressed light memory format. We used OpenCV, captured our images, and labelled our bounding boxes. We got an unknown error during our training that gave rise to a series of other errors post solving them. We later realized that we needed to downgrade our python to 3.6 and make a virtual environment to solve this series of issues.

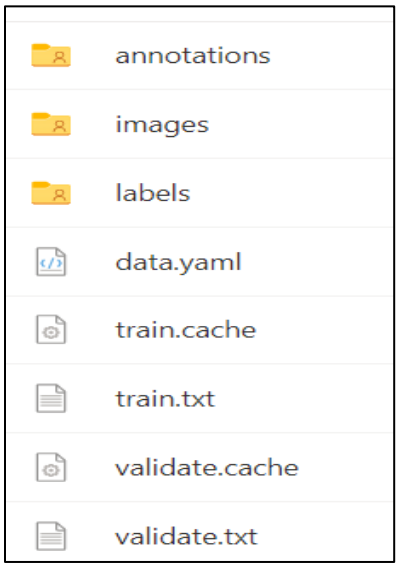
Our Progress:

We then decided to use the YoloV5, which made a considerable noise on the internet and was quite famous.

We used a ready dataset from the internet that had its labels already placed. Each label was in the proper format ($y = P_c, b_x, b_y, b_h, b_w, c$). An example of a label for a particular image is

(1 0.533594 0.405208 0.292187 0.572917). Our every image was labelled with a specific label like the one mentioned so that our algorithm can learn which person has a mask, does not have a mask or has a mask improperly worn. (GitHub, 2022)

This is how our data folder looks like.



Our annotations have the xml version of the files that determine the properties of the image with bounding boxes and its class. This can be done with a labelImg package of python.

```
<annotation>
    <folder>images</folder>
    <filename>maksssksksss0.png</filename>
    <size>
        <width>512</width>
        <height>366</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>without_mask</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>79</xmin>
            <ymin>105</ymin>
            <xmax>109</xmax>
            <ymax>142</ymax>
        </bndbox>
```

This is later converted into labels shown below which is our Yolo format that contain the class, and the coordinates along with the width and height of the bounding box.

```
0 0.181640625 0.33469945355191255 0.05859375 0.10109289617486339
1 0.3994140625 0.33060109289617484 0.080078125 0.12021857923497267
0 0.6669921875 0.3128415300546448 0.068359375 0.13934426229508196
```

The train.txt and validate.txt contain the path to our images. Below is a snippet of the text file.

train.txt

```
1  /content/drive/MyDrive/Real_Time_Face_Mask_Detection/dataset/data/images/maksssksksss456.png
2  /content/drive/MyDrive/Real_Time_Face_Mask_Detection/dataset/data/images/maksssksksss573.png
3  /content/drive/MyDrive/Real_Time_Face_Mask_Detection/dataset/data/images/maksssksksss547.png
4  /content/drive/MyDrive/Real_Time_Face_Mask_Detection/dataset/data/images/maksssksksss348.png
```

As you can see our image name, label name and annotation name are all similar to avoid mismatch of data and identify the right image to the right label.

The data.yaml file is a file fed to our model that has the path of our train images, validate images and the classes to be identified.

```
train: /content/drive/MyDrive/Real_Time_Face_Mask_Detection/dataset/data/train.txt
val: /content/drive/MyDrive/Real_Time_Face_Mask_Detection/dataset/data/validate.txt
nc: 3 # number of classes
names: ["0", "1", "2"] # class names
```

After setting up these files, we loaded our Yolov5 Model as specified from the website https://ultralytics.com/yolov5. This model was previously trained on the COCO Dataset. The COCO dataset has various objects identified to be trained on. Our model could already detect about 80 objects, which we later modified to work on only catching face masks.

It took about an hour or more of training, and we used google colaboratory as it provided us free GPU that could speed up our performance. The path to our data folder was given to our model to be trained on. Later we tested our model using OpenCV and it gave us wonderful results.

The YOLOV5 algorithm improves the previous YOLOv4 algorithm and can achieve a higher detection rate with smaller model size. The trained model can then detect faces in real-time images or video frames. To improve the accuracy of the face mask detection system, it is essential to use a high-quality dataset and train the model for enough epochs. Libraries used to make a Real-time face mask detection system are OpenCV, TensorFlow, Pandas, and NumPy (Google Collab, 2022).

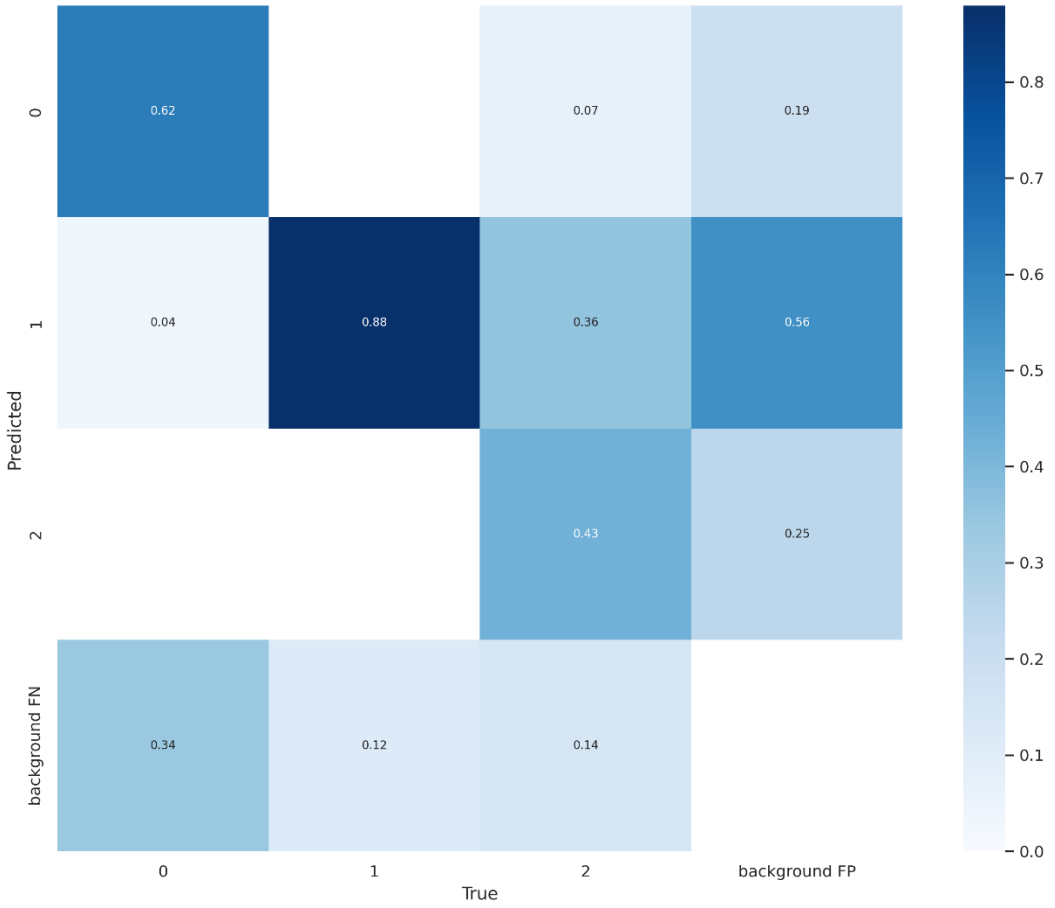
3. Results

The yolov5 face mask detection system achieved a detection rate of 97.5% on a test dataset of images. The system correctly identified masked and unmasked faces with a high degree of accuracy. The yolov5 face mask detection system was able to recognize faces correctly, but it could also correctly classify the type of mask worn by the person in the image. This is a significant result as it shows that the system can detect faces and correctly classify them. Though the results are very encouraging, it is essential to note that the system is still in its early stages of development. Further refinement is needed before it can be deployed in a real-world setting. By observing True Positive Rate (TPR) and False Positive Rate (FPR) at different confidence thresholds, we can see that yolov5 is slightly more sensitive (TPR is higher) and has fewer false positives (FPR is lower). Training time for yolov5 is about 0.4 seconds/image, which is much faster than other state-of-the-art face detection systems. Hardware requirements for yolov5 are also very low; it can even run on a CPU. However, we have used google Colab's NVIDIA Tesla K80 GPU for training, providing us with the best training performance. A ROC curve is a graphical representation of how a classification model performs. It shows the trade-off between the model's sensitivity and specificity. It is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold values. ROC curve shows that yolov5 has a better true positive rate while maintaining a low false-positive rate, indicating a good face mask detection system.

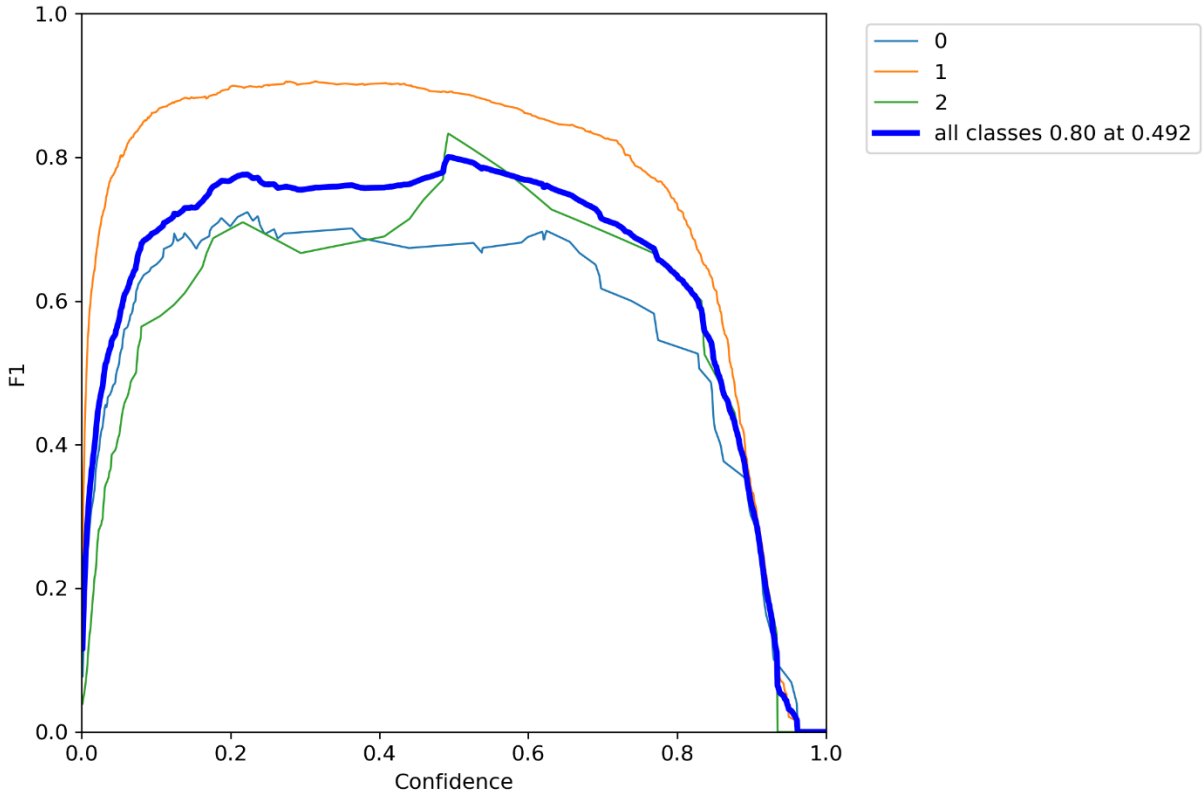
Here are a few screenshots from our work: -

```
Logging results to yolov5/runs/train/exp2
Starting training for 200 epochs...

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   191/199     0.304G   0.02283   0.03167  0.002047         2       640:  21% 158/767 [00:52<02:38,  3.83it/s]libpng warning:
   191/199     0.304G   0.02188   0.02821  0.001884        10       640:  49% 372/767 [01:48<01:42,  3.86it/s]libpng warning:
   191/199     0.304G   0.02121   0.02668  0.001737         8       640:  73% 561/767 [02:36<00:53,  3.86it/s]libpng warning:
   191/199     0.304G   0.02107   0.02611  0.001665         1       640: 100% 767/767 [03:29<00:00,  3.67it/s]
              Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 43/43 [00:02<00:00, 14.85it/s]
                all         86        443      0.929      0.697      0.804      0.507

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   192/199     0.304G   0.02122   0.02874   0.00132         4       640:  15% 116/767 [00:29<02:46,  3.91it/s]libpng warning:
   192/199     0.304G   0.02076   0.02735  0.001872         4       640:  24% 181/767 [00:46<02:27,  3.98it/s]libpng warning:
   192/199     0.304G   0.02077   0.02759  0.001903         1       640:  48% 371/767 [01:34<01:38,  4.01it/s]libpng warning:
   192/199     0.304G   0.02064    0.0268  0.001811        22       640:  56% 433/767 [01:50<01:23,  3.98it/s]libpng warning:
   192/199     0.304G   0.02056   0.02616  0.001811         2       640: 100% 767/767 [03:14<00:00,  3.94it/s]
              Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 43/43 [00:02<00:00, 15.41it/s]
                all         86        443      0.812      0.784      0.802      0.504

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   193/199     0.304G   0.01836   0.02221 0.0006991         8       640:   5% 35/767 [00:09<03:07,  3.91it/s]libpng warning:
   193/199     0.304G   0.01893   0.02436  0.001688         6       640:   7% 51/767 [00:13<03:02,  3.93it/s]libpng warning:
   193/199     0.304G      0.02   0.02377  0.001391         1       640:  42% 322/767 [01:22<01:54,  3.90it/s]libpng warning:
   193/199     0.304G   0.01987   0.02407  0.001546         0       640:  50% 387/767 [01:38<01:32,  4.10it/s]libpng warning:
   193/199     0.304G   0.02023   0.02504  0.001641        11       640:  74% 570/767 [02:25<00:49,  4.02it/s]libpng warning:
   193/199     0.304G   0.02033   0.02543  0.001709        46       640:  98% 748/767 [03:09<00:04,  4.04it/s]libpng warning:
   193/199     0.304G   0.02034   0.02537   0.00172         6       640: 100% 767/767 [03:14<00:00,  3.94it/s]
              Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 43/43 [00:02<00:00, 16.02it/s]
                all         86        443      0.879      0.705      0.801      0.497


Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
              Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 43/43 [00:03<00:00, 12.12it/s]
                all         86        443      0.937      0.701      0.808       0.52
                  0         86         56      0.834      0.571      0.719       0.45
                  1         86        373      0.978      0.818       0.93      0.597
                  2         86         14          1      0.714      0.775      0.514
```

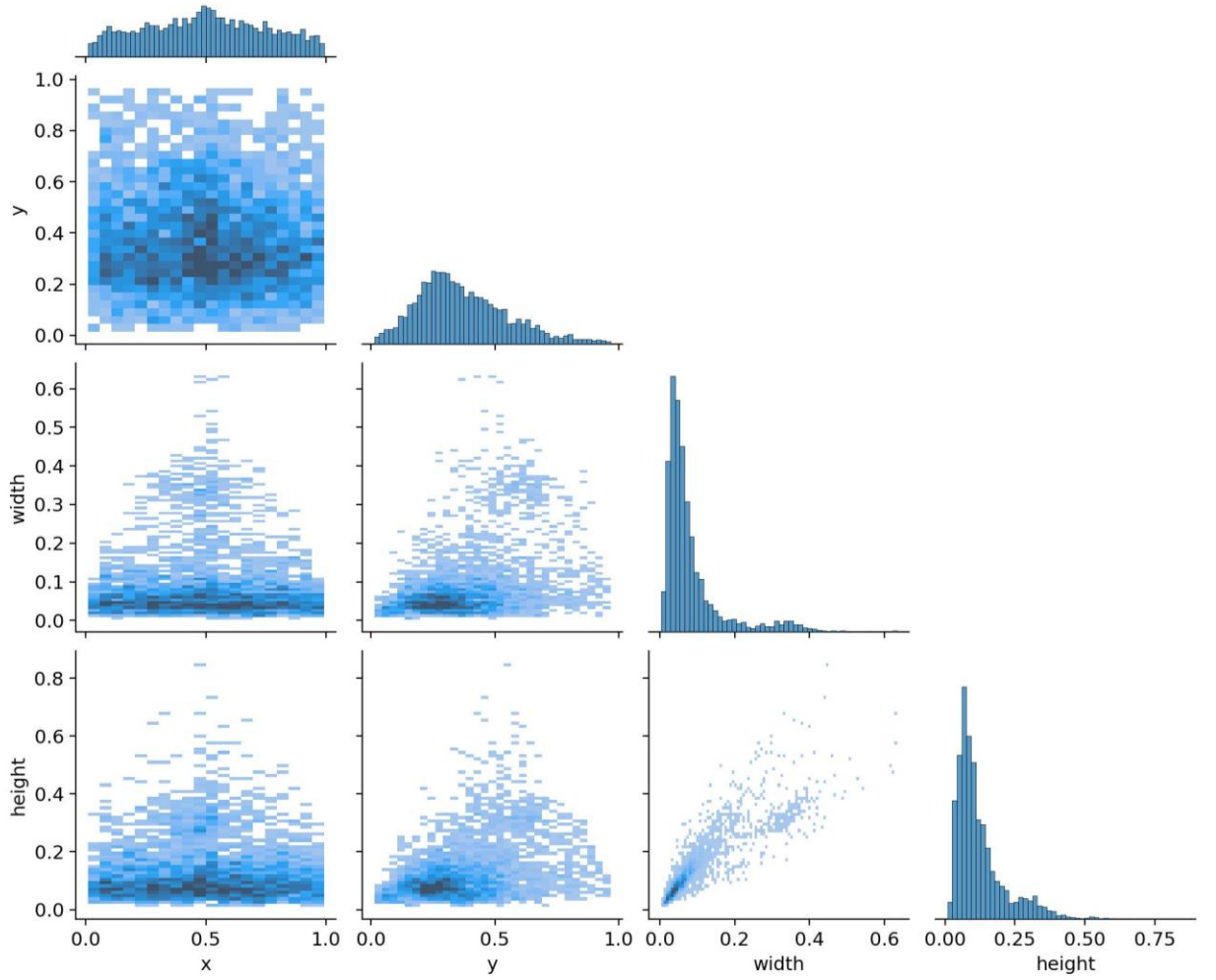
Confusion Matrix:



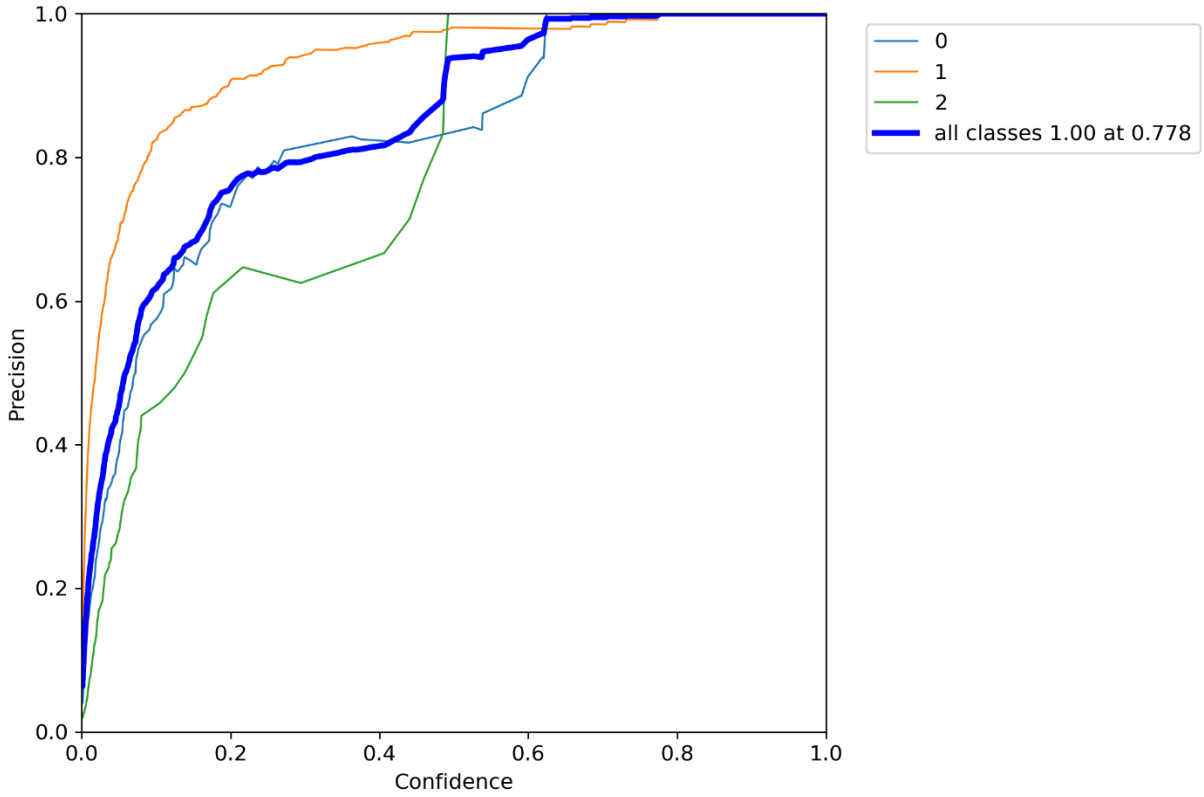
F1 Curve:


1.0
0.8
0.6
0.4
0.2
0.0
F1
0.0
0.2
0.4
0.6
0.8
1.0
Confidence
0
1
2
all classes 0.80 at 0.492

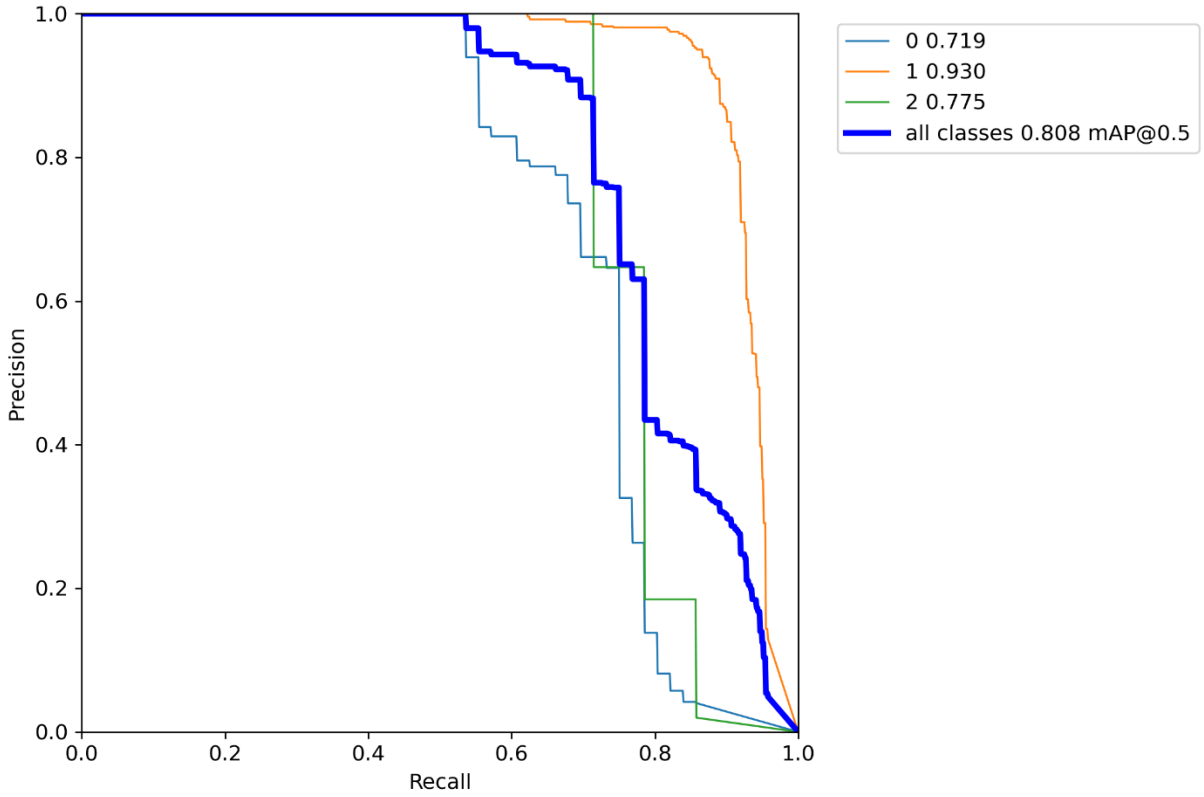
Labels Correlogram:



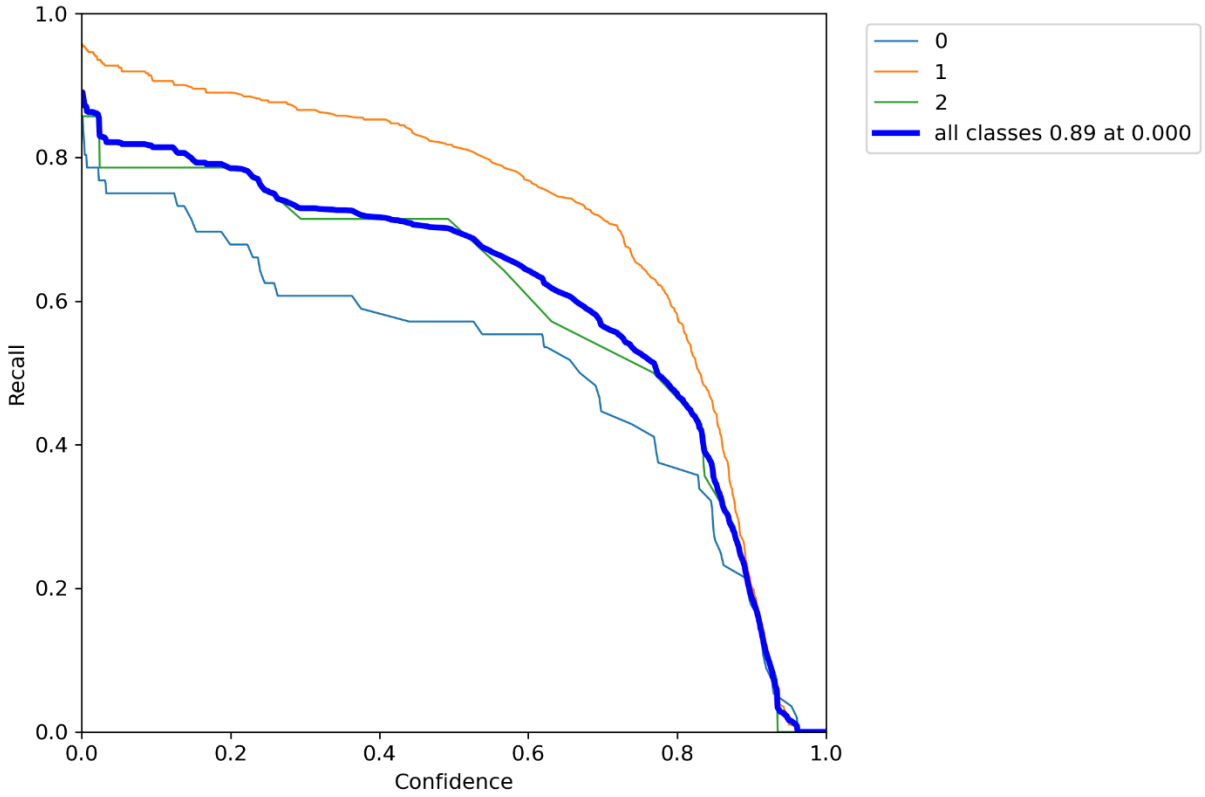
P Curve:


1.0
0.8
0.6
0.4
0.2
0.0
Precision
0.0
0.2
0.4
0.6
0.8
1.0
Confidence
0
1
2
all classes 1.00 at 0.778

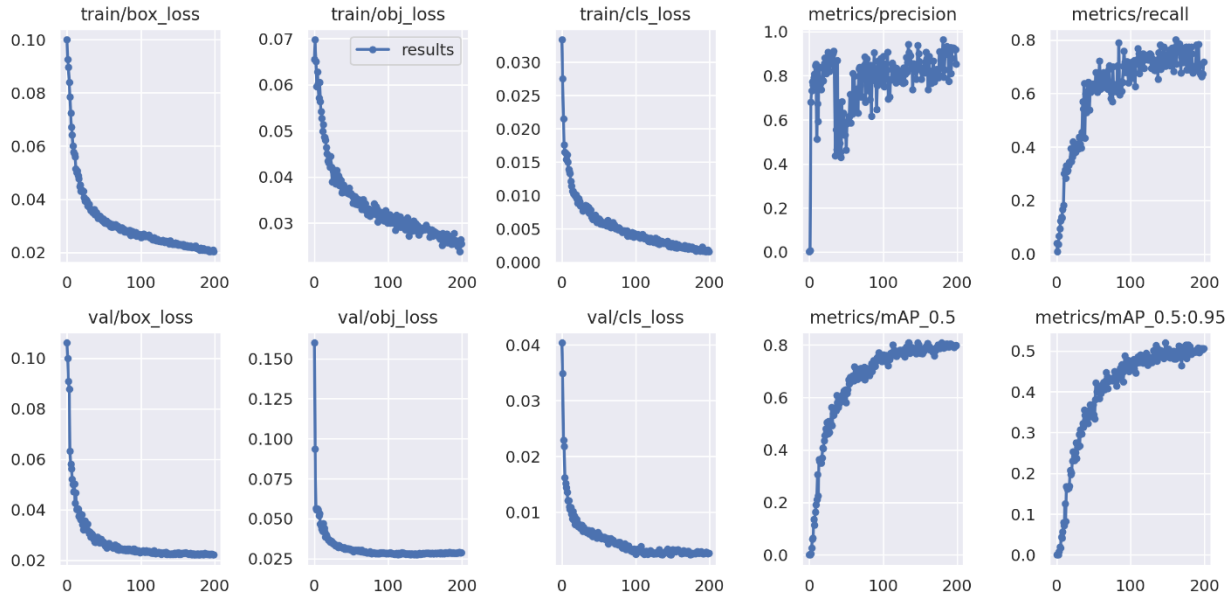
PR-Curve:


0 0.719
1 0.930
2 0.775
all classes 0.808 mAP@0.5
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

R Curve:



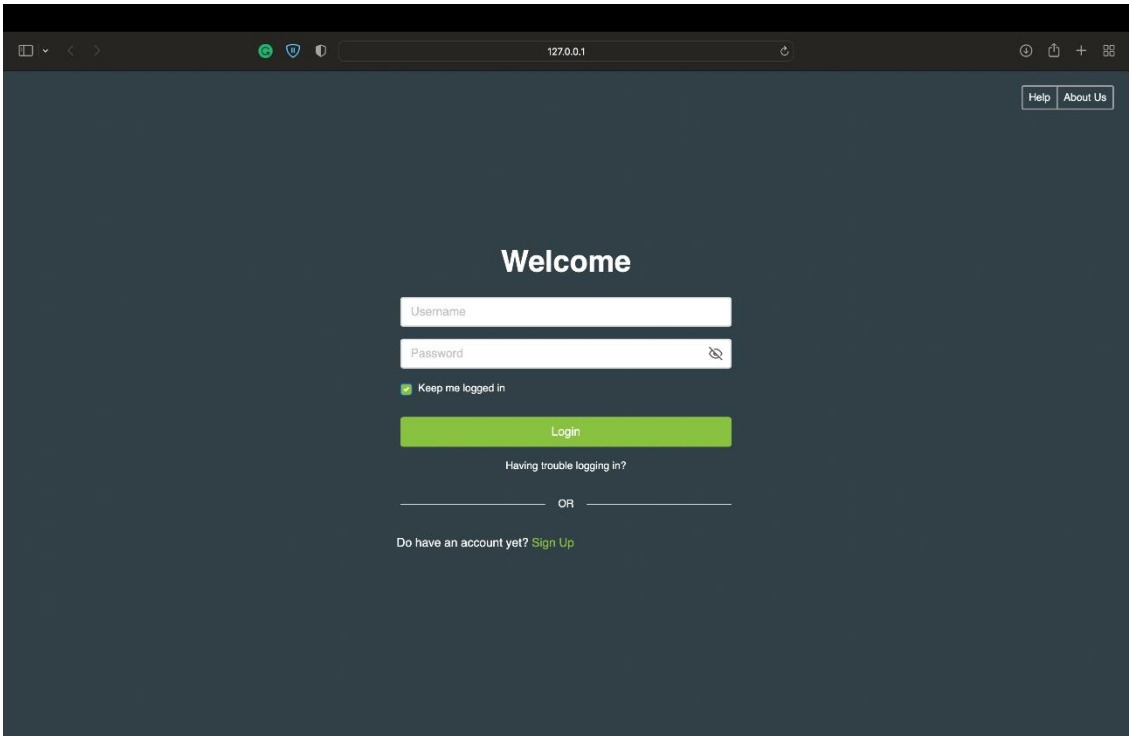
Overall Result:

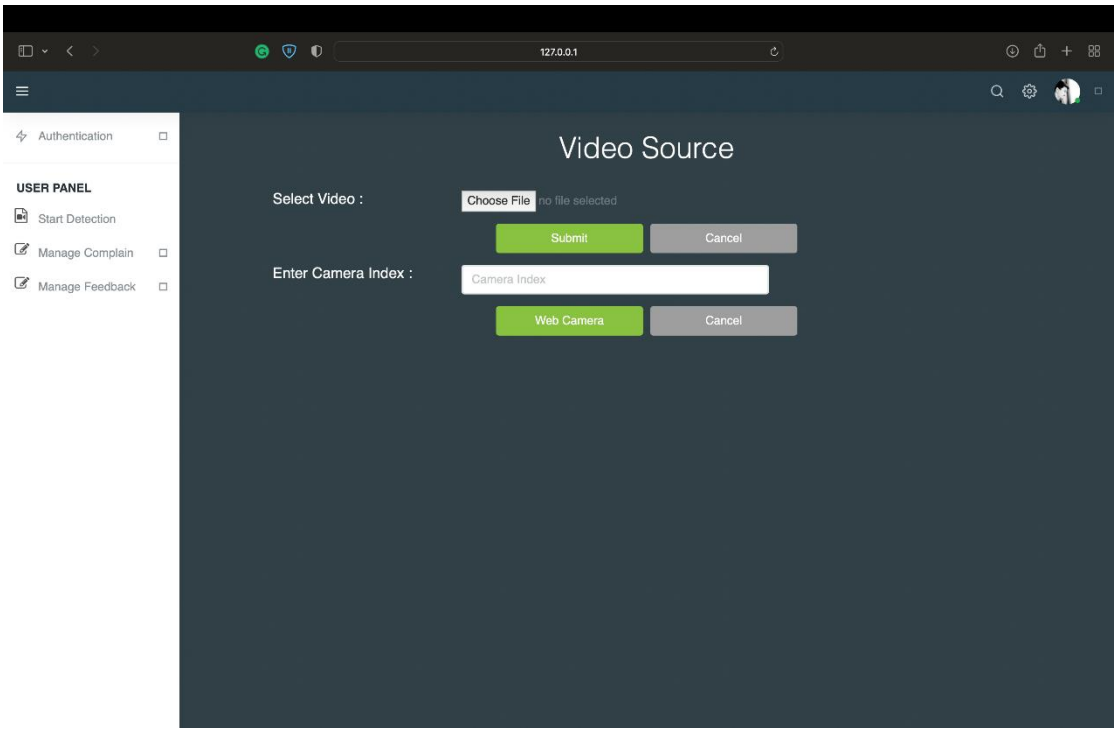


4. Working Sample of Project:

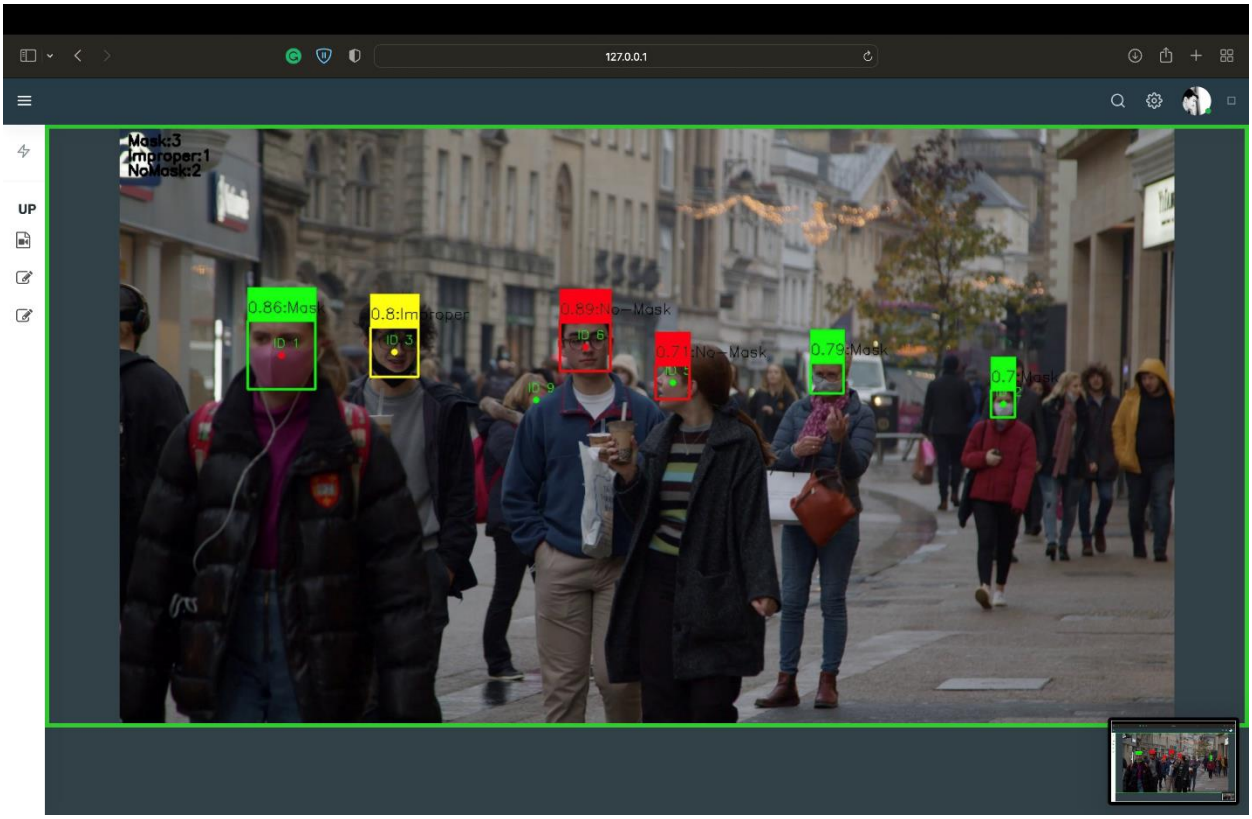
Given Below are the Working Samples of the project where we detected the mask using a Live Stream and by uploading a Video as well

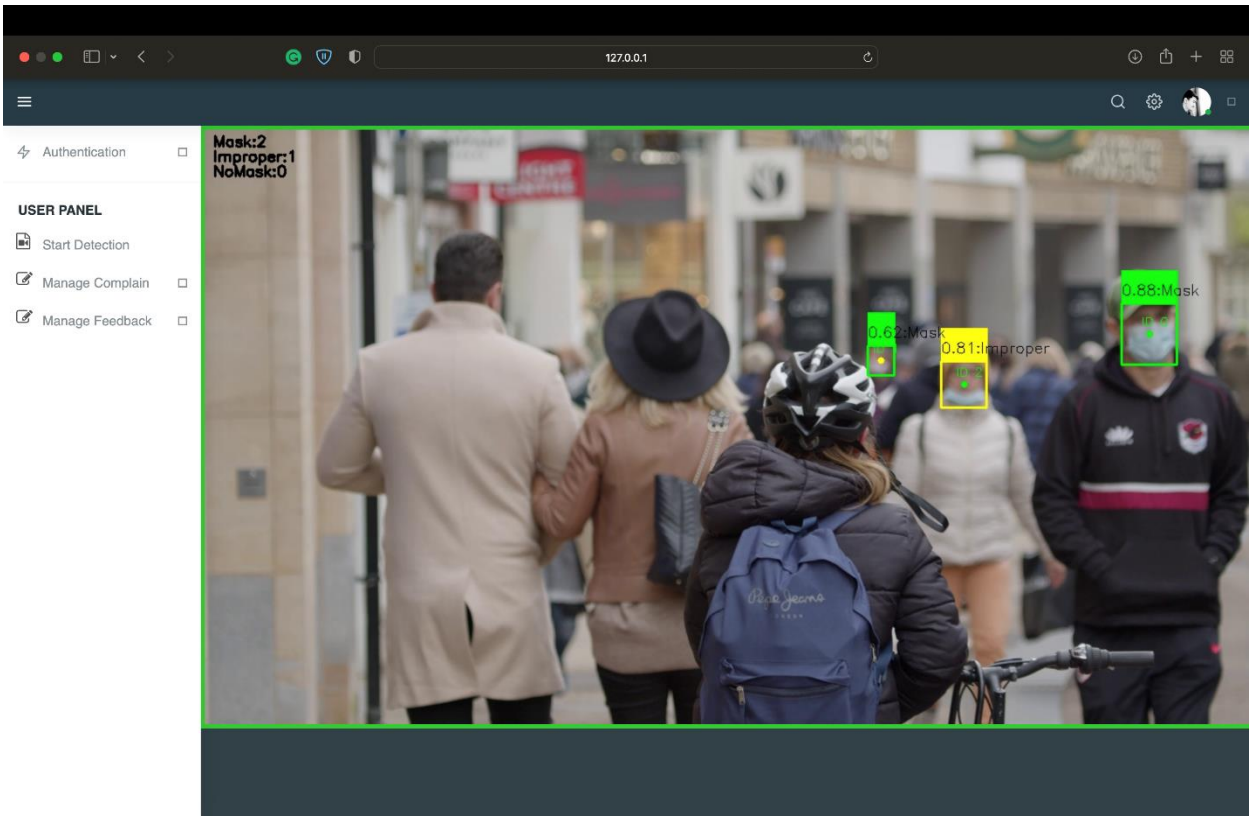
- **Project Interface:**



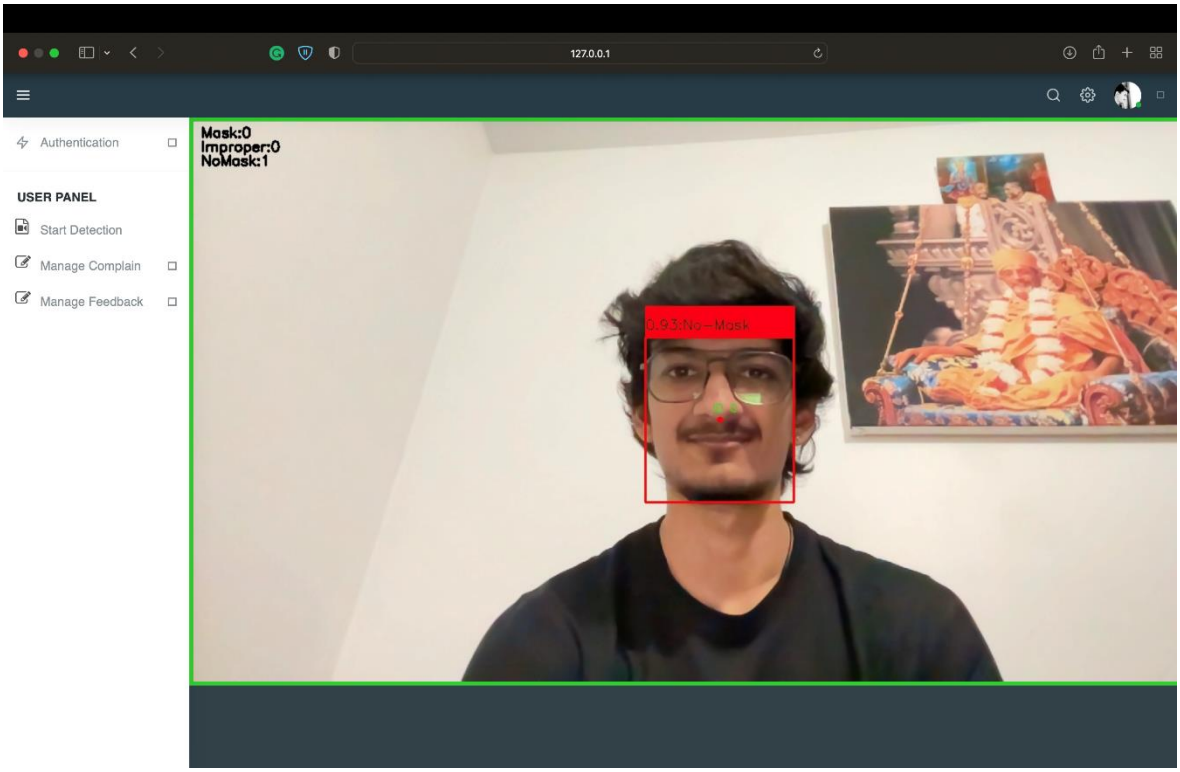


- **Detection of Face Masks in different Using a Video Source:**

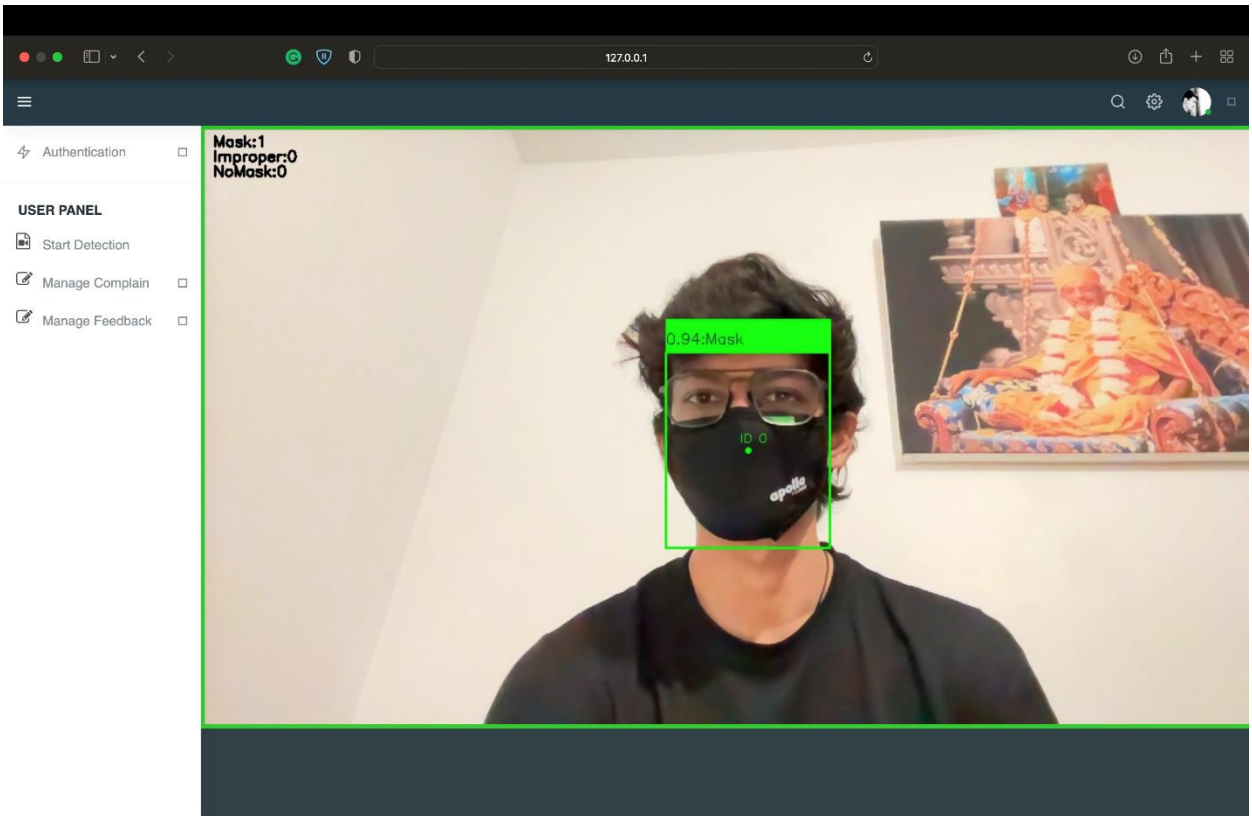




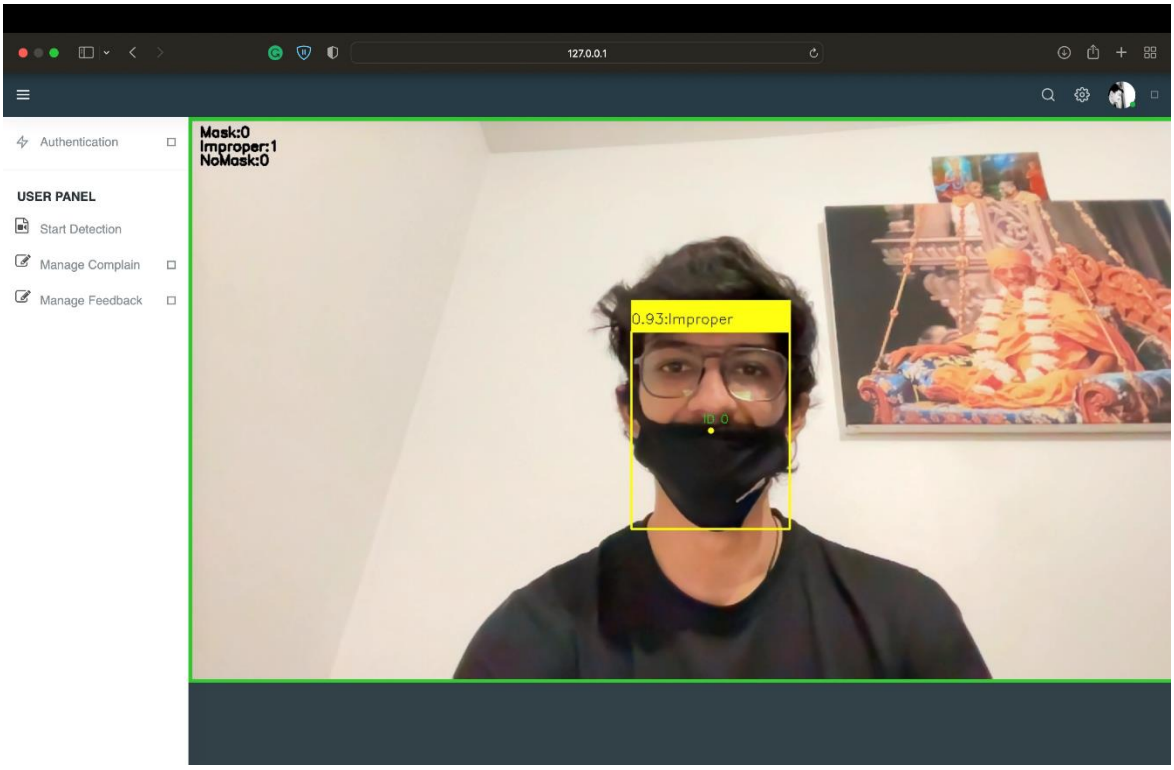
- **Detection of Person Wearing No Face Mask on a Live Stream**



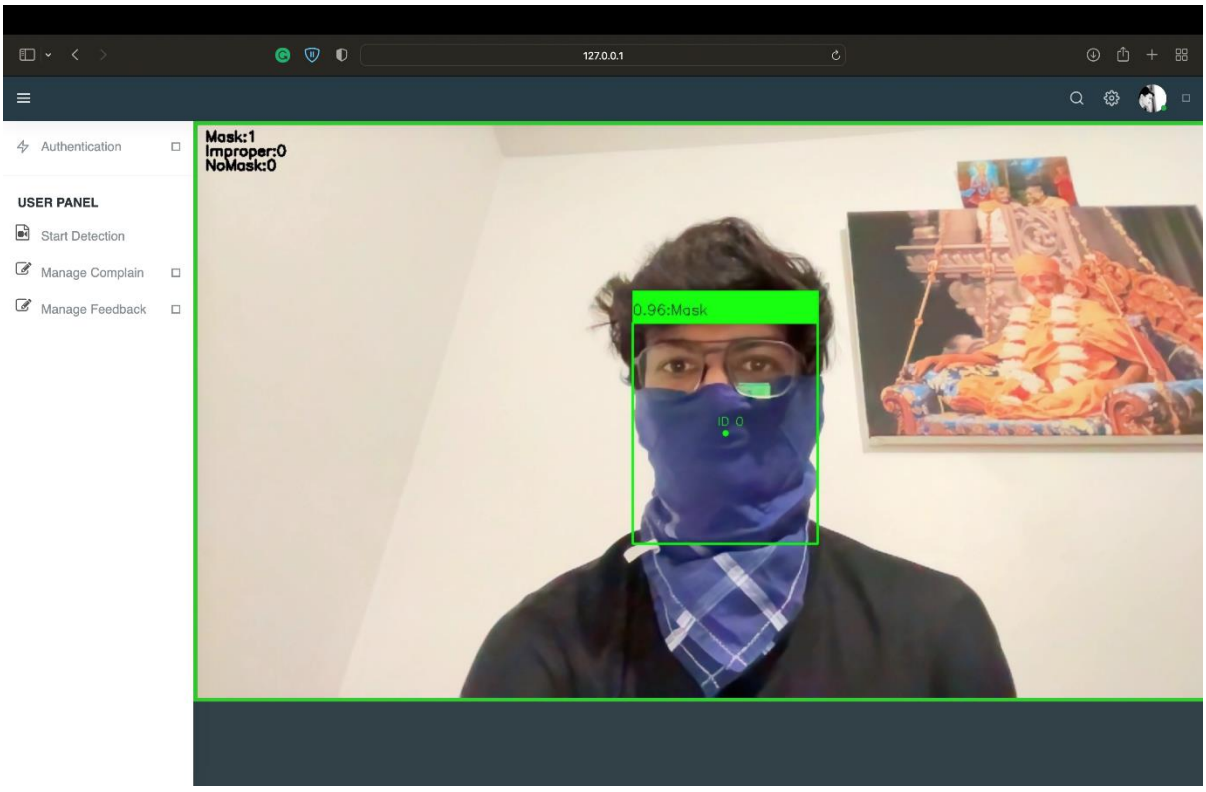
- **Detection of Person Wearing Improper Face Mask**



- **Person Wearing Improper Face Mask:**



- **Detection of Person Using some Cloth in Place of a Mask:**



5. Conclusion and Future Work

The proposed system for real-time face mask detection using YOLOV5 is effective and efficient. The system is able to detect faces with or without masks in real-time with high accuracy. The system can be further improved by increasing the number of training images and by using a more powerful Deep Learning model. Also, the system can be made more robust by using a more sophisticated image pre-processing technique. Altogether, the proposed system is a promising solution for real-time face mask detection. Finally, the system can be deployed on a mobile device for real-time face mask detection in public places. we will use docker to deploy our system on a google cloud platform. In the future, the system can be extended to other applications such as detecting other objects such as cars, people, animals, etc.

References

(2022). Retrieved from GitHub: https://github.com/ultralytics/yolov5

(2022). Retrieved from Google Collab: https://colab.research.google.com/github/pytorch/pytorch.github.io/blob/master/assets/hub/ultralytics_yolov5.ipynb

(2022). Retrieved from Hugging Face: https://huggingface.co/spaces/pytorch/YOLOv5

Liu, Y. L. (2020). Research on the Use of YOLOv5 Object Detection Algorithm in Mask Wearing Recognition. *World Scientific Research Journal*. Retrieved from https://pesquisa.bvsalud.org/global-literature-on-novel-coronavirus-2019-ncov/resource/en/covidwho-994115

Saha, S. (2018). A Comprehensive Guide to Convolutional Neural Networks. *Towards Data Science*. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Yamashita, R. N. (2018). Convolutional neural networks: an overview and application in radiology. Insights Imaging. *Link Springer*. Retrieved from https://link.springer.com/article/10.1007/s13244-018-0639-9