FACE RECOGNITION FOR SECURITY USING PYTHON

SUBMITTED BY

S.NO	REG.NO	NAME OF THE STUDENT
1	21IT003	ACKSHAY KRISHNAN J
2	21IT016	JESUVINSANRAJ A
3	**21IT022**	**LOGESHVARADHAN S**
4	21IT015	MATHIVANAN V. S

Under the guidance of

Sri R. KULANDAIVEL MCS, M.PHIL.

Project submitted in partial fulfilment of the requirements for the award of

DIPLOMA IN INFORMATION TECHNOLOGY

Of State Board Of Technical Education And Training Department Of Technical Education, Tamil Nadu

2023- 2024

SRI RAMAKRISHNA MISSION VIDYALAYAPOLYTECHNIC COLLEGE

(An Autonomous and ISO 9001:2015 Certified institution)

Sri Ramakrishna Vidyalaya post, Coimbatore-641020

0422-2692432 Email: srkvtech56@gmail.com **Fax: (**0422**)** 2692582
Website: www.srkv.org

CERTIFICATE

SRI RAMAKRISHNA MISSION VIDYALAYAPOLYTECHNIC COLLEGE

(An Autonomous and ISO 9001:2015 Certified institution)

Sri Ramakrishna Vidyalaya post, Coimbatore-641020

DEPARTMENT OF INFORMATION TECHNOLOGY

This is to certify that it is a Bonafede record of work done by

Sri ……………………………..…… Reg. No: ………………..… in partial fulfilment of the requirements for the award of DIPLOMA IN INFORMATION TECHNOLOGY of Directorate of Technical Education, Chennai-25, during the academic year 2023 – 2024

…..………………………..	...................……..……..
Sri R. KULANDAIVEL	**Sri S. RAMESH**
Faculty Guide	**Head of the Department**

Submitted for the autonomous examination held on …………………………….. at Sri Ramakrishna Mission Vidyalaya Polytechnic College, Coimbatore-641020

…..………………………....	…..………………………
(INTERNAL EXAMINER)	**(EXTERNAL EXAMINER)**

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We express our deep sense of gratitude, indebtedness and whole hearted thanks to **Swami Tadbhasananda Maharaj**, Correspondent, Sri Ramakrishna Mission Vidyalaya Polytechnic College, for lab facilities made available in furnishing this project in our institution.

We are thankful to **Dr.S. Chandrasekaran, M.E., Ph.D Principal**, Sri Ramakrishna Mission Vidyalaya Polytechnic College, for providing us with the facilities necessary to pursue our project inside the college and to complete the work successfully.

We are in sense of gratitude to **Sri Ramesh MS(IT)., M.Phil., M.I.S.T.E.,** Head of the Department of information technology, Sri Ramakrishna Mission Vidyalaya Polytechnic College, for his valuable support and encouragement throughout the project.

Success in this work not have been possible without the constant encouragement of our guide, **Sri R. KULANDAIVEL MCS, M.PHIL.,** Lecturer, Department of Information Technology.

We extend our thanks to each and every faculty member of our department for their kind support and suggestion gives to us.

Abstract

Face Authentication and OTP based Control Login Page

Abstract:

This document presents a novel approach to enhance login security by combining face recognition and OTP-based authentication using Tkinter for the graphical user interface, OpenCV for face detection and recognition, and a Fast SMS API for OTP delivery. The system aims to provide an efficient and secure login process.

The login process begins with face recognition, where the user's face is captured through the webcam using OpenCV. If the face is recognized successfully, the system grants access to the user without OTP verification. However, in the event of a failed face recognition attempt, the system redirects the user to the login page.

To enhance security, an OTP is generated and sent to the user's registered mobile number using a Fast SMS API. The user is required to enter this OTP on the login page to gain access. This two-factor authentication (2FA) ensures that only authorized users can access their accounts, even in the event of a face recognition failure. In summary, the system combines biometric authentication with OTP-based security to offer a robust and user-friendly login solution. It provides an extra layer of security while maintaining a convenient and efficient user experience.

CONTENTS

CONTENTS

INTRODUCTION..**1**

METHODOLOGY OF THIS PROJECT..**3**

LITERATURE SURVEY..**6**

EXISTING OF THIS PROJECT..**10**

BIOMETRIC AUTHENTICATION WITH FACE RECOGNITION........**13**

FLOW DIAGRAM..**16**

REAL-TIME FACE RECOGNITION...**18**

TKINTER MODULE..**28**

THIS ARTICLE WILL COVER THE FOLLOWING TOPICS.................**33**

FAST SMS API DOCUMENTATION..**42**

CONCLUSION..**47**

MAINCODE..**49**

INTRODUCTION

INTRODUCTION

In today's digital age, ensuring secure access to personal accounts and sensitive information is of paramount importance. Traditional username and password-based authentication systems, while widely used, are susceptible to various security threats, such as password breaches, phishing attacks, and unauthorized access. To address these concerns, we present a novel solution that combines the power of face recognition and OTP-based authentication to create a robust and secure login process.

This project leverages modern technologies, including Tkinter for the graphical user interface, OpenCV for facial detection and recognition, and a Fast SMS API for OTP delivery. By merging these technologies, we aim to enhance the security of user authentication while maintaining user-friendliness and efficiency.

The primary goal of this project is to offer a multi-factor authentication approach. First, users' faces are recognized through their webcams, providing a seamless and intuitive login experience. However, recognizing the limitations of facial recognition technology, we've integrated OTP-based authentication as a fallback option. In the event of a failed face recognition attempt, the system redirects users to a login page where they must enter a one-time password (OTP) sent to their registered mobile number. This two-factor authentication (2FA) ensures that even if a potential attacker fails to pass the initial face recognition check, they cannot gain unauthorized access without the OTP.

This project addresses the need for enhanced security in user authentication processes, ultimately safeguarding user accounts and sensitive data from potential breaches. In the subsequent sections, we will delve deeper into the technical details of our solution, outlining the components, methodology, and how they work together to create a secure and user-friendly login system.

METHODOLOGY OF THIS PROJECT

Methodology of this project

User Registration and Face Enrolment

Users are required to register for the service, providing their personal information, including a mobile number.

During registration, users are prompted to enrols their facial features. Several facial images are captured and used for creating a unique face template.

Login Page with Face Recognition

Upon login, users are directed to the face recognition page. The OpenCV library is used to access the webcam and capture the user's face in real-time. The captured face is processed, and the system attempts to match it with the enrolled face template. If a successful match occurs, the user is granted access without OTP verification.

OTP Generation and Delivery

In cases where the face recognition check fails, an OTP is generated using a Fast SMS API. The OTP is sent to the user's registered mobile number for verification.

OTP-Based Login Page

Users redirected to the OTP-based login page are prompted to enter the OTP received on their mobile devices. If the entered OTP matches the one generated, access is granted.

Security and Error Handling

Comprehensive error handling is implemented to address various scenarios, such as face recognition failures and OTP mismatches. Security measures are in place to prevent unauthorized access and brute force attacks.

Database Integration

User data, including face templates and mobile numbers, is securely stored in a database. Databases are maintained and updated as users register or modify their information.

User Experience

The graphical user interface is designed using Tkinter to ensure a seamless and intuitive user experience. Users receive clear instructions and feedback throughout the login process, enhancing usability.

Logging and Monitoring

System activity is logged for monitoring and auditing purposes. Logs are used to track successful and failed login attempts and detect suspicious activities.

Fallback Mechanisms

In the rare event of both face recognition and OTP failure, users have the option to recover their accounts through a predefined recovery process.

Continuous Improvement

Regular updates and improvements to the system are planned to adapt to new security threats and enhance user convenience. By combining face recognition and OTP-based authentication, this project offers a secure and user-friendly login solution that balances convenience with robust security measures, protecting user accounts from unauthorized access and ensuring a positive user experience.

LITERATURE SURVEY

Literature survey

1. Authentication Using Face Recognition and OTP SMS-Token

The access control system is an important security feature to protect assets from being accessed illegally. One of the security techniques for access control is the use of biometrics, such as facial recognition, which is currently widely applied. The ease and uniqueness of everyone's face are the distinct advantages of this technique. However, like biometric factors in general, facial recognition also has a weakness against spoofing attacks where someone can cheat the system using an undercover method. In order to overcome these spoofing attacks, two factors are required to ensure the authenticity of the user. In this research, we designed and implemented a secure access control system based on two-factor authentication using facial recognition and SMS-token OTP to avoid spoofing attacks and to meet our own needs and avoid reliance on proprietary products. Based on the results of our research, the design scheme was successfully implemented so that it is secured to apply.

2. Design for Visitor Authentication Based on Face Recognition Technology Using CCTV

The Recently, image recognition technology using deep learning has improved significantly, and security systems and home services that use biometric information such as fingerprints, iris scans, and face recognition are attracting attention. User authentication methods that utilize face recognition have been studied at length. This study presents a visitor authentication technology that uses CCTV with a Jetson Nano and webcam. In the preprocessing phase for face recognition, face data with 7 features that can be identified as a person are collected using CCTV. The collected dataset goes through the annotation process to classify the data, and facial features are detected using deep learning. If there are four or more detected features, the image data is determined to be a person, and the visitor's face is matched with stored user data in detail using 81 feature vectors. Additionally, the security of the access control system was enhanced by implementing logging functions such as recording the face of the visitor, the number of visitors, and the time of the visit. This paper implements a visitor authentication system using a Jetson Nano and evaluates performance by analysing the accuracy and detection speed of the system. The tiny-YOLOv3 in the Jetson Nano was effective in real-time verification for the real-time face authentication system with an average detection speed of 6.5 FPS and 86.3% accuracy. Through this study, we designed a system based on deep learning technology that recognizes and authenticates the face of a user during the visitor access process and controls user access.

3. A Frictionless and Secure User Authentication in Web-Based Premium Applications

By and large, authentication systems employed for web-based applications primarily utilize conventional username and password-based schemes, which can be compromised easily. Currently, there is an evolution of various complex user authentication schemes based on the sophisticated encryption methodology. However, many of these schemes suffer from either low impact full consequences or offer security at higher resource dependence. Furthermore, most of these schemes don't consider dynamic threat and attack strategies when the clients are exposed to unidentified attack environments. Hence, this paper proposes a secure user authentication mechanism for web applications with a frictionless experience. An automated authentication scheme is designed based on user behaviour login events. The uniqueness of user identity is validated in the proposed system at the login interface, followed by implying an appropriate user authentication process. The authentication process is executed under four different login mechanisms,

which depend on the profiler and the authenticator function. The profiler uses user behavioural data, including login session time, device location, browser, and details of accessed web services. The system processes these data and generates a user profile via a profiler using the authenticator function. The authenticator provides login mechanism to the user to perform the authentication process. After successful login attempts, the proposed system updates database for future evaluation in the authentication process. The study outcome shows that the proposed system excels to other authentication schemes for an existing web-based application. The proposed method, when comparatively examined, is found to offer approximately a 10% reduction in delay, 7% faster response time, and 11% minimized memory usage compared with existing authentication schemes for premium web-based applications. INDEX TERMS Frictionless experience, internet.

4. **IoT-Based Biometric Recognition Systems in Education for Identity Verification Services: Quality Assessment Approach**

Traditional identity verification of students based on the human proctoring approach can cause a scam identity verification and ineffective processing time, particularly among vast groups of students. Most student identification cards outdated personal information. Several biometric recognitions approaches have been proposed to strengthen students' identity verification. Most educational adoption technologies struggle with evaluation and validation techniques to ensure that biometric recognition systems are unsuitable for utilization and implementation for student identity verification. This study presents the internet of things to develop flexible biometric recognition systems and an approach to assess the quality of biometric systems for educational use by investigating the effectiveness of identity verification of various biometric recognition technologies compared to the traditional verification method. The unimodal, multimodal, and semi-multimodal biometric technologies were tested using the developed internet of things-base biometric recognition systems examined by applying the proposed quality metrics of scoring factors based on accuracy, error rate, processing time, and cost. Hundreds of undergraduate exam takers were a sample group.

5.**Face Recognition-based Door Locking System with Two-Factor Authentication Using OpenCV**

This project develops a face recognition-based door locking system with two-factor authentication using OpenCV. It uses Raspberry Pi 4 as the microcontroller. Face recognition-based door locking has been around for many years, but most of them only provide face recognition without any added security features, and they are costly. The design of this project is based on human face recognition and the sending of a One-Time Password (OTP) using the Twilio service. It will recognize the person at the front door. Only people who match the faces stored in its dataset and then inputs the correct OTP will have access to unlock the door. The Twilio service and image processing algorithm Local Binary Pattern Histogram (LBPH) has been adopted for this system. Servo motor operates as a mechanism to access the door. Results show that LBPH takes a short time to recognize a face. Additionally, if an unknown face is detected, it will log this instance into a "Fail" file and an accompanying CSV sheet.

EXISTING OF THIS PROJECT

Existing Of This Project

User Registration and Face Enrolments

Users are required to register for the service, providing their personal information, including a mobile number. During registration, users are prompted to enrol their facial features. Several facial images are captured and used for creating a unique face template.

Login Page with Face Recognition

Upon login, users are directed to the face recognition page. The OpenCV library is used to access the webcam and capture the user's face in real-time. The captured face is processed, and the system attempts to match it with the enrolled face template. If a successful match occurs, the user is granted access without OTP verification.

OTP Generation and Delivery

In cases where the face recognition check fails, an OTP is generated using a Fast SMS API. The OTP is sent to the user's registered mobile number for verification.

OTP-Based Login Page:

Users redirected to the OTP-based login page are prompted to enter the OTP received on their mobile devices. If the entered OTP matches the one generated, access is granted.

Security and Error Handling:

Comprehensive error handling is implemented to address various scenarios, such as face recognition failures and OTP mismatches. Security measures are in place to prevent unauthorized access and brute force attacks.

Database Integration:

User data, including face templates and mobile numbers, is securely stored in a database. Databases are maintained and updated as users register or modify their information.

User Experience:

The graphical user interface is designed using Tkinter to ensure a seamless and intuitive user experience. Users receive clear instructions and feedback throughout the login process, enhancing usability.

Logging and Monitoring:

System activity is logged for monitoring and auditing purposes. Logs are used to track successful and failed login attempts and detect suspicious activities.

Fallback Mechanisms:

In the rare event of both face recognition and OTP failure, users have the option to recover their accounts through a predefined recovery process.

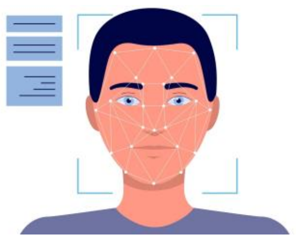
Continuous Improvement:

Regular updates and improvements to the system are planned to adapt to new security threats and enhance user convenience.

Proposed system of this project:

The proposed system aims to overcome the limitations of the existing traditional username and password-based authentication system by introducing a more secure and user-friendly login process that combines face recognition and OTP-based authentication. The key components and features of the proposed system are as follows:

13



BIOMETRIC WITH FACE RECOGNITION AUTHENTICATION

Biometric Authentication with Face Recognition

Users are required to enrol their facial features during registration. The system employs OpenCV for real-time face detection and recognition, ensuring secure and efficient access.

OTP-based authentication

In the event of a failed face recognition attempt, users are directed to an OTP-based login page. One-time passwords (OTPs) are generated and sent to users' registered mobile numbers via a Fast SMS API.

Multi-Factor Authentication (MFA):

The combination of face recognition and OTP-based authentication provides a robust two-factor authentication (2FA) mechanism, significantly enhancing security.

Secure User Data Storage:

User data, including facial templates and mobile numbers, are securely stored in a database, protecting sensitive information.

Graphical User Interface with Tkinter: The system is equipped with a user-friendly graphical interface developed using Tkinter, providing a seamless and intuitive user experience.

Enhanced Security Measures:

The system incorporates advanced security measures to prevent unauthorized access and mitigate potential threats.

Error Handling and Logging:

Comprehensive error handling is implemented, providing clear user feedback in case of login failures. System activity is logged to monitor login attempts and detect suspicious activities.

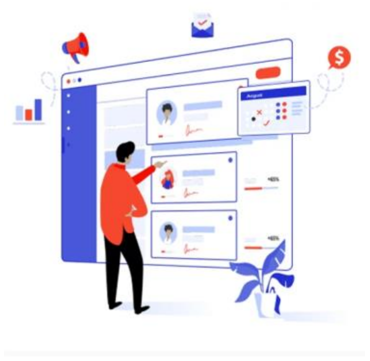
User Convenience and Account Recovery:

Users are provided with a fallback mechanism for account recovery in the rare event of both face recognition and OTP failure.

Continuous Improvement:

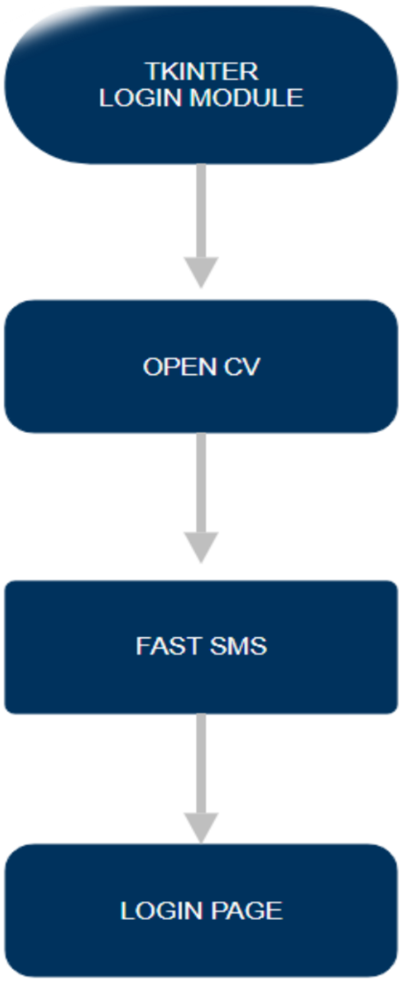
The system is designed to accommodate updates and enhancements to adapt to evolving security threats and user needs.

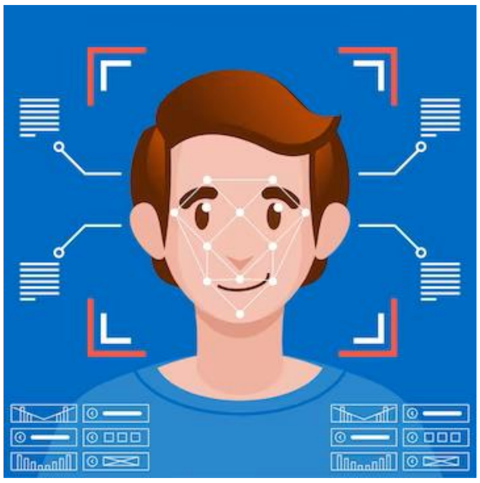
The proposed system offers a secure and convenient means of user authentication, ensuring that only authorized users can access their accounts. By combining the strength of face recognition with OTP-based authentication, it addresses the vulnerabilities of traditional authentication methods and provides an innovative and efficient solution to protect user accounts and sensitive data.



FLOW DIAGRAM

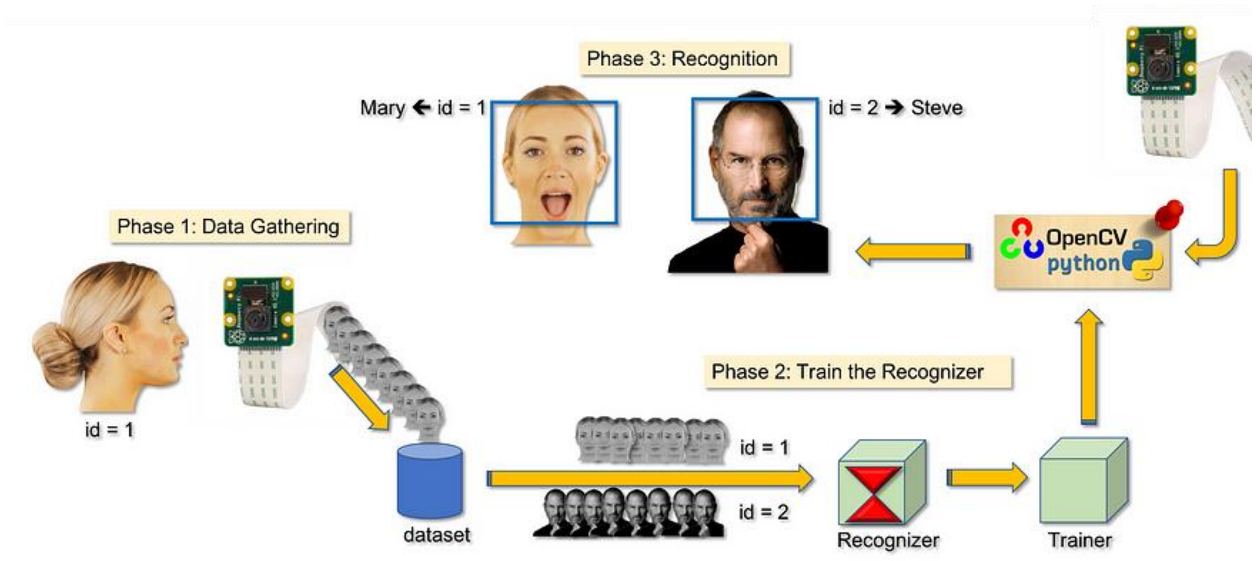
Flow Diagram





REAL-TIME FACE RECOGNITION

REAL-TIME FACE RECOGNITION



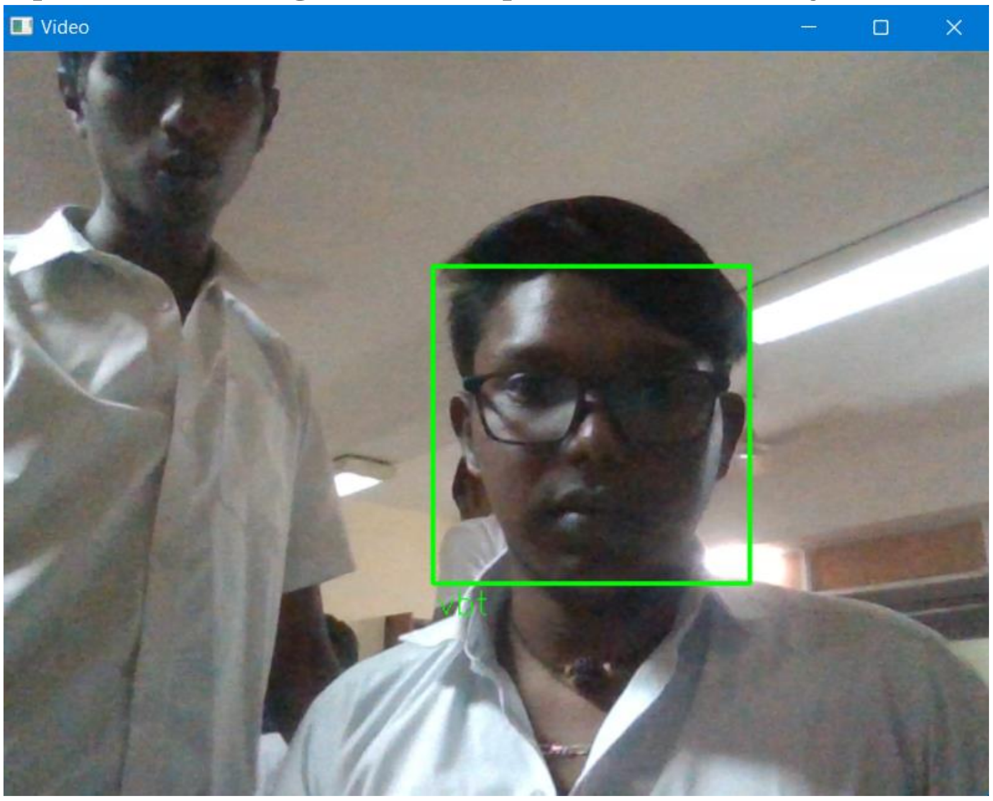
1. Introduction

On my tutorial exploring OpenCV, we learned AUTOMATIC VISION OBJECT TRACKING. Now we will use our PiCam to recognize faces in real-time, as you can see below:

This project was done with this fantastic "Open Source Computer Vision Library", the OpenCV. On this tutorial, we will be focusing on Raspberry Pi (so, Raspbian as OS) and Python, but I also tested the code on my Mac and it also works fine.

"To run it on a Mac, there is a couple of changes that should be made on code. Do not worry, I will comment about it"

OpenCV was designed for computational efficiency and with a strong



focus

The 3 Phases

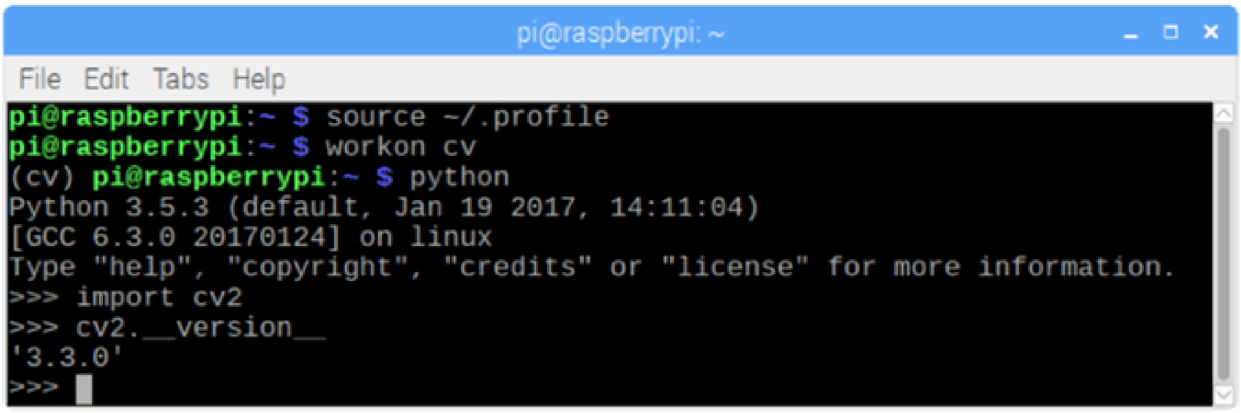
To create a complete project on Face Recognition, we must work on 3 very distinct phases:

- Face Detection and Data Gathering
- Train the Recognizer
- Face Recognition

2. Installing OpenCV 3 Package

I am using a Raspberry Pi V3 updated to the last version of Raspbian (Stretch), so the best way to have OpenCV installed, is to follow the excellent tutorial developed by Adrian Rosebrock: Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi.

that the cv Python virtual environment is entirely independent and sequestered from the default Python version included in the download of Raspbian Stretch. So, any Python packages in the global site-packages directory will not be available to the cv virtual environment. Similarly, any Python packages installed in site-packages of cv will not be available to the global install of Python.



```
pi@raspberrypi:~ $ source ~/.profile
pi@raspberrypi:~ $ workon cv
(cv) pi@raspberrypi:~ $ python
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.3.0'
>>> 
```

The above Terminal Print Screen shows the previous steps.

3. Testing Your Camera

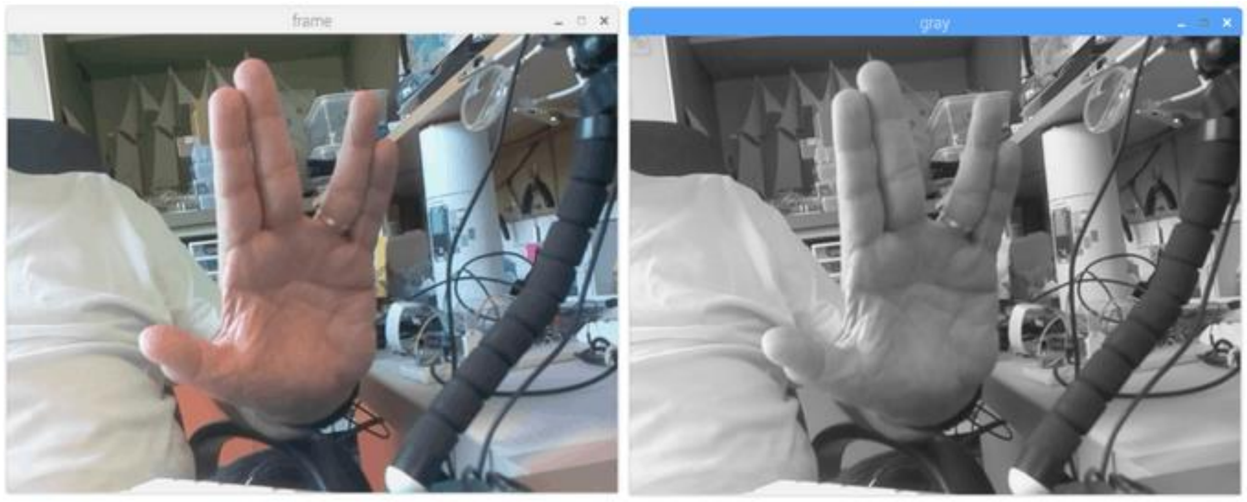
Once you have OpenCV installed in your RPi let’s test to confirm that your camera is working properly.

I am assuming that you have a PiCam already installed and enabled on your Raspberry Pi.

The above code will capture the video stream that will be generated by your PiCam, displaying both, in BGR colour and gray mode.

Note that I rotated my camera vertically due the way it is assembled. If it is not your case, comment or delete the “flip” command line.

To finish the program, you must press the key [ESC] on your keyboard. Click your mouse on the video window, before pressing [ESC].



The above picture shows the result.

Some people found issues when trying to open the camera and got “Assertion failed” error messages. That could happen if the camera was not enabled during OpenCv installation and so, camera drivers did not install correctly.

4. Face Detection

The most basic task on Face Recognition is of course, “Face Detecting”. Before anything, you must “capture” a face (Phase 1) in order to recognize it, when compared with a new face captured on future (Phase 3).

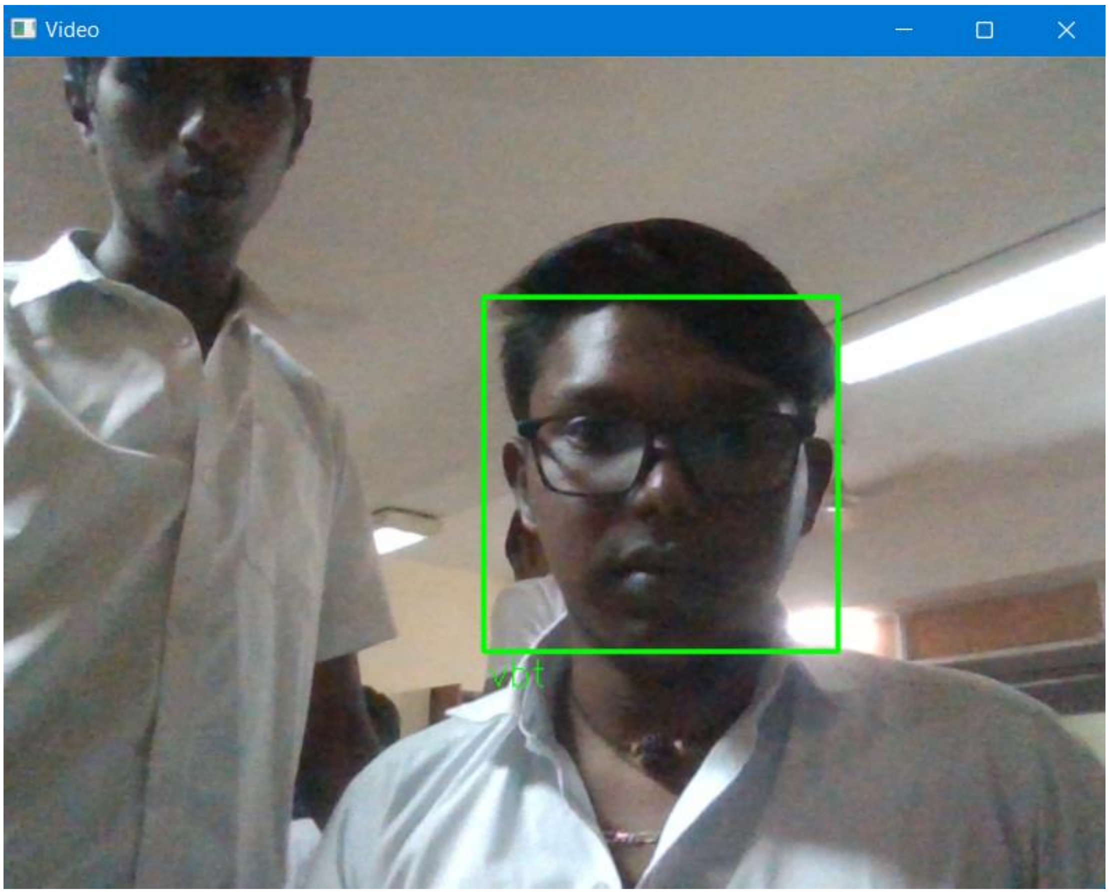
The most common way to detect a face (or any objects), is using the “Haar Cascade classifier”

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. The good news is that OpenCV comes with a trainer as well as a detector. If you want to train your own classifier for any object like car, planes etc. you can use OpenCV to create one. Its full details are given here: Cascade Classifier Training.

If you do not want to create your own classifier, OpenCV already contains many pre-trained classifiers for face, eyes, smile, etc. Those XML files can be download from harasses directory.

The result:



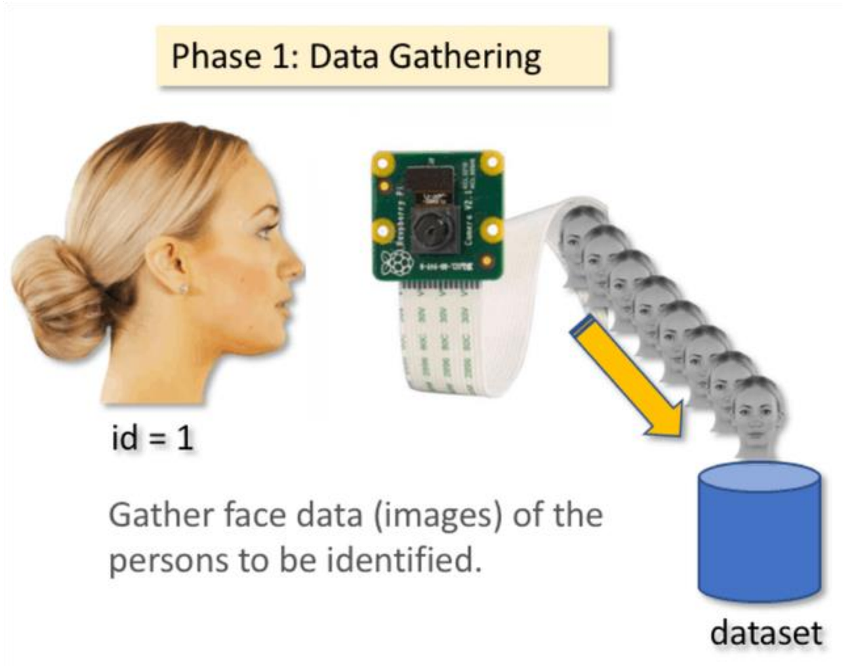
You can also include classifiers for “eyes detection” or even “smile detection”. On those cases, you will include the classifier function and rectangle draw inside the face loop, because would be no sense to detect an eye or a smile outside of a face.

Note that on a Pi, having several classifiers at same code will slow the processing, once this method of detection (Hackspaces) uses a great amount of computational power. On a desktop, it is easer to run it.

5. Data Gathering

First of all, I must thank Ramiz Raja for his great work on Face Recognition on photos.

Saying that, let’s start the first phase of our project. What we will do here, is starting from last step (Face Detecting), we will simply create a dataset, where we will store for each id, a group of photos in Gray with the portion that was used for face detecting.

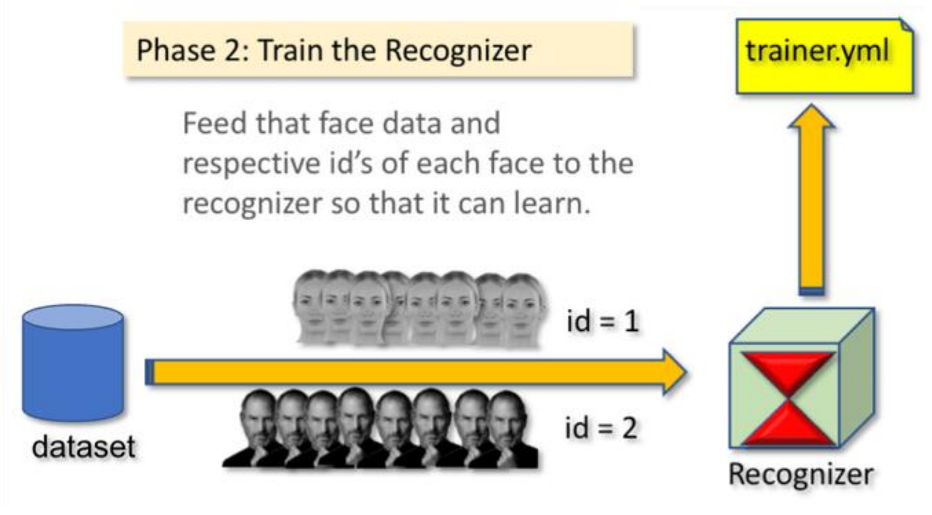


On my code, I am capturing 30 samples from each id. You can change it on the last “elfin”. The number of samples is used to break the loop where the face samples are captured.

Run the Python script and capture a few Ids. You must run the script each time that you want to aggregate a new user (or to change the photos for one that already exists).

6. Trainer

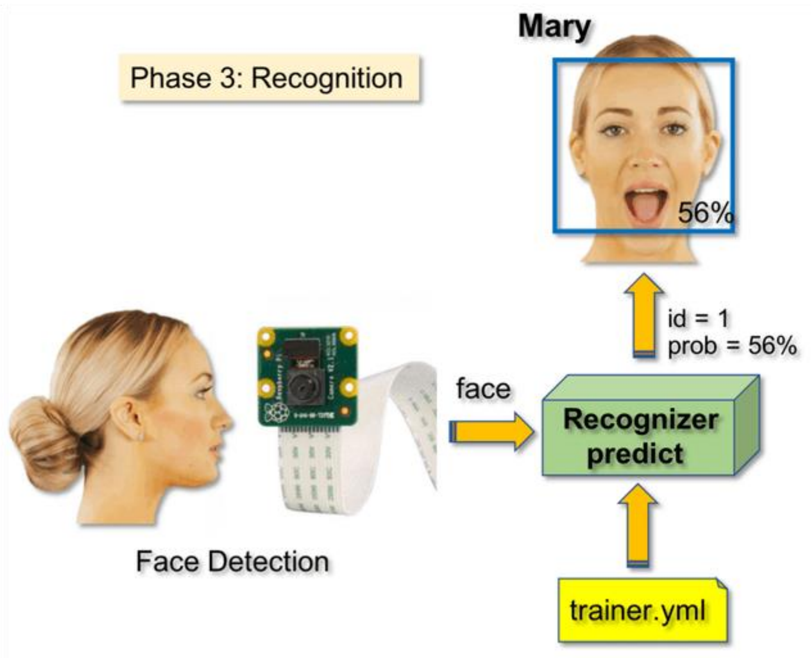
On this second phase, we must take all user data from our dataset and "trainer" the OpenCV Recognizer. This is done directly by a specific OpenCV function. The result will be a .yam file that will be saved on a "trainer/" directory.


Phase 2: Train the Recognizer
trainer.yml
Feed that face data and respective id's of each face to the recognizer so that it can learn.
id = 1
id = 2
dataset
Recognizer

So, let's start creating a subdirectory where we will store the trained as a result, a file named "trainer" will be saved in the trainer directory that was previously created by us. That's it! I included the last print statement where I displayed for confirmation, the number of User's faces we have trained. Every time that you perform Phase 1, Phase 2 must also be run.

7. Recognizer

Now, we reached the final phase of our project. Here, we will capture a fresh face on our camera and if this person had his face captured and trained before, our recognizer will make a "prediction" returning its id and an index, shown how confident the recognizer is with this match.



So, for example: Marcelo will the user with id = 1; Paula: id=2, etc. Next, we will detect a face, same we did before with the Haas Cascade classifier.

The recognizer. Predict (), will take as a parameter a captured portion of the face to be analysed and will return its probable owner, indicating its id and how much confidence the recognizer is in relation with this match.

Note that the confidence index will return "zero" if it will be considered a perfect match and at last, if the recognizer could predict a face, we put a text over the image with the probable id and how much is the "probability" in % that the match is correct ("probability" = 100 — confidence index). If not, an "unknow" label is put on the face.

On the picture, I show some tests done with this project, where I also have used photos to verify if the recognizer works.

TKINTER MODULE

TKINTER MODULE

Python is well known for its large set of libraries and extensions, each for different features, properties and use-cases. To handle PDF files, Python provides **PyPDF2** toolkit which is capable of processing, extracting, merging multiple pages, encrypting PDF files, and many more. It is a very useful Package for managing and manipulating the file streams such as PDFs. Using PyPDF2, we will create a Tkinter application that reads the PDF file by asking users to select and open a PDF file from the local directory.

To create the application, we will follow the steps given below −

- Install the requirement by typing
  pip install PyPDF2
  in the command Shell. Once installed, import the library in the notebook
  using **import Pypdf2** in Notebook.
- Import **file dialog** to create a dialog box for selecting the file from the local directory.
- Create a Text Widget and add some Menus to it like Open, Clear, and Quit.
- Define a function for each Menu.
- Define a function to open the file. In this function, first, we will read the file using PdfFileReader(file). Then, extract the pages from the file.
- Insert the content in the Text Box.
- Define the function for Quit Menu.

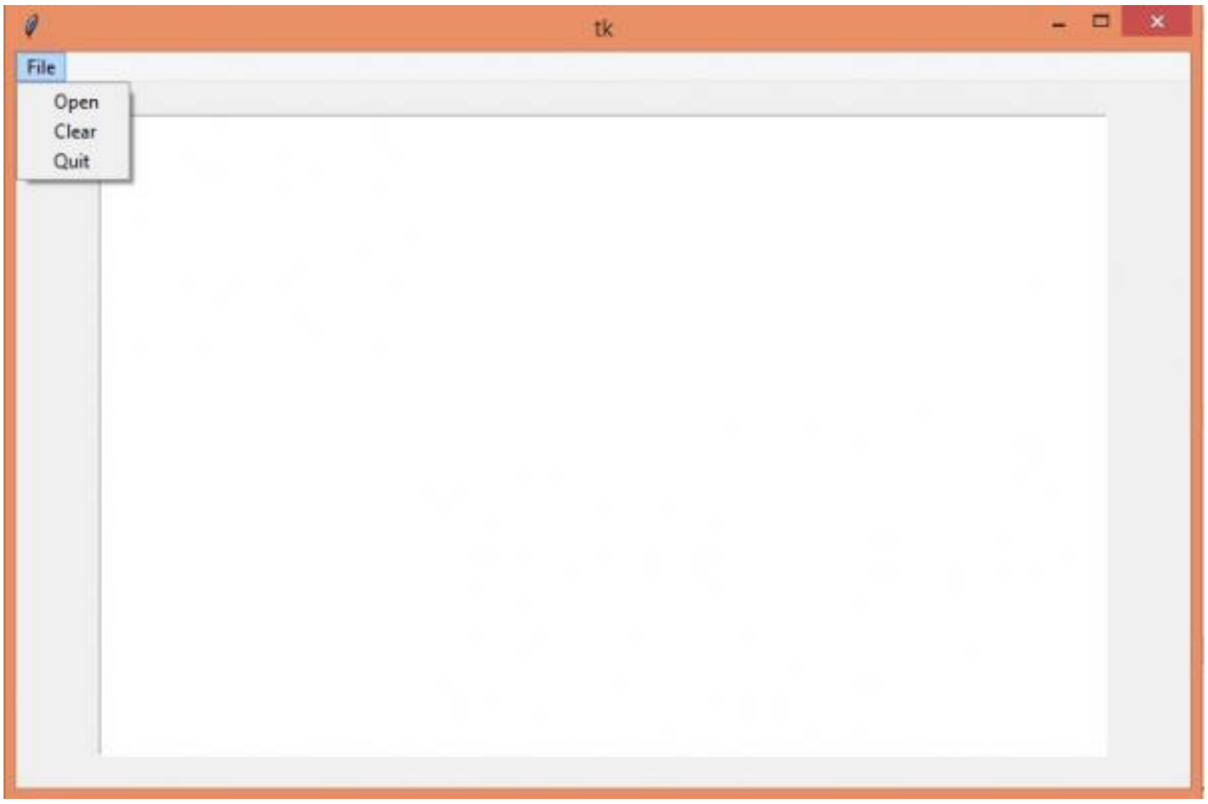
Example

```
#Import the required Libraries
import PyPDF2
from tkinter import *
from tkinter import filedialog
#Create an instance of tkinter frame
win= Tk()
#Set the Geometry
win.geometry("750x450")
#Create a Text Box
text= Text(win,width= 80,height=30)
text.pack(pady=20)
#Define a function to clear the text
def clear_text():
   text.delete(1.0, END)
#Define a function to open the pdf file
def open_pdf():
   file= filedialog.askopenfilename(title="Select a PDF",
filetype=(("PDF    Files","*.pdf"),("All Files","*.*")))
   if file:
      #Open the PDF File
      pdf_file= PyPDF2.PdfFileReader(file)
      #Select a Page to read
      page= pdf_file.getPage(0)
      #Get the content of the Page
      content=page.extractText()
      #Add the content to TextBox
      text.insert(1.0,content)

#Define function to Quit the window
def quit_app():
   win.destroy()
#Create a Menu
```

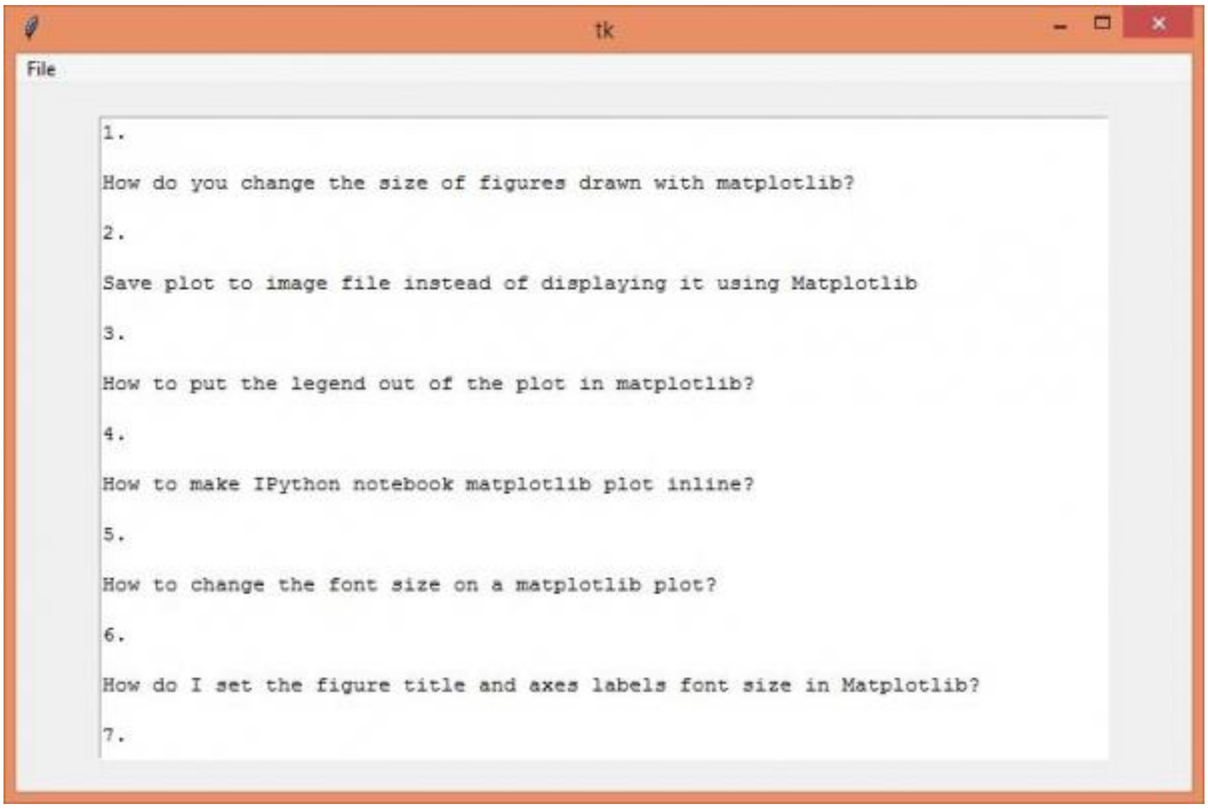
```
my_menu= Menu(win)
win.config(menu=my_menu)
#Add dropdown to the Menus
file_menu=Menu(my_menu,tearoff=False)
my_menu.add_cascade(label="File",menu= file_menu)
file_menu.add_command(label="Open",command=open_pdf)
file_menu.add_command(label="Clear",command=clear_text)
file_menu.add_command(label="Quit",command=quit_app)
win.mainloop()
```

Output

Running the above code will display a full-fledged tkinter application. It has functionalities of opening the file, clearing the file, and quit to terminate the application.



Click the "File" Menu on upper left corner of the application, open a new PDF File in the Text Box.



```
1.

How do you change the size of figures drawn with matplotlib?

2.

Save plot to image file instead of displaying it using Matplotlib

3.

How to put the legend out of the plot in matplotlib?

4.

How to make IPython notebook matplotlib plot inline?

5.

How to change the font size on a matplotlib plot?

6.

How do I set the figure title and axes labels font size in Matplotlib?

7.
```

This Article Will Cover

&

The Following Topics

This article will cover the following topics

1. Reading an image
2. Extracting the RGB values of a pixel
3. Extracting the Region of Interest (ROI)
4. Resizing the Image
5. Rotating the Image
6. Drawing a Rectangle
7. Displaying text

This is the original image that we will manipulate throughout the course of this article.

Let’s start with the simple task of reading an image using OpenCV.

Reading an image

```
# Importing the OpenCV library
importcv2
# Reading the image using misread () function
image =cv2.imread('image.png')

# Extracting the height and width of an image
h, w =image. Shape [:2]
# Displaying the height and width
print ("Height = {}, Width = {}”. format (h, w))
```

Now we will focus on extracting the RGB values of an individual pixel.

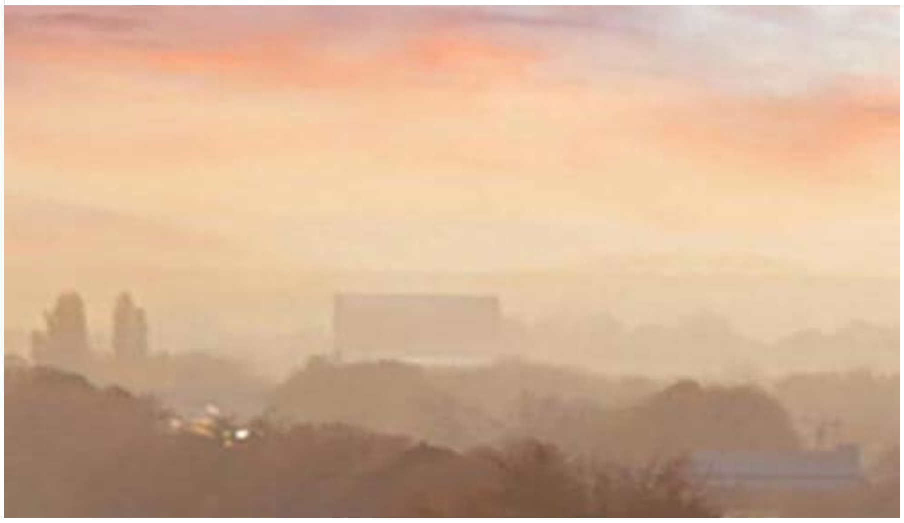
Note – OpenCV arranges the channels in BGR order. So the 0th value will correspond to Blue pixel and not Red.

Extracting the RGB values of a pixel

```
# Extracting RGB values.
# Here we have randomly chosen a pixel
# by passing in 100, 100 for height and width.
(B, G, R) =image[100, 100]

# Displaying the pixel values
print ("R = {}, G = {}, B = {}”. format (R, G, B))

# We can also pass the channel to extract
# The value for a specific channel
B =image [100, 100, 0]
print ("B = {}”. format(B))
```

Extracting the Region of Interest (ROI)

```
# We will calculate the region of interest
# by slicing the pixels of the image
Roi=image [100: 500, 200: 700]
```



Resizing the Image

```
# Resize () function takes 2 parameters,
# The image and the dimensions
resize =cv2.resize(image, (800, 800))
```

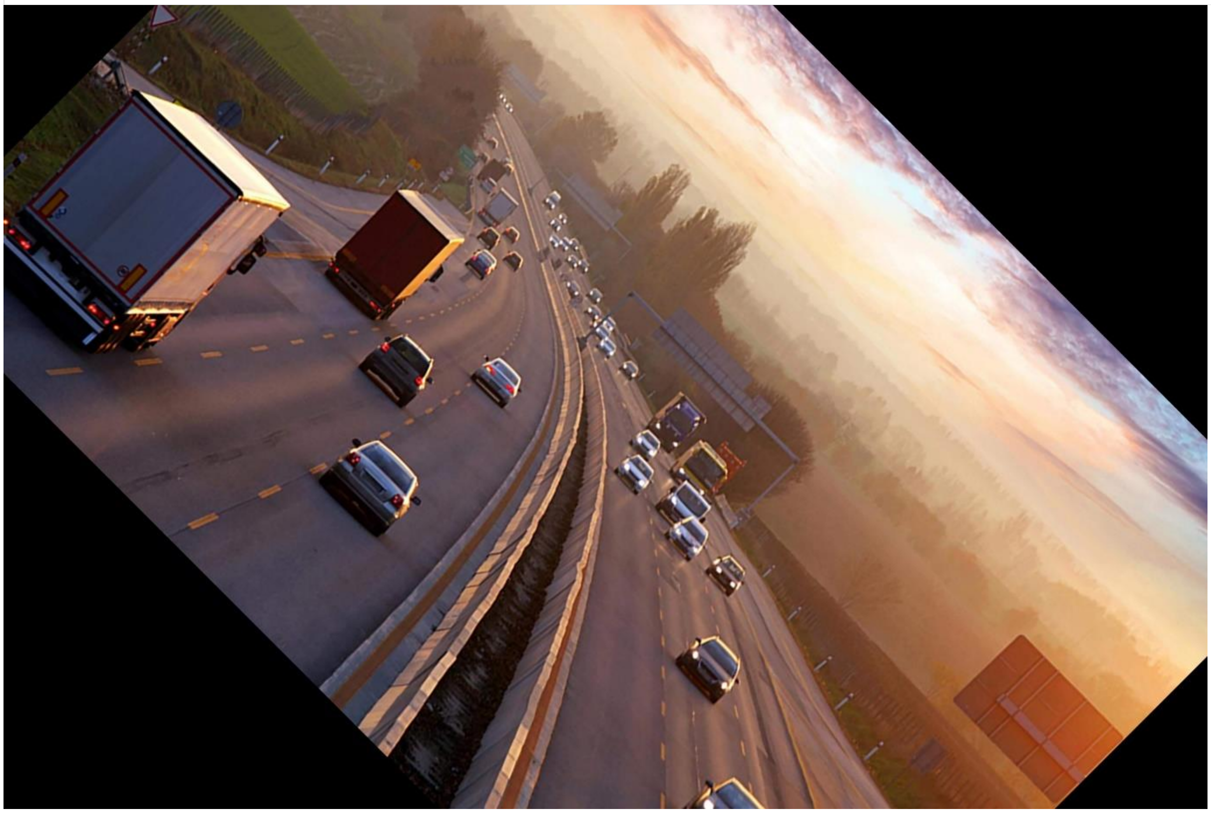
The problem with this approach is that the aspect ratio of the image is not maintained. So we need to do some extra work in order to maintain a proper aspect ratio.

```
# Calculating the ratio
ratio =800/w

# Creating a tuple containing width and height
dim = (800, int (h *ratio))

# Resizing the image
resize aspect=cv2.resize(image, dim)
```

Rotating the Image

```
# Calculating the canter of the image
canter=(w //2, h //2)

# Generating a rotation matrix
matrix =cv2.getRotationMatrix2D (canter, -45, 1.0)
```

```
# Performing the affine transformation
rotated =cv2.warpAffine(image, matrix, (w, h))
```



There are a lot of steps involved in rotating an image. So, let me explain each of them in detail.

The 2 main functions used here are –

- getRotationMatrix2D ()
- warp Affine ()

getRotationMatrix2D ()

It takes 3 arguments –

- **canter –** The canter coordinates of the image
- **Angle –** The angle (in degrees) by which the image should be rotated
- **Scale –** The scaling factor

It returns a 2*3 matrix consisting of values derived from alpha and beta alpha = scale * cos(angle)

beta = scale * sine(angle)

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot \texttt{center.x} - \beta \cdot \texttt{center.y} \\ -\beta & \alpha & \beta \cdot \texttt{center.x} + (1-\alpha)\cdot \texttt{center.y} \end{bmatrix}$$

warp Affine ()

The function warp Affine transforms the source image using the rotation matrix:

dist. (x, y) = sac(M11X + M12Y + M13, M21X + M22Y + M23)

Here M is the rotation matrix, described above.
It calculates new x, y co-ordinates of the image and transforms it.

Drawing a Rectangle

It is an in-place operation.

```
# We are copying the original image,
# as it is an in-place operation.
output =image.copy()

# Using the rectangle() function to create a rectangle.
rectangle =cv2.rectangle(output, (1500, 900),
                        (600, 400), (255, 0, 0), 2)
```

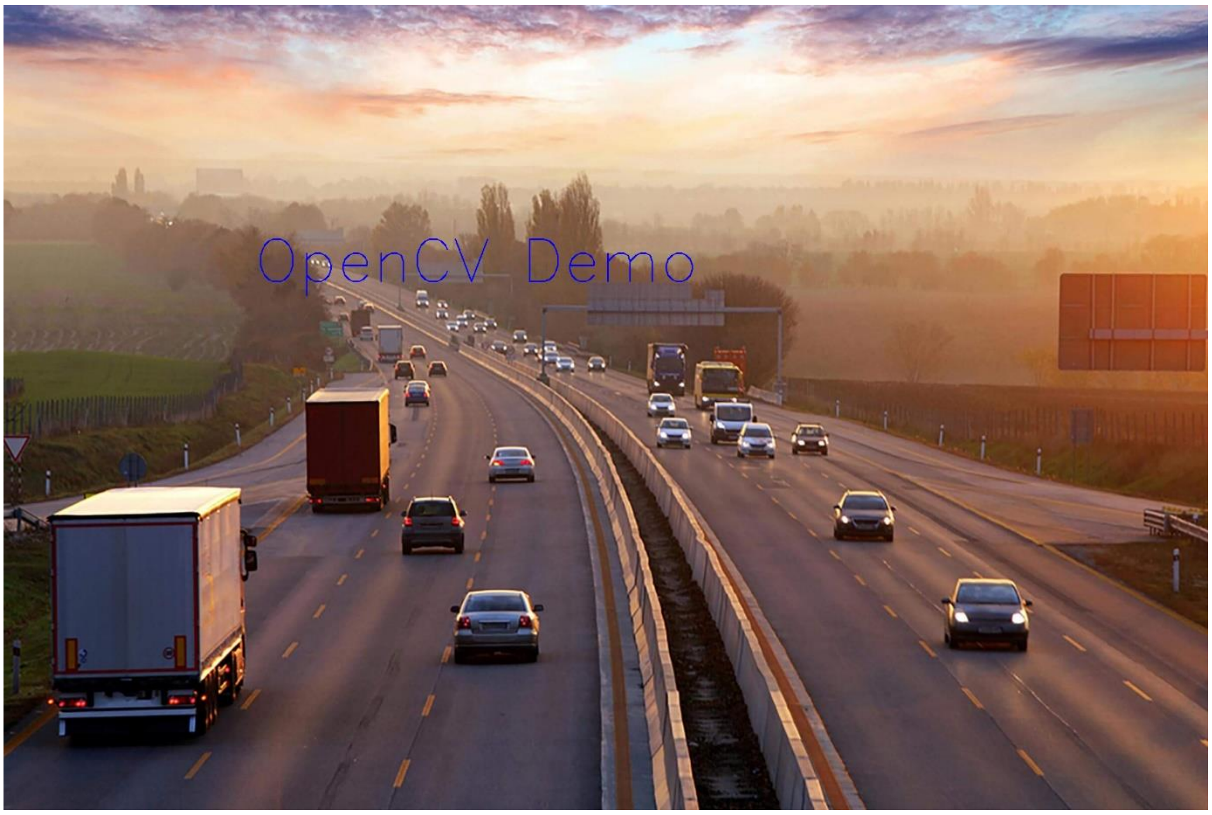
It takes in 5 arguments –

- Image
- Top-left corner co-ordinates
- Bottom-right corner co-ordinates
- Colour (in BGR format)
- Line width

Displaying text

It is also an in-place operation

```
# Copying the original image
output =image. Copy()


# Adding the text using put Text() function
text =cv2.putText(output, 'OpenCV Demo', (500, 550),
                  cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 0, 0), 2)
```



It takes in 7 arguments –

- Image
- Text to be displayed
- Bottom-left corner co-ordinates, from where the text should start
- Font
- Font size
- Colour (BGR format)
- Line width

Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, Geeks for Geeks Courses are your key to success. We provide top-quality content at affordable prices, all geared towards accelerating your growth in a time-bound manner. Join the millions we've already empowered, and we're here to do the same for you. Don't miss out - check it out now!

FAST2SMS

Fast SMS API Documentation

Fast SMS API Documentation

Our Fast SMS API allows you to send SMS messages to any mobile phone around the world. Below is the documentation of our API endpoints, parameters, and responses. That's it! You can now start using our Fast SMS API to send messages quickly and easily. If you have any questions or need help, feel free to contact our support team.

Table of Contents

1. Introduction
2. Authentication
3. Endpoints
4. Request Parameters
5. Response Format
6. Example Requests
7. Rate Limiting
8. Error Handling
9. Security
10. Conclusion

Introduction

- <a name="introduction"></a>

The Fast SMS API is a service that enables the sending of SMS messages to mobile numbers. It provides a fast and reliable way to deliver messages to recipients. This documentation outlines the key aspects of the API, including authentication, endpoints, request parameters, and response formats.

2. Authentication

<a name="authentication"></a>

To use the Fast SMS API, you need to authenticate your requests. The authentication method may vary but typically involves providing an API key or token in the request headers. Contact your service provider to obtain the necessary authentication credentials.

3. Endpoints

<a name="endpoints"></a>

The following endpoints are available for interacting with the Fast SMS API:

POST /SMS/send: Send an SMS message.
GET /SMS/status/{messaged}: Check the status of a previously sent SMS.

4. **Request Parameters**

<a name="request-parameters"></a>

4.1 Send SMS (POST /SMS/send)
to (string): The recipient's mobile number.
message (string): The content of the SMS.
from (string, optional): Sender ID or phone number (if supported).
schedule time (string, optional): Schedule the message for a specific time.

type (string, optional): Message type (e.g., promotional, transactional).

callback URL (string, optional): URL for delivery status callbacks.

4.2 Check SMS Status (GET /SMS/status/{massaged})

massaged (string): The unique identifier of the sent SMS message.

5. Response Format

a. <a name="response-format"></a>

The API responds with JSON data containing information about the sent SMS or the status of a sent SMS.

Example response for sending an SMS:

Json

Copy code

```
{
   "massaged": "12345",
   "status": "sent",
   "Sent at": "2023-11-09T14:00:00Z"
}
```

6. Example Requests

<a name="example-requests"></a>

6.1 Send SMS (POST /SMS/send)

http

Copy code

```
POST /SMS/send HTTP/1.1
Host: api.example.com
Authorization: Bearer Foretoken
Content-Type: application/Json

{
   "to": "+1234567890",
   "message": "Hello, this is a test SMS."
}
```

6.2 Check SMS Status (GET /SMS/status/{messaged})

http

Copy code

```
GET /SMS/status/12345 HTTP/1.1
Host: api.example.com
Authorization: Bearer Foretoken
```

7. Rate Limiting

a. <a name="rate-limiting"></a>

The API may impose rate limits on the number of requests you can make per minute or per day. Ensure compliance with the rate limits specified by the service provider to avoid service disruptions.

8. Error Handling

<a name="error-handling"></a>

The API may return error responses in case of invalid requests or other issues. Error responses typically include an error code and a description of the problem. Handle errors gracefully in your application.

9. Security

<a name="security"></a>

To ensure the security of your API requests, always use HTTPS for communication. Protect your API credentials (API keys or tokens) and implement security best practices in your code.

10. Conclusion

<a name="conclusion"></a>

The Fast SMS API provides a convenient way to send SMS messages to mobile numbers. Proper authentication, correct usage of endpoints, and handling of responses are essential for effective integration. Always refer to the service provider's specific documentation for more detailed information and updates on the API.

CONCLUSION

CONCLUSION

The project presented a novel approach to user authentication by combining face recognition and OTP-based authentication to create a secure and user-friendly login system. The proposed system aimed to address the limitations of traditional username and password-based authentication methods, enhancing security while maintaining a positive user experience.

In conclusion, this project successfully achieved

Enhanced Security: By integrating face recognition and OTP-based authentication, the system provided a robust two-factor authentication (2FA) mechanism. This significantly improved security by reducing the risk of unauthorized access and mitigating potential threats. User-Friendly Interface: The graphical user interface developed using Tkinter ensured a seamless and intuitive user experience. Users were provided with clear instructions and feedback throughout the login process, enhancing usability.

Secure Data Storage

User data, including facial templates and mobile numbers, were securely stored in a database, safeguarding sensitive information from potential breaches.

Account Recovery

In the rare event of both face recognition and OTP failure, the system offered a fallback mechanism for account recovery, ensuring users could regain access to their accounts.

Continuous Improvement

The system was designed to accommodate updates and enhancements to adapt to evolving security threats and user needs, ensuring its long-term relevance.

By combining these elements, the proposed system offers a comprehensive and effective solution to the challenges posed by traditional authentication methods.

MAIN CODE

MAINCODE

```
import requests
from tkinter import *
from PIL import Image, ImageTk
import cv2, os, sys, pickle
import time
import numpy as np
from configs import *
import random
import webbrowser as wb


faceCascade = cv2.CascadeClassifier('bin/haarcascade_frontalface_default.xml')
profileCascade = cv2.CascadeClassifier('bin/haarcascade_profileface.xml')

recognizer = cv2.face_LBPHFaceRecognizer.create()

recognizer.read('face_rec_saved_model.yaml')

with open(label_name_map_file, 'rb') as handle:
label_name_map = pickle.load(handle)

print("Press 'q' to quit\n\n\n")
video_capture = cv2.VideoCapture(0)
def otp_verify():

    password1 = password_verify5.get()
    print("enter password:",password1)
    print("otpis:",d)
    if  password1==str(d):
wb.open_new_tab('http://www.google.com')
        print("hai")
```

```
def otp():

    global main_screen1
    main_screen1= Toplevel(login_screen)
    main_screen1.configure(background='#3d5705')
    Label( main_screen1, text="OTP Method", bg="#c6eb73", font=("Verdana",
20)).place(x=150,y=10)
    main_screen1.title("READING")
    main_screen1.geometry("800x1200")
    global username_verify5
    global password_verify5

    username_verify5 = StringVar()
    password_verify5 = StringVar()

    global username_login_entry5
    global password_login_entry5



    Label(main_screen1, text="ENTER OTP",font=("Verdana",
15),bg="#c6eb73",fg="black").place(x=320,y=300)
password_login_entry = Entry(main_screen1,
justify=RIGHT,textvariable=password_verify5, show=
'*',font=('Verdana',15,'bold')).place(x=250,y=340)
    Button(main_screen1, text="ENTER", width=30, height=2,bd=5, command =
otp_verify,bg="#a6ed07",activebackground="#c6eb73").place(x=270,y=400)



def predictFacesFromWebcam(label2name_map):
video_capture = cv2.VideoCapture(0)
i=0
        global d
        e=0
        while e==0:
                ret, frame = video_capture.read()
                print(ret)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                faces = faceCascade.detectMultiScale(gray, scaleFactor, minNeighbors,
cascadeFlags, minSize)
                for (x, y, w, h) in faces:
i=i+1
                        print(i)
                        if i>0:
i=0

                           d=random.randint(1000,9999)
                           print(d)
send_message(d,"9384444652")
otp()
                           e=1
                        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
name_predicted, confidence = recognizer.predict(cv2.resize(gray[y: y + h, x: x + w],
face_resolution))
                        print(str(name_predicted) +' , ' +str(confidence))
                        if(name_predicted!=0 and confidence<confidence_threshold):
                                print("It is predicted as "+label2name_map[name_predicted])
                                cv2.putText(frame, label2name_map[name_predicted],
(x+3,y+h+20), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,0))

                cv2.imshow('Video', frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                        print("\nQuitting")
                        break
video_capture.release()
        cv2.destroyAllWindows()
def login_verify():
    username1 = username_verify.get()
    password1 = password_verify.get()



    if username1=="admin" and password1=="1234":
predictFacesFromWebcam(label_name_map)

    else:

password_not_recognised()
time.sleep(1)

def send_message(d,num):
        import requests

url = "https://www.fast2sms.com/dev/bulkV2"

querystring =
{"authorization":"zNVnDvTOWoPyud5w8f20sQUrZjKcJiBeHa3GCq16I7XmAYgl
pE0Aas9lQHyfObBdkM8KVSEcTZoD2Yz7","variables_values":str(d),"route":"otp
","numbers":str(num)}

        headers = {
            'cache-control': "no-cache"
        }

        response = requests.request("GET", url, headers=headers, params=querystring)

        print(response.text)
def mess():
send_message(otp,"9894938971")
def login():
    global login_screen
login_screen = Tk()
login_screen.title("Login")
login_screen.geometry("766x708")
login_screen.configure(background='#3d5705')
    Label(login_screen, text="Please enter details below to login",bg="#c6eb73",
font=("Calibri", 30)).place(x=150,y=10)

    global username_verify
    global password_verify

username_verify = StringVar()
password_verify = StringVar()

    global username_login_entry
    global password_login_entry

    Label(login_screen, text="USERNAME",font=("Verdana",
15),bg="#c6eb73",fg="black").place(x=320,y=200)
username_login_entry = Entry(login_screen,justify=RIGHT,
textvariable=username_verify,font=('Verdana',15,'bold')).place(x=250,y=240)
    Label(login_screen, text="PASSWORD ",font=("Verdana",
15),bg="#c6eb73",fg="black").place(x=320,y=300)
password_login_entry = Entry(login_screen,
justify=RIGHT,textvariable=password_verify, show=
'*',font=('Verdana',15,'bold')).place(x=250,y=340)
    Button(login_screen, text="Login", width=30, height=2,bd=5, command =
login_verify,bg="#a6ed07",activebackground="#c6eb73").place(x=270,y=400)
def main_account_screen():
    login()

login_screen.mainloop()

main_account_screen()
```

Train code:

```
import cv2, os, sys, pickle
import numpy as np
from PIL import Image
from configs import *

faceCascade = cv2.CascadeClassifier('bin/haarcascade_frontalface_default.xml')
profileCascade = cv2.CascadeClassifier('bin/haarcascade_profileface.xml')

recognizer = cv2.face_LBPHFaceRecognizer.create()
#can also use createEigenFaceRecognizer() or createFisherFaceRecognizer() or
createLBPHFaceRecognizer()
#Read http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html
to understand ML behind


def getFacesAndNames(path):
        image_paths = [os.path.join(path, f) for f in os.listdir(path)  if
f.endswith(pic_format)]
        faces = []
        names = []
        count = 0
        label2name_map = {}
        name2label_map = {}

        for i in image_paths:

                #Convert to grayscale and get as np_array
                img_gray = Image.open(i).convert('L')
                width, height = img_gray.size
                img = np.array(img_gray, 'uint8')
                #cv2.imshow('q',img)
                #Create label for person
                #name =str(os.path.split(i)[1].split(".")[0].replace("subject", ""))
                person_name = os.path.split(i)[1].split(".")[0]
                print(person_name)
                if person_name in name2label_map:
                    name = name2label_map[person_name]
                else:
                    count += 1
                    name2label_map[person_name] = count
                    name = count
                    label2name_map[count] = person_name

                #to detect frontal face
                face = faceCascade.detectMultiScale(img, scaleFactor,
minNeighbors, cascadeFlags, minSize)
                #to detect left side face
                #sideface_left = profileCascade.detectMultiScale(img, scaleFactor,
minNeighbors, cascadeFlags, minSize)
                #to detect right side face (mirror flip the image and use same
cascade)
                #sideface_right = profileCascade.detectMultiScale(np.fliplr(img),
scaleFactor, minNeighbors, cascadeFlags, minSize)

                #Add all detected faces to the list
                for(x, y, w, h) in face:
                    faces.append(cv2.resize(img[y: y + h, x: x + w],
face_resolution))
                    names.append(name)
                    #cv2.imshow("Adding faces to traning set...", img[y: y + h, x:
x + w])
                    #cv2.waitKey(0)
                    print("Frontal Face found in "+i)
                '''for(x, y, w, h) in sideface_left:
                    faces.append(cv2.resize(img[y: y + h, x: x + w],
face_resolution))
                    names.append(name)
                    #cv2.imshow("Adding faces to traning set...", img[y: y + h, x:
x + w])
                    #cv2.waitKey(0)
                    print("Left Side Face found in "+i)

                for(X, y, w, h) in sideface_right:
                    x = width-(X+w) #reflip to unmirror
                    faces.append(cv2.resize(img[y: y + h, x: x+w],
face_resolution))
                    names.append(name)
                    #cv2.imshow("Adding faces to traning set...", img[y: y + h, x:
x + w])
                    #cv2.waitKey(0)
                    print("Right Side Face found in "+i)'''
            return faces, np.array(names), label2name_map

####_MAIN_####

faces, names, label_name_map = getFacesAndNames('images_db') #Setup the facial
pictures
#cv2.destroyAllWindows()

recognizer.train(faces, names) #Train for facial recognition
recognizer.write(outfile) #Dump the trained model
with open(label_name_map_file, 'wb') as handle:
            pickle.dump(label_name_map, handle,
protocol=pickle.HIGHEST_PROTOCOL) #Dump the label:name map
```

Photo code:

```
import cv2, os, sys, time
import numpy as np
from PIL import Image
from configs import *
i=0
video_capture = cv2.VideoCapture(0) #Set the source webcam
video_capture .set(3,640)
video_capture .set(4,480)
print("Enter 'c' to capture the photo\n")
print("Enter 'q' to quit..\n\n")
print("Waiting to capture photo......\n\n")
while True:
    n = input("Enter: ")
    if(n=='q'):
        print("Quitting..")
        break
    if(n=='c'):
        name = input("Enter name: ")
neram = str(int(time.time()))
        while i<30:
            ret, frame = video_capture.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            cv2.imshow('Video', gray)
            if cv2.waitKey(1) & 0xFF == ord('n'):
                cv2.imwrite(db_path+"/"+str(name)+"."+neram+str(i)+".png",
gray)
                print("Saved as "+str(name)+"."+neram+str(i)+".png"+"\n\n")
i+=1
        print("Waiting to capture photo......")
print("\n\nPROCESS STOPPED......")
video_capture.release()
```