**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.

**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:

**User Registration and Face Enrollment:**

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

During registration, users are prompted to enroll 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**

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.

1. **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 recognitionhave 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.

1. **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 behavior login events. The uniqueness ofuser identity is validated in the proposed system at the login interface, followed by implying an appropriateuser 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 behavioral data, includinglogin 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.

1. **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. Key findings indicate that the designed and presented systems suitably attain identity verification of exam students using a unimodal biometric. The unimodal facial biometric system promises excellent support. A unimodal fingerprint biometric system ensures second excellent aid for student identity verification. However, multimodal, and semi-multimodal biometric systems provide better accuracy with fewer handling times and higher costs. This study contributes significantly to the knowledge of utilizing biometric recognition for identity verification in smart educational applications.

1. **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:**

**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 enroll 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:

**Biometric Authentication with Face Recognition:**

Users are required to enroll 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 Daigram:

opencv

Tkinter

Login Module

Login Page

Fast SMS

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 **filedialog** 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)

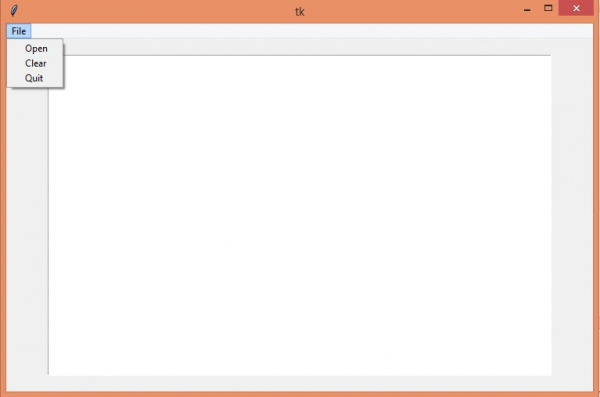
file\_menu.add\_command(label="Clear",command=clear\_text)

file\_menu.add\_command(label="Quit",command=quit\_app)

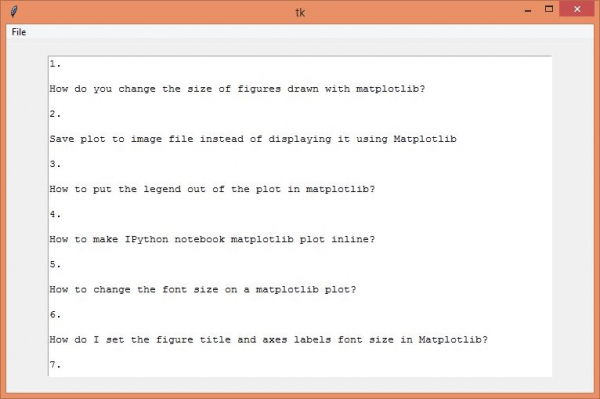
win.mainloop()

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.



How to add PDF in Tkinter GUI Python?

This article will teach us how to display PDF files in the Tkinter GUI. We will be using the PyMuPDF library to read the pdf files and convert them into images which will then be displayed using Tkinter.

For our task, we will do the following steps −

* Read the PDF file.
* Define a transformation matrix to apply on the pages of PDF to get their images.
* Count the total number of pages for error checking.
* Define the screen (the canvas) for our GUI.
* Define a helper function for converting a PDF page to a PIL image.
* Define a function to show the image of a page in our GUI.
* Add buttons, labels, scrollbars etc and configure them.

To get started, let's first install the PyMuPDF package by running the following command in our terminal.

python -m pip install --upgrade pip

python -m pip install --upgrade pymupdf

Now we need to import the libraries we will be working with. import fitz

import fitz

from tkinter import \*

from PIL import Image, ImageTk

Here fitz is a utility library that comes along with the PyMuPDF library and is used to get PDF pages as images.

We first open the PDF file we want to display in our GUI by using the fitz.open function. This returns a document object which can be used to access pages.

# open pdf file

file\_name = "sample.pdf"

doc = fitz.open(file\_name)

Now we need to specify how we would like to view the pages. By defining the below matrix function we are `fitz` to keep a 1x zoom.

# transformation matrix we can apply on pages

zoom = 1

mat = fitz.Matrix(zoom, zoom)

Let us also count the number of pages −

# count number of pages

num\_pages = 0

for p in doc:

num\_pages += 1

Now we will define a screen, add a scroll bar and a canvas to our code −

# initialize and set screen size

root = Tk()

root.geometry('750x700')

# add scroll bar

scrollbar = Scrollbar(root)

scrollbar.pack(side = RIGHT, fill = Y)

# add canvas

canvas = Canvas(root, yscrollcommand = scrollbar.set)

canvas.pack(side = LEFT, fill = BOTH, expand = 1)

Here our screen is represented by the name “root” and is of dimensions 750x700. We have also added a scroll bar and specified it to be on the right side and fill the entire height. Similarly, we have added the canvas to the left side and fill the entire screen on both sides!

In order to receive the page number the user wants to read, we need to take the page number as an input. We can do it as follows −

# define entry point (field for taking inputs)

entry = Entry(root)

# add a label for the entry point

label = Label(root, text="Enter page number to display:")

The arguments ‘root’ tell us that the objects will be associated with the screen “root”.

Now we will define a helper function to get the image of the PDF file from a page number.

def pdf\_to\_img(page\_num):

page = doc.load\_page(page\_num)

pix = page.get\_pixmap(matrix=mat)

return Image.frombytes("RGB", [pix.width, pix.height], pix.samples)

Here we first load the page using doc.load\_page and passing the page number as an argument. Then we convert the PDF page to get pixel representation of that page in the second line. Finally, we convert the pixel representation into the PIL Image format.

Now we define a function show\_image to display the image in our Tkinter GUI.

def show\_image():

try:

page\_num = int(entry.get()) - 1

assert page\_num >= 0 and page\_num < num\_pages

im = pdf\_to\_img(page\_num)

img\_tk = ImageTk.PhotoImage(im)

frame = Frame(canvas)

panel = Label(frame, image=img\_tk)

panel.pack(side="bottom", fill="both", expand="yes")

frame.image = img\_tk

canvas.create\_window(0, 0, anchor='nw', window=frame)

frame.update\_idletasks()

canvas.config(scrollregion=canvas.bbox("all"))

except:

pass

Here first we read the page number from the entry point we defined earlier and convert it to an integer. If it’s not a valid integer, an exception occurs and the exception block handles it. We then check if the page number is between 0 or the number of pages since that is the only valid range of pages we can display. Then using our previously defined helper function, we get a PIL Image of the user specified page. We then set the frame, panel and the canvas to appropriately display this image.

Next we add a button to display the page. We pass the function `show\_image` as a command. Hence whenever the button is pressed the `show\_image` function will be called.

# add button to display pages

button = Button(root, text="Show Page", command=show\_image)

Now we set the visual locations of the various components we defined previously −

# set visual locations

label.pack(side=TOP, fill=None)

entry.pack(side=TOP, fill=BOTH)

button.pack(side=TOP, fill=None)

We also set the initial page as the first page of the PDF as follows −

entry.insert(0, '1')

show\_image()

We configure the scroll bar

scrollbar.config(command = canvas.yview)

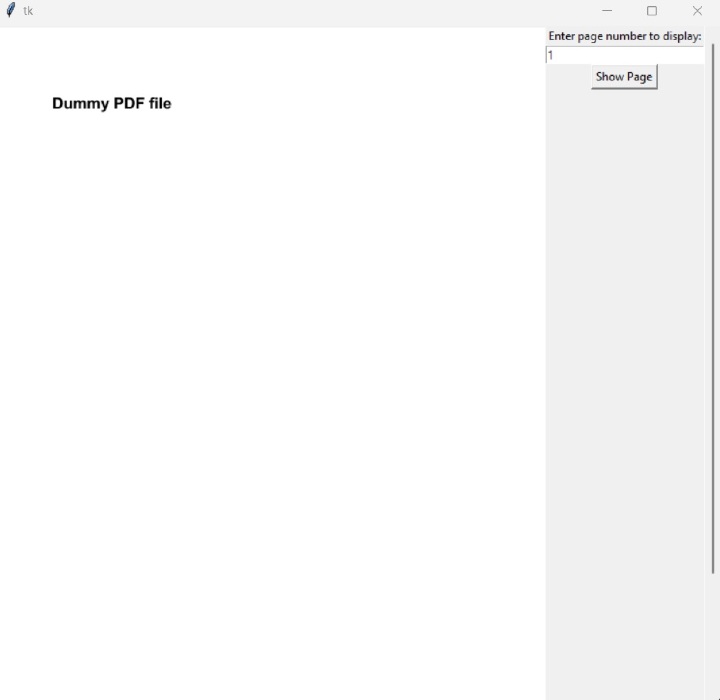
root.mainloop()

Finally we close the PDF so it doesn’t get corrupted by our computer.

doc.close()

Output

After running the program we see the following output!



# How to create an impressive GUI in Python using Tkinter?

Tkinter is a standard Python GUI library in Python, which gives us an object-oriented interface with **Tk** GUI Toolkit. It's amazing how quickly one can create some really impressive looking apps. Actions in GUI are usually performed through direct manipulation of graphical elements.

We will take a simple "addition" application to show how easy it is to create an impressive GUI in Python using tkinter. GUI is all about widgets and windows and these are available in Tkinter.

First, we will import the Tkinter library, then create a **window** object (class **Tk** is used to create **window** object) and create a **label** widget in which we can show any text or image (used in GUI application). The widget is in the insert window and on executing, we will get the output screen.

## Example

# Import the required libraries

from tkinter import \*

win=Tk()

win.geometry("700x300")

def sum():

a=int(entry1.get())

b=int(entry2.get())

c=a+b

# insert(index,value)

entry3.insert(0,c)

def clearing():

# delete(0,END)

entry1.delete(0,END)

entry2.delete(0,END)

entry3.delete(0,END)

label1=Label(win, text="Enter number 1:", padx=20, pady=10)

label2=Label(win, text="Enter number 2:", padx=20, pady=10)

entry1=Entry(win, width=30, borderwidth=2)

entry2=Entry(win, width=30, borderwidth=2)

entry3=Entry(win, width=30, borderwidth=2)

add=Button(win, text="Add", padx=20, pady=10, command=sum)

clear=Button(win, text="Clear", padx=20, pady=10, command=clearing)

label1.grid(row=0, column=0)

label2.grid(row=1, column=0)

entry1.grid(row=0, column=1)

entry2.grid(row=1, column=1)

add.grid(row=2, column=0)

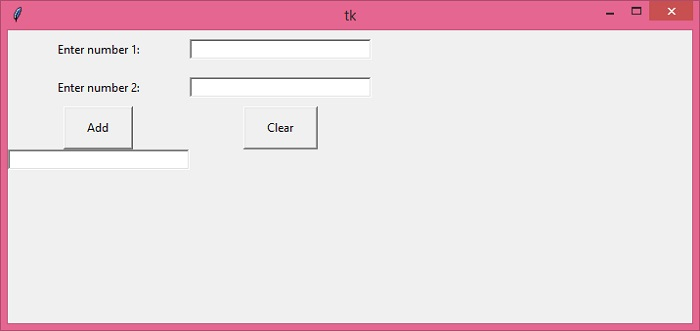
entry3.grid(row=3, column=0)

clear.grid(row=2, column=1)

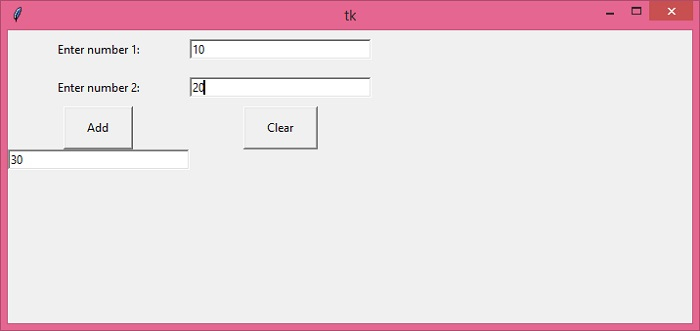
win.mainloop()

## Output

If we run the above code, it will display a window with a basic addition widget.



Now, insert two numbers (number 1 and number 2) and click the **Add** button to show the summation of number 1 and number 2 or click the **Clear** button to refresh the output screen.



# How do I create an automatically updating GUI using Tkinter in Python?

GUI window has many controls such as labels, buttons, text boxes, etc. We may sometimes want the content of our controls such as labels to update automatically while we are viewing the window.

We can use **after()** to run a function after a certain time. For example, 1000 milliseconds mean 1 second. The function which we call continuously after a certain amount of time will update the text or any updation you want to happen.

We have a label on our window. We want the text of the label to update automatically after 1 second. To keep the example easy, suppose we want the label to show some number between 0 and 1000. We want this number to change after each 1 second.

We can do this by defining a function that will change the text of the label to some random number between 0 and 1000. We can call this function continuously after an interval of 1 second using the after().

## Example

from Tkinter import \*

from random import randint

root = Tk()

lab = Label(root)

lab.pack()

def update():

   lab['text'] = randint(0,1000)

   root.after(1000, update) # run itself again after 1000 ms

# run first time

update()

root.mainloop()

This will automatically change the text of the label to some new number after 1000 milliseconds. You can change the time interval according to need. The update function can be modified to perform the required updation.

### **root.after(1000,update)**

This line of the code performs the main function of recalling the function update().

The first parameter in root.after() specifies the time interval in milliseconds after which you want the function to be recalled.

The second parameter specifies the name of the function to be recalled.

**OpenCV** is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as [Numpy](https://www.geeksforgeeks.org/python-numpy/" \t "_blank) which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.

This OpenCV tutorial will help you learn the Image-processing from Basics to Advance, like operations on Images, Videos using a huge set of Opencv-programs and projects.

OpenCV is one of the most popular computer vision libraries. If you want to start your journey in the field of computer vision, then a thorough understanding of the concepts of OpenCV is of paramount importance.  
In this article, I will try to introduce the most basic and important concepts of OpenCV in an intuitive manner.  
**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  **import** cv2  # Reading the image using imread() 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 center of the image  center **=** (w **//** 2, h **//** 2)    # Generating a rotation matrix  matrix **=** cv2.getRotationMatrix2D(center, **-**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.

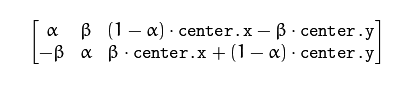
The 2 main functions used here are –

* getRotationMatrix2D()
* warpAffine()

**getRotationMatrix2D()**  
It takes 3 arguments –

* **center –** The center 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)



**warpAffine()**

The function warpAffine transforms the source image using the rotation matrix:

dst(x, y) = src(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
  + Color (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 putText() 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
  + Color (BGR format)
  + Line width

Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, [GeeksforGeeks Courses](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses" \t "_blank) 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!](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses)

**Fast SMS API Documentation**

**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
    1. **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/{message\_id}: 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/{message\_id})

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

1. **Response Format**
   1. <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

{

"message\_id": "12345",

"status": "sent",

"sent\_at": "2023-11-09T14:00:00Z"

}

1. **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 YourAPIToken

Content-Type: application/json

{

"to": "+1234567890",

"message": "Hello, this is a test SMS."

}

6.2 Check SMS Status (GET /sms/status/{message\_id})

http

Copy code

GET /sms/status/12345 HTTP/1.1

Host: api.example.com

Authorization: Bearer YourAPIToken

1. **Rate Limiting**
   1. <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.

1. **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.

1. **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.

This template serves as a basic structure for your Fast SMS API documentation. Be sure to customize it with the specific details and features provided by your SMS service provider.

**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 the following objectives:**

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. It not only strengthens security but also provides a user-friendly and efficient means of accessing user accounts. In an era where security breaches are a growing concern, this project represents a significant step forward in enhancing the protection of user accounts and sensitive data.

Reference Paper:

{1] Harris, S. (2012). CISSP Exam Guide 6th

Edition. MCGraw-Hill.

[2] Samarati, Pierengela., & di Vimercati, de Capitani, Sabrina. (2001).

Access Control: Policies, Models, and Mechanisms. R. Focardi and R.

Gorrieri (Eds.): FOSAD 2000, LNCS 2171, pp. 137–196, 2001.

[3] N. Pato. Josep & I. Millett. Lynette, “Biometric Recognition,

Chaleenges and Opportunities” NATIONAL RESEARCH COUNCIL,

2010.

[4] Marti, Ni Wayan. (2010). Pemanfaatan GUI dalam Pengembangan

Perangkat Lunak Pengenalan Citra Wajah Manusia Menggunakan

Metode Eigenfaces. Seminar Nasional Aplikasi Teknologi Informasi

2010 (SNATI 2010) ISSN: 1907-5022 Yogyakarta, 19 June 2010.

[5] Li, Lei. Feng, Xiaoyi. Xia, Zhaoqiang. Jiang, Xiaoyue. Hadid

Abdenour. (2018). Face Spoofing Detection with Local Binary Pattern.

Preprint submitted to Journal of Visual Communication and Image

Representation.

[6] Wijaya, Christian Hadi. Hendrawan, Ade Hendri. K.P, Andik Eko.

Goeritno, Arief. Implementasi Sistem One-Time Password (OTP)

sebagai Key Penggerak Kunci Pintu Berbantuan Arduini Uno. Seminar

Nasional Inovasi Dan Aplikasi Teknologi Di Industri 2017 ISSN 2085-

4218 ITN Malang, 4 February 2017.

[7] Gupta, Sandeep. Buriro, Attaullah, Crispo, Bruno. (2018).

Demystifying Authentication Concepts in Smartphones: Ways and Type

to Secure Access. Received 22 August 2017; Revised 11 December

2017; Accepted 9 January 2018; Published 11 March 2018.

[8] X. Geng, Z. Zhou and K. Smith-Miles, "Individual Stable Space: An

Approach to Face Recognition Under Uncontrolled Conditions," in

IEEE Transactions on Neural Networks, vol. 19, no. 8, pp. 1354-1368,

Aug. 2008, doi: 10.1109/TNN.2008.2000275.

[9] S. Z. Li and Juwei Lu, "Generalizing capacity of face database for face

recognition," Proceedings Third IEEE International Conference on

Automatic Face and Gesture Recognition, Nara, 1998, pp. 402-406,

[10] W. Zhao, R. Chellappa and A. Krishnaswamy, "Discriminant analysis

of principal components for face recognition," Proceedings Third IEEE

International Conference on Automatic Face and Gesture Recognition,

Nara, 1998, pp. 336-341, doi: 10.1109/AFGR.1998.670971.

[11] Y. Wang and J. Su, "Symmetry description and face recognition using

face symmetry based on local binary pattern feature," Proceedings of

the 32nd Chinese Control Conference, Xi'an, 2013, pp. 3955-3960.

[12] I. M. de Diego, C. Conde, Á. Serrano and E. Cabello, "Combination of

kernels applied to face verification," 2009 16th IEEE International

Conference on Image Processing (ICIP), Cairo, 2009, pp. 1241-1244,

**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("otp is:",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":"zNVnDvTOWoPyud5w8f20sQUrZjKcJiBeHa3GCq16I7XmAYglpE0Aas9lQHyfObBdkM8KVSEcTZoD2Yz7","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()