PROJECT

Digital Image Processing

Iris detection, iris visual features and comparison

MADE BY:

Olgica Upcheva 211130

Content

1. Introduction

This documentation provides a comprehensive guide for setting up, understanding, and customizing an Iris Detection, Visual Features, and Comparison Demo App. The app allows users to detect irises, extract visual features, and compare them for similarity. Target audience:

- Developers interested in biometrics.
- Biometric security enthusiasts.
- Students and researchers exploring iris recognition.

Biometric technology deals with recognizing the identity of individuals based on their unique physical or behavioral characteristics. Physical characteristics such as fingerprint, palm print, hand geometry and iris patterns or behavioral characteristics suck as typing pattern and handwritten signature present unique information about a person and can be used in authentication applications.

Biometric identification technology is based on personal biometrics, and identity authentication through pattern recognition is accurate and convenient, which has become a hot research direction today. Among them, the emerging iris-based recognition technology has broad application prospects.

Therefore, compared with other branches in biometrics, iris recognition is more accurate and difficult to forge due to its stable and unique characteristics. Despite the development of iris recognition over the years, it not only has significant progress in imaging devices, but also been fully developed in the recognition algorithm, but its application value remains to be discovered.

The detection of the iris image is the detection of the inner and outer edges of the iris. According to the gray shape characteristics of the iris image, the edge detection of the iris is performed by Hough transform combined with Canny edge detection. The motivation behind this project lies in the ever-growing demand for robust and secure biometric identification systems. As traditional methods often fall short in terms of accuracy and susceptibility to fraud, iris recognition emerges as a forefront solution due to its non-intrusiveness, high uniqueness, and resistance to tampering.

The significance of this project lies in its potential applications across various domains, including access control, identity verification, and forensic investigations. The primary objectives of this project are twofold: first, to develop a sophisticated iris detection system capable of accurately locating and isolating the iris region within an image, and second, to employ advanced visual features and comparison methods for precise iris recognition. The robust iris detection, coupled with advanced visual features and customizable comparison methods, positions this system as a versatile and secure solution in the realm of biometric identification.

2. Overview

The code for this project loads the image with iris and creates a descriptor for the specific image, firstly the program is showing us the uploaded images. After that it tries to select some iris visual features such as color histogram values, for both of the images and print it to the output. With the following lines the program is comparing the two found irises using Euclidean distance as a similarity metric, and for this situation we are setting a threshold to determine whether the irises are similar or not. If the distance is below the threshold the program return us a positive reply.

3. Used libraries

The code uses the Python programming language along with two main libraries for computer vision and numerical operations: OpenCV and NumPy. We are using this specific language because we need to do some file operations and manipulations that this language makes them easier to use. Also we are using it for the ability of scripting and control flow.

- **OpenCV (Open Source Computer Vision Library):** This is a powerful library designed for computer vision tasks, image processing, and machine learning. We are using the following functions:

➔ cv2.imread: Reads an image from a file.
➔ cv2.cvtColor: Converts the image from one color space to another.
➔ cv2.GaussianBlur: Applies a Gaussian blur to the image to reduce noise.
➔ cv2.threshold: Applies thresholding to segment the image.
➔ cv2.findContours: Finds contours in the thresholded image.
➔ cv2.boundingRect: Computes the bounding box of the largest contour.
➔ cv2.resize: Resizes the iris region of interest (ROI).
➔ cv2.calcHist: Calculates the color histogram of the entire image.
➔ cv2.imshow, cv2.waitKey, cv2.destroyAllWindows: Displays the cropped iris region.

- **NumPy:** Purpose**:** This is a fundamental package for scientific computing with Python, providing support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays. In this program we are using:

➔ np.concatenate: Combines iris texture and color histogram features into a single descriptor.

➔ np.linalg.norm: Calculates the Euclidean distance between two descriptors.

- **Image Processing Techniques:**

➔ **Gaussian Blur:** Applied to reduce noise in the image and improve contour detection.
➔ **Thresholding:** Used to segment the image and create a binary mask.
➔ **Contour Detection:** Used to find the boundaries of objects in the image.
➔ **Bounding Box:** Determines the rectangular region around the largest contour, assumed to be the iris.
➔ **Histogram Calculation:** Computes a color histogram to represent the color distribution of the image.

The code showcases a simple iris comparison system using a combination of iris texture (resized iris region) and color histogram features. The similarity between irises is determined by calculating the Euclidean distance between their descriptors and comparing it to a predefined threshold.

These technologies and techniques make the code suitable for iris recognition and similarity comparison, which can find applications in biometrics, security systems, and healthcare.

4. Implementation

Function ‘extract_descriptor’ – this function is designed to process an input image containing an iris, extracting relevant features, and generating a descriptor for iris recognition. The key steps involve image preprocessing, contour detection, region of interest (ROI) extraction, resizing, color histogram calculation, and descriptor creation. This function also encapsulates the essential steps for iris feature extraction, providing a structured and modular approach for iris recognition. The resulting descriptor integrates both texture and color information for enhanced accuracy in iris comparison. We only got one parameter: **image_path (str)** - the file path to the input image containing an iris. Steps:

Load the input image using OpenCV.

Convert the loaded image to grayscale for simplicity and reduced computational complexity.

Apply Gaussian blur to the grayscale image to reduce noise.

Create a binary mask using thresholding, enhancing contour detection.

Identify contours in the binary mask.

Select the largest contour based on its area.

Determine the bounding box around the largest contour.

Extract the region of interest (ROI) around the iris using the bounding box coordinates.

Resize the iris ROI to a fixed size (128x64 pixels) for consistency in feature representation.

Calculate the color histogram of the entire image using 8 bins for each channel.

Print a summarized version of the color histogram information, showing the summed values for each channel.

Combine the flattened resized iris and color histogram arrays to form a comprehensive descriptor.

Show the cropped iris region for visual inspection.

If contours are found, return the generated descriptor.

If no contours are detected, print a message and return **None**.

Function 'compare_irises' – this function serves the purpose of comparing two iris descriptors using Euclidean distance as a similarity metric. The function allows users to set a threshold for determining whether the irises are considered similar or dissimilar. function provides a straightforward method for comparing iris descriptors. By utilizing Euclidean distance as a similarity measure and allowing users to set a threshold, the function enables a flexible approach to iris similarity determination in the context of iris recognition. We've got two parameters: **descriptor1 (numpy.ndarray)** - the descriptor for the first iris, and **descriptor2 (numpy.ndarray)** - the descriptor for the second iris. Steps:

Compute the Euclidean distance between the two input descriptors. The Euclidean distance is a measure of the straight-line distance between two points in a multi-dimensional space.

Set a predefined threshold value (similarity_threshold) to determine whether the irises are similar or dissimilar.

Return True if the calculated Euclidean distance is below the threshold, indicating similarity.

Return False otherwise.

Main – this code segment demonstrates the practical application of iris recognition. It loads two iris images, extracts their descriptors using the extract_descriptor function, compares the descriptors using the compare_irises function, and finally prints a message indicating the similarity of the irises. The modular design of the code facilitates easy integration and understanding of the iris recognition process. Steps:

Specify the file path for the first iris image (iris-image2.jpg)

Use the extract_descriptor function to process the first iris image and obtain its descriptor (descriptor1).

Specify the file path for the second iris image (iris-image3.jpg).

Use the extract_descriptor function to process the second iris image and obtain its descriptor (descriptor2).

Use the compare_irises function to compare the descriptors of the two irises (descriptor1 and descriptor2).

The result (are_similar) is a boolean indicating whether the irises are considered similar (True) or not (False).

Print a message indicating whether the irises are similar or not based on the result of the iris comparison.

5.Organizational structure

With this organizational structure, the project is well-organized and maintainable. It also allows for easy integration with version control systems like Git and collaboration with other developers. Additionally, you can expand the project by adding more modules or directories as needed while keeping everything neatly organized. The code doesn't take up a lot of space, so I gathered up in one file, this way it's easier to use it and test it.

1. projectDirectory/

    1.1 images/ - this directory is used to store input images required for iris descriptor extraction and comparasion. You can add more images here as needed.

    1.2 src/ - this is the source code directory

        1.2.1 main.py – this is the main script to execute iris descriptor extraction and comparasion process. It may contain the logic for loading images, calling functions and handling the comparasion results.

6. Usage and features

Usage:

Iris recognition is widely used in biometric identification systems. It provides a highly accurate and secure method for authenticating individuals based on the unique patterns present in their iris. Commonly employed in access control systems, iris recognition ensures secure entry to restricted areas or systems, minimizing the risk of unauthorized access. Iris recognition is utilized for identity verification in various applications, such as unlocking smartphones, accessing personal devices, and authenticating online accounts. In forensic science, iris

recognition can play a crucial role in identifying individuals based on images captured at crime scenes or from surveillance footage. Iris recognition can be employed in healthcare for patient identification, ensuring accurate medical records and preventing identity theft or medical fraud.

Features:

Robustness - The algorithm demonstrates robustness by effectively handling variations in iris patterns caused by factors such as lighting conditions, image quality, and occlusions.

Multi – modal features - The algorithm combines both texture and color features for a more comprehensive iris descriptor. This multi-modal approach enhances the overall accuracy of iris recognition.

Contour detection - Utilizing contour detection ensures precise identification of the iris region, allowing for accurate feature extraction and reducing the impact of noise.

Color histogram analysis - Incorporating color histogram analysis provides a representation of the iris's color distribution, contributing additional discriminative features to the recognition process.

Euclidean distance metric - The use of Euclidean distance as a similarity metric allows for efficient comparison of iris descriptors, providing a quantifiable measure of similarity or dissimilarity.

Thresholding mechanism - The inclusion of a thresholding mechanism in the comparison process offers flexibility in determining the acceptable level of similarity between irises, allowing customization based on specific application requirements.

Resize for consistency - Resizing the iris region to a fixed size ensures consistency in feature representation, making the algorithm more robust to variations in image resolutions.

User – friendly output - The algorithm provides a clear output message, indicating whether the compared irises are considered similar or not, facilitating user interpretation.

The implemented iris recognition algorithm showcases a versatile and reliable solution for various applications requiring biometric identification. Its combination of robust features, multi-modal descriptors, and customizable similarity metrics makes it suitable for use in diverse scenarios, ranging from security systems to healthcare and forensic investigations. The algorithm's modular design allows for easy integration and adaptation to specific use cases, contributing to its practicality and effectiveness.

7. Customization and algorithm integration

The iris recognition system is versatile and can be customized to suit specific requirements. Customizations can range from incorporating alternative feature extraction methods to

integrating different similarity metrics. Here are some customization options for alternative feature extraction techniques, advanced matching algorithms and deep learning architectures:

SIFT (Scale-Invariant Feature Transform) - consider integrating SIFT, a robust feature extraction technique, into the system. SIFT features are invariant to scale and rotation, potentially enhancing the system's ability to handle variations in iris images.

SURF (Speeded-Up Robust Features) - the integration of SURF, which offers similar advantages to SIFT but with improved computational efficiency. SURF features can provide stable and distinctive representations of iris patterns.

Hamming Distance - instead of Euclidean distance, experiment with the Hamming distance for iris comparison. This can be particularly useful when working with binary descriptors, offering a different approach to measuring similarity.

Mahalanobis Distance - integrate the Mahalanobis distance, which considers the correlation between feature dimensions. This can be beneficial for capturing complex relationships within iris descriptors.

Convolutional Neural Networks (CNNs) - explore the implementation of CNNs for end-to-end learning of iris features. CNNs have demonstrated remarkable success in image recognition tasks and may uncover intricate patterns in iris images.

Siamese Networks - Consider Siamese networks for learning similarity between pairs of iris images. Siamese networks are designed to handle one-shot learning scenarios, potentially improving performance with limited labeled data.

These customization options open avenues for tailoring the iris recognition system to specific needs, whether it be for increased accuracy, enhanced security, or adaptability to different environments. The selection of customization options should align with the project's goals and the unique requirements of the application.

8. Future development

The current iris recognition code provides a solid foundation for biometric identification, but there are several avenues for future development and enhancement to further improve its capabilities and usability. Explore the integration of deep learning techniques, such as Convolutional Neural Networks (CNNs), for feature extraction and iris recognition.

Deep learning models have demonstrated success in learning intricate patterns from images, potentially enhancing the accuracy and robustness of the system. Expand the diversity of the dataset used for training and testing the algorithm. Augment the dataset with variations in

lighting conditions, occlusions, and image resolutions to improve the algorithm's ability to handle real-world scenarios. Optimize the code for real-time processing, enabling the system to perform iris recognition on live video streams or camera feeds. This enhancement would be valuable for applications requiring instantaneous authentication, such as access control systems. Experiment with different parameter values, such as threshold levels and blur kernel sizes, to fine-tune the algorithm for better performance under varying conditions. Implementing an adaptive thresholding mechanism could improve the algorithm's adaptability to different datasets.

Create a user-friendly interface to allow users to interact with the iris recognition system easily. The interface could include options for enrolling new iris images, managing user profiles, and displaying recognition results in a clear and comprehensible manner. Implement additional security measures, such as encryption of stored iris descriptors and the incorporation of anti-spoofing techniques, to enhance the overall security of the system and prevent potential vulnerabilities.

Establish a continuous evaluation and testing framework to assess the algorithm's performance over time. Regularly update the dataset, retrain the model, and conduct thorough testing to identify and address potential issues. The outlined future developments aim to elevate the iris recognition system to new heights by incorporating advanced techniques, improving adaptability, and enhancing user experience. By embracing these enhancements, the system can stay at the forefront of biometric identification technology and meet the evolving needs of various applications.

9. Potential problems

Threshold Selection: The threshold value used in the cv2.threshold() function might not be optimal for all images, potentially leading to poor segmentation results.

Assumption of Largest Contour: Assuming the largest contour represents the iris might not hold true in all cases, especially if there are multiple objects in the image or if the iris is partially obscured.

Color Histogram Features: The calculation of color histogram features assumes that the iris region is sufficiently colored to provide meaningful information. This may not always be the case, especially in images with poor lighting or low contrast.
Euclidean Distance Threshold: The similarity threshold used for comparing iris descriptors

(similarity_threshold) is arbitrary and might not accurately represent similarity across different datasets. Fine-tuning this threshold may be necessary for specific applications.
Error Handling: The script lacks comprehensive error handling mechanisms. For instance, it does not handle scenarios where the input image paths are invalid or where the iris extraction process fails due to image quality issues.

10. References

Daughman Rubber Sheet Model. | PDF | Image Segmentation | Biometrics

iris-daugman-rubber-sheet/iris_rubber_sheet.py at main · anitoanto/iris-daugman-rubber-sheet

28,100+ Iris Eye Stock Photos, Pictures & Royalty-Free Images - iStock | Pink iris eye, Iris eye brown, Iris eye abstract

iris.pdf

020044_1_online.pdf