Artificial Intelligence Report
FACE MASK DETECTION
(UCS411)
Fourth Semester

Submitted by:
JASLEEN KAUR 102103658
AAYUSHI PURI 102103676
SHIVANSH PANDEY 102103682
ASEEM BHASKAR 102103679

BE Second Year, COE24

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology, Patiala

APRIL 2023

INTRODUCTION

In this Face-Mask Detection project, we have created a face mask recognizer using OpenCV, Keras/TenserFlow and Python. The proposed model can be integrated with surveillance camera to detect people are wearing mask and not wearing masks. Here, the hand region is extracted from the background and the hand sign shown by the person in front of the camera is detected and the alphabet appears on the screen accordingly. Deep learning has gained more attention in object detection and is used for human detection purposes. Using deep learning we developed a face mask detection tool that can detect whether the individual is wearing mask or not. This is done by evaluation of the classification results by analyzing real-time streaming from the Camera.

LITERATURE SURVEY

1. "Face Mask Detection using Machine Learning and Image Processing Techniques" by S. Gope et al. (2021)

This paper proposes a face mask detection system that uses machine learning and image processing techniques. The authors used a dataset of images of individuals wearing and not wearing masks, and developed a set of features that could be used to distinguish between the two classes. They then trained a support vector machine (SVM) classifier on these features. The system achieved an accuracy of 95.7% on a test set of images.

2. "Real-Time Mask Detection using Deep Learning on Mobile Devices" by M. S. Islam et al. (2021)

This paper proposes a real-time face mask detection system that can run on mobile devices. The authors used a deep learning approach, training a CNN on a dataset of images of individuals wearing and not wearing masks. The system achieved a real-time processing speed of 25 frames

3. "Automated Detection and Recognition of Face Masks" by H. S. Jang et al. (2021)

This paper proposes an automated face mask detection and recognition system that uses a combination of deep learning techniques and feature extraction. The authors used a dataset of images of individuals wearing and not wearing masks and used a pre-trained CNN to extract features from the images. They then used these features to train an SVM classifier. The system achieved an accuracy of 98.7% on a test set of images.

4. "Face Mask Detection using Transfer Learning and Ensemble Learning Techniques" by D. Thota et al. (2021)

This paper proposes a face mask detection system that uses transfer learning and ensemble learning techniques. The authors used a pre-trained CNN to extract features from images of individuals wearing and not wearing masks, and then used an ensemble of classifiers to make predictions. The system achieved an accuracy of 98.5% on a test set of images.

5. "A Deep Learning Based Face Mask Detection System using Thermal Imaging Cameras" by M. Khan et al. (2021)

This paper proposes a face mask detection system that uses thermal imaging cameras to detect the presence of masks. The authors used a deep learning approach, training a CNN on a dataset of thermal images of individuals wearing and not wearing masks. The system achieved an accuracy of 99.1% on a test set of images, demonstrating the potential of thermal imaging cameras for mask detection.

6. "A Real-Time Deep Learning Based Face Mask Detection System for Video Streams" by A. Garg et al. (2021)

This paper proposes a real-time face mask detection system that can process video streams in real-time. The authors used a deep learning approach, training a CNN on a dataset of video frames of individuals wearing and not wearing masks. The system achieved a real-time processing speed of 30 frames per second, demonstrating its potential for real-world applications.

7. A Systematic Review of Face Mask Detection Techniques by S. Narayan et al. (2021)

This paper provides a systematic review of recent research on face mask detection techniques. The authors identify common approaches used in this area, such as deep learning, machine learning, and image processing techniques, and discuss the strengths and weaknesses of each approach.

8. “Face mask detection using deep learning: An approach to reduce risk of Coronavirus spread” by Shilpa Sethi, Mamta Kathuria and Trilok Kaushik.

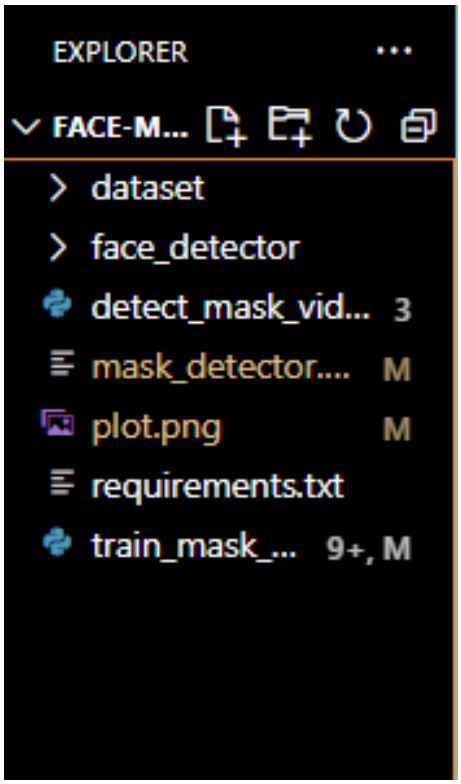
This paper proposes a real time face mask detection system that Develops a novel object detection method that combines one-stage and two-stage detectors for accurately detecting the object in real-time from video streams with transfer learning at the back end.

METHODOLOGY

Dataset

https://drive.google.com/drive/folders/1WCxe1EuxLo6qyGVpupcEMTgN83xpgHM

A directory structure can be visualized as in the following snapshot: -



Important modules: -

```python
# 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
```

IMPORTING USED MODULES

Brief description of the imported modules:

1. cv2: OpenCV-Python is a library of Python bindings designed to solve computer vision problems. cv2.imshow() method is used to display an image in a window. The window automatically fits to the image size.

2. numpy: NumPy contains a multi-dimensional array and matrix data structures. It can be utilized to perform a number of mathematical operations on arrays.

3. Keras: Keras is a high-level neural network API written in Python that runs on top of various lower-level deep learning frameworks like TensorFlow, Theano, and CNTK. It provides a user-friendly and modular interface for building and training deep learning models

4. Tensorflow: TensorFlow is an open-sourced end-to-end platform, a library for multiple machine learning tasks, while Keras is a high-level neural network library that runs on top of TensorFlow. Both provide high-level APIs used for easily building and training models, but Keras is more user-friendly because it's built-in Python.

5. imutils: Imutils is a Python package providing a series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much more easier with OpenCV and both Python 2.7 and Python 3. OpenCV is a popular computer vision library, and imutils extends OpenCV by providing a collection of functions that can be used to streamline common tasks such as resizing, rotating, and cropping images.

6. matplotlib: Matplotlib is a Python library used for creating static, interactive, and animated visualizations in Python. It provides a wide range of tools for creating various types of charts, plots, histograms, scatterplots, and other visualizations. Matplotlib is one of the most widely used visualization libraries in Python and is an essential tool for data analysis and visualization.

7. scipy: Scipy is a Python library for scientific and technical computing. It provides a wide range of tools for mathematics, engineering, statistics, and data analysis, and is built on top of the NumPy library, which provides support for numerical operations on arrays and matrices.

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

DIRECTORY = r"D:\Mask_Detection\Face-Mask-Detection\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)
```

In the Beginning of code, there are three hyperparameters used (INIT_LR , EPOCHS , BS) .These Hyperparameters are important for training a model for Face Mask Detection and can be adjusted to improve model performance during training.

INIT_LR: This variable stands for "initial learning rate". It is a hyperparameter that determines the step size at which the model's parameters are updated during training

EPOCHS: This variable determines the number of times the entire dataset will be passed through the model during training. One epoch means one pass through the entire dataset.

BS: This variable stands for "batch size". It determines the number of samples that will be processed by the model at once before updating the parameters.

Further, code snippet appears to be loading images from a dataset directory for a face mask detection task. The images are loaded using the load_img function from the keras. preprocessing.image module and are resized to a target size of (224, 224) using the target_size argument. The loaded images are then converted to NumPy arrays using img_to_array and preprocessed using preprocess_input function from keras.applications.mobilenet_v2 module.The image data and corresponding labels are stored in two separate lists, data and labels. The images are loaded from two subdirectories named "with_mask" and "without_mask" in the main dataset directory specified by DIRECTORY. The categories are defined in the CATEGORIES list.

```
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)
```

label binarizer converts categorical labels (like with mask and without mask) into binary format. It is used to convert the string labels into binary format, so that the machine learning model can work with them.and next, we have to convert labels and data into numpy arrays. Train_test_split is a function that splits the data into training and testing sets. Here, data and labels are split into trainX, testX, trainY, and testY, with a test size of 20% and stratified sampling based on the labels. The random_state parameter is set to 42 to ensure reproducibility.

```
# 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
```

The model architecture used is MobileNetV2 architecture. Firstly, an instance of ImageDataGenerator is created to perform data augmentation on the input images, such as rotation, zoom, shifting, shearing, and flipping. Then, the MobileNetV2 pre-trained model is loaded with its head FC layer removed using the "include_top=False" argument. The head of the model is constructed by adding a global average pooling layer, a flattened layer, a dense layer with 128 units, and a dropout layer with a rate of 0.5. Finally, a softmax activation dense layer with two units is added as the output layer for binary classification (mask or no mask). After constructing the head of the model, it is then placed on top of the base model using the Keras functional API. Then, all the layers in the base model are frozen to prevent updating the pre-trained weights during the first training process. The resulting model is used for training with the frozen base layers and the newly added head layers. During the training process, the model will learn to classify whether a face has a mask or not based on the input image.

```
# compile our model
print("[INFO] compiling model...")
opt = Adam(learning_rate=INIT_LR)
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)
```

The training data is augmented with various transformations to improve the model's accuracy. The model is compiled with the Adam optimizer, binary cross-entropy loss, and accuracy as the evaluation metric. Then, it is trained using the "fit" method, which takes in the training data, batch size, and the number of epochs. The validation data is used to evaluate the model after each epoch, and the model's history is saved to monitor its performance. The training history includes information on the loss and accuracy of the model on both the training and validation data, which can be used to evaluate the model's performance and make any necessary adjustments to improve it.

```
# 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")
```

The code first evaluates the trained model's performance on a set of images that it has never seen before (the testing set). It does this by creating predictions for each image in the testing set using the model. predict() function. The predictions are made by passing the testing set images through the trained model, which outputs a probability for each class (face with a mask, a face without a mask).To determine the predicted class for each image, the np. argmax() function is used to find the index of the class with the highest probability.The code then shows a nicely formatted classification report using the classification_report() function. This report provides information about the precision, recall, F1 score, and support for each class. This information helps to assess how well the model is performing. Next, the trained model is saved to disk using the save () function. This is done so that the model can be easily loaded and used again in the future, without having to retrain it from scratch. Finally, the code plots the training loss and accuracy for each epoch (iteration) of the training process using the matplotlib library. This plot helps to visualize how the model's performance improved during the training process. The resulting plot is saved to disk using the savefig() function.

```
# 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 = []
```

This code defines a function that is responsible for detecting faces in an image or video frame and classifying whether or not each detected face is wearing a mask. To do this, the function takes in the image or video frame and two neural networks: one for detecting faces (faceNet) and one for classifying whether or not a face is wearing a mask (maskNet).he code first prepares

the input image or video frame for the faceNet by converting it into a preprocessed image called a "blob". This is done by resizing the frame to a fixed size of 224x224 pixels and then normalizing the pixel values by subtracting the mean RGB values of the dataset. The faceNet is then used to detect faces in the image by passing the blob through the network and obtaining the output of the network, which contains information about the location and confidence score of each detected face. The code then initializes empty lists to store the detected faces (faces), their corresponding locations (locs), and the predicted class for each face (preds). These lists will be filled with information about each detected face and its mask classification in the subsequent code.

```
# 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))
```

This code snippet is part of a face mask detection application. The purpose of this code is to extract faces from an input image or video frame and prepare them for processing by the face mask detection model. The function takes three inputs: the input image frame, a pre-trained face detection model (faceNet), and a pre-trained face mask detection model (maskNet).The first step is to create a blob from the input image frame. A blob is a formatted image that is passed through the face detection model. The face detection model then returns a set of detections used to identify the image regions containing a face. The code then initializes several empty lists to store the face images, the locations of the detected faces, and the corresponding face mask detection predictions. Next, the code loops through each detection returned by the face detection model. It filters out weak detections by checking that the confidence score associated with the detection is greater than a minimum value (in this case, 0.5). For each

detection that meets the minimum confidence threshold, the code extracts the (x, y)-coordinates of the bounding box around the detected face. It ensures that the bounding box falls within the dimensions of the image frame and extracts the face region of interest (ROI) by cropping the image based on the bounding box coordinates. The face ROI is then preprocessed by converting it to RGB channel ordering, resizing it to 224x224 pixels, and normalizing the pixel values. The processed face ROI, along with its bounding box location, is appended to the respective lists. Overall, this code is responsible for detecting faces in an input image or video frame and preparing them for processing by the face mask detection model.

```
# 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)
```

If there are one or more faces in the list, the function performs batch predictions on all the faces at the same time, rather than one-by-one predictions. It uses the pre-trained face mask detection model to predict whether each face in the list is wearing a mask or not. The function returns a tuple containing the face locations and their corresponding mask predictions. Finally, the function loads the pre-trained face detection and face mask detection models from the disk. The face detection model is stored in a.prototxt file that defines the model architecture and a .caffemodel file containing the model weights. The face mask detection model is stored in a .h5 file that contains the model architecture and weights.

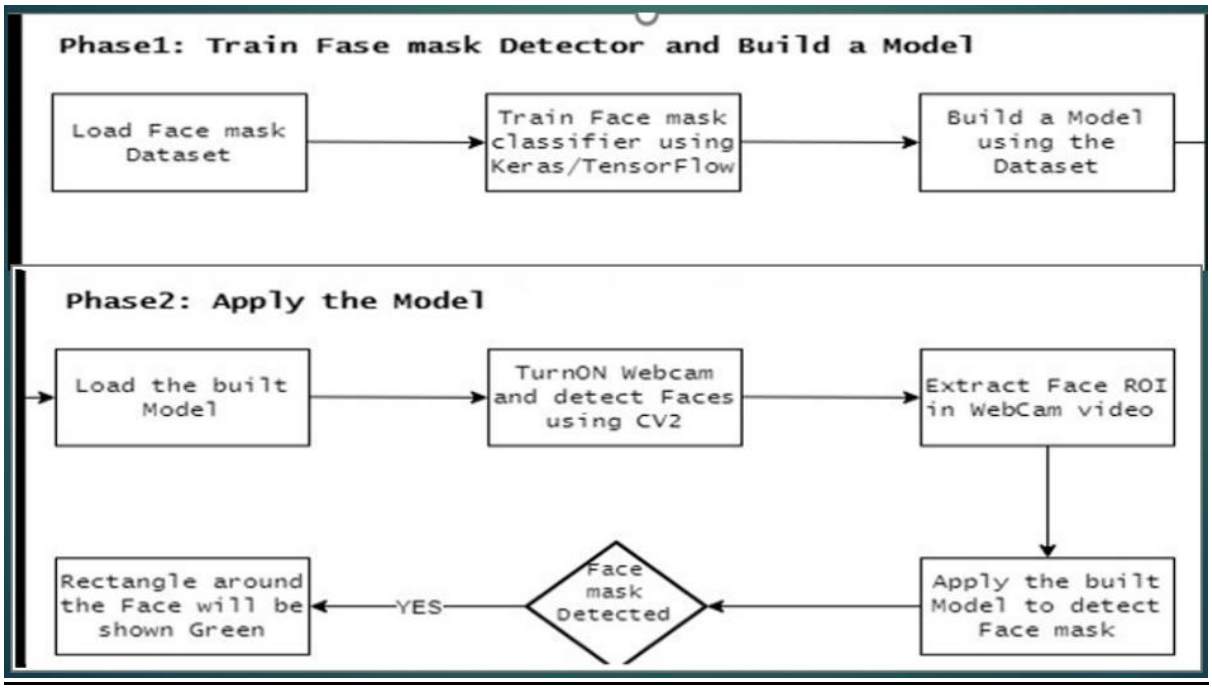
```
# initialize the video stream
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()

# 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)
```

The above code is the main logic for running face mask detection on a video stream. It reads frames from the video stream, resizes them to a maximum width of 400 pixels, and then uses the detect_and_predict_mask() function to detect faces and determine if they are wearing a mask or not. Once the faces and their corresponding predictions are obtained, the code loops over each face, and prediction unpacks the bounding box and prediction values and determines the class label based on whether the mask probability is greater than the without mask probability. The colour of the bounding box is set to green if the label is "Mask" and red if the label is "No Mask". The label text is then created by including the class label and the probability of the prediction.

```
        # 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()
```

This code snippet shows how the output of the face detection and mask prediction is displayed in a window. The loop iterates over each detected face and draws a label (either "Mask" or "No Mask") with the corresponding confidence score on the frame. It also draws a rectangle around each face, coloured green if the person is wearing a mask and red if not. The output frame is then displayed in a window using the cv2.imshow() function. The loop continues until the user presses the 'q' key, at which point the program exits and the window is closed. Finally, any remaining windows and the video stream are closed with cv2.destroyAllWindows() and vs. stop(), respectively.

Flowchart: -



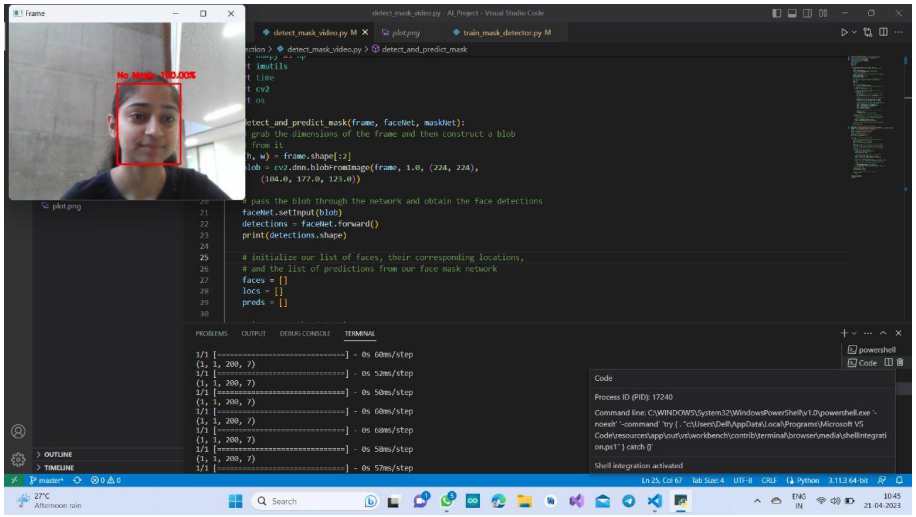
Working of Project: -

RESULTS

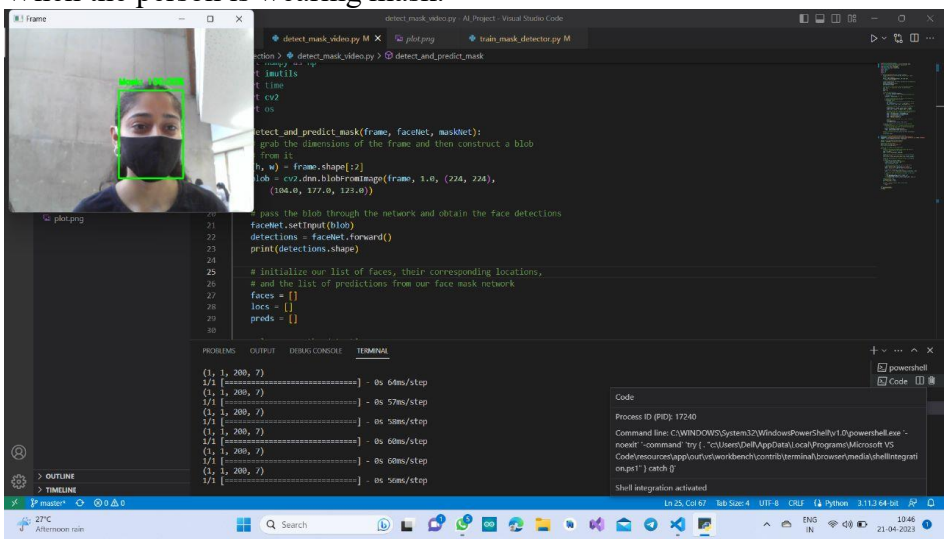
The result is a project that provides the detection of a face mask. The output “face mask on/off” is displayed with accuracy on the screen along with the detected face in a box. This eliminates the manual checking of face masks as it is difficult for the human eye to spot the ones wearing face masks and the ones not in highly populated areas, say a mall, airports, railway stations, a stadium, and many such locations.

A box is drawn over the face of the person which describes whether the person is wearing mask or not. Green box shows that the person is wearing mask and red box shows not wearing mask.

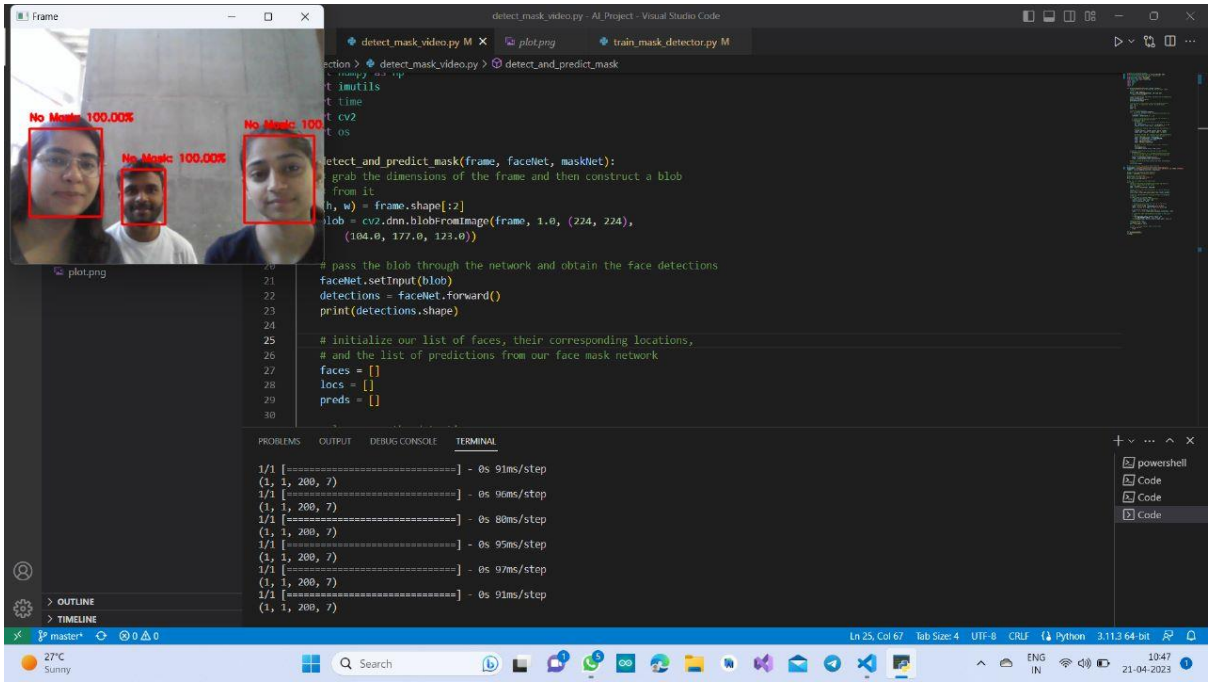
When the person is not wearing mask.

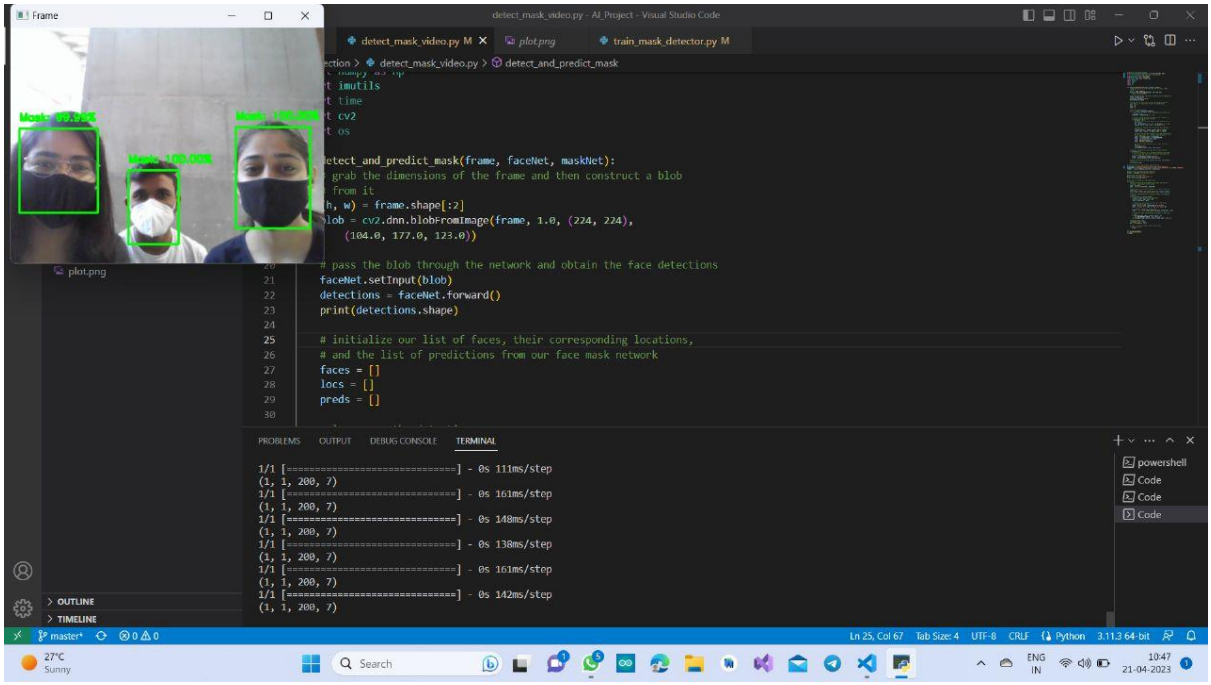


When the person is wearing mask.



Multiple faces can be detected in one single frame.





FUTURE SCOPE

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 integrated system not only helps to achieve high accuracy, but also increases the detection speed. 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 benefit.

This machine may be employed in public places like railway stations, bus stands, airports, department shops, etc. It is going to be of a top notch help in corporations and big establishments where there are quite a few workers. This gadget may be of high quality assist as it is easy to attain and save the information of personnel working in that organisation.

Finally, the work opens interesting future directions for researchers. Firstly, the proposed technique can be integrated into any high-resolution video surveillance device and is not limited to mask detection only. Secondly, the model can be extended to detect facial landmarks with a facemask for biometric purposes.