TABLE OF CONTENTS

Chapter -1

INTRODUCTION

1.1 GENERAL INTRODUTION

The rapid worldwide spread of Coronavirus Disease 2019 (COVID-19) has resulted in a global pandemic. Correct facemask wearing is valuable for infectious disease control, but the effectiveness of facemasks has been diminished, mostly due to improper wearing. However, there have not been any published reports on the automatic identification of facemask-wearing conditions. In this study, we develop a new facemask-wearing condition identification using object detection model YOLOv5.

The technology behind the real-time face mask detection system is not new. In Machine Learning, face mask detection is the problem of computer vision. Too often we see computer vision applications of this technology in our daily lives. A common example is a face unlocking in smartphones.

The goal of a face mask detection system is to create an image recognition system that understands how image classification works, and it should work with great accuracy so that our model can be applied in the real time situations. It will work by recognizing the boundaries of the face and predicting whether or not you are wearing a face mask in real-time.

Our project Real Time Face Mask Detection System is a direct application of Object Detection using YOLOv5 where we aim to train the yolov5 to detect whether a person is wearing mask properly, unproperly or not at all.

1.2 PROBLEM STATEMENT

In these difficult times of covid-19, there is a necessary need of maintaining social distancing and wearing a mask. In Spite of its importance nowadays, wearing a mask is avoided by much of our population.

The goal of a face mask detection system is to create an image recognition system that understands how image classification works, and it should work with great accuracy so that our model can be applied in the real time situations. It will work by recognizing the boundaries of the face and predicting whether or not you are wearing a face mask in real-time and if weared it's proper or not.

Respiratory droplets are generated when an infected person coughs or sneezes. Any person in close contact (within 1 m) with someone who has respiratory symptoms (coughing, sneezing) is at risk of being exposed to potentially infective respiratory droplets.

Droplets may also land on surfaces where the virus could remain viable; thus, the immediate environment of an infected individual can serve as a source of transmission (contact transmission). Wearing a medical mask is one of the prevention measures that can limit the spread of certain respiratory viral diseases, including COVID-19.

1.3 ABSTRACT

After the breakout of the worldwide pandemic COVID-19, there arises a severe need of protection mechanisms, face mask being the primary one. The basic aim of the project is to detect the presence of a face mask on human faces on live streaming video as well as on images. We have used object detection model and trained it to detect masks in crowded places.

1.4 MOTIVATION

Although many people are already convinced of the interest for wearing face protection mask such as suggested by the World Health Organization and scientific studies, one can observe that many individuals do not correctly wear their masks (see various mask wearing configurations in Fig. 1). These observations have led nurses and other citizens to initiate prevention campaigns related to the public health education in wearing the mask. Precisely, these campaigns consist of sensitizing people about the correct and incorrect manner to wear face protection masks by disseminating prevention posters and drawings. In our case, we propose to support these public health campaigns by designing an image - based analysis method and an associated digital tool that are dedicated to the verification of the correct mask wearing.

Figure 1: Various configurations related to the mask wearing. (a) no mask; (b) unprotected nose; (c) unprotected chin (d) unprotected; (e) unprotected chin and mouth; (f) protected nose, mouth and chin

1.5 AIM AND OBJECTIVES

Aim –

Aim of our project is to detect people without masks in a crowded place

Objectives –

Coronavirus disease 2019 (COVID-19) is an emerging respiratory infectious disease caused by Severe Acute Respiratory Syndrome coronavirus 2 (SARS-CoV2). At present, COVID-19 has quickly spread to the majority of countries worldwide, affecting more than 14.9 million individuals, and has caused 618,017 deaths, according to the report from the World Health Organization (WHO) on 23 July 2020 (https://www.who.int/emergencies/diseases/novel-coronavirus-2019). To avoid global tragedy, a practical and straightforward approach to preventing the spread of the virus is wearing mask outside or public places. So mask detection at the crowded public plaes is very much important in the current scenerio.

Figure 2

Chapter- 2
LITERATURE REVIEW

In this chapter, a brief overview of studies made on face detection and recognition and object detection model used in this project (YOLOv5). This will give a general idea of the history of systems and approaches that have been used so far.

2.1 YOLO v5 :

You Only Look Once (YOLO) is a state-of-the-art, real-time object detection system. On a Pascal Titan X it processes images at 30 FPS and has a mAP of 57.9% on COCO test-dev. YOLOv3 is extremely fast and accurate. In mAP measured at .5 IOU YOLOv3 is on par with Focal Loss but about 4x faster. Moreover, you can easily tradeoff between speed and accuracy simply by changing the size of the model, no retraining required!

Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

We use a totally different approach. We apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

https://github.com/harpreet-22/yolov5

Figure 3

2.2 Matplotlib (3.2.2) :

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Installation :

Windows, Linux and macOS distributions have matplotlib and most of its dependencies as wheel packages. Run the following command to install matplotlib package :

python -mpip install -U matplotlib

Importing matplotlib :

from matplotlib import pyplot as plt

or

import matplotlib.pyplot as plt

Figure 4

2.3 OpenCV (4.1.2)

OpenCV was started at Intel in 1999 by **Gary Bradsky**, and the first release came out in 2000. **Vadim Pisarevsky** joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day.
OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development.

OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.
Python is a general purpose programming language started by **Guido van Rossum** that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.
Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.
OpenCV-Python makes use of **Numpy**, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

Figure 5

2.4 PyYAML (5.3.1)

YAML Ain't Markup Language (YAML) is a data serialization language for most programming languages. Let's understand in detail.

YAML a strict superset of JSON, so anything written in JSON can be parsed to YAML. It's mostly used for configuration files in projects and it's super easy to understand and read the code.

The file extension for YAML files is .yaml or .yml

In this tutorial, we are going to learn about different data types that are present in YAML and work with YAML in Python. By the end of this tutorial, you will be able to understand the YAML and its syntax.

The YAML follows indentation syntax similar to Python. But, it doesn't allow tab (remember it while writing YAML files) for indentation.

2.5 PyTorch (1.7.1)

PyTorch is defined as an open source machine learning library for Python. It is used for applications such as natural language processing. It is initially developed by Facebook artificial-intelligence research group, and Uber's Pyro software for probabilistic programming which is built on it.

Originally, PyTorch was developed by Hugh Perkins as a Python wrapper for the LusJIT based on Torch framework. There are two PyTorch variants.

PyTorch redesigns and implements Torch in Python while sharing the same core C libraries for the backend code. PyTorch developers tuned this back-end code to run Python efficiently. They also kept the GPU based hardware acceleration as well as the extensibility features that made Lua-based Torch.

Figure 6

Features

The major features of PyTorch are mentioned below −

Easy Interface − PyTorch offers easy to use API; hence it is considered to be very simple to operate and runs on Python. The code execution in this framework is quite easy.

Python usage − This library is considered to be Pythonic which smoothly integrates with the Python data science stack. Thus, it can leverage all the services and functionalities offered by the Python environment.

Computational graphs − PyTorch provides an excellent platform which offers dynamic computational graphs. Thus a user can change them during runtime. This is highly useful when a developer has no idea of how much memory is required for creating a neural network model.

PyTorch is known for having three levels of abstraction as given below −

- Tensor − Imperative n-dimensional array which runs on GPU.
- Variable − Node in computational graph. This stores data and gradient.
- Module − Neural network layer which will store state or learnable weights.

Advantages of PyTorch

The following are the advantages of PyTorch −

- It is easy to debug and understand the code.
- It includes many layers as Torch.
- It includes lot of loss functions.
- It can be considered as NumPy extension to GPUs.
- It allows building networks whose structure is dependent on computation itself.

2.6 Pandas (1.2.4)

i. Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.

In 2008, developer Wes McKinney started developing pandas when in need of high performance, flexible tool for analysis of data.

Prior to Pandas, Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze.

Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Key Features of Pandas :-

- Fast and efficient DataFrame object with default and customized indexing.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of date sets.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.
- Time Series functionality.

Figure 7

2.7. Seaborn (0.11.0)

In the world of Analytics, the best way to get insights is by visualizing the data. Data can be visualized by representing it as plots which is easy to understand, explore and grasp. Such data helps in drawing the attention of key elements.

To analyse a set of data using Python, we make use of Matplotlib, a widely implemented 2D plotting library. Likewise, Seaborn is a visualization library in Python. It is built on top of Matplotlib.

Seaborn Vs Matplotlib

It is summarized that if Matplotlib "tries to make easy things easy and hard things possible", Seaborn tries to make a well-defined set of hard things easy too."

Seaborn helps resolve the two major problems faced by Matplotlib; the problems are –

- Default Matplotlib parameters
- Working with data frames

As Seaborn compliments and extends Matplotlib, the learning curve is quite gradual. If you know Matplotlib, you are already half way through Seaborn.

Important Features of Seaborn

Seaborn is built on top of Python's core visualization library Matplotlib. It is meant to serve as a complement, and not a replacement. However, Seaborn comes with some very important features. Let us see a few of them here. The features help in −

- Built in themes for styling matplotlib graphics
- Visualizing univariate and bivariate data
- Fitting in and visualizing linear regression models
- Plotting statistical time series data
- Seaborn works well with NumPy and Pandas data structures
- It comes with built in themes for styling Matplotlib graphics

Figure 8

Chapter-3

DESCRIPTION AND METHODOLOGY

The first step in implementing our project is accumulation of data- in our case, images of people wearing masks, not wearing masks and wearing masks unproperly. Then we convert our data into a specific format to meet our requirements and finally, feed the data into a model which trains on the input data. Once the data is trained, we test the model for test data, which is around 80% of total data. We keep training the model until we obtain a good accuracy to achieve system objectives and finally test the trained weights on the test images. Now the model is ready to be tested on the real world images and videos. Step by step procedure: -

3.1 Image Directory Structure :

The main directory consist of two folders namely,

i. **test** – This folder contains the test images for each of the classes. Each class contains about of 85 images for test. The test images are used for checking the accuracy of system.

ii. **train** – This directory contains the training data for each student. Each student has at least 2,000 images. However 2,000 images are not enough to get an accuracy of 98% around but it can provide us the accuracy of about 85 to 90%. The deep learning models are data hungry, the more you feed them the more accuracy you'll get.

Figure 9

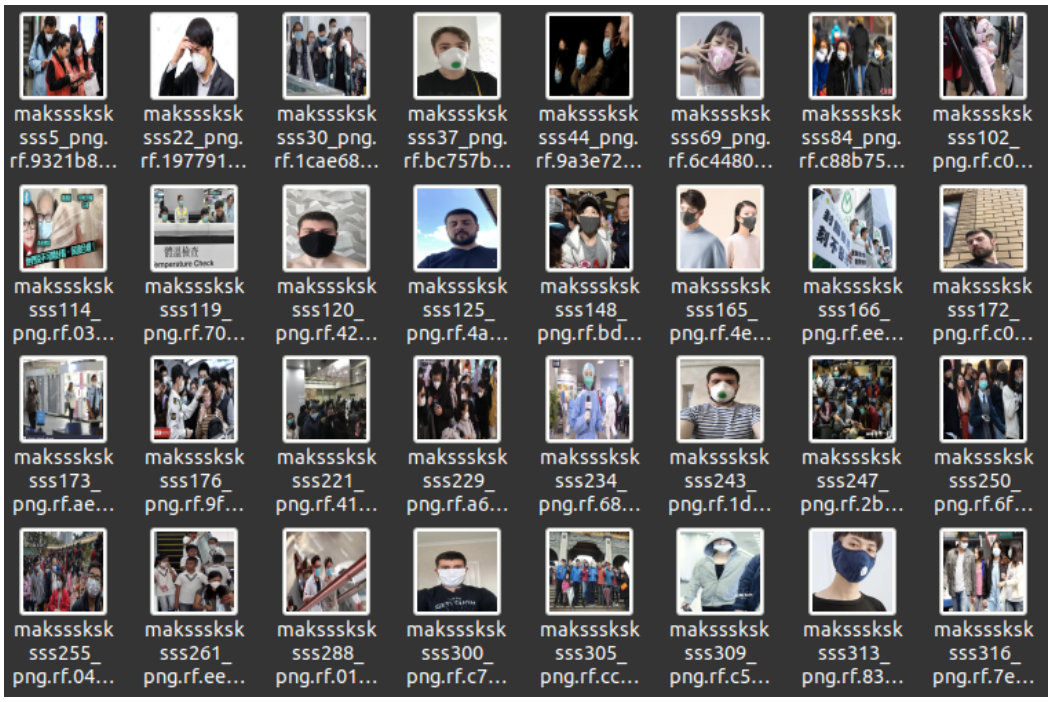

Figure 10. Test Images

3.2 Clone the YOLO v5 :

Now we clone the YOLOv5 model from github (https://github.com/harpreet-22/yolov5). Then we install all the required libraries of python in google colab.

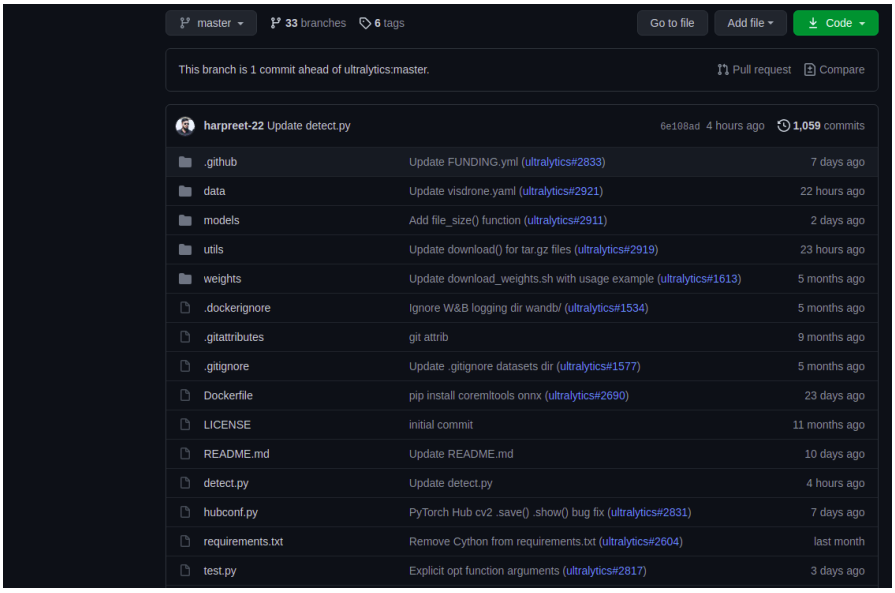

FIgure 11. YOLO v5 Repository

```
%%writetemplate /content/yolov5/models/custom_yolov5s.yaml

# parameters
nc: {num_classes}  # number of classes  # CHANGED HERE
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]],  # 13
```

Figure 12 Customising YOLO v5

3.3 Training the Weights :

So now we train weights after customising the YOLO v5 model :

Here, we are able to pass a number of arguments:

- **img:** define input image size
- **batch:** determine batch size
- **epochs:** define the number of training epochs.
- **data:** set the path to our yaml file
- **cfg:** specify our model configuration
- **weights:** specify a custom path to weights.
- **cache:** cache images for faster training
- **nosave:** only save the final checkpoint

```
Analyzing anchors... anchors/target = 5.49, Best Possible Recall (BPR) = 0.9986
Image sizes 416 train, 416 test
Using 2 dataloader workers
Logging results to runs/exp0_yolov5s_results
Starting training for 100 epochs...

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
      0/99        8G    0.1042    0.1089   0.03403    0.2471       429       416: 100% 24/24 [00:20<00:00,  1.15it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:07<00:00,  3.53s/it]
                 all         127         757           0           0     0.00012    1.68e-05

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
      1/99     6.05G   0.09975    0.1119   0.02604    0.2377       454       416: 100% 24/24 [00:12<00:00,  1.90it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:01<00:00,  1.11it/s]
                 all         127         757           0           0    0.000112    1.62e-05

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
      2/99     6.05G   0.09877    0.1152    0.0228    0.2367       495       416: 100% 24/24 [00:12<00:00,  1.89it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:02<00:00,  1.27s/it]
                 all         127         757           0           0    0.000118    1.66e-05

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
      3/99     6.05G   0.09837    0.1144   0.02202    0.2347       413       416: 100% 24/24 [00:12<00:00,  1.94it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:02<00:00,  1.33s/it]
                 all         127         757           0           0    0.000288    4.27e-05
```

Figure 13 Training the Weights

```
     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     99/99     6.05G   0.04351   0.07009  0.005646    0.1193       383       416: 100% 24/24 [00:12<00:00,  1.92it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:03<00:00,  1.67s/it]
                 all         127         757       0.303       0.534       0.472       0.187
Optimizer stripped from runs/exp0_yolov5s_results/weights/last.pt, 14.8MB
Optimizer stripped from runs/exp0_yolov5s_results/weights/best.pt, 14.8MB
100 epochs completed in 0.400 hours.

CPU times: user 5.33 s, sys: 885 ms, total: 6.21 s
Wall time: 24min 31s
```

Figure 14 100 Epochs Completed

3.4 Plotting the training results on the Tensorboard :

So now we have trained weights, it's time to plot the perfomance of weights.

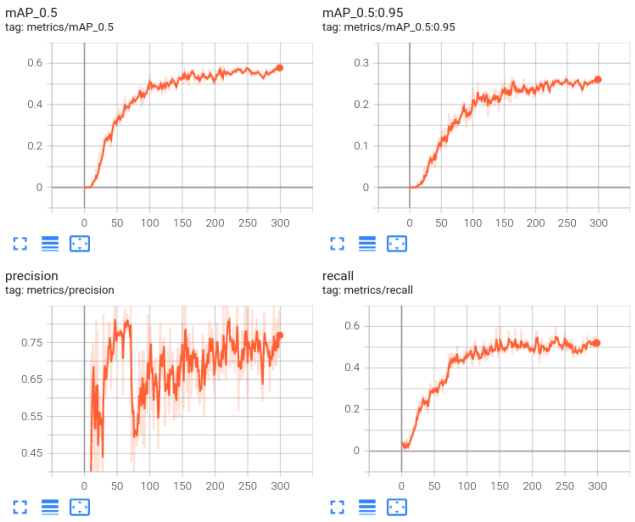

Figure 15 Plots

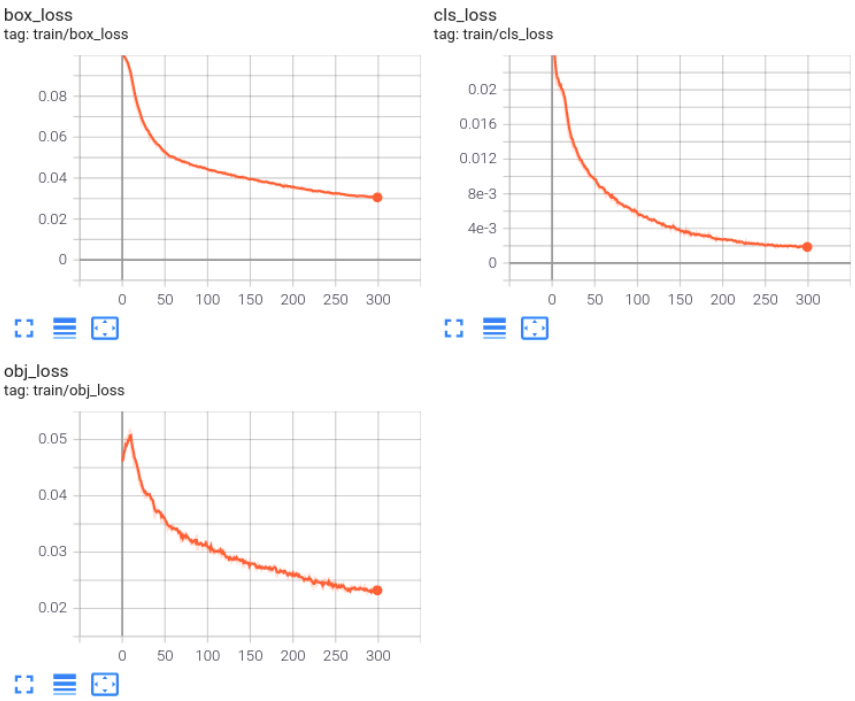

Figure 16 Plots

Now mount your google drive in the runtime enviroment of googe colab and then save weights to your google drive for future inference.

```
[ ]  from google.colab import drive
     drive.mount('/content/gdrive')

     Mounted at /content/gdrive

[ ]  %cp -r runs/train /content/gdrive/MyDrive
```

Figure 17 Saving the weights

3.5 Testing on the input from webcam :

```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode

def take_photo(filename='/content/photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);
```

Figure 18 Setting up webcam

```
      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename
```

```
from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to {}'.format(filename))

  # Show the image which was just taken.
  display(Image(filename))
except Exception as err:
  # Errors will be thrown if the user does not have a webcam or if they do not
  # grant the page permission to access it.
  print(str(err))
!python detect.py --weights /content/gdrive/MyDrive/train/yolov5s_results/weights/best.pt --img 416 --conf 0.5 --source /content/photo.jpg
```

Figure 19 Testing on the Webcam

Chapter-4

HARDWARE AND SOFTWARE REQUIREMENTS

I. HARDWARE REQUIREMENTS:

The system requires good quality images for prediction thus a quality web cam is required. The system must have upgraded RAM and ROM with at least 2GB of RAM and 250GB of ROM. For fast and quality performance the system requires GPU for high order mathematical computation in neural network with the image.

II. SOFTWARE REQUIREMENTS:

i. Any Operating System.

ii. Google Colab

iii. Scientific Python modules (Numpy, Keras, Scikit, Tensorflow)

Chapter-5

RESULTS

After implementing the project, we tested it on few test samples and received generally positive and accurate results with around 90% accuracy. Our trained Neural Network also works as per the system requirements. Finally, we are able to detect whether the person is wearing the mask or not.

5.1 Sample test image – 01

Fig-20: Test image 01

This is a group photo of 4 people. Now we are going to input this image in our model. Let us see the results when we ran the scripts to detect faces:

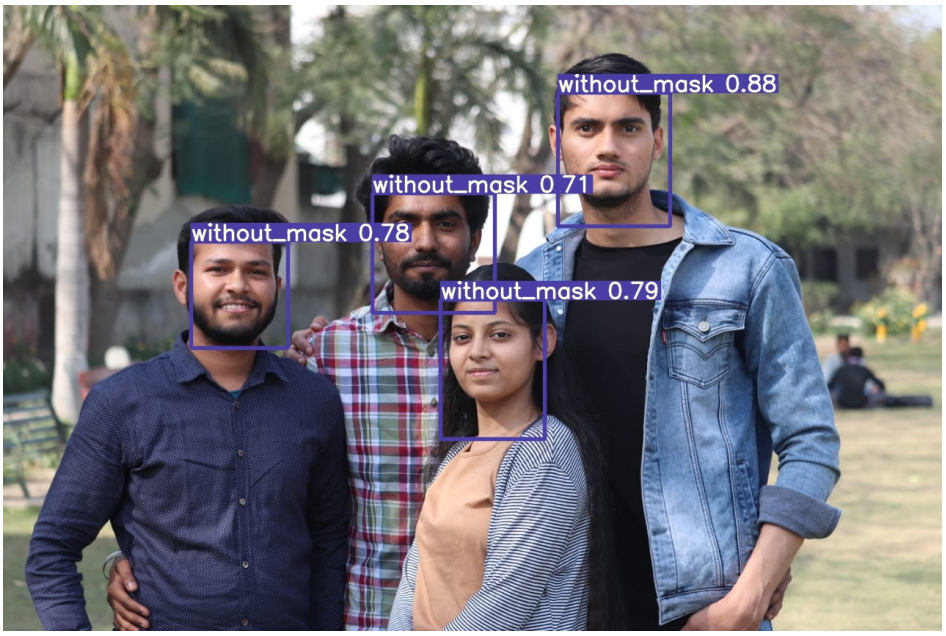

Fig 21: Detection in test image 01

5.2 Sample test image – 02

Fig 22: Test image 02

Fig-23: Detection in test image 02

5.3 Sample test image - 03

Figure 24 Test image 03

Chapter-6
CONCLUSION

This report describes how to detect masks on multiple faces using object detection model YOLOv5. The above method of detecting masks can be used at any crowded public place :

Airports

The Face Mask Detection System could be used at airports to detect travelers without masks. Face data of travelers can be captured in the system at the entrance. If a traveler is found to be without a face mask, their picture is sent to the airport authorities so that they could take quick action.

Hospitals

Using Face Mask Detector System, Hospitals can monitor if quarantined people required to wear a mask are doing so or not. The same holds good for monitoring staff on duty too.

Offices & Working Spaces

The Face Mask Detection System can be used at office premises to ascertain if employees are maintaining safety standards at work. It monitors employees without masks and sends them a reminder to wear a mask.

Government

To limit the spread of coronavirus, the police could deploy the face mask detector on its fleet of surveillance cameras to enforce the compulsory wearing of face masks in public places.

The current study used OpenCV, Pytorch and CNN to detect whether people were wearing face masks or not. The models were tested with images and real-time video streams. Even though the accuracy of the model is around 80%, the optimization of the model is a continuous process

REFERENCES

We believe that the following links/tutorials/YouTube videos played a major role in helping us understand the concepts and implement what we learnt from them.

MACHINE LEARNING

i. http://neuralnetworksanddeeplearning.com/chap1.html

ii. https://towardsdatascience.com/understanding-neural-networks-19020b758230

COMPUTER VISION

i. https://www.youtube.com/watch?v=dWeWCQmewLc&list=PLiHa1s-EL3vjr0Z02ihr6Lcu4Q0rnRvjm

FACE DETECTION AND RECOGNITION AND NORMALIZATION

https://machinelearningmastery.com/how-to-develop-a-face-recognition-system-using-facenet-in-keras-and-an-svm-classifier/

YOLO v5 :

https://youtu.be/MdF6x6ZmLAY

https://github.com/harpreet-22/yolov5