Facial Recognition Surveillance

Jason Serrano,
Dept. of Computer Science and Engineering
Santa Clara University

Farhaan Pishori,
Dept. of Computer Science and Engineering
Santa Clara University

Hugo Muro Avila,
Dept. Computer Science and Engineering
Santa Clara University

***Abstract* — This paper presents a comprehensive overview of the development and implementation of a facial recognition system using Python and OpenCV. The system captures real-time video feed from a webcam, processes the frames to detect and recognize faces, and provides immediate feedback on the identified individuals. The implementation is divided into two main components: encoding known faces and performing real-time recognition. The encoding phase involves preprocessing images of known individuals to generate facial encodings, which are then stored for later use. In the recognition phase, the system captures live video frames, detects faces, and matches them against the stored encodings. This paper covers the software and hardware components used, the methodology of the implementation, and the results obtained from the system. Additionally, the paper discusses the challenges faced during development, the performance of the system, and potential future improvements for the complete product. The results confirm the system's ability to accurately identify known individuals and send alerts in case of intruders, making it a viable solution for security and identification applications.**

***Keywords* - Facial Recognition, Raspberry Pi, Surveillance, OpenCv, Machine Learning**

1. Introduction

The United States experiences on average 3,062 burglaries each day [1]. 72% of these grievances occur when the resident isn't home. Alongside this, families, who don't own any security system, are 300% more likely to experience a home invasion. That being said, only 25% of households in the US have adopted a home security system. Our goal for designing a surveillance system with facial recognition is to help decrease the number of home burglaries and to add extra safety and reassurance for the homeowner who currently possess home security systems.

Through this paper, we will talk about the vision we have intended for our IoT device, with all the possible use cases. We will also be going into detail about the software and hardware components of our device.

Vision

Our Internet of Things Device will acts as a security camera already widely used in Smart Home security systems. The camera will scoop the household for potential threats or burglars. The camera will constantly monitor throughout the day. Once a human is detected through Artificial Intelligence incorporated with the technology, the camera will scan the person's face. If the person's face is not registered in the house databases as a resident or friend, an alert will be sent to the primary user of the cameras, ideally the head of the household. In the message, a snapshot of the stream will be sent to the user. Here, the head of the household can decide, in the app, whether the stranger is a friend or foe. If they choose"friend", the alert will be dropped, and no further action will occur. If the foe option is chosen, the app will automatically call the police. After this, the rest of the work will be on the user to communicate with law enforcement dispatch about a burglar attempting to break into their residence.

Present Scope

Currently, our Facial Recognition Technology project is able to display the stream on a separate webpage. Here the stream will display if the person is the resident or a stranger. It is being powered by Farhaan's computer through using a virtual machine. The virtual machine uses the Raspberry Pi OS, Rasbian, to simulate the actual hardware environment.

It is through here that our program is running, and with our Logitech camera we are able to make our facial recognition surveillance work. As it stands, the webpage will say the recognized person's name if they are detected on the camera and comparable to a current person in a database. If the person is not in the facial recognitions' database, the webpage will return an intruder alert.

This paper details the hardware and software components used in the system, the design and implementation process, and the results of our performance evaluation. We demonstrate the system's capability to accurately identify known individuals and alert users in real-time when an unknown face is detected."

General Overview:

Existing Systems*:* Currently, facial recognition systems are employed in various domains such as mobile phone security, public surveillance, and law enforcement. These systems typically rely on high-end hardware and proprietary software, making them expensive and less accessible for general home use.

Technology Overview: Facial recognition technology involves several steps, including image acquisition, face detection, feature extraction, and recognition. Commonly used algorithms include Haar Cascades for face detection and deep learning models for feature extraction.

Comparison: The Raspberry Pi 3 offers an affordable and versatile platform for developing facial recognition systems, especially when combined with open-source libraries such as OpenCV and dlib. This approach is cost-effective compared to commercial surveillance solutions.

2. System Design and Architecture

Software Components:

- **Python**: Python was chosen as the primary programming language for this project due to its simplicity, readability, and libraries and frameworks for both Raspberry Pi and facial recognition models. The codebase for this project is structured into two main Python scripts: encode_faces.py for encoding known faces and recognize_faces.py for real-time recognition.
- **OpenCV (Open Source Computer Vision Library)**: OpenCV is a library for computer vision and image processing. It provides a comprehensive set of tools for a variety of computer vision tasks. In this project, we utilize OpenCV to capture real-time video feed from the webcam and display the video frames with annotations regarding the results of the scan.
- **face_recognition**: The face_recognition library, built on top of the dlib library, offers face detection and recognition capabilities. It provides simple and intuitive functions for loading images, detecting faces, and generating facial encodings. The library's high accuracy and ease of use make it a popular choice for facial recognition projects. In this project, face_recognition is used for both the encoding and recognition phases.
- **pickle**: The pickle module in Python is used for serializing and deserializing Python objects. In this project, pickle is used to save the known face encodings and names to a file, which can be loaded during the recognition phase. This ensures that the system can quickly access the preprocessed face data without having to reprocess the images each time.

Hardware Components:

- **Webcam**: A USB webcam is used to capture real-time video feed. The camera provides the frames for the facial recognition process. The quality and resolution of the webcam can impact the accuracy and performance of the system, so a high-quality webcam is recommended for best results.
- **Virtual Machine**: The project runs on a virtual machine set up with a Raspberry Pi OS, in order to simulate the environment of a physical Raspberry Pi. The computer must have sufficient processing power and memory to handle real-time video processing and facial recognition tasks.

3. Implementation

- **Setup**: Setting up the Raspberry Pi 3 involves installing the Raspbian OS and configuring the camera module. The OpenCV and face recognition libraries are installed via pip. The camera is then added via USB port to the computer running the program, which is then accessed by the virtual machine.
- **Facial Recognition Algorithm**: The facial recognition process starts with capturing an image using the webcam. The OpenCV library is used for face detection, and the face recognition library extracts facial features. These features are compared against a pre-registered database of known faces, which are analyzed using the library.

Encoding Faces

The following script, named encode_faces.py is responsible for encoding the faces of known individuals. This preprocessing step involves loading images of known individuals, detecting faces in the images, and generating facial encodings. The data, along with the corresponding names are then saved to a file using the pickle module. During the recognition phase, this file can be quickly accessed to compare facial encodings.

```
python
```

```
Copy code
import face_recognition
import os
import pickle


KNOWN_FACES_DIR = "known_faces"
TOLERANCE = 0.6
FRAME_THICKNESS = 3
FONT_THICKNESS = 2
MODEL = "cnn"   # Can be "hog" or "cnn"


print("Loading known faces...")
known_faces = []
known_names = []


for name in os.listdir(KNOWN_FACES_DIR):
    for filename in os.listdir(f"{KNOWN_FACES_DIR}/{name}"):
        image = face_recognition.load_image_file(f"{KNOWN_FACES_DIR}/{name}/{filename}")
        encodings = face_recognition.face_encodings(image)


        if len(encodings) > 0:
            encoding = encodings[0]

            known_faces.append(encoding)
            known_names.append(name)
        else:
            print(f"No faces found in {filename}")


with open("known_faces.dat", "wb") as f:
    pickle.dump((known_faces, known_names), f)

print("Encoding complete!")
```

- **Loading Known Faces**: The script begins by initializing empty lists for storing face encodings (known_faces) and names (known_names). It then iterates through each directory in the known_faces folder, where each directory corresponds to an individual's name. For each image file in these directories, the script loads the image and generates face encodings using the face_recognition library. If a face is detected in the image, the encoding is added to the known_faces list, and the corresponding name is added to the known_names list. If no faces are found, the script prints a message indicating the file with no faces detected.
- Saving Encoded Faces: After processing all images, the script uses the pickle module to save the known_faces and known_names lists to a file named known_faces.dat. This file is used during the later recognition phase to quickly load the preprocessed face data. Once this process is complete, it prints a confirmation message.

Real-Time Recognition

The next script, named recognize_faces.py, captures real-time video feed from the webcam, detects faces within each frame, and compares them against the stored face encodings. The system provides real-time feedback on recognized individuals and sends alerts if an intruder is detected.

```
python
Copy code
import face_recognition
import cv2
import pickle
from server import app, run_server, status
import threading


KNOWN_FACES_DIR = "known_faces"
TOLERANCE = 0.6
FRAME_THICKNESS = 3
FONT_THICKNESS = 2
MODEL = "hog"
```

```
with open("known_faces.dat", "rb") as f:
    known_faces, known_names = pickle.load(f)

video_capture = cv2.VideoCapture(0)
if not video_capture.isOpened():
    print("Cannot open camera")
    exit()

def update_status(new_status):
    global status
    status = new_status
    print(f"{status}")

def face_recognition_loop():
    while True:
    ret, frame = video_capture.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break

    locations = face_recognition.face_locations(frame, model=MODEL)
    encodings = face_recognition.face_encodings(frame, locations)

    if len(encodings) == 0:
        update_status("No face detected")
    else:
        intruder_alert = True
        for face_encoding, face_location in zip(encodings, locations):
            results = face_recognition.compare_faces(known_faces, face_encoding, TOLERANCE)
            match = None
            if True in results:
                match = known_names[results.index(True)]
                intruder_alert = False

update_status(f"Match Found: {match}")

                top_left = (face_location[3], face_location[0])
                bottom_right = (face_location[1], face_location[2])
                color = [0, 255, 0]

cv2.rectangle(frame, top_left, bottom_right, color, FRAME_THICKNESS)

                top_left = (face_location[3], face_location[2])
                bottom_right = (face_location[1], face_location[2] + 22)

cv2.rectangle(frame, top_left, bottom_right, color, cv2.FILLED)
                cv2.putText(frame, match, (face_location[3] + 10, face_location[2] + 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (200, 200, 200), FONT_THICKNESS)

            if intruder_alert:
            update_status("INTRUDER ALERT")
            print("Intruder Alert")

    cv2.imshow("Video", frame)
```

```
        if cv2.waitKey(1) & 0xFF ==
ord('q'):
            break


        video_capture.release()
        cv2.destroyAllWindows()


if __name__ == "__main__":

threading.Thread(target=run_server).
start()
        face_recognition_loop()
```

- **Loading Known Faces**: The script begins by loading the preprocessed face encodings and names from the known_faces.dat file using the pickle module.
- **Video Capture Initialization**: The script initializes video capture from the default webcam (index 0). If the camera cannot be opened, the script prints an error message and exits.
- **Face Recognition Loop**: The main loop captures video frames continuously from the webcam. For each frame, the script detects faces and generates face encodings. It then compares these encodings against the known face encodings to determine if any matches are found. If the match is found, the script updates the status with the name of the recognized individual and draws a green rectangle around the face with the name displayed above it. If no match is found, the script updates the status to "INTRUDER ALERT". The video frames with annotations are displayed in a window, and the loop continues until the 'q' key is pressed.

4. Testing and Results

- **Testing Procedure**: Testing involved adding an image of one of our team members into the directory to be processed by encode_faces.py. Once the script ran and the facial encoding was stored, the second phase of this project began. We utilize the camera's video frames and process them through the facial recognition script that will then compare them to the existing database. We then observe the results with our current knowledge of the facial encodings included in the database. *Figure 1* showcases the system accurately recognizing a face and comparing it to an existing encoding.

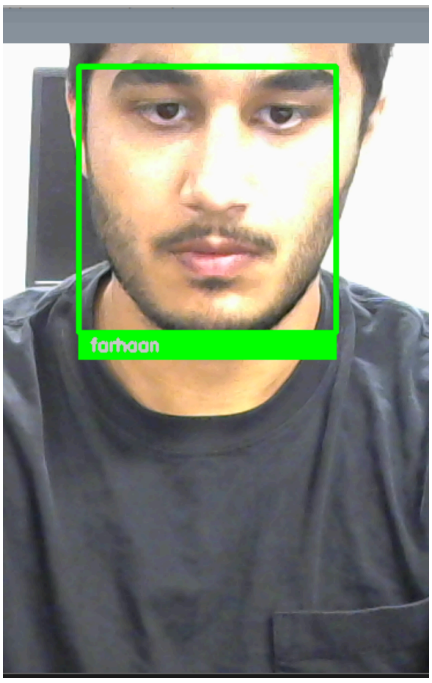

Figure 1: Recognized Face

- **Performance Metrics**: The system was tested mutiple times, with each test being able to see the differences between familiar and unfamiliar faces. The program was able to accurately compare facial encodings and display the appropriate name, or print out "INTRUDER ALERT" in cases where the face is unfamiliar.

5. Discussion

Challenges:

- **Raspberry Pi**: A challenge faced during the development of this project was the incapability of testing this on an actual Raspberry Pi due to a variety of circumstances. This obstacle was overcome by using a virtual machine, in conjunction with RasbianOS, to simulate the hardware environment of a Raspberry Pi and complete the current iteration of the project.

*Potential Future Improvements***:**

- **Web Application**: There are a number of different improvements that could be made in the future. Firstly, the development of a web app that can be used to stream the results of the facial recognition portion of the project. This web app would let users

access the feed and notifications from beyond the Raspberry Pi's native hardware.

- **Power Considerations:** While the current setup of the project works well within our conditions, care must be taken in building the project for a real world scenario. There are both energy and processing power concerns in a typical home security system, as the camera must be high quality enough to get accurate results, while also being energy efficient. In addition, current home security devices are not built in order to accommodate a full Raspberry Pi 3 board, and will likely need something smaller yet powerful enough to process video frames and run OpenCV's facial recognition libraries.
- **Script Optimization:** While more than enough for the current iteration of the project, future improvements could move away from the pickle module in search of more optimized methods of encoding and storing facial encodings.

6. Conclusion

This paper presented the development and implementation of a facial recognition system using Python and OpenCV. The system captures real-time video feed from a webcam, detects faces, and recognizes known individuals, providing immediate feedback. The project involved two main components: encoding known faces and performing real-time recognition. The results indicate that the system is reliable and effective for real-time identification and security applications.

The project demonstrated the capabilities of modern computer vision and machine learning techniques in creating a robust facial recognition system. The use of Python libraries such as OpenCV and face_recognition provided an effective framework for implementing the system. Despite the challenges faced during development, the system achieved high accuracy and real-time performance.

Future work could include optimizing the encoding and recognition processes, and integrating additional functionalities such as a web application service. The web application can host the live video stream and act as a facial recognition security bot that can be accessed from anywhere. This project provides a solid foundation for further development in the field of facial recognition.

References

[1] L. Pelchen, "Surprising Home Burglary Facts and Stats," Forbes, Apr. 30, 2024. https://www.forbes.com/home-improvement/home-security/home-invasion-statistics/ (accessed Aug. 24, 6AD).

[2] A. Rosebrock, "OpenCV face recognition," PyImageSearch, https://pyimagesearch.com/2018/09/24/opencv-face-recognition/ (accessed Jun. 11, 2024).

Github Repository: https://github.com/Farhaanp9/IoT_project