Face Mask Detection Using Transfer Learning And Computer Vision

Project Report

Submitted in the partial fulfillment for the award of the degree of:

BACHELOR OF ENGINEERING

IN

CSE-AIML

Submitted by:

Aashish Kalra (20BCS6852) **Harshil Jain (20BCS6877)**
Shubharthak Sangharasha(20BCS6872)

Under the Supervision of:

Mr. Amit Garg

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
APEX INSTITUE OF TECHNOLOGY
CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413, PUNJAB

May & 2022

DECLARATION

I, **'Aashish Kalra'**, student of **'Bachelor of Engineering in CSE-AIML, Session: 2020-2024**, Department of Computer Science and Engineering, Apex Institute of Technology, Chandigarh University, Punjab, hereby declare that the work presented in this Project Work entitled '**Face Mask Detection Using Transfer Learning and Computer Vision'** is the outcome of our own bona fide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics. It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Aashish Kalra

Date: 15/05/2022

20BCS6852

ABSRACT

The novel Coronavirus had brought a new normal life in which the social distance and wearing of face masks plays a vital role in controlling the spread of virus. But most of the people are not wearing face masks in public places which increases the spread of viruses. Hence to avoid such situations we have to scrutinise and make people aware of wearing face masks. Humans cannot be involved in this process, due to the chance of getting affected by corona. Hence here comes the need for artificial intelligence (AI), which is the main theme of our project. Our project involves the identification of persons wearing face masks and not wearing face masks in public places by means of using image processing or OpenCV and other AI techniques for giving alert through our UI to that person that they need to wear a mask. The object detection algorithms are used for identification of persons with and without wearing face masks which also gives the count of persons wearing mask and not wearing face mask. Based on the count of persons wearing and not wearing face masks the status is obtained. We can also keep the record of who is entering without a mask at which time so we can use that record for future purposes. We can also use Raspberry pi and Arduino for mechanical Work. We can use this model in offices where our model can keep the record of person who is not wearing mask and can send the mail or contact them that they need to wear the mask.

Global pandemic COVID-19 circumstances emerged in an epidemic of dangerous disease in all over the world. Wearing a face mask will help prevent the spread of infection and prevent the individual from contracting any airborne infectious germs. Using Face Mask Detection System, one can monitor if the people are wearing masks or not.

According to this motivation we demand mask detection as a unique and public health service system during the global pandemic COVID-19 epidemic. The model is trained by face mask image and non-face mask image.

ACKNOWLEDGEMENT

An endeavor over a long period can be successful with the advice and support of many well-wishers. We take this opportunity to express our gratitude and appreciation to all of them.
We sincerely express our whole hearted thanks to our mentor Dr. Amit Garg, Chandigarh University, for his constant encouragement and moral support during the course of this project.
We sincerely thank our HOD Mr. Aman, for his valuable approval and guidance throughout the project. We wish to express our sincere thanks to the program leader Miss Shikha and all staff members, Department of Computer Software Engineering for their valuable help and guidance rendered to us throughout the project. Above all we are grateful to all our friends for their cooperation and their exhilarating company.
We express our warm and sincere thanks to everyone for the encouragement, untiring guidance and the confidence they had shown in us. We are immensely indebted for valuable guidance throughout our project. We also thank all the staff members of CSE department for their valuable advices.

LIST OF FIGURES

LIST OF TABLES \ SYMBOLS

We haven't used any tables in this project report.

But for the table of symbols used are:

ABBREVIATIONS

The following abbreviations are used in this Project Report:

- AP — Average Precision
- CNN — Convolutional Neural Network
- COVID-19 — Coronavirus Disease 2019
- FPN — Feature Pyramid Network
- IoU — Intersection over Union
- mAP — mean Average Precision
- PWMFD — Properly-Wearing-Masked Face Detection
- SE — Squeeze and Excitation

Table of Contents

1. INTRODUCTION

Definitions:

1. Transfer Learning:

Transfer Learning is a machine learning method where the application of knowledge obtained from a model used in one task, can be reused as a foundation point for another task.

Machine learning algorithms use historical data as their input to make predictions and produce new output values. They are typically designed to conduct isolated tasks. A source task is a task from which knowledge is transferred to a target task. A target task is when improved learning occurs due to the transfer of knowledge from a source task.

During transfer learning, the knowledge leveraged and rapid progress from a source task is used to improve the learning and development to a new target task. The application of knowledge is using the source task's attributes and characteristics, which will be applied and mapped onto the target task.

However, if the transfer method results in a decrease in the performance of the new target task, it is called a negative transfer. One of the major challenges when working with transfer learning methods is being able to provide and ensure the positive transfer between related tasks, whilst avoiding the negative transfer between less related tasks.

The What, When and How of Transfer Learning

1. What do we transfer? To understand which parts of the learned knowledge to transfer, we need to figure out which portions of knowledge best reflect both the source and target. Overall, improving the performance and accuracy of the target task.
2. When do we transfer? Understanding when to transfer is important, as we don't want to be transferring knowledge which could, in turn, make matters worse, leading to negative transfer.

Our goal is to improve the performance of the target task, not make it worse.

3. How do we transfer? Now we have a better idea of what we want to transfer and when we can then move on to working with different techniques to transfer the knowledge efficiently. We will speak more about this later on in the article.

Before we dive into the methodology behind transfer learning, it is good to know the different forms of transfer learning. We will go through three different types of transfer learning scenarios, based on relationships between the source task and target task. Below is an overview of the different types of transfer learning:

Different Types of Transfer Learning

Inductive Transfer Learning: In this type of transfer learning, the source and target task are the same, however, they are still different from one another. The model will use inductive biases from the source task to help improve the performance of the target task. The source task may or may not contain labelled data, further leading onto the model using multitask learning and self-taught learning.

Unsupervised Transfer Learning: I assume you know what unsupervised learning is, however, if you don't, it is when an algorithm is subjected to being able to identify patterns in datasets that have not been labelled or classified. In this case, the source and target are similar, however, the task is different, where both data are unlabelled in both source and target. Techniques such as dimensionality reduction and clustering are well known in unsupervised learning.

Transductive Transfer Learning: In this last type of transfer learning, the source and target tasks share similarities, however, the domains are different. The source domain contains a lot of labelled data, whereas there is an absence of labelled data in the target domain, further leading onto the model using domain adaptation.

Transfer Learning vs. Fine-tuning:

Fine-tuning is an optional step in transfer learning and is primarily

incorporated to improve the performance of the model. The difference between Transfer learning and Fine-tuning is all in the name.
Transfer learning is built on adopting features learned from one task and "transferring" the leveraged knowledge onto a new task. Transfer learning is usually used on tasks where the dataset is too small, to train a full-scale model from scratch. Fine-tuning is built on making "fine" adjustments to a process in order to obtain the desired output to further improve performance. The parameters of a trained model during fine-tuning, are adjusted and tailored precisely and specifically, whilst trying to validate the model to achieve the desired outputs.

Why Use Transfer Learning?

Reasons to use transfer learning:

Not needing a lot of data - Gaining access to data is always a hindrance due to its lack of availability. Working with insufficient amounts of data can result in low performance. This is where transfer learning shines as the machine learning model can be built with a small training dataset, due to it being pre-trained.
Saving training time - Machine learning models are difficult to train and can take up a lot of time, leading to inefficiency. It requires a long period of time to train a deep neural network from scratch on a complex task, so using a pre-trained model saves time on building a new one.

Transfer Learning Pros

Better base: Using a pre-trained model in transfer learning offers you a better foundation and starting point, allowing you to perform some tasks without even training.
Higher learning rate: Due to the model already having been trained on a similar task beforehand, the model has a higher learning rate.
Higher accuracy rate: With a better base and higher learning rate, the model works at a higher performance, producing more accuracy outputs.

When Does Transfer Learning *Not* Work?

Transfer learning should be avoided when the weights trained from your source task are different from your target task. For example, if your previous network was trained for classifying cats and dogs and your new network is trying to detect shoes and socks, there is going to be a problem as the weights transferred from your source to the target task will not be able to give you the best of results. Therefore, initialising the network with pre-trained weights that correspond with similar outputs to the one you are expecting is better than using weights with no correlation.
Removing layers from a pre-trained model will cause issues with the architecture of the model. If you remove the first layers, your model will have a low learning rate as it has to juggle working with low-level features. Removing layers reduces the number of parameters that can be trained, which can result in overfitting. Being able to use the correct amount of layers is vital in reducing overfitting, however, this is also a timely process.

Transfer Learning Cons:

Negative transfer learning: As I mentioned above, negative transfer learning is when a previous learning method obstructs the new task. This only occurs if the source and target are not similar enough, causing the first round of training to be too far off. Algorithms don't have to always agree with what we deem as similar, making it difficult to understand the fundamentals and standards of what type of training is sufficient.

Transfer Learning in 6 Steps:

Let’s dive into a better understanding of how transfer learning is implemented and the steps taken. There are 6 general steps taken in transfer learning and we will go through each of them.

1. Select Source Task: The first step is selecting a pre-trained model that holds an abundance of data, having a relationship between the input and output data with your chosen Target task.

2. Create a Base Model: Instantiate a base model with pre-trained weights. Pre-trained weights can be accessed through architectures such as Exception. This is developing your source model, so that it is better than the naive model we started with, ensuring some increase in learning rate.
3. Freeze Layers: To reduce initializing the weights again, freezing the layers from the pre-trained model is necessary. It will redeem the knowledge already learned and save you from training the model from scratch.

4. Add new Trainable Layers: Adding new trainable layers on top of the frozen layer, will convert old features into predictions on a new dataset.
5. Train the New Layers: The pre-trained model contains the final output layer already. The likelihood that the current output on the pre-trained model and the output you want from your model will be different, is high. Therefore, you have to train the model with a new output layer. Therefore, adding new dense layers and the final dense layer in correspondence to your expected model, will improve the learning rate and produce outputs of your desire.
6. Fine-tuning: You can improve the performance of your model by fine-tuning, which is done by unfreezing all or parts of the base models and then retraining the model with a very low learning rate. It is critical to use a low learning rate at this stage, as the model you are training is much larger than it was initially in the first round, along with it being a small dataset. As a result, you are at risk of overfitting if you apply large weight updates, therefore you want to fine-tune in an incremental way. Recompile the model as you have changed the model's behaviour and then retrain the model again, monitoring any overfitting feedback.

2. **Computer Vision:**

Computer Vision is a subfield of Deep Learning and Artificial Intelligence where humans teach computers to see and interpret the world around them.

While humans and animals naturally solve vision as a problem from a very young age, helping machines interpret and perceive their surroundings via vision remains a largely unsolved problem.

Limited perception of the human vision along with the infinitely varying scenery of our dynamic world is what makes Machine Vision complex at its core.

A brief history of computer vision

Like all great things in the world of technology, computer vision started with a cat.

Two Swedish scientists, Hubel and Wiesel, placed a cat in a restricting harness and an electrode in its visual cortex. Scientists showed the cat a series of images through a projector, hoping its brain cells in the visual cortex would start firing.

With no avail with images, the eureka moment happened when a projector slide was removed, and a single horizontal line of light appeared on the wall—

Neurons fired, emitting a crackling electrical noise.

The scientists had just realized that the early layers of the visual cortex respond to simple shapes, like lines and curves, much like those in the early layers of a deep neural network.

They then used an oscilloscope to create these and observe the brain's reaction.

This experiment marks the beginning of our understanding of the interconnection between computer vision and the human brain, which will be helpful for our understanding of artificial neural networks.

However, Before, cat brains entered the scene, analogue computer vision began as early as the 1950s at universities pioneering artificial intelligence.

Computer vision vs. human vision:

The notion that machine vision must be derived from the animal vision was predominant as early as 1959—when the neurophysiologists mentioned above tried to understand cat vision.

Since then, the history of computer vision is dotted with milestones formed by the rapid development of image capturing and scanning instruments complemented by state-of-the-art image processing algorithms' design.

The 1960s saw the emergence of AI as an academic field of study, followed by the development of the first robust Optical Character Recognition system in 1974.

By the 2000s, the focus of Computer Vision has been shifted to much more complex topics, including:

- Object identification
- Facial recognition
- Image Segmentation
- Image Classification

And more—

All of them have achieved commendable accuracies over the years.

The year 2010 saw the birth of the ImageNet dataset with millions of labelled images freely available for research. This led to the formation of the AlexNet architecture two years later— making it one of the biggest breakthroughs in Computer Vision, cited over 82K times.

Image Processing as a Part of Computer Vision

Digital Image Processing, or Image Processing, in short, is a subset of Computer Vision. It deals with enhancing and understanding images through various algorithms.

More than just a subset, Image Processing forms the precursor of modern-day computer vision, overseeing the development of numerous rule-based and optimization-based algorithms that have led machine vision to what it is today.

Image Processing may be defined as the task of performing a set of operations on an image based on data collected by algorithms to analyse and manipulate the contents of an image or the image data.

Now that you know the theory behind computer vision let’s talk about its practical side.

How does computer vision work?

Here’s a simple visual representation that answers this question on the most basic level.

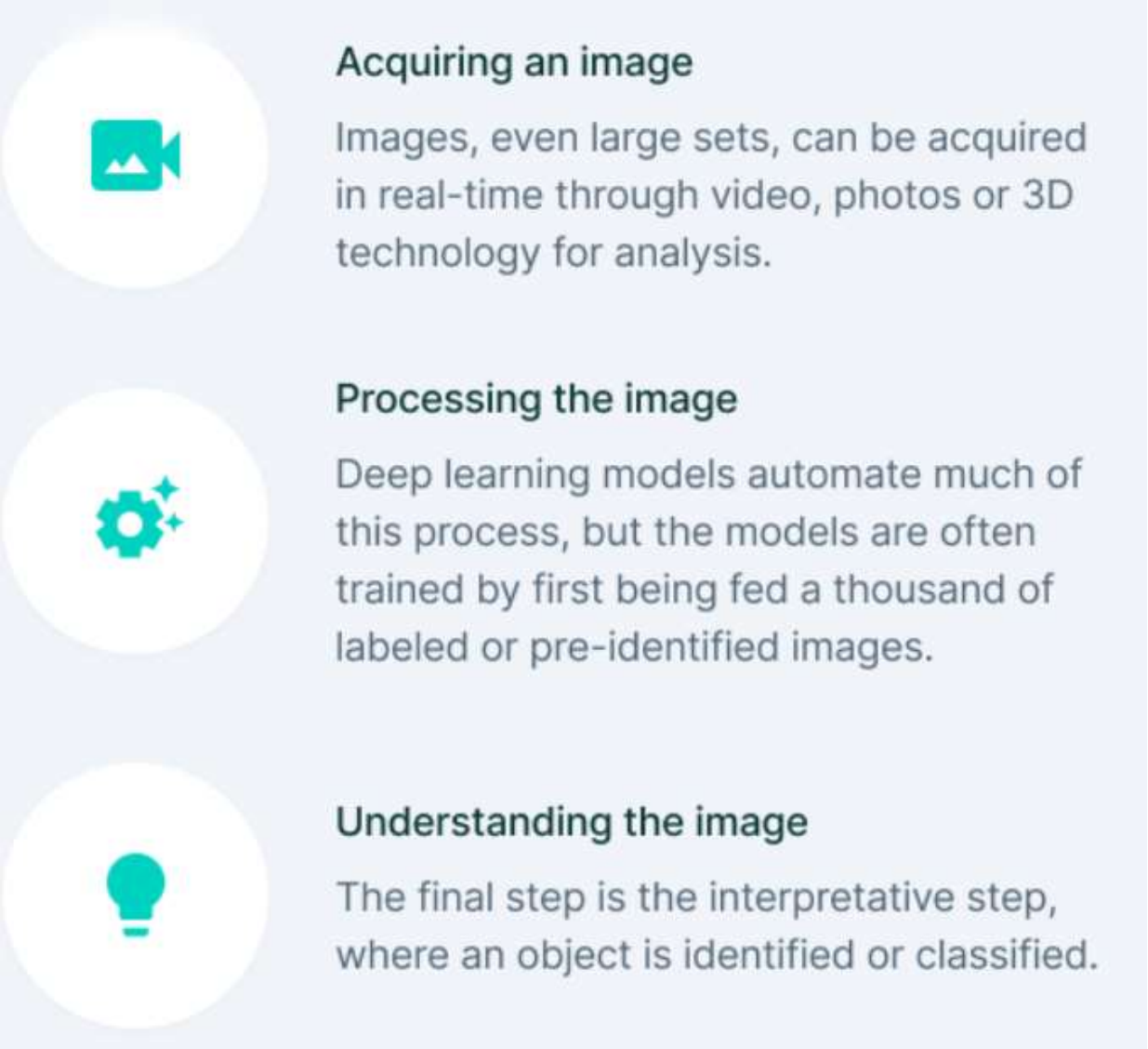


Figure 01: How does computer vision work?

However, While the three steps outlining the basics of computer vision seem easy, processing and understanding an image via machine vision are quite difficult.

Here's why, A image consists of several pixels, with a pixel being the smallest quanta in which the image can be divided into.

Computers process images in the form of an array of pixels, where each pixel has a set of values, representing the presence and intensity of the three primary colours: red, green, and blue.

All pixels come together to form a digital image.

The digital image, thus, becomes a matrix, and Computer Vision becomes a study of matrices. While the simplest computer vision algorithms use

linear algebra to manipulate these matrices, complex applications involve operations like convolutions with learnable kernels and down sampling via pooling.

Some operations commonly used in computer vision based on a Deep Learning perspective include:

1. **Convolution**: Convolution in computer vision is an operation in which a learnable kernel is "convolved" with the image. In other words—the kernel is slided across the image pixel by pixel, and an element-wise multiplication is performed between the kernel and the image at every pixel group.
2. **Pooling:** Pooling is an operation used to reduce the dimensions of an image by performing operations at a pixel level. A pooling kernel slides across the image, and only one pixel from the corresponding pixel group is selected for further processing, thus reducing the image size., example: Max Pooling, Average Pooling.
3. **Non-Linear Activations:** Non-Linear activations introduce non-linearity to the neural network, thereby allowing the stacking of multiple convolutions and pooling blocks to increase model depth.

Computer vision technology challenges:

One of the biggest challenges in machine vision is our lack of understanding of how the human brain and the human visual system works.

We have an enhanced and complex sense of vision that we can figure out at a very young age but are unable to explain the process by which we can understand what we see.

Furthermore, day-to-day tasks like walking across the street at the zebra crossing, pointing at something in the sky, checking out the time on the clock, require us to know enough about the objects around us to understand our surroundings.

Such aspects are quite different from simple vision, but are largely inseparable from it. The simulation of human vision via algorithms and mathematical representation thus requires the identification of an object in an image and an understanding of its presence and its behaviour.

1.1 Motivation of Work:

The world has not yet fully Recover from this pandemic and the vaccine that can effectively treat Covid-19 is yet to be discovered. However, to reduce the impact of the pandemic on the country's economy, several governments have allowed a limited number of economic activities to be resumed once the number of new cases of Covid19 has dropped below a certain level. As these countries cautiously restarting their economic activities, concerns have emerged regarding workplace safety in the new post-Covid-19 environment. To reduce the possibility of infection, it is advised that people should wear masks and maintain a distance of at least 1 meter from each other.

The world health organisation has clearly stated that until vaccines are found the wearing of masks and social distancing are key tools to reduce spread of virus.So it is important to make people wear masks in public places. In densely populated regions it is difficult to find the persons not wearing the face mask and warn them.Hence we are using image processing techniques for identification of persons wearing and not wearing face masks. In real time images are collected from the camera and it is processed

in Raspberry Pi embedded development kit. The real time images from the camera are compared with the trained dataset and detection of wearing or

The world health organisation has clearly stated that until vaccines are found the wearing of masks and social distancing are key tools to reduce spread of virus. So it is important to make people wear masks in public places. In densely populated regions it is difficult to find the persons not wearing the face mask and warn them. Hence, we are using image processing techniques for identification of persons wearing and not wearing face

masks. In real time images are collected from the camera and it is processed

in Raspberry Pi embedded development kit. The real time images from the camera are compared with the trained dataset and detection of wearing or

The world health organisation has clearly stated that until vaccines are found the wearing of masks and social distancing are key tools to reduce spread of virus.So it is important to make people wear masks in public places. In densely populated regions it is difficult to find the persons not

wearing the face mask and warn them.Hence we are using image processing techniques for identification of persons wearing and not wearing face masks. In real time images are collected from the camera and it is processed
in Raspberry Pi embedded development kit. The real time images from the camera are compared with the trained dataset and detection of wearing or Even world health organization has clearly stated that vaccines are found but still wearing of masks and social distancing are key tools to reduce spread of virus. So, it is important to make people wear masks in public places. In densely populated regions it is difficult to find the persons not wearing the face mask and warn them and Computer Vision has gained more attention in object detection and was used for human detection purposes and develop a face mask detection tool that can detect whether the individual is wearing mask or not. This can be done by evaluation of the classification results by analyzing real-time streaming from the Camera. Hence, we are using image processing techniques for identification of persons wearing and not wearing face masks. In real time images are collected from the camera and it is processed through our model. The real time images from the camera are compared with the trained dataset and detection of wearing or not. Our model is based on Transfer Learning, we need a training data set. It is the actual dataset used to train the model for performing various actions and we train our model and try to work that model on our testing dataset and if our model passes the bench mark of accuracy, then we can launch our model.

1.2 **Project Overview:**

The project which we are proposing we are trying to eliminate the cons of the existing system by making it more beneficial and reliable. In the current system, the accuracy of the system is very low and inefficient and even it doesn't guide or warn our user to wear the mask, in our project our UI will guide the user to wear the mask and even our program will send the mail to the authorized committee which will tell that this person is not wearing any mask by sending the picture as well. The System which we are proposing needs very low maintenance and it is automated, we just need to run the program and then it will handle everything else on its own

and by this, our work will be done with the least effort and effectively. We also added a feature that we can add a video or an image and our program can tell which person is wearing a mask and which person is not wearing a mask with an accuracy percentage. We are also trying to add some other warning devices which will warn the person that he/she is not wearing a mask like a beeping device or a LED which will turn Red when a person is not wearing any mask. If someone tries to get inside without wearing any mask then the machine will take the picture of that person and send it to the authorized committee in real-time and will start beeping and our UI will tell him/her that they need to wear the mask to control the spreading of Coronavirus.

In this system, a face is detected from an image that has several attributes in it. According to, research into face detection requires expression recognition, face tracking, and pose estimation. Given a solitary image, the challenge is to identify the face from the picture. Face detection is a difficult errand because the faces change in size, shape, color, etc. and they are not immutable. It becomes a laborious job for opaque images impeded by some other thing not confronting the camera, and so forth. Authors in think occlusive face detection comes with two major challenges:

1) unavailability of sizably voluminous datasets containing both masked and unmasked faces, and

2) exclusion of facial expression in the covered area. Utilizing the locally linear embedding (LLE) algorithm and the dictionaries trained on an immensely colossal pool of masked faces, synthesized mundane faces, several mislaid expressions can be recuperated and the ascendancy of facial cues can be mitigated to great extent. According to the work reported in, a convolutional neural network (CNN) in computer vision comes with a strict constraint regarding the size of the input image. The prevalent practice reconfigures the images before fitting them into the network to surmount the inhibition.

1.3 Problem Statement:

The main objective of the face detection model is to detect the face of individuals and conclude whether they are wearing masks or not at that particular moment when they are captured in the image.

1.4 Hardware Requirement:

We don’t require much of a hardware in this project, we just need a security camera to identify and recognize faces and their face mask, and we are also using a UI guidance system, which will necessitate the use of a speaker. We’d also need a computer to maintain all these things and where our software will be loaded; and for future, the ideal choice would be a Raspberry Pi or an Arduino Board, but as we're simply submitting software, we won't be utilizing one. Also, our program is OS independent as long as all dependencies are loaded. We developed it on our personal computers. We've also tested on a variety of operating systems, including Windows, Ubuntu, etc.
security camera

1.5 Software Requirement:

The most basic requirement is python, as the entire software is built on it. We would also require pip to install additional packages. A large number of Python modules would be required like TensorFlow and Keras Modules for face building modules and making predictions. Google Text to Speech (gTTS), Computer Vision (cv2), and Smtplib Module for sending Emails to the authorized committee. The most fundamental need is Github. GitHub is a version management and collaboration tool for programming. For receiving the response from the owner, we will use the web framework Flask, which is developed in Python and we are also using caffeemodel and prototxt file for face detection and we'll also need a dataset for training our data. We have used different text editors and IDEs like spyder, emacs, vscode, jupyter-notebooks for writing code.

2. LITERATURE SURVEY

2.1 Existing System:

In the current system for face mask detection, the efficiency is very low if a person is wearing a mask on the mouth but not covering their nose then also it doesn't warn that person but, in our project, if the efficiency is less than 75% our UI will start warning that person that they need to wear the mask properly and if he/she still doesn't wear mask properly then we will click the photo of that person and send the authorized committee for taking the further actions.

Furthermore, there is no automation for doing all the things adjacently, kind of, there was always a need for a person to maintain all these features adjacently and there are many places where committee assigns a person for this work only to report them who is wearing the mask and who is not which will increase the handwork and that person need to do everything on his own like warning people that they need to wear the mask and all so our project is solving every problem we might face in the future as well.

There is one more problem that the existing system is facing, i.e., the real-time pictures and the reporting of that pictures on time, the existing program takes at least 10-15 minutes for reporting that picture and in the meantime, that person might leave or wears a mask so that is also a concerning problem that we need to warn them in the real-time. The existing systems are controlled manually.

Also, existing systems are very complex and hard to maintain like we need to fetch details of every minute from the existing project and then we

can proceed and there is no real-time gathering of information and sending it to the authorized committee.

2.2 Proposed System:

The system we are proposing is capable to train the dataset of both persons wearing masks and without wearing masks and After training the model the system can predicting whether the person is wearing the mask or not with higher accuracy than any other existing model It also can access the webcam and predict the result and it is live feeding like if our model saw a person without mask it will hardly going to take 2 minutes to click the picture of that person and send it to the authorized committee and then they can press charger on that person, even our project got it's own UI which will work as a guidance system and it is also helping to make our project work with higher efficiency.

We also added a feature that we can add a video or an image and our program can tell which person is wearing a mask and which person is not wearing a mask with an accuracy percentage. We are also trying to add some other warning devices which will warn the person that he/she is not wearing a mask like a beeping device or a LED which will turn Red when a person is not wearing any mask.

2.3 Future Updates For Our System:

Our current program is only recognizing whether we are wearing mask or not, but if a person is eating something so we can't count that thing as a problem because at that time we surely need to remove mask, so our

model should be able to detect whether a person is eating anything or not. Another future update which can be required is that we haven't added this project as a website or as an App so we will need to make this project as a web based or application-based project as well.

2.4 Feasibility Study for our project:

The analyst does study to evaluate the likelihood of the usability of the system to the organization. The feasibility team ought to carry initial architecture and design of the high-risk requirements to the point at which we can answer the question like, if the requirements pose risk that would likely make the project infeasible. They have to check if the defects were reduced to a level matching the application needs. The analyst has to be sure that the organization has the resources needed in the requirements may be negotiated. We might also have to download some additional sources and packages file and we also need to require a security camera at a place where our committee wants to analyze, that is all we are going to need.

The following feasibility technique has been in this project.

1. Economic Feasibility
2. Technological Feasibility.

3. PROBLEM FORMULATION

- ❖ While installing the cv2 module we encounter a problem, which for the most part is fairly significant.
- ❖ The face a lot of problem to collect the dataset. However, we did it well in few days. but then the difficulty was to convert it into an array and add another dimension layer because our CNN Architecture needs fine tuning for it.
- ❖ We also face problems in speech recognition accuracy
- ❖ While installing the cv2 module we encountered a problem, which particularly is quite significant.
- ❖ We face problem while setting up smtp and mostly send mails to user.
- ❖ We also face problem to get the model trained easily because none of us have the GPU in our home. Also, google collab does not let us upload the dataset in an easy comfort. It actually took a lot of time to upload the dataset in the collab.
- ❖ Again, we face a problem in attaching the picture of a person without a mask in mail, which basically is quite significant.
- ❖ ❖We also really try kind of hard to actually make the confusion matrix scores well accurate as defined by the model.
- ❖ We also really try kind of hard to actually make the confusion matrix scores well accurate as defined by the model.

- The hardest part was is to train the model and add our labels layers in the last layers.
- We actually even lost the model and whole dataset and also needed to train it again but we actually saved the data and model in the Google Drive.
- While detecting if someone for all intents and purposes is there and starting the process of matching the face in a definitely big way.
- We have to increase the accuracy of our model so we have to add some other resources in it like we need to increase the rate of frame for our program.

4. OBJECTIVES

1) Detection of the Face, also accepts multiple Faces.

2) Check whether the person wear face mask or not.

3) If person wearing face mask, then show green box else raise alarm.

4) If person not wearing face mask for more than 30 seconds then send person's ROI to the owner's mail.

5) We want to guide a person as well that he/she should wear the mask so for that we created a UI as well.

6) We need to send the real time images of the person who is not wearing the mask so we need to increase the frames per second and we also need to enhance the backend work.

7) Our model should be precise with the results and the mask which a person is wearing should be on the perfect spot, i.e., covering the mouth as well as nose, so we have to work on that, our model should he very specific for nose and mouth covering.

5. METHODOLOGY

5.1 Incorporated Libraries, Packages and Concepts:

- **Tensorflow Framework:**

Tensor flow is an open-source software library.
Tensor flow was originally developed by researchers and engineers.
It is working on the Google Brain Team within Google's Machine Intelligence research organization the purposes of conducting machine learning and deep neural networks research.
It is an opensource framework to run deep learning and other statistical and predictive analytics workloads.
It is a python library that supports many classification and regression algorithms and more generally deep learning.
TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks.
It is a symbolic math library, and is also used for machine learning applications such as neural networks.
It is used for both research and production at Google, TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units).

Tensor Flow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.
Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.
The name Tensor Flow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors.

- **OpenCV:**

1.It is a cross-platform library using which we can develop real-time computer vision applications. 2.It mainly focuses on image processing, video capture and analysis including feature like face detection and object detection.
3. Currently Open CV supports a wide variety of programming languages like C++, Python, Java etc. and is available on different platforms including Windows, Linux, OS X, Android, iOS etc.
4. Also, interfaces based on CUDA and OpenCL are also under active development for high-speed GPU operations. Open CV-Python is the Python API of Open CV.
5. It combines the best qualities of Open CV C++ API and Python language.
6. OpenCV (Open-Source Computer Vision Library) is an opensource computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision

applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

7. The library has more than 2500optimized algorithms, which includes a comprehensive set of both classic and state-of -the-art computer vision and machine learning algorithms.

8.Algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

- **Numpy:**

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Num array into Numeric, with extensive modifications. NumPy is opensource software and has many contributors. The Python programming language was not initially designed for numerical

computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier.

An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become Numeric also variously called Numerical Python extensions or NumPy Hugunin, a graduate student at Massachusetts Institute of Technology (MIT) joined the Corporation for National Research Initiatives (CNRI) to work on J Python in 1997 leaving Paul Dubois of Lawrence Livermore National Laboratory (LLNL) to take over as maintainer. In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported num-array's features to Numeric, releasing the result as NumPy 1.0 in 2006.

This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy.

- **MATPLOT:**

Matplot is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, WX Python, Qt, or GTK+. There is also a procedural "Pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged SciPy makes use of Matplotlib. Matplotlib was originally written by John D. Hunter, has an active development community and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012 and further joined by Thomas Caswell.

- **Python:**

What exactly is Python? You may be wondering about that. You may be referring to this book because you wish to learn editing but are not familiar with editing languages. Alternatively, you may be familiar with programming languages such as C, C ++, C #, or Java and wish to learn more about Python language and how it compares to these "big word" languages.

- **Python Concepts:**

You can skip to the next chapter if you are not interested in how and why Python.

In this chapter, I will try to explain why I think Python is one of the best programming languages available and why it is such a great place to start. Python was developed into an easy-to-use programming language. It uses English words instead of punctuation, and has fewer syntax than other languages. Python is a highly developed, translated, interactive, and object-oriented language. Python translated - Interpreter processing Python during launch. Before using your software, you do not need to install it. This is similar to PERL and PHP editing languages. Python interactive - To write your own applications, you can sit in Python Prompt and communicate directly with the interpreter. Python Object-Oriented - Python supports the Object-Oriented program style or method, encoding the code within objects. Python is a language for beginners - Python is an excellent language for beginners, as it allows for the creation of a variety of programs, from simple text applications to web browsers and games.

  - **Python Features:**

    Python features include –

    1.Easy-to-learn - Python includes a small number of keywords, precise structure, and well-defined syntax. T This allows the student to learn the language faster

    2.Easy to read - Python code is clearly defined and visible to the

naked eye.

3.Easy-to-maintain - Python source code is easy to maintain.

4.Standard General Library - Python's bulk library is very portable and shortcut compatible with UNIX, Windows, and Macintosh.

5.Interaction mode - Python supports interaction mode that allows interaction testing and correction of captions errors.

6.Portable - Python works on a variety of computer systems and has the same user interface for all.

7.Extensible - Low-level modules can be added to Python interpreter. These modules allow system developers to improve the efficiency of their tools either by installing or customizing them.

8.Details - All major commercial information is provided by Python ways of meeting.

9.GUI Programming - Python assists with the creation and installation of a user interface for images of various program phones, libraries, and applications, including Windows MFC, Macintosh, and Unix's X Window.

10.Scalable - Major projects benefit from Python building and support, while Shell writing is not Aside from the characteristics stated above, Python offers a long list of useful features, some of which are described below. –

- It supports OOP as well as functional and structured programming methodologies.

- It can be used as a scripting language or compiled into Byte-code for large-scale application development.
- It allows dynamic type verification and provides very high-level dynamic data types.
- Automatic garbage pickup is supported by IT.

ADVANTAGES/BENEFITS OF PYTHON:

The diverse application of the Python language is a result of the combination of features which give this language an edge over others.

Some of the benefits of programming in Python include:

1. Presence of Third-Party Modules: The Python Package Index (PyPI) contains numerous third-party modules that make Python capable of interacting with most of the other languages and platforms.

2. Extensive Support Libraries: Python provides a large standard library which includes areas like internet protocols, string operations, web services tools and operating system interfaces. Many high use programming tasks have already been scripted into the standard library which reduces length of code to be written significantly.

3. Open Source and Community Development: Python language is developed under an OSI-approved opensource license, which makes it free to use and distribute, including for

commercial purposes. Further, its development is driven by the community which collaborates for its code through hosting conferences and mailing lists, and provides for its numerous modules.

4. Learning Ease and Support Available: Python offers excellent readability and uncluttered simple-to-learn syntax which helps beginners to utilize this programming language. The code style guidelines, PEP 8, provide a set of rules to facilitate the formatting of code. Additionally, the wide base of users and active developers has resulted in a rich internet resource bank to encourage development and the continued adoption of the language.

5. User-friendly Data Structures: Python has built-in list and dictionary data structures which can be used to construct fast runtime data structures. Further, Python also provides the option of dynamic high-level data typing which reduces the length of support code that is needed.

6. Productivity and Speed: Python has Clean object-oriented design, provides enhanced process control capabilities, and possesses strong integration and text processing capabilities and its own unit testing framework, all of which contribute to the increase in its speed and productivity. Python is considered a viable option for building complex multi-protocol network applications.

As can be seen from the above-mentioned points, Python offers a

number of advantages for software development. As upgrading of the language continues, its loyalist base could grow as well.

Python has five standard data types –

- ❖ Numbers
- ❖ String
- ❖ List
- ❖ Tuple
- ❖ Dictionary

- **Python Numbers:**

Numeric values are stored in number data types. When you give a number a value, it becomes a number object.

- **Python Strings:**

In this python uses a string is defined as a collection set of characters enclosed in quotation marks. Python allows you to use any number of quotes in pairs. The slice operator ([] and [:]) can be used to extract subsets of strings, with indexes starting at 0 at the start of the string and working their way to -1 at the end.

- **Python Lists:**

The most diverse types of Python data are lists. Items are separated by commas and placed in square brackets in the list ([]). Lists are similar to C-order in some ways. Listings can be for many types of data, which is one difference between them.

The slide operator ([] and [:]) can be used to retrieve values stored in the list, the indicators start with 0 at the beginning of the list and work their way to the end of the list. The concatenation operator of the list is a plus sign (+), while the repeater is an asterisk (*).

- **Python Tuples:**

A cone is a type of data type similar to a sequence of items. Cone is a set of values separated by commas. The pods, unlike the list, are surrounded by parentheses.

Lists are placed in parentheses ([]), and the elements and sizes can be changed, but the lumps are wrapped in brackets (()) and cannot be sorted. Powders are the same as reading lists only.

- **Python Dictionary:**

Python dictionaries in Python are a way of a hash table. They are similar to Perl's combination schemes or hashes, and are made up of two key numbers.

The dictionary key can be any type of Python, but numbers and strings are very common. Prices, on the other hand, can be anything you choose Python.

Curly braces () surround dictionaries, and square braces ([) are used to assign and access values. Different modes in python Python Normal and interactive are the two basic Python modes.

The scripted and completed.py files are executed in the Python interpreter in the regular manner. Interactive mode is a command line shell that provides instant response for each statement while

simultaneously running previously provided statements in active memory.
The feed program is assessed in part and whole as fresh lines are fed into the interpreter.

- **Pandas:**

Pandas is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users.
Pandas is a python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, realworld data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible opensource data analysis/manipulation tool available in any language. It is already well on its way toward this goal.
Explore data analysis with Python. Pandas Data Frames make manipulating your data easy, from selecting or replacing columns and indices to reshaping your data.
Pandas is well suited for many different kinds of data:
Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet.
Ordered and unordered (not necessarily fixed-frequency) time series data.

Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels.

Any other form of observational / statistical data sets. The data need not be labeled at all to be placed into a Pandas data structure.

- **Caffe Model:**

Caffe Model(Convolutional Architecture For Fast Embedding) is a deep learning framework that allows users to create image classification and image segmentation models.

  - **Deep Learning:**

    1.Deep learning is an AI function that mimics the workings of the human brain in processing data for use in detecting objects, recognizing speech, translating languages, and making decisions.

    2.Deep learning AI is able to learn without human supervision, drawing from data that is both unstructured and unlabeled.

    3.In this, face mask detection is built using Deep Learning technique called as Convolution Neural Networks (CNN).

    Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower-level features. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-

crafted features. Deep learning algorithms seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels, with higher-level learned features defined in terms of lower-level features.

- **Neural Network Versus Conventional Computers:**

  Neural networks take a different approach to problem solving than that of conventional computers. Conventional computers use an algorithmic approach i.e the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem-solving capability of conventional computers to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do.

  Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable. On the other hand, conventional computers use

a cognitive approach to problem solving; the way the problem is solved must be known and stated in small unambiguous instructions. These instructions are then converted to a high-level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault.

Neural networks and conventional algorithmic computers are not in competition but complement each other. There are tasks are more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to neural networks. Even more, a large number of tasks, require systems that use a combination of the two approaches (normally a conventional computer is used to supervise the neural network) in order to perform at maximum efficiency.

- **Convolution Neural Network:**

  A convolution neural network is a special architecture of artificial neural network proposed by yann lecun in 1988. One of the most popular uses of the architecture is image classification. CNNs have wide applications in image and video recognition, recommender systems and natural language processing. In this article, the example that this project will take is related to Computer Vision. However, the basic concept remains the same and can be applied to any other use-case!

CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function and all the tips and tricks that we developed for neural networks still apply on CNNs. In more detail the image is passed through a series of convolution, nonlinear, pooling layers and fully connected layers, then generates the output.

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks, most commonly applied to analyzing visual imagery. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the visual cortex. CNNs use relatively little pre-processing compared to other image classification algorithms. CNN is a special kind of multi- layer NNs applied to 2-d arrays (usually images), based on spatially localized neural input. CNN Generate ‘patterns of patterns’ for pattern recognition.

Each layer combines patches from previous layers. Convolutional Networks are trainable multistage architectures composed of multiple stages Input and output of each stage are sets of arrays called feature maps. At output, each feature map represents a particular feature extracted at all locations on

input. Each stage is composed of: a filter bank layer, a non-linearity layer, and a feature pooling layer. A ConvNet is composed of 1, 2 or 3 such 3-layer stages, followed by a classification module.

- **Convolution Layer:**

  It is always first. The image (matrix with pixel values) is entered into it. Image that the reacting of the input matrix begins at the top left of image. Next the software selects the smaller matrix there, which is called a filter. Then the filter produces convolution that is moves along the input image. The filter task is to multiple its value by the original pixel values. All these multiplications are summed up and one number is obtained at the end. Since the filter has read the image only in the upper left corner it moves further by one unit right performing a similar operation. After passing the filter across all positions, a matrix is obtained, but smaller than a input matrix.

  This operation, from a human perspective is analogous to identifying boundaries and simple Colors on the image. But in order to recognize the fish whole network is needed. The network will be -consists of several convolution layers mixed with nonlinear and pooling layers. Convolution is the first layer to extract features from an input image. Convolution features using small squares of input data. It is a mathematical

operation that takes two inputs such as image matrix and a filter or kernel.

- An image matrix of dimension (h x w x d)
- A filter (fh x fw x d)
- Outputs a volume dimension (h-fh+1) x (w-fw+1) x1.

- **Keras:**

KERAS is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. It also has extensive documentation and developer guides. Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel. Keras is a minimalist Python library for deep learning that can run on top of Theano or Tensor Flow.

It was developed to make implementing deep learning models as fast and easy as possible for research and development.

FOUR PRINCIPLES:

✓ Modularity: A model can be understood as a sequence or a graph

alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.

- ✓ Minimalism: The library provides just enough to achieve an outcome, no frills and maximizing readability.
- ✓ Extensibility: New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.
- ✓ Python: No separate model files with custom file formats. Everything is native Python. Keras is designed for minimalism and modularity allowing you to very quickly define deep learning models and run them on top of a Theano or TensorFlow backend.

- **System Architecture:**

Data Visualization.

Data Augmentation.

Splitting the data.

Labeling the Information.

Importing the Face detection.

Detecting the Faces with and without Masks.

Data Visualization:

In the first step, let us visualize the total number of images in our dataset in both categories. We can see that there are 690 images in the ‘yes’ class and 686 images in the ‘no’ class.

Data Augmentation:

In the next step, we augment our dataset to include number of images for our training. In this step of data augmentation, we rotate and flip each of the images in our dataset.

Splitting the data:

In this step, we split our data into the training set which will contain the images on which our model will be trained and the test set with the images on which our model will be tested.

Building the Model:

In the next step, we build our model by providing the training dataset to our model and then we can check the accuracy of our model by using that model on testing dataset.

Labeling the Information:

After building the model, we label two probabilities for our results. [‘0’ as ‘without_ ask’ and ‘1’ as ‘with_ mask’]. We are also setting the boundary rectangle color using the RGB values.

Importing the Face detection Program:

After this, we intend to use it to detect if we are wearing a face mask using our PC’s webcam. For this, first, we need to implement face detection. In this, we are using the caffe model for detecting the features of the face.

Detecting the Faces with and without Masks:

In the last step, we use the OpenCV library to run an infinite loop to use our web camera in which we detect the face.

6. Code Review and Explanation:-

1. Dataset

The dataset we'll be using here today was created and consists of **1,376 images** belonging to two classes:

- with_mask
  : 690 images
- without_mask
  : 686 images

Our goal is to train a custom deep learning model to detect whether a person *is* or *is not* wearing a mask.

How was our face mask dataset created?

I and My fellow team mates were been feeling down and depressed about the state of the world — thousands of people are dying each day, and for many of us, there is very little (if anything) we can do.

To help keep her spirts up, I decided to distract herself by applying computer vision and deep learning to solve a real-world problem:

- Best case scenario — she could use her project to help others
- Worst case scenario — it gave her a much needed mental escape

Either way, it's win-win!

As programmers, developers, and computer vision/deep learning practitioners, we can *all* learn from a book page— let your skills become

your distraction and your haven.

To create this dataset, I had the ingenious solution of:

1. Taking *normal images of faces*
2. Then creating a *custom computer vision Python script* to add face masks to them, **thereby creating an *artificial* (but still real-world applicable) dataset**

This method is actually a lot easier than it sounds once you apply facial landmarks to the problem.

Facial landmarks allow us to automatically infer the location of facial structures, including:

- Eyes
- Eyebrows
- Nose
- Mouth
- Jawline

To use facial landmarks to build a dataset of faces wearing face masks, we need to first start with an image of a person *not* wearing a face mask:

From there, we apply face detection to compute the bounding box location of the face in the image:

Figure 2: The next step is to apply face detection. Here we've used a deep learning method to perform face detection with OpenCV.

Once we know *where* in the image the face is, we can extract the face Region of Interest (ROI):

Figure 3: The next step is to extract the face ROI with OpenCV and NumPy slicing.

And from there, we apply facial landmarks, allowing us to localize the eyes, nose, mouth, etc.:

Figure 4: Then, we detect facial landmarks using dlib so that we know where to place a mask on the face.

Next, we need an image of a mask (with a transparent background) such as the one below:

Figure 5: An example of a COVID-19/Coronavirus face mask/shield. This face mask will be overlaid on the original face ROI automatically since we know the face landmark locations.

This mask will be *automatically* applied to the face by using the facial landmarks (namely the points along the chin and nose) to compute *where* the mask will be placed.

The mask is then resized and rotated, placing it on the face:

Figure 6: In this figure, the face mask is placed on the person's face in the original frame. It is difficult to tell at a glance that the COVID-19 mask has been applied with computer vision by way of OpenCV and dlib face landmarks.

We can then repeat this process for all of our input images, thereby creating our artificial face mask dataset:

Figure 7: An artificial set of COVID-19 face mask images is shown. This set will be part of our "with mask" / "without mask" dataset for COVID-19 face mask detection with computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras.

However, there is a caveat you should be aware of when using this method to artificially create a dataset!

If you use a set of images to create an artificial dataset of people wearing masks, **you *cannot* "re-use" the images *without masks* in your training set — you still need to gather non-face mask images that were *not* used in the artificial generation process!**

If you include the original images used to generate face mask samples as non-face mask samples, your model will become heavily biased and fail to generalize well. Avoid that at all costs by taking the time to gather new examples of faces without masks.

Project structure

```
COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning
$ tree --dirsfirst --filelimit 10
.
├── dataset
│   ├── with_mask [690 entries]
│   └── without_mask [686 entries]
├── examples
│   ├── example_01.png
│   ├── example_02.png
│   └── example_03.png
├── face_detector
│   ├── deploy.prototxt
│   └── res10_300x300_ssd_iter_140000.caffemodel
├── detect_mask_image.py
├── detect_mask_video.py
├── mask_detector.model
├── plot.png
└── train_mask_detector.py

5 directories, 10 files
```

The

dataset/directory contains the data described in the *"Our COVID-19 face mask detection dataset"* section. Three image examples/ are provided so that you can test the static image face mask detector.

- train_mask_detector.py
  : Accepts our input dataset and fine-tunes MobileNetV2 upon it to create our
  mask_detector.model
  . A training history
  plot.png
  containing accuracy/loss curves is also produced
- detect_mask_image.py
  : Performs face mask detection in static images
- detect_mask_video.py
  : Using your webcam, this script applies face mask detection to every frame in the stream

Train_Mask_Detector.py

Step 1: Import the Libararies

The imports for our training script may look intimidating to you either because there are so many or you are new to deep learning. If you are new, Our set of tensorflow.keras imports allow for:

- Data augmentation
- Loading the MobilNetV2 classifier (we will fine-tune this model with pre-trained ImageNet weights)
- Building a new fully-connected (FC) head
- Pre-processing
- Loading image data

We'll use scikit-learn (sklearn) for binarizing class labels, segmenting our dataset, and printing a classification report.

My imutils paths implementation will help us to find and list images in our dataset. And we'll use matplotlib

```
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import os
```

Step 2: construct the argument parser and parse the arguments

Our command line arguments include:

- --dataset : The path to the input dataset of faces and and faces with masks
- --plot : The path to your output training history plot, which will be generated using matplotlib

- --model : The path to the resulting serialized face mask classification model

```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
        help="path to input dataset")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
        help="path to output loss/accuracy plot")
ap.add_argument("-m", "--model", type=str,
        default="mask_detector.model",
        help="path to output face mask detector model")
args = vars(ap.parse_args())
```

Step3: initialize the initial learning rate, number of epochs to train for,

and batch size and grab the list of images in our dataset directory, then initialize

Here, I've specified hyperparameter constants including my initial learning rate, number of training epochs, and batch size. Later, we will be applying a learning rate decary schedule, which is why we've named the learning rate variable INIT_LR.

At this point, we're ready to load and pre-process our training data:

```
38
39 # initialize the initial learning rate, number of epochs to train for,
40 # and batch size
41
42 INIT_LR = 1e-4
43 EPOCHS = 20
44 BS = 32
45
46 # grab the list of images in our dataset directory, then initialize
47 # the list of data (i.e., images) and class images
48 print("[INFO] loading images...")
49 imagePaths = list(paths.list_images(args["dataset"]))
50 data = []
51 labels = []
52
```

Step4: loop over the image paths and construct the training image generator for data augmentation

In this block, we are:

Grabbing all of the imagePaths in the dataset (Line 44),

Initializing data and labels lists (Lines 45 and 46)

Looping over the imagePaths and loading + pre-processing images (Lines 49-60).

Pre-processing steps include resizing to 224×224 pixels, conversion to array format, and scaling the pixel intensities in the input image to the range [-1, 1] (via the preprocess_input convenience function)

Appending the pre-processed image and associated label to the data and labels lists, respectively (Lines 59 and 60)

Ensuring our training data is in NumPy array format (Lines 63 and 64)

The above lines of code assume that your entire dataset is small enough to fit into memory. If your dataset is larger than the memory you have available, I suggest using HDF5, Our data preparation work isn't done yet. Next, we'll encode our labels , partition our dataset, and prepare for data augmentation:

```
53 # loop over the image paths
54 for imagePath in imagePaths:
55         # extract the class label from the filename
56         label = imagePath.split(os.path.sep)[-2]
57
58         # load the input image (224x224) and preprocess it
59         image = load_img(imagePath, target_size=(224, 224))
60         image = img_to_array(image)
61         image = preprocess_input(image)
62
63         # update the data and labels lists, respectively
64         data.append(image)
65         labels.append(label)
66
67 # convert the data and labels to NumPy arrays
68 data = np.array(data, dtype="float32")
69 labels = np.array(labels)
70
71 # perform one-hot encoding on the labels
72 lb = LabelBinarizer()
73 labels = lb.fit_transform(labels)
74 labels = to_categorical(labels)
75
76 # partition the data into training and testing splits using 75% of
77 # the data for training and the remaining 25% for testing
78 (trainX, testX, trainY, testY) = train_test_split(data, labels,
79         test_size=0.20, stratify=labels, random_state=42)
80
81 # construct the training image generator for data augmentation
82 aug = ImageDataGenerator(
83         rotation_range=20,
84         zoom_range=0.15,
85         width_shift_range=0.2,
86         height_shift_range=0.2,
87         shear_range=0.15,
88         horizontal_flip=True,
89         fill_mode="nearest")
```

one-hot encode our class labels, meaning that our data will be in the following format:

As you can see, each element of our labels array consists of an array in which only one index is "hot" (i.e., 1).

Using scikit-learn's convenience method, **Lines 73 and 74** segment our

data into 80% training and the remaining 20% for testing.

During training, we'll be applying on-the-fly mutations to our images in an effort to improve generalization. This is known as data augmentation, where the random rotation, zoom, shear, shift, and flip parameters are established on **Lines 77-84**. We'll use the aug object at training time.

But first, we need to prepare MobileNetV2 for fine-tuning:

Step5: Load the MobileNetV2 network, ensuring the head FC layer sets are let off and loop over all layers in the base model and freeze them so they will *not* be updated during the first training process

Fine-tuning setup is a three-step process:

1. Load MobileNet with pre-trained ImageNet weights, leaving off head of network (**Lines 88 and 89**)
2. Construct a new FC head, and append it to the base in place of the old head (**Lines 93-102**)
3. Freeze the base layers of the network (**Lines 106 and 107**). The weights of these base layers will not be updated during the process of backpropagation, whereas the head layer weights *will* be tuned.

Fine-tuning is a strategy I nearly always recommend to establish a baseline model while saving considerable time.

```
 90
 91 # load the MobileNetV2 network, ensuring the head FC layer sets are
 92 # left off
 93 baseModel = MobileNetV2(weights="imagenet", include_top=False,
 94         input_tensor=Input(shape=(224, 224, 3)))
 95
 96 # construct the head of the model that will be placed on top of the
 97 # the base model
 98 headModel = baseModel.output
 99 headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
100 headModel = Flatten(name="flatten")(headModel)
101 headModel = Dense(128, activation="relu")(headModel)
102 headModel = Dropout(0.5)(headModel)
103 headModel = Dense(2, activation="softmax")(headModel)
104
105 # place the head FC model on top of the base model (this will become
106 # the actual model we will train)
107 model = Model(inputs=baseModel.input, outputs=headModel)
108
109 # loop over all layers in the base model and freeze them so they will
110 # *not* be updated during the first training process
111 for layer in baseModel.layers:
112         layer.trainable = False
113
```

Step6: Compile the model, train it and serialize the model to disk

With our data prepared and model architecture in place for fine-tuning, we're now ready to compile and train our face mask detector network: compile our model with the Adam optimizer, a learning rate decay schedule, and binary cross-entropy. If you're building from this training script with > *2* classes, be sure to use categorical cross-entropy.

Face mask training is launched via **Lines 117-122**. Notice how our data augmentation object (aug) will be providing batches of mutated image data.

Once training is complete, we'll evaluate the resulting model on the test set:

Here, **Lines 126-130** make predictions on the test set, grabbing the highest probability class label indices. Then, we print a classification report in the

terminal for inspection.

Line 138 serializes our face mask classification model to disk

```
114 # compile our model
115 print("[INFO] compiling model...")
116 opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
117 model.compile(loss="binary_crossentropy", optimizer=opt,
118         metrics=["accuracy"])
119
120 # train the head of the network
121 print("[INFO] training head...")
122 H = model.fit(
123         aug.flow(trainX, trainY, batch_size=BS),
124         steps_per_epoch=len(trainX) // BS,
125         validation_data=(testX, testY),
126         validation_steps=len(testX) // BS,
127         epochs=EPOCHS)
128
129 # make predictions on the testing set
130 print("[INFO] evaluating network...")
131 predIdxs = model.predict(testX, batch_size=BS)
132
133 # for each image in the testing set we need to find the index of the
134 # label with corresponding largest predicted probability
135 predIdxs = np.argmax(predIdxs, axis=1)
136
137 # show a nicely formatted classification report
138 print(classification_report(testY.argmax(axis=1), predIdxs,
139         target_names=lb.classes_))
140
141 # serialize the model to disk
142 print("[INFO] saving mask detector model...")
143 model.save(args["model"], save_format="h5")
144
```

Step7: Plot the training loss and accuracy

```
145 # plot the training loss and accuracy
146 N = EPOCHS
147 plt.style.use("ggplot")
148 plt.figure()
149 plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
150 plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
151 plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
152 plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
153 plt.title("Training Loss and Accuracy")
154 plt.xlabel("Epoch #")
155 plt.ylabel("Loss/Accuracy")
156 plt.legend(loc="lower left")
157 plt.savefig(args["plot"])
```

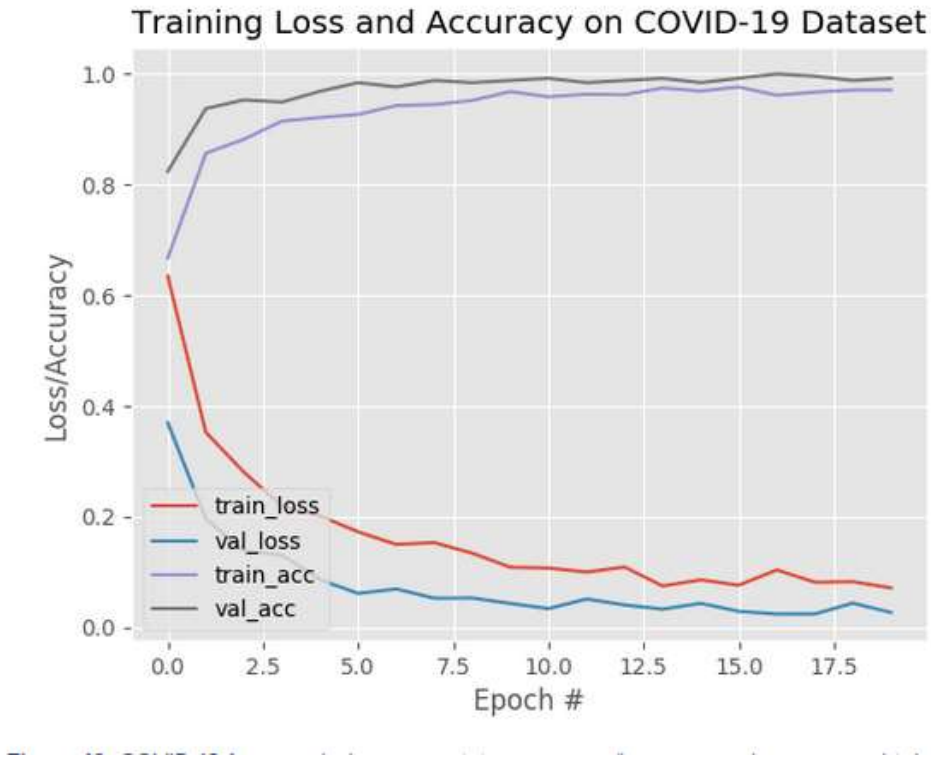


Figure 10: COVID-19 face mask detector training accuracy/loss curves demonstrate high accuracy and little signs of overfitting on the data. We're now ready to apply our knowledge of computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras to perform face mask detection.

Looking at **Figure 10**, we can see there are little signs of overfitting, with the validation loss *lower* than the training loss.

Given these results, we are hopeful that our model will generalize well to images *outside* our training and testing set.

Implementing our COVID-19 face mask detector for images with OpenCV

Now that our face mask detector is trained, let's learn how we can:

1. Load an input image from disk
2. Detect faces in the image
3. Apply our face mask detector to classify the face as either with_mask or without_mask

Open up the detect_mask_image.py file in your directory structure, and let's get started:

Our driver script requires three TensorFlow/Keras imports to

(1) load our MaskNet model and

(2) pre-process the input image.

OpenCV is required for display and image manipulations.

```
#By Shubharthak
# USAGE
# python detect_mask_image.py --image images/pic1.jpeg

# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import numpy as np
import argparse
import cv2
import os
```

Our four command line arguments include:

- --image: The path to the input image containing faces for inference
- --face: The path to the face detector model directory (we need to localize faces prior to classifying them)
- --model: The path to the face mask detector model that we trained earlier in this tutorial
- --confidence: An optional probability threshold can be set to override 50% to filter weak face detections

```
13 def mask_image():
14         # construct the argument parser and parse the arguments
15         ap = argparse.ArgumentParser()
16         ap.add_argument("-i", "--image", required=True,
17                 help="path to input image")
18         ap.add_argument("-f", "--face", type=str,
19                 default="face_detector",
20                 help="path to face detector model directory")
21         ap.add_argument("-m", "--model", type=str,
22                 default="mask_detector.model",
23                 help="path to trained face mask detector model")
24         ap.add_argument("-c", "--confidence", type=float, default=0.5,
25                 help="minimum probability to filter weak detections")
26         args = vars(ap.parse_args())
27
```

Next, we'll load both our face detector and face mask classifier models:

```
28         # load our serialized face detector model from disk
29         print("[INFO] loading face detector model...")
30         prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
31         weightsPath = os.path.sep.join([args["face"],
32                 "res10_300x300_ssd_iter_140000.caffemodel"])
33         net = cv2.dnn.readNet(prototxtPath, weightsPath)
34
35         # load the face mask detector model from disk
36         print("[INFO] loading face mask detector model...")
37         model = load_model(args["model"])
38
```

With our deep learning models now in memory, our next step is to load and pre-process an input image:

Upon loading our --image

from disk (**Line 37**), we make a copy and grab frame dimensions for future scaling and display purposes (**Lines 38 and 39**).

Pre-processing is handled by OpenCV's blobFromImage function (**Lines 42 and 43**). As shown in the parameters, we resize to *300×300* pixels and perform mean subtraction.

Lines 47 and 48 then perform face detection to localize *where* in the image all faces are.

Once we know where each face is predicted to be, we'll ensure they meet the --confidence threshold before we extract the faceROIs:

```
38
39          # load the input image from disk, clone it, and grab the image spatial
40          # dimensions
41          image = cv2.imread(args["image"])
42          orig = image.copy()
43          (h, w) = image.shape[:2]
44
45          # construct a blob from the image
46          blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),
47                  (104.0, 177.0, 123.0))
48
49          # pass the blob through the network and obtain the face detections
50          print("[INFO] computing face detections...")
51          net.setInput(blob)
52          detections = net.forward()
53
```

Here, we loop over our detections and extract the confidence to measure against the --confidence threshold (**Lines 51-58**). We then compute bounding box value for a particular face and ensure that the box falls within the boundaries of the image (**Lines 61-67**).

```
53
54          # loop over the detections
55          for i in range(0, detections.shape[2]):
56                  # extract the confidence (i.e., probability) associated with
57                  # the detection
58                  confidence = detections[0, 0, i, 2]
59
60                  # filter out weak detections by ensuring the confidence is
61                  # greater than the minimum confidence
62                  if confidence > args["confidence"]:
63                          # compute the (x, y)-coordinates of the bounding box for
64                          # the object
65                          box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
66                          (startX, startY, endX, endY) = box.astype("int")
67
68                          # ensure the bounding boxes fall within the dimensions of
69                          # the frame
70                          (startX, startY) = (max(0, startX), max(0, startY))
71                          (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
72
```

Next, we'll run the face ROI through our MaskNet model:

```
72
73                      # extract the face ROI, convert it from BGR to RGB channel
74                      # ordering, resize it to 224x224, and preprocess it
75                      face = image[startY:endY, startX:endX]
76                      face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
77                      face = cv2.resize(face, (224, 224))
78                      face = img_to_array(face)
79                      face = preprocess_input(face)
80                      face = np.expand_dims(face, axis=0)
81
82                      # pass the face through the model to determine if the face
83                      # has a mask or not
84                      (mask, withoutMask) = model.predict(face)[0]
85
```

In this block, we:

- Extract the face ROI via NumPy slicing (**Line 71**)
- Pre-process the ROI the same way we did during training (**Lines 72-76**)
- Perform mask detection to predict with_mask or without_mask (**Line 80**)

From here, we will annotate and display the result!

First, we determine the class label based on probabilities returned by the mask detector model (**Line 84**) and assign an associated color for the

annotation (**Line 85**). The color will be “green” for with_mask

and “red” for without_mask.

We then draw the label text (including class and probability), as well as a bounding box rectangle for the face, using OpenCV drawing functions (**Lines 92-94**).

Once all detections have been processed, **Lines 97 and 98** display the output image.

```
                    # determine the class label and color we'll use to draw
                    # the bounding box and text
                    label = "Mask" if mask > withoutMask else "No Mask"
                    color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

                    # include the probability in the label
                    label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

                    # display the label and bounding box rectangle on the output
                    # frame
                    cv2.putText(image, label, (startX, startY - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
                    cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)

        # show the output image
        cv2.imshow("Output", image)
        cv2.waitKey(0)

if __name__ == "__main__":
        mask_image()
```

Implementing our COVID-19 face mask detector in real-time video streams with OpenCV

At this point, we know we can apply face mask detection to static images — ***but what about real-time video streams?***

Is our COVID-19 face mask detector capable of running in real-time?

Let's find out. Open up the detect_mask_video.py file in your directory structure, and insert the following code:

```
#By Shubharthak
# USAGE
# python detect_mask_video.py

# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import threading
import numpy as np
import argparse
import imutils
import time
import cv2
import os
from send_mail import sendEmail
from speaking import speak
from gtts import gTTS
import speech_recognition as sr
import datetime
i = 1
username = os.environ.get('mymail')
password = os.environ.get('mypass')
```

The algorithm for this script is the same, but it is pieced together in such a way to allow for processing every frame of your webcam stream.

Thus, the only difference when it comes to imports is that we need a

VideoStream class and time. Both of these will help us to work with the stream. We'll also take advantage of imutils for its aspect-aware resizing method.

Our face detection/mask prediction logic for this script is in the

detect_and_predict_mask function:

```
38
39 def detect_and_predict_mask(frame, faceNet, maskNet):
40     # grab the dimensions of the frame and then construct a blob
41     # from it
42     (h, w) = frame.shape[:2]
43     blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
44                                  (104.0, 177.0, 123.0))
45
46     # pass the blob through the network and obtain the face detections
47     faceNet.setInput(blob)
48     detections = faceNet.forward()
49
50     # initialize our list of faces, their corresponding locations,
51     # and the list of predictions from our face mask network
52     faces = []
53     locs = []
54     preds = []
55
```

By defining this convenience function here, our frame processing loop will be a little easier to read later.

This function detects faces and then applies our face mask classifier to each face ROI. Such a function consolidates our code — it could even be moved to a separate Python file if you so choose.

Our

detect_and_predict_mask

function accepts three parameters:

- frame : A frame from our stream
- faceNet: The model used to detect where in the image faces are
- maskNet: Our COVID-19 face mask classifier model

Inside, we construct a blob, detect faces, and initialize lists, two of which

the function is set to return. These lists include our faces (i.e., ROIs),locs (the face locations), and preds (the list of mask/no mask predictions).

From here, we'll loop over the face detections:

```
55
56      # loop over the detections
57      for i in range(0, detections.shape[2]):
58          # extract the confidence (i.e., probability) associated with
59          # the detection
60          confidence = detections[0, 0, i, 2]
61
62          # filter out weak detections by ensuring the confidence is
63          # greater than the minimum confidence
64          if confidence > args["confidence"]:
65              # compute the (x, y)-coordinates of the bounding box for
66              # the object
67              box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
68              (startX, startY, endX, endY) = box.astype("int")
69
70              # ensure the bounding boxes fall within the dimensions of
71              # the frame
72              (startX, startY) = (max(0, startX), max(0, startY))
73              (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
74
```

nside the loop, we filter out weak detections (**Lines 34-38**) and extract bounding boxes while ensuring bounding box coordinates do not fall outside the bounds of the image (**Lines 41-47**).

Next, we'll add face ROIs to two of our corresponding lists:

```
74
75              # extract the face ROI, convert it from BGR to RGB channel
76              # ordering, resize it to 224x224, and preprocess it
77              face = frame[startY:endY, startX:endX]
78              if face.any():
79                  face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
80                  face = cv2.resize(face, (224, 224))
81                  face = img_to_array(face)
82                  face = preprocess_input(face)
83
84                  # add the face and bounding boxes to their respective
85                  # lists
86                  faces.append(face)
87                  locs.append((startX, startY, endX, endY))
88
```

After extracting face ROIs and pre-processing (**Lines 51-56**), we append the the face ROIs and bounding boxes to their respective lists.

We're now ready to run our faces through our mask predictor:

```
89      # only make a predictions if at least one face was detected
90      if len(faces) > 0:
91          # for faster inference we'll make batch predictions on *all*
92          # faces at the same time rather than one-by-one predictions
93          # in the above `for` loop
94          faces = np.array(faces, dtype="float32")
95          preds = maskNet.predict(faces, batch_size=32)
96
97      # return a 2-tuple of the face locations and their corresponding
98      # locations
99      return (locs, preds)
```

The logic here is built for speed. First we ensure at least one face was detected (**Line 63**) — if not, we'll return empty preds.

Line 72 returns our face bounding box locations and corresponding mask/not mask predictions to the caller.

Next, we'll define our command line arguments:

```
108 # construct the argument parser and parse the arguments
109 ap = argparse.ArgumentParser()
110 ap.add_argument("-f", "--face", type=str,
111                 default="face_detector",
112                 help="path to face detector model directory")
113 ap.add_argument("-m", "--model", type=str,
114                 default="mask_detector.model",
115                 help="path to trained face mask detector model")
116 ap.add_argument("-c", "--confidence", type=float, default=0.5,
117                 help="minimum probability to filter weak detections")
118 args = vars(ap.parse_args())
119
```

Our command line arguments include:

- --face: The path to the face detector directory
- --model: The path to our trained face mask classifier
- --confidence: The minimum probability threshold to filter weak face detections

With our imports, convenience function, and command line args

ready to go, we just have a few initializations to handle before we loop over frames:

```
121 # load our serialized face detector model from disk
122 print("[INFO] loading face detector model...")
123 prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
124 weightsPath = os.path.sep.join([args["face"],
125                                 "res10_300x300_ssd_iter_140000.caffemodel"])
126 faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
127
128 # load the face mask detector model from disk
129 print("[INFO] loading face mask detector model...")
130 maskNet = load_model(args["model"])
131
132 # initialize the video stream and allow the camera sensor to warm up
133 print("[INFO] starting video stream...")
134 vs = VideoStream(src=0).start()
135 time.sleep(2.0)
136 count = 0
```

Here we have initialized our:

- Face detector
- COVID-19 face mask detector
- Webcam video stream

Let’s proceed to loop over frames in the stream:

```
138 # loop over the frames from the video stream
139 while True:
140     # grab the frame from the threaded video stream and resize it
141     # to have a maximum width of 400 pixels
142     frame = vs.read()
143     frame = imutils.resize(frame, width=400)
144     # detect faces in the frame and determine if they are wearing a
145     # face mask or not
146     (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
147
```

We begin looping over frames on **Line 103**. Inside, we grab a frame from the stream and resize it (**Lines 106 and 107**).

From there, we put our convenience utility to use; **Line 111** detects and predicts whether people are wearing their masks or not.

Let's post-process (i.e., annotate) the COVID-19 face mask detection results:

```
147
148     # loop over the detected face locations and their corresponding
149     # locations
150     for (box, pred) in zip(locs, preds):
151         # unpack the bounding box and predictions
152         (startX, startY, endX, endY) = box
153         (mask, withoutMask) = pred
154
155         # determine the class label and color we'll use to draw
156         # the bounding box and text
157         if mask < withoutMask:
158             label = "No Mask"
159             color = (0, 0, 255)
160             flag = 1
161             os.system('aplay output4.wav &')
162             count += 1
163         else:
164             count = 0
165             label = "Mask"
166             color = (0, 255, 0)
167
168         # include the probability in the label
169         label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
170
171         # display the label and bounding box rectangle on the output
172         # frame
173         cv2.putText(frame, label, (startX, startY - 10),
174                     cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
175         cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
176
177         if count >= 15:
```

Inside our loop over the prediction results (beginning on **Line 115**), we:

- Unpack a face bounding box and mask/not mask prediction (**Lines 117 and 118**)
- Determine the label and color (**Lines 122-126**)
- Annotate the label and face bounding box (**Lines 130-132**)

Finally, we display the results and perform cleanup:

```
186     # show the output frame
187     cv2.imshow("Frame", frame)
188     key = cv2.waitKey(1) & 0xFF
189
190     # if the `q` key was pressed, break from the loop
191     if key == ord("q"):
192         break
193
194 # do a bit of cleanup
195 cv2.destroyAllWindows()
196 vs.stop()
```

After the frame is displayed, we capture key presses. If the user presses q (quit), we break out of the loop.

Mail, gTTS and Alert Algorithm:-

gTTS:-

Import the gTTS library and os module

```
1 from gtts import gTTS
2 import os
```

Now import the speak function, the speak function basically use gTTS

module to convert text to speech and save it as audio in mp3 format.

Then we use os.system method to play the audio using mpg123 player.

```
def speak(text):
    speech = gTTS(text=text, lang="en", slow=False)
    speech.save("text.mp3")
    os.system("mpg123 text.mp3")
```

Finally, Write wishMe function logic, In this function, it will wish the person according to the current time. We will get the time using import datetime module and datetime.now() method

```
def wishMe():
    '''
    It wishes the User according the time and return the file which played using os.system()
    '''
    hour = int(datetime.datetime.now().hour)
    if hour >= 0 and hour < 12:
        speak("Good Morning!")
    elif hour >= 12 and hour < 18:
        speak("Good Afternoon! ")
    else:
        speak("Good Evening!")
    speak('This is a Face Mask Detection Program. We will recognize the person whether he or she is wearing mask or not. Please wear a mask.')
```

Alert:-

Alert system is nothing but playing an alarm wav file using aplay from os.system. The thing to notice is we are calling it in background as we don’t want to interfere are current video stream.

```
flag = 1
os.system('aplay output4.wav &')
count += 1
```

Mail Notification:-

Now, to implement mail we are using 2 methods, one for mailing and another for taking pic / ROI(region of interest) of current users who didn't put the face mask.

First of all import the smtp module, os and imghdr

```
from email.message import EmailMessage
import smtplib
import os
import imghdr
#authlogin
username = os.environ.get('mymail')
```

Next, Use os.environ.get() method to load the username and password environment variables from your system

```
username = os.environ.get('mymail')
password = os.environ.get('mypass')
```

Now define sendEmail which will take a list of user pic location, owner's mail, and subject.

Through, smptplib method we are sending the mail but first we set the pic using imghdr attachment

```
def sendEmail(files=[],to='shubharthakofficial@gmail.com', content='This person is not wearing mask'):
    msg = EmailMessage()
    msg['Subject'] = 'Mail From Face Mask Detection'
    msg['From'] = username
    msg['To'] = to
    msg.set_content(content)
    for file in files:
        with open (file, 'rb') as f:
            file_data = f.read()
            file_type = imghdr.what(f.name)
            file_name = f.name

        msg.add_attachment(file_data, maintype='image', subtype=file_type,filename=file_name)
        print('Email Have been sent!!')
    with smtplib.SMTP_SSL("smtp.gmail.com", 465) as smtp:
        smtp.login(username, password)
        smtp.send_message(msg)

#sendEmail(input('Enter the sending email: '), input('Enter the message: '))
sendEmail()
```

Next, take_picture method will be used to take picture when user didn't put the face mask enforcement till 15 seconds.

We will take the frame, and ROI locations, and take the picture using cv2.imwrite method and append the image in a list and return the list finally.

```
100
101 def take_pic(frame,startX, endX, startY, endY):
102     img_list = []
103     cv2.imwrite(f'send_images/{i}.jpeg', frame[startY: endY, startX: endX])
104     img_list.append(f'send_images/{i}.jpeg')
105     cv2.imwrite(f'send_images/{i+1}.jpeg', frame)
106     img_list.append(f'send_images/{i+1}.jpeg')
107     return img_list
```

Use this list in the sendMail parameter.

The thing to notice is we will use Multi-threading concept:

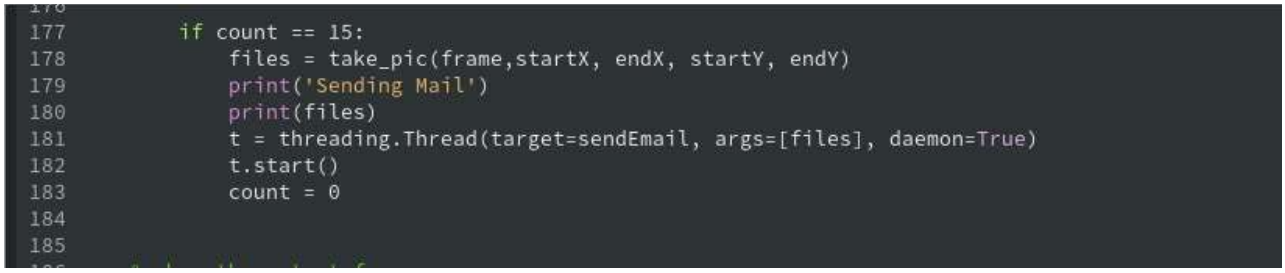
Running several threads is similar to running several different programs concurrently, but with the following benefits –

- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
- Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes.

A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is

currently running.

- It can be pre-empted (interrupted)
- It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.



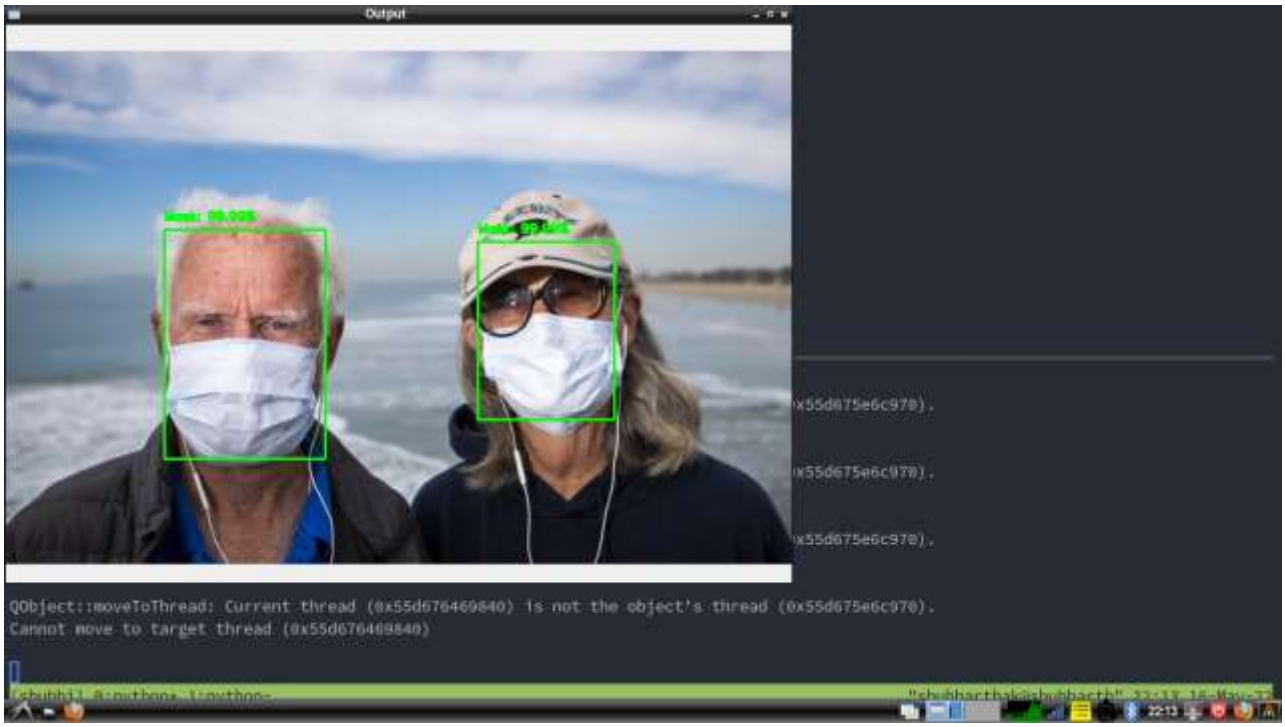
7. FINAL OUTPUT

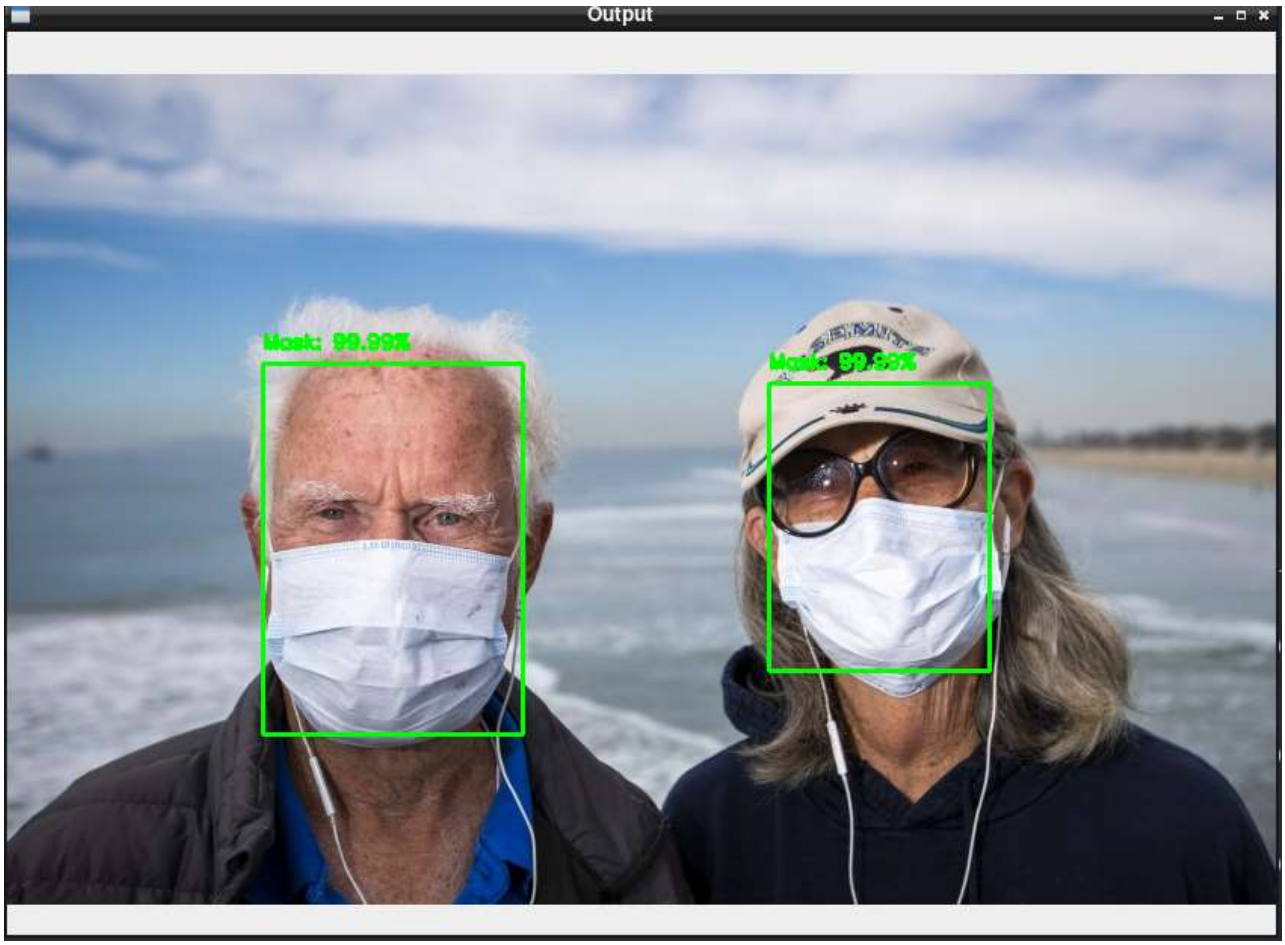
Checking Output Using Detect Image.py

Starting the detect_image.py using -i argument and telling the image location)

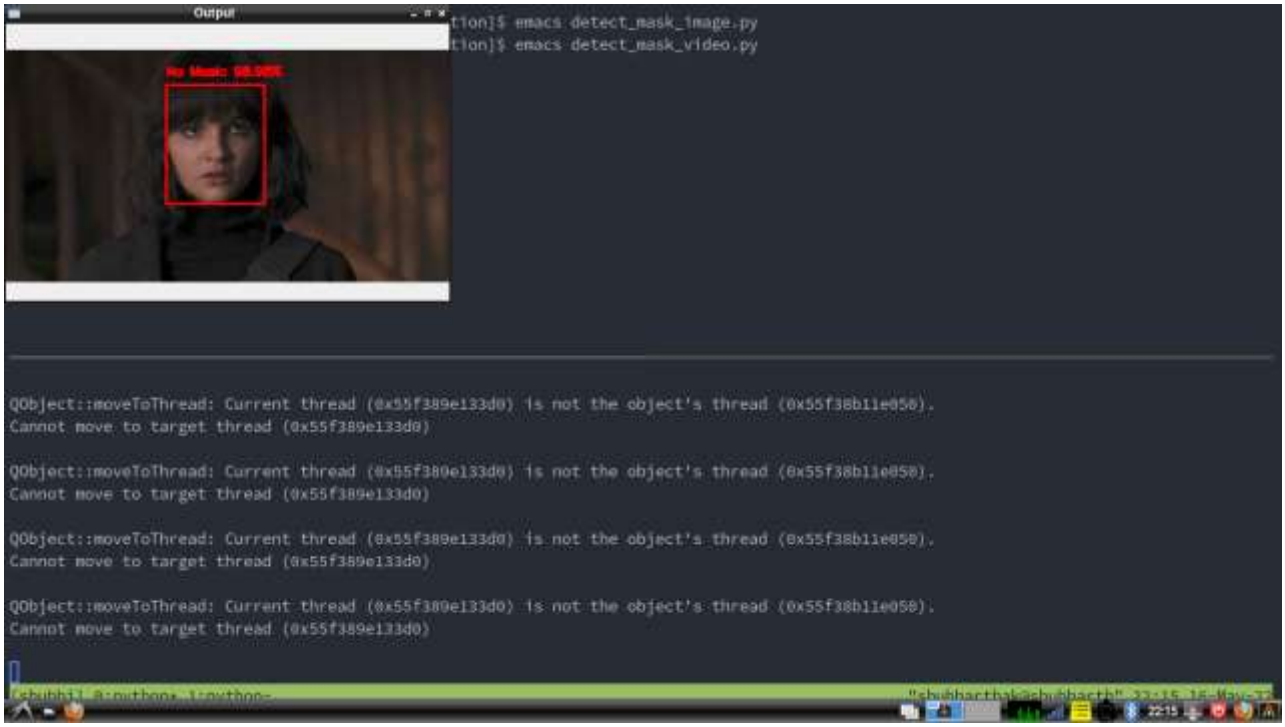
```
[shubharthak@shubharthak face_mask_detection]$ ls images/
out.jpg  pic1.jpeg  pic2.jpg  pic3.jpg
[shubharthak@shubharthak face_mask_detection]$ python detect_mask_image.py -i images/pic1.jpeg
```

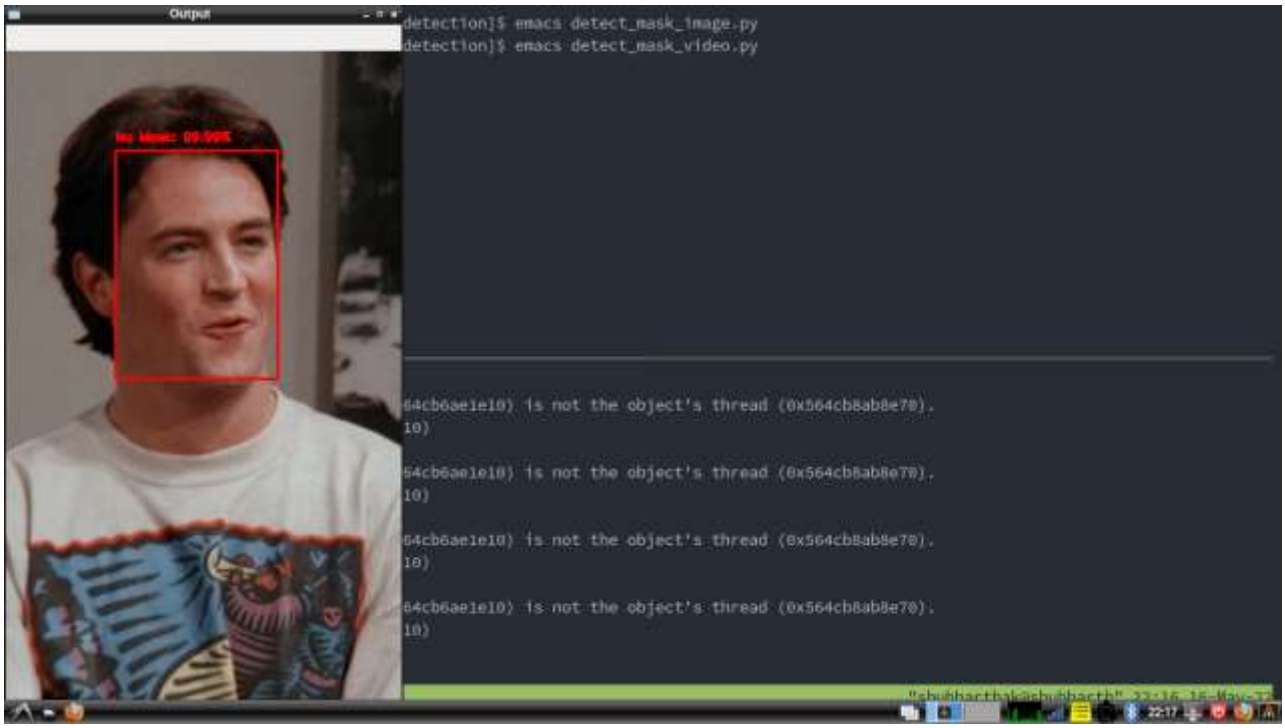
OUTPUT:-




Output
Mask: 99.99%
Mask: 99.98%


Output
(0x558361066960).
(0x558361066960).
Cannot move to target thread (0x5583627628f0)
QObject::moveToThread: Current thread (0x5583627628f0) is not the object's thread (0x558361066960).
Cannot move to target thread (0x5583627628f0)
QObject::moveToThread: Current thread (0x5583627628f0) is not the object's thread (0x558361066960).
Cannot move to target thread (0x5583627628f0)



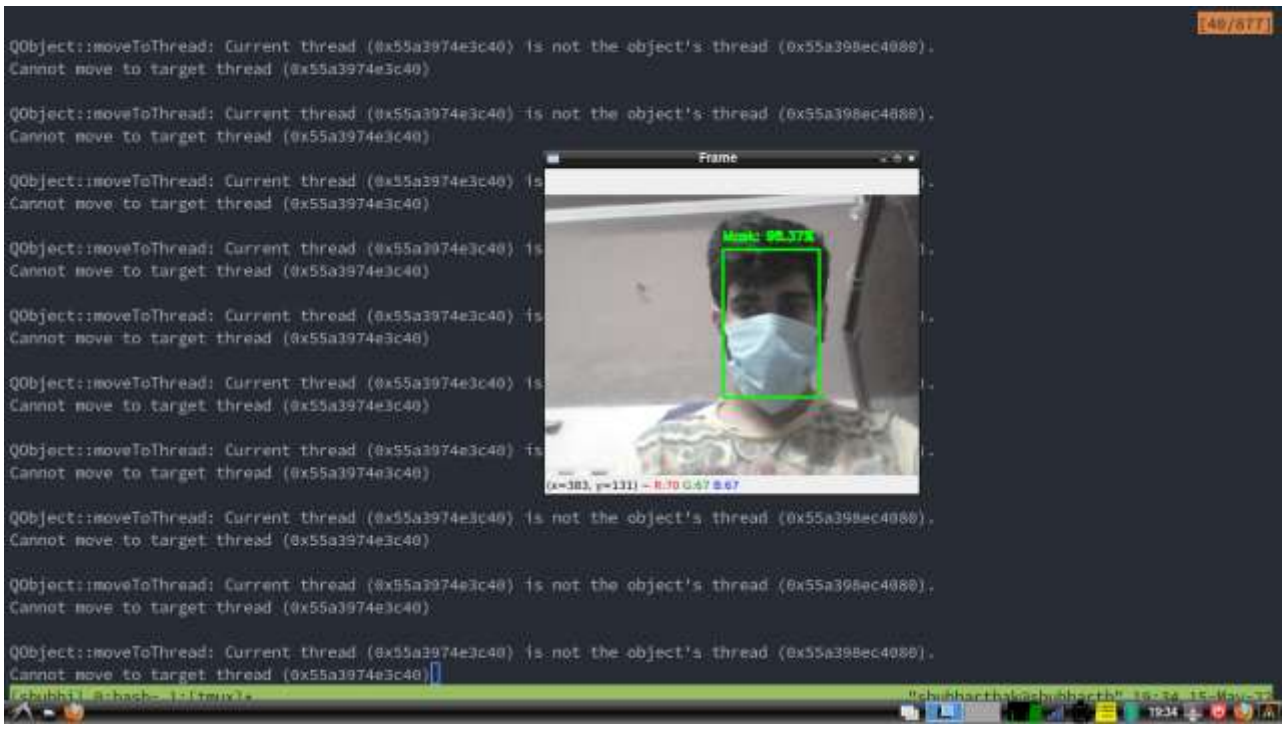


Checking Output Using Detect Video.py

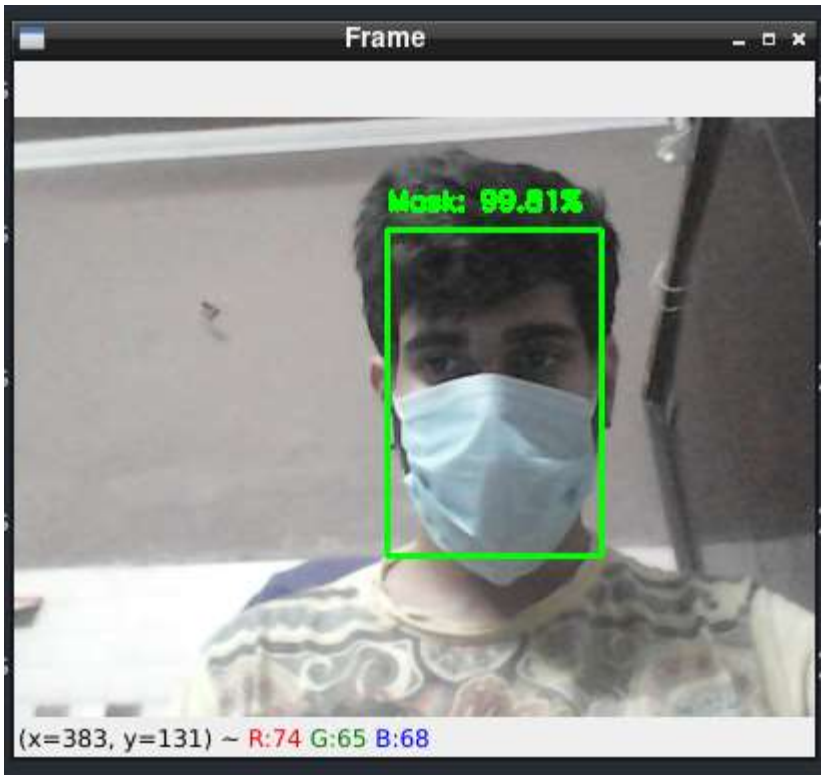
Launching the detect_video.py

```
(base) [shubharthak@shubharthak face_mask_detection]$ python detect_mask_video.py
High Performance MPEG 1.0/2.0/2.5 Audio Player for Layers 1, 2 and 3
        version 1.29.3; written and copyright by Michael Hipp and others
        free software (LGPL) without any warranty but with best wishes


Terminal control enabled, press 'h' for listing of keys and functions.

Playing MPEG stream 1 of 1: text.mp3 ...

MPEG 2.0 L III cbr32 24000 mono

[0:01] Decoding of text.mp3 finished.
High Performance MPEG 1.0/2.0/2.5 Audio Player for Layers 1, 2 and 3
        version 1.29.3; written and copyright by Michael Hipp and others
        free software (LGPL) without any warranty but with best wishes


Terminal control enabled, press 'h' for listing of keys and functions.

Playing MPEG stream 1 of 1: text.mp3 ...

MPEG 2.0 L III cbr32 24000 mono

[shubhi] 0:bash- 1:python*                                    "shubharthak@shubharth" 10:34 15-May-22
```

It will first welcome the user and tell him/her to wear your face mask



Next, it will launch the model, here it will detect your face and then detect the face mask. If user have put the face mask it will turn the ROI green



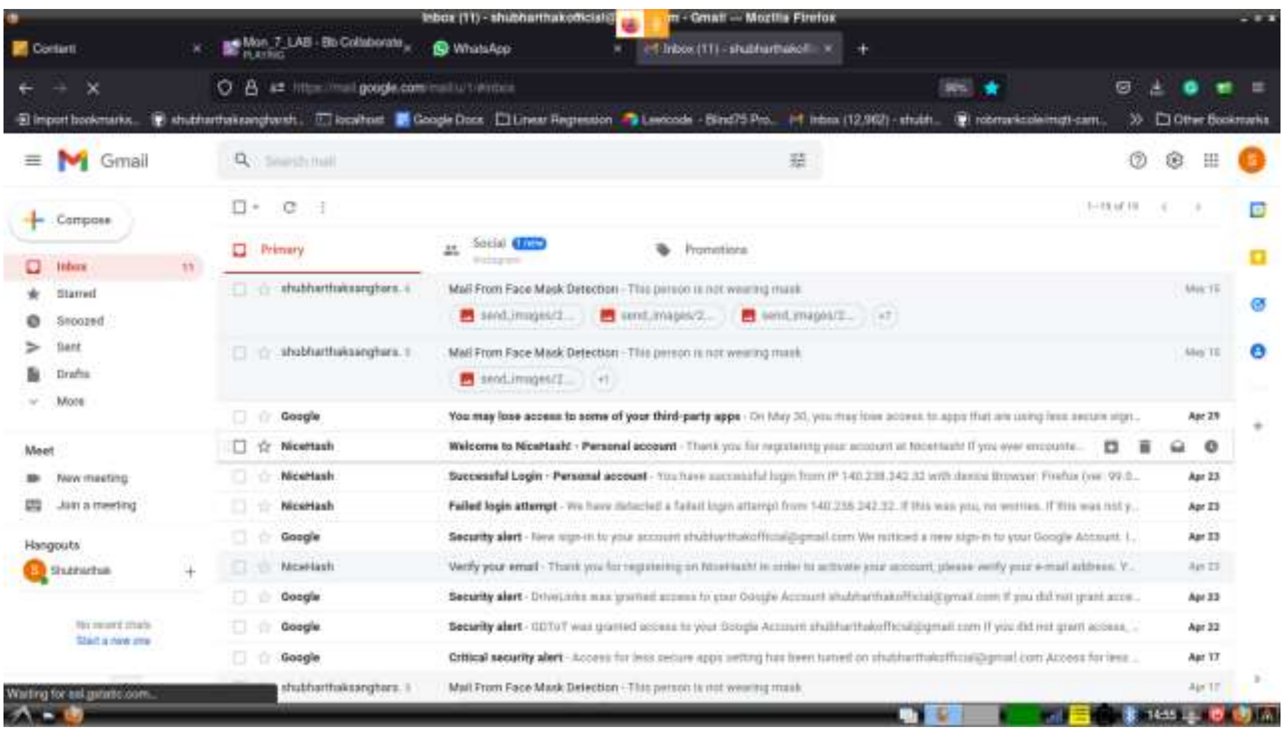
If the user didn’t put the face mask then it will turn the ROI as Red

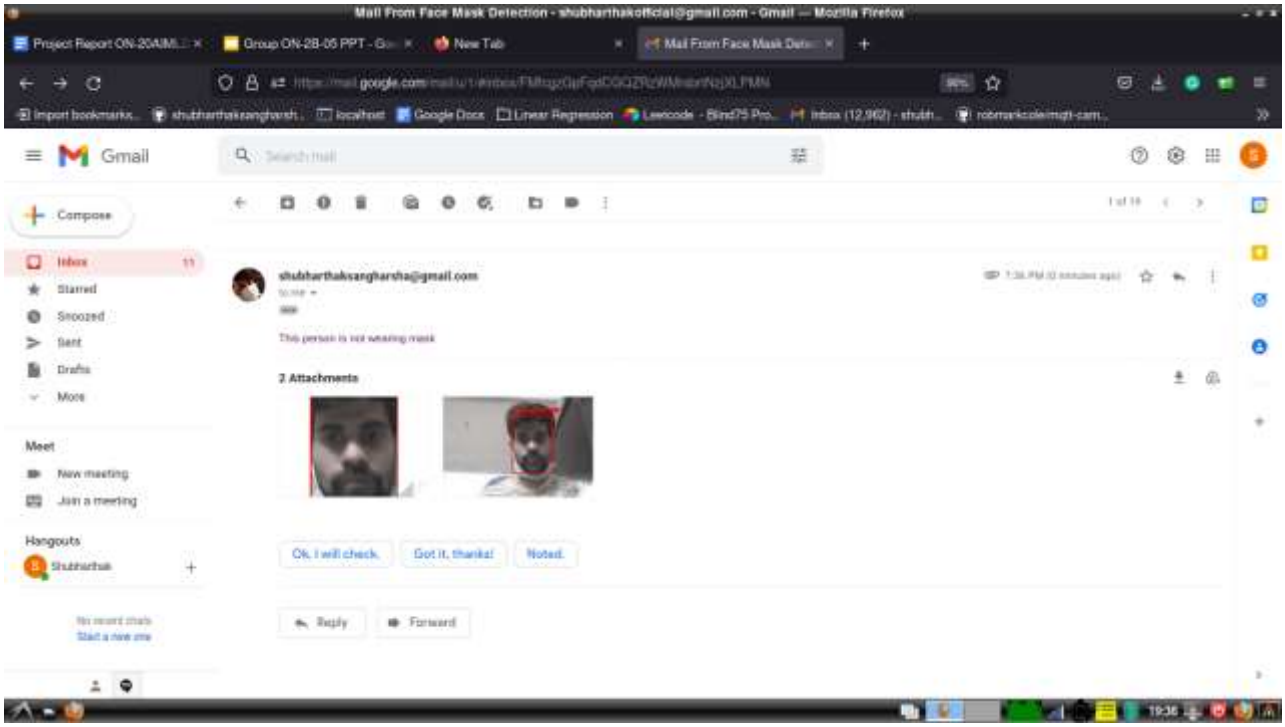
and also raise the alarm

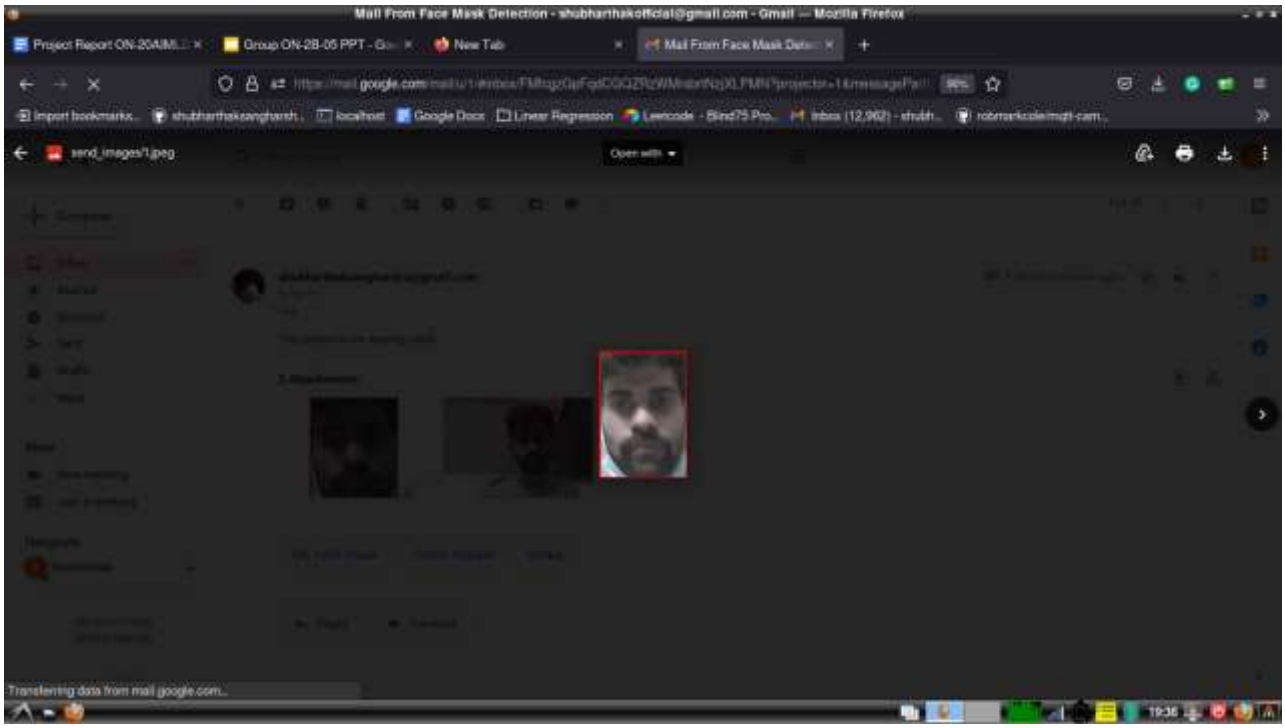
```
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'output4.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

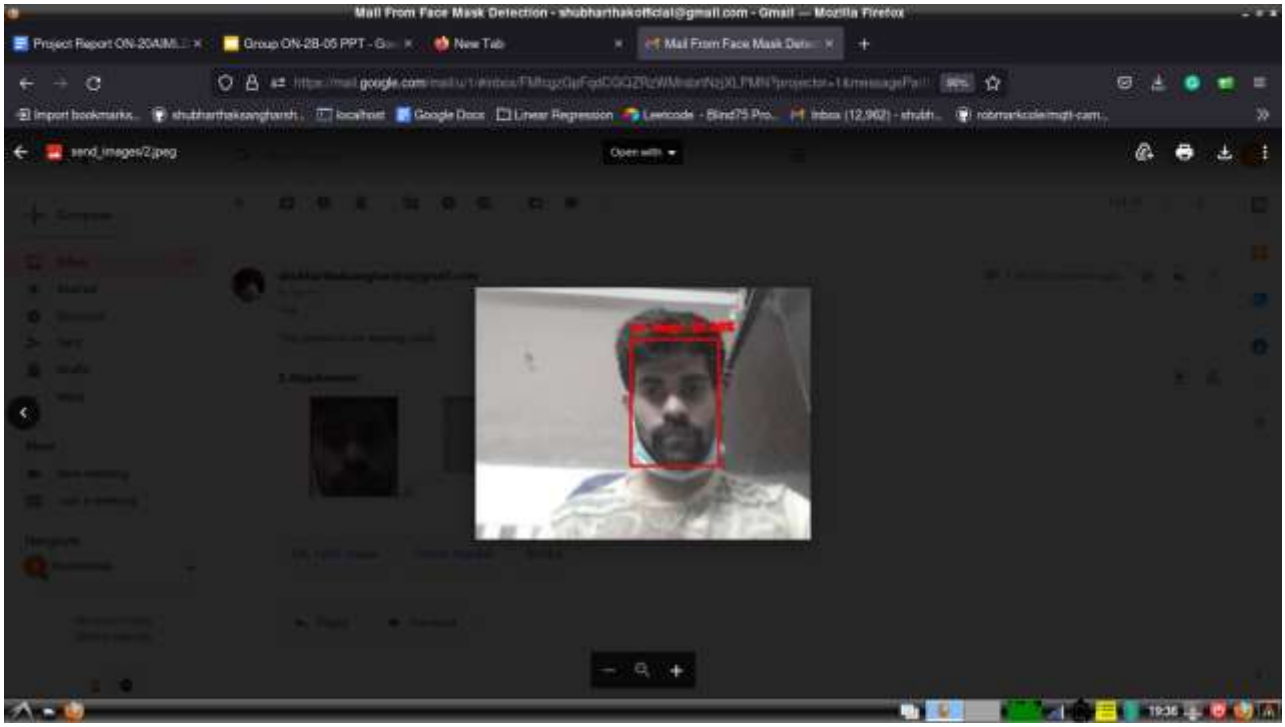
if mask enforcement is not present for 15seconds

then it will send mail to the owner's mail




Mail From Face Mask Detection - shubharthakofficial@gmail.com - Gmail — Mozilla Firefox
Gmail
Compose
Inbox
Starred
Snoozed
Sent
Drafts
More
Meet
New meeting
Join a meeting
Hangouts
shubharthaksangharsha@gmail.com
This person is not wearing mask
2 Attachments
Ok, I will check.
Got it, thanks!
Noted.
Reply
Forward


Mail From Face Mask Detection - shubharthakofficial@gmail.com - Gmail — Mozilla Firefox
send_images/1.jpeg
Open with
Transferring data from mail.google.com...


Mail From Face Mask Detection - shubharthakofficial@gmail.com - Gmail — Mozilla Firefox
send_images/2.jpeg
Open with

REFERENCES

[1] FACE MASK DETECTION USING AI AND IOT A PROJECT REPORT Submitted By BACHELOR OF ENGINEERING IN ELECTRONICS AND COMMUNICATION ENGINEERING INSTITUTE OF ROAD AND TRANSPORT TECHNOLOGY, ERODE ANNA UNIVERSITY : CHENNAI 600 025

[2] Real-Time Implementation of AI-Based Face Mask Detection and Social Distancing Measuring System for COVID-19 Prevention

[3] Face mask detection using deep learning: An approach to reduce risk of Coronavirus spread

[4] Detection of Face Mask using Convolutional Neural Network

[5] COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

[6] Facial Recognition System for People with and without Face Mask in Times of the COVID-19 Pandemic

[7] Face Mask Detection Market Research Report by Technology (Optical and eBeam), Component, Function, Application, Region (Americas, Asia-Pacific, and Europe, Middle East & Africa) - Global Forecast to 2027 - Cumulative Impact of COVID-19

[8] Face Mask Detection Market by Technology (Optical, and e-beam), Component (Hardware and Software), and Application (Airport, Hospital, and Others): Global Opportunity Analysis and Industry Forecast, 2020–2030

[9] The Face Mask Detection For Preventing the Spread of COVID-19 at Politeknik Negeri Batam

[10] Real-Time Face Mask Detection to Ensure COVID-19 Precautionary Measures in the Developing Countries

[11] A real time face mask detection system using convolutional neural network

[12] COVID-19 Face Mask Detection

[13] Validating the Correct Wearing of Protection Mask by Taking a Selfie: Design of a Mobile Application “CheckYourMask” to Limit the Spread of COVID-19

[14] Face Mask Detection System using Artificial Intelligence

[15] A novel DeepMaskNet model for face mask detection and masked facial recognition

[16] Our own research paper Detection of Face mask using transfer learning and OPENCV.

THANK YOU