**Higher Level Overview of the application:**

This is a simple machine learning application for performing facial recognition , in simple words the functionality works in the following ways:

1. Register the User , by capturing their face and ask them to input their name
2. After the user gives the data , the facial data along with their name will be stored in the sqlite database
3. Then if the User wants to verify if his face is stored , then the live captured face image is compared to that of the stored image.
4. If they both are the same then the name of the User is shown , else it shows “Unknown User”.

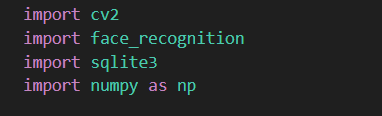
**Detailed Code Explanation of the Face recognition code:**

**Introduction:**

Here the objective is to make a face recognition system that captures the users face and name during registering and then verifies the stored face with the live captured face using pre-trained deep learning model.

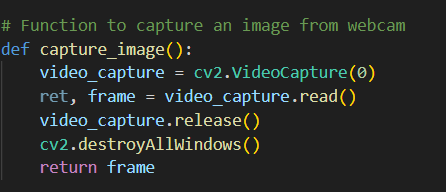
**1)Face\_recognition\_code module elaboration:**

**Libraries :**



1. OpenCV(cv2) it is mainly imported to capture image through the webcam and process them . It is a open-source machine learning library.
2. face\_recognition This is a machine learning library that contains pre-trained deep learning models by using dlib under the hood , and helps in face detection , feature extraction and verification.
3. Sqlite3 -> This is the database that is being used for storing the captured faces of the users along with their names.
4. Numpy ->This library is used to handle different numerical operations and manipulations of the face encodings further in the code.

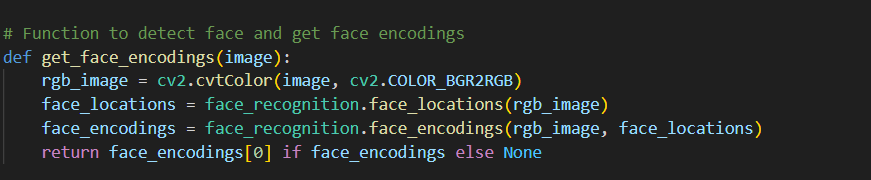
**Capture the image of the User through the Webcam:**



Inside the capture\_image() function :

1. Webcam is initialized by openCV
2. A single image on the frame is captured
3. Webcam is released
4. OpenCV windows are closed
5. Captured frame is returned

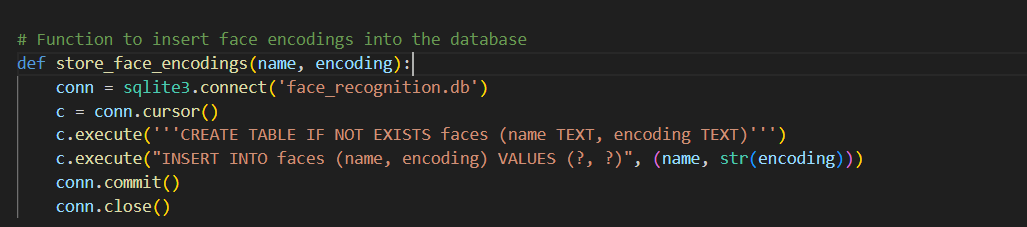
**Extract the face encodings from the captured :**



Inside the get\_face\_encodings(image) function:

1. Change the colour format of the image from BGR which is by default in OpenCV to RGB which is required by face\_recognition library.
2. Uses deep learning pre-trained model to identify the faces in the RGB captured image.
3. Face encodings of the detected face locations to extract and encode face features in 128 dimensional encodings for each face.
4. Encodings of the first face detected is returned else it returns None.

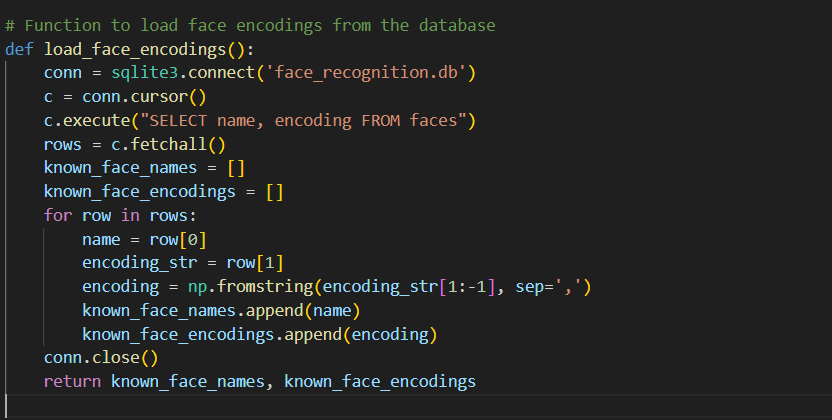
**Insert the face encodings of the captured faces along with names in SQLite database:**



Inside the store\_face\_encodings(name,encoding) function:

1. Connecting to the sqlite3 database
2. Create a cursor to access the database
3. Create a table called ‘Faces’ that will contain the user name and face encodings .
4. If the table is already created , just insert the name and face encodings
5. Commit and close the connection .

**Load the face encodings and name from the database it will be needed in the next step for verification :**

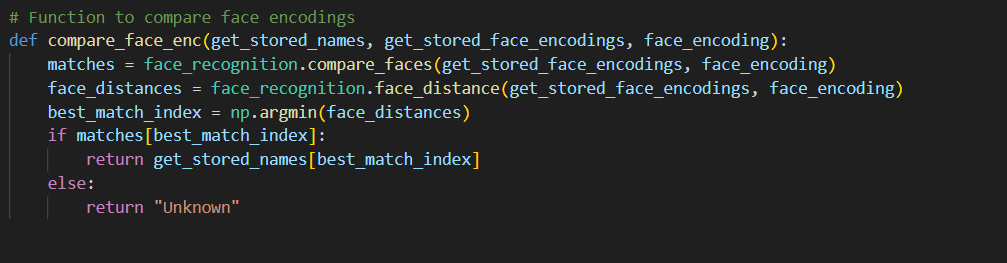


Inside the load\_face\_encodings() function:

1. Connects to the sqlite3 database
2. Create a cursor to execute the commands
3. Sql statement to select all the name and face encodings from the faces table
4. All the selected rows are fetched
5. Two empty lists are initialized
6. With a for loop iterate over each row that are fetched one by one
7. Store the name part in one variable
8. Store the encoding in another variable
9. Convert the encoding to numpy array from string
10. Finally append the stored name in the variable to the empty list
11. Append the stored and modified encoding to the empty list.
12. Finally close the connection to database
13. The face name and its encodings are returned

**Compare the stored face encodings in the database with the live captured face encodings:**

**(live face capturing procedure is done using the capture\_image() , function call from the flask app)**



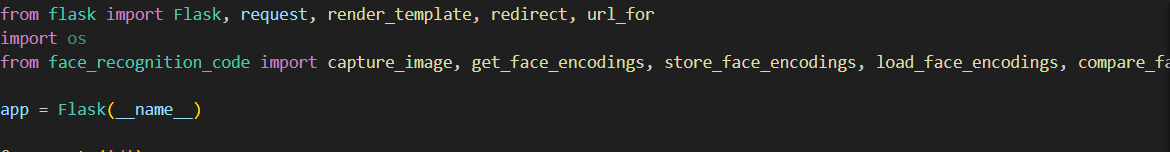
Inside the function compare\_face\_enc() :

1. The is a pre-trained deep learning model that compares the detected face encodings with the stored face encodings
2. The Euclidean distance between the captured face and the stored face is calculated and the one with least distance is a match.
3. Finds the index of the smallest distance in the list which will be the best match
4. Checks if the best match value exist , if it does then,
5. The name of the best match is return
6. Else “Unknown” is returned.

**2)Flask app documentation( app.py) elaboration :**

**Objective: Here a detailed explanation of how the above functions are called in the interactive flask application to make the entire system work properly.**

**Libraries:**



Flask->It is the main class for creating flask application.

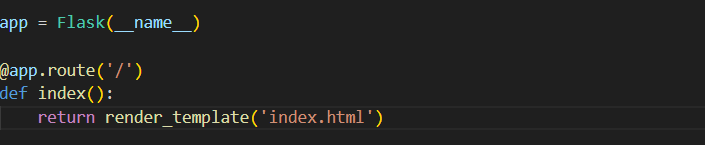
Request->Used to handle incoming requests

render\_template->Used to render the HTML pages

redirect & url\_for ->Used to redirect urls

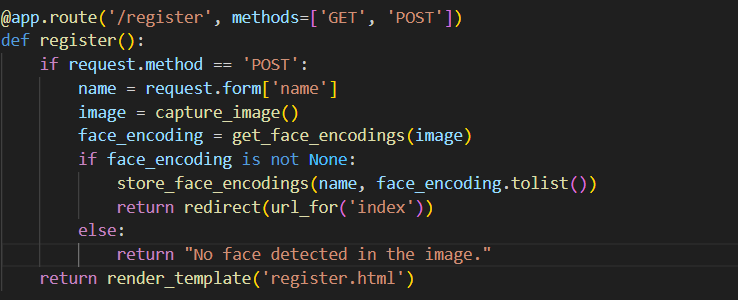
capture\_image,get\_face\_encodings,store\_face\_encodings,load\_face\_encodings,compare\_face\_encodings -> These are functions that are used for performing face recognition tasks from image capture to verification.

**Creating a flask application and defining routes:**



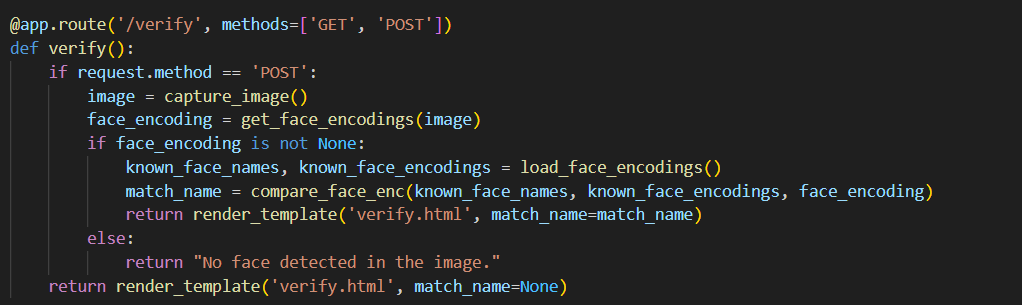
1. At first an instance of the Flask class is created.
2. Then, the root for the home page is defined.
3. Inside the function ,index(), when it is accessed it renders and returns the index.html page.

**Creating second route :**



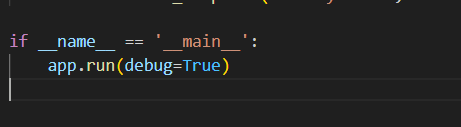
1. Another router ‘/register’ is defined which can handle both ‘GET’ and ‘POST’ requests.
2. Inside the register() function
3. A check is done for if the request method is ‘POST’ that means user submitted the form
4. If so, the name is retrieved from the form
5. Then, the capture image() function is called to capture the image of the user
6. The captured image is processed then and the face encodings are stored in a variable using get\_face\_encoding() function.
7. Checks if the face encoding is captured properly
8. If so, then the face encodings along with the name of the user is stored using store\_face\_encoding() function and then converted to a list.
9. Finally the user is redirected to the index page after successful registration.
10. Elso, if no face encoding is captured , an error message will be shown stating “No face detected in the image”.
11. If the request method is ‘GET’ it render the register.html page.

**Creating third route:**



1. The third route is created here, called ‘verify’ and it can also handle both ‘GET’ and ‘POST’ requests.
2. Inside the verify() function, it is checked if the request is ‘POST’, which happens when user has started a verification process.
3. If so then, capture\_image() function is called , that is used to capture the live image of the user.
4. Then, the get\_face\_encoding() function is called, that helps to extract the face encodings of the newly captured image.
5. If face encoding is extracted properly from the new image ,
6. Then name and face encodings are loaded using the load\_face\_encodings() functions , which helps to retrieve the data that was saved during previous operation of registration.
7. The face encodings extracted are then compared with the face encodings of the face that is stored during registration using the load\_face\_encodings(), if they are matched.
8. The name of the user is returned.
9. If face encodings are not extracted properly , then an error message will be displayed as “No face detected in the image”.
10. If the request method is ‘GET’ it display the initial verification page called ‘verify.html’.

**Running the Flask application:**



1. This block is used to ensure , that the application starts only when run directly.
2. Starts the flask application server, with debugging enabled.

**3)Frontend part of the application:**



The entire code in this script is used for creating the front-end and the landing page for our simple facial recognition project.



This page contains the html script for the user registration frontend part.



This page contains the html script for the user verification frontend part.

**Summary:**

Hence, this project consists of a facial recognition system using a Flask Web application. Here, the user’s face and name is captured during registration and stored in the SQLite database and verification of faces is performed using a pre-trained deep learning model. The facial\_recognition library is used for processing facial data .The flask routes help user interactions during registration and verification by connecting the backend with the frontend.