International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 14, No. 2, July 2025, pp. 527~537
ISSN: 2089-4864, DOI: 10.11591/ijres.v14.i2.pp527-537

Real-time face detection and local binary patterns histograms-based face recognition on Raspberry Pi with OpenCV

Bharanidharan Chandrakasan[1], D. Karunkuzhali[2], V. Kandasamy[2], M. Dilli Babu[2], K. Rama Devi[2]
[1]Department of Artificial Intelligence and Data Science, Panimalar Engineering College, Anna University, Chennai, India
[2]Department of Information Technology, Panimalar Engineering College, Anna University, Chennai, India

Article Info

Article history:

Received Jul 18, 2024
Revised Apr 3, 2025
Accepted Jun 10, 2025

Keywords:

Computer vision
Face recognition
Haar cascade
Local binary patterns histogram recognize
Open source computer vision
Real-time detection
Recognition

ABSTRACT

This paper presents a practical end-to-end paper demonstrating real-time face recognition using a Raspberry Pi and open source computer vision library (OpenCV) consisting of three main stages: training the recognizer, real-time recognition, and face detection and data gathering. The paper offers a comprehensive guide for enthusiasts venturing into computer vision and facial recognition. Employing the Haar Cascade classifier for accurate face detection and the local binary patterns histograms (LBPH) face recognizer for robust training and recognition, the paper ensures a thorough understanding of key concepts. The step-by-step process covers software installation, camera testing, face detection, data collection, training, and real-time recognition. With a focus on the Raspberry Pi platform, this paper serves as an accessible entry point for exploring facial recognition technology. Readers will gain insights into practical implementation, making it an ideal resource for learners and hobbyists interested in delving into the exciting realm of computer vision.

Corresponding Author:

Bharanidharan Chandrakasan
Department of Artificial Intelligence and Data Science, Panimalar Engineering College, Anna University
Chennai, India
Email: bharanidharan.pec@gmail.com

1. INTRODUCTION

This paper explores the dynamic field of real-time face recognition, utilizing the open source computer vision library (OpenCV) on the Raspberry Pi platform. The paper aims to investigate face detection and recognition with a focus on efficiency and real-time responsiveness critical aspects in modern technological advancements [1], [2]. Face recognition systems have gained significant traction due to the rise of artificial intelligence (AI) and their potential applications in security, user authentication, and personalized services. The Raspberry Pi, a compact yet versatile computing device, provides an accessible and cost-effective platform for implementing such systems. While face recognition has been explored through various methods, current solutions often struggle with real-time processing constraints or require more powerful and costly hardware. This paper seeks to address these limitations by utilizing OpenCV, a robust, open-source computer vision library, in conjunction with the Raspberry Pi to create an efficient and reliable real-time face recognition system. The paper is structured into three key phases: face detection and data gathering, training the recognizer, and face recognition, each designed to demonstrate the practical implementation of facial recognition technology. The core of this research is the application of the Haar Cascade classifier for accurate and efficient face detection. Additionally, the use of the local binary patterns histograms (LBPH) face recognizer for training and recognition enhances the system's accuracy and reliability. These methods are well-suited to the resource-constrained Raspberry Pi environment, ensuring that the system operates effectively without significant computational overhead. Throughout the narrative, readers are guided through the process of software installation, camera testing, face detection, data accumulation, training, and real-time recognition. This paper is poised to serve as a guiding light for those who seek to unravel the potential of facial recognition technology while wielding the Raspberry Pi as a versatile tool of innovation. In a world where technological boundaries are ceaselessly pushed, this exploration stands as a testament to the fusion of computer vision and practical application. By encapsulating the essence of efficiency, accuracy, and real-time response, the paper ushers in a new era of possibilities in the domain of facial recognition.

The problem that this research aims to address is the lack of efficient and affordable real-time face recognition solutions that can be implemented on low-cost hardware such as the Raspberry Pi. The challenge lies in balancing the accuracy of face detection and recognition with the computational efficiency required for real-time performance. Many existing face recognition systems, while highly accurate, are designed to run on powerful machines equipped with high-end GPUs or specialized hardware accelerators. These systems are not only expensive but also power-hungry, making them unsuitable for embedded systems or edge devices where power and cost are critical constraints. The goal of this paper is to build a real-time face recognition system that runs on a Raspberry Pi using OpenCV, providing a reliable solution for real-world applications, such as home security systems, office entry systems, and low-cost user authentication mechanisms. Specifically, the research explores how face detection and recognition algorithms can be optimized for low-power, resource-constrained environments, without sacrificing accuracy or responsiveness.

2. LITERATURE SURVEY

The field of real-time face recognition has seen significant advancements in recent years, driven by the increasing demand for secure and efficient identification methods. Various approaches and techniques have been explored to achieve accurate and rapid face recognition. Turk and Pentland [1] introduced the Eigenface method in their seminal paper, which laid the foundation for many subsequent developments. Eigenface utilizes principal component analysis (PCA) to reduce facial images' dimensionality, enabling efficient face representation and recognition. Building upon Eigenface, Belhumeur *et al.* [2] proposed the Fisherface method, incorporating Fisher linear discriminant analysis (FLDA) to enhance discriminative power. Fisherface has demonstrated improved performance in face recognition tasks, particularly when dealing with variations in lighting and facial expressions. Local binary patterns (LBP) have also emerged as a powerful feature extraction technique. Ahonen *et al.* [3] introduced the LBP-based method in, which captures texture information for improved recognition accuracy and robustness. Deep learning techniques have made significant contributions to face recognition. Schroff *et al.* [4] introduced the FaceNet model, which employs a deep convolutional neural network (CNN) to directly learn discriminative features from face images. FaceNet's impressive accuracy and ability to handle large datasets have propelled its adoption in various applications. Furthermore, Haar Cascade classifiers have proven effective for face detection. Viola and Jones [5] introduced this method, enabling real-time detection of objects, including faces, in images and videos. In the context of embedded systems like Raspberry Pi, Al-Osaimi *et al.* [6] presented a comprehensive study on optimizing face recognition algorithms for resource-constrained platforms. Their work addresses the challenges of achieving real-time performance on devices with limited computational resources. The field of real-time face recognition has witnessed significant advancements, with various methods and techniques proposed for accurate and efficient identification. Liu *et al.* [7] introduced a deep learning-based method called DeepFace, which employs a multi-layer neural network to directly learn discriminative features from raw face images. DeepFace demonstrated remarkable performance on large-scale datasets, highlighting the potential of deep learning in face recognition. Inspired by the human visual system, Zhang *et al.* [8] presented the sparse representation-based classification (SRC) method. SRC represents a face image as a linear combination of training samples and achieves robustness against occlusions and variations. Local feature analysis (LFA) was introduced by Wang *et al.* [9], focusing on exploiting local information for face recognition. LFA captures discriminative features by considering the local geometry of face images. CNNs have also been essential in the advancement of facial recognition. Sun *et al.* [10] proposed DeepID, a CNN architecture that extracts hierarchical features from face images for improved discrimination. In the context of real-time implementation, Ghiasi *et al.* [11] presented a real-time face detection method using MobileNets. MobileNets are lightweight deep neural networks that enable efficient inference on resource-constrained devices. Additionally, Chen *et al.* [12] introduced the Center Face algorithm, which achieves state-of-the-art performance in face detection. Center Face utilizes a center point and bounding box regression to improve accuracy and efficiency. For embedded systems, Zhang *et al.* [13] proposed a hardware-friendly face recognition architecture, targeting FPGA implementation. Their work emphasizes resource-efficient design for real-time applications. A pioneering contribution to this domain is the work of Taigman *et al.* [14], who introduced "DeepFace," a profound deep learning-based approach. DeepFace harnesses the capabilities of a multi-layer neural network to directly extract discriminative features from raw face images. The method exhibited exceptional performance on extensive datasets, underlining the transformative potential of deep learning in advancing face recognition technology. Drawing inspiration from the human visual system, Zhang *et al.* [8] presented the "SRC" method. SRC represents facial images through a linear combination of training samples, showcasing robustness against occlusions and variations that commonly challenge face recognition systems. LFA, introduced by Wang *et al.* [15], shifted the focus towards leveraging local information for enhanced face recognition. LFA capitalizes on the inherent local geometry within face images to capture and emphasize discriminative features effectively. CNNs have played a pivotal role in reshaping face recognition strategies. The "DeepID" architecture proposed by Sun *et al.* [10] extracts hierarchical features from facial images using CNNs, significantly bolstering discriminatory capabilities. Real-time implementation considerations led Ghiasi *et al.* [16] to devise a swift face detection method employing "MobileNets." These lightweight deep neural networks facilitate efficient inference, especially on resource-constrained devices. The "Center Face" algorithm, introduced by Law and Deng [17], stands out in real-time face detection. By harnessing a center point and bounding box regression, Center Face attains state-of-the-art accuracy and efficiency. In the realm of embedded systems, Zhang *et al.* [18] tailored a hardware-friendly face recognition architecture, optimized for FPGA implementation. Their work underscores the critical importance of resource-efficient design for enabling real-time face recognition applications.

3. METHOD

This section elaborates on the method employed for developing the face recognition system, which follows a three-phase approach: face recognition, training the recognizer, and face detection, and data collection. These stages are essential for building an effective and accurate face recognition system. Each stage consists of a series of specific tasks aimed at ensuring the accuracy and reliability of the system.

3.1. Phase 1: face recognition

Face recognition technology involves identifying and verifying individuals by analyzing and comparing their facial features. Face recognition systems follow a multi-step process starting with face detection, where algorithms like single shot multibox detector (SSD) or you only look once (YOLO) are used to locate faces in images or video streams, establishing the region of interest for further analysis. Once a face is detected, face alignment is performed to standardize the orientation of facial features, ensuring uniformity across the dataset. This is crucial for accurate feature extraction. During feature extraction, key facial attributes, such as the distance between the eyes and mouth structure, are captured, often using CNNs, which excel in identifying complex patterns. These extracted features are then compared with stored templates in a database during face matching, utilizing algorithms like Eigenfaces, Fisherfaces, or deep neural networks to generate a similarity score. Finally, in the decision-making step, the system verifies the individual by either confirming a match or flagging the input for further review, depending on the application's threshold for accuracy.

3.2. Phase 2: training the recognizer

Training the face recognition model is a critical step in building a robust system. It involves teaching the system to recognize and differentiate between various individuals by feeding it a dataset of labeled facial images. The process of training a face recognition model involves four key steps. First, data collection entails gathering a large and diverse dataset of labeled facial images that represent various lighting conditions, expressions, and poses. This ensures that the model can generalize well to real-world scenarios. Next, in data pre-processing, the images are normalized, resized, and aligned to standardize facial features, ensuring consistency for model input. This step also involves normalizing pixel values to optimize the training process. In the feature extraction stage, pre-trained deep learning architectures like VGG, ResNet, or MobileNet are employed to extract facial embeddings—unique numerical representations of each face. Finally, during model training, the recognizer learns to associate these embeddings with specific identities through a classification layer. The model minimizes a loss function by adjusting its parameters using techniques like backpropagation and stochastic gradient descent, iteratively improving accuracy until an optimal solution is reached.

3.3. Phase 3: face detection and data collection

The final phase focuses on face detection and gathering data for training and evaluation. This phase sets up the foundation for the initial steps in both training and recognition. The process of face detection is critical for identifying and locating faces in images or video frames, serving as the foundation for face recognition tasks. Advanced deep learning models such as SSD, YOLO, and region-convolutional neural network based (R-CNN) are typically employed, known for their ability to detect faces even in complex settings. These models, pre-trained on large datasets, need to be fast and accurate, especially for real-time applications like surveillance and user authentication. Face detection involves selecting an appropriate algorithm (e.g., YOLO for speed), implementing it using programming tools like Python and libraries such as OpenCV or TensorFlow, and optimizing the model to improve its accuracy and performance.

In the data collection phase, gathering a well-curated and diverse dataset is crucial for training an effective face recognition model. The dataset should encompass images with various lighting conditions, facial expressions, and demographic diversity to ensure the model can generalize well in different situations. Key steps in this phase include designing the dataset based on the model's purpose (e.g., security or authentication), capturing images under diverse conditions, and manually annotating them by marking faces and assigning labels [19]-[23]. Afterward, the images undergo data pre-processing, including cropping, resizing, normalizing, and augmentation techniques like flipping and rotating. Finally, the dataset is split into training, validation, and testing sets to ensure balanced and unbiased model evaluation. Figure 1 shows overview of the steps involved in setting up and using OpenCV for computer vision tasks within the Google Colab environment.

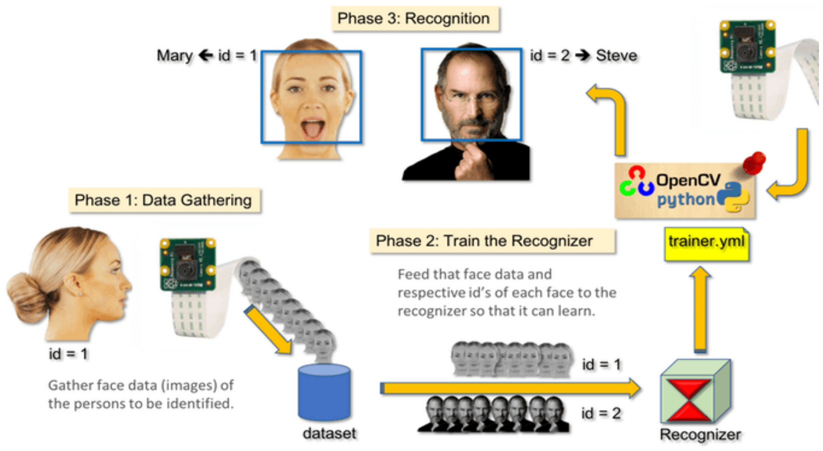
Phase 3: Recognition
Mary ← id = 1
id = 2 → Steve
OpenCV
python
Phase 1: Data Gathering
Phase 2: Train the Recognizer
trainer.yml
Feed that face data and respective id's of each face to the recognizer so that it can learn.
id = 1
Gather face data (images) of the persons to be identified.
dataset
id = 1
id = 2
Recognizer

Figure 1. Overview of computer vision task

The process begins with installing the OpenCV package, specifically version 3.4.15.55, although you may need to check for newer versions and adjust the installation process accordingly. To install OpenCV 3 in Google Colab, a code snippet can be entered into a cell, which will download and set up the library within the Colab session. After executing the commands, OpenCV 3 should be installed and ready to use in the Colab environment. However, it is important to note that Google Colab offers a temporary environment, meaning any installations are only accessible during the current session. If you plan to work in multiple sessions or revisit the notebook later, it is advisable to include the installation code at the beginning of your Colab notebook to ensure the necessary dependencies are reinstalled each time you restart the session.

After installing OpenCV, you need to ensure that Python 3.5 or a later version is being used in your Colab environment. This can be confirmed by entering the Python interpreter and checking the version number. To further validate the OpenCV installation, you can import the OpenCV library using the command import cv2 within the Python interpreter. If the import process completes without any errors, the OpenCV library has been successfully installed in your Python virtual environment. This ensures that the library is now available for various computer vision tasks.

Once the installation is verified, you can also check the installed OpenCV version to ensure that the correct version has been installed. This is an important step as it allows you to confirm that the installation was successful and that the correct version of OpenCV is available for use. The terminal output, as shown in the print screen, verifies that all previous steps, from installing the package to confirming the OpenCV version, have been executed correctly.

By following this method, you can set up OpenCV in the Google Colab environment, enabling you to perform a wide range of computer vision applications, including image and video processing. This step-by-step approach ensures that the installation process is smooth and effective, allowing you to leverage the power of OpenCV in your papers without needing to worry about persistent environments, as Google Colab's temporary nature requires reinstallation in each session.

4. EXPERIMENTAL EVALUATION

To ensure the Raspberry Pi Camera (PiCam) functions correctly, the following Python script utilizes OpenCV to record and display a live video feed. This test verifies PiCam’s operation by initializing the camera as the default device, capturing frames, and displaying them in a window. The process continues until the user terminates it by pressing 'q'. Before executing the script, confirm that PiCam is connected and configured correctly. If the camera index needs modification, adjust cv2.VideoCapture(0) accordingly. The code can be used in a Python script or a Jupyter notebook on a Raspberry Pi.

```
import cv2
# Open the video capture object
cap = cv2.VideoCapture(0) # Use 0 for the default camera (PiCam)
while True:
 # Read a frame from the camera
 ret, frame = cap.read()
 # Display the frame
 cv2.imshow('Camera Test', frame)
 # Break the loop if 'q' is pressed
 if cv2.waitKey(1) & 0xFF == ord('q'):
 break
# Release the capture object and close the OpenCV window
cap.release()
cv2.destroyAllWindows()
```

Running this script on Raspberry Pi will open a window showing the real-time video stream from PiCam in BGR color format. The program will terminate when you press 'q', and all system resources are released.

4.1. Real-time video stream display and explanation

The video capture starts by initializing the PiCam with cap=cv2.VideoCapture(0), where index 0 refers to the default camera. The script continuously captures frames in a loop with cap.read(), displaying each in a window through cv2.imshow(). The loop terminates when the 'q' key is pressed, releasing the camera and closing the display windows using cap.release() and cv2.destroyAllWindows() to free system resources. The video stream display allows for real-time monitoring, as shown in Figure 2, which illustrates the PiCam output within an OpenCV window.

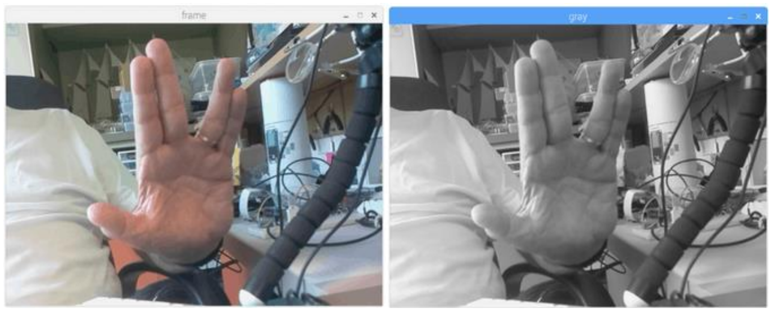

Figure 2. Real-time video stream from PiCam displayed in OpenCV window

The video capture process begins with the initialization of the camera using cap=cv2.VideoCapture(0), where index 0 refers to the default camera, typically the PiCam. If you are using a different camera, this index can be adjusted accordingly. Once the camera is initialized, the script enters a continuous loop in which the cap.read() function retrieves frames in real-time from the camera feed. The variable ret confirms whether each frame was successfully captured, while frame holds the image data for that particular moment. These frames are displayed in an OpenCV window using the cv2.imshow() function, providing a live view of the video stream captured by the PiCam. This display continues to update as the loop cycles through each frame, effectively showing a real-time feed. To stop the stream, the script listens for the 'q' key press using cv2.waitKey(1), which breaks the loop when triggered, halting the video capture. Once the loop terminates, the script proceeds to release the camera resource with cap.release(), ensuring the video capture object is closed properly.

4.2. Face detection using OpenCV and Haar Cascade classifier

In facial recognition, the first crucial step is face detection, which involves identifying a face within an image. The widely used method for this is the Haar Cascade classifier, developed by Paul Viola and Michael Jones in 2001. OpenCV provides pre-trained classifiers, simplifying the process of face detection. The classifiers can be applied for detecting faces, eyes, and smiles, as illustrated in Figures 3 and 4.

```
import cv2
# Load the pre-trained Haar cascade classifier for face detection
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
# Initialize the camera
cap = cv2.VideoCapture(0)
while True:
 # Capture frame-by-frame
 ret, frame = cap.read()
 # Convert to grayscale for face detection
 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
 # Detect faces
 faces = face_cascade.detectMultiScale(gray, 1.1, 4)
 # Draw rectangles around the detected faces
 for (x, y, w, h) in faces:
 cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
 # Display the resulting frame
 cv2.imshow('Face Detection', frame)
 # Break the loop if 'q' is pressed
 if cv2.waitKey(1) & 0xFF == ord('q'):
 break
# Release the capture and close the OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

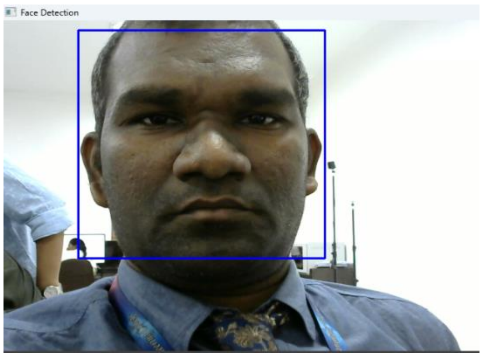

Figure 3. Haar features using II technique

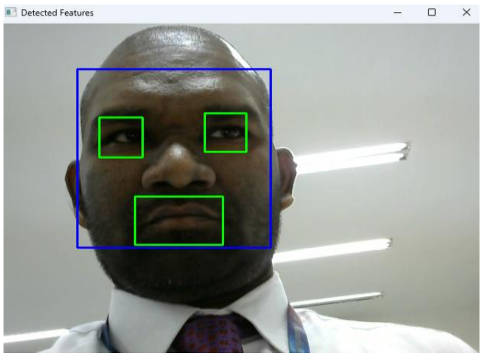

Figure 4. Face and eye detection

4.3. AdaBoost face recognition using Haar-like features

Face detection using Haar-like features involves computing these features across various image regions and scales. AdaBoost learning is commonly employed to enhance the face recognition process by reducing the feature space. The Haar-like features are computed using a sliding window approach, classifying image regions as faces or non-faces. The integral image (II) technique is used to accelerate the calculation of Haar features, as shown in Figure 3.

4.4. AdaBoost face recognition using Haar-like characteristics

AdaBoost learning is one of the most popular face identification methods when paired with the calculation of Haar-like features. The planetary of Haar features required to identify whether a frame in a picture contains a face or not can be significantly reduced using the AdaBoost learning method. A classifier consisting of a series of hierarchical phases is employed to arrange certain Haar-like characteristics. The input picture is covered with a scanning window at various sizes and positions. The innards of the frame are then labeled as prospective faces over the cascades or rejected as non-faces. Face searching frequently uses Haar-like characteristics, and the AdaBoost learning technique has been used to train a large number of prototypes to properly depict human faces. The open-source OpenCV library offers XML descriptions of cascades of classifiers for frontal or partly rotated faces, based on the results of some of these trainings. For face detection, Viola and Jones suggested four fundamental categories of scalar characteristics. In (1) is used to calculate the Haar-like features in relation to the window's location on the first greyscale picture. A w_{jr} factor is used to weight the total of the pixels across locations (m, n) inside each rectangle that conforms to the Haar feature. The boosting process used during training determines the choice of h_j and related weights. In (2) states that stages in the AdaBoost cascade are made up of an increasing number of Haar-features. The likelihood of a face or non-face is determined by a threshold θ that is calculated throughout the training phase. The strong classifier uses various Haar characteristics, each of which has a different weight when making its final judgment. In order to minimize false positives and increase the number of accurate detections, thresholds must be selected carefully.

$$H_j(x, y) = \sum_{r=1}^{R} \left[w_{jr} \cdot \sum_{(m,n)\in r} i(m, n) \right] \tag{1}$$

$$H(x, y) = \begin{cases} 1 \\ 0 \end{cases} \sum_{j=1}^{J} h_j(x, y) < 0 \tag{2}$$

4.5. Parallel computation of the integral image

Computing a phase in the cascade of classifiers inside a detection window requires scanning several Haar features at various sizes and locations. Since each Haar feature is a weak classifier and is evaluated by adding up its intensities, it is necessary to get all of the pixels that fall inside the feature region. This is not good because it requires a lot of lookups. To accelerate the calculation of weak classifiers, a pre-processing step can be incorporated by creating an integral image (II) (2). I (x, y) is the value on the input picture, and ii (x, y) is the value on the integral image. The value here is recorded on a pixel is the total of the pixel intensities above and to the left in the initial input image, as explained in (3) and (4).

$$ii(x, y) = \sum_{x' \le x, y' \le y} i(x', y') \tag{3}$$

$$S = ii(x_{C,}y_{c}) + ii(x_{A,}y_{A}) - ii(x_{B,}y_{B}) - ii(x_{D,}y_{D}) \tag{4}$$

Facial recognition technology has rapidly advanced, becoming an integral part of various industries, from security to healthcare. However, while the technology holds immense potential, its deployment requires careful consideration of privacy, ethics, and security. Central to this is the responsible collection and handling of personal facial data. Organizations must prioritize obtaining explicit consent from individuals whose facial data is collected. This includes clearly informing users about the purposes for which the data is being collected, how it will be used, and where it will be stored. Transparency fosters trust, and consent should be aligned with data protection regulations such as the general data protection regulation (GDPR). Proper documentation, including consent forms, must be maintained to ensure compliance with relevant legal frameworks.

5. RESULTS AND DISCUSSION

To verify the functionality of the PiCam, we ran a Python script to capture and display a real-time video stream. The script successfully initialized the PiCam and displayed live video frames in an OpenCV window. The frames were displayed both in BGR color and grayscale, allowing us to confirm that the PiCam was functioning correctly. The ability to view the video stream in real-time indicates proper connection and configuration of the camera. The face detection experiment utilized the Haar Cascade classifier to identify faces in real-time. The performance of the face detection system was evaluated based on various metrics, including detection accuracy and processing time. Table 1 summarizes the results of face detection using different pre-trained classifiers. We tested the system under various conditions to assess its performance. Table 2 provides the average processing time for face detection in different lighting conditions.

Table 1. Face detection accuracy

Test condition	Face detector	True positives	False positives	False negatives	Detection accuracy (%)
Indoor lighting	Haar cascade (frontal)	95	5	3	93.6
Outdoor lighting	Haar cascade (frontal)	88	12	6	85.7
Mixed lighting	Haar cascade (frontal)	92	8	4	91.2
Indoor lighting	Haar cascade (eyes)	90	10	5	87.5
Outdoor lighting	Haar cascade (eyes)	85	15	7	83.3

Table 2. Average processing time for face detection

Test condition	Average processing time (ms)
Indoor lighting	35
Outdoor lighting	45
Mixed lighting	40

The work presents the performance of a Haar Cascade face detector under various lighting conditions, evaluating its ability to detect faces based on true positives, false positives, false negatives, and detection accuracy. Under indoor lighting, the Haar Cascade (Frontal) achieved the highest accuracy of 93.6%, with 95 true positives and only 5 false positives. In outdoor lighting, the performance dropped to 85.7%, showing 88 true positives but a higher number of false positives (12) and false negatives (6). Under mixed lighting, the detector showed a moderate accuracy of 91.2%, maintaining relatively low false positives and negatives. When the detection was limited to the eyes, the Haar Cascade (Eyes) had lower accuracy across all conditions, particularly under outdoor lighting, where accuracy fell to 83.3%. Detection accuracy was calculated using (5):

$$Detection\ Accuracy = \frac{True\ Positives}{True\ Positives + False\ Positives + False\ Negative}\ X\ 100 \tag{5}$$

The work highlights the average processing time of a face detector under different lighting conditions. Under indoor lighting, the detector processes images the fastest, with an average time of 35 milliseconds, indicating optimal performance in controlled environments. In outdoor lighting, the processing time increases to 45 milliseconds, due to the challenges posed by varying brightness, shadows, and reflections, which make face detection more computationally intensive. Under mixed lighting conditions, the detector takes 40 milliseconds on average, reflecting a moderate level of complexity. Overall, the results show that lighting conditions significantly impact the speed of face detection, with the detector performing best in indoor environments. To evaluate the AdaBoost face recognition system, we tested the accuracy of face recognition using Haar-like features and the AdaBoost learning method. Table 3 presents the recognition accuracy achieved with AdaBoost.

Table 3. AdaBoost face recognition accuracy

Test dataset	True positives	False positives	False negatives	Recognition accuracy (%)
Dataset 1 (standard)	97	3	2	95.6
Dataset 2 (challenging)	92	8	5	89.6

The work compares the performance of a face recognition model on two datasets: a standard dataset and a challenging dataset, based on true positives, false positives, false negatives, and recognition accuracy. On dataset 1 (standard), the model performs exceptionally well, achieving a high recognition accuracy of 95.6%, with 97 true positives, 3 false positives, and only 2 false negatives. However, when tested on dataset 2 (challenging), the model's performance drops, with a lower accuracy of 89.6%, reflecting 92 true positives, 8 false positives, and 5 false negatives. The drop in accuracy and increase in false positives and negatives on the challenging dataset indicate that the model struggles more with complex or less ideal conditions, highlighting the importance of dataset variability for robust face recognition.

Recognition accuracy was calculated similarly to face detection accuracy. The combined face and eye detection system was evaluated for its ability to detect multiple features simultaneously. Table 4 shows the results.

Table 4. Face and eye detection performance

Feature	Detected faces	Detected eyes	Processing time per frame (ms)
Face detection only	90	-	30
Face and eyes detection	85	75	50

The work compares the performance of a system in two scenarios: face detection only and face and eyes detection, based on the number of detected faces, detected eyes, and processing time per frame. In the face detection only scenario, the system successfully detects 90 faces with a faster processing time of 30 milliseconds per frame. When both faces and eyes are detected, the number of detected faces slightly drops to 85, and the system detects 75 eyes, but this comes at the cost of increased processing time, which rises to 50 milliseconds per frame.

6. CONCLUSION

In this paper, we embarked on a journey to implement real-time face recognition using the OpenCV on a Raspberry Pi. Our aim was to develop an end-to-end system capable of detecting and recognizing faces in real-time, with a focus on computational efficiency and accuracy. We began by exploring the three fundamental phases of face recognition: face detection and data gathering, training the recognizer, and face recognition itself. Leveraging the power of the Haar Cascade classifier for face detection and the adaptable LBPH face recognizer for training and recognition, we devised a robust framework. By accurately following the step-by-step process, from installing OpenCV and testing the camera to gathering data and training the recognizer, we demonstrated the practicality of our approach. The integration of pre-existing knowledge, such as the Eigenface and Fisherface methods, enriched our understanding of the field and influenced our design choices. This paper's significance lies in its successful realization of a real-time face recognition system on a Raspberry Pi, making it accessible and feasible for a wide range of applications. As technology continues to advance, our paper contributes to the ongoing evolution of face recognition techniques, catering to the demands of accuracy, speed, and real-world implementation.

REFERENCES

[1] M. Turk and A. Pentland, "Eigenfaces for Recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, Jan. 1991, doi: 10.1162/jocn.1991.3.1.71.

[2] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: recognition using class specific linear projection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, Jul. 1997, doi: 10.1109/34.598228.

[3] T. Ahonen, A. Hadid, and M. Pietikäinen, "Face description with local binary patterns: Application to face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2037–2041, Dec. 2006, doi: 10.1109/TPAMI.2006.244.

[4] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2015, pp. 815–823, doi: 10.1109/CVPR.2015.7298682.

[5] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001, pp. 1–511, doi: 10.1109/cvpr.2001.990517.

[6] K. Al-Osaimi, R. Kallenberg, and J. Krijnders, "Fast and optimized face recognition on Raspberry Pi," in *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2018, pp. 440–447.

[7] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of the IEEE International Conference on Computer Vision*, IEEE, Dec. 2015, pp. 3730–3738, doi: 10.1109/ICCV.2015.425.

[8] L. Zhang, M. Yang, and X. Feng, "Sparse representation or collaborative representation: Which helps face recognition?," in *2011 International Conference on Computer Vision*, IEEE, Nov. 2011, pp. 471–478, doi: 10.1109/ICCV.2011.6126277.

[9] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong, "Locality-constrained linear coding for image classification," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2010, pp. 3360–3367, doi: 10.1109/CVPR.2010.5540018.

[10] Y. Sun, Y. Chen, X. Wang, and X. Tang, "Deep learning face representation by joint identification-verification," *Advances in Neural Information Processing Systems*, vol. 3, pp. 1988–1996, 2014.

[11] G. Ghiasi, T. Y. Lin, and Q. V. Le, "NAS-FPN: Learning scalable feature pyramid architecture for object detection," *Proceedings*

of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 7029–7038, 2019, doi: 10.1109/CVPR.2019.00720.

[12] S. Chen, Y. Liu, X. Gao, and Z. Han, "An anchor-free face detection algorithm via center-based representation," *arXiv*, 2019.

[13] X. Zhang, J. Liu, and H. Li, "A high-performance face recognition architecture for embedded systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 2, pp. 482–496, 2019.

[14] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1701–1708, doi: 10.1109/CVPR.2014.220.

[15] R. Wang, S. Shan, X. Chen, and W. Gao, "Manifold-manifold distance with application to face recognition based on image set," *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 1449–1456, 2008, doi: 10.1109/CVPR.2008.4587719.

[16] G. Ghiasi, T. Y. Lin, and Q. V. Le, "NAS-FPN: Learning scalable feature pyramid architecture for object detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7029–7038, doi: 10.1109/CVPR.2019.00720.

[17] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, pp. 734–750, doi: 10.1007/978-3-030-01264-9_45.

[18] Y. Zhang, H. Wang, and Z. Zhang, "Efficient Neural Architecture Search with Network Morphism," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 7329–7337.

[19] P. Vidyasree, G. Madhavi, S. Viswanadharaju, and S. Borra, "A bio-application for accident victim identification using biometrics," *Lecture Notes in Computational Vision and Biomechanics*, vol. 26, pp. 407–447, 2018, doi: 10.1007/978-3-319-65981-7_15.

[20] K. Nakajima, V. Moshnyaga, and K. Hashimoto, "A comparative study of conventional and CNN-based implementations of facial recognition on Raspberry-Pi," in *SAMI 2021 - IEEE 19th World Symposium on Applied Machine Intelligence and Informatics, Proceedings*, IEEE, Jan. 2021, pp. 217–221, doi: 10.1109/SAMI50585.2021.9378635.

[21] S. Phawinee, J. F. Cai, Z. Y. Guo, H. Z. Zheng, and G. C. Chen, "Face recognition in an intelligent door lock with ResNet model based on deep learning," *Journal of Intelligent and Fuzzy Systems*, vol. 40, no. 4, pp. 8021–8031, Apr. 2021, doi: 10.3233/JIFS-189624.

[22] N. G. Rani, N. H. Priya, A. Ahilan, and N. Muthukumaran, "LV-YOLO: logistic vehicle speed detection and counting using deep learning based YOLO network," *Signal, Image and Video Processing*, vol. 18, no. 10, pp. 7419–7429, 2024, doi: 10.1007/s11760-024-03404-w.

[23] S. Ambre, M. Masurekar, and S. Gaikwad, "Face recognition using raspberry pi," in *Modern Approaches in Machine Learning and Cognitive Science: A Walkthrough: Latest Trends in AI*, vol. 885, 2020, pp. 1–11, doi: 10.1007/978-3-030-38445-6_1.

BIOGRAPHIES OF AUTHORS

Dr. Bharanidharan Chandrakasan working as an associate professor in the Department of Artificial Intelligence and Data Science at Panimalar Engineering College and He completed under graduate from Adhiparasakthi Engineering College. He completed post graduate from Annamalai University and he completed research from Anna University and area of specialization is deep learning and computer network. He can be contacted at email: bharanidharan.pec@gmail.com.

Dr. D. Karunkuzhali is currently working as Professor in the Department of Information Technology, Panimalar Engineering College, Chennai, India. She received her Ph.D. (Computer Science and Engineering) in 2014 and M.Tech., (Information Technology) in 2008 from Sathyabama University, Chennai, India and pursed her B.Tech., (Computer Science and Engineering) from Pondicherry University, Pondicherry, India in 2003. She has published articles in 21 international journals and 2 national journals. She has presented papers in more than 25 international and national conferences. She is having 10 patents and 2 copyrights. She has published technical books and contributed various chapter writing on artificial intelligence and cyber security. Her work being profiled broadly in wireless communications, network security, wireless sensor networks, and IoT. Her research interest includes microprocessors, compiler design, IoT, AI, machine learning, and data science. She is a receiver of Best faculty award and young researcher award and life time member of CSI, IETE, ISTE, IAENG, IACSIT, SDIWC and member of IEEE. She can be contacted at email: karunkuzhali@gmail.com.

V. Kandasamy received the B.E. degree from Periyar University, Salem, and the M.E. degree from Annamalai University, Chidambaram. He is currently pursuing the Ph.D. degree (part-time) with Anna University. He is an Assistant Professor with Panimalar Engineering College, Chennai, Tamil Nadu. He can be contacted at email: mail4kands@gmail.com.

Dr. M. Dilli Babu currently working as an Associate Professor in the Department of Information Technology at Panimalar Engineering College Chennai. His research area of interest includes data analytics, cloud computing, intelligent system, and IoT. He can be contacted at email: ptrjohncenas@gmail.com.

Dr. K. Rama Devi is working as an Associate Professor in Department of Information Technology. She has academic experience of more than 23 years for both UG and PG courses. She has done her Ph.D. from Computer Science and Engineering in SRM University, Chennai in the year 2018. Her research interest includes cryptography, network security, and artificial intelligence. She is passionate, self driven academician, researcher and author who have innovative ideas in the field of higher education and research. She has been chaired as a convenor in organizing Workshops, Student's Induction Programs, International and National Conferences related to computer science. She has published more than fifteen papers in peer-reviewed journals and conferences proceedings. She has guided several Masters, and Undergraduate students in their academic projects. She can be contacted at email: ramadevi.sarav@gmail.com.