2024

Analyzation of the morphological features of the face

STRUCTURE OF COMPUTER SYSTEMS

GEORGE-DORIAN DAMIAN

TABLE OF CONTENTS

1 INTRODUCTION

1.1 CONTEXT

The aim of this project is creating a system that can detect and analyze the facial features of a person. The analyzation of those features can determine a person's identity, and therefore allowing them access to confidential data or higher security zones.

Detecting facial key points is a very challenging problem. Facial features differ greatly between individuals, but even considering one person, factors like 3D pose, size, viewing angle, and lighting conditions can cause significant variation. While computer vision research has come a long way, there are still many opportunities for improvement.

1.2 OBJECTIVES

The project will be developed using Python programming language. The project also requires OpenCV (Open Source Computer Vision Library), which is a open source computer vision and machine learning software library. The system we design should be able to detect the features of someone's face (eyes, mouth, nose etc.) by receiving a file input (image or video) or by taking real-time video using a webcam. The detected features should be contoured on the input received.

2 BIBLIOGRAPHIC RESEARCH

2.1 WHAT IS FACIAL RECOGNITION?

Facial Recognition is a way of identifying an individual's identity using their face. Facial Recognition systems can be used to recognize people in photos, videos or in real-time.

Facial Recognition is a category of biometric security, along with other forms of recognition like Voice Recognition, Fingerprint Recognition or Iris Recognition.

The most common Facial Recognition Systems are used for securing smartphones. The more advanced systems are used in apartment building or businesses in order to identify a person by unique physiological features.

2.2 HOW TO DETECT FACIAL FEATURES?

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, „Rapid Object Detection using a Boosted Cascade of Simple Features" (2001). It is a machine learning-based approach where a cascade function is trained with a high volume of positive and negative images.

This approach works by using Haar-like features, which are simple patterns of adjacent rectangular regions that capture differences in pixel intensity, like edges, lines, or other structures in an image. The algorithm scans an image at multiple scales and positions to detect features like eyes, nose or mouth.

This method is fast and effective for real-time detection but can be less accurate in complex scenarios compared to more modern techniques.

3 ANALYSIS

3.1 PROJECT PROPOSAL

This project proposes a real-time facial feature recognition system that can detect faces and their key features in video feeds and static images. Using Python and OpenCV, the system aims to demonstrate a modular approach to face detection. By capturing facial features, the project serves as a building block and start-up for various applications, including security, emotion detection, and human-computer interaction.

3.2 PROJECT ANALYSIS

3.2.1 Face Detection with Haar Cascades

Haar Cascades are a machine learning-based method for real-time object detection. It relies on "Haar-like features", which detect simple patterns, like edges or textures, by analyzing contrasts in image regions. Haar Cascades operate by sliding a window across the image, detecting regions that match pre-trained feature patterns.

By adjusting parameters such as *scaleFactor* and *minNeighbors,* the system can optimize detection speed and accuracy, balancing resource efficiency with real-time performance requirements.

Although fast and suitable for low-power devices, Haar cascades are less accurate than modern deep learning techniques, especially in complex backgrounds or under varying lightning.

3.2.2 Detection of Facial Features in Real-Time

To enhance detection accuracy and efficiency, the system performs the feature recognition only within the detected face region, narrowing down the area where it looks for eyes and the mouth. This approach reduces computation and increases precision, allowing for smoother real-time processing. These improvements make the system effective for real-time applications, where rapid and accurate face recognition is essential.

4 DESIGN

4.1 FACE DETECTION DESIGN

4.1.1 Face Boundary Detection

The face boundary is detected first. It is later used as a reference for detecting the features like eyes and mouth. Using the pre-trained Haar Cascade classifier, the system is able to identify the rectangular regions that match the general structure of a human face. The detected face is then marked with a bounding box that will serve as the guide for the feature detection steps. This design choice improves performance by focusing detection on the face area, rather than the entire image.

4.1.2 Eye and Mouth Detection within Face Region

After detecting the face, the system isolates this area and searches for the eyes and the mouth within it, therefore reducing the processing required for the detection. The detection mechanism is the same as the step before, the only difference being that the system uses Haar Cascade classifiers trained for detecting eyes and mouths. After the system calculates the coordinates for those features, it draws one rectangle for each eye and one rectangle for the mouth. This is the way the system visualizes the detection, providing a very intuitive output for the user.

4.2 DETECTION OPTIMIZATION

4.2.1 Scaling Factor

This parameter determines how much the image size is reduced at each image scale during the detection process. The detection algorithm starts with the original image and keeps scaling it down by the factor specified.

A lower Scaling Factor results in more precise detections, but more time-consuming searches, while a higher Scaling Factor results in less accuracy but faster detections.

4.2.2 Minimum Neighbors

The Minimum Neighbors parameter specifies how many neighbors each detected rectangle (potential object region) should have to retain for it to be a valid detection.

When a feature is detected in an image, the algorithm considers nearby overlapping detections as "neighbors". A detection is confirmed only if it has at least the number of neighbors specified in the parameter.

A lower threshold of Minimum Neighbors allows more detections, including some possible false positives. A higher threshold makes the system more selective, reducing false positives but potentially missing real features.

5 IMPLEMENTATION

5.1 SYSTEM SETUP

5.1.1 Loading Haar Cascades for Faces, Eyes and Mouth

Firstly, the system loads the pre-trained Haar Cascades classifiers for the face and features, which are stored in XML files. These classifiers allow the program to detect these facial features accurately without requiring custom training.

5.1.2 Setting up Input Mode

Based on the number of arguments provided, the system can either start detection on a provided image, either initiate a webcam feed and provide real-time feature detection.

For the detection based on an image, the system reads the path provided by the user, signaling if the image cannot be found. If the path leads to a valid image, the program loads it and then waits for the user to press any key, signaling the program to close the window and stop.

In case that the user wants real-time detection, the OpenCV VideoCapture function initiates the webcam feed. Each frame of the feed is processed to detect faces and features. The user can stop the feed anytime by pressing the 'q' key.

5.2 CORE DETECTION FUNCTIONS

5.2.1 *draw_boundaries* Function

This function is responsible for drawing a rectangular boundary for each set of coordinates, representing detected instances. This provides the functionality of marking multiple faces.

5.2.2 *detect* Function for Face, Eyes and Mouth

This function performs the main detection task, calling *draw_boundaries* for each feature and ensuring that each detected face has its own annotated boundaries. It firstly detects the face and then scans the region for eyes and mouth, creating a modular pipeline that is easy to extend.

6 TESTING AND VALIDATION

The testing phase involves:

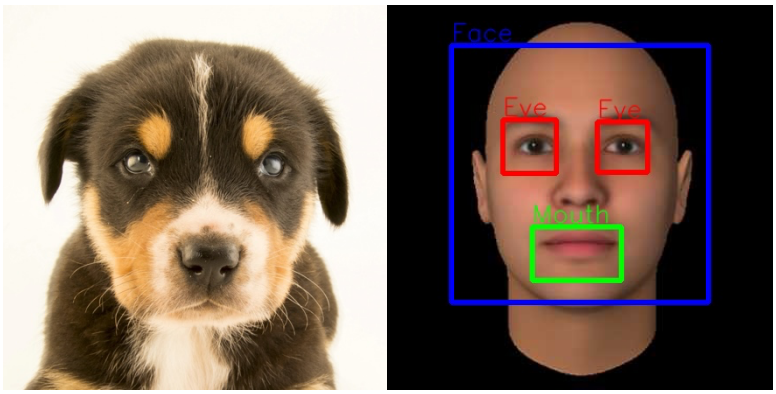
- giving the program multiple images and checking if the boundaries colored by the script are correct.
- using the real-time feature detection in multiple condition with different light levels, faces and distances.

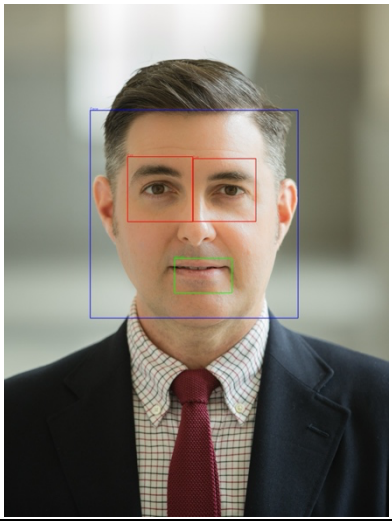
6.1 PROVIDING IMAGES AS INPUT

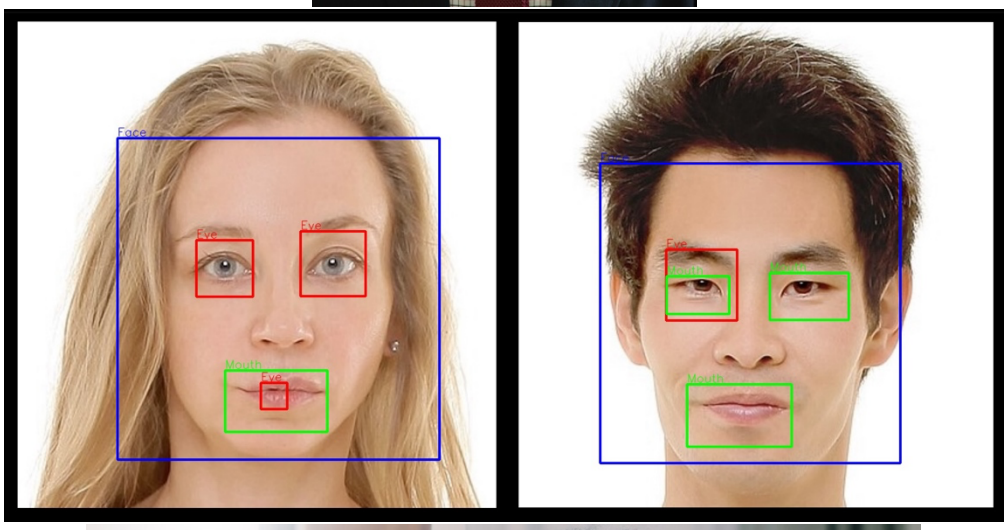
To test the program accuracy, a set of images will be provided as input, and then the correctness of the algorithm will be decided based on the outputs given.

For a simpler testing experience, the user can simply provide the argument '*-test'* followed by a directory path and the program will search for each image inside that directory. The program will create another directory named *'output'* and place all the output images inside that new directory.

These are the outputs given based on a input set:







Based on the script's output, we can see that it mostly is correct. Multiple faces inside the same image might bring difficulties.

A larger set of images will need to be provided in order to test the full capabilities of this script, but based on these 4 images we can make some assumptions:

- A picture with a higher number of people (e.g. group photo) would result in a smaller efficiency and therefore some false positives.
- Faces that are at an angle would be harder to detect.
- Facial expressions would make it more difficult for the program to correctly detect faces.
- Bad lighting is also a factor in bad detection.

6.2 REAL-TIME CAMERA FEED AS INPUT

Real-time detection is more challenging because of the shorter computing time and usually lower resolution. Mostly, if the user is not too far away from the camera, facing straight and in good lighting, the algorithm will detect the features correctly, without errors. If not, the script will have a more difficult time detecting facial features, and more errors will be made.

7 CONCLUSIONS

The project successfully demonstrates a real-time facial feature detection system using Haar Cascades, implemented using Python and OpenCV. By using a modular approach, it identifies facial boundaries and key features like eyes and mouth in both images and real-time video feeds. While the system performs well under favorable conditions, challenges such as angled faces and poor lightning affect its accuracy. These limitations highlight opportunities for future improvements. Overall, the project serves as a basic and fundamental framework for applications in security and human-computer interaction.