Project Report

June 04, 2024

Course: **AI Lab**

Project topic: **Face Recognition System**

Worked by: Goidan Matei
Udrea Ionut Alexandru
Favia Abazi
Giulia Stefania Imbrea

Content

1. Introduction

Face Recognition is a technology that enables the identification or verification of individuals based on their facial features. It has gained significant attention due to its applications in security, surveillance, access control, and personal device authentication. This technology involves various processes, from capturing an image of a face to comparing it against a database of known faces.

History:

The development of face recognition technology can be traced back to the 1960s when the first semi-automated system was developed. This early system required the administrator to locate features (such as eyes, ears, nose, and mouth) on photographs before it computed distances and ratios to a common reference point.

As an homage to traditional methods, we have created our model to identify faces using a person's eyes.

Applications:

Some examples of **Face Recognition** are:

- Apple's 2020 Face ID Security: Facial recognition in personal device authentication, ensuring secure access to iPhones and iPads by using sophisticated neural networks to map and recognize users' faces.
- CNN report, face recognition technology: Diagnose rare genetic disorders by analyzing facial features, thus providing critical support in medical settings where early and accurate diagnosis can vastly improve patient outcomes.
- Google's photos: Advancements in photo organization leverage face recognition to automatically tag and organize photos, making it easier for users to manage their digital memories.

These examples underscore the diverse and impactful applications of face recognition technology, showcasing its ability to enhance security, convenience, and healthcare outcomes.

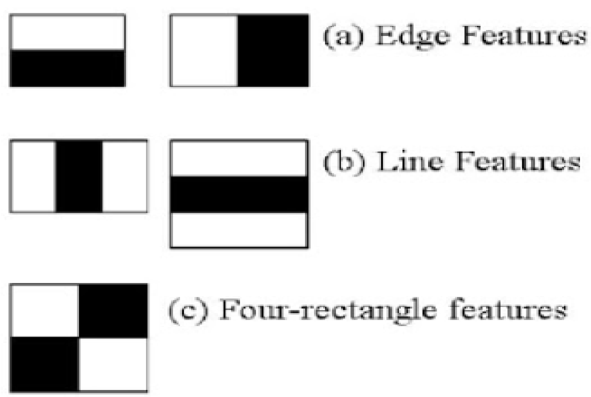
2. Methodology

Our project aims **to develop a comprehensive face recognition system that can collect images using a camera interface, train a recognition model, and accurately identify faces.** The key components of our project are:

A. Face Detection

In order to capture a face we decided to use the **"Haar Cascade classifier"** for our project since it's one of the most common use classifiers.

This classifier was made by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive images (images with faces) and negative images (images without faces). It is then used to detect objects in other images.



To maximize feature extraction, various sizes and locations are considered. However, amidst this abundance, many are deemed irrelevant and computationally intensive. Fortunately, during the process of scaling an image, emphasis is placed on key facial features such as the eyes and nose, as the darkness of the eye region compared to the more light nose and cheeks region. Nonetheless, applying these features indiscriminately to irrelevant areas, like the cheeks, yields no useful information. To sift through the extensive array of features, a machine learning algorithm called Adaboost, designed for binary classification tasks—distinguishing between positive and negative instances—is employed.

image

Adaboost applies each feature to all training images, identifying the best threshold to classify faces as positive or negative. Features with the lowest error rates are selected, meaning they best distinguish between face and non-face images. Initially, all images have equal weight, but weights of misclassified images are increased after each iteration. This process continues until the desired accuracy or number of features is achieved.

The final classifier is a weighted sum of these weak classifiers, which alone can't classify images but form a strong classifier together. Even 200 features can achieve 95% accuracy, with the final setup using around 6,000 features, a significant reduction from the initial 160,000+.

B. Data Gathering

The images for training were gathered in two ways: by capturing photos via a webcam and uploading files.

Initially, when executing the program the user is prompted to enter a numeric face ID to label the captured images (his\her name). Then the program enters a loop where it continuously captures 30 frames from the webcam that we set with a resolution of 640x480, converts them to grayscale, and detects faces in these frames using the classifier.

Each label is then stored in the dataset directory, having an individual repository with each image captured.

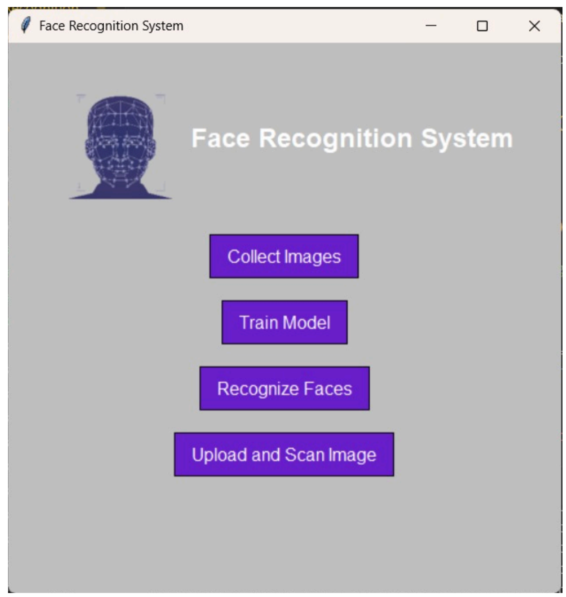
C. Model Training and Face Recognizer

We use the LBPH (Local Binary Patterns Histograms Stogra) Face Recognizer, included in the OpenCV package. This takes all images from the label directory, creating 2 arrays: “Ids” and “faces”. As a result, a file named “trainer.yml” will be saved in the trainer directory as a result of training our recognizer.

For face recognition we use the laptops camera. With it we will cathc the face of the person that had his face captured and trained before. The recognizer will make a “prediction” returning its id and an index (for the label array), showing how confident the recognizer is with this match.

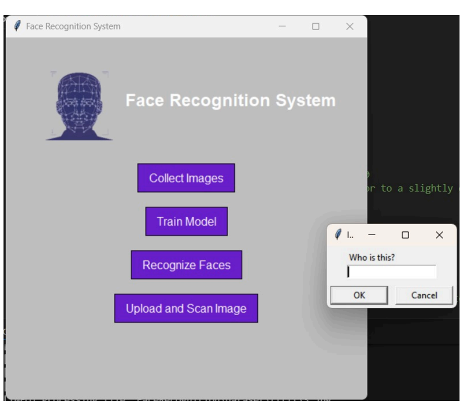
D. User Interface

To make our program more friendly to the user we implemented the GUI which is built upon the Tkinter library. Tkinter offers a simple yet powerful toolkit for creating visually appealing interfaces, facilitating seamless interaction with the system's functionalities. Leveraging Tkinter's capabilities, the system's interface is structured to enhance user experience.



Moreover, we employed threading from the Python threading module to manage concurrent tasks within the system. By utilizing multi-threading, our program optimizes performance by executing tasks concurrently, thereby preventing UI freezes and enhancing responsiveness.

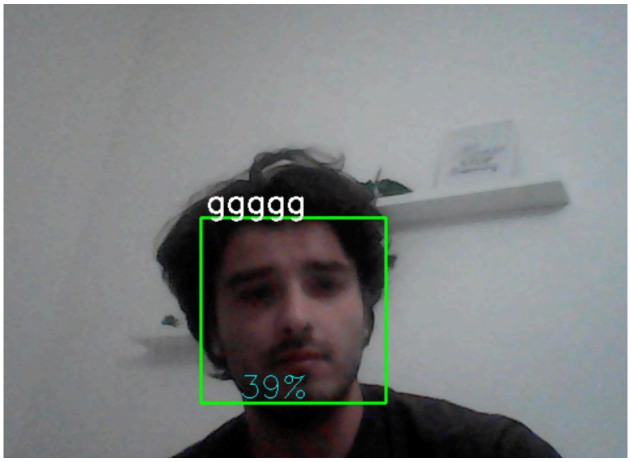
3. Results



The UI is organized in buttons. Firstly, the user has to click the Collect Images button, and a pop-up window shows up for the user to input his ID and then the camera collects the image frames.

The Train Model button can be used after the image collection is done, the user just needs to click the button.

When the user wants to use the Face Recognition feature he can click the Recognize Face button. The camera will open and the face will be detected and put in a blue rectangle, in the meantime, the user ID and the percentage of face compatibility are shown.



Lastly, the user can use this program to detect faces in images. After clicking the button, a dialogue window appears for the user to upload an image from his device. Then a pop-up window shows up with the number of faces detected and another one displays the detected faces.

4. Problems Encountered

In the end we also tried to detect faces from images but unfortunately we didn't have enough time to finish it and implement it in time.

5. References

[1] *OpenCV: Face Detection using Haar Cascades*. (2017, August 4). Retrieved June 3, 2024, from https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html

[2] *Rovai, M. (2019, September 20). Real-Time Face Recognition: an End-To-End Project - towards Data science. Medium. Retrieved May 20, 2024, from* https://towardsdatascience.com/real-time-face-recognition-an-end-to-end-project-b738bb0f7348

[3] *exter minar. (2021, April 7). 26 face and eye detection [Video]. YouTube. Retrieved May 20, 2024, from* https://www.youtube.com/watch?v=deMJacPb0gQ

[4] *SuperDataScience (2017, August 03) Face recognition using OpenCV and Python: A beginner's guide, from https://www.superdatascience.com/blogs/opencv-face-recognition*