ENCM 509
Fundamentals of Biometric Systems Design
Department of Electrical and Software Engineering
Schulich School of Engineering

Laboratory # 5

Fingerprint Biometrics
Part II: Matching

1 Introduction

This Lab is Part II of the exercise on fingerprint image processing and matching. The input data for this lab is your fingerprints collected using the provided DigitalPersona USB devices. You can also use the limited sample data provided on D2L.

The focus of Part II is to investigate two fingerprint matching algorithms. We will use two different notebooks (`Lab05-Fingerprint2-Minutiae.ipynb` and `Lab05-Fingerprint2-GaborFeatures.ipynb`), each one corresponding to a specific type of features used for the matching. In addition, we will use `Lab05Fingerprint2gab.ipynb` for the exercise explaining the usage of Gabor filter for ridge detection.

This lab uses the library called OpenCV[1]. It is available on Anaconda repositories, but it is not installed by default, so you will need to install it manually. To proceed with the OpenCV installation, you should follow the same steps we used to install the PyAgrum library:

1. Go to the main menu > Anaconda3 (64-bit) > Anaconda Powershell Prompt.
2. Type the command: `conda install opencv -c conda-forge`.
3. Accept all the dependencies typing `y` for "yes".

2 Fingerprint matching

The matching in this lab is based on two approaches:

- Using the count of the minutiae of two types: ridge ending and bifurcation, and
- Using scores obtained based on the Gabor filtering.

2.1 Matching using the minutiae count

For this approach based on minutiae, the notebook `Lab05-Fingerprint2-Minutiae.ipynb` is used. It requires extraction of the ridge skeleton, and, consequently, finding the coordinates of the minutiae on the skeleton images. The *skeleton* image (which is a 2D array) is considered a *template*. The previously collected templates are "database templates", and the one submitted for matching is a "probe template".

For matching, image alignment of the "database template" and the "probe template" is required and is performed by the function `align2(...)`. The alignment is achieved by calculating the 2-dimensional correlation for every minutiae location in the database and the probe templates. The

[1] https://opencv.org/

highest correlation factor determines the part of the database template that best matches the probe. The matching score determines the similarity of two fingerprint templates Fp1 and Fp2.

The minutiae features are compared as follows: a minutia m_i in Fp1 and a minutia m_j in Fp2 are matched if the Euclidean distance between them is smaller than some pre-determined threshold.

2.2 Gabor filter

In this section, is describe the use of Gabor filter for Ridge detection (notebook Lab05-Fingerprint2gab.ipynb) and for Matching using Gabor features (notebook Lab05-Fingerprint2-GaborFeatures.ipynb).

2.2.1 Gabor filter for ridge detection

Step 5 of the fingerprint processing (see Lab 4 - Part I) involved ridge detection using a 2D Gabor filter. Let us explore this special filter in detail as it is important for the second approach we consider for the fingerprint matching.

A Gabor filter belongs to a tuned band-pass filter bank. It has a Gaussian transfer function in the frequency domain. The 2D Gabor filter is a Gaussian modulated by a complex sinusoid (with the centre frequencies along x and y-axes respectively). It is characterized by the frequency (f) and angle (angle).

An example of the Gabor filter is provided in the notebook Lab05Fingerprint2gab.ipynb. A function called gabor_kernel(...) is available in the Scikit-Image library[2].

A fragment of such implementation is shown below.

```
# fragment of Scikit-Image implementation of Gabor filter.
# source: https://is.gd/dIonDF

def gabor_kernel(frequency, theta=0, bandwidth=1,
                 sigma_x=None, sigma_y=None, n_stds=3, offset=0):
    .    .    .
    y, x = np.mgrid[-16:16, -16:16]

    rotx = x * np.cos(theta) + y * np.sin(theta)
    roty = -x * np.sin(theta) + y * np.cos(theta)

    g = np.zeros(y.shape, dtype=np.complex)
    g[:] = np.exp(-0.5 * (rotx ** 2 / sigma_x ** 2 + roty ** 2 / sigma_y ** 2))
    g /= 2 * np.pi * sigma_x * sigma_y
    g *= np.exp(1j * (2 * np.pi * frequency * rotx + offset))

    return g
```

To create the filter, a Gabor kernel shall be created using the size specified by the "window", for example, 32×32. The other filter parameters include sigma_x, sigma_y, angle which specify an orientation (usually, one of 8), and the frequency (computed using local ridge frequencies for the given image), respectively.

[2]https://scikit-image.org/docs/stable/api/skimage.filters.html#skimage.filters.gabor_kernel

To create the filter, we use the function available in the notebook `Lab04-Fingerprint2gab.ipynb`:

```
def gabor_template(im, angle, freq=0.11):
    r, c = im.shape
    ysize = int(np.floor(r/10))
    t = int(np.floor(ysize/8))
    fi = np.real(gabor_kernel(freq, theta=angle, n_stds=t))

    return fi
```

The next step is to perform the filtering, which is a convolution between the filter and the image, using the function `convolve(...)` from SciPy library. In addition, we will use the function which performs the equalization of the result image for a better visualization:

```
def gabor_filtering(img, fi):
    # 'mode' specifies how the convolution algorithm will deal with the borders
    If1 = equalize_adapthist(ndi.convolve(img, fi, mode='wrap'))
    If1 = ((If1 - np.min(If1)) / (np.max(If1) - np.min(If1))) * 255 # normalize
    return If1
```

The filter result is illustrated by plotting the 2D image (see the Fig. 1) for 8 various orientations:

```
# +0.1 because otherwise Python discart the last value
angle = np.arange(0, np.pi—np.pi/8+0.1, np.pi/8)
# filter size: x, y
xsize = 32; ysize = 32
# standard deviation of the gaussian envelope
dx = 8; dy = 8
# frequency
fq = 0.11

plt.figure(figsize=(10,5))

# filtering and feature extraction
for i,a in enumerate(angle):
    # build the filter
    gabor = np.real(gabor_kernel(fq, theta=a, sigma_x=dx, sigma_y=dy, n_stds=4))

    # show Gabor filters
    plt.subplot(2, 4, i+1)
    plt.imshow(gabor, cmap='gray', aspect='auto')
    plt.tick_params(left=False, bottom=False, labelleft=False, labelbottom=False)
```

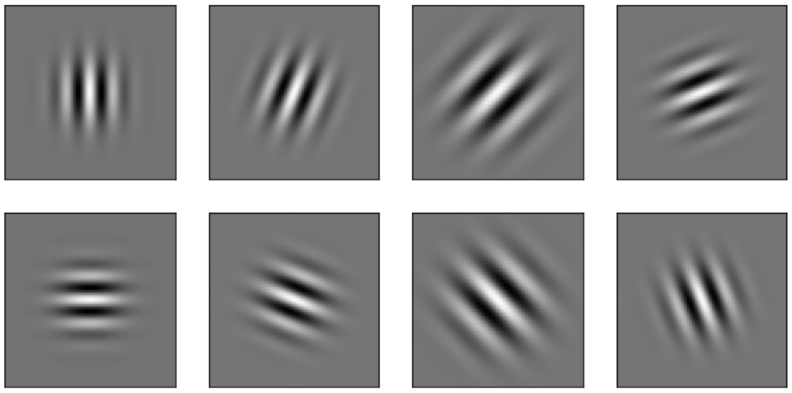
Figure 1: Gabor filters (kernels) for several orientations.

To apply the Gabor filter to the fingerprint image, 1) load the image; 2) create the Gabor filter with the desired parameters; and 3) perform the convolution between the filter and your image. These steps are shown below:

```
# 1)
img = imread('FPsamples//1.bmp', as_gray=True)
# 2)
fi = gabor_template(img, angle=np.pi/4)
# 3)
img_f = gabor_filtering(img, fi)
```

Figure 2 shows the original image, the filter created (using angle $\pi/4$), and the convolution result.

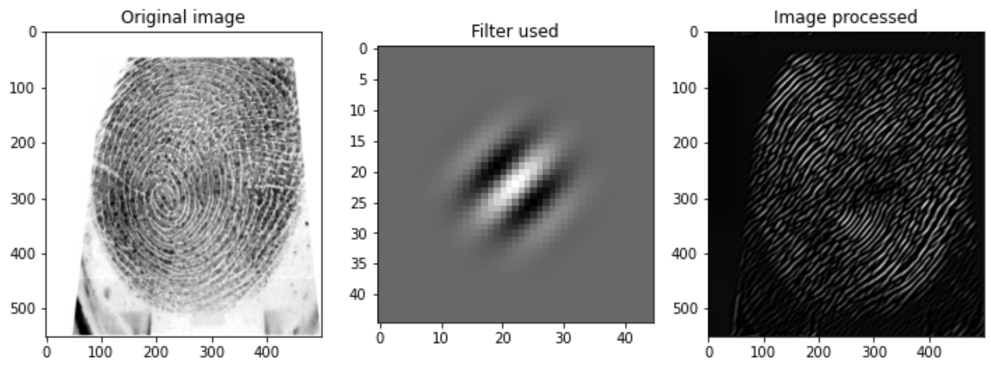

Figure 2: Application of a Gabor filter to a fingerprint image.

2.2.2 Gabor feature-based matching

For this approach, we provide the Jupyter notebook `Lab05-Fingerprint2-GaborFeatures.ipynb`. After loading two fingerprint images, you can calculate the score between them using the function `MatchGaborFeat(...)`, as shown in the code below:

```
# Loading two images of the same individual:
im1 = img_as_ubyte(imread('BTLab_Fingerprints/right_thumb/14.bmp', as_gray=True))
im2 = img_as_ubyte(imread('BTLab_Fingerprints/right_thumb/15.bmp', as_gray=True))

# Calculating the score between im1 and im2:
score1 = MatchGaborFeat(im1, im2)

# Printing the result
print('Score using Gabor features: %.4f' % (score1))

OUT: Score using Gabor features: 15.2020
```

If you want to visualize the features calculated for each fingerprint, add the argument, `plot_res=True` to the function `MatchGaborFeat(..., plot_res=True)`. The result is shown in Fig. 3: the first row contains the eight directional Gabor kernels (filters) and the second and third rows show the Gabor features extracted from fingerprints #1 and #2, respectively.

After filtering using Gabor kernels, the absolute deviation from the mean on the 16×16 blocks of the Gabor filtered images is calculated. Remember that each Gabor kernel is oriented in a specific angle, thus detecting the patterns with different orientations. The deviation values are combined in a long vector of numbers then, the matching score is calculated as the mean of the differences between the two feature vectors.

3 Lab Report

Your report in the form of a Jupyter Notebook/Python (file extension `.ipynb`) shall include the description of each exercise with illustrations/graphs and analysis of the results (marks are distributed

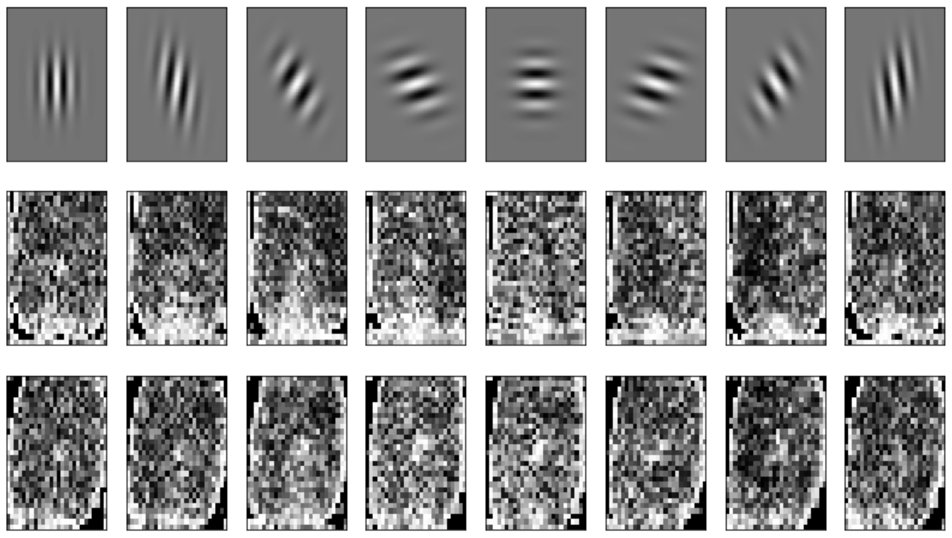
Figure 3: Results of the application of eight Gabor kernels (filters) to two fingerprint images.

as shown in the Exercise section, 10 marks total). Save your Notebook using menu "Download As" as `.ipynb`, and submit to D2L dropbox for Lab 5, by the deadline (next Thursday).

4 Lab Exercise in Jupyter Notebook with Python

For the following exercises, use the fingerprints you have recorded, or the ones provided on the D2L. A detailed description of each exercise to be included in your report (10 marks total) is given below.

- **Exercise 1** (1 mark): Use the demo file `Lab04Fingerprint2gab.ipynb` to perform the Gabor filtering on your fingerprint image, while adjusting the parameters `frequency` and `angle` of the filter. Choose two different values for each of those parameters, one at a time, and visually evaluate the results. What is the impact of changing the parameters of the Gabor filtering results (visually)? Draw the conclusions.

- **Exercise 2** (3 marks): Use 9 of your fingerprints as a database, or a gallery. Use the remaining one as a probe. The database of 9 templates (feature vectors) represents the same "individual".

  Develop a simple identification procedure (use loops that repeat 1 to 1 matching several times), to match a probed fingerprint of your finger (e.g. left thumb) against 9 different impressions of the same finger. Use the Jupyter notebook `Lab04Fingerprint1.ipynb` that performs 1 to 1 matching.

  Record the scores (use a table - see the first Lab on Markdown how to do a table in Jupyter notebooks) for the **minutiae based matching**. Note that for this type of matching, the higher the score the better the match. Record the number of matches against the same fingers (True Positives - TP), and the number of mismatches against the same fingers (False Negatives - FN). How the choice of the threshold affects the matching? Use the numbers or graphs to illustrate your answer.

- **Exercise 3** (2 marks): Repeat the same experiment (using the same images) from Exercise 2, but **instead of minutiae based matching, use the method based on the Gabor filter**.

Note that the lower the score, the better the match. Again, record the numbers of the True positives and False negatives.

Compare the results of Gabor score matching with the minutiae based matching. Draw conclusions upon both experiment results.

- **Exercise 4** (4 marks): Use one of your fingerprints (e.g. left thumb) to be used as the probed one. Use the database with one impression of the same finger and 10 of some other finger(s). Therefore, you have a database with 11 templates one of which represents the same "individual", and the rest of the templates represent another "individual(s)". Develop a ranking identification procedure for minutiae-based matching only. A sample of such procedure is as follows:
  - Record the matching scores for each comparison; use some index corresponding to the fingerprint number. Sort the matching scores: the top score is the maximum score for the Minutia based approach.
  - Do you have a true match? If your true match is not on top of your ranking, which rank is it? Use this rank's score as a "thresholding score" that separates a top-ranked group of the closest matches from the others.
  - Record the number of matches against different fingers as False Positives, and the mismatches as True Negatives. Some False positives may belong to the top ranked group.
  - Draw the conclusions upon your experiment, such as the threshold choice etc.

5 Acknowledgments

The acquisition executable `digitalPersona.exe` was created in the BT Lab based on the Digital Persona SDK. The Python implementation of minutiae extraction using OpenCV is credited to Manuel Cuevas. The template for this Lab was developed by Dr. H. C. R. Oliveira (postdoc in the Biometric Technologies Laboratory in 2020-2022). We also acknowledge this course TAs, I. Yankovyi and M. Zakir, for verifying this lab code.

Dr. S. Yanushkevich
January 30, 2023.