FPT UNIVERSITY HCMC

Face recognition and video tracking

Computer Vision - SU24

Vu Hoang Minh Khanh— SE183169

Le Tu Quoc Huy— SE181552

Vu Ngoc Thien An— SE181651

1 Introduction

Facial recognition and video tracking are pivotal technologies for security, surveillance, and user authentication on devices like smartphones and laptops. These technologies enhance security and offer personalized user experiences by accurately detecting and recognizing faces in real-time. Facial recognition, an intelligent biometric application, identifies individuals through tracking and detection, making it especially useful in crowded areas such as airports, train stations, universities, and shopping centers, ensuring safety and security.

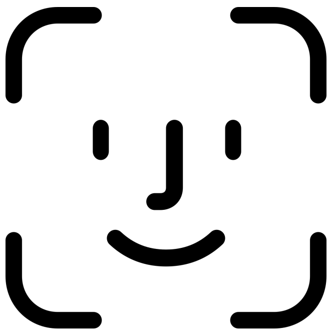


Figure 1: Face id

With applications ranging from national security to social media platforms like Facebook and Instagram, facial recognition helps users tag and manage images automatically and efficiently. The benefits of facial recognition technology include higher accuracy, increased security levels, easy integration with existing security systems, and support for businesses to enhance their authentication processes. The widespread use and flexibility of facial recognition and video tracking are transforming our approach to security and convenience, opening up new potentials and opportunities for future technological development.

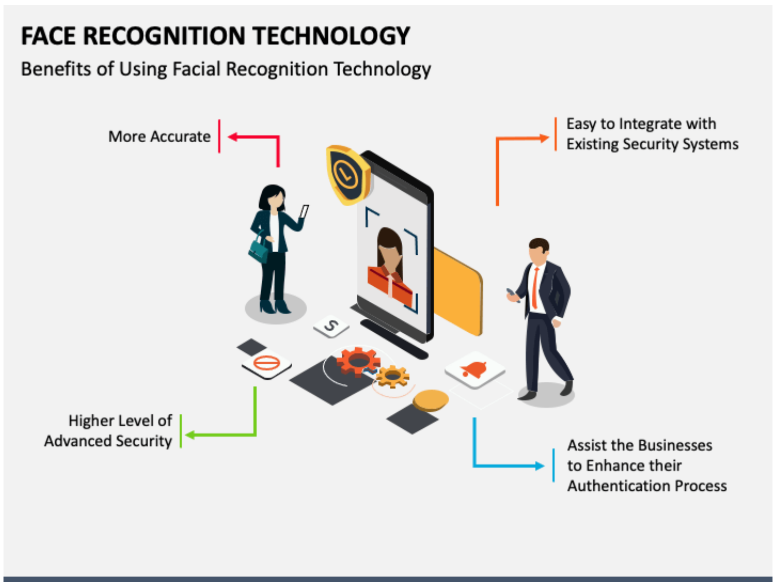

FACE RECOGNITION TECHNOLOGY
Benefits of Using Facial Recognition Technology
More Accurate
Easy to Integrate with Existing Security Systems
Higher Level of Advanced Security
Assist the Businesses to Enhance their Authentication Process

Figure 2: Benifits of face recognition

2 Problem Definition

1. **Problem Statement**

    - In the field of image recognition and classification, face detection is a crucial and complex task that demands rapid and accurate processing to ensure the practicality and effectiveness of the system.
    - In real-world applications, the ability to detect faces plays a vital role in security systems, surveillance, and user identification applications. This requires a software solution capable of processing multiple images and videos.
    - Key challenges in face detection include ensuring accuracy and sensitivity of the system, along with rapid and efficient processing to meet real-time requirements in practical applications.

2. **Project Scope**

    The main goal of this project is to explore and implement face detection and recognition algorithms using Haar Cascade and Local Binary Patterns Histogram (LBPH). The objective is to develop a robust face recognition system that can accurately identify individuals based on their facial features in case the faces are not obscured. The system will include capabilities for capturing facial images, training the recognition model, and recognizing faces in real-time. This project aims to achieve the following:

    - Face Detection: Using Haar Cascade algorithms to detect faces in images.
    - Face Recognition: Implementing LBPH algorithms to recognize and identify individuals.
    - Real-time Processing: Ensuring the system can capture and process facial images in real-time.
    - User Interface: Displaying the recognized faces with overlaid names and confidence scores in a real-time video stream with the interface created by Tkinter.

3. **Problem Challengers**

    This project develops a reliable face recognition system that operates in real-time to meet the need for rapid and accurate individual identity verification. The system addresses the following challenges:

    - Achieving high-accuracy face recognition.
    - Processing and recognizing faces in real-time to meet performance requirements.
    - Providing a user-friendly interface to visually display recognition results.

3 Method

1. **Haar Cascade Algorithm**

   The Haar Cascade algorithm is a machine learning technique used for object detection, particularly face detection, developed in 2001 by Paul Viola. This algorithm is trained with positive images (containing faces) and negative images (not containing faces) to identify Haar-like features. By subtracting the sum of pixel values under black rectangular regions from the sum of pixel values under white rectangular regions, Haar Cascade creates powerful features for object recognition. To optimize, an integral image is used to speed up the computation.

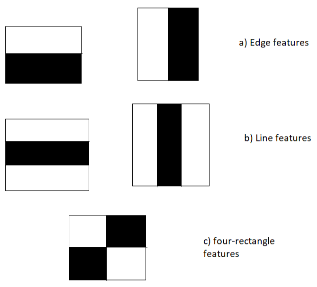


Figure 3: Block diagram

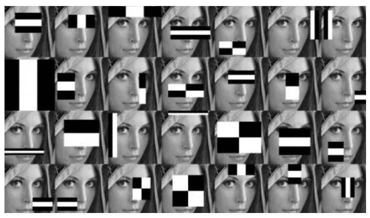


Figure 4: Haar like features

   Haar features are efficiently calculated by referencing sub-rectangles, simplifying the processing. The algorithm also employs the Adaboost classifier to combine 'weak classifiers' into a 'strong classifier' by selecting features and training the classifier. This process involves sliding a window across the image, calculating Haar features, and comparing them with a threshold to distinguish objects. The cascading classifiers are designed to quickly eliminate non-object samples, focusing only on what matters. Each stage consists of weak learners, also known as weights, trained by Boosting to minimize the false negative rate, creating a highly accurate classifier and ensuring the object detection system operates more efficiently and reliably.

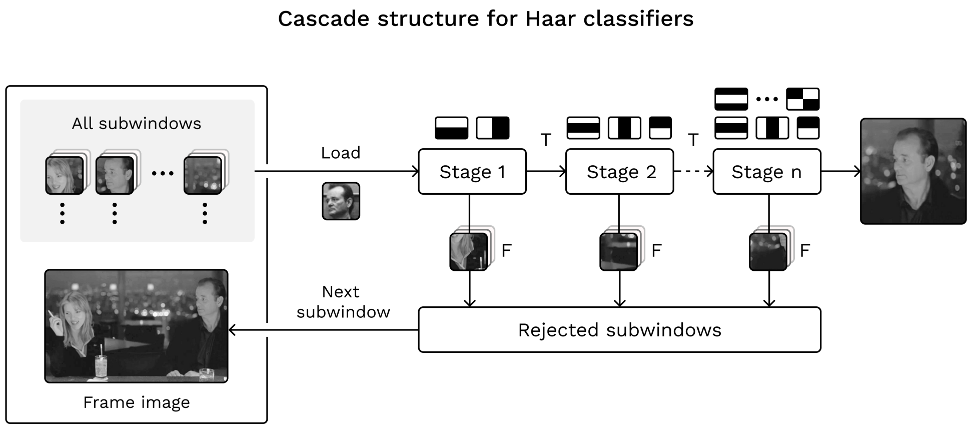


Figure 5: Haar cascade work chart

Weakness: Haar Cascade relies on Haar features, which are pairs of black and white rectangles to detect intensity changes in a region of the image. When lighting changes, the difference between light and dark areas can decrease or change, making Haar features harder to accurately identify.

2. **Local Binary Pattern Histogram (LBPH) Algorithm**

   Local Binary Pattern Histogram (LBPH) is a powerful algorithm for face recognition, developed in 1996, and has proven effective in texture classification. LBPH labels the pixels of an image by thresholding the neighborhood of each pixel and converting the result into binary numbers. When these Local Binary Patterns (LBP) are combined with Histogram of Oriented Gradients (HOG), detection performance improves significantly. LBPH works by dividing the face image into several blocks, computing the histogram for each block, and comparing each pixel with the central pixel to create binary numbers, which are then converted into decimal values. These values form a feature vector representing the face, aiding in accurate recognition against a database.

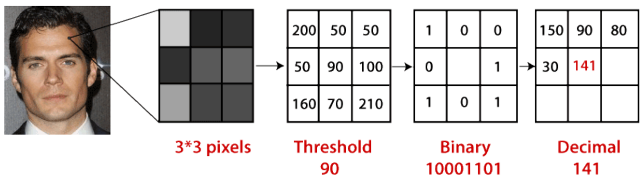


Figure 6: LBP operation

The LBPH algorithm operates with four main parameters: radius, number of neighboring points, grid X, and grid Y. The radius, usually set to 1, is the distance from the central pixel to the perimeter. Typically, the number of neighboring points, set to 8, represents the number of points in a circular LBP. Grid X and Grid Y, both set to 8, determine the number of cells horizontally and vertically. Images with unique identifiers are used to create histograms representing the frequency of LBP codes. Classification involves comparing these histograms to find similarities.

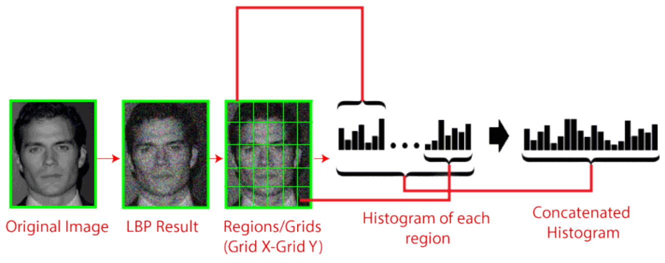


Figure 7: LBP operation

For face recognition, the process includes adding a new image to the program, creating an LBP histogram for the image, comparing this histogram with those in the database, and determining the closest match. The algorithm uses the

Euclidean distance method to calculate the distance which called confidence. So we can understand that the smaller confidence, the higher the accuracy.

$$D = \sqrt{\sum_{i=0}^{n} (hist1_i - hist2_i)^2}$$

Weakness: LBPH works by analyzing local binary patterns around each pixel. When lighting changes, the intensity values of the surrounding pixels also change, altering the binary patterns and affecting the final result.

4 Implementation and Results

1. **Work chart**

   Steps to address the problem include:

   - Loading Haar Cascade models for face detection and creating a camera object for real-time image capture.
   - Capturing a new image and using the Haar Cascade algorithm to detect faces. If no faces are detected, returning to the image capture step.
   - Once a face is detected, cropping the image to isolate the facial region.
   - Initializing and configuring the LBPH face recognizer.
   - Reading a list of names from a text file and mapping them to recognized face IDs.
   - Passing the cropped face image through the LBPH recognizer to predict the individual's identity.
   - Overlaying the predicted name and confidence score on the image along with a bounding box around the detected face.
   - Finally, displaying the annotated image in real-time to ensure quick system responsiveness.

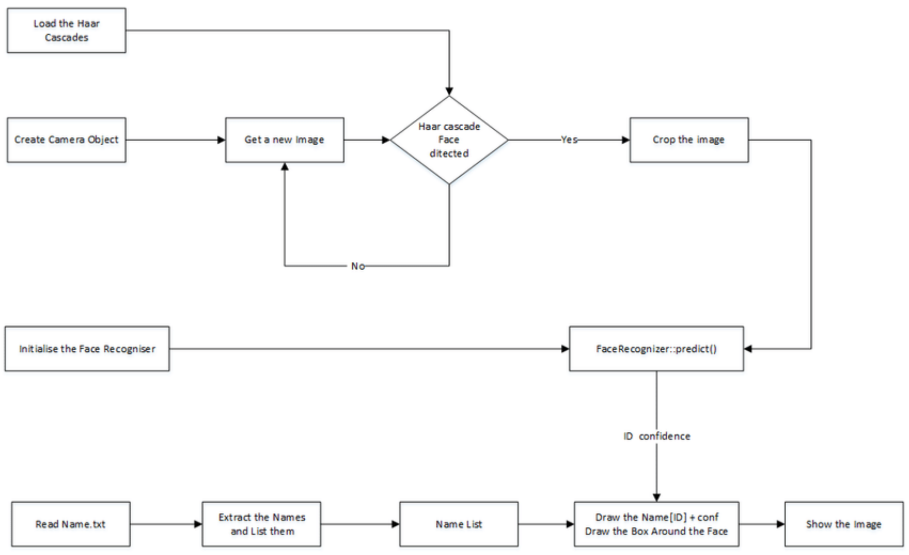

Load the Haar Cascades
Create Camera Object
Get a new Image
Haar cascade Face dictected
Yes
Crop the image
No
Initialise the Face Recogniser
FaceRecognizer::predict()
ID confidence
Read Name.txt
Extract the Names and List them
Name List
Draw the Name[ID] + conf Draw the Box Around the Face
Show the Image

Figure 8: Problem work chart

2. **Implementation**

   The implementation includes creating a dataset of face images for training and testing. The Haar Cascade algorithm is used to detect faces in video frames, while the LBPH algorithm is used to recognize the detected faces. The system is trained with multiple images of individuals, each labeled with a unique identifier.

   Below is the source code using the OpenCV and Tkinter libraries for face data collection, training, and face recognition:

   (a) Install and import necessary libraries for the project.

```
!pip install tkinter as tk
!pip install opencv-python
!pip install opencv-contrib-python
!pip install Pillow
!pip install numpy
```

```
import tkinter as tk
from tkinter import simpledialog, messagebox
import cv2
import os
from PIL import Image, ImageTk
import numpy as np
import json
```

   (b) Create a custom dialog in a Tkinter application to create a user interface with labels and entry widgets. It prompts the user to enter an ID and a Name.

```
# Create a custom dialog for user input
class CustomDialog(simpledialog.Dialog):
    def body(self, master):
        # Creating labels and entry widgets for ID and Name
        tk.Label(master, text='ID:').grid(row=0)
        tk.Label(master, text='Name:').grid(row=1)

        self.id_entry = tk.Entry(master)
        self.name_entry = tk.Entry(master)

        self.id_entry.grid(row=0, column=1)
        self.name_entry.grid(row=1, column=1)
        return self.id_entry

    def apply(self):
        # Retrieve the user input when the dialog is confirmed
        self.result = (self.id_entry.get(), self.name_entry.get())
```

   (c) Initialize two global lists `id_list` and `name_list` to store names and IDs of users. Then define `save_data` and `load_data` for saving these lists to a JSON file and loading them back, respectively. The `save_data` function writes the lists to a file named `data.json` and the `load_data` function reads the data from this file if it exists, ensuring persistence of user data across sessions.

```
# Initialize ID and name lists
id_list = []
name_list = []
```

```python

# Save lists to JSON file
def save_data():
    with open('data.json', 'w') as file:
        json.dump({'id_list': id_list, 'name_list': name_list},
    file)

# Load the data JSON file
def load_data():
    global id_list, name_list
    if os.path.exists('data.json'):
        with open('data.json', 'r') as file:
            data = json.load(file)
            id_list = data.get('id_list', [])
            name_list = data.get('name_list', [])
```

We will put step (d) and (e) into one function is `collect_and_train()` according to our gui's suitability.
Before step (d), we will create a folder name `collect_data` to save the images.
And we need to download the `haarcascade_frontalface_default.xml`-a haar cascade designed by OpenCV to detect the frontal face- from here

(d) Collect face's images starts by invoking the custom dialog to get the user ID and Name. It then validates the input, ensuring that both fields are filled and that the ID is unique. If valid, the user information is appended to the lists, which are then saved. Then, we will track the video using OpenCV and uses a Haar Cascade classifier to detect faces with the suitable parameters as: gray frames, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30). Then we captures 500 images of the user's face, saving them to `collect_data` with names defined by `user_id`, index of `user_id` and number of captures.

```python
# Show custom dialog to get user ID and Name
input_data = CustomDialog(root)
if input_data.result is None:
    return
user_id, user_name = input_data.result

# Validate the input data
if not user_id or not user_name:
    tk.messagebox.showerror('Error', 'ID and Name must not be empty
    .')
    return

if user_id in id_list:
    tk.messagebox.showerror('Error', 'ID already exists.')
    return

# Append the ID and Name to the respective lists
id_list.append(user_id)
name_list.append(user_name)

save_data()

notice_label.config(text='Waitting to get face information...')
root.update()

# Initialize video capture and face detector
```

```python
video = cv2.VideoCapture(0)
facedetect = cv2.CascadeClassifier('haarcascade_frontalface_default
    .xml')

count = 0

while True:
    ret, frame = video.read()
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = facedetect.detectMultiScale(gray_frame, scaleFactor
    =1.1, minNeighbors=5, minSize=(30, 30))

    for (x, y, w, h) in faces:
        count += 1
        face = gray_frame[y:y+h, x:x+w]
        # Save the captured face images to the directory
        cv2.imwrite(f'collect_data/{user_id}.{id_list.index(user_id
    )}.{count}.jpg', face)
        cv2.rectangle(frame, (x, y), (x+w, y+h), (50, 50, 255), 2)

    cv2.imshow('Collecting Face Data', frame)
    if cv2.waitKey(1) & 0xFF == ord('q') or count >= 500:
        break

video.release()
cv2.destroyAllWindows()
```

(e) Training model, when the data collection is complete, we will convert the images into arrays and extract the index of the `user_id` embedded in the images to train a face recognizer using LBPH (Local Binary Patterns Histograms) and save the trained model into a file named `Trainer.yml`.

```python
# Initialize the face recognizer and prepare for training
recognizer = cv2.face.LBPHFaceRecognizer_create()
path = 'collect_data'

def get_images_and_ids(path):
    # Function to retrieve images and their corresponding IDs
    image_paths = [os.path.join(path, i) for i in os.listdir(path)]
    faces = []
    ids = []
    for image in image_paths:
        face_image = Image.open(image).convert('L')
        face_arr = np.array(face_image, 'uint8')
        id = int(os.path.split(image)[-1].split('.')[1])
        faces.append(face_arr)
        ids.append(id)
    return ids, faces

# Train the recognizer with the collected face data
ids, faces = get_images_and_ids(path)
recognizer.train(faces, np.array(ids))
recognizer.write('Trainer.yml')
notice_label.config(text='')
root.update()
messagebox.showinfo('Info', 'Collecting data complete!')
```

(f) `recognize_faces` function handles real-time face recognition. It sets up video

capture and loads the pre-trained LBPH face recognizer. For each frame captured from the webcam, it converts the frame to grayscale and detects faces using the Haar Cascade classifier. The recognizer predicts the ID of each detected face, and if the confidence is below a threshold=30, it displays the predicted name and confidence score on the video feed. If the confidence is above the threshold, it labels the face as 'Unknown'.

```python
# Function to recognize faces
def recognize_faces():
    video = cv2.VideoCapture(0)
    face_detect = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    recognizer = cv2.face.LBPHFaceRecognizer_create()
    recognizer.read('Trainer.yml')

    while True:
        ret, frame = video.read()
        if not ret:
            break
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_detect.detectMultiScale(gray_frame, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
        for (x, y, w, h) in faces:
            face = gray_frame[y:y+h, x:x+w]
            id, conf = recognizer.predict(face)
            # Choose threshold is 30 for higher accuracy in recognize
            if conf < 30:
                name = name_list[id]
                confidence = round(conf, 2)
                color = (50, 255, 50)
            else:
                name = 'Unknown'
                confidence = round(conf, 2)
                color = (50, 50, 255)

            cv2.putText(frame, f'{name}  {confidence}', (x+5, y-5), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
            cv2.rectangle(frame, (x, y), (x+w, y+h), color, 2)

        cv2.imshow('Frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    video.release()
    cv2.destroyAllWindows()
```

(g) The gui function sets up the main application window using Tkinter. It configures the window with a background image(you can download here) and creates buttons for collecting data and recognizing faces, each linked to their respective functions (`collect_and_train` and `recognize_faces`. It also includes a label for displaying notices to the user. The `load_data` function is called to initialize the ID and name lists when the application starts. Finally, the Tkinter main loop is started to keep the GUI active.

```python
# GUI function
def gui():
    global root, notice_label

    # Main application window
    root = tk.Tk()
    root.title('Face Recognition')

    # Set background image
    background_image = Image.open('background_image.jpg')
    background_photo = ImageTk.PhotoImage(background_image)

    canvas = tk.Canvas(root, width=background_image.width, height=background_image.height)
    canvas.pack(fill='both', expand=True)
    canvas.create_image(0, 0, image=background_photo, anchor='nw')

    canvas.create_text(600, 50, text='Face recognition', font=('Arial', 40), fill='white')

    # Create buttons for collecting data and recognizing faces
    button1 = tk.Button(root, text='Input your ID and name', command=collect_and_train)
    button2 = tk.Button(root, text='Recognize Faces', command=recognize_faces)

    canvas.create_window(600, 150, window=button1)
    canvas.create_window(600, 200, window=button2)

    notice_label = tk.Label(root, text='', fg='red', bg='white')
    canvas.create_window(600, 250, window=notice_label)

    # Load the lists when the application starts
    load_data()

    root.mainloop()

# Run the GUI
if __name__ == '__main__':
    gui()
```

Additionally, from this source code, we can develop a simple attendance system model. You can refer to it through the following GitHub link: https://github.com/KhanhVHM1/Face-recognition-project

3. **Experiment Results**
This code combines multiple libraries to create a face recognition model with fairly good accuracy. Additionally, with Tkinter, it provides a relatively user-friendly interface for collecting user data, capturing face images, training a recognition model, and recognizing faces in real-time. The use of JSON ensures that user data is preserved across sessions, allowing for reuse without the need for repeated data collection and training.

However, the main weakness of the two algorithms mentioned above—sensitivity to lighting conditions—is also a significant weakness of the model. The accuracy of the model decreases when used in different lighting environments. Specifically, even when an additional light is placed near the user, the accuracy can decrease(the confidence increase as figure 11 and 12). To overcome this, it is necessary to train the model with a dataset collected under various lighting conditions.

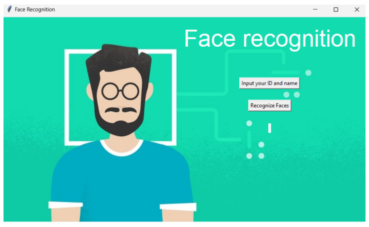
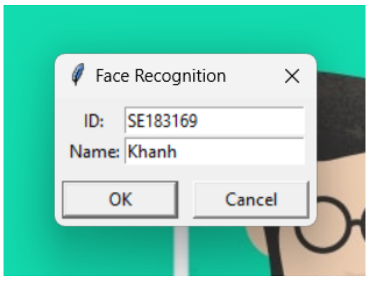


Figure 9: Display interface


Face Recognition

ID: SE183169
Name: Khanh

Figure 10: Display input data interface

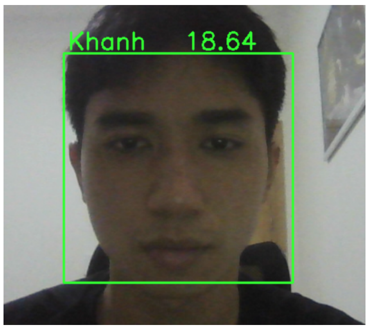


Figure 11: Recognize demo 1

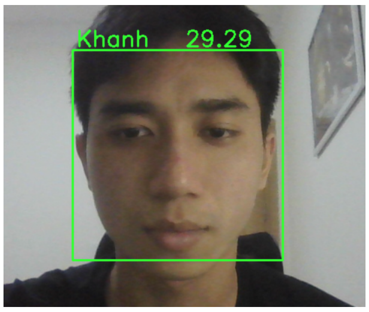


Figure 12: Recognize demo 2

Overall, this model has successfully implemented a face recognition and video tracking system using LBPH and Haar Cascade classifiers. Although improvements are needed to enhance accuracy under varying conditions, the results demonstrate the effectiveness of the algorithms used in controlled environments. Additionally, more advanced feature extraction methods, such as deep learning-based techniques, can be employed to improve performance and accuracy in diverse environments.

References

1. Rapid Object Detection using a Boosted Cascade of Simple Features 2001
2. Haar Cascade Algorithm And Local Binary Pattern Histogram LBPH Algorithm In Face Recognition, Vol 3, no 4, pp 2395-2398, April 2022
3. Comparative Study of LBPH and Haar features in Real Time Recognition Under Varying Light Intensities, Volume-9 Issue-4S, May 2020
4. Real Time Face Recognition in Group Images using LBPH, Volume-8 Issue-2, July 2019
5. Haar Cascade Algorithm - Javatpoint
6. Haar-like-feature - Wiki
7. Local binary patterns - Wiki
8. Local binary patterns - Wiki