Multi-Biometric Iris Recognition System Based on a Deep Learning Approach

Used libraries:

os: Used for interacting with the file system (e.g., creating directories, reading file names).

cv2: OpenCV library used to read, process, and manipulate images.

numpy: Used for handling arrays and saving image data to .npy files.

Matplotlib: used for plotting the images

Tensorflow: used for training the model (CNN)

Data loading (dataset_loader.py)
this will load the iris image dataset for multi-biometric iris recognition. We are doing this because the database we are using, some images are have different sizes and incomplete.

- The database_path variable holds the path to the iris image database.
- The ignore_folders set contains the names of folders (subjects) to be excluded from processing due to incomplete or corrupted data. These folders are skipped when processing images.
- os.makedirs("results", exist_ok=True) ensures that a "results" directory is created, or if it already exists, it does nothing. This directory will store the output files.
- Two pairs of lists (left_images, left_labels and right_images, right_labels) are initialized to store the images and their corresponding subject labels separately for the left and right eyes.
- for each image in the folder, it checks whether the image is a BMP file (with .bmp extension). The image is read in grayscale mode using cv2.imread and resized to a resolution of 320x240 pixels using cv2.resize.
- The images are classified as left or right eye images based on the presence of _L or _R in the image filename. These images are appended to the corresponding lists (left_images/left_labels or right_images/right_labels).
- After loading all images, the number of left and right eye images loaded is printed to the console.

Preprocessing (preprocess.py) : Localization and Normalization

We perform two main operations on the iris images from the dataset: localization and normalization. Localization involves detecting the iris within the image, while normalization transforms the detected iris into a standard format for further analysis.

1. Loading the Dataset

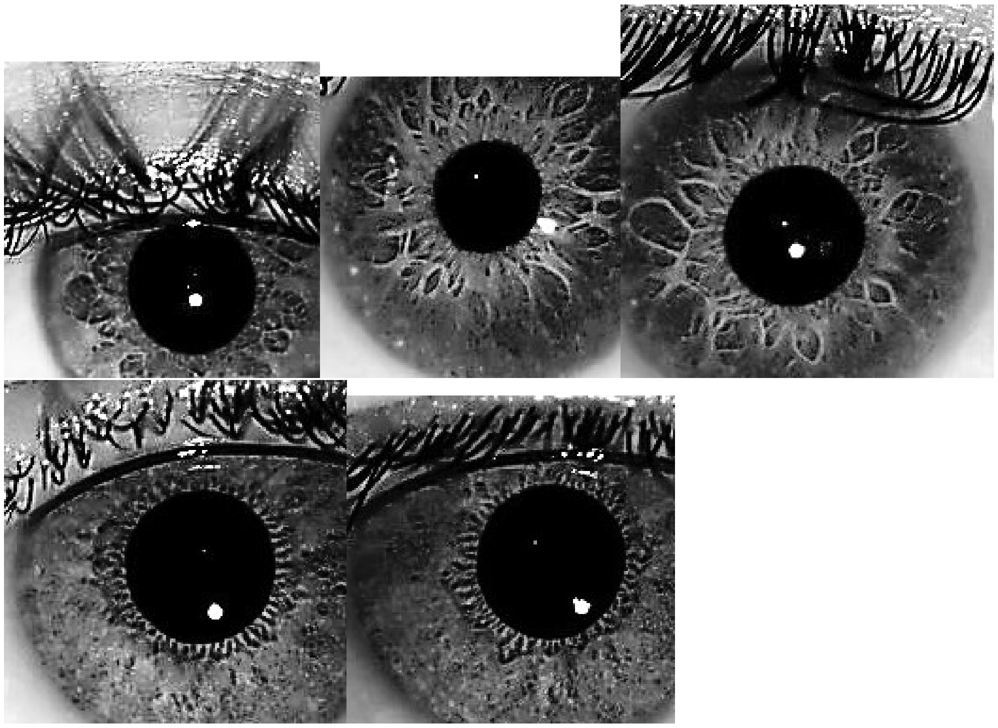
- The dataset, which includes left and right eye images, as well as their respective labels, is loaded using np.load(). The data is stored as NumPy arrays.

2. Creating Output Directories

- Two directories, localized and normalized, are created in the "results" folder using os.makedirs(). These directories will store the localized and normalized iris images respectively.
- The exist_ok=True argument ensures that the directories are created only if they don’t already exist.

3. Localization Function: localize_iris()

- **Purpose**: Detects the iris in an eye image using the Hough Circle Transform and saves the localized iris image.
- **Steps**:
    1. **Histogram Equalization**: The image is enhanced using cv2.equalizeHist() to improve contrast.
    2. **Gaussian Blur**: The image is blurred using cv2.GaussianBlur() to reduce noise.
    3. **Hough Circle Detection**: The cv2.HoughCircles() function is used to detect circular shapes (i.e., the iris) in the image.
    4. If circles are found, the function calculates the center and radius of the detected circle. It adjusts the radius to ensure it fits within the image dimensions.
    5. The region of the iris is cropped from the image using the calculated center and radius.
    6. If the cropped iris region is valid (non-empty), it is saved as a .bmp file in the localized directory.
- The localized iris image and its circle parameters (center and radius) are returned for further processing. Some localized images are:



4. Normalization Function: normalize_iris()

- Normalizes the localized iris image using Daugman's rubber sheet model, which transforms the iris region into a fixed-size (64x64) representation.

    1. The cv2.warpPolar() function is used to map the circular iris region into a polar coordinate system, transforming it into a rectangular representation.
    2. The result is saved as a .bmp file in the normalized directory.

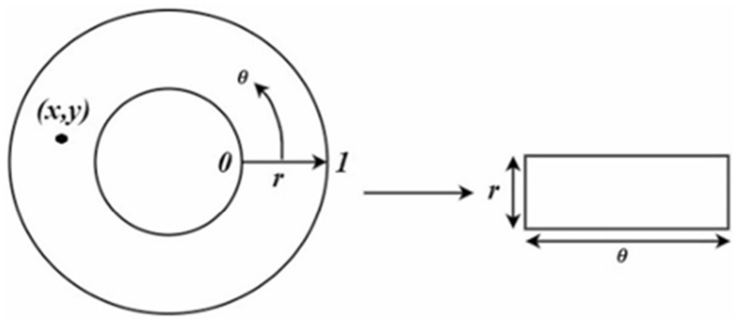
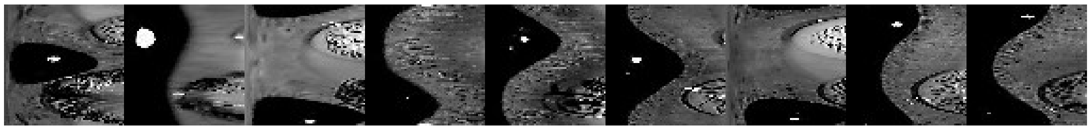


Fig. 4 Daugman's rubber sheet model to transfer the iris region from the Cartesian coordinates to the polar coordinates

- The normalized iris image is returned for use in further processing and training.

Some of the normalized images are:



5. Processing the Images

- **Left Eye Images**:
    - The script processes each left eye image (left_images) by calling the localize_iris() function to detect and localize the iris.
    - If localization is successful, it then calls the normalize_iris() function to normalize the localized iris image.
    - Both the localized and normalized left-eye images are appended to the respective lists (localized_left and normalized_left).
- **Right Eye Images**:
    - A similar process is applied to the right eye images (right_images), with localized and normalized images being saved in the corresponding lists (localized_right and normalized_right).
- After processing all images, the localized and normalized iris images are saved as .npy files using np.save(). These files are stored in the "results" folder for later use.
- The allow_pickle=True argument ensures that the arrays can store Python objects, like images, which are not basic data types.

CNN-Training model (cnn_training.py)

The process involves loading normalized iris images, preprocessing them for CNN input, training a feature extraction model, and saving the extracted features for further analysis.

1. Data Loading and Preprocessing

Two sets of normalized iris images, representing the left and right eyes, are loaded from pre-saved .npy files:

- **X_left.npy**: Contains normalized images of left eye samples.
- **X_right.npy**: Contains normalized images of right eye samples.

Each image is reshaped into a 64x64 grayscale format (64×64×1) to match the expected input shape of the CNN. Labels are assigned as follows:

- Left Eye: 0
- Right Eye: 1

The left and right eye datasets are combined into a single dataset:

- Features: X = [X_left, X_right]
- Labels: y = [y_left, y_right]

A **70-15-15 split** is applied to divide the dataset into training, validation, and test sets using **stratified sampling** to maintain class balance:

- **Training Set (70%)**
- **Validation Set (15%)**
- **Test Set (15%)**

2. CNN Architecture for Feature Extraction

A CNN model is constructed using the **Functional API** in TensorFlow, ensuring flexibility and explicit input layer definition. The architecture consists of:

- **Input Layer**: (64, 64, 1) for grayscale iris images.
- **Convolutional Layer 1**: 32 filters of size (3,3), ReLU activation.
- **MaxPooling Layer 1**: (2,2) pooling to reduce dimensionality.
- **Convolutional Layer 2**: 64 filters of size (3,3), ReLU activation.
- **MaxPooling Layer 2**: (2,2) pooling.
- **Flatten Layer**: Converts feature maps into a single vector.
- **Dense Layer**: 128 neurons, ReLU activation, outputs feature vector.
- Dense(2, activation='softmax') layer at the end will output the class probabilities for "Left Eye" (class 0) and "Right Eye" (class 1).
- The softmax function will convert the raw logits into probabilities that sum to 1, which is common in classification tasks.

The model outputs a **128-dimensional feature vector** for each image, which will be used for further biometric matching.

3. Model Training

The CNN model is compiled and trained using:

- **Optimizer**: Adam (Adaptive Moment Estimation)
- **Loss Function**: Sparse Categorical Crossentropy (since labels are integers 0 or 1)
- **Evaluation Metric**: Accuracy
- **Epochs**: 10
- **Batch Size**: 32

Training is performed on the **training set**, while validation is done using the **validation set**.

4. Feature Extraction & Storage

Once trained, the CNN model is used to extract feature vectors from images in all three datasets:

- **Training Set Features**: X_train_features
- **Validation Set Features**: X_val_features
- **Test Set Features**: X_test_features

These feature vectors are stored in .npy format for further processing:

- X_train_features.npy
- X_val_features.npy
- X_test_features.npy

5. Model Saving

To reuse the trained CNN model for future feature extraction, it is saved in **Keras format**:

cnn_model.save("results/cnn_feature_extraction_model.keras")

This allows loading the model without retraining. This successfully extracts iris features using a CNN model.

Feature extraction(feature_extract.py)

A subset of normalized iris images (first 50 images) is loaded from .npy files:

- **normalized_left.npy**: Contains left iris images.
- **normalized_right.npy**: Contains right iris images.

Each image is:

- Converted into a NumPy array.
- Reshaped into a 64×64 grayscale format (64×64×1).
- Normalized by dividing by 255.0 to scale pixel values between 0 and 1.

Feature Extraction Using CNN

The trained CNN model (cnn_feature_extraction_model.keras) is loaded, and a **feature extractor** is created by stopping at the last dense layer. This model extracts meaningful feature representations from the iris images.

1. Load the full CNN model.

2. Create a new model (feature_extractor) that outputs the feature vector from the last dense layer.
3. Predict (extract) feature vectors for left and right iris images.
4. Save extracted features as .npy files:
    - features_left.npy
    - features_right.npy

4. Image Saving and Visualization

- Two directories are created:
    - results/left_images (for left iris images)
    - results/right_images (for right iris images)
- Each image is plotted using **Matplotlib** and saved as a .png file.

Matching(match.py)

The system utilizes a pre-trained CNN-based feature extractor to extract deep features from an input query image and compare them with a stored database using **Cosine Similarity** and **Euclidean Distance** metrics.

Pre-extracted feature vectors of left and right iris images are loaded from stored .npy files:

- **features_left.npy**: Deep feature vectors for left-eye images.
- **features_right.npy**: Deep feature vectors for right-eye images.
- **dataset_left_labels.npy**: Subject labels for left-eye images.
- **dataset_right_labels.npy**: Subject labels for right-eye images.

These serve as the reference database for matching input query images.

Query Image Preprocessing

Before feature extraction, the query image undergoes preprocessing:

1. The image is loaded in **grayscale**.
2. It is resized to **64×64 pixels** to match the CNN input size.
3. It is reshaped to **(1, 64, 64, 1)** to add batch dimension.
4. Pixel values are **normalized** by dividing by 255.0.

Feature Extraction Using CNN

A pre-trained CNN feature extraction model (cnn_feature_extraction_model.keras) is loaded. The query image is passed through this model to obtain a **feature vector**, which is then flattened for similarity computation.

Match Score Computation

The extracted feature vector from the query image is compared with stored database feature vectors using two similarity metrics:

- **Cosine Similarity**: Measures angular similarity between vectors.
    - Score range: **[-1 to 1]** (higher means more similar)
- **Euclidean Distance**: Measures absolute distance between vectors.
    - Score is normalized using 1 / (1 + distance) to scale it to **(0 to 1]** (higher means more similar)

Matching Process

It computes match scores between the query image and the **corresponding eye dataset**.

1. The best match is determined by:
    - **Highest Cosine Similarity Score**.
    - **Highest Normalized Euclidean Score**.
2. The **subject ID** of the best match is extracted and formatted as a three-digit identifier.

For each query image, the system prints:

- **Best Match (Cosine Similarity):** Subject ID and similarity score.
- **Best Match (Euclidean Distance):** Subject ID and similarity score.
- **Matched Image Paths:** Paths to the retrieved matched iris images.

Additionally, match scores are saved in CSV files:

- query_match_scores_cosine.csv
- query_match_scores_euclidean.csv

This matches a query iris image against a stored biometric database using deep feature extraction and similarity comparison techniques. The retrieved matches and their corresponding scores provide a basis for further biometric analysis and fusion strategies.

Fusion (fusion.py)

To improve the accuracy of matching individuals in a multi-biometric iris recognition system by using three fusion strategies. These strategies combine the match scores from two different algorithms (Cosine Similarity and Euclidean Distance) to select the best matching subject from a set of possible candidates. The match scores are loaded from CSV files, and the subject labels are loaded from a .npy file. These strategies combine the match scores from both algorithms to select the best match based on different methods.

- **Match Scores:** The match scores from two different algorithms (Cosine Similarity and Euclidean Distance) are loaded using np.loadtxt() from CSV files. These scores represent the similarity of query images to the dataset images.
    - match_scores_cosine holds the cosine similarity match scores.
    - match_scores_euclidean holds the Euclidean distance match scores.
- **Labels:** The labels of the subjects are loaded from a .npy file (dataset_left_labels.npy). These labels correspond to the dataset of the left-eye images, but they can be adjusted for right-eye images as needed.

Fusion Strategies:

The three fusion methods used here are:

- **Highest Rank Method:**
    - **Rank Calculation:** For both match score arrays (scores1 and scores2), the ranks are calculated by sorting the scores in descending order (np.argsort(-scores1)).
    - **Consensus Rank:** The consensus rank is computed by taking the minimum rank from the two algorithms for each subject.
    - **Best Match:** The subject with the lowest consensus rank is chosen as the best match.
- **Borda Count Method:**
    - **Rank Calculation:** Ranks are calculated for both sets of match scores, similar to the Highest Rank method.
    - **Consensus Rank:** The consensus rank is the sum of the individual ranks from both algorithms, i.e., rank1 + rank2.
    - **Best Match:** The subject with the lowest total rank (sum of ranks) is chosen as the best match.
- **Logistic Regression Fusion:**
    - **Training Data:** The match scores from both algorithms are combined into a 2D feature array (X_train), where each row contains the match scores for a single subject.
    - **Labels:** Dummy labels are generated (y_train = np.arange(len(scores1))), which are used to train the Logistic Regression model. Ideally, real labels should be used, but for simplicity, this code uses a placeholder.

   - **Prediction:** The logistic regression model is trained and used to predict probabilities. The subject with the highest predicted probability is chosen as the best match.

Applying Fusion Strategies:

The three fusion methods are applied in sequence to the match scores:

- best_highest_rank holds the result from the Highest Rank method.
- best_borda_count holds the result from the Borda Count method.
- best_logistic_regression holds the result from the Logistic Regression method.

The fusion results are formatted into a string and printed to the console. These results are also appended to a text file (fusion_results.txt).

1. Highest Rank Fusion:

This method ranks the subjects based on the match scores from both algorithms, where higher scores correspond to better matches (lower ranks). The consensus rank is determined by selecting the minimum rank from both algorithms. The subject with the lowest consensus rank is selected as the best match. This method assumes that both algorithms independently produce a valid ranking of the subjects.

2. Borda Count Fusion:

In this approach, ranks from both algorithms are summed to produce a consensus rank. The subject with the lowest total rank is considered the best match. This method is particularly useful when both algorithms have different perspectives on the best match, as it provides a balanced consideration of both algorithms.

3. Logistic Regression Fusion:

In this method, a machine learning model (Logistic Regression) is used to combine the match scores from both algorithms. The match scores are treated as features, and the model is trained to predict the likelihood of each subject being the best match. The subject with the highest predicted probability is selected as the best match. This method requires a training phase, and while it can adapt to complex relationships between the scores, it requires labeled data for proper training.

After applying all three fusion strategies, the best matching subject is identified for each method. The results are stored and can be compared for performance evaluation. The comparison of the fusion methods can include aspects like accuracy, execution time, and robustness, with the ultimate goal of improving the overall performance of the multi-biometric recognition system.

The fusion of match scores from multiple biometric algorithms provides a more robust and accurate approach to identification in iris recognition systems. The three strategies (Highest Rank, Borda Count, and Logistic Regression) each offer unique advantages and can be selected based on the specific needs of the application. By combining the strengths of both the cosine and Euclidean algorithms, these fusion methods enhance the system's ability to correctly identify individuals, even in the presence of noise or variation in the data.

Steps to Run the codes:

1. run python dataset_loader.py
2. run python preprocess.py
3. run python cnn_training
4. run python feature_extract.py
5. run python match.py
6. run python fusion.py

to view .npy files you can run npy_view.py code by keeping the path of the file in it.

https://drive.google.com/drive/folders/1C_7S0fHYJlK3NGs2FknZyl86S1GRGVAd?usp=sharing