EE588 Advanced Image Processing
Project #2 Morphology
(due Monday, Oct. 8, 2018, 10:30:00 am)

Matlab commands are specified in the text and in `blue Courier font` and python commands in parenthesis and in `red Courier font`; variables and filenames (or commands that are consistent between Matlab and python) are specified in black Courier font. In the following python syntax, I have used `import matplotlib.pyplot as plt`, `import numpy as np`, `import os`, and `import glob`; additionally, you probably want to use `%matplotlib inline` to include figures inline in your jupyter notebook.

Your code should be well commented, and your submission should include enough narration to lead me through your solution. You should be briefly discussing your decisions, your approach to coding, and your results as you progress through the code. Generally this will mean, for each sub-instruction, a brief narrative, your commented code, and the requested output (printed or images displayed). If it is easier, you may upload a separate pdf document for each part (please name each file according to the part so that I can easily navigate—e.g., part_a.pdf). Please also include the last page of this assignment as a cover page to assist me in grading. You may fill that out and upload it as a separate document, or include it as the first page of your merged pdf.

(a) **Introductory morphology I.**
In this problem, you will learn to use the basic erosion, dilation, opening, and closing commands to morphologically process an image.

(i) Download textbook figure Figure 9.11(a) (available on canvas as `Fig0911(a).png`), read it into variable `A`, and display the image. Define the structuring element `B_test=strel('disk',11)` (`B_test=skimage.morphology.disk(11)`). What is the variable type and dimensions of `B_test`? What happens when you try to visualize `B_test` using `imshow(B_test)` (`plt.imshow(B_test,cmap='gray')`)? What happens when you try to visualize `B_test` using `imshow(B_test.getnhood)` and what are the dimensions of the image (no additional instructions for python)? Define a structuring element `B` to be the same as the structuring element used in Figure 9.11 in the textbook. Looking at `help strel` (`help(skimage.morphology.selem)`) will be helpful here.

(ii) Define the erosion of image `A` by structuring element `B` as `Ac=imerode(A,B)` (`Ac=skimage.morphology.binary_erosion(A,B)`). Display `Ac`, which should correspond to Figure 9.11(c).

(iii) Define `Ad1` as the dilation of image `Ac` by structuring element `B`, i.e., `Ad1=imdilate(Ac,B)` (`Ad1=skimage.morphology.binary_dilation(Ac,B)`). Display `Ad1`, which should correspond to Figure 9.11(d). We know that the dilation of an erosion is equivalent to an opening. Define `Ad2` as the morphological opening of image `A` by structuring element `B`, i.e., `Ad2=imopen(A,B)` (`Ad2=skimage.morphology.binary_opening(A,B)`). Display image `Ad2`, which should also correspond to Figure 9.11(d). Prove that `Ad1` and `Ad2` are equivalent by computing the maximum of the absolute difference between `Ad1` and `Ad2` and printing this value to the command window. (In python, you will need to convert the boolean arrays `Ad1` and `Ad2` to integer before subtracting them).

(iv) Define `Ae` as the dilation of `Ad1` by structuring element `B`. Display `Ae`, which should correspond to Figure 9.11(e).

(v) Define `Af1` as the erosion of `Ae` by structuring element `B`. Display `Af1`, which should correspond to Figure 9.11(f). We know that the erosion of a dilation is equivalent to a closing. Define `Af2` as the morphological closing of `Ad1` by structuring element `B`, i.e., `Af2=imclose(Ad1,B)` (`Af2=morphology.binary_closing(Ad1,B)`). Display image `Af2`, which should also correspond to Figure 9.11(f). Prove that `Af1` and `Af2` are equivalent by computing the maximum of the absolute difference between `Af1` and `Af2` and printing this value to the command window.

(b) **Introductory morphology II**

This problem will introduce you to more complex morphological operations possible using reconstruction operations. You will replicate the results of Figures 9.31, 9.33, and 9.34 (9.29, 9.31, and 9.32, respectively in the third edition of the book).

(i) Download Figure 9.31(a) (available on canvas as `Fig0931(a).tif`), read it into variable `A` and display the image. Define structuring element `B` as defined in the caption for Figure 9.31. The option `'rectangle'` for command `strel` (the command `skimage.morphology.rectangle`) will be helpful here. Erode image `A` with structuring element `B`, call the result `A_eroded`, and display `A_eroded` which should correspond to Figure 9.31(b).

(ii) Dilate `A_eroded` with structuring element `B`, call the result `A_open`, and display `A_open` which should correspond to Figure 9.31(c).

(iii) Use `A_eroded` as a seed image (i.e., the pixels which belong to the objects of interest) and `A` as the mask image (i.e., the pixels which define the maximum extent the objects are allowed to occupy) and replicate the results in Figure 9.31(d). The command `imreconstruct` (`skimage.morphology.reconstruction` with option `method='dilation'`) will be helpful here.

(iv) Fill the holes in image `A` to replicate the results of Figure 9.33(d). The command `imfill` with option `'holes'` (`skimage.morphology.remove_small_holes`) will be useful here.

(v) Remove all connected components in image `A` that touch the image border to replicate the results on Figure 9.34(b). The command `imclearborder` (`skimage.segmentation.clear_border`) will be useful here.

(vi) Remove all connected components in image `A` with less than 100 pixels and display the resulting image. The command `bwareaopen` (`skimage.morphology.remove_small_objects`) will be useful here. (The command `skimage.morphology.remove_small_objects` wants a label image as input, so you will need to use `skimage.measure.label` first. It will also provide a label image as output, which you can convert to a binary image by thresholding >0.) Which portions of the image are less than 100 pixels?

(c) **Fingerprint dataset preprocessing.**

Since the National Institute of Standards and Technology (NIST) fingerprint database doesn't appear to be currently available, we will be using a dataset from the Fingerprint Verification Competition (FVC). As provided, the dataset is `.tif` images which, for some reason, python does not like. As such, I have converted the dataset to .png format for you which is available from canvas (`DB1_B_png.zip`). This dataset consists of 8 grayscale images of a fingerprint from 10 individuals, i.e., 80 images total.

- Generate binary (segmented) fingerprint images. You may use any combination of image segmentation methods and morphological methods. It is your job to develop a method that can result in well-separated fingerprint ridges. You may judge the adequate segmentation qualitatively by eye. Describe your resulting segmentation algorithm and why you chose that method. As appropriate, illustrate intermediate steps in your processing (e.g., if you first thresholded the image and then eroded the image, display the thresholded image and the eroded image) on image `101_1.png`.
- Loop over the 80 images and generate binary fingerprint images for each. Display the resulting binary image for the first image for each individual, i.e., `10*_1.png`.

(d) **Minutiae detection.**

This and part (e) will follow the general process of the Farina et al. 1999 paper (available on canvas as `farina1999.pdf`). We will skip some steps and replace some of the steps with alternatives.

(i) Call your segmented image from the previous step I_bin and compute the skeleton `I_skel` of the image using `I_skel=bwmorph(I_bin,'skel')` (`I_skel=skimage.morphology.medial_axis(I_bin)`). Display the original `101_1.png` grayscale image `I`, the corresponding segmented image `I_bin`, and the skeleton `I_skel`.

(ii) Write a function `I_code=code_branches(I_skel)` which performs the pixel codification step as described in Section 2.1 of `farina1999.pdf`. In particular, given the skeletonized input image

I_skel, your code_branches function should assign each pixel on the skeleton an integer corresponding to how many outgoing branches originate from the current pixel. You may NOT use the 'branch' option in bwmorph. You may assume that the number of branches corresponds to the number of neighbors. Thus (ideally), a pixel at a ridge end will have a value 1 in I_code, a pixel along a ridge will have a value 2 in I_code, and a pixel at a bifurcation will have a value 3 in I_code; you may have other configurations (e.g., crosses that end up with more than 3 neighbors).

(iii) For the image 101_1.png, how many pixels are in the skeleton? How many pixels correspond to ridge ends, i.e., I_code==1? How many pixels correspond to being along a ridge, I_code==2? How many pixels correspond to bifurcations, I_code==3? How many pixels correspond to other configurations, I_code>3? What is the maximum number of branches detected in I_code?

(iv) Display the original grayscale image I. On top of that image, plot all the points correspond to ridge ends as red pluses, the bifurcations as green x's, and other configurations (I_code>3) as blue dots. The command find (np.where) may be helpful here. Also—it might be helpful as an intermediate step to plot the ridge lines (I_code==2) as a sanity check that you are plotting things in the right coordinates. Does it seem that you have false minutiae detections? Do those false minutiae occur in specific parts of the image more than others? Do you appear to have correctly detected actual minutiae?

(e) **Minutiae filtering.**

(i) Define an image I_branches to be labeled individual branches. One way to do this is to remove all pixels in I_code that are any sort of junction (not endpoints or ridge points)--what remains are branches. Use bwlabel (skimage.measure.label) to label each branch. How many branches do you have? Display I_branches.

(ii) Remove all branches that correspond to configurations with I_code>3 and generate a new coded image I_code3 and labeled image I_branches3. (This is predicated on the assumption that you have very few of these configurations compared to bifurcations and endpoints. If not—you need to clean up your skeleton image which probably means you need to clean up your segmentation.) How many branches are left? Display I_branches3. If you determine that I_branches3 has removed valid minutiae, you can revert to I_branches and continue with the following instructions.

(iii) Remove all branches with a length <4 pixels and generate a new coded image I_code3a and labeled image I_branches3a. How many branches are left now? Display I_branches3a.

(iv) Remove all branches that correspond to a bridge, resulting in I_code3b and I_branches3b. Using the definition in the farina1999 paper, you will need to determine the directions of each of the three branches at a bifurcation. Some of the following may be helpful: Assuming that variable I_b contains a binary image of just one branch, you can pull off the x- and y-coordinate values using [x,y]=find(I_b) (x=np.where(I_b)[0]; y=np.where(I_b)[1]). You can fit a line to those coordinates using p=polyfit(x,y,1) (p=np.polyfit(x,y,1)), where p will be vector where p(1) (p[0]) is the slope and p(2) (p[1]) is the intercept. If you take an arctangent of the slope, you will have an angle with respect to the x-axis. Display I_branches3b.

(v) Remove all branches that correspond to a spur, resulting in I_code3c and I_branches3c. Using the definition in the farina1999 paper, you will need to determine some threshold t_spur such that any branch outgoing from a bifurcation less than t_spur pixels is a spur and subsequently removed. Display I_branches3c. How many minutiae are left in your image? Repeat the visualization from (d-iv) with your filtered minutiae.

(f) **Minutiae matching.**

This part will follow the general process of the Jiang et al. 2000 paper (available on canvas as jiang2000.pdf) with some modifications.

(i) For each of the K minutia M_k resulting from part (e), define the length-4 feature vector $F_k = [x_k, y_k, \phi_k, t_k]^T$ according to Equation (1) in the jiang2000 paper. For our purposes, x_k is the x-coordinate of the minutia, y_k is the y-coordinate of the minutia, ϕ_k is the average direction of the branches originating from the minutia (use a similar process as for the bridge removal in part (e)), and define $t_k = 1$ for end-points and $t_k = 3$ for bifurcations. Store your results for each image in a $K \times 4$ matrix. Print out the first 10 rows of your feature matrix for image `101_1.png`.

(ii) We will use $l = 3$ neighbors for the definition of the local neighborhood feature. For each minutia M_k, find the three closest minutia (using the x_k and y_k of the feature vectors and a Euclidean distance), and define the length-13 local feature vector $F3_k = [d_{k1}, d_{k2}, d_{k3}, \theta_{k1}, \theta_{k2}, \theta_{k3}, \phi_{k1}, \phi_{k2}, \phi_{k3}, t_k, t_1, t_2, t_3]^T$, similar to that of Equation (6) in the `jiang2000` paper (note that we are not using the ridge count features n_k here). Store your results for each image in a $K \times 13$ matrix. Print out the first 10 rows of your feature matrix for image `101_1.png`.

(iii) Create a function `sl=similarity_level(Fli,Flk)` which computes the similarity level according to equations (7) and (8) in the `jiang2000` paper. Use each of the `i_1.png`, `i=101,...,110` images as the template and the remaining `i_k.png`, `k=2,...,8` images as the input image. You will thus be comparing the feature vectors $F3_i^I$ for 70 images (and the $i = 1,\ldots, K_L$ minutiae in each of those 70 images) to the feature vectors $F3_j^T$ for 10 template images (and the $j = 1,\ldots, K_T$ minutiae in each of those 10 images) and computing the similarity level $sl(i, j)$. For each image pair (I, T), you will be computing a $K_I \times K_T$ matrix of similarity levels `sl` where K_I is the total number of minutiae in image I and K_T is the total number of minutiae in template image T. You may define your `similarity_level` function to accept two feature vectors and output a single similarity level, in which case you will need to loop over $i = 1,\ldots, K_L$ and $j = 1,\ldots, K_T$ and call your `similarity_level` function each time through the loop. Alternatively, you may define your `similarity_level` function to accept two feature matrices and output a similarity level matrix, in which case you will loop over $i = 1,\ldots, K_L$ and $j = 1,\ldots, K_T$ within your `similarity_level` function. For I as the image `101_2.png` and T as the template image `101_1.png`, print out `sl(i,j)` for `i=1:10`, `j=1:10` (`sl[i,j]` for `i=np.arange(0,10)`, `j=np.arange(0,10)`). For I as the image `101_2.png` and T as the template image `101_1.png`, print out the maximum similarity level, along with the values `i` and `j` which correspond to that maximum similarity level.

(iv) Based on the best matching minutiae (those corresponding to the maximum similarity level), compute the global polar coordinate feature vector Fg_k as defined in Equation (10) in the `jiang2000` paper. You will thus be defining a $K \times 3$ matrix for each image, where K is the total number of minutiae in the image. Print out the first 10 rows of your global feature matrix for image `101_1.png`.

(v) Create a function `ml=matching_level(Fgi,Fgk)` which computes the matching certainty level according to equation (11) in the `jiang2000` paper. Use each of the `i_1.png`, `i=101,...,110` images as the template and the remaining `i_k.png`, `k=2,...,8` images as the input image. You will thus be comparing the feature vectors Fg_i^I for 70 images (and the $i = 1,\ldots, K_L$ minutiae in each of those 70 images) to the feature vectors Fg_j^T for 10 template images (and the $j = 1,\ldots, K_T$ minutiae in each of those 10 images) and computing the matching certainty level $ml(i, j)$. For each image pair (I, T), you will be computing a $K_I \times K_T$ matrix of matching certainty levels `ml` where K_I is the total number of minutiae in image I and K_T is the total number of minutiae in template image T. You may define your `matching_level` function to accept two feature vectors and output a single matching certainty level, in which case you will need to loop over $i = 1,\ldots, K_L$ and $j = 1,\ldots, K_T$ and call your `matching_level` function each time through the loop. Alternatively, you may define your `matching_level` function to accept two feature matrices and output a matching certainty level matrix, in which case you will loop over $i = 1,\ldots, K_L$ and $j = 1,\ldots, K_T$ within your `matching_level` function. Be sure to carefully read the statement about avoiding double counting of minutiae (this statement is in the paragraph between equations (11) and (12) in the `jiang_2000` paper). Use this matching certainty level matrix `ml` to compute the overall matching score `Ms` according to equation (12) in the `jiang2000` paper. Using template image `101_1.png`, compute and print the matching score `Ms` for each of `i_2.png` for `i=101,...,110`.

(g) **Fingerprint verification.**

(i) Create a 10×10 confusion matrix `C`. The (i, j)-th entry in this matrix is the number of times you determined an input image from individual i as belonging to individual j. Thus, if you got 100% accuracy, you would have a solely diagonal matrix. You will determine which individual your input fingerprint belongs to by choosing the identity associated with the maximum matching score `Ms` across the 10 template images. So if image `105_2.png` has the largest matching score `Ms` with template `101_8.png`, you increment the value in `C(2,8)` (`C[1,7]`) (Note that the largest value you should have in your confusion matrix is 7). Display your confusion matrix `C`.

(ii) The overall accuracy is the sum of the diagonal elements divided by the sum of the whole confusion matrix (70 in this case). Compute and display your accuracy to at least 3 decimal points (e.g., 0.706). Random guessing would give you an accuracy of 10%. How does your accuracy compare to random guessing? What do you think are the sources of any errors in your fingerprint verification? Are there specific individuals who are more commonly mistaken for each other or are your errors relatively evenly distributed?

I will give extra credit as follows:

- 5% extra credit for the highest accuracy, 4% for the second highest, 3% for the third highest, 2% for the fourth highest, and 1% for the fifth highest.
- In the case of a tie, multiple students will receive the same amount of extra credit, but I reserve the right to limit the total students receiving extra credit to 5 total

EE588 Advanced Image Processing
Project #2 Morphology
(due Monday, Oct. 8, 2018, 10:30:00 am)

Confidence (number 0 to 100% about your confidence in your performance on this project and optional statement about areas where you are particularly confidence or not):

Difficulty (number 0 to 100%):

Time Spent (hours):

Problem	**Points**
(a) Intro Morphology I	/5
(b) Intro Morphology II	/5
(c) Dataset preprocessing	/10
(d) Minutiae detection	/20
(e) Minutiae filtering	/25
(f) Minutiae matching	/25
(g) Fingerprint verification	/10
TOTAL	/100