**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:

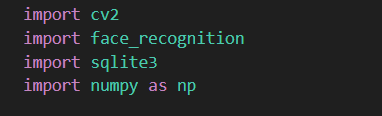
1. Register the User , by capturing their face and ask them to input their name
2. After the user gives the data , the facial data along with their name will be stored in the sqlite database
3. Then if the User wants to verify if his face is stored , then the live captured face image is compared to that 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:**

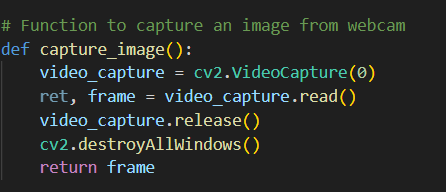
Here the objective is to make a face recognition system that captures the users face and name during registering and then verifies the stored face with the live captured face using pre-trained deep learning model.

**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.

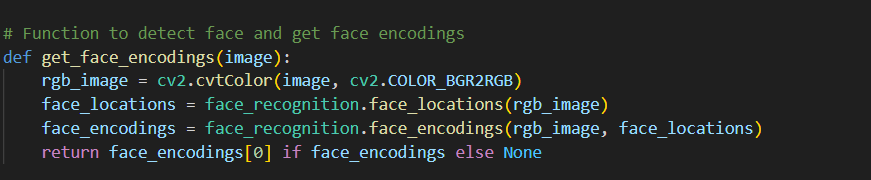
**Capture the image of the User through the Webcam:**



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

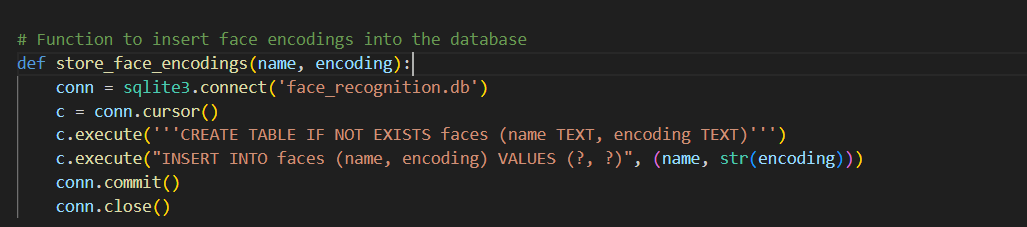
**Extract the face encodings from the captured :**



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.

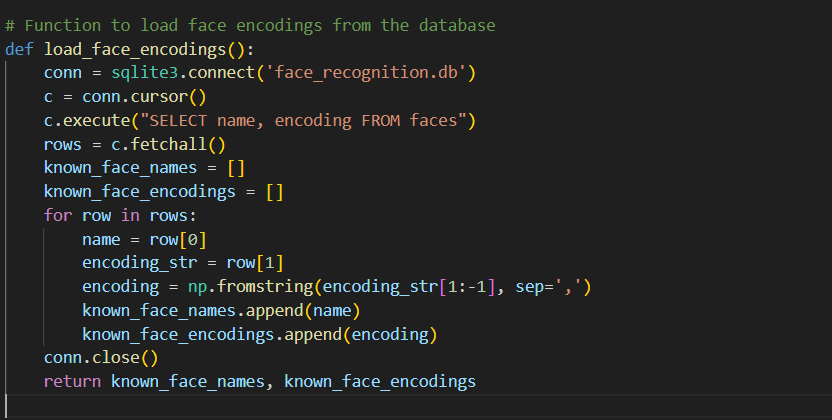
**Insert the face encodings of the captured faces along with names in SQLite database:**



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 :**

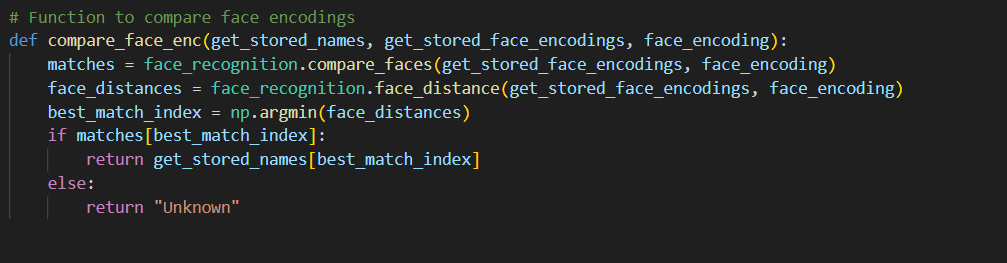


Inside the load\_face\_encodings() function:

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.

**Flask app documentation:**