A brief overview of our code:

This model is a working fingerprint recognition system that highlights a proposed algorithm design to increase the speed and efficiency of fingerprint recognition from an identification standpoint.

The design of the algorithm:

By taking a fingerprint image, the software leverages capabilities in using only areas of interest rather than the entire fingerprint. We extract the areas of interest in the following process, starting with the enrollment:

1. Extract and count the total number of significant minutiae terminations and bifurcations and average the total number of minutiae.

- Starting within the Erollment_Phase file, the folder path that holds raw image files is identified and then run through a pipeline function that is within the Database_Load_Processor file.
- In this file, each fingerprint within the folder is enhanced using Normalization, Segmentation, Ridge Angle calculations, Ridge Frequency calculations, Gabor Filtering, Skeletonization, Minutiae Extraction, and Singularity extraction all handled within their respective files. This process leaves the program with a single enhanced skeleton image that is centered using the singularity of the fingerprint.

1. From there, divide the fingerprint image into equal 1x1 square grids of the desired size.
2. Calculate which fingerprint grid segments have the highest amount of minutiae, classifying them as “High-Density Grids.”

- Now that the process is left with a single enhanced fingerprint skeleton, the grid division begins. This happens within our function “divide_into_grids()” which allows you to specify the the individual grid sizes.
- A list is then stored of the respective High-Density Grids.

1. Within each of the selected High-Density Grids, we classify the minutiae by using minutiae triplets within each grid. Each grid in turn holds several unique minutiae triplets in geometric shapes

- Each grid is run through a file called “Triplet_Extraction.” In this file, each minutia coordinate is captured and then assigned to its respective grid, and within each grid, the triplets are then formed with coordinates, angles, and distances to help the minutiae points become invariant to distortions and adjustments of the fingerprint placement on the sensor.

1. After all the triplet data has been calculated, the data is then stored in a SQLite database for querying.
2. From there, the database is filled with multiple fingerprints from different people within the system. At this point, the identification process now begins.

- Within the file “database_worker” the triplet data of the fingerprint is then queried into a SQL database using 3 distinct tables.

1. A probe fingerprint (a fingerprint to be identified) is then put through the same processes as steps 1-5, excluding the SQLite storage portion

- Using the file “Probe_Processor,” the probe fingerprint from a sensor is put through the functions probe_processor and probe_dataBuilder completes the same enhancement and extraction process as the fingerprints in the database went through.

1. Each fingerprint within the database is then deemed as a “candidate fingerprint” and each candidate is run through a direct matching process against the probe and then given a matching score.

- In the file “high_density_matching,” a group of utility functions are then used by the function match_probe_fingerprint() takes in the probe data and the database path to implement matching.
- The function connects to the database and then queries the fingerprint from the MinutiaeData table. From the saved results given after that query, each FingerprintID is queried for its GridLabels, MinutiaePoints, Distances, and Angles from the MinutiaeData table.
- The results from each query of the fingerprints are then saved and cycled through utility functions to compare each GridLabel, MinutiaePoints, Distances, and Angles against the Probe fingerprint's identical dictionary of data.

1. The lowest matching score is deemed to be the identified fingerprint

Instrcutions:

Step 1:

- Once the virtual environment is set up with the proper requirements, navigate to the file named “Enrollment_Phase.py”. Within this file, you will find a declared raw folder path. Alter this path to include the path from your directory of the fingerprint images you would like to enroll in.
- Run this file in a dedicated terminal. Now the Database full of the fingerprints’ stored data is within your directory named “Database. db”

Step 2:

- Ensure the Database. db was successfully made with proper integrity.
- Navigate to the file named “Matching_Main.py” within the same directory as “Enrollment_Phase.py”
- In this file, you will find a database path declaration, copy the Database. db path created from the previous file, and enter it in the db_path declaration.
- You will find a probe_path, this will be the path to the probe fingerprint image you would like to identify.

- Run this file, the resulting most likely match will be output as well as the matching scores of the remainder fingerprints.