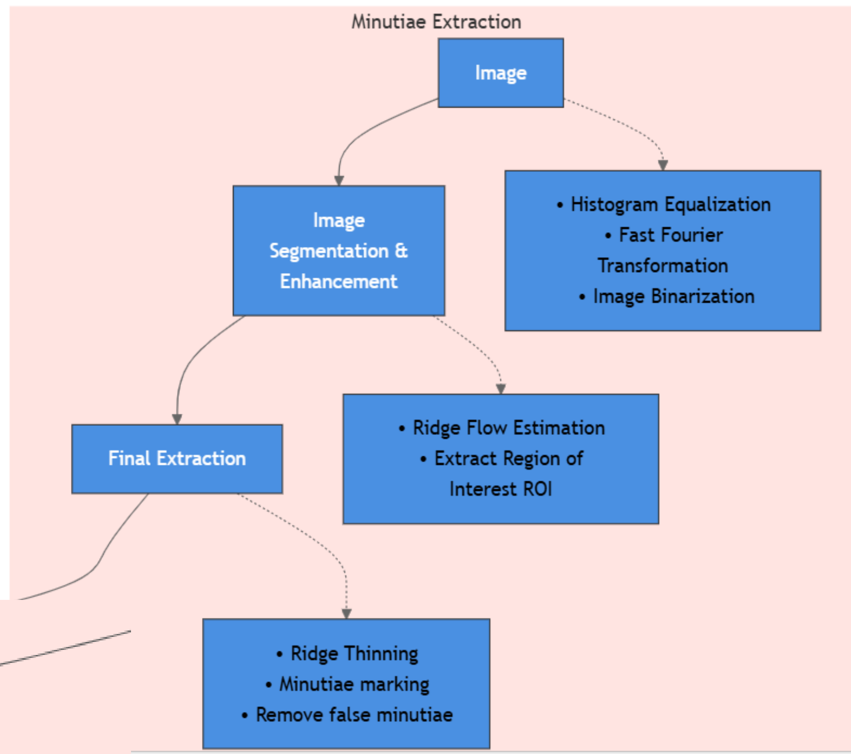
By Nour Bassem ID: 2022/00080

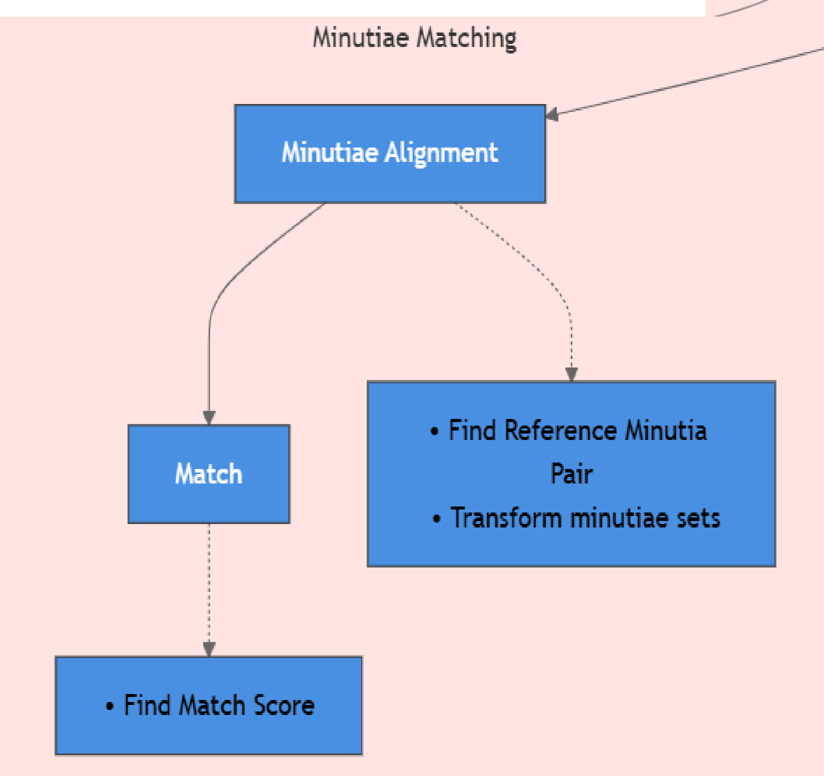
<u>Fingerprint recognition</u>

1- Sample of dataset:

2- Sequence of steps:

1) **Image Acquisition**: There will be a folder for each person (10 people) containing five fingerprint images. The extracted features are then matched against the dataset to identify the person based on the folder name.





2) **Image Enhancement**: Improve image quality by applying histogram equalization for contrast enhancement and Gaussian blur to reduce noise
3) **Image Segmentation:** Threshold-based segmentation to isolate the fingerprint region
4) **Image Skeletonization**: Thin the fingerprint ridges to a one-pixel-wide representation, simplifying the ridge patterns for analysis
5) **Feature Extraction**: Detect minutiae points like ridge endings and bifurcations
6) **Feature Matching**: Use cosine similarity to compare features and identify the person

Phase 2: Implementation Documentation

Overview

The steps include preprocessing, minutiae extraction, matching fingerprints, and training machine learning models for classification.

---

1. Data Preparation

- Dataset Loading:
  - Mounted Google Drive to access the dataset using drive.mount().
  - Loaded fingerprint images from a specified directory.

```
img = cv2.imread('/content/drive/MyDrive/imgprocessing_project/Images
dataset/train_data/person1/00000_00.bmp', cv2.IMREAD_GRAYSCALE)
cv2_imshow(img)
```

---

2. Image Preprocessing

- Histogram Calculation:
  - visualized the histogram of pixel intensity values.

```
hist = cv2.calcHist([img], [0], None, [255], [0, 256])
plt.plot(hist)
```

- Contrast Stretching:
  - using minimum and maximum pixel values.

```
stretched = ((img - r_min) * ((s_max - s_min) / (r_max - r_min)) + s_min).astype(np.uint8)
```

- CLAHE:
  - Enhanced the image using Contrast Limited Adaptive Histogram Equalization (CLAHE).

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
enhanced_img = clahe.apply(img)
```

- Noise Reduction:
  - Used Gaussian Blur to reduce noise.

```
blurred_img = cv2.GaussianBlur(enhanced_img, (5, 5), 0)
```

- Skeletonization (Thinning):
    - Applied thinning to reduce the fingerprint to a single-pixel-wide skeleton.

```
thinned = cv2.ximgproc.thinning(binary_img)
```

---

3. Minutiae Extraction

- Extracted ridge endings and bifurcations from the thinned fingerprint image.
- Algorithm:
    1. Iterate through each pixel.
    2. Count white neighbors.
    3. Classify based on the count:
        - Ridge Ending: 1 neighbor.
        - Bifurcation: More than 2 neighbors.

```
if neighbor_count == 1:  # Ridge ending
    input_minutiae.append((i, j, 'ending'))
elif neighbor_count > 2:  # Bifurcation
    input_minutiae.append((i, j, 'bifurcation'))
```

---

4. Fingerprint Matching

- Minutiae-Based Matching:
    - Compared input fingerprint minutiae with stored fingerprint minutiae in the dataset.
    - Calculated similarity score based on matched points.

```
similarity_score = matched_points / max(len(input_minutiae), len(stored_minutiae))
```

- Result:
    - Determined a match if the similarity score exceeded a predefined threshold.

```
if similarity_score > threshold:
    print(f"Input fingerprint matches with {folder_name}'s fingerprint {filename}.")
```

---

5. Feature Extraction and Data Augmentation

- Features Extracted:
  - Minutiae counts (ridge endings and bifurcations).
  - Orientation map (using Sobel operator).
  - Ridge density.

```
ridge_density = np.sum(thinned_img) / 255
orientation = np.arctan2(sobely, sobelx).mean()
```

- Data Augmentation:
  - Performed rotation and flipping to enhance training data.

```
for angle in [15, -15]:
   matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
   rotated = cv2.warpAffine(image, matrix, (w, h))
```

---

6. Model Training and Evaluation

- Feature Extraction from Dataset:
  - Processed each fingerprint image to extract feature vectors.

```
feature_vector = extract_features(img)
```

- Model Training:
  - Used K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest classifiers.
  - Tuned hyperparameters using GridSearchCV.

```
grid_search = GridSearchCV(model, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

- Splitting:
  - Split dataset into training and testing sets.
  - Evaluated models using accuracy metrics.

```
accuracy = accuracy_score(y_test, y_pred)
print(f"{model_name} Accuracy: {accuracy * 100:.2f}%")
```

---

7. Testing

- Tested the trained models on unseen fingerprints.

```
test_fingerprint(test_image_path, best_models)
```

- Displayed predictions for each model.

---

8. Results

- Successfully matched input fingerprints with the dataset.
- Achieved high accuracy using optimized machine learning models.