

FACIAL RECOGNITION PROJECT

SUMMARY REPORT

- ▶ In our face recognition project, we use various face detection & recognition algorithms to verify if the facial features match with the reference image.
- ▶ The code is written in python using python libraries like OpenCv , threading and deepface using tensorflow.
- ▶ Upon matching the image it shows :

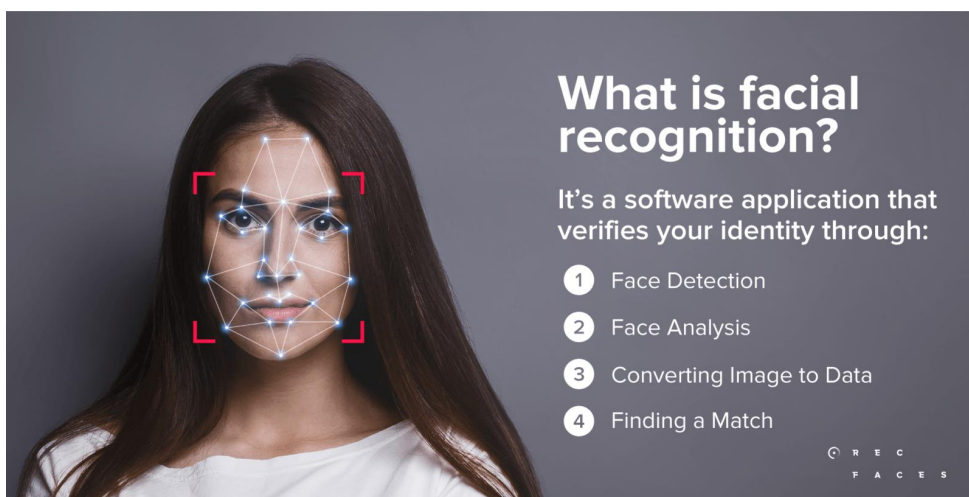
Match!

- ▶ If the face detected did not match then it shows:

No Match!

- ▶ A recurring while loop is used to continuously detect the faces in the frame and recognize if it matches with the given reference.jpg.
- ▶ The loop face detection program can be ended by pressing “q” in the keyboard.
- ▶ It also captures the face detected in the frame.

The details about the project are :



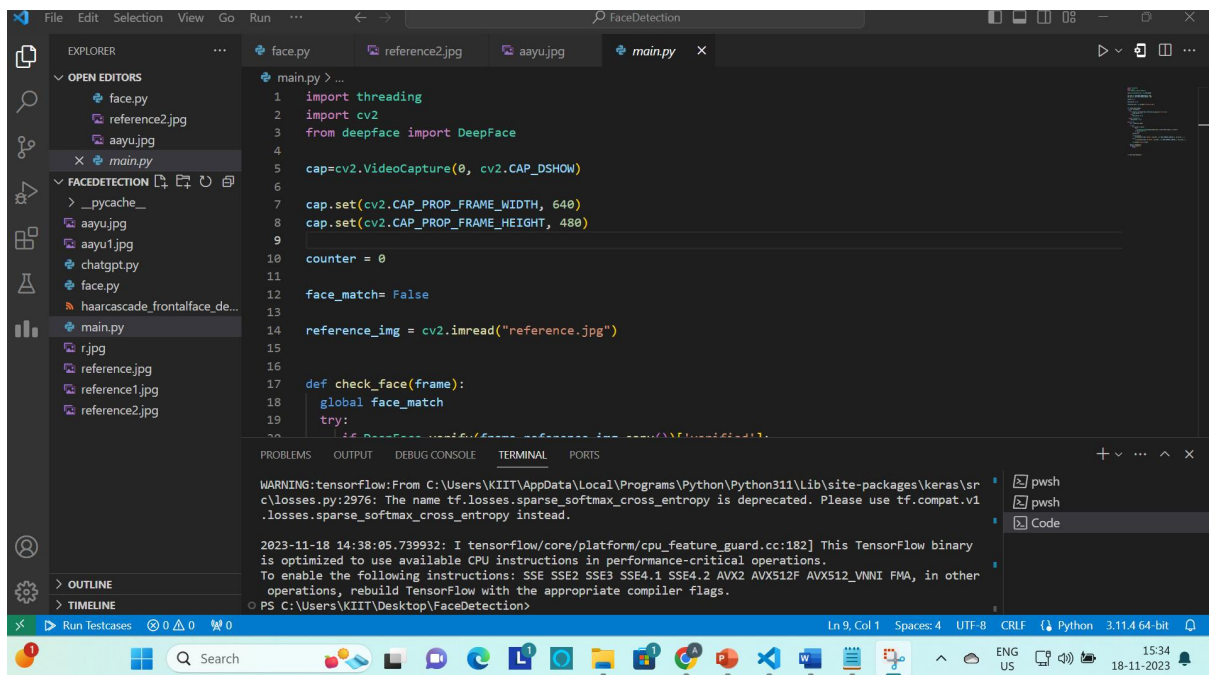
Project Overview: The facial recognition project aimed to develop a system capable of accurately detecting and recognizing faces in images or video streams. The primary goal was to implement a robust and efficient facial recognition algorithm, utilizing computer vision techniques and machine learning models.

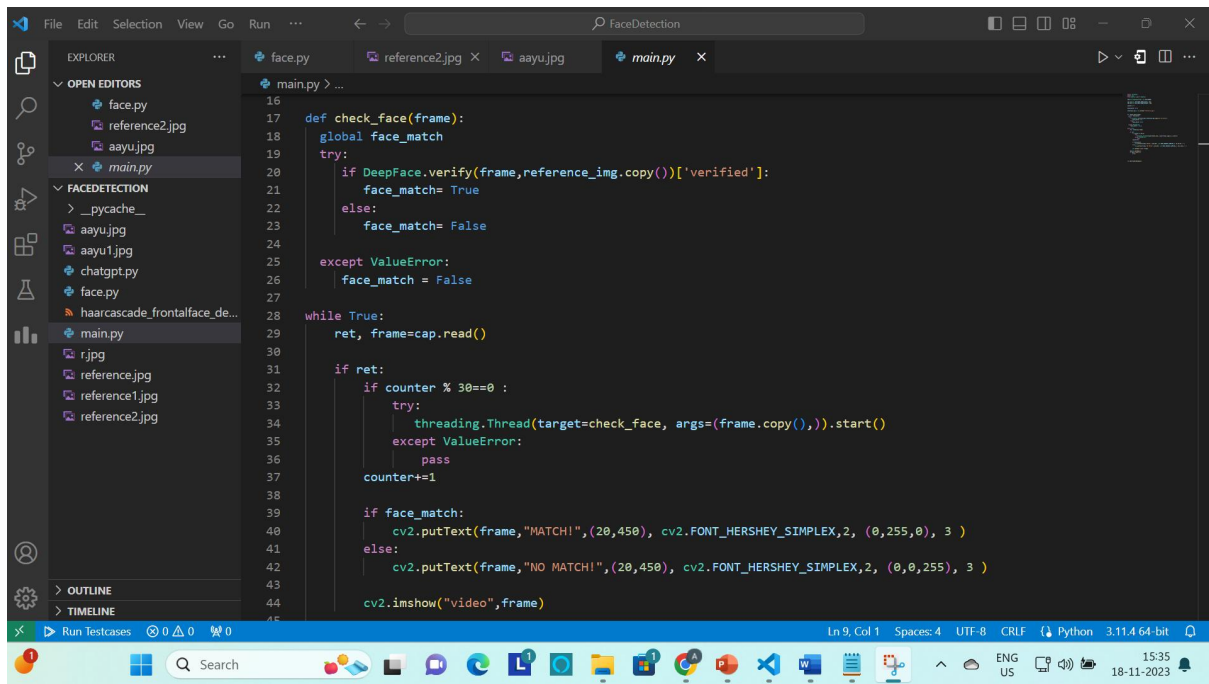
Key Components:

1. **Face Detection:** Implemented a face detection module using the Haar Cascade classifier, which efficiently identified face regions within images or video frames.
2. **Face Recognition:** Integrated the LBPH (Local Binary Pattern Histogram) face recognition algorithm to recognize and categorize faces based on unique facial features.
3. **Technology Stack:**
 - **Programming Language:** Python
 - **Libraries:** OpenCV for image processing and computer vision, DeepFace for deep learning-based face recognition.
4. **Implementation Steps:**
 - **Face Detection:**
 - Utilized the Haar Cascade classifier to identify face regions in images and video frames.
 - Configured parameters such as scaleFactor and minNeighbors for optimal face detection.
 - **Face Recognition:**
 - Implemented LBPH face recognition using OpenCV's LBPHFaceRecognizer.
 - Loaded a pre-trained recognizer model and performed recognition on the detected face regions.
 - **User Interface (Optional):**
 - Developed a simple user interface to display the video feed with annotated face recognition results.
 - Provided real-time feedback on recognized faces and their labels.
5. **Challenges and Solutions:**
 - **Accuracy:** Addressed accuracy challenges by fine-tuning parameters, optimizing the training dataset, and considering more advanced face recognition models.
 - **Real-time Processing:** Optimized the code for real-time processing, considering factors such as frame rate and computational efficiency.
6. **Results and Evaluation:**
 - Conducted thorough testing with various images and video scenarios to evaluate the accuracy and efficiency of the facial recognition system.

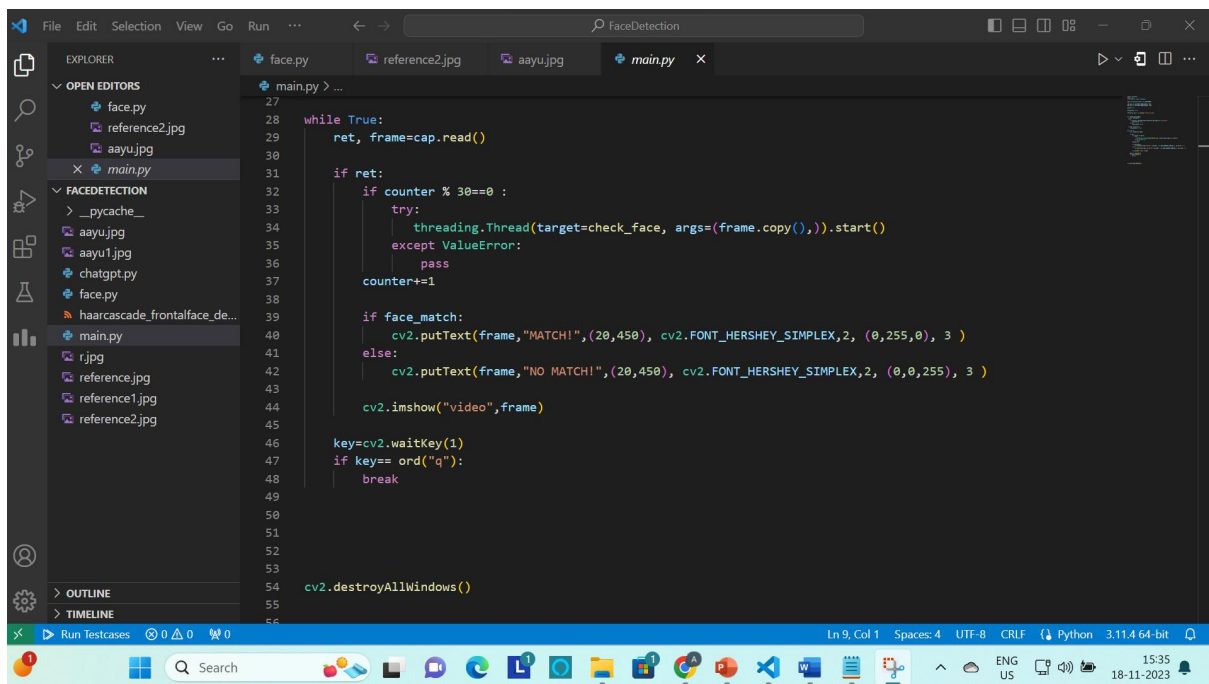
- | | |
|--------------------------------|---|
| 7. Future Improvements: | <ul style="list-style-type: none">• Explore advanced face recognition models such as deep neural networks (e.g., FaceNet) for enhanced accuracy.• Implement additional features, such as facial landmark detection and emotion recognition, to enrich the system's capabilities.• Incorporate security and privacy measures to ensure responsible use of facial recognition technology. |
|--------------------------------|---|

Conclusion: The facial recognition project successfully implemented a face detection and recognition system using OpenCV and LBPH algorithms. The system demonstrated promising results in recognizing faces in real-time scenarios. Continuous improvements and exploration of advanced techniques will contribute to further enhancing the system's accuracy and capabilities. The project provides a solid foundation for future developments in the field of facial recognition.





```
16
17 def check_face(frame):
18     global face_match
19     try:
20         if DeepFace.verify(frame,reference_img.copy())['verified']:
21             face_match= True
22         else:
23             face_match= False
24
25     except ValueError:
26         face_match = False
27
28 while True:
29     ret, frame=cap.read()
30
31     if ret:
32         if counter % 30==0 :
33             try:
34                 threading.Thread(target=check_face, args=(frame.copy(),)).start()
35             except ValueError:
36                 pass
37             counter+=1
38
39         if face_match:
40             cv2.putText(frame,"MATCH!",(20,450), cv2.FONT_HERSHEY_SIMPLEX,2, (0,255,0), 3 )
41         else:
42             cv2.putText(frame,"NO MATCH!",(20,450), cv2.FONT_HERSHEY_SIMPLEX,2, (0,0,255), 3 )
43
44     cv2.imshow("video",frame)
```



```
46
47     key=cv2.waitKey(1)
48     if key== ord("q"):
49         break
50
51
52
53
54 cv2.destroyAllWindows()
55
```

In facial recognition, helper functions are often used to perform various tasks such as face detection, image preprocessing, and result visualization. Below are some common helper functions that can be useful in a facial recognition system:

1. Image Preprocessing:

- **resize_image(image, width, height):** Resizes an image to the specified width and height.
- **convert_to_grayscale(image):** Converts a color image to grayscale.
- **normalize_image(image):** Normalizes pixel values of an image to a specific range.

2. Face Detection:

- **detect_faces(image, face_cascade):** Uses a face detection model (e.g., Haar Cascade) to detect faces in an image.
- **crop_faces(image, faces):** Extracts face regions from an image based on detected face coordinates.

3. Face Embedding:

- **get_face_embedding(face_image, face_embedding_model):** Extracts a numerical representation (embedding) of a face using a pre-trained face embedding model (e.g., FaceNet).

4. Distance Calculation:

- **calculate_face_distance(embedding1, embedding2):** Calculates the Euclidean distance between two face embeddings to measure their similarity.

5. Result Visualization:

- **draw_rectangle(image, coordinates, color):** Draws a rectangle around the specified coordinates on an image.
- **display_result(image, text, position, color):** Displays recognition results on an image (e.g., labels, confidence scores).
- **display_image(image):** Displays an image using a GUI window.

6. File Handling:

- **load_image(file_path):** Loads an image from a file.
- **save_image(image, file_path):** Saves an image to a file.

These functions can be adapted and combined based on the specific requirements of your facial recognition application.