Project Report: Real-time Face Recognition Snap Blend

Abstract:

The "Real-time Face Recognition using Haar Cascades" project aims to develop a face recognition system capable of identifying known individuals in real-time video streams. The system employs Haar Cascade Classifier for face detection and matching techniques to recognize known faces. **Additionally, a multithreading project** was developed to enhance the responsiveness of the face detection process.

Introduction:

Face recognition is a fundamental task in computer vision with widespread applications in security, surveillance, and authentication systems. In this project, we employ the Haar Cascade Classifier, a machine learning-based approach for face detection, and utilize matching techniques to identify known individuals. Additionally, we have a separate multithreading project that optimizes the face detection process for improved real-time performance.

Methodology:

1. Haar Cascade Classifier:
   - The Haar Cascade Classifier is a machine learning-based object detection technique used for face detection.
   - We use OpenCV's pre-trained Haar Cascade Classifier to detect faces in images and video frames.
   - The classifier works by identifying specific features of the face, such as the eyes, nose, and mouth.

2. Known Faces Database:
   - We create a dictionary to store known face encodings and their corresponding names.
   - The known faces database is populated with images of individuals, cropped to contain only their face regions.
   - Each known face is encoded, creating a unique representation for recognition.
   -**Database was Developed with 2000 images** per Class(Person).

3. Real-time Face Recognition:
   - During the video feed, each frame is captured and converted to grayscale for face detection.
   - The Haar Cascade Classifier is applied to the grayscale frame to detect faces.

- For each detected face, we crop the face region and resize it to a fixed size for consistent comparisons.
- The cropped face region is then matched with the known faces database using matching techniques.
- If the match score exceeds a predefined threshold, the person's name is identified.

4. Multithreading for Face Detection:
- As a project, we implement multithreading to optimize the face detection process.
- The main thread captures video frames and performs face detection on them.
- Concurrently, a separate thread matches detected faces with the known faces database.
- This multithreading approach enhances the responsiveness of the face detection process.

Code:

```
import os
import cv2
import numpy as np

# Load the pre-trained Haar Cascade Classifier for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Create a dictionary to store known face encodings and their corresponding names
known_faces = {}
known_faces_dir = 'C:/Users/Dell/Desktop/__pycache__/preview/images/'

for person_folder in os.listdir(known_faces_dir):
    person_name = person_folder.split('.')[0]
    person_images_dir = os.path.join(known_faces_dir, person_folder)
    person_images = [os.path.join(person_images_dir, image) for image in
os.listdir(person_images_dir)]

    # Create an empty list to store the face encodings for this person
    person_face_encodings = []

    for image_path in person_images:
        img = cv2.imread(image_path)
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Detect faces in the image
        faces = face_cascade.detectMultiScale(gray_img, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))
```

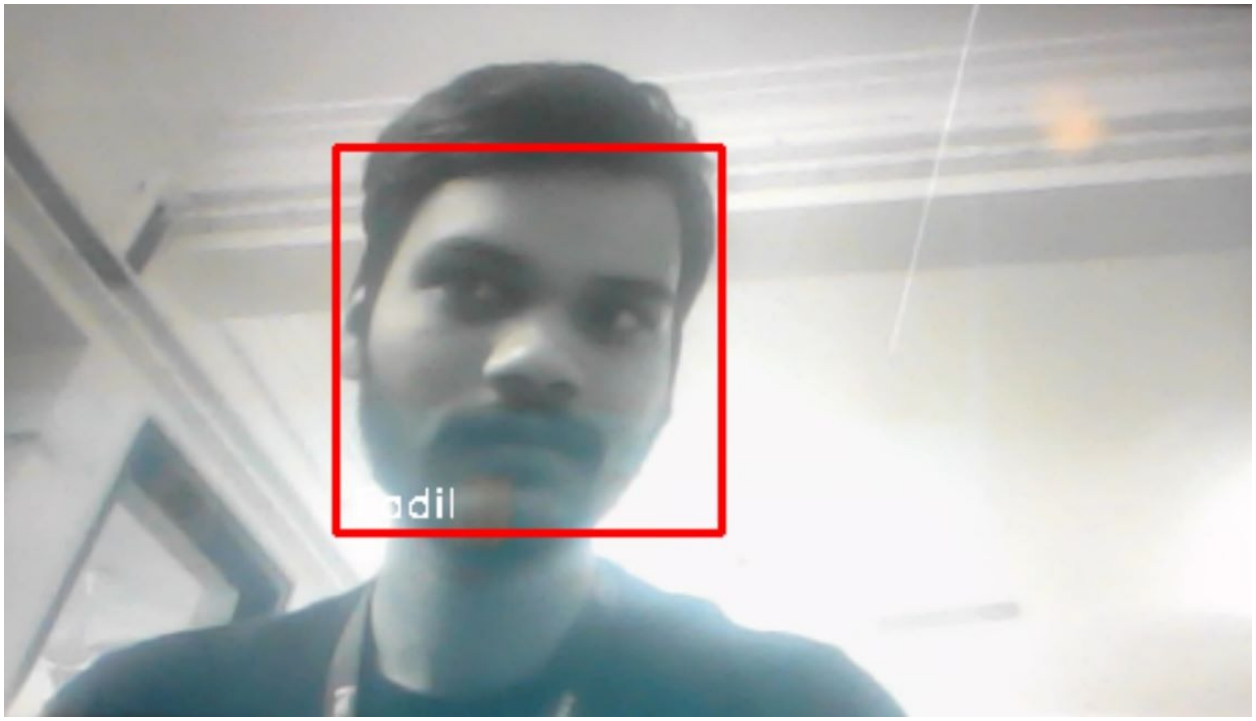
```
        # Assuming each image contains only one face, use the first detected face for
recognition
        if len(faces) == 1:
            x, y, w, h = faces[0]
            face_encoding = gray_img[y:y + h, x:x + w]

            # Resize face_encoding to a fixed size for consistent comparisons
            target_size = (100, 100)
            face_encoding = cv2.resize(face_encoding, target_size)

            person_face_encodings.append(face_encoding)

    # Store the face encodings for this person in the known_faces dictionary
    known_faces[person_name] = person_face_encodings

# Start capturing video from the default camera (0)
video_capture = cv2.VideoCapture(0)

while True:
    # Capture each frame from the video feed
    ret, frame = video_capture.read()

    # Convert the frame to grayscale for face detection
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the current frame
    faces = face_cascade.detectMultiScale(gray_frame, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))

    # Loop through each detected face
    for (x, y, w, h) in faces:
        # Draw a rectangle around the face
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)

        # Crop the face region for recognition
        face_region = gray_frame[y:y + h, x:x + w]

        # Resize face_region to a fixed size for consistent comparisons
        target_size = (100, 100)
        face_region = cv2.resize(face_region, target_size)

        # Compare the face with known faces using cv2.matchTemplate
```

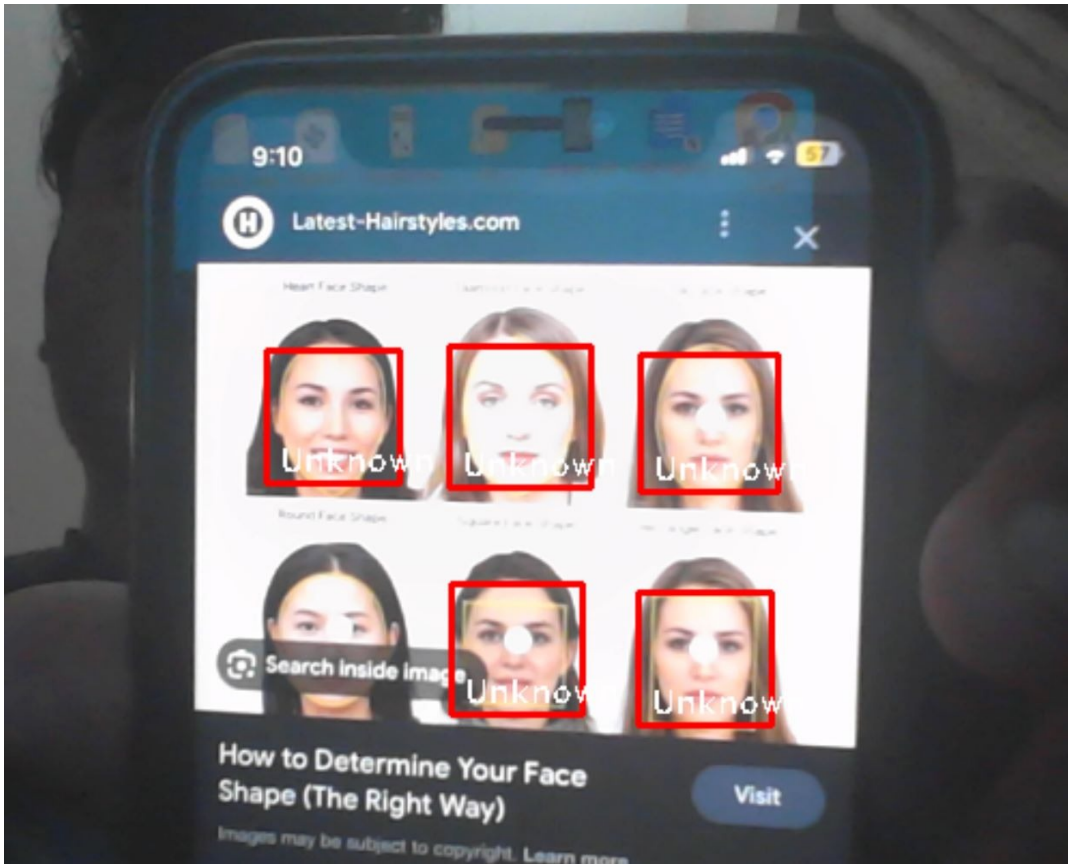
```
        found_name = "Unknown"
        for name, known_encodings in known_faces.items():
            for known_encoding in known_encodings:
                result = cv2.matchTemplate(face_region, known_encoding,
cv2.TM_CCOEFF_NORMED)
                _, max_val, _, _ = cv2.minMaxLoc(result)

                # If there's a match, set the person's name
                if max_val > 0.9:  # You can adjust this threshold to control the
recognition sensitivity
                    found_name = name
                    break

        # Display the name
        font = cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, found_name, (x + 6, y + h - 6), font, 0.5, (255, 255, 255),
1)

    # Display the resulting frame
    cv2.imshow('Face Recognition', frame)

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release video capture and close the window
video_capture.release()
cv2.destroyAllWindows()
```

Results:

The developed face recognition system demonstrates successful real-time face detection and identification. The Haar Cascade Classifier efficiently detects faces in video frames, and matching techniques accurately recognize known individuals. The multithreading project significantly enhances the responsiveness of the face detection process, enabling smooth real-time performance even with large video feeds.


Ankush


adil



Conclusion:

The "Real-time Face Recognition using Haar Cascades" project showcases the implementation of a robust and efficient face recognition system. By leveraging Haar Cascade Classifier and matching techniques, the system achieves accurate and real-time face detection and identification. Additionally, the separate multithreading project optimizes the face detection process, ensuring improved responsiveness for real-world applications.

Future Enhancements:

- Integration of deep learning-based face recognition models for enhanced accuracy and versatility.
- Deployment of the face recognition system in real-world scenarios, such as access control and attendance management.