2024JCS2040

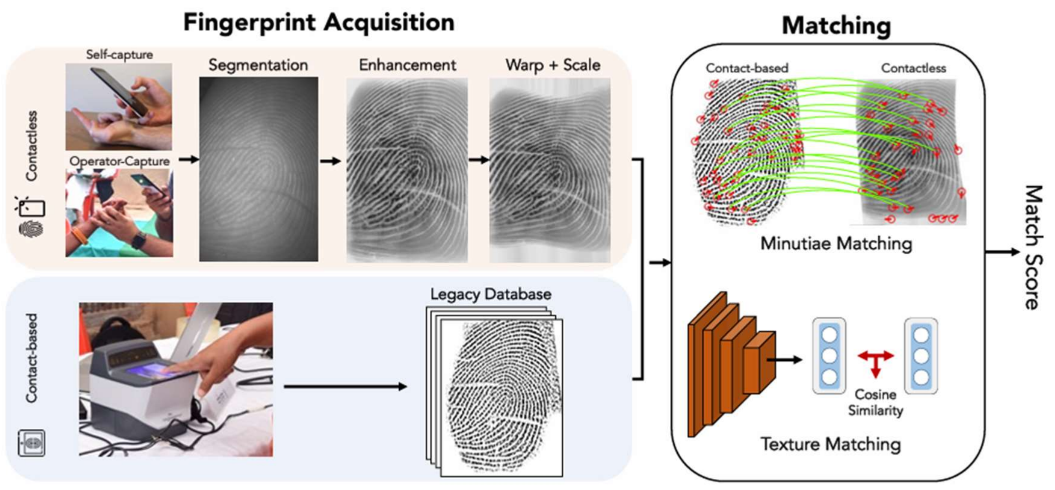
Report on - Contact to Contactless Fingerprint Matching (C to CL)

Database used - The Hong Kong Polytechnic University Contactless 2D to Contact-based 2D Fingerprint Images Database

Contains the contactless 2D fingerprint images(stored in bitmap format) and 'corresponding' contact-based fingerprints images(stored in JEPG format)

Extracting features from contactless fingerprints are very difficult than contact-based fingerprints.

The key idea is this



Here, with contactless fingerprints, we need to do preprocessing steps like segmentation, enhancement, distortion correction and scaling and then identify fingerprint orientation fields and extract minutiae points; and also from contact-based fingerprints we need to extract the minutiae points.

Here other point also we need to consider, in contact-based fingerprints the images are horizontally flipped. So while doing matching we need to match the flipped version of this to the contactless based fingerprints.

Using these minutiae points we try to match contactless fingerprints with contact-based fingerprints.

Stating for segmentation step I tried to trained the U-net based model but for pretraining it is taking more computational power in my system if we want more efficient model. So I trained with less dataset but it is not that much accurate. So, at last I used other technique(edge detection filter) for segmentation step.

Used Libraries

- **OpenCV (cv2)**: Used for image processing tasks, such as reading images, grayscale conversion, thresholding, and saving processed images.
- **NumPy (numpy)**: Used for numerical operations, such as matrix calculations and image array manipulations.
- **Torch (torch) & TorchVision (torchvision)**: Provides support for deep learning operations (not directly utilized in the current script but imported for potential enhancements).

- **Scikit-Image (skimage)**: Used for skeletonization and filtering, particularly skeletonize and sobel functions.
  skimage.morphology: For fingerprint skeletonization.
  skimage.filters: For additional filtering methods.
  skimage.util: For image conversion utilities.
- **Matplotlib (matplotlib.pyplot)**: Used for image visualization and debugging purposes.
- **OS (os)**: Used for file and directory operations, such as creating output folders and accessing image paths.
- **SciPy**: scipy.spatial.distance.cdist for computing pairwise distances between minutiae points.
- **DEAP (Distributed Evolutionary Algorithms in Python)**: For implementing the genetic algorithm.

Part 1: Fingerprint Image Preprocessing (Implementation code: pre.py)

- **Segmentation**: Identifying fingerprint ridges using edge detection.
- **Enhancement**: Improving fingerprint contrast and quality.
- **Distortion Correction**: Addressing image misalignment and distortions.
- **Alignment**: Ensuring contactless fingerprints match the orientation of contact-based ones.(in database the alignment is not appropriate with the contact-based fingerprints. So, I done this alignment)

1) Image Acquisition

Fingerprint images are loaded from two datasets:

- Contact-based fingerprints from: fingerprints/contact-based_fingerprints/first_session
- Contactless fingerprints from: fingerprints/contactless_2d_fingerprint_images/first_session

Each image is read in grayscale for further processing.

2) Image Preprocessing

a) Image Alignment (for Contactless Fingerprints Only)
Contactless fingerprint images are rotated **90 degrees counterclockwise** to align them with contact-based fingerprints.

b) Segmentation
Technique used: A Sobel edge detection filter is applied to extract fingerprint ridge structures.
Output: segmented.png (an 8-bit grayscale edge-enhanced image).

c) Enhancement
Technique used: Contrast Limited Adaptive Histogram Equalization (CLAHE) is applied to improve image contrast.

Output: enhanced.png (contrast-enhanced fingerprint image).

d) **Distortion Correction**

Technique: A simple **affine transformation (translation by 5 pixels)** is applied as a placeholder for future distortion correction.

Output: transformed.png (preliminarily corrected image).

Storage of Processed Images

- Processed images are stored in an output directory: f2_processed_output/

Part 2: Fingerprint Orientation Field Computation(implementation code: orien.py)

In this for pre-processed fingerprint images applies Sobel filtering to compute image gradients, derives the orientation field, and saves the processed output.

Orientation Field Computation:

- The grayscale fingerprint image is loaded using OpenCV.
- Sobel filters (horizontal and vertical) are applied to compute gradients:
  ```
  sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
  sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
  ```
- The orientation field is obtained using np.arctan2(sobely, sobelx), which calculates the angle of gradient vectors.
- The field is normalized and saved as an image for visualization.

The python script iterates through preprocessed images stored in the OUTPUT_PATH directory.It checks for the presence of enhanced.png in each fingerprint folder and processes it accordingly.The processed images are saved in the corresponding subfolders.

The computed orientation fields are stored in the output directory as orientation_field.png.

Challenges

- **Image Quality:** Variations in contrast and noise levels could affect gradient computation.
- **Edge Handling:** Boundary effects may introduce artifacts in the orientation field.
- **Dataset Variability:** Contact-based and contactless fingerprints exhibit differences in ridge structure and image resolution, requiring normalization adjustments.

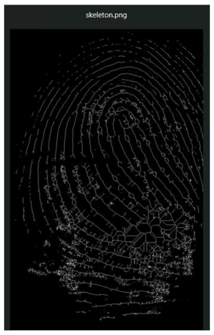
Part 3: Fingerprint Minutiae Extraction(minu.py)

Minutiae points include ridge endings and bifurcations, which are crucial for fingerprint matching.

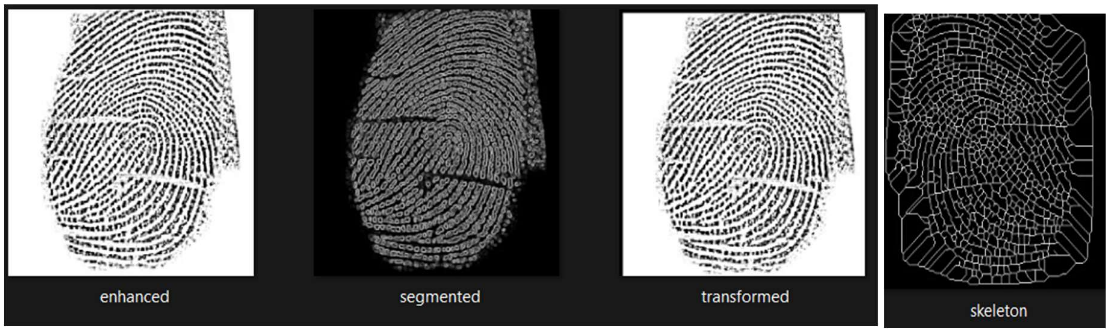
1. **Binarization**: The grayscale fingerprint is thresholded using cv2.threshold(image, 127, 255, cv2.THRESH_BINARY) to create a binary image.
2. **Skeletonization**: The binary image is skeletonized using skimage.morphology.skeletonize() to reduce ridges to a single-pixel width.
3. **Minutiae Detection**: A placeholder method is used, extracting all white pixels from the skeletonized image.

   For each fingerprint image from pre-processed dataset, Extracts minutiae points and stores them in a text file. np.column_stack(np.where(skeleton > 0)) extracts non-zero pixels as potential minutiae.

Skeletonized Fingerprints (skeleton.png) and **Extracted Minutiae Points** (minutiae.txt)



Now we separately extract minutiae from contact based fingerprints, here we do horizontal flip (minu_contact.py).



Part 4: Matching (match.py)

The methods used for matching the fingerprints are

1. Hough Transform Matching
2. Genetic Algorithm Matching
3. Core Point Matching

The objective is to compare their performance based on accuracy, precision, recall, and F1-score using minutiae points extracted from fingerprint images.

The fingerprint matching process consists of the following steps:

1. **Loading Minutiae Points**: Minutiae points are loaded from text files containing fingerprint feature coordinates.
2. **Applying Matching Techniques**: Three different algorithms are used to determine the degree of similarity between two fingerprint datasets.
3. **Computing Performance Metrics**: Matching results are evaluated using accuracy, precision, recall, and F1-score.

Results are saved in a CSV file for analysis.

1. Hough Transform Matching

The Hough Transform method aligns minutiae points based on their centroids. The transformation involves:

- Calculating the centroid of both fingerprint minutiae sets.
- Translating one set of minutiae points to align with the other.
- Computing Euclidean distances between matched points and determining matches based on a threshold.

2. Genetic Algorithm Matching

The Genetic Algorithm (GA) is used to optimize fingerprint alignment:

- Individuals in the population represent transformation matrices applied to the minutiae points.
- The fitness function evaluates the number of matching minutiae points, penalizing excessive transformation.
- Selection, crossover, and mutation operators are applied iteratively to optimize the transformation matrix.

3. Core Point Matching

This method aligns fingerprints using their core points:

- The core points (average minutiae positions) of both sets are computed.
- Minutiae points are translated based on core points to align fingerprints.
- Matches are determined using Euclidean distances.

Performance Metrics

The effectiveness of each method is evaluated using the following metrics:

1. **Accuracy**: Measures the proportion of correctly identified minutiae points.

$$\text{Accuracy} = \frac{TP}{TP + FP + FN + TN}$$

2. **Precision**: Indicates the proportion of correctly matched minutiae points out of all matched points.

$$\text{Precision} = \frac{TP}{TP + FP}$$

3. **Recall**: Measures how well the method identifies relevant minutiae points.

$$\text{Recall} = \frac{TP}{TP + FN}$$

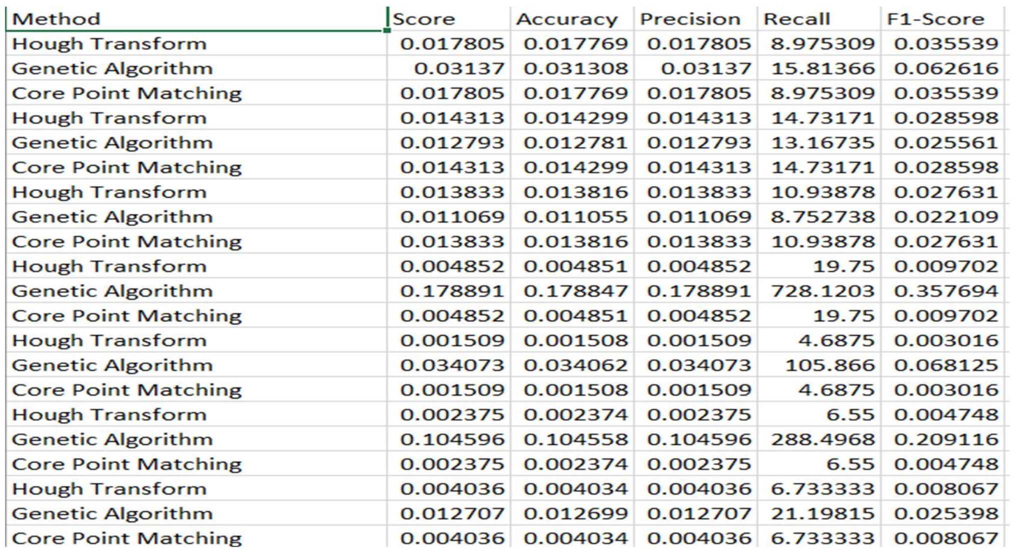
4. **F1-score**: The harmonic mean of precision and recall.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

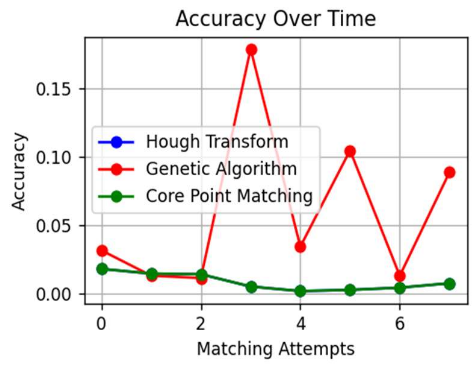
Where:

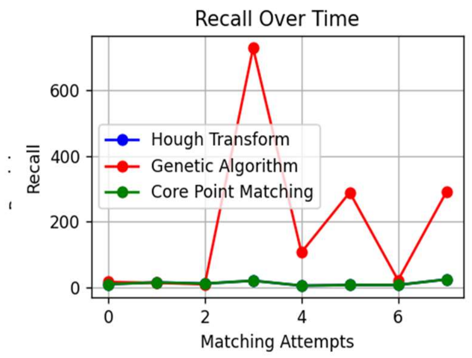
- TP (True Positives): Correctly matched minutiae points.
- FP (False Positives): Incorrectly matched minutiae points.
- FN (False Negatives): Missed minutiae points.
- TN (True Negatives): a non-matching minutia point is correctly identified as not matching.

Some Matching performace metrics for some inputs are



Method	Score	Accuracy	Precision	Recall	F1-Score
Hough Transform	0.017805	0.017769	0.017805	8.975309	0.035539
Genetic Algorithm	0.03137	0.031308	0.03137	15.81366	0.062616
Core Point Matching	0.017805	0.017769	0.017805	8.975309	0.035539
Hough Transform	0.014313	0.014299	0.014313	14.73171	0.028598
Genetic Algorithm	0.012793	0.012781	0.012793	13.16735	0.025561
Core Point Matching	0.014313	0.014299	0.014313	14.73171	0.028598
Hough Transform	0.013833	0.013816	0.013833	10.93878	0.027631
Genetic Algorithm	0.011069	0.011055	0.011069	8.752738	0.022109
Core Point Matching	0.013833	0.013816	0.013833	10.93878	0.027631
Hough Transform	0.004852	0.004851	0.004852	19.75	0.009702
Genetic Algorithm	0.178891	0.178847	0.178891	728.1203	0.357694
Core Point Matching	0.004852	0.004851	0.004852	19.75	0.009702
Hough Transform	0.001509	0.001508	0.001509	4.6875	0.003016
Genetic Algorithm	0.034073	0.034062	0.034073	105.866	0.068125
Core Point Matching	0.001509	0.001508	0.001509	4.6875	0.003016
Hough Transform	0.002375	0.002374	0.002375	6.55	0.004748
Genetic Algorithm	0.104596	0.104558	0.104596	288.4968	0.209116
Core Point Matching	0.002375	0.002374	0.002375	6.55	0.004748
Hough Transform	0.004036	0.004034	0.004036	6.733333	0.008067
Genetic Algorithm	0.012707	0.012699	0.012707	21.19815	0.025398
Core Point Matching	0.004036	0.004034	0.004036	6.733333	0.008067





Matching Results

The fingerprint matching algorithms were tested on two different fingerprint minutiae files

- The **Hough Transform Matching** method provides reasonable alignment but may suffer when fingerprints have rotation or significant translation variations.
- The **Genetic Algorithm Matching** method shows improved performance due to its optimization-based approach, allowing for flexible transformations.
- The **Core Point Matching** method performs well when core points are accurately detected but may be less effective when fingerprints have distortions or noise.

The Genetic Algorithm approach appears to provide robust performance by optimizing transformations. However, the choice of method depends on specific application requirements, such as computational efficiency and resistance to variations in fingerprint images.

Steps to run:
1) download the required dataset and libraries
2) run python pre.py → for preprocessing
3) run python orien.py → for orientation
4) run python minu.py → for minutiae points extraction of contactless fingerprints
5) run python minu_contact.py → for minutiae points extraction of contact-based fingerprints
6) run python match.py → matching two fingerprints from C to CL fingerprints
7) for analysis of different matching methods run python plot.py

In every step the intermediate results are stored.