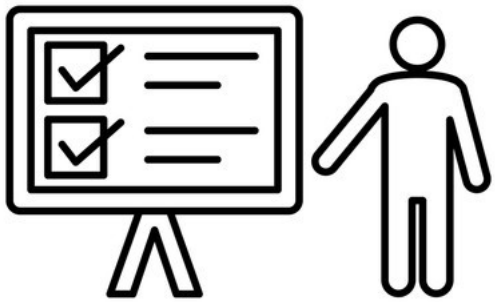
TABLE OF CONTENTS

PROBLEM STATEMENT



The objective of face recognition is, from the incoming image, to find a series of data of the same face in a set of training images in a database. The great difficulty is ensuring that this process is carried out in real-time, something that is not available to all biometric face recognition software providers.

The one in which, for the first time, a facial recognition system addresses a face to register it and associate it with an identity, in such a way that it is recorded in the system. This process is also known as digital onboarding with facial recognition.

The variant in which the user is authenticated, before being registered. In this process, the incoming data from the camera is crossed with the existing data in the database. If the face matches an already registered identity, the user is granted access to the system with his credentials.

The facial recognition system was created and compiled using PyCharm and visual Studio Code.

METHODOLGY/ PROCEDURE

We used dlib, one of Pythons open-source library. The following packages were also installed

PACKAGE	VERSION	PACKAGE	VERSION
Pillow	9.3.0	imageio	2.22.4
certifi	2020.6.20	imageio-ffmpeg	0.4.7
chardet	3.0.4	moviepy	1.0.3
charset-normalizer	2.1.1	numpy	1.23.5
click	7.1.2	opencv-python	4.6.0.66
cmake	3.18.2.post1	pip	21.3.1
colorama	0.4.6	proglog	0.1.10
decorator	4.4.2	requests	2.28.1
dlib	19.24.0	setuptools	60.2.0
face-recognition	1.3.0	tqdm	4.64.1
face-recognition-models	0.3.0	urllib3	1.26.13
idna	3.4	wheel	0.37.1

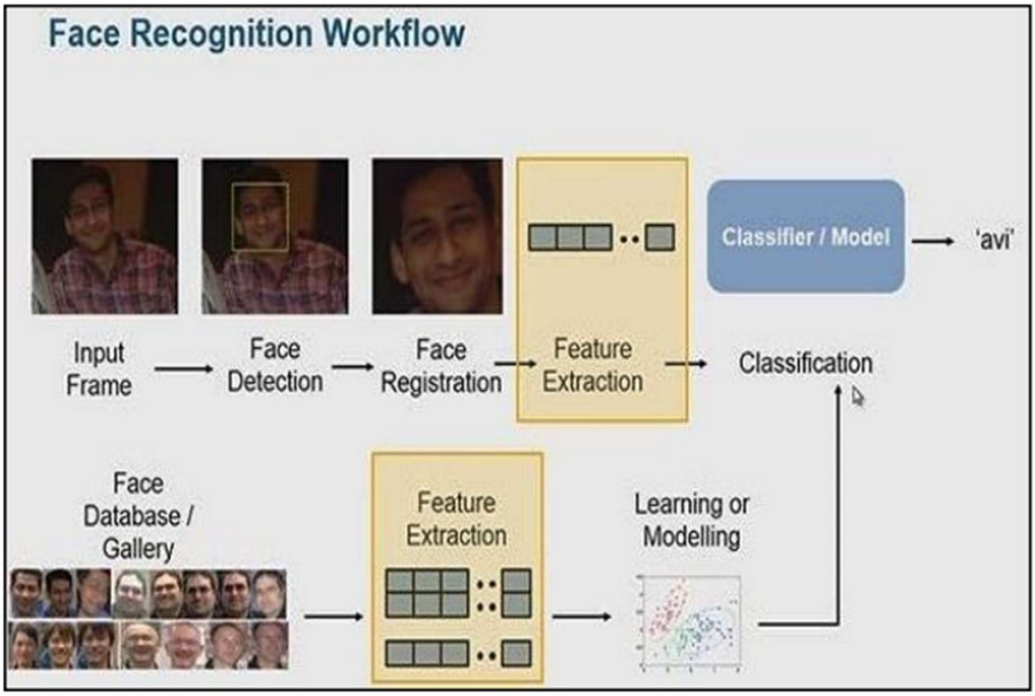


Figure 1: Face recognition workflow.

The program was created and compiled using PyCharm and Visual studio Code. The above packages were imported to code

CODE

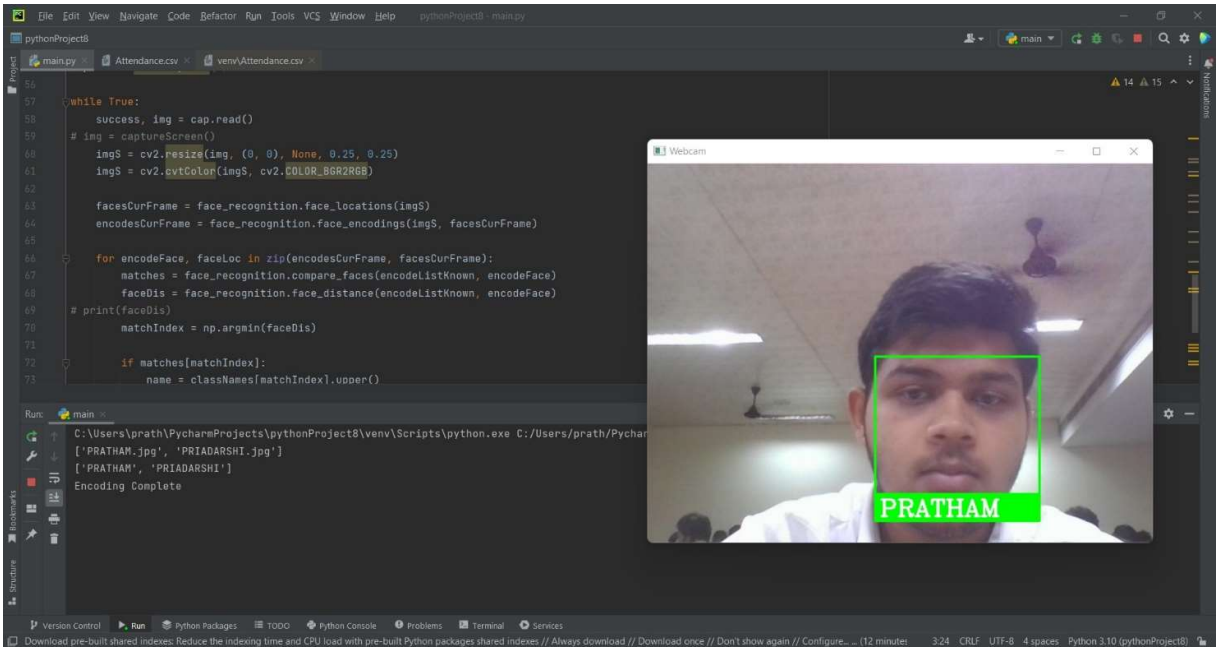
```python
import cv2
import numpy as np
import face_recognition
import os
from datetime import datetime

# from PIL import ImageGrab

path = 'Training_images'
images = []
classNames = []
myList = os.listdir(path)
print(myList)
for cl in myList:
    curImg = cv2.imread(f'{path}/{cl}')
    images.append(curImg)
    classNames.append(os.path.splitext(cl)[0])
print(classNames)


def findEncodings(images):
    encodeList = []


    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encode = face_recognition.face_encodings(img)[0]
        encodeList.append(encode)
    return encodeList

```

```python

def markAttendance(name):
    with open('Attendance.csv', 'r+') as f:
        myDataList = f.readlines()


        nameList = []
        for line in myDataList:
            entry = line.split(',')
            nameList.append(entry[0])
            if name not in nameList:
                now = datetime.now()
                dtString = now.strftime('%H:%M:%S')
                f.writelines(f'\n{name},{dtString}')

#### FOR CAPTURING SCREEN RATHER THAN WEBCAM
# def captureScreen(bbox=(300,300,690+300,530+300)):
#     capScr = np.array(ImageGrab.grab(bbox))
#     capScr = cv2.cvtColor(capScr, cv2.COLOR_RGB2BGR)
#     return capScr

encodeListKnown = findEncodings(images)
print('Encoding Complete')

cap = cv2.VideoCapture(0)

while True:
    success, img = cap.read()
# img = captureScreen()
    imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)
```

```
while True:
    success, img = cap.read()
# img = captureScreen()
    imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)
    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)

    facesCurFrame = face_recognition.face_locations(imgS)
    encodesCurFrame = face_recognition.face_encodings(imgS, facesCurFrame)

    for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):
        matches = face_recognition.compare_faces(encodeListKnown, encodeFace)
        faceDis = face_recognition.face_distance(encodeListKnown, encodeFace)
# print(faceDis)
        matchIndex = np.argmin(faceDis)

        if matches[matchIndex]:
            name = classNames[matchIndex].upper()
# print(name)
            y1, x2, y2, x1 = faceLoc
            y1, x2, y2, x1 = y1 * 4, x2 * 4, y2 * 4, x1 * 4
            cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.rectangle(img, (x1, y2 - 35), (x2, y2), (0, 255, 0), cv2.FILLED)
            cv2.putText(img, name, (x1 + 6, y2 - 6), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 2)
            markAttendance(name)

    cv2.imshow('Webcam', img)
    cv2.waitKey(1)
```

RESULT SHOWN BY PyCharm Integrated Development Environment (IDE)

RESULT



As seen, the webcam is used, a new window is opened which identifies the person and a green box covers the face highlighting the name as well.

CONCLUSION

Face recognition has many challenges due to illumination variations, large dimensionality, uncontrolled environments, aging and pose variations. In the recent years, Face recognition get remarkable improvement and accuracy to overcome these challenges, but matching in the heterogamous environment such as near infrared and visible spectrum is very challenging task. Matching of face images capture in near infrared spectrum (NIR) to face images of the visible spectrum (VIS) is a very challenging task. Recent research is categorized in three aspects such as face synthesis analysis, sub space methods, and local feature-based approaches.