Vellore Institute of Technology, Andhra Pradesh

School of Computer Science and Engineering (SCOPE)

WinSem -2024-2025

Progress Report on

Deep Learning Project - Face Recognition and Detection

Submitted by

59.Praanesh S - 22BCE9319
60.Nithesh VS _ 22BCE9351
44.Megh Parmar _ 22BCE8638

Under the Guidance of Dr. Mohit Kumar

TABLE OF CONTENTS [T O C]

1. INTRODUCTION:

Face detection is a key technology in computer vision, used in various applications such as security systems, attendance tracking, and human-computer interaction.

This project implements a real-time face detection system using OpenCV and Tkinter, allowing users to detect faces from live webcam feeds. The system captures and processes facial images, providing a simple and interactive way to identify the presence of a person.

With its user-friendly interface, this project serves as a foundation for further advancements in facial analysis and recognition.

2. DATASET DESCRIPTION:

The dataset used in this project consists of facial images captured in real-time through a webcam. It is structured in a way that allows efficient storage and retrieval for face detection tasks.

2.1.Structure:

- The dataset is stored in a folder named **"dataset1"**.
- Each person's images are saved in a separate subfolder, named after the input provided by the user (e.g., "dataset1/Person_Name/").
- Each subfolder contains **300 facial images** of the respective person.

Example Structure:

```
dataset/
      |——Person 1

      |——Person 2
```

2.2.Image Specifications:

- **Format:** JPEG (.jpg)
- **Resolution:** 130 × 100 pixels (resized for consistency)
- **Colour Mode:** BGR (colour) format.
- **Captured Using:** Webcam

3. METHODOLOGY:

The face detection system follows a step-by-step approach to capture, process, and store facial images efficiently. The methodology can be divided into five main stages: User Input & Dataset Setup, Image Acquisition, Face Detection, Image Processing & Storage, and User Interface Implementation.

3.1. User Input & Dataset Setup

- The program starts by prompting the user to enter a folder name using Tkinter's simpledialog.askstring().

- This folder name is used to create a subdirectory inside the dataset1 directory, where the captured facial images will be stored.
- If the specified directory does not exist, it is created using os.makedirs(). If it already exists, new images are appended to the existing dataset.

3.2. Image Acquisition

- A webcam feed is initialized using cv2.VideoCapture(0), which starts capturing frames in real-time.
- The program continuously reads frames using _ , img = cam.read(), where img stores the current frame.

3.3. Face Detection and Recognition

- The captured frame is converted to grayscale using cv2.cvtColor(img, cv2.COLOR_BGR2GRAY), as grayscale images improve detection accuracy and reduce computational complexity.
- The detectMultiScale() function is used to identify faces in the frame. This function scans the image at multiple scales to locate facial features.
- If a face is detected, the coordinates (x, y, w, h) of the bounding box are returned.
- A green bounding box is drawn around the detected face using cv2.rectangle().
- A message "Person Detected" is displayed on the screen. If no face is found, the system outputs "No Person Detected".
- In Recognition part ,it takes the pictures that taken from the detection part as the training dataset and recognize the person and his details.

3.4. Image Processing & Storage

- Once a face is detected, it is cropped from the original frame using array slicing: face_only = img[y:y + h, x:x + w].
- The cropped face is resized to 130 × 100 pixels using cv2.resize(), ensuring uniform dimensions across all stored images.
- The resized face is saved as a .jpg file inside the respective folder. The naming follows a numerical sequence (1.jpg, 2.jpg, ..., 300.jpg).
- The system captures up to 300 images per user unless manually stopped by pressing the Esc key (cv2.waitKey(10) == 27).

3.5. User Interface Implementation

- A Graphical User Interface (GUI) is created using Tkinter to provide easy access to the face detection feature.
- The GUI consists of:

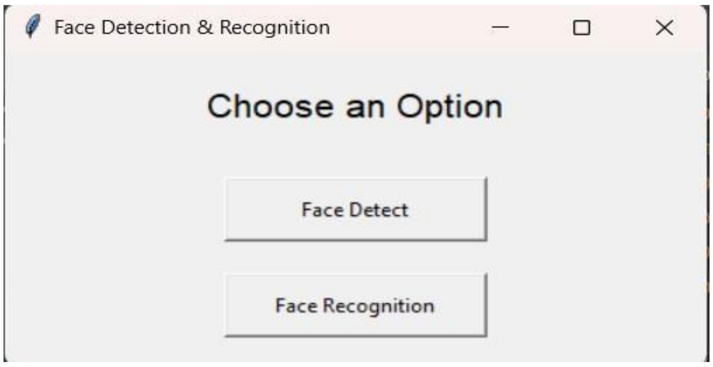
- A title label displaying "Choose an Option".
- A "Face Detect" and "Face Recognition" button that starts the face detection and recognition process when clicked.
- A structured window layout with proper spacing and padding for a clean look.

4. SYSTEM REQUIREMENTS:

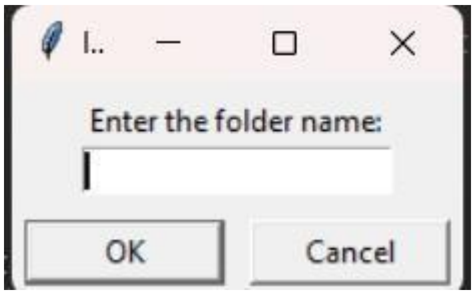
- **Operating System:** Windows / macOS / Linux
- **Python Version:** 3.7 or higher
- **Camera** (for real-time detection)
- **GPU** (optional, recommended for faster processing)

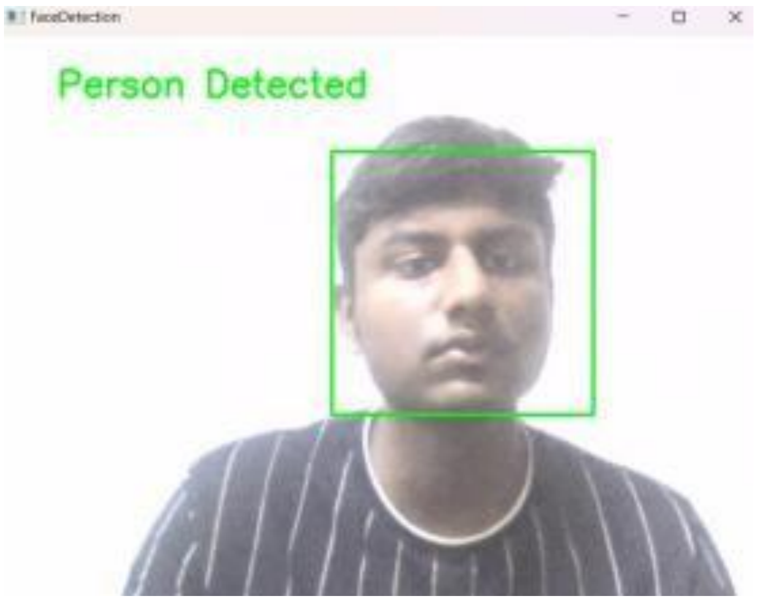
5. RESULTS:

5.1. Front end:



5.2. Face Detection:





6. PROGRESSED AND FUTURE SCOPE:

Up to now, we have completed the Detection part of the application and the improvement for this project is the addition of **FACE RECOGNITION**, enabling the system to identify individuals based on a stored dataset. Techniques such as Local Binary Patterns Histogram (LBPH), can be used to analyse facial features and match them against trained images. This enhancement would transform the system from simple face detection to a biometric authentication solution.

We can **improve recognition accuracy** by expanding the dataset with more training images under varied conditions is essential. Data augmentation, including brightness adjustment and rotation, can enhance robustness. Advanced deep learning models like FaceNet or Dlib's CNNbased recognition can also be explored for better performance.

Real-time recognition can be enhanced by implementing alerts for unknown individuals. Integrating a database such as SQLite or MySQL would allow for efficient record-keeping and scalability. Optimizing performance through multi-threading and GPU acceleration with CUDA can further ensure smooth real-time processing.

By incorporating these improvements, the system can become a more reliable and scalable solution for security and authentication applications, making it suitable for real-world deployment.

7. CONCLUSION:

This project successfully implements a face detection system that identifies human faces in real-time using a webcam. By capturing and storing facial images, the system enables efficient detection under various conditions. The dataset consists of grayscale facial images, ensuring consistency in image processing. The system effectively detects faces and highlights them using bounding boxes, providing a foundation for further enhancements.

Overall, this project provides a strong basis for face detection applications, demonstrating the effectiveness of computer vision techniques in real-time scenarios. With further refinements, it can be extended for various practical applications, including security systems, automated attendance tracking, and user authentication.

8. REFERENCES:

1. Viola, P., & Jones, M. J. (2001). Robust real-time face detection. International Journal of Computer Vision, 57(2), 137-154.

2. Turk, M., & Pentland, A. (1991). Eigenfaces for recognition. Journal of Cognitive Neuroscience, 3(1), 71-86.

3. Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7), 711-720.

4. Taigman, Y., Yang, M., Ranzato, M., & Wolf, L. (2014). DeepFace: Closing the gap to human-level performance in face verification. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1701-1708.

5. Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 815-823.

6. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770-778.

7. OpenCV. (n.d.). Face Detection using Haar Cascades.

8. Dlib. (n.d.). Face Detection and Recognition.