PROJECT NAME: Secure Access Control with Mask and Face Recognition Technology

Final Project Report

Course Name: Electrical and Electronic Workshop

Course Title **:** EEE 4518

Group Members:

Name	**ID**
Khandaker Adeba Tabassum	200021102
Sadat Al Rashad	200021106
Mohammad Mohibul Haque	200021111
Muslima Siddiqui	200021117
Md. Nazmul Aman	200021132

Table of Contents

Objectives

Providing Safe and Secured access to sensitive places.

Description

Our project tries to introduce a sophisticated security and hygiene solution tailored for environments like hospitals. The primary objective of our project is to regulate access through a door by employing facial recognition technology which is also paired with mask detection. It goes through a sequence which is like:

- The camera identifies an individual's face as they approach the door.
- A comparison is made between the detected face and those stored in a database to correctly know the person’s identity.
- Upon recognition, the system prompts the individual to wear a mask.
- If the person is successfully recognized and wearing a mask, the system grants access by opening the door.
- To maintain security, the door remains open for a predetermined duration (e.g., 20 seconds) before closing automatically.

This comprehensive system not only enhances security by allowing access solely to recognized individuals with proper mask compliance but also promotes hygiene practices. This system can be particularly useful for Doctors in Hospitals, in sensitive laboratories etc. Nevertheless, key considerations include ensuring the accuracy of detection methods, adaptability to various conditions, and regular maintenance to keep pace with evolving technology and security standards.

Learning Roadmap

Our learnings for this project began with the essential step of acquiring programming language proficiency. Python and R are the most suitable languages for computer vision and deep learning. We decided to start with Python due to its beginner-friendly nature and a bit of previous experiences with the language.

Next, we learnt how to use the OpenCV library, which is considered a really useful library in computer vision. We used OpenCV for real-time webcam feeds and incorporated it into our face and mask detection system.

Later on, we opted to learn about deep learning. Deep learning allows the systems to identify and classify the objects and execute complex tasks as per the requirements. To implement deep learning algorithms, a suitable programming framework TensorFlow, a Python library for training models for facial recognition and mask detection was chosen.

Lastly, we started to learn about Convolutional Neural Networks (CNNs), for understanding and implementing advanced computer vision tasks.

We also learned how to perform serial communication, a method of establishing connections between multiple devices through the use of COM ports. This would be useful for incorporating the use of ARDUINO in our project to control the door lock.

Convolutional Neural Network

A Convolutional Neural Network is a type of deep neural network designed to process and analyze visual data. Unlike traditional neural networks, CNNs are built upon a unique architecture which was inspired

by the visual processing of living organisms. This network architecture enables them to automatically and adaptively learn and extract features from input images.

There are multiple key layers of neurons in a Convolutional Neural Network.

Convolutional Layers are the layers that set CNNs apart from other neural networks. They are used to apply filters or kernels to input data, which allows the network to detect patterns and features. Convolutional layers allow the filters to slide over the input, then performing convolutions and extracting features like edges, textures, and shapes.

Pooling layers reduce the dimensions of the convolved features. This down-samples the extracted information. This allows the network to retain only the crucial features while discarding unnecessary details. This helps in enhancing the network's efficiency and reducing computational time.

A dense layer is fully connected to the previous layer, meaning that each neuron in the dense layer receives input from every neuron in the preceding layer. A dense layer performs a linear operation on the input and then applies an activation function to give the prediction.

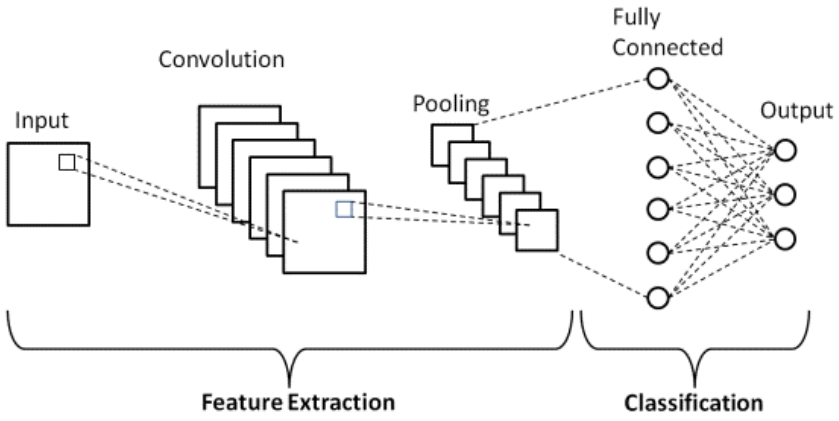
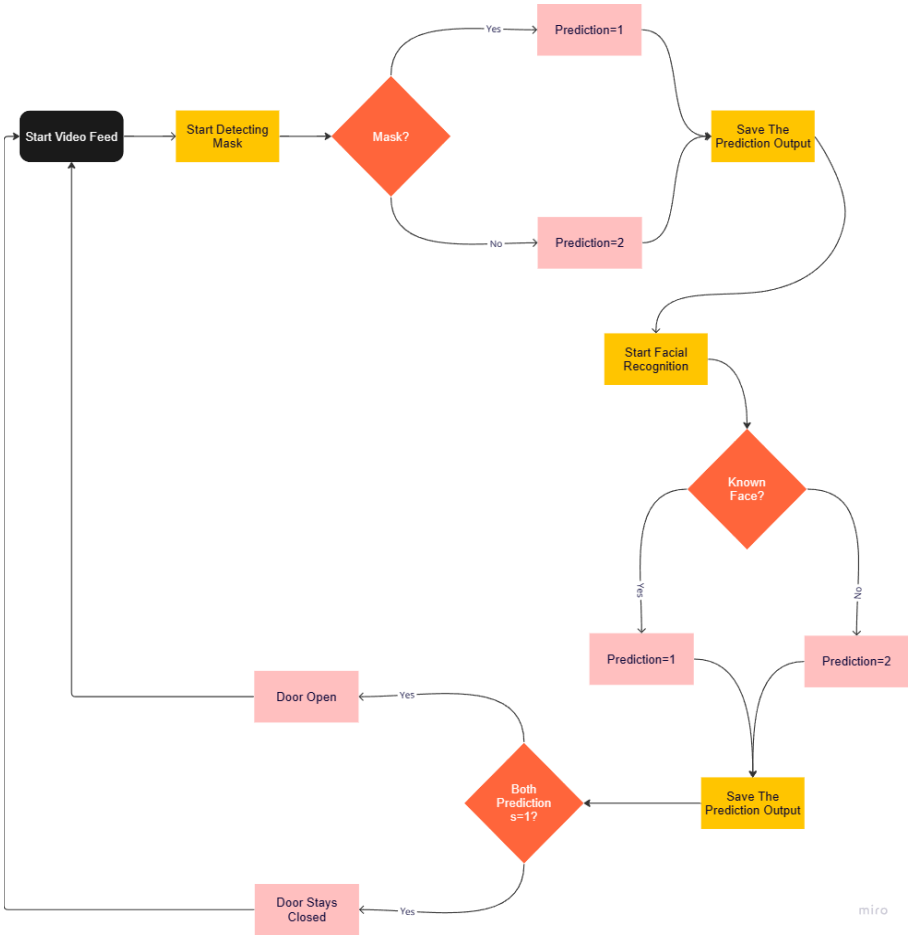


Fig. : Brief Overview of Layers in a Convolutional Neural Network

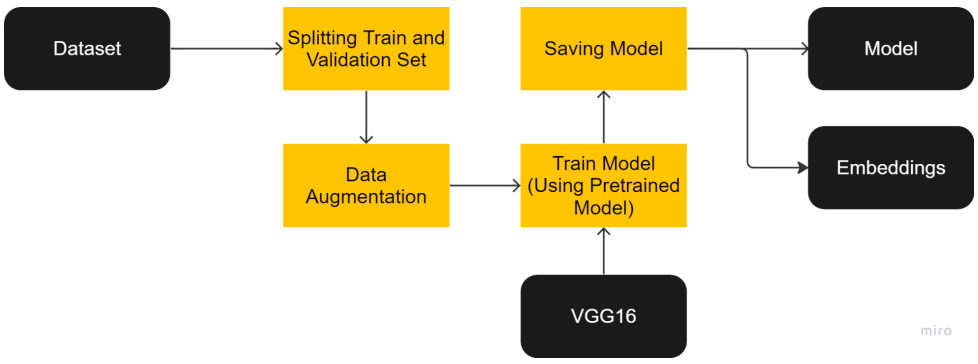
Algorithm:



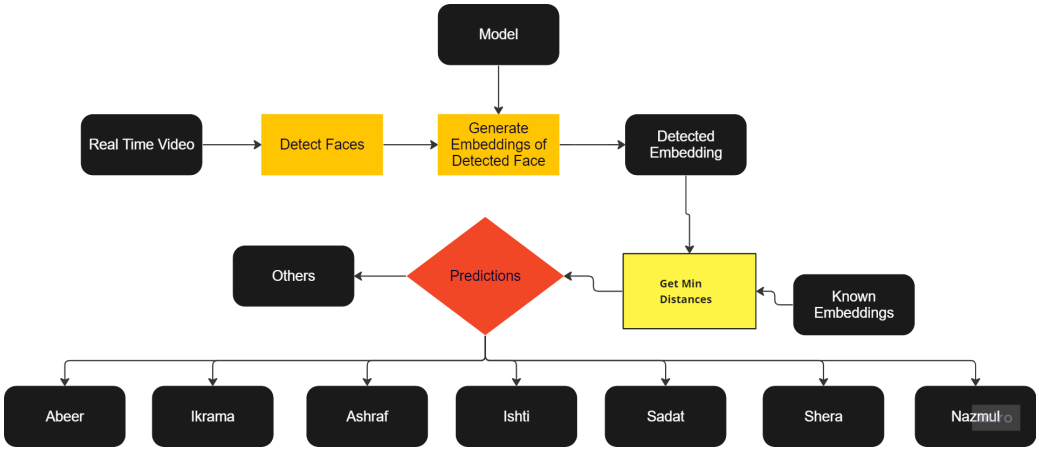
Methodology

Face Recognition:

Algorithm For Training Models:



Algorithm For Real Time Face Verification:



Model Training with Code:

Importing necessary libraries

```
import os
import shutil
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.callbacks import EarlyStopping
```

Setting Directories and Specifying the train ratio

```
original_dataset_dir = 'Data'

base_dir = 'split_data'
os.makedirs(base_dir, exist_ok=True)

# Creating the train and validation directories
train_dir = os.path.join(base_dir, 'train')
os.makedirs(train_dir, exist_ok=True)

valid_dir = os.path.join(base_dir, 'valid')
os.makedirs(valid_dir, exist_ok=True)

# Splitting Ratio
train_ratio = 0.9
```

Splitting the dataset into train and test sets using a loop that will go through all the subdirectories in the dataset directory.

```
for class_name in os.listdir(original_dataset_dir):
    class_dir = os.path.join(original_dataset_dir, class_name)

    if not os.path.isdir(class_dir):
        continue

    # Split images as training and validation datasets
    all_images = os.listdir(class_dir)
    train_images, valid_images = train_test_split(all_images, train_size=train_ratio, random_state=42)

    for img in train_images:
        src = os.path.join(class_dir, img)
        dst = os.path.join(train_dir, class_name, img)
        os.makedirs(os.path.join(train_dir, class_name), exist_ok=True)
        shutil.copyfile(src, dst)

    for img in valid_images:
        src = os.path.join(class_dir, img)
        dst = os.path.join(valid_dir, class_name, img)
        os.makedirs(os.path.join(valid_dir, class_name), exist_ok=True)
        shutil.copyfile(src, dst)
```

Specifying necessary parameters and getting the number of classes for our classifier. It also loads the pretrained VGG-16 Model excluding the top layer for the purpose of Transfer Learning. VGG-16 is a popular pretrained model due to its simplicity and effectiveness in image classification tasks.

```
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
NUM_CLASSES = len(os.listdir(original_dataset_dir))
EPOCHS = 10
DROPOUT_RATE = 0.5


# Load the pre-trained VGG-16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the layers except the last one
for layer in base_model.layers:
    layer.trainable = False
```

Adding further layers for extracting features of our Dataset.

The first line initializes a sequential model. This model represents a linear stack of layers. Then adds the VGG-16 model. Then it adds a Global Average Pooling 2D layer which reduces the spatial dimensions of the previous layers' outputs by taking the average for each feature map, effectively flattening the data while retaining important information. Then Adds a dense layer with 512 neurons and ReLU (Rectified Linear Unit) activation. This layer allows the network to learn more complex representations of the data. Then adds a Dropout Regularization layer which prevents overfitting. Final Dense layer gives predictions based on our number of classes. The softmax activation function normalizes the output vector into a probability distribution across the different classes, giving the probability scores for each class.

```
model = models.Sequential()
model.add(base_model)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(DROPOUT_RATE))
model.add(layers.Dense(8, activation='softmax'))
```

Compiling the Model with a learning rate specified with Adam Optimizers and Categorical Crossentropy Loss.

```
# Compile the model with Adam optimizer and an adjusted learning rate
model.compile(optimizer=optimizers.Adam(learning_rate=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In this portion we generating the Training Set along with some Data Augmentation which creates further variations in our dataset.

```
# Augmenting Data
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1,
    brightness_range=[0.8, 1.2],
    rotation_range=20
)

# Training set generator
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)
```

Validation data is generated in the same way.

```
# Validation set generator
valid_datagen = ImageDataGenerator(rescale=1./255)

valid_generator = valid_datagen.flow_from_directory(
    valid_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)
```

We define another callback that will monitor the validation accuracy while training. If the accuracy is not increasing for 3 (patience) epochs then it will restore the best epoch weights.

```
# Define EarlyStopping callback
early_stopping_callback = EarlyStopping(monitor='val_accuracy', patience=3, mode='max', verbose=1, restore_best_weights=True)
```

Then we fit our dataset into the model and save the model.

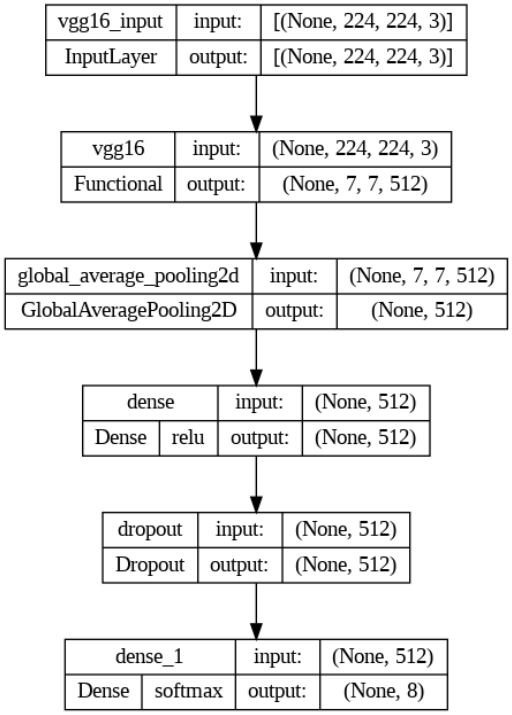
```
# Train the model with EarlyStopping callback
history = model.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=valid_generator,
    callbacks=[early_stopping_callback]
)

# Save model
model.save('NotunModel.h5')
```

Saving the embeddings and Labels.

```
known_embeddings = np.load('known_embeddings.npy')
known_labels = np.load('known_labels.npy')
print(known_embeddings)
```

Model Structure:



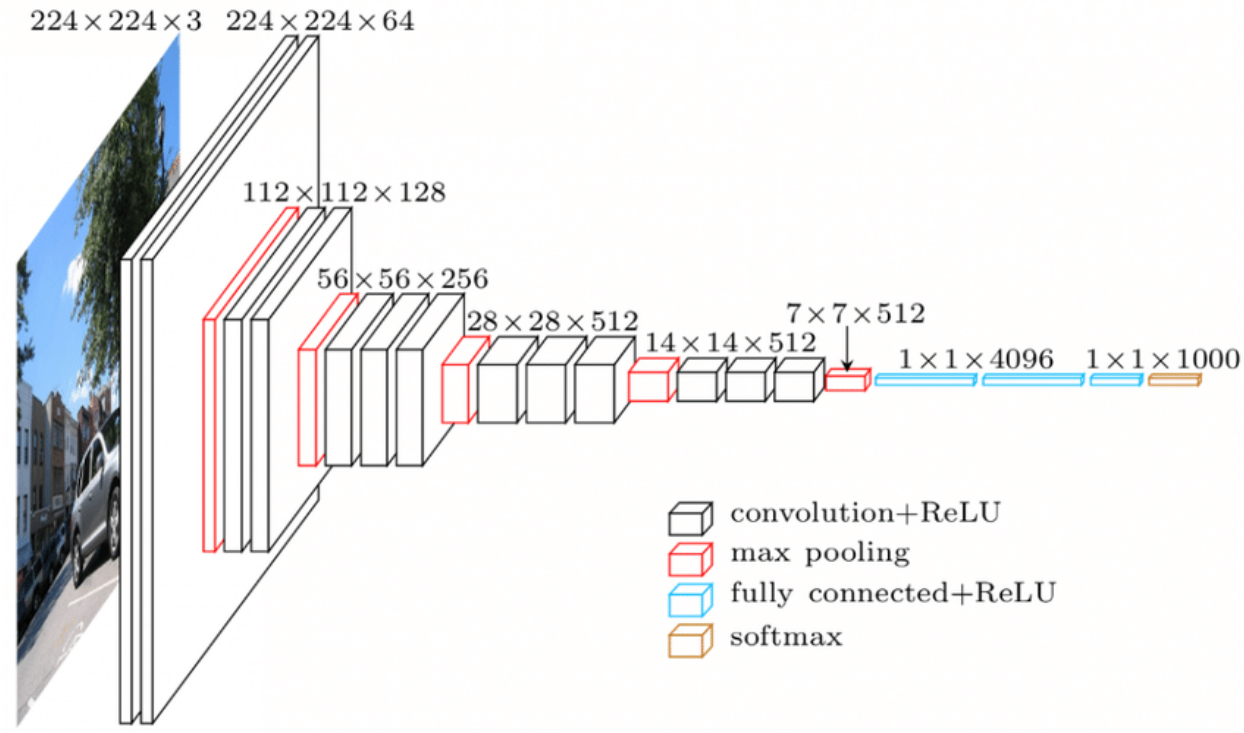


Fig. : Brief Overview of Architecture of VGG-16

Real Time Face Recognition with Code:

At first we access the webcam and then using mtcnn we detect faces from the frame. We are using mtcnn instead of haarcascade classifier due to its accurate results while detecting faces.

```
while True:
    if not self.paused:
        ret, frame = self.video_capture.read()

    if not ret:
        break

    frame = imutils.resize(frame, width=900)
    faces = self.detect_faces_mtcnn(frame)
```

Then we use a loop in case there are multiple faces and preprocess the captured face. Then we get embeddings of the detected face by passing it

to the model. Then the distance is calculated from our known embeddings. then we get the minimum distance and similarity index.

```
for face in faces:
    x, y, w, h = face['box']
    roi = frame[y:y + h, x:x + w]

    preprocessed_face = cv2.resize(roi, (224, 224)) / 255.0
    preprocessed_face = np.expand_dims(preprocessed_face, axis=0)

    face_embedding = self.face_recognition_model.predict(preprocessed_face)

    distances, indices = self.neighbors.kneighbors(face_embedding, n_neighbors=1)
    similarity_score = 1 - distances[0][0]
    confidence_threshold = 0.6
```

Then we display the labels in the video feed. We are also storing predictions as 0 (For Others) and 1(For Known).

```
    if similarity_score > confidence_threshold:
        closest_label = self.known_labels[indices[0][0]]
        if closest_label == 5:
            self.current_prediction = 0
        else:
            self.current_prediction = 1
        recognized_name = self.label_to_name.get(closest_label, 'Unsure')
        cv2.putText(frame, f'Recognized: {recognized_name} ({similarity_score:.2f})',
                    (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2, cv2.LINE_AA)
    else:
        cv2.putText(frame, 'Unsure', (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2,
                    cv2.LINE_AA)

cv2.imshow('Real-Time Facial Recognition', frame)
```

If we press e it will store the predictions in a file and based on the file's it will decide whether to operate our servo or not. (Discussed later on in the hardware portion).

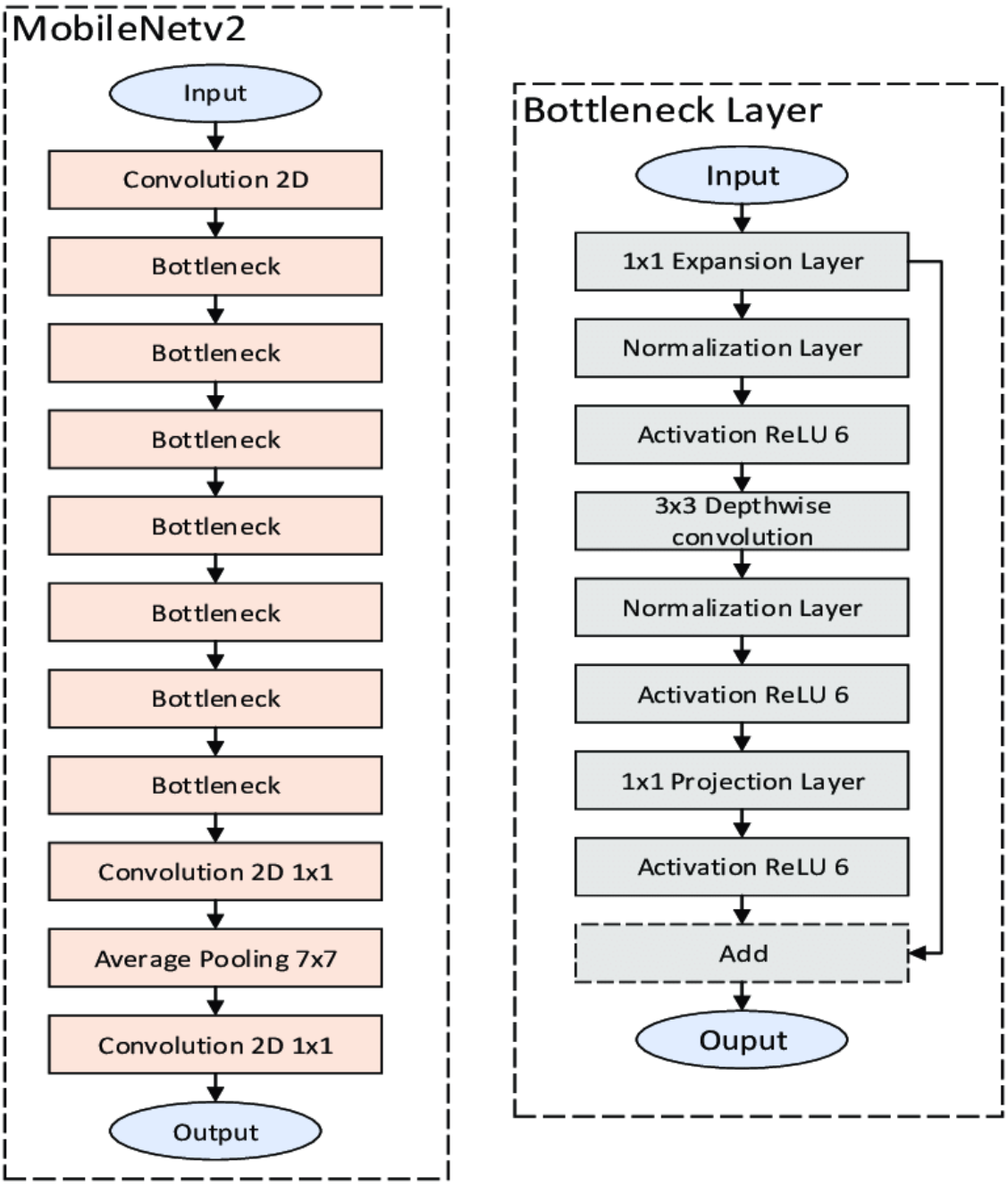
```
key = cv2.waitKey(1) & 0xFF

if key == ord("q"):
    break
elif key == ord("e"):
    with open(self.prediction_file_path, "w") as file:
        file.write(str(self.current_prediction) + "\n")
    path1="mask_predictions.txt"
    path2=self.prediction_file_path
    predicts=[]
    with open(path1, 'r') as file1:
        predicts.append(int(file1.read().strip()))
    with open(path2, 'r') as file2:
        predicts.append(int(file2.read().strip()))
    print(predicts)
    if sum(predicts) == 2:
        doorAutomate(0)
        time.sleep(10)
        doorAutomate(1)
```

Mask Detection Model:

As for the mask detection, at first we built the model. For mask detection purpose, we decided to take the dataset from Kaggle. At first we tried to build the model from scratch creating each layer one by one. We used sequential model and implemented convolutional neural network on it. But as we did not get that much accuracy we decided to use a pre-trained model. For building the model of mask detection we took MobilenetV2 as our pre-trained model. MobileNetV2 is a deep learning architecture that is specifically designed for mobile and edge devices. It is an evolution of the original MobileNet architecture and incorporates several features that make it well-suited for efficient and lightweight neural network deployments. It has approximately 157 layers and 3.4 million parameters. It predominantly uses depthwise separable convolution which is a combination of both depthwise convolutions and pointwise convolution.

MobilenetV2 Architecture:



Model Training with Code:

Importing necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from termcolor import colored
import cv2
import tensorflow as tf
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
#     from     tensorflow.keras.callbacks     import     EarlyStopping, ModelCheckpoint

from IPython.display import clear_output

import warnings
warnings.filterwarnings('ignore')
```

Setting Directories

```
train_dir = 'Face Mask Dataset\Train'
test_dir = 'Face Mask Dataset\Test'
val_dir = 'Face Mask Dataset\Validation'
```

In this portion we are generating the Training Set along with some Data Augmentation which creates further variations in our dataset.

```
IMG_SIZE = (256, 256)

train_datagen = ImageDataGenerator(rescale=1/255.0,
```

```
                                        rotation_range=45,
                                    shear_range=0.2,
                                    zoom_range=0.2,
                                    horizontal_flip=True,
                                    vertical_flip=True
                                    )

test_datagen = ImageDataGenerator(rescale= 1 / 255.0)

train_dataset     =     train_datagen.flow_from_directory(train_dir,
target_size=(IMG_SIZE),
                                        color_mode="rgb",
                                    batch_size=200,
                                        shuffle=True,
                                    class_mode="categorical")

test_dataset      =      test_datagen.flow_from_directory(test_dir,
target_size=(IMG_SIZE),
                                        color_mode="rgb",
                                    batch_size=64,
                                        shuffle=True,
                                      class_mode="categorical")

validation_dataset    =    train_datagen.flow_from_directory(val_dir,
target_size=(IMG_SIZE),
                                        color_mode="rgb",
                                    batch_size=64,
                                        shuffle=True,
                                    class_mode="categorical")
```

Specifying necessary parameters for the pretrained MobilnetV2 Model excluding the top layer for the purpose of Transfer Learning.

```
mobilenet                                                          =
MobileNetV2(weights='imagenet',include_top=False,input_shape=(256,2
56,3))
# make pre trained model into non trainable bcoz its takes much time
for layer in mobilenet.layers:
    layer.trainable = False
```

Adding further layers for extracting features of our Dataset.

```
model = Sequential()
model.add(mobilenet)
model.add(Flatten())
model.add(Dense(128,                                    activation="relu",
kernel_initializer="he_uniform"))
model.add(Dense(2,activation='sigmoid'))
model.summary()
```

Summary of the model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Functi (None, 8, 8, 1280)     2257984
_________________________________________________________________
flatten (Flatten)            (None, 81920)             0
_________________________________________________________________
dense (Dense)                (None, 128)               10485888
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 12,744,130
Trainable params: 10,486,146
Non-trainable params: 2,257,984
_________________________________________________________________
```

Compiling the Model with Adam Optimizers and Categorical Crossentropy Loss.

```
model.compile(optimizer=Adam(),
              loss='categorical_crossentropy',
              metrics = ['accuracy'])
```

Then we fit our dataset into the model and save the model.

```
history = model.fit(train_dataset,
                            validation_data=validation_dataset,
                            epochs=5,
                            verbose=1)
model.save('Mask2.h5')
```

Plotting accuracy and loss vs number of epochs.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(len(acc))
plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, acc,"go-", label='Training Accuracy')
plt.plot(epochs_range, val_acc,"ro-", label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss,"go-", label='Training Loss')
plt.plot(epochs_range, val_loss,"ro-", label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Real time mask detection with code:

Firstly the code imports necessary libraries/modules like OpenCV (cv2), TensorFlow, Keras, imutils for image processing, and others

```
    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
```

Declaring a Threshold for face detection confidence and An array to store the predictions (1 for mask, 0 for no mask).

```
lowConfidence = 0.75
predictions = []
```

Then we define a function which uses our pre-trained face detection model to identify faces in the frame; it then extracts and preprocesses face regions and uses a mask detection model to determine whether a mask is present.

```
def detectAndPredictMask(frame, faceNet, maskNet):
```

Then we extracts the input frame's width and height and use OpenCV's 'blobFromImage' function to create a blob, a suitable representation for neural network input, before processing the frame.

```
(h, w) = frame.shape[:2]
blob = cv2.dnn.blobFromImage(frame, 1.0, (256, 256),
(104.0, 177.0, 123.0))
faceNet.setInput(blob)
```

Then we use the prepared blob to identify faces in the frame and use a preset threshold in the processing of the detections to filter out low-confidence detections.

```
detections = faceNet.forward()
faces = []
locs = []
preds = []
```

```
for i in range(0, detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > lowConfidence:
        box = detections[0, 0, i, 3:7] * np.array([w,
h, w, h])
        (startX,     startY,     endX,     endY)     =
box.astype("int")
        (startX,  startY)  =  (max(0,  startX),  max(0,
startY))
        (endX, endY) = (min(w - 1, endX), min(h - 1,
endY))
```

Then we preprocess our face to use it to make our predictions.

```
face = frame[startY:endY, startX:endX]
 face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
 face = cv2.resize(face, (256, 256))
 face = img_to_array(face)
 face = preprocess_input(face)
 faces.append(face)
 locs.append((startX, startY, endX, endY))
```

Predicts whether or not each detected face is wearing a mask using our mask detection neural network (‘maskNet’). A list contains the forecasts for every face.

```
if len(faces) > 0:
    faces = np.array(faces, dtype="float32")
    preds = maskNet.predict(faces, batch_size=32)

return (locs, preds)
```

‘faceNet’-Loads the pre-trained face detection model
‘maskNet’- Loads the pre-trained mask detection model

```
prototxtPath = "deploy.prototxt"
                    weightsPath                    =
"res10_300x300_ssd_iter_140000.caffemodel"
```

```
 faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
 maskNet = load_model("Mask2.h5")

vs = VideoStream(src=0).start()
```

The Video stream loops endlessly in order to read frames, resize them, identify faces, and predict whether a face is mask- or non-mask-covered.

```
while True:
    if not paused:
        frame = vs.read()
        frame = imutils.resize(frame, width=900)
        (locs, preds) = detectAndPredictMask(frame,
faceNet, maskNet)
```

Then we indicate whether or not a mask is present on the frame by drawing rectangles and labels.

```
for (box, pred) in zip(locs, preds):
    (startX, startY, endX, endY) = box
    (mask, withoutMask) = pred
    label = "Mask" if mask > withoutMask else "No
Mask"
    color = (0, 255, 0) if label == "Mask" else (0, 0,
255)
```

Predictions are kept in the ‘predictions’ array (1 for mask, 0 for no mask).

```
if label == "Mask":
    predictions.append(1)  # Store 1 for mask
 else:
    predictions.append(0)  # Store 0 for no mask

label = "{}: {:.2f}%".format(label, max(mask,
withoutMask) * 100)
    cv2.putText(frame, label, (startX, startY - 10),
```

```
cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
    cv2.rectangle(frame,   (startX,   startY),   (endX,
endY), color, 2)

 cv2.imshow("Press q to quit", frame)

key = cv2.waitKey(1) & 0xFF

    if key == ord("q"):
        break

 cv2.destroyAllWindows()
 vs.stop()
```

Prints the predictions array when the loop ends.

```
print("Predictions:", predictions)
```

User Interface:

```
class WebcamApp:
    def __init__(self, window):
        self.window = window
        self.window.title("Eleven")
        self.window.configure(bg="#596174")  # Set background color
        self.video_capture = None
        self.current_image = None
        self.canvas = tk.Canvas(window, width=640, height=480, bg="#596174")  # Set canvas background color
        self.canvas.pack()

        # Start Video button
        self.start_button = tk.Button(window, text="Start camera", command=self.start_video)
        self.start_button.pack(pady=10)  # Add padding (space) below the button

        # Stop Video button
        self.stop_button = tk.Button(window, text="Stop camera", command=self.stop_video, state=tk.DISABLED)
        self.stop_button.pack(pady=10)  # Add padding (space) below the button

        # Detect button
        self.detect_button = tk.Button(window, text="Mask Detection", command=self.detect_masks, state=tk.DISABLED)
        self.detect_button.pack(pady=10)  # Add padding (space) below the button

        # New Face Detection button
        self.face_detection_button = tk.Button(window, text="Face Detection", command=self.detect_faces, state=tk.DISABLED)
        self.face_detection_button.pack(pady=10)  # Add padding (space) below the button

        self.is_video_running = False
        self.paused = False
        self.predictions = []

        # Load face detection model
        self.faceNet = cv2.dnn.readNet("deploy.prototxt", "res10_300x300_ssd_iter_140000.caffemodel")
        self.maskNet = load_model("Mask.h5")

        # Load face recognition model
        self.face_recognition_model = load_model('NotunModel.h5')
        self.known_embeddings = np.load('known_embeddings.npy')
        self.known_labels = np.load('known_labels.npy')
        self.label_to_name = {0: 'Abeer', 1: 'Ashraf', 2: 'Ikrama', 3: 'Ishti', 4: 'Nazmul', 5: 'Others', 6:'Sadat', 7: 'Shera'}
        self.neighbors = NearestNeighbors(n_neighbors=1, algorithm='auto', metric='cosine')
        self.neighbors.fit(self.known_embeddings)
```

The provided code defines a WebcamApp class for a graphical user interface (GUI) application that manages webcam operations, mask detection, and face recognition. The GUI, created using Tkinter, features buttons for starting and stopping the camera, initiating mask detection, and triggering face detection. The class handles the integration of pre-trained models for face detection, mask detection, and face recognition. Specifically, it utilizes a faceNet model for face detection, a maskNet model for mask detection, and a custom face recognition model. The GUI allows users to seamlessly interact with the webcam, enabling functionalities such as mask detection, face recognition, and real-time display updates. The code also loads pre-trained models and necessary data for face recognition, providing a comprehensive and user-friendly interface for webcam-related tasks.

```
def start_video(self):
    if not self.is_video_running:
        self.video_capture = cv2.VideoCapture(0)
        self.is_video_running = True
        self.start_button.config(state=tk.DISABLED)
        self.stop_button.config(state=tk.NORMAL)
        self.detect_button.config(state=tk.NORMAL)
        self.face_detection_button.config(state=tk.NORMAL)
        self.update_webcam()

def stop_video(self):
    if self.is_video_running:
        self.is_video_running = False
        self.start_button.config(state=tk.NORMAL)
        self.stop_button.config(state=tk.DISABLED)
        self.detect_button.config(state=tk.DISABLED)
        self.face_detection_button.config(state=tk.DISABLED)

        # Release the video capture object to stop the video
        if self.video_capture:
            self.video_capture.release()
```

The provided code defines methods within a WebcamApp class to control the video capture functionality of a graphical user interface (GUI) application. The start_video method initiates the webcam capture, disabling the "Start" button and enabling buttons for functionalities like stopping the video, detecting masks, and face recognition. It also triggers the continuous update of the Tkinter canvas with the live webcam feed. On the other hand, the stop_video method gracefully halts the video capture, releasing associated resources, and updating button states to reflect the stopped state. These methods collectively orchestrate the start and stop mechanisms for the webcam, enhancing the user interface's control over video-related operations within the GUI.

```
def update_webcam(self):
    if self.is_video_running:
        ret, frame = self.video_capture.read()

        if ret:
            self.current_image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
            self.photo = ImageTk.PhotoImage(image=self.current_image)
            self.canvas.create_image(0, 0, image=self.photo, anchor=tk.NW)

        if not self.paused:
            self.window.after(15, self.update_webcam)
```

The provided code defines the update_webcam method within a WebcamApp class, responsible for maintaining a real-time display of the webcam feed on the Tkinter canvas. While the video is running, the method continuously captures frames from the webcam using the OpenCV library, converting each frame to an ImageTk.PhotoImage format. This processed image is then displayed on the Tkinter canvas. The method utilizes the Tkinter after method to create a loop, ensuring a constant refresh rate of the webcam feed at approximately 15 milliseconds. Additionally, the method incorporates a check for a paused state, allowing users to temporarily halt the update process by toggling a pause flag. Overall, the update_webcam method provides a seamless and responsive visual experience, continuously updating the GUI with the live webcam feed.

```
def detect_masks(self):
    if self.is_video_running:
        # Disable buttons during detection
        self.start_button.config(state=tk.DISABLED)
        self.stop_button.config(state=tk.DISABLED)
        self.detect_button.config(state=tk.DISABLED)
        self.face_detection_button.config(state=tk.DISABLED)

        # Run detection in a separate thread
        Thread(target=self.run_detection).start()
```

The presented code snippet is part of a method called detect_masks within a WebcamApp class, designed for initiating mask detection in the GUI

application. If the video is currently running, the method temporarily disables relevant buttons, including those for starting and stopping the camera, mask detection, and face detection, preventing interference during the detection process. Subsequently, it triggers the actual mask detection logic, encapsulated in the run_detection method, to execute in a separate thread. This approach ensures that mask detection operates independently in the background, enhancing the responsiveness and uninterrupted functionality of the graphical user interface.

```
def run_detection(self):
    self.predictions = []

    while True:
        if not self.paused:
            ret, frame = self.video_capture.read()

        if not ret:
            break

        frame = imutils.resize(frame, width=900)
        (locs, preds) = self.detect_and_predict_mask(frame, self.faceNet, self.maskNet)

        for (box, pred) in zip(locs, preds):
            (startX, startY, endX, endY) = box
            (mask, withoutMask) = pred
            label = "Mask" if mask > withoutMask else "No Mask"
            color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

            if label == "Mask":
                self.predictions.append(1)  # Store 1 for mask
                print("ACCESS GRANTED")
                self.paused = True  # Pause the video feed when a mask is detected
            else:
                self.predictions.append(0)  # Store 0 for no mask
                print("ACCESS DENIED")

            label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
            cv2.putText(frame, label, (startX, startY - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
            cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

        cv2.imshow("Press q to quit", frame)

        key = cv2.waitKey(1) & 0xFF

        if key == ord("q"):
            break
        elif key == ord("p"):
            self.paused = not self.paused  # Toggle the paused state

            if not self.paused:
                self.predictions = []

    cv2.destroyAllWindows()
    self.video_capture.release()

    # Enable buttons after detection
    self.start_button.config(state=tk.NORMAL)
    self.stop_button.config(state=tk.DISABLED)
    self.detect_button.config(state=tk.NORMAL)
    self.face_detection_button.config(state=tk.NORMAL)

    # Print the predictions array when the loop ends
    print("Predictions:", self.predictions)
```

The provided code constitutes the run_detection method within a WebcamApp class, orchestrating the mask detection process in the GUI application. Operating in a loop, the method continuously captures frames from the webcam feed and resizes them for efficient processing.

Leveraging a pre-trained face detection model (self.faceNet) and a mask detection model (self.maskNet), it detects faces and predicts whether masks are worn. Detected faces are marked with rectangles, and the predictions are dynamically displayed on the GUI. If a mask is detected, access is granted with a corresponding message, and the video feed is paused. Conversely, if no mask is detected, access is denied. The method allows users to toggle the paused state with the 'p' key and exit the detection loop with the 'q' key. Upon loop termination, the GUI buttons are re-enabled, and the predictions array is printed for analysis. Overall, this method efficiently manages real-time mask detection, providing instant feedback and enhancing user control in the application.

```
def detect_and_predict_mask(self, frame, faceNet, maskNet):
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (256, 256), (104.0, 177.0, 123.0))
    faceNet.setInput(blob)
    detections = faceNet.forward()
    faces = []
    locs = []
    preds = []

    for i in range(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > 0.5:
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
            face = frame[startY:endY, startX:endX]
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (256, 256))
            face = img_to_array(face)
            face = preprocess_input(face)
            faces.append(face)
            locs.append((startX, startY, endX, endY))

    if len(faces) > 0:
        faces = np.array(faces, dtype="float32")
        preds = maskNet.predict(faces, batch_size=32)

    return (locs, preds)
```

The provided code defines the detect_and_predict_mask method within a WebcamApp class, responsible for processing a given frame to detect faces and predict whether masks are worn. Utilizing a pre-trained face detection model (faceNet) and a mask detection model (maskNet), the method extracts faces from the frame by evaluating the model's confidence in detecting facial features. Detected faces are then preprocessed, resized, and converted to an array for input into the mask detection model. The model predicts whether each detected face is wearing a mask, and the results, along with the corresponding face locations, are returned as tuples. This method effectively integrates face detection and mask prediction, providing crucial information for the real-time mask detection process in the GUI application.

```
def detect_faces(self):
    if self.is_video_running:
        # Disable buttons during face detection
        self.start_button.config(state=tk.DISABLED)
        self.stop_button.config(state=tk.DISABLED)
        self.detect_button.config(state=tk.DISABLED)
        self.face_detection_button.config(state=tk.DISABLED)

        # Run face detection in a separate thread
        Thread(target=self.run_face_detection).start()
```

The provided code is part of the detect_faces method within a WebcamApp class, designed to initiate face detection in the graphical user interface (GUI) application. If the video is currently running, the method temporarily disables relevant buttons associated with camera control and other detection functionalities to prevent user interference during the face detection process. Subsequently, it triggers the actual face detection logic, encapsulated in the run_face_detection method, to execute in a separate thread. This concurrent execution ensures that face detection operates independently in the background, enhancing the responsiveness and uninterrupted functionality of the GUI.

```python
def run_face_detection(self):
    self.predictions = []

    while True:
        if not self.paused:
            ret, frame = self.video_capture.read()

        if not ret:
            break

        frame = imutils.resize(frame, width=900)
        faces = self.detect_faces_mtcnn(frame)

        for face in faces:
            x, y, w, h = face['box']
            roi = frame[y:y + h, x:x + w]

            # Preprocess the face image
            preprocessed_face = cv2.resize(roi, (224, 224)) / 255.0
            preprocessed_face = np.expand_dims(preprocessed_face, axis=0)

            # Obtain face embeddings for the detected face
            face_embedding = self.face_recognition_model.predict(preprocessed_face)

            # Perform approximate nearest neighbor search
            distances, indices = self.neighbors.kneighbors(face_embedding, n_neighbors=1)
            similarity_score = 1 - distances[0][0]
            confidence_threshold = 0.6

            # Check if similarity score exceeds the threshold
            if similarity_score > confidence_threshold:
                closest_label = self.known_labels[indices[0][0]]

                # Retrieve the name associated with the label
                recognized_name = self.label_to_name.get(closest_label, 'Unknown')

                # Display the recognized name on the frame
                cv2.putText(frame, f'Recognized: {recognized_name} ({similarity_score:.2f})',
                            (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2, cv2.LINE_AA)
            else:
                # Display an "Unknown" label if confidence is below the threshold
                cv2.putText(frame, 'Unknown', (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2,
                            cv2.LINE_AA)

        cv2.imshow('Real-Time Facial Recognition', frame)

        key = cv2.waitKey(1) & 0xFF

        if key == ord("q"):
            break
        elif key == ord("p"):
            self.paused = not self.paused  # Toggle the paused state

    self.video_capture.release()
    cv2.destroyAllWindows()
```

```python
# Enable buttons after face detection
self.start_button.config(state=tk.NORMAL)
self.stop_button.config(state=tk.DISABLED)
self.detect_button.config(state=tk.NORMAL)
self.face_detection_button.config(state=tk.NORMAL)
```

The provided code encapsulates the run_face_detection method within a WebcamApp class, orchestrating real-time face detection and recognition in the graphical user interface (GUI) application. Operating in a continuous loop while considering a paused state, the method captures frames from the webcam feed, resizes them for optimal processing, and employs the MTCNN model to detect faces. For each detected face, the method pre-processes the image and extracts face embeddings using a

pre-trained face recognition model. Subsequently, it performs an approximate nearest neighbor search, comparing the embeddings with known faces, and displays the recognized names on the video feed if the similarity score surpasses a specified threshold. The GUI allows users to toggle the paused state with the 'p' key and exit the recognition loop with the 'q' key. Upon loop termination, the GUI buttons are re-enabled, providing a seamless and interactive experience for real-time facial recognition within the application.

```
def detect_faces_mtcnn(self, frame):
    detector = MTCNN()
    faces = detector.detect_faces(frame)
    return faces
```

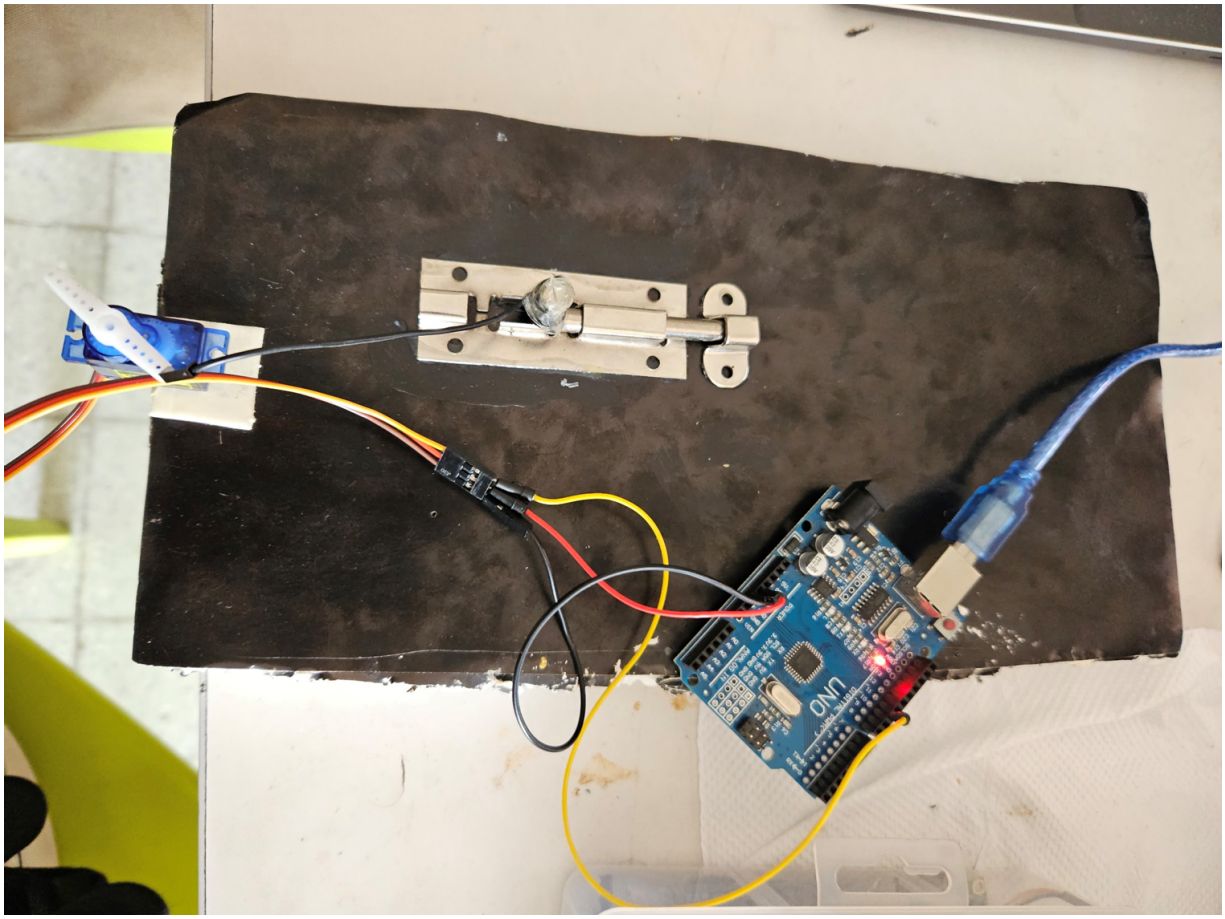
The presented code defines the detect_faces_mtcnn method within a WebcamApp class, leveraging the MTCNN (Multi-Task Cascaded Convolutional Networks) model for face detection in a given frame. The method initializes an instance of the MTCNN detector and applies it to the input frame, identifying faces along with their bounding box coordinates. The detected face information, represented as dictionaries containing box coordinates, confidence scores, and facial landmarks, is then returned. This method encapsulates the functionality of using the MTCNN model to precisely locate faces in the provided frame, forming a critical component of the real-time face detection process within the graphical user interface (GUI) application.

Hardware Implementation:

To implement this project in hardware, we used arduino and servo motor. Servo motor is connected with door lock. To control the servo motor, Standard Firmata of arduino was uploaded and we wrote a controller code in python. In controller code, two functions were defined, rotateServo(pin,angle) to rotate the motor at a certain angle and another is doorAutomate(val) where for 0 value of 'val', rotateServo() function is called to rotate to open the door and for 1 value of 'val', rotateServo is called to rotate and close the door. Later, this controller code was imported in main code to GUI. In the maincode we checked for the mask and facial recognitions predictions. If both return 1, then at press of a button it will call our rotateServo() function.

```
elif key == ord("e"):
    with open(self.prediction_file_path, "w") as file:
        file.write(str(self.current_prediction) + "\n")
    path1="mask_predictions.txt"
    path2=self.prediction_file_path
    predicts=[]
    with open(path1, 'r') as file1:
        predicts.append(int(file1.read().strip()))
    with open(path2, 'r') as file2:
        predicts.append(int(file2.read().strip()))
    print(predicts)
    if sum(predicts) == 2:
        doorAutomate(0)
        time.sleep(10)
        doorAutomate(1)
```

Hardware Setup:



Problems Faced

Learning all the prerequisite knowledge in limited time wasn't easy. We faced many errors while running the code and had to troubleshoot step by step to correct the error. We also faced difficulties collecting a diversified dataset. We also faced another difficulty in hardware implementation as opening a door is easily done by our project but to close the door, a forward force is needed which we couldn't give using wire.

Limitations and Future Work

It doesn't show 100% accurate output. So we aim to modify this project in future. Modification can be done in training models by using even more image datasets and a complex model to increase our accuracy. Our system can only unlock the door but cannot lock the door as it needs some force to lock the door which cannot be provided by the servo motor. Also there are some functions that we have to do mannually. We aim to use an improved design that can both open and close the door and make the whole system automatic.

Conclusion

Besides some errors, our project runs successfully. With further improvements and enough time, we believe we can develop our project that can give really secure entries to confidential places and so on.

Learning Resources

1. Learning Python Programming

   Link

2. Learning OpenCV Library

   OpenCV Course - Full Tutorial with Python

3. Learning TensorFlow

   Link

Codes And Outputs:

Drive Link containing Codes:

https://drive.google.com/drive/folders/1y3x5F5eGbu3HDIF-YrX-4Qdu0B1FMyz4?usp=drive_link

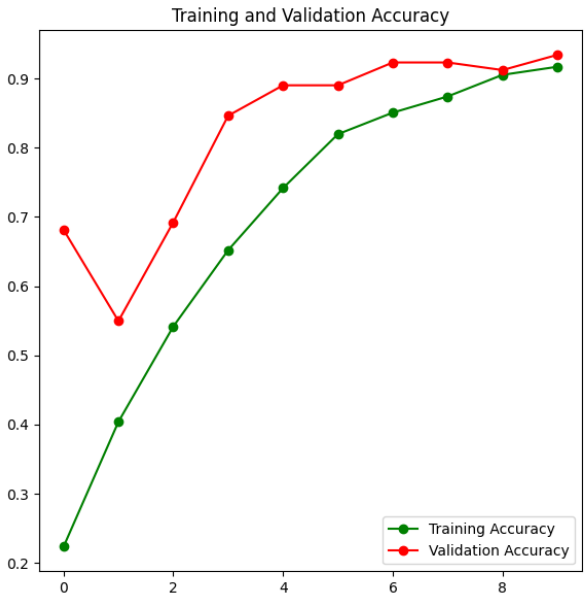
Facial Recognition Model Training Output:

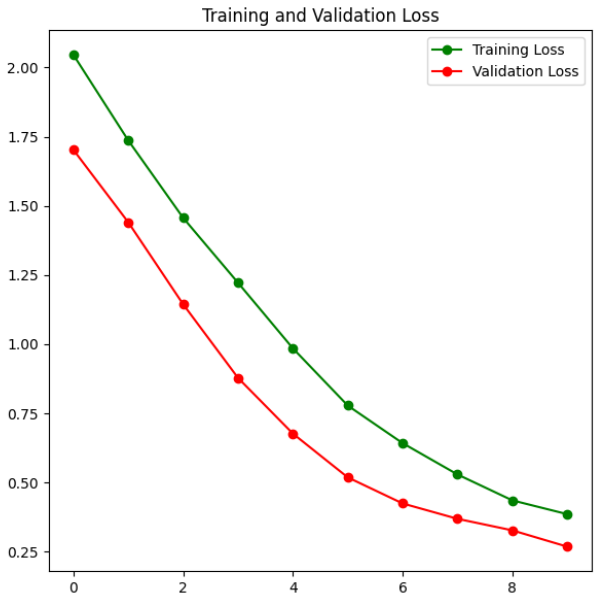
Class Labels:

```
['Abeer', 'Ashraf', 'Ikrama', 'Ishti', 'Nazmul', 'Others', 'Sadat', 'Shera']
```

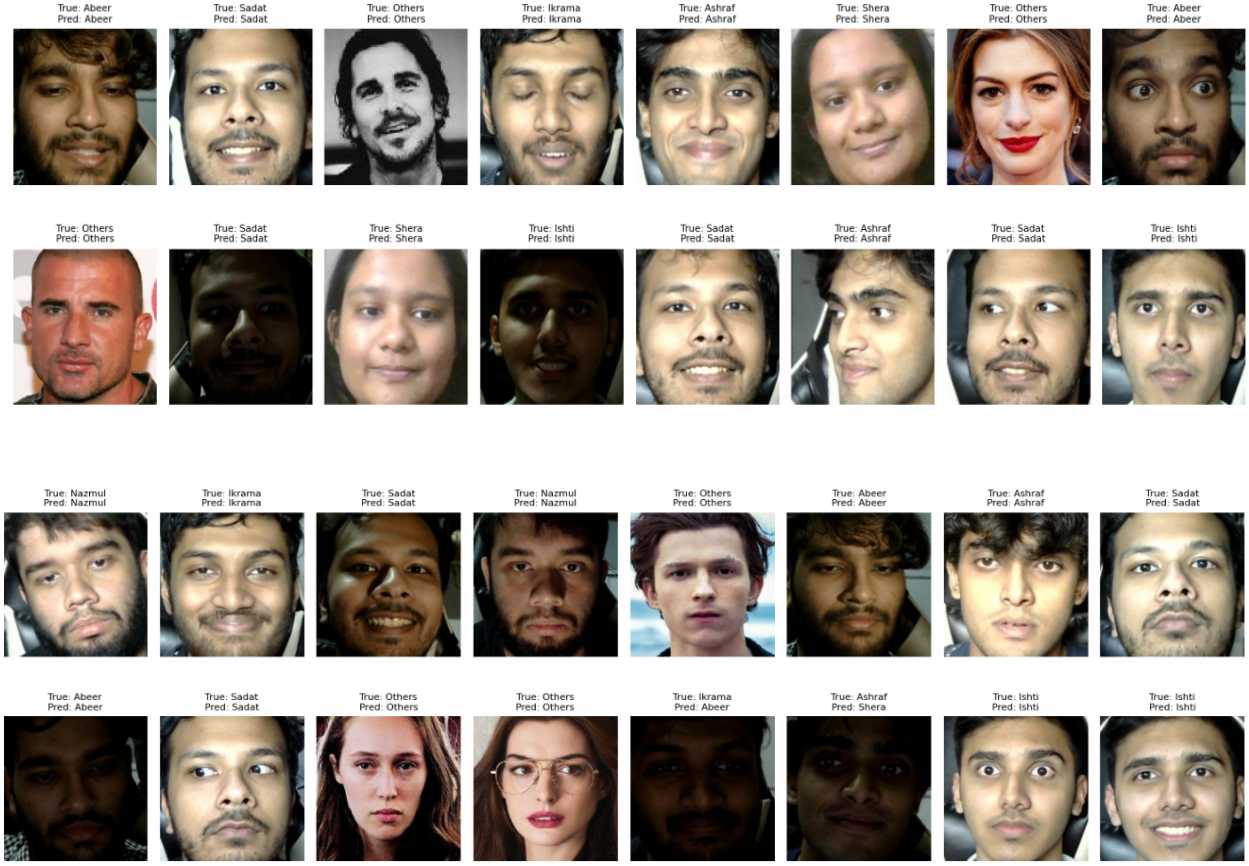
Training Results:

```
Epoch 1/10
WARNING:tensorflow:From C:\Progs\Python\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is
WARNING:tensorflow:From C:\Progs\Python\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_
25/25 [==============================] - 93s 4s/step - loss: 2.0458 - accuracy: 0.2234 - val_loss: 1.7024 - val_accuracy: 0.6813
Epoch 2/10
25/25 [==============================] - 112s 4s/step - loss: 1.7363 - accuracy: 0.4039 - val_loss: 1.4404 - val_accuracy: 0.5495
Epoch 3/10
25/25 [==============================] - 106s 4s/step - loss: 1.4565 - accuracy: 0.5416 - val_loss: 1.1441 - val_accuracy: 0.6923
Epoch 4/10
25/25 [==============================] - 115s 5s/step - loss: 1.2219 - accuracy: 0.6519 - val_loss: 0.8780 - val_accuracy: 0.8462
Epoch 5/10
25/25 [==============================] - 127s 5s/step - loss: 0.9848 - accuracy: 0.7416 - val_loss: 0.6766 - val_accuracy: 0.8901
Epoch 6/10
25/25 [==============================] - 122s 5s/step - loss: 0.7780 - accuracy: 0.8195 - val_loss: 0.5184 - val_accuracy: 0.8901
Epoch 7/10
25/25 [==============================] - 124s 5s/step - loss: 0.6421 - accuracy: 0.8506 - val_loss: 0.4242 - val_accuracy: 0.9231
Epoch 8/10
25/25 [==============================] - 126s 5s/step - loss: 0.5289 - accuracy: 0.8740 - val_loss: 0.3686 - val_accuracy: 0.9231
Epoch 9/10
25/25 [==============================] - 125s 5s/step - loss: 0.4353 - accuracy: 0.9052 - val_loss: 0.3269 - val_accuracy: 0.9121
Epoch 10/10
25/25 [==============================] - 126s 5s/step - loss: 0.3855 - accuracy: 0.9169 - val_loss: 0.2680 - val_accuracy: 0.9341
```





Predictions:

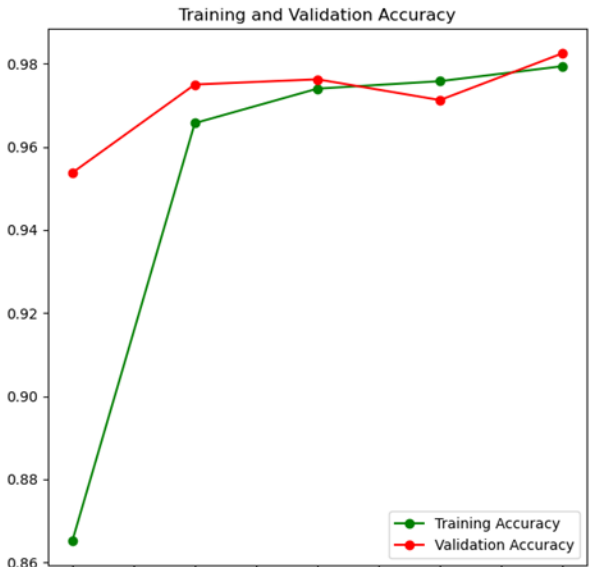


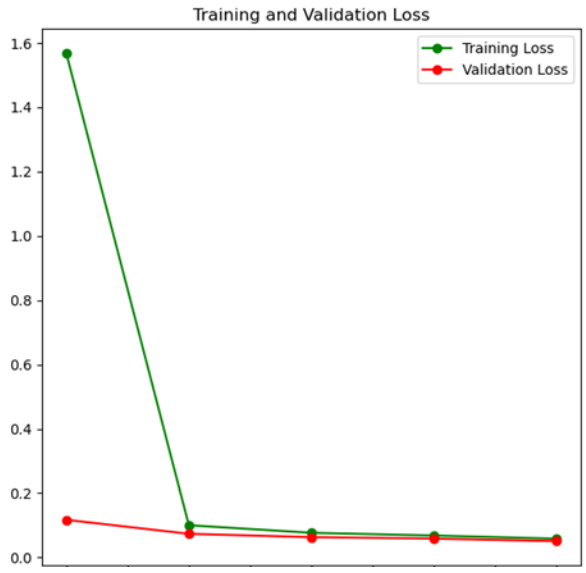
Known Embeddings:

```
[[8.1248420e-01 2.2047563e-03 1.7507198e-01 ... 1.7747637e-03
  2.5815435e-03 2.2553660e-04]
 [7.8534943e-01 1.4380630e-03 1.7587540e-01 ... 3.4041998e-03
  1.7977908e-02 5.6140823e-04]
 [7.9009485e-01 2.6484225e-02 1.1196383e-01 ... 5.6689936e-03
  1.0366131e-02 1.1396776e-02]
 ...
 [5.5335765e-04 1.0726976e-03 5.9796926e-03 ... 9.3663987e-03
  1.1270752e-03 9.7917193e-01]
 [5.5401208e-04 1.1735931e-02 1.2169433e-02 ... 9.2494681e-02
  2.8637769e-02 8.2684612e-01]
 [2.4780758e-05 2.3356079e-04 1.6247202e-03 ... 2.1045430e-02
  6.0613337e-04 9.7535115e-01]]
```

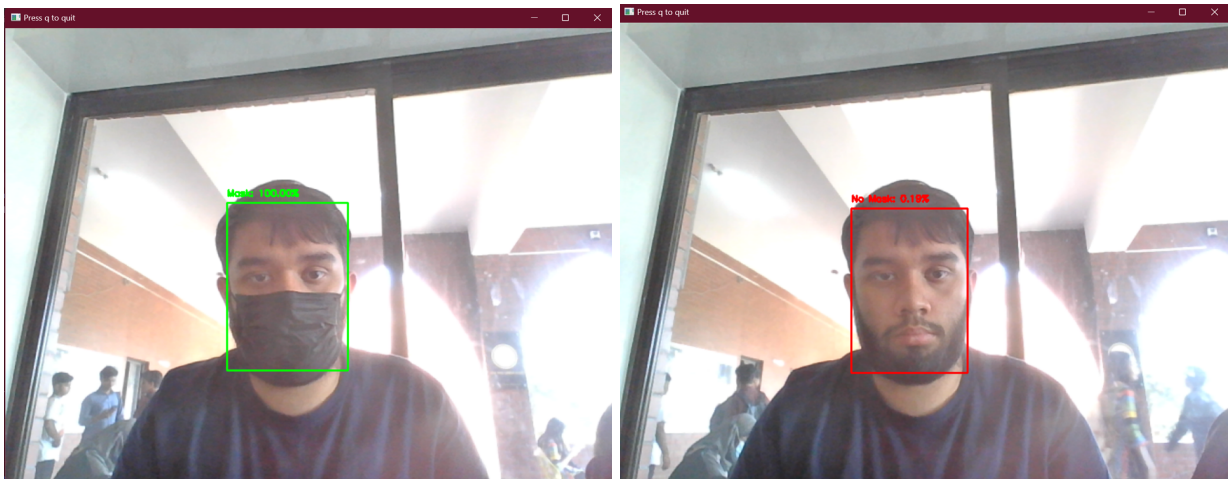
Real Time Face Recognition Output:

Mask Detection Model Training Result:





Real Time Mask Detection Output:



Output of GUI: