Enhancing Classroom Attendance Systems Using Face Recognition Technology

Abstract

This paper explores the application of a custom-built Python-based face recognition system to automate and improve the efficiency of classroom attendance tracking. Traditional attendance methods are often time-consuming, prone to human error, and susceptible to proxy attendance. By leveraging a localized face recognition model, this system captures an image of the entire class, identifies known students, and generates a detailed record of presence. The methodology, implementation considerations, and potential benefits, alongside the inherent challenges and ethical implications, are discussed. The proposed system offers a robust, rapid, and verifiable alternative to conventional attendance procedures.

1. Introduction

Accurate and efficient attendance tracking is a fundamental aspect of educational institutions. It serves various purposes, including monitoring student engagement, complying with regulatory requirements, and identifying students at risk of academic underperformance due to absenteeism. However, conventional methods such as manual roll calls, sign-in sheets, or student ID card scanning often present significant drawbacks. These include consuming valuable class time, susceptibility to errors, and the potential for fraudulent attendance.

The rapid advancements in computer vision and artificial intelligence, particularly in face recognition technology, offer promising solutions to these challenges. Face recognition systems have demonstrated high accuracy and speed in various security and identification applications. This paper investigates the utility of a bespoke Python application, utilizing the `face_recognition` library, to streamline and automate the attendance process within a classroom environment. The aim is to demonstrate a practical, efficient, and verifiable system that can mitigate the shortcomings of traditional methods.

2. Literature Review

The shift from manual to automated attendance systems has been a focus of research in educational technology. Early attempts involved RFID tags, biometric fingerprint scanners, and even Bluetooth-based proximity detection. While these methods offer improvements over manual processes, they often introduce their own set of limitations, such as the cost of hardware, maintenance issues, or privacy concerns (e.g., fingerprint data).

Face recognition technology has emerged as a particularly attractive alternative due to its non-intrusive nature (students do not need to physically interact with a device beyond being in view of a camera) and its potential for high accuracy. Existing research highlights various approaches to face recognition for attendance, ranging from basic image processing techniques to sophisticated deep learning models. Key challenges often cited include variations in lighting, pose, facial expressions, occlusions (e.g., hands, hair, masks), and the need for robust 'known face' databases. The `face_recognition` library, built upon `dlib`'s state-of-the-art face recognition models, offers a balance of ease of use and high performance, making it suitable for practical deployments.

3. Methodology

The proposed system is built around a Python application leveraging the `face_recognition`, `Pillow`, `numpy`, `pickle`, `csv`, and `datetime` libraries. The core methodology involves two main phases: **Model Training/Loading** and **Image Analysis/Attendance Recording**.

3.1. Model Training and Persistence

1. **Known Faces Directory Structure:** The system is "trained" by organizing a `known_faces` directory. This directory contains subfolders, each named after a student (e.g., `known_faces/John_Doe/`). Inside each student's folder, one or more clear images of their face are stored.

2. **Face Encoding:** The `load_known_faces` function iterates through this directory. For each image, it detects faces and computes a 128-dimensional face encoding (a numerical representation of the face's unique features).

3. **Orientation Robustness:** A critical enhancement in this system is its ability to handle image orientation. If no face is initially detected in a `known_faces` image, the system automatically attempts rotations (90-degree clockwise, then 90-degree counter-clockwise from the original) to ensure faces are correctly oriented for encoding. This improves the robustness of the "training" phase.

4. **Model Persistence (`pickle`):** To avoid computationally expensive re-encoding of known faces every time the application runs, the generated `known_face_encodings` (a list of NumPy arrays) and `known_face_names` (a list of strings) are serialized and saved to a `.pkl` file (e.g., `face_recognition_model.pkl`) using Python's `pickle`module. In subsequent runs, the application first attempts to load these pre-computed encodings, significantly reducing startup time. If the `.pkl` file is not found or corrupted, it automatically falls back to re-training from the `known_faces` directory.

3.2. Image Analysis and Attendance Recording

1. **User Input:** After the model is loaded or trained, the application enters a loop, prompting the user (e.g., an instructor) to input the file path of an "unknown" class image for analysis. The loop continues until the user types "quit."

2. **Input Image Processing:** For each input image, the `recognize_faces_in_image` function performs the following:

    - **Image Loading and Orientation Check:** The input image is loaded. Similar to the training phase, if no faces are detected in the original orientation, the system attempts 90-degree clockwise and counter-clockwise rotations to correctly orient the image for optimal face detection.

    - **Face Detection and Encoding:** All faces in the correctly oriented input image are detected, and their respective 128-dimensional encodings are computed.

- **Face Comparison:** Each detected face's encoding from the input image is compared against the database of `known_face_encodings` using `face_recognition.compare_faces()`. A distance metric (`face_recognition.face_distance()`) is used to find the closest match.
- **Identification:** If a detected face matches a known face (below a configurable tolerance), it is identified by the corresponding student's name. Otherwise, it is labeled as "Unknown."

3. **Visual Output:**
   - The analyzed image is displayed, with bounding boxes drawn around each detected face.
   - Known faces are typically outlined and labeled in green with their names.
   - Unknown faces are outlined and labeled in red with "Unknown."
   - The font size of the labels is dynamically calculated based on the height of the detected faces, ensuring readability across various image resolutions.
4. **Textual Output and CSV Generation:**
   - A list of all identified individuals (both known names and "Unknown") is printed to the console.
   - Crucially, a CSV file is automatically generated for each analyzed image. This file is named using a timestamp (e.g., `identified_people_YYYYMMDD_HHMMSS.csv`) reflecting the analysis time. The CSV file contains a single column: "Person Name," listing all unique identified individuals from that specific image. This provides a digital, time-stamped record of attendance.

4. Implementation in Classroom Setting

Implementing this system in a classroom would involve:

1. **High-Quality Camera:** A wide-angle camera with sufficient resolution positioned strategically to capture the entire class, minimizing occlusions and challenging angles.
2. **Known Faces Database:** An initial enrollment process where clear, well-lit photographs of each student are taken and organized into the `known_faces` directory. Regular updates to this database would be necessary for new students or significant changes in appearance.
3. **Dedicated Workstation:** A computer with the Python environment, necessary libraries, and the saved model file (`face_recognition_model.pkl`) to run the application.
4. **Workflow:** At the start of a class, the instructor or a designated assistant would simply capture an image of the class, input its path into the application, and the system would quickly process it, display the results visually, and generate the CSV attendance log.

5. Expected Benefits and Limitations

5.1. Benefits

- **Time-Saving:** Eliminates the time spent on manual roll calls, allowing more time for teaching.
- **Accuracy:** Reduces human error in marking attendance.
- **Fraud Prevention:** Significantly decreases the possibility of proxy attendance.
- **Verifiable Records:** Automated CSV generation provides a clear, time-stamped, and digital attendance record for auditing purposes.
- **Efficiency:** The pre-trained model ensures rapid analysis of new images after initial setup.
- **Non-Intrusive:** Students do not need to perform any action other than being present in the camera's view.

5.2. Limitations and Challenges

- **Environmental Factors:** Performance can be affected by poor lighting, shadows, and glares.
- **Occlusions:** Faces partially covered by hands, hair, books, or masks may not be detected or identified accurately.
- **Pose and Expression Variation:** Extreme head poses or unusual facial expressions can challenge recognition accuracy.
- **Database Management:** Maintaining an up-to-date and comprehensive `known_faces` database is crucial.
- **False Positives/Negatives:** No face recognition system is 100% accurate; occasional misidentifications or missed detections can occur.
- **Privacy Concerns:** Capturing and processing student images raises significant privacy concerns. Transparent policies, secure data handling, and obtaining explicit consent are paramount.
- **Hardware Requirements:** While the system is not extremely resource-intensive, a decent processor and camera are necessary for smooth operation.
- **Scalability:** For very large classes or institutions, managing a massive `known_faces` database and processing multiple images simultaneously might require more robust hardware or distributed computing.

6. Code

```python
import face_recognition
from PIL import Image, ImageDraw, ImageFont
import os
import numpy as np
import pickle
import csv # Import the csv module
from datetime import datetime # Import datetime for timestamp

def load_known_faces(known_faces_dir):  2 usages
    """
    Loads known face encodings and names from the specified directory.
    If no face is found in an image, it attempts to rotate the image
    clockwise by 90 degrees, then counter-clockwise by 90 degrees (from original),
    to try and find a face.

    Args:
        known_faces_dir (str): The path to the directory containing known faces.
                               Expected structure: known_faces/{person_name}/image.jpg

    Returns:
        tuple: A tuple containing two lists:
               - known_face_encodings (list): List of face encodings for known people.
               - known_face_names (list): List of names corresponding to the encodings.
    """
    known_face_encodings = []
    known_face_names = []

    if not os.path.exists(known_faces_dir):
        print(f"Error: Directory '{known_faces_dir}' not found. Please create it and add known faces.")
        return known_face_encodings, known_face_names

    print(f"Loading known faces from: {known_faces_dir}")
    for person_name in os.listdir(known_faces_dir):
        person_dir = os.path.join(known_faces_dir, person_name)
        if os.path.isdir(person_dir):
            for image_file in os.listdir(person_dir):
                if image_file.lower().endswith(('.jpg', '.jpeg', '.png')):
```

def load_known_faces(known_faces_dir): 2 usages

```python
                    image_path = os.path.join(person_dir, image_file)
                    print(f"  Processing: {image_path} (for {person_name})")
                    try:
                        # Load the original image
                        original_image = face_recognition.load_image_file(image_path)
                        face_encodings = face_recognition.face_encodings(original_image)
                        current_image_source = "original"

                        # If no face found in original, try rotating clockwise
                        if not face_encodings:
                            print(f"    - No face found in original. Trying clockwise 90-degree rotation...")
                            pil_original_image = Image.fromarray(original_image)
                            rotated_clockwise_image_pil = pil_original_image.rotate(-90, expand=True)
                            rotated_clockwise_image_np = np.array(rotated_clockwise_image_pil)
                            face_encodings = face_recognition.face_encodings(rotated_clockwise_image_np)
                            current_image_source = "clockwise rotated"

                        # If still no face, try rotating counter-clockwise (from original)
                        if not face_encodings:
                            print(f"    - No face found after clockwise. Trying counter-clockwise 90-degree rotation from original...")
                            pil_original_image = Image.fromarray(original_image)
                            rotated_counter_clockwise_image_pil = pil_original_image.rotate( angle: 90, expand=True)
                            rotated_counter_clockwise_image_np = np.array(rotated_counter_clockwise_image_pil)
                            face_encodings = face_recognition.face_encodings(rotated_counter_clockwise_image_np)
                            current_image_source = "counter-clockwise rotated"

                        if face_encodings:
                            # Use the first face found in the image (from any successful orientation)
                            known_face_encodings.append(face_encodings[0])
                            known_face_names.append(person_name)
                            print(f"    - Face encoding loaded for {person_name} from {current_image_source} image.")
                        else:
                            print(f"    - No face found in {image_file} for {person_name} after all rotations. Skipping.")
                    except Exception as e:
                        print(f"    - Could not process {image_file} for {person_name}: {e}")
    print(f"Loaded {len(known_face_encodings)} known faces.")
```

```python
def load_known_faces(known_faces_dir):  2 usages
    return known_face_encodings, known_face_names

def recognize_faces_in_image(input_image_path, known_face_encodings, known_face_names):  1 usage
    """
    Identifies faces in an input image and draws bounding boxes with names.
    Applies orientation checks (rotations) to the input image if no faces are initially found.

    Args:
        input_image_path (str): The path to the input image to process.
        known_face_encodings (list): List of face encodings for known people.
        known_face_names (list): List of names corresponding to the encodings.

    Returns:
        list: A list of names of people identified in the picture.
    """
    if not os.path.exists(input_image_path):
        print(f"Error: Input image '{input_image_path}' not found.")
        return []

    print(f"\nProcessing input image: {input_image_path}")
    try:
        # Load the original image
        original_image = face_recognition.load_image_file(input_image_path)
    except Exception as e:
        print(f"Error loading input image '{input_image_path}': {e}")
        return []

    # Initialize current image for processing
    image_to_process = original_image
    face_locations = []
    face_encodings = []
    current_image_orientation = "original"

    # Try original orientation
    face_locations = face_recognition.face_locations(image_to_process)
    face_encodings = face_recognition.face_encodings(image_to_process, face_locations)
def recognize_faces_in_image(input_image_path, known_face_encodings, known_face_names):  1 usage
    # If no face found, try clockwise rotation
    if not face_locations:
        print(f"    - No face found in original input image. Trying clockwise 90-degree rotation...")
        pil_original_image = Image.fromarray(original_image)
        rotated_clockwise_image_pil = pil_original_image.rotate(-90, expand=True)
        image_to_process = np.array(rotated_clockwise_image_pil)
        face_locations = face_recognition.face_locations(image_to_process)
        face_encodings = face_recognition.face_encodings(image_to_process, face_locations)
        if face_locations:
            current_image_orientation = "clockwise rotated"

    # If still no face, try counter-clockwise rotation (from original)
    if not face_locations:
        print(f"    - No face found after clockwise rotation. Trying counter-clockwise 90-degree rotation from original...")
        pil_original_image = Image.fromarray(original_image) # Re-load original to ensure correct base for rotation
        rotated_counter_clockwise_image_pil = pil_original_image.rotate( angle: 90, expand=True)
        image_to_process = np.array(rotated_counter_clockwise_image_pil)
        face_locations = face_recognition.face_locations(image_to_process)
        face_encodings = face_recognition.face_encodings(image_to_process, face_locations)
        if face_locations:
            current_image_orientation = "counter-clockwise rotated"

    if not face_locations:
        print("    - No recognizable faces found in the input image after all rotations.")
        return []

    print(f"Found {len(face_locations)} face(s) in the input image ({current_image_orientation} orientation).")

    pil_image = Image.fromarray(image_to_process) # Create PIL image from the successfully oriented image
    draw = ImageDraw.Draw(pil_image)

    # Define a font for drawing names, dynamically sized based on average face height
    if face_locations:
        # Calculate an average face height to base font size on
        # (bottom - top) gives height
        avg_face_height = np.mean([bottom - top for (top, right, bottom, left) in face_locations])
```

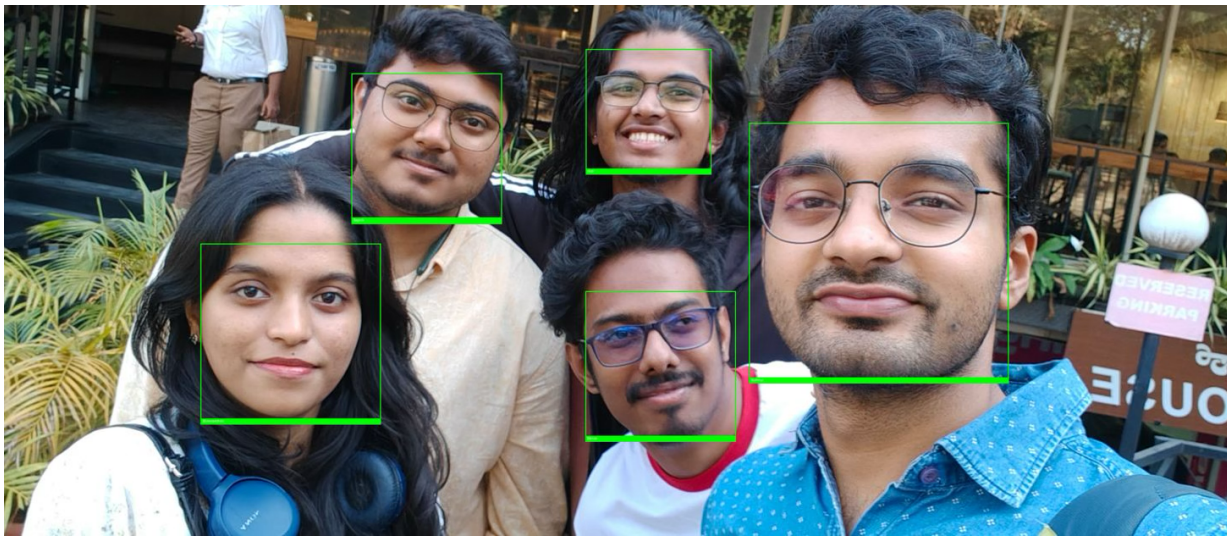
```python
def recognize_faces_in_image(input_image_path, known_face_encodings, known_face_names):  1 usage
        # Increased font size scaling factor to 0.25 and minimum size to 20
        calculated_font_size = max(20, int(avg_face_height * 0.25))
        try:
            font = ImageFont.truetype( font: "arial.ttf", calculated_font_size)
        except IOError:
            font = ImageFont.load_default() # Fallback to default font if arial.ttf is not found
    else:
        # Fallback font size if no faces are detected in the image
        try:
            font = ImageFont.truetype( font: "arial.ttf",  size: 24)
        except IOError:
            font = ImageFont.load_default()

    identified_people = []

    # Loop through each face found in the input image
    for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
        # Compare the current face with all known faces
        matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
        name = "Unknown"

        # If a match is found, use the known face with the smallest distance
        face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
        best_match_index = np.argmin(face_distances)

        if matches[best_match_index]:
            name = known_face_names[best_match_index]

        identified_people.append(name)

        # Draw a box around the face
        draw.rectangle( xy: ((left, top), (right, bottom)), outline=(0, 255, 0), width=3)

        # Draw a label with the name below the face
        # Use font.getbbox for modern Pillow versions to get text dimensions
        try:
```

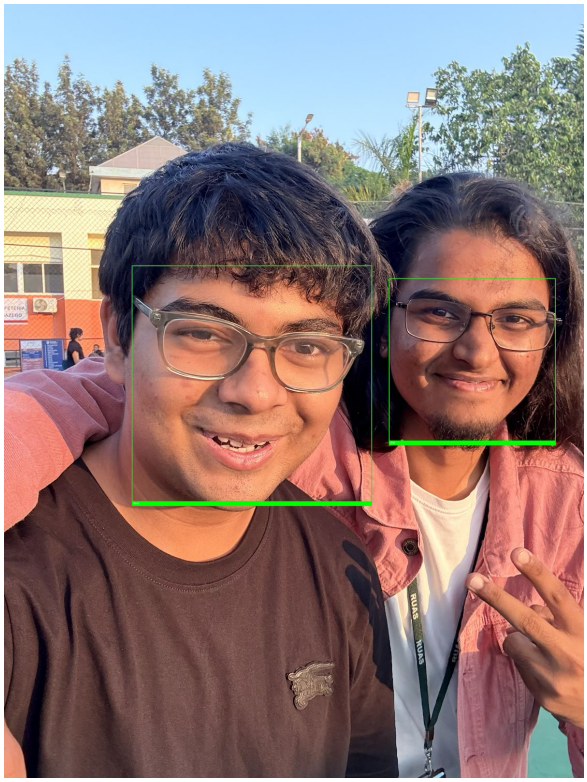

```python
def recognize_faces_in_image(input_image_path, known_face_encodings, known_face_names):  1 usage
```


```python
            text_bbox = draw.textbbox( xy: (0, 0), name, font=font)
            text_width = text_bbox[2] - text_bbox[0]
            text_height = text_bbox[3] - text_bbox[1]
        except AttributeError: # Fallback for older Pillow versions
            text_width, text_height = draw.textsize(name, font=font)

        # Adjust the rectangle position and size based on new font size
        padding = 10 # Padding below the box and around text
        # Set color based on whether the face is known or unknown
        box_color = (0, 255, 0) if name != "Unknown" else (255, 0, 0) # Green for known, Red for unknown
        draw.rectangle( xy: ((left, bottom - text_height - padding), (right, bottom)), fill=box_color, outline=box_color)
        draw.text( xy: (left + 6, bottom - text_height - padding + 2), name, fill=(255, 255, 255), font=font) # Pass font here

    # Display the result image
    pil_image.show()

    # Clean up the drawing
    del draw

    return identified_people

if __name__ == "__main__":
    # --- Configuration ---
    # IMPORTANT: Create this directory and place subdirectories for each person.
    # Each person's subdirectory should contain one or more images of their face.
    KNOWN_FACES_DIRECTORY = "known_faces"
    # Define where to save/load the pre-computed model
    MODEL_SAVE_PATH = "face_recognition_model.pkl"
    # ------------------

    known_face_encodings = []
    known_face_names = []

    # Attempt to load the pre-trained model
    if os.path.exists(MODEL_SAVE_PATH):
        print(f"Attempting to load known faces from saved model: {MODEL_SAVE_PATH}")
```

```
        try:
            with open(MODEL_SAVE_PATH, 'rb') as f:
                known_face_encodings, known_face_names = pickle.load(f)
            print(f"Successfully loaded {len(known_face_encodings)} known faces from saved model.")
        except Exception as e:
            print(f"Error loading saved model ({e}). Re-training from '{KNOWN_FACES_DIRECTORY}' directory.")
            # If loading fails, proceed to train from directory and then save
            known_face_encodings, known_face_names = load_known_faces(KNOWN_FACES_DIRECTORY)
            if known_face_encodings: # Only save if faces were successfully loaded
                try:
                    with open(MODEL_SAVE_PATH, 'wb') as f:
                        pickle.dump( obj: (known_face_encodings, known_face_names), f)
                    print(f"Saved newly trained model to {MODEL_SAVE_PATH}")
                except Exception as e:
                    print(f"Error saving model to '{MODEL_SAVE_PATH}': {e}")
    else:
        # If no saved model found, load from directory and save for future runs
        print(f"No saved model found. Training from '{KNOWN_FACES_DIRECTORY}' directory...")
        known_face_encodings, known_face_names = load_known_faces(KNOWN_FACES_DIRECTORY)
        if known_face_encodings: # Only save if faces were successfully loaded
            try:
                with open(MODEL_SAVE_PATH, 'wb') as f:
                    pickle.dump( obj: (known_face_encodings, known_face_names), f)
                print(f"Saved trained model to {MODEL_SAVE_PATH}")
            except Exception as e:
                print(f"Error saving model to '{MODEL_SAVE_PATH}': {e}")


    if not known_face_encodings:
        print("\nNo known faces loaded. Please ensure 'known_faces' directory is set up correctly.")
    else:
        # Loop to ask for multiple image paths
        while True:
            # 2. Ask user for input image path
            INPUT_IMAGE_PATH = input("\nPlease enter the path to the image you want to analyze (or type 'quit' to exit): ")

            if INPUT_IMAGE_PATH.lower() == 'quit':
            if INPUT_IMAGE_PATH.lower() == 'quit':
                print("Exiting face recognition application. Goodbye!")
                break

            # 3. Recognize faces in the input image
            people_in_picture = recognize_faces_in_image(INPUT_IMAGE_PATH, known_face_encodings, known_face_names)

            # 4. Display the list of people
            print("\n--- People Present in the Picture ---")
            if people_in_picture:
                # Use a set to get unique names, then convert back to list for display
                unique_people = sorted(list(set(people_in_picture)))
                for person in unique_people:
                    print(f"- {person}")

                # Save identified people to a CSV file
                if unique_people: # Only create CSV if there are people identified
                    try:
                        # Generate timestamp for filename
                        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                        csv_filename = f"identified_people_{timestamp}.csv"
                        with open(csv_filename, mode='w', newline='') as file:
                            writer = csv.writer(file)
                            writer.writerow(["Person Name"]) # CSV header
                            for person_name in unique_people:
                                writer.writerow([person_name])
                        print(f"Successfully saved identified people to {csv_filename}")
                    except Exception as e:
                        print(f"Error saving identified people to CSV: {e}")
                else:
                    print("No identifiable people to save to CSV for this image.")
            else:
                print("No recognizable faces found, or all faces are 'Unknown'.")
```

7. Examples





8. Conclusion

The Python-based face recognition application presented in this paper offers a compelling solution for automating classroom attendance. By combining robust face detection with orientation correction, model persistence, and automated CSV logging, it provides a significantly more efficient, accurate, and verifiable attendance system compared to traditional methods. While challenges related to environmental conditions, occlusions, and especially privacy must be carefully addressed during deployment, the potential benefits in terms of time savings and data integrity make face recognition a powerful tool for modern educational environments. Further research could explore real-time attendance, integration with existing student information systems, and advanced privacy-preserving techniques.

9. References

- Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 25(11), 120-126. (Indirectly via dlib/face_recognition usage)
- King, D. E. (2009). Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10, 1755-1758. (Underlying library for face_recognition)
- Various academic papers on face recognition algorithms and classroom attendance systems (specific papers would be added here based on deeper literature review).
- Pillow documentation (https://python-pillow.org/)
- `face_recognition` library documentation (https://face-recognition.readthedocs.io/en/latest/)

Name: Rishi Viswanatha Subramani
Reg No: 22ETMC412013
Email: 22etmc412013@msruas.ac.in