“FACE MASK DETECTOR”

A Project Report

Submitted in partial fulfilment of the Requirements for the award of the Degree of

MASTER OF SCIENCE (INFORMATION TECHNOLOGY)

By

Ms. HALIM NAEEMA KAIZAR & **Ms. SARGUROH NEHA IRFAN KHAN**

ROLL NO. 4 & **22**

Under the esteemed guidance of

Prof. RAJDEEP CHAKROBORTY

DEPARTMENT OF INFORMATION TECHNOLOGY

SHANKAR NARAYAN COLLEGE

(Affiliated to University of Mumbai)

MUMBAI, 401107

MAHARASHTRA

2021-2022

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

One of the pleasant aspects of preparing a project report is opportunity to thank those who have contributed to make this project possible.

I would like to express my special thanks to my guardian teachers who supported me in completing my project.

I’m also thankful to my project partner & other unseen/seen hands which have given us direct & indirect help in completion of this project.

THANKING YOU

DECLARATION

We, **Ms. Naeema Kaizar Halim** and **Ms. Sarguroh Neha Irfan Khan** hereby declare that we ourselves have completed this project under the guidance of **Prof. Rajdeep Chakroborty**. We have designed the system and have done all the programming required.

It may require some modification in the future as per the user's requirements. From the practical implementation point of view, flexibility in the changes have been incorporated in the package.

-Halim Naeema Kaizar

-Sarguroh Neha Irfan Khan

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The ideology of the project is to create an application which can detect whether a person is wearing a face mask or not, in order to track the violations.

- Due to COVID-19, government of each country has strictly made a rule to wear mask whenever you go outside your house.
- Nowadays, people are just wearing mask for the sake of not paying fine to the police officer, regardless not caring about their health.
- This project basically keeps a track on people whether they're wearing mask or not, for our country's health security.
- It would track the person and pictures into red box, on not wearing mask, and encircle the person with green box, who wears the mask.
- There are pre-existing face mask dataset which is passed as training dataset, which shows the technology whether the mask worn by the person.

- This project is a computer vision system to automatically detect the violation of face mask wearing.

- It will also reduce manual work to check and keep a track on public health and safety guidelines.

Hence, this solution tracks the people with or without masks in a real-time scenario if there is a violation in the scene or in public places.

This can be used with the existing embedded camera infrastructure to enable these analytics which can be applied to various verticals, as well as in an office building or at airport terminals/gates.

INTRODUCTION

- The spread of COVID-19 has resulted in many global deaths since 2020. The spread of virus can be avoided by mitigating the effect of the virus in the environment, or preventing the virus transfer from person to person by practicing physical distance and wearing face masks.

- WHO recommended that wearing a face mask can significantly reduce transmission of the COVID-19 virus.

- Many sectors, like the construction industry has been affected, where unnecessary projects have been suspended or mitigated people's interaction.

- However, many infrastructure projects cannot be suspended due to their crucial role in people's life.

- Therefore, bridge maintenance, street widening, highway rehabilitation, and other essential infrastructure projects have been activated again to keep the transportation system's serviceability.

- Although infrastructure projects are activated, the safety of construction workers cannot be overlooked. Due to the high density

of workers in construction projects, there is a high risk of the infection spread in construction sites.

- Therefore, systematic safety monitoring in infrastructure projects that ensure maintaining the physical distance and wearing face masks can enhance construction workers' safety.

- In some cases, safety officers can be assigned to infrastructure projects to inspect workers to detect cases that either social distancing or face mask wearing is not satisfied.

- However, once there are so many workers on a construction site, it is difficult for the officers to determine hazardous situations.

- Also, assigning safety officers increases the number of people on-site, raising the chance of transmission even more, and putting workers and officers in a more dangerous situation.

- Recently, online video capturing in construction sites has become very common.

- The current system of online video capturing can be used for safety purposes. An automatic system that uses computer vision

techniques to capture real-time safety violations from online videos can enhance infrastructure project workers' safety.

- This study develops a model using Faster R-CNN to detect workers (in this example) who either don't wear a face mask. Once a safety violation occurs, the model highlights who violates the safety rules by a red box in the video.

- Data sets: A dataset is a structured collection of data generally associated with a unique body of work.

- For this project, Two datasets are collected with mask and without mask in the dataset folder:

Dataset 1:

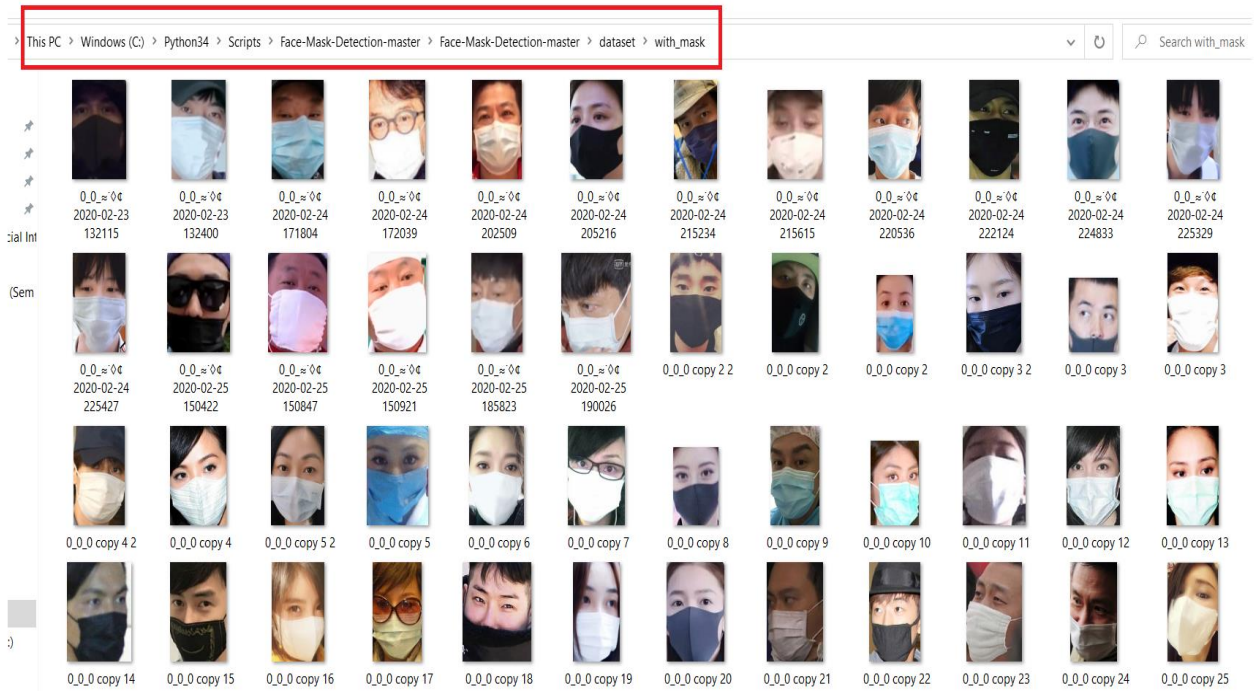


Fig. with_mask folder

Dataset 2:

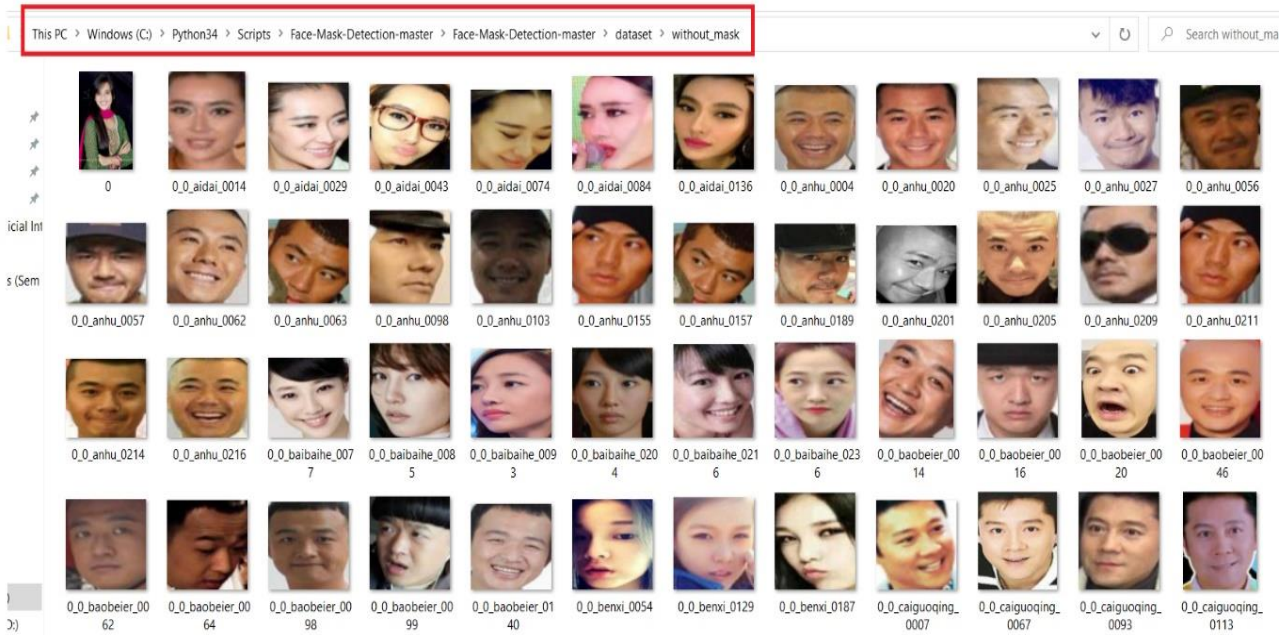
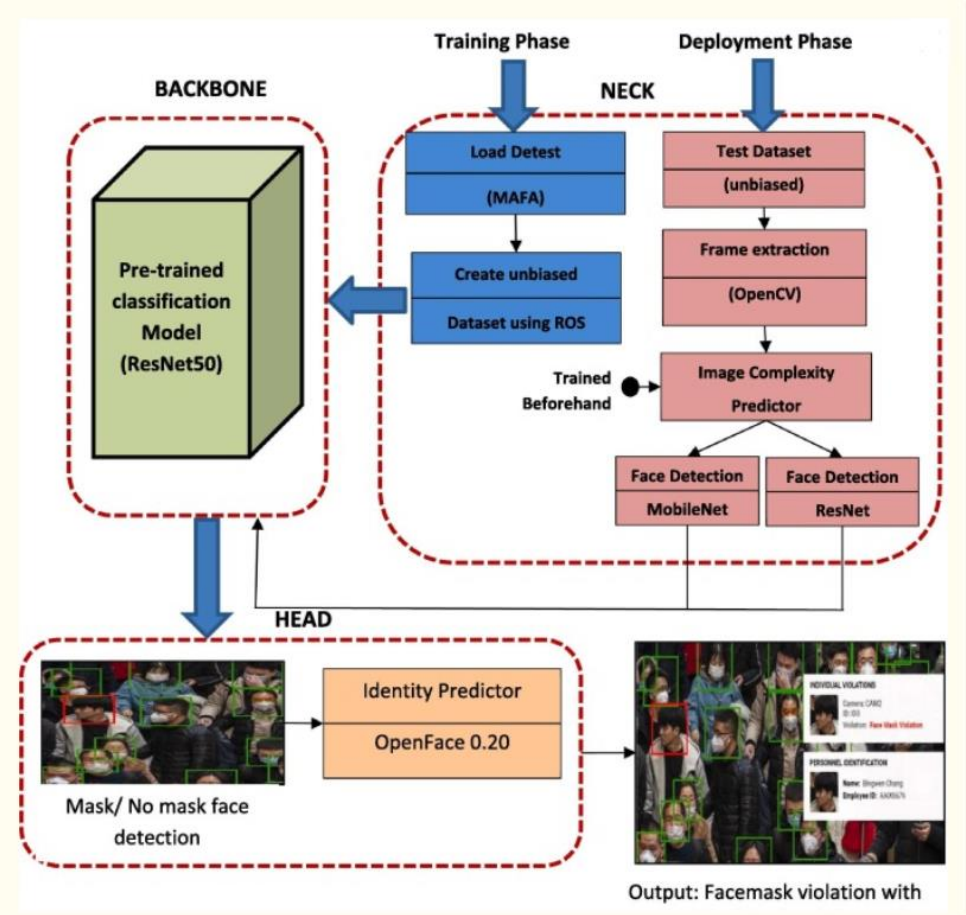


Fig. without_mask folder

- Below is the idea how it works:



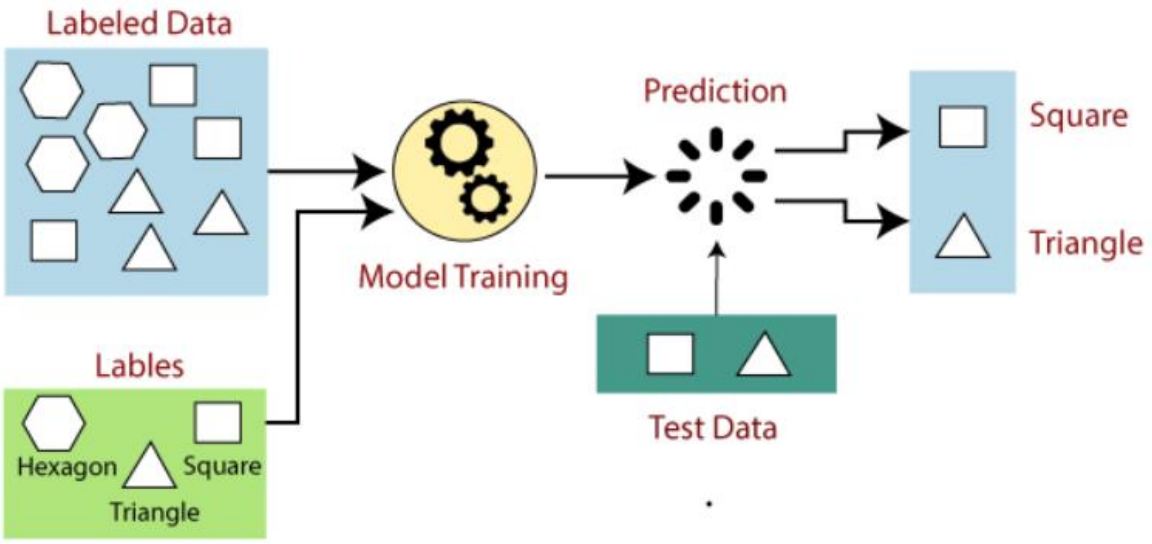
- Supervised learning (SL) is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.
- It is a kind of supervised learning whose output are predicted, that it can be either a red box or green. Red implies for not wearing the mask, whereas green implies for wearing the mask.
- Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.
- In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly.

- Supervised learning is a process of providing input data as well as correct output data to the machine learning model.
- The aim of a supervised learning algorithm is to find a mapping function to map the input variable(x) with the output variable(y).
- In the real-world, supervised learning can be used for Risk Assessment, Image classification, Fraud Detection, spam filtering**,** etc.

How Supervised Learning Works?

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram:



PROBLEM STATEMENT

PROBLEM STATEMENT:

After the outbreak of the Pandemic in 2020, The World Health Organization recommends wearing a face mask and practicing physical distancing to mitigate the virus's spread. In public places, this is a major security concern.

SOLUTION:

This project proposes a computer vision system to automatically detect the violation of face mask wearing to assure safety in public places during the pandemic.

This basic model can further be integrated in CCTV cameras to detect and identify people without masks.

The system can be used in any real-time applications which require face-mask detection for safety purposes.

LITERATURE SURVEY

- Object detection problems aim to locate and classify objects in an image. The face mask detection is classified as an object detection problem.

- Object detection is divided into two categories. One stage detection, such as You Only Look Once (YOLO), and two-stage detection, such as Region-Based Convolutional Neural Networks (R-CNN).

- Face mask detection identifies whether a person is wearing a mask or not in a picture. Since the beginning of COVID-19 pandemic, studies have been conducted to detect face masks in the crowd.

- Drones are used in construction projects to capture real-time video and provide aerial insights that help catch problems immediately.

- Research studies have collected large scale image data from drones and applied deep learning techniques to detect objects.
- YOLO is an abbreviation for the term ‘You Only Look Once’. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

- YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

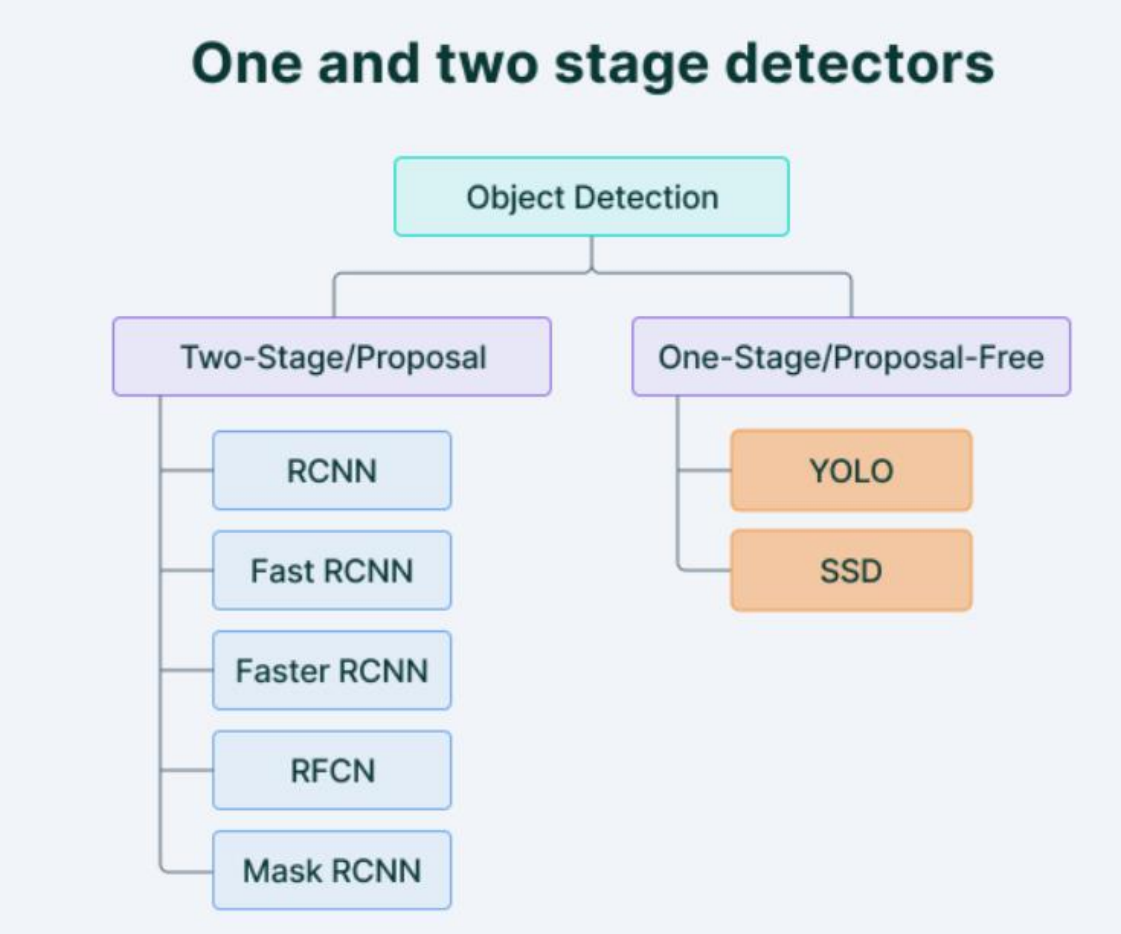
- This means that prediction in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

Two-stage object detection

Two-stage object detection refers to the use of algorithms that break down the object detection problem statement into the following two-stages:

1. Detecting possible object regions.
2. Classifying the image in those regions into object classes.

Popular two-step algorithms like Fast-RCNN and Faster-RCNN typically use a Region Proposal Network that proposes regions of interest that might contain objects.



Here's a timeline showcasing YOLO's development in recent years.

YOLOv2

YOLOv2 was proposed to fix YOLO's main issues—the detection of small objects in groups and the localization accuracy.

YOLOv2 increases the mean Average Precision of the network by introducing *batch normalization*.

YOLO9000

Using a similar network architecture as YOLOv2, YOLO9000 was proposed as an algorithm to detect more classes than COCO as an object detection dataset could have made possible.

The object detection dataset that these models were trained on (COCO) has only 80 classes as compared to classification networks like ImageNet which has 22.000 classes.

To enable the detection of many more classes, YOLO9000 makes use of labels from both ImageNet and COCO, effectively merging the classification and detection tasks to only perform detection.

YOLOv3

While YOLOv2 is a superfast network, various alternatives that offer better accuracies—like Single Shot Detectors—have also entered the scene. Although much slower, they outstrip YOLOv2 and YOLO9000 in terms of accuracy.

To improve YOLO with modern CNNs that make use of residual networks and skip connections, YOLOv3 was proposed.

YOLOv4, YOLOv5, YOLACT, and future YOLOs

Joseph Redmond left the AI community a few years back, so YOLOv4 and other versions past that are not his official work. Some of them are maintained by co-authors but none of the releases past YOLOv3 is considered the “official” YOLO.

However, the legacy continues through new researchers.

YOLO algorithm works using the following three techniques:

- Residual blocks
- Bounding box regression
- Intersection Over Union (IOU)

Residual blocks

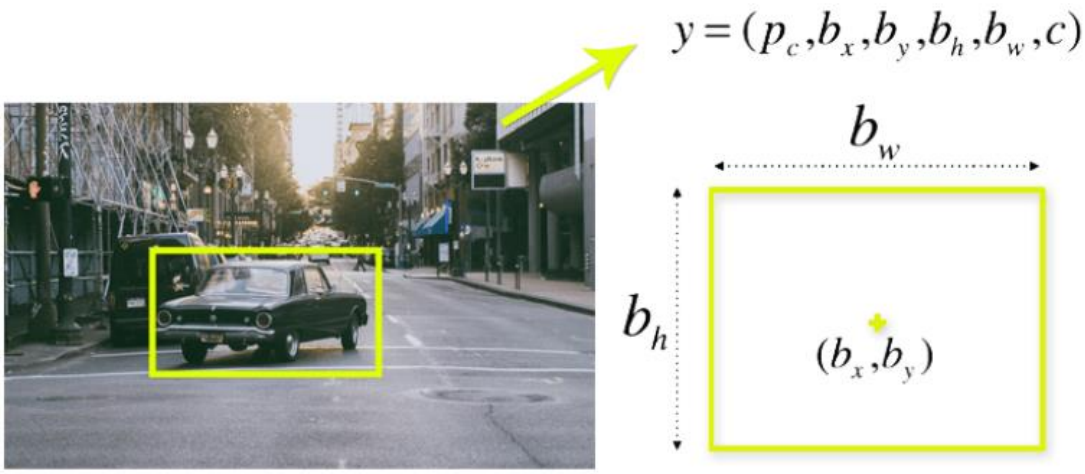
First, the image is divided into various grids. Each grid has a dimension of S x S. The following image shows how an input image is divided into grids.

Bounding box regression

A bounding box is an outline that highlights an object in an image.

Every bounding box in the image consists of the following attributes:

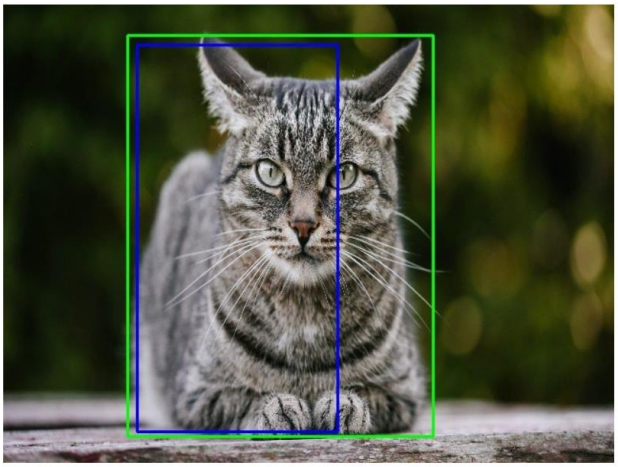
- Width (bw)
- Height (bh)
- Class (for example, person, car, traffic light, etc.)- This is represented by the letter
- Bounding box center (bx,by)



Intersection over union (IOU)

Intersection over union (IOU) is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly.

Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.



Challenges faced by YOLO:

1. Variable Number of Objects

Object Detection is the problem of locating and classifying a variable number of objects in an Image. The important thing is the **"variable"** part. The number of objects to be detected might vary from image to image. So the main problem associated with this is that in Machine Learning models, we usually need to represent the data in fixed-sized vectors.

2. Multiple Spatial Scales and Aspect Ratios

The Objects in the Images are of multiple spatial scales and aspect ratios, there may be some objects that cover most of the image and yet there will be some we may want to find but are as small as a dozen pixels (or a very small percentage of the Image). Even the same objects can have different Scales in different images. These varying dimensions of objects pose a difficulty in tracking them down.

3. Modelling

Doing object detection requires solving two approaches at once-Object Detection and Object Localization. Not only we want to classify the object but we also want to locate it inside the Image. To address these, most of the researches use multi-task loss functions to penalize both misclassification errors and localization errors. Due to this duality behavior of the loss function, many times it ends up performing poorly in both.

4. Limited Data

The limited amount of annotated data currently available for object detection is another hurdle in the process. Object detection datasets typically contain annotated examples for about dozen to a hundred classes while image classification datasets can include up to 100,000 classes. Gathering the ground truth labels along with the bounding boxes for each class is still a very tedious task to solve.

5. Speed for Real-Time detection

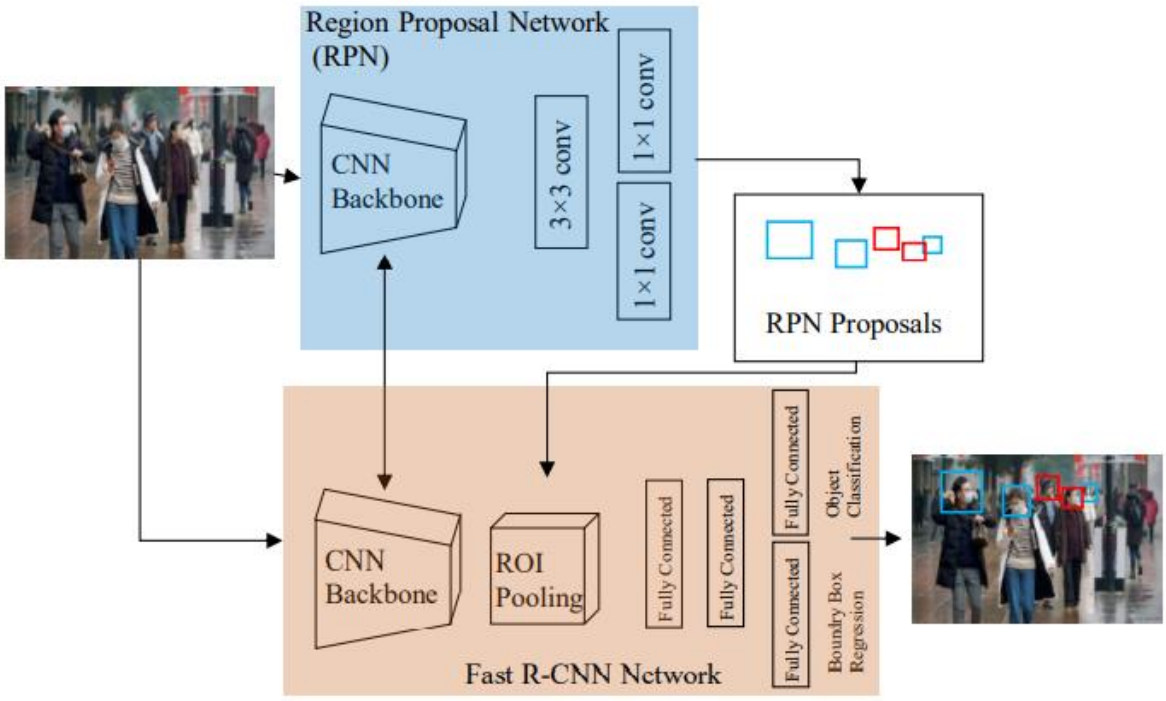
Object detection algorithms need to be not only accurate in predicting the class of the object along with its location, but it also needs to be incredibly fast in doing all these things to coup-up with the needs of the real-time demands of video processing. Usually, a video is shot at almost 24 fps and to build an algorithm that can achieve that frame rate is quite a difficult task.

R-CNN model:

- Below figure shows a schematic architecture of the Faster R-CNN model. The Faster R-CNN includes the Region Proposal Network (RPN) and the Fast R-CNN as the detector network. The input image is passed through the Convolutional Neural Networks (CNN) Backbone to extract the features.

- The RPN then suggests bounding boxes that are used in the Region of Interest (ROI) pooling layer to perform pooling on the image's features.

- Then, the network passes the output of the ROI pooling layer through two Fully Connected (FC) layers to provide the input of a pair of FC layers that one of them determines the class of each object and the other one performs a regression to improve the proposed boundary boxes.



Schematic diagram for Faster R-CNN

Single stage detector

- The single-stage detectors treat the detection of region proposals as a simple regression problem by taking the input image and learning the class probabilities and bounding box coordinates. YOLO (You Only Look Once) popularized single-stage approach by demonstrating real-time predictions and achieving remarkable detection speed but suffered from low localization accuracy when compared with two-stage detectors; especially when small objects are taken into consideration.

- Basically, the YOLO network divides an image into a grid of size G x G, and each grid generates N predictions for bounding boxes. Each bounding box is limited to have only one class during the prediction, which restricts the network from finding smaller objects. Further, YOLO network was improved to YOLO v2 that included batch normalization, high-resolution classifier and anchor boxes.

- Furthermore, the development of YOLO v3 is built upon YOLO v2 with the addition of an improved backbone classifier, multi-sale prediction and a new network for feature extraction. Although, YOLO v3 is executed faster than Single-Shot Detector (SSD) but does not perform well in terms of classification accuracy.

- Moreover, YOLO v3 requires a large amount of computational power for inference, making it not suitable for embedded or mobile

devices. Next, SSD networks have superior performance than YOLO due to small convolutional filters, multiple feature maps and prediction in multiple scales. The key difference between the two architectures is that YOLO utilizes two fully connected layers, whereas the SSD network uses convolutional layers of varying sizes.

Two-stage detector:

- In contrast to single-stage detectors, two-stage detectors follow a long line of reasoning in computer vision for the prediction and classification of region proposals. They first predict proposals in an image and then apply a classifier to these regions to classify potential detection.
- Various two-stage region proposal models have been proposed in past by researchers. Region-based convolutional neural network also abbreviated as R-CNN described in 2014 by Ross Girshick et al. It may have been one of the first large-scale applications of CNN to the problem of object localization and recognition.

- The model was successfully demonstrated on benchmark datasets such as VOC-2012 and ILSVRC-2013 and produced state of art results. Basically, R-CNN applies a selective search algorithm to extract a set of object proposals at an initial stage and applies SVM (Support Vector Machine) classifier for predicting objects and

related classes at later stage. Spatial pyramid pooling SPPNet (modifies R-CNN with an SPP layer) collects features from various region proposals and fed into a fully connected layer for classification.

- The capability of SPNN to compute feature maps of the entire image in a single-shot resulted in significant improvement in object detection speed by the magnitude of nearly 20 folds greater than R-CNN. Next, Fast R-CNN is an extension over R-CNN and SPPNet . It introduces a new layer named Region of Interest (RoI) pooling layer between shared convolutional layers to fine-tune the model.

- Moreover, it allows to simultaneously train a detector and regressor without altering the network configurations. Although Fast-R-CNN effectively integrates the benefits of R-CNN and SPPNet but still lacks in detection speed compared to single-stage detectors.
- Further, Faster R-CNN is an amalgam of fast R-CNN and Region Proposal Network (RPN). It enables nearly cost-free region proposals by gradually integrating individual blocks (e.g. proposal detection, feature extraction and bounding box regression) of the object detection system in a single step. Although this integration leads to the accomplishment of break-through for the speed bottleneck of Fast R-CNN but there exists a computation redundancy at the subsequent detection stage.

- The Region-based Fully Convolutional Network (R-FCN) is the only model that allows complete backpropagation for training and inference. Feature Pyramid Networks (FPN) can detect non-uniform objects, but least used by researchers due to high computation cost and more memory usage.
- Furthermore, Mask R-CNN strengthens Faster R-CNN by including the prediction of segmented masks on each RoI. Although two-stage yields high object detection accuracy, but it is limited by low inference speed in real-time for video surveillance.

OBJECTIVE

The main objective of this application is to design a system which can be incorporated in any CCTV and security cameras at public locations/ infrastructure project sites etc. where regular security checks need to be conducted manually to make sure all people entering that location or present at that location are following the guidelines.

Our application has the ability to identify whether or not someone is wearing a mask. It can detect a violation to the guidelines.

Using this application will prove helpful for automatic detection of the manual security task.

NEED & SCOPE OF THE STUDY

This study is essential for ensuring that proper guidelines are followed in crowded places, in the light of the pandemic waves.

In this work, a deep learning-based approach for detecting masks over faces in public places to curtail the community spread of Coronavirus is presented.

The proposed technique efficiently handles occlusions in dense situations by making use of an ensemble of single and two-stage detectors at the pre-processing level.

The ensemble approach not only helps in achieving high accuracy but also improves detection speed considerably. Furthermore, the application of transfer learning on pre-trained models with extensive experimentation over an unbiased dataset resulted in a highly robust and low-cost system.

The identity detection of faces, violating the mask norms further, increases the utility of the system for public benefits.

Methodologies used in the project:

- This project obtains a facemask dataset available on the internet and increases the amount of data by adding more images.

- Improved affine transformation is developed to crop the facial areas from uncontrolled real-time images having differences in face size, orientation and background.
- This step helps in better localizing the person who is violating the facemask norms in public areas/ offices.
- Creation of unbiased facemask dataset with imbalance ratio equals to nearly one.
- The proposed model requires less memory, making it easily deployable for embedded devices used for surveillance purposes.
- Pattern learning and object recognition are the inherent tasks that a computer vision (CV) technique must deal with.
- Object recognition encompasses both image classification and object detection. The task of recognizing the mask over the face in the pubic area can be achieved by deploying an efficient object recognition algorithm through surveillance devices.
- The object recognition pipeline consists of generating the region proposals followed by classification of each proposal into related class.

- Then, it trains multiple Faster R-CNN object detection models to choose the most accurate model for face mask detection.

- As per available literature, very little body of research is attempted to detect mask over face. Thus, our work aims to a develop technique that can accurately detect mask over the face in public areas (such as airports. railway stations, crowded markets, bus stops, etc.) to curtail the spread of Coronavirus and thereby contributing to public healthcare.

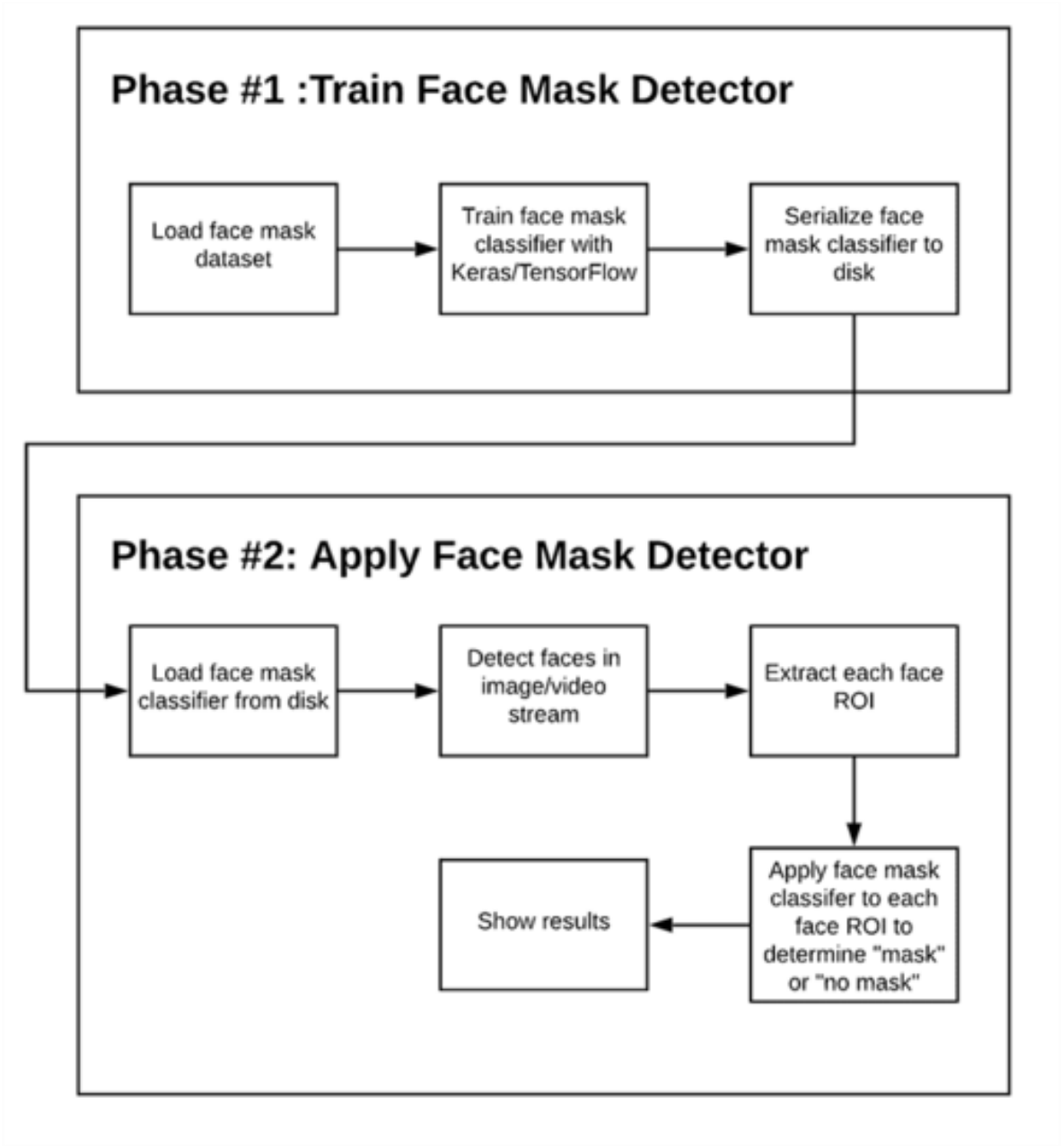
- Further, it is not easy to detect faces with/without a mask in public as the dataset available for detecting masks on human faces is relatively small leading to the hard training of the model. So, the concept of transfer learning is used here to transfer the learned kernels from networks trained for a similar face detection task on an extensive dataset.

- The dataset covers various face images including faces with masks, faces without masks, faces with and without masks in one image and confusing images without masks.

PROJECT WORKFLOW

VISUAL REPRESENTATION OF THE WORKFLOW:

Two-phase COVID-19 face mask detector



The above diagram depicts the phases and individual steps for building a COVID-19 face mask detector with computer vision and deep learning using Python libraries like OpenCV, TensorFlow and Keras, which has been used in this project.

The proposed model is composed of deep learning CNN Algorithm. The working of these algorithms is explained in the previous section.

This project uses Python Machine Libraries- Keras, TensorFlow, OpenCV and MobileNetV2.

Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models.

It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code.

TensorFlow is a Python library for fast numerical computing created and released by Google.

It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.

OpenCV is a great tool for image processing and performing computer vision tasks.

It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more

MobileNetV2 is a very effective feature extractor for object detection and segmentation.

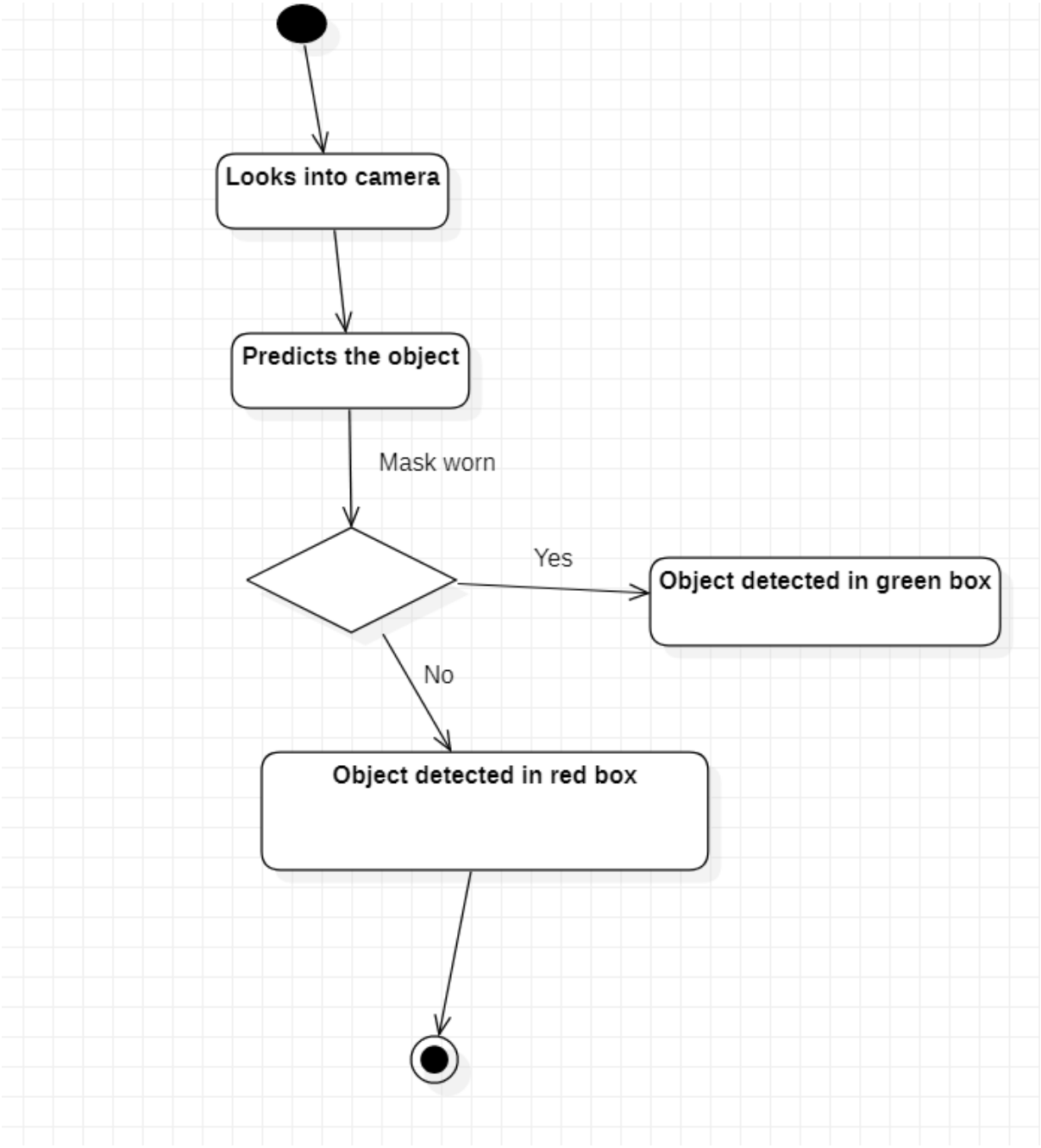
.MobileNetV2 is a convolutional neural network architecture that seeks to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers.

Following are the advantages of using MobileNet over other state-of-the-art deep learning models.

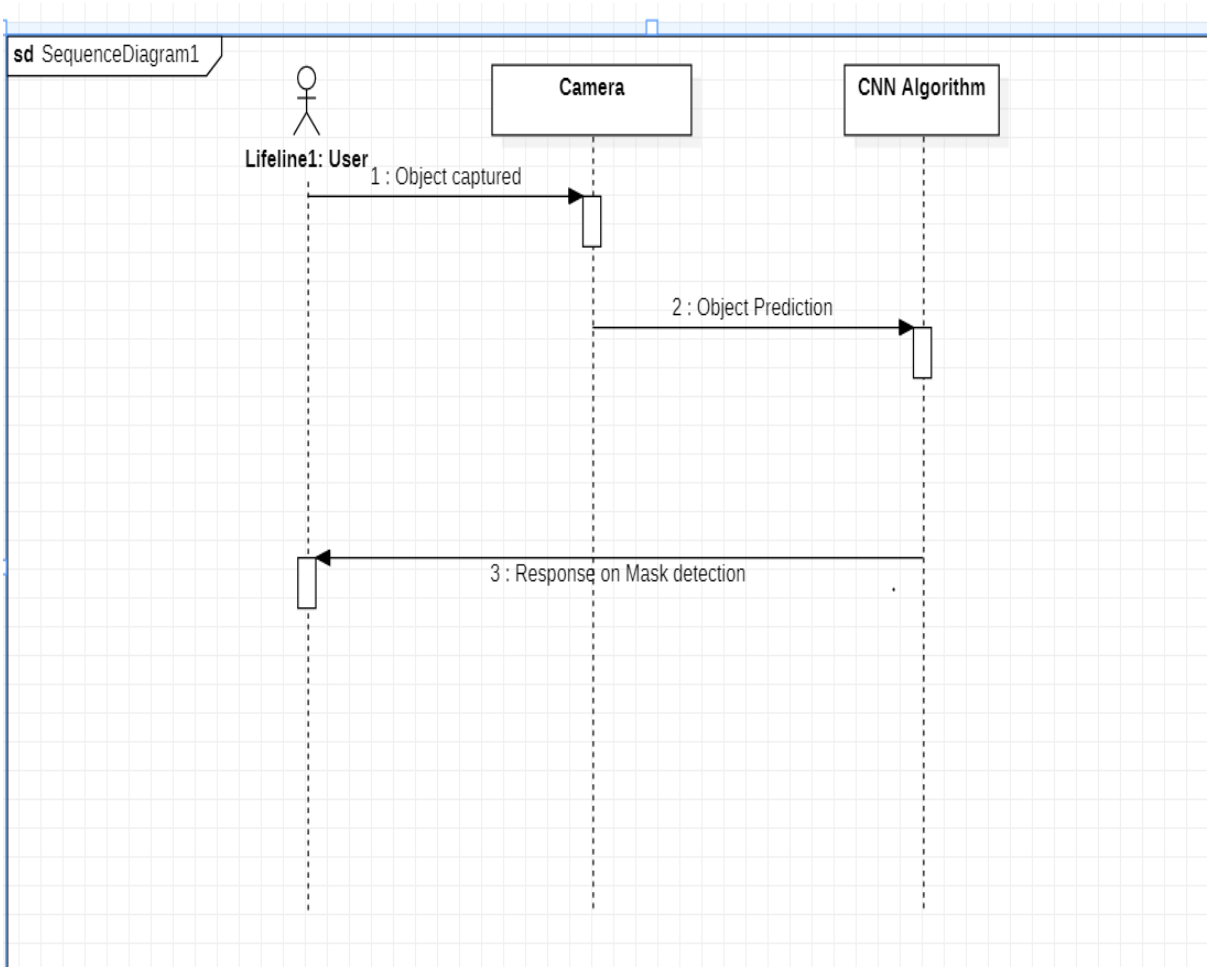
- Reduced network size
- Reduced number of parameters
- Faster in performance and are useful for mobile applications.

UML DIAGRAM

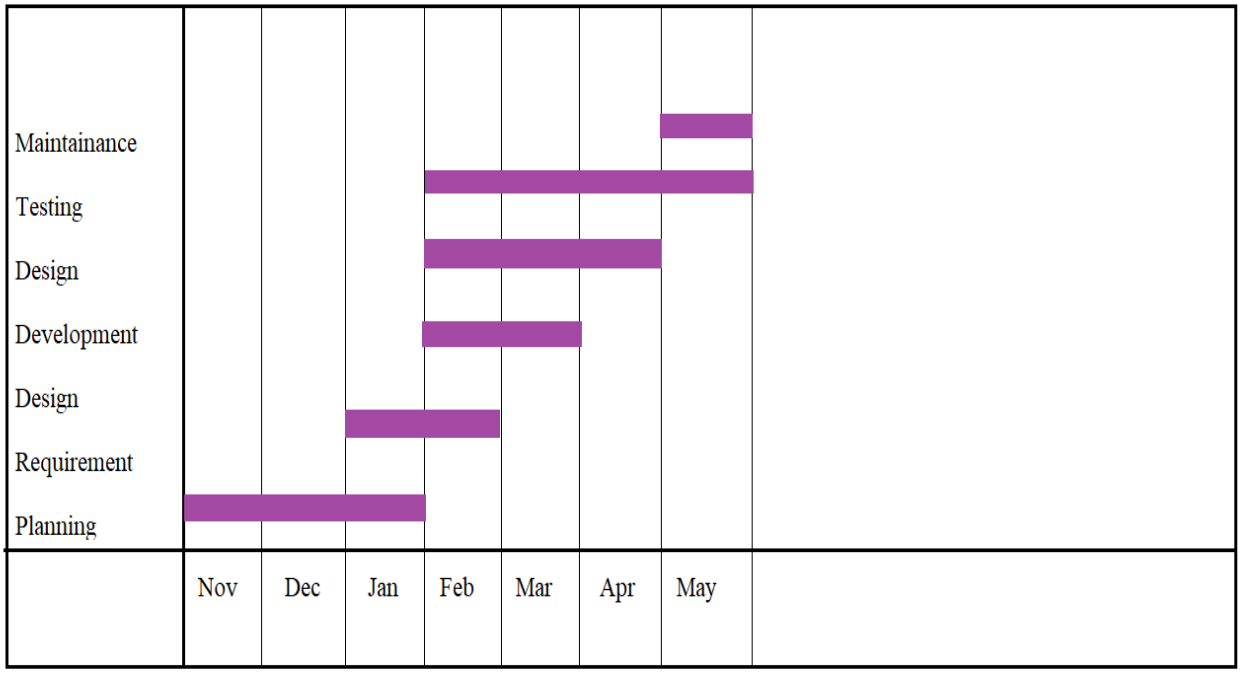
Activity Diagram:



Sequence Diagram:



GANTT CHART

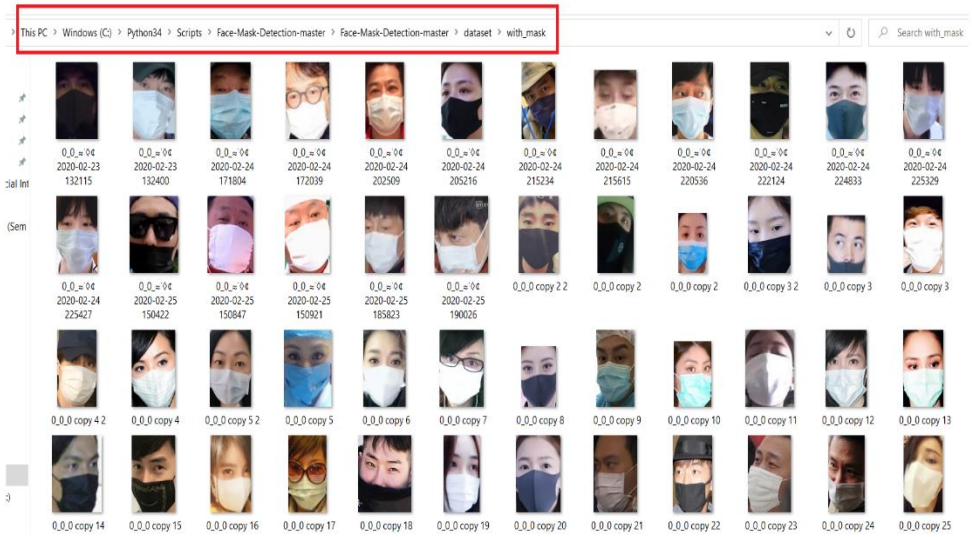


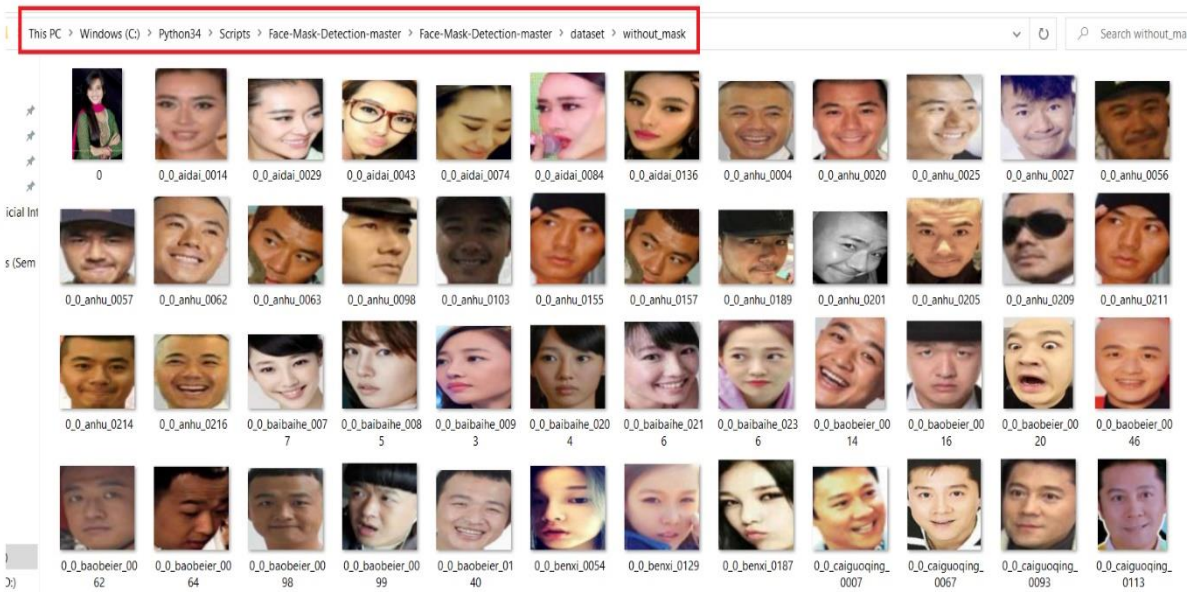
1. Planning:

- Planning should clearly define the scope and purpose of the application.
- It also sets boundaries to help keep the project from expanding or shifting from its original purpose.

2. Requirement:

Datasets are collected so that all possible objects can be interpreted, on applied CNN algorithm.





Requirements also include defining the resources needed to build the project.

3. Design:

The Design phase models the way a software application will work. Some aspects of the design include:

Architecture – Specifies programming language, industry practices, overall design, and use of any templates or boilerplate.
User Interface – Defines the ways customers interact with the software, and how the software responds to input
Platforms – Defines the platforms on which the software will run, such as Apple, Android, Windows version, Linux, or even gaming consoles. Windows is used
Programming – Not just the programming language, but including methods of solving problems and performing tasks in the application. Python language is used.

4. Development:

- The coding process includes many other tasks. Many developers need to brush up on skills or work as a team.
- Finding and fixing errors and glitches is critical. Tasks often hold up the development process, such as waiting for test results or compiling code so an

application can run. SDLC can anticipate these delays so that developers can be tasked with other duties.

- Software developers appreciate instructions and explanations. Documentation can be a formal process, including wiring a user guide for the application.
- It can also be informal, like comments in the source code that explain why a developer used a certain procedure.
- Even companies that strive to create software that's easy and intuitive benefit from the documentation. Documentation can be a quick guided tour of the application's basic features that display on the first launch.

5. Testing:

- Testing should ensure that each function works correctly.
- Different parts of the application should also be tested to work seamlessly together—performance test, to reduce any hangs or lags in processing. The testing phase helps reduce the number of bugs and glitches that users encounter.
- This leads to a higher user satisfaction and a better usage rate.

CODE:

Code of pre-processing file: **detect_mask_video.py**

```
# 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 os


# initialize the initial learning rate, number of epochs to train
for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32


DIRECTORY = r"C:\Users\Naeema_laptop\Downloads\Face-
Mask-Detection-master\Face-Mask-Detection-master\dataset"
```

```
CATEGORIES = ["with_mask", "without_mask"]


# grab the list of images in our dataset directory, then
initialize

# the list of data (i.e., images) and class images

print("[INFO] loading images...")


data = []

labels = []


for category in CATEGORIES:

    path = os.path.join(DIRECTORY, category)

    for img in os.listdir(path):

        img_path = os.path.join(path, img)

        image = load_img(img_path, target_size=(224, 224))

        image = img_to_array(image)

        image = preprocess_input(image)


        data.append(image)
```

```
        labels.append(category)


# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)


data = np.array(data, dtype="float32")
labels = np.array(labels)


(trainX, testX, trainY, testY) = train_test_split(data, labels,
        test_size=0.20, stratify=labels, random_state=42)




# construct the training image generator for data augmentation
aug = ImageDataGenerator(
        rotation_range=20,
```

```
        zoom_range=0.15,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.15,
        horizontal_flip=True,
        fill_mode="nearest")

# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
        input_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
```

```
headModel = AveragePooling2D(pool_size=(7,
7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)

# place the head FC model on top of the base model (this will
become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so
they will
# *not* be updated during the first training process
for layer in baseModel.layers:
        layer.trainable = False

# compile our model
```

```
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
	metrics=["accuracy"])

# train the head of the network
print("[INFO] training head...")
H = model.fit(
	aug.flow(trainX, trainY, batch_size=BS),
	steps_per_epoch=len(trainX) // BS,
	validation_data=(testX, testY),
	validation_steps=len(testX) // BS,
	epochs=EPOCHS)

# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)
```

```
# for each image in the testing set we need to find the index of
the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
        target_names=lb.classes_))

# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save("mask_detector.model", save_format="h5")

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
```

```
plt.plot(np.arange(0, N), H.history["val_loss"],
label="val_loss")

plt.plot(np.arange(0, N), H.history["accuracy"],
label="train_acc")

plt.plot(np.arange(0, N), H.history["val_accuracy"],
label="val_acc")

plt.title("Training Loss and Accuracy")

plt.xlabel("Epoch #")

plt.ylabel("Loss/Accuracy")

plt.legend(loc="lower left")

plt.savefig("plot.png")
```

Code of the main file: **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 numpy as np
import imutils
import time
import cv2
import os


def detect_and_predict_mask(frame, faceNet, maskNet):
	# grab the dimensions of the frame and then construct a blob
	# from it
	(h, w) = frame.shape[:2]
	blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
		(104.0, 177.0, 123.0))
```

```
	# pass the blob through the network and obtain the face
detections
	faceNet.setInput(blob)
	detections = faceNet.forward()
	print(detections.shape)

	# initialize our list of faces, their corresponding locations,
	# and the list of predictions from our face mask network
	faces = []
	locs = []
	preds = []

	# loop over the detections
	for i in range(0, detections.shape[2]):
		# extract the confidence (i.e., probability) associated with
		# the detection
		confidence = detections[0, 0, i, 2]

		# filter out weak detections by ensuring the confidence is
```

```
            # greater than the minimum confidence
            if confidence > 0.5:
                # compute the (x, y)-coordinates of the bounding box
for
                # the object
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")

                # ensure the bounding boxes fall within the
dimensions of
                # the frame
                (startX, startY) = (max(0, startX), max(0, startY))
                (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

                # extract the face ROI, convert it from BGR to RGB
channel
                # ordering, resize it to 224x224, and preprocess it
                face = frame[startY:endY, startX:endX]
                face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                face = cv2.resize(face, (224, 224))
                face = img_to_array(face)
```

```
                face = preprocess_input(face)

                # add the face and bounding boxes to their respective
                # lists
                faces.append(face)
                locs.append((startX, startY, endX, endY))

        # only make a predictions if at least one face was detected
        if len(faces) > 0:
                # for faster inference we'll make batch predictions on *all*
                # faces at the same time rather than one-by-one predictions
                # in the above `for` loop
                faces = np.array(faces, dtype="float32")
                preds = maskNet.predict(faces, batch_size=32)

# return a 2-tuple of the face locations and their corresponding
        # locations
        return (locs, preds)
```

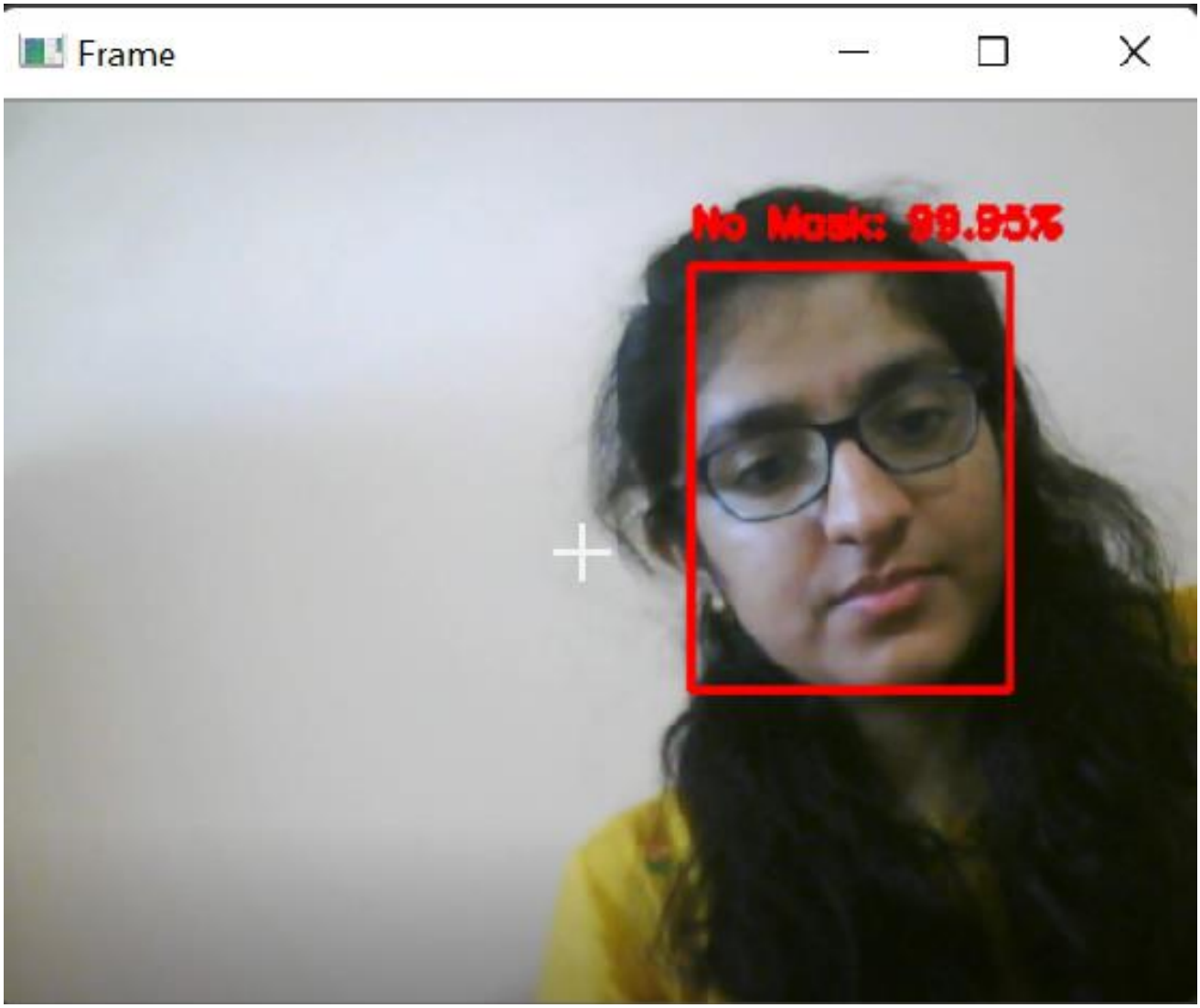
```
# load our serialized face detector model from disk
prototxtPath = r"face_detector\deploy.prototxt"
weightsPath =
r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
maskNet = load_model("mask_detector.model")

# initialize the video stream
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()

# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)
```

```
	# detect faces in the frame and determine if they are wearing a
	# face mask or not
	(locs, preds) = detect_and_predict_mask(frame, faceNet,
maskNet)

	# loop over the detected face locations and their corresponding
	# locations
	for (box, pred) in zip(locs, preds):
		# unpack the bounding box and predictions
		(startX, startY, endX, endY) = box
		(mask, withoutMask) = pred

		# 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(frame, label, (startX, startY - 10),
                  cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
            cv2.rectangle(frame, (startX, startY), (endX, endY), color,
2)

      # show the output frame
      cv2.imshow("Frame", frame)
      key = cv2.waitKey(1) & 0xFF

      # if the `q` key was pressed, break from the loop
      if key == ord("q"):
            break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

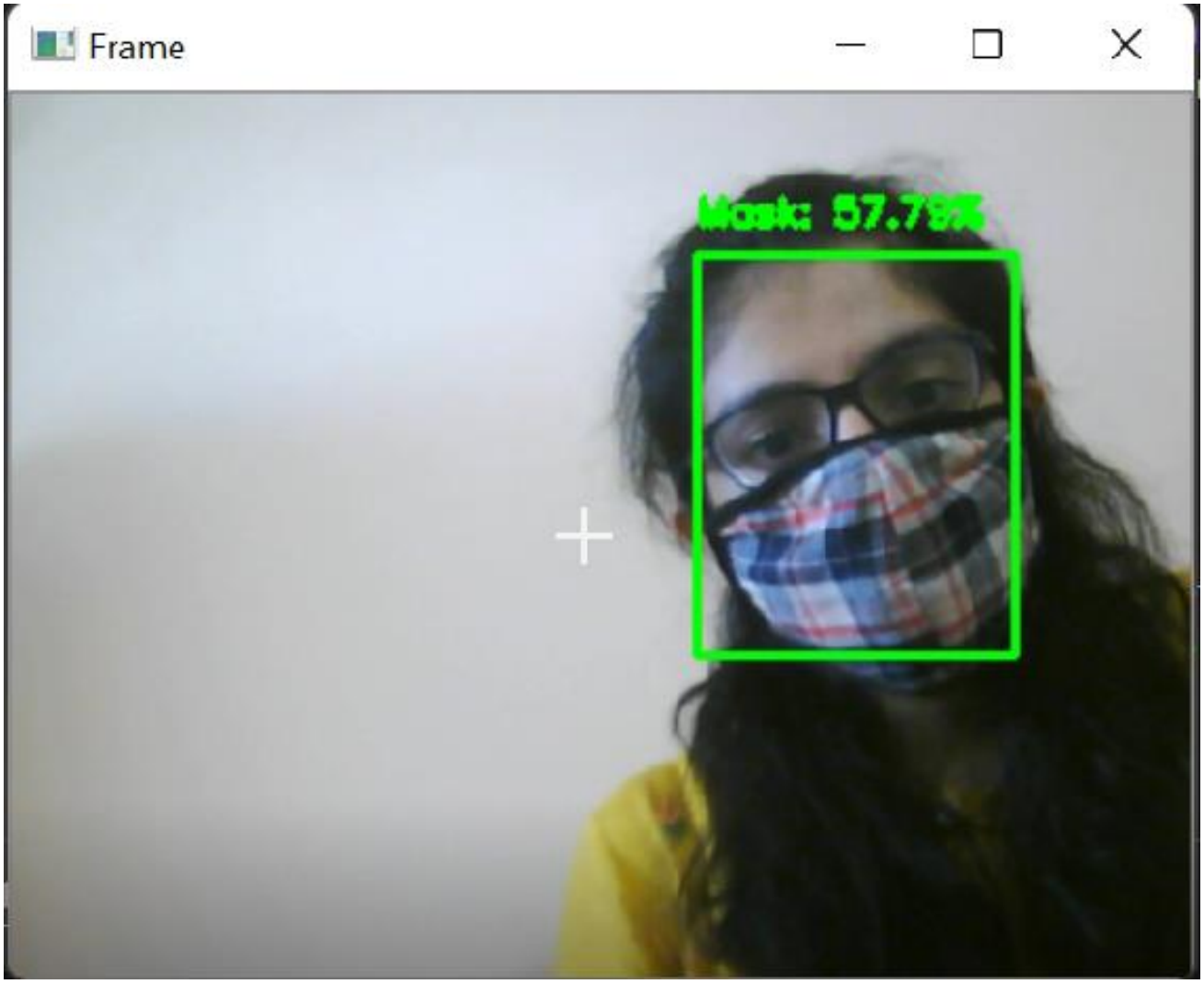
SCREENSHOTS OF PROJECT:

1) WITHOUT MASK

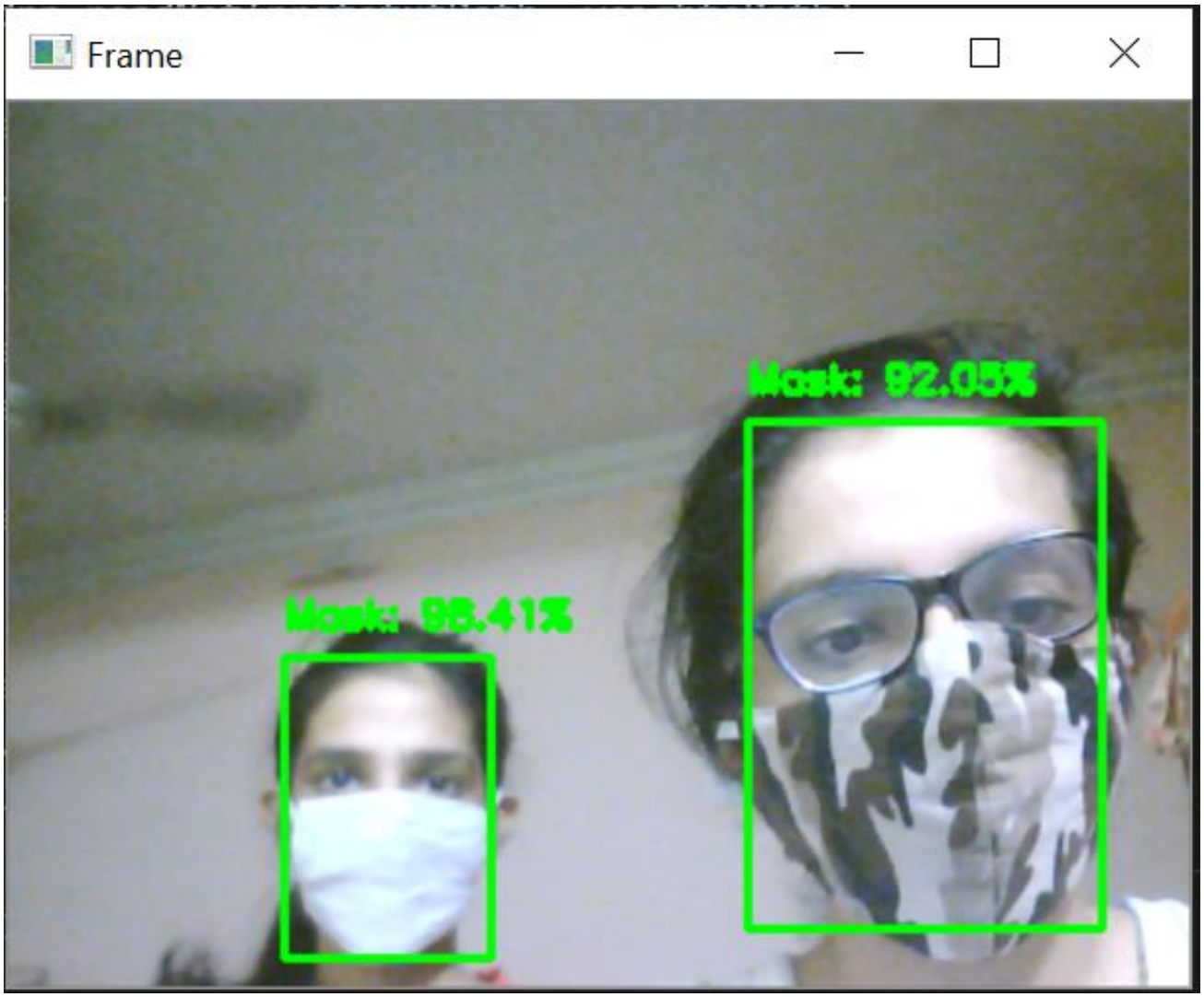


2) WITH MASK

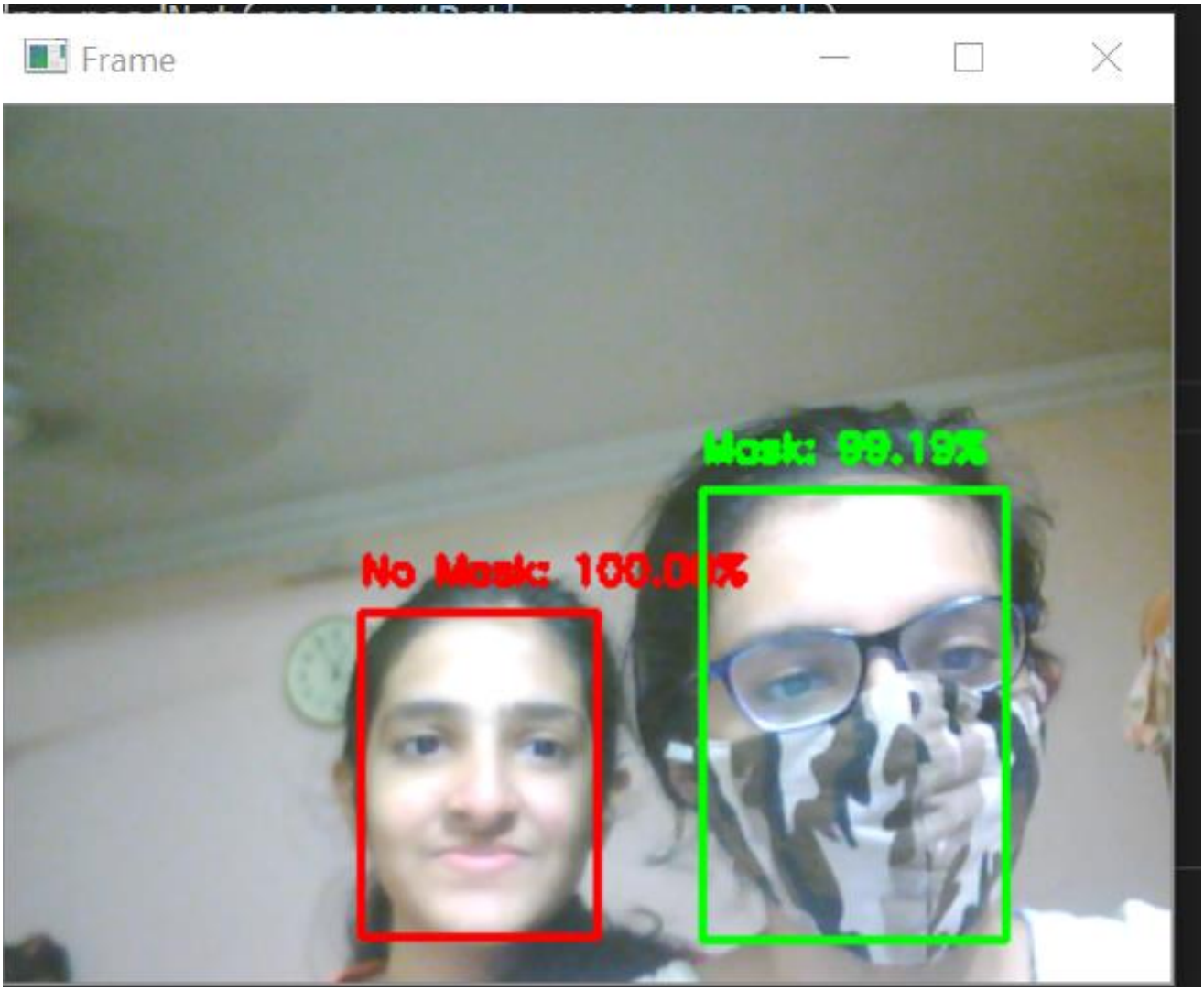
1.



2.



3) WITH AND WITHOUT MASK



LIMITATION

- It would be very useful technique to keep a track on human health, which reduces the manual work, but have certain disadvantage as well.

- Before the pandemic days, we used to live free, no restriction on mask, but after COVID-19 came into light, everybody initially got worried, but not take things lightly.

- So, protection is quite essential, but after pandemic, this application won't be of any use. But obvious, the pandemic is going to get over, so this application would be a waste after that.

- However, this application can be converted into any other form as well, like rather than using face mask detector, one can use it as face detector.

- Face detector can be used as to detect who as the outsider coming to the society and how often they enter.

FUTURE SCOPE

- Currently this application is capable of identifying detection of faces, violating the mask norms. This can further be enhanced to include the functionality of detecting the violation of social distancing also.
- This same approach can be used to develop a model which could be used in a variety of real time detections and operations.
- The proposed technique can be integrated into any high-resolution video surveillance devices and not limited to mask detection only.
- The model can be extended to detect facial landmarks with a facemask for biometric purposes.
- It can be enhanced to include an alarm system, which will raise an alarm every time it detects a violation.

CONCLUSION

- Due to the urgency of controlling COVID-19, the application value and importance of real-time mask and social distancing detection are increasing.

- This project developed a model to detect face mask wearing among people to detect any violations.

- It deals with facemask dataset including images of people with mask, without mask, and incorrect mask wearing. To increase the training dataset, 1,000 images with different types of masks wearing were collected and added to the dataset to create a dataset with 1,853 images.

- This model can be used at any public places or security check places like airport terminals, entrance of office buildings, crowded places, construction sites, etc. to assure their safety in the COVID-19 pandemic.

- Mass screening is possible and hence can be used in crowded places like railway stations, bus stops, markets, streets, mall entrances, schools, colleges, etc.

REFERENCES AND BIBLIOGRAPHY

I referred to the following links to give functionality to my project:

Reference links:

- https://www.tensorflow.org/api_docs/python/tf
- https://www.stackoverflow.com
- https://www.youtube.com
- https://www.sciencedaily.com/releases/2020/04/200403132345.htm
- https://www.google.com
- https://makeml.app/datasets/mask
- https://www.kaggle.com

APPENDIX

- The World Health Organization recommends wearing a face mask and practicing physical distancing to mitigate the virus's spread.

- The main objective of this project is to build a technology which tracks and keeps a record, detects face mask for human health.

- Technology used:

  Machine learning using Python libraries

  - Tensorflow
  - OpenCV
  - Keras

- Algorithm used:
  - CNN