Prof. Raghuveer Parthasarathy
University of Oregon; Fall 2024

Physics 410/510 Image Analysis: Homework 8

Due date: Wednesday November 20 by 11:59 pm, submitted through Canvas. You'll see in "Assignments" a place to submit a **PDF**.

Readings: None
Note: This is intended to be a short assignment! Problems 2-5 are *very* short – if you don't find this to be the case, let me know; perhaps an instruction is unclear. Even Problem 1 isn't long.

1 Spot removal. (8 pts.) Download the file "Lichtenstein_imageDuplicator_1963.png" – a painting called "Image Duplicator" by Roy Lichtenstein (1963). (If you're unfamiliar with Roy Lichtenstein's comic-strip-inspired work, see https://en.wikipedia.org/wiki/Roy_Lichtenstein or Google him.) Also download the gray version, which I made by averaging the color channels: "Lichtenstein_imageDuplicator_1963_gray.png". Note the large number of dots! First, play around with the morphological operations of dilation, erosion, opening, and closing, applied either to the entire color image or just one of the color channels. See the note at the end of the assignment about toolboxes. Your task: Remove the dots on the man's face! (Leaving the rest of the image as intact as possible. Note that perfection is impossible.)

(a) (8 pts. 410 students, 6 pts. 510 students) First, remove the dots as best you can with the grayscale version. For now, don't worry if you also disturb the dots in the eyes or eyebrows, or if the lettering is messed up. Can you do this in one line of code? Write the relevant lines of code and also show your output image.

(b) (2 pts.) **[510 students]** Now consider the color image. See if you can figure out a way to remove *only* the red dots. This takes some thinking about what's different between the different color channels. I'm only assigning this 2 points not because it's easy (it isn't!), but because it's not important.

2 Segmentation by thresholding (4 pts.) Before trying out watershed segmentation, let's revisit thresholding. In the following problems, we'll look at an image of yeast colonies on a petri dish, "dsc_0357_gray.png" (source: https://sciencebrewer.wordpress.com/2012/04/). The *overall* goal will be to write a function that, with a few user-input parameters, returns the location of each colony, its size (pixels), and eccentricity (which may indicate errors in identification of colonies).

(a) (3 pts.) Apply whatever filtering seems appropriate to the yeast colony image. Keep in mind both high- and low- pass filtering. (Recall what's useful about each. As always, be careful about intensity ranges!) We'll use this filtered image now, and in later problems. Threshold to make a binary image that highlights the colonies. (Colonies = 1, background = 0.) Submit the resulting image – we'll call this array `im_bw` – and explain what filtering you did.

(b) (1 pt.) Groups of connected "1" pixels are called regions. We can get various properties of all the regions in a binary image using standard tools noted in class (Nov. 14). This is quite boring; I've written the code for you. Run the following:

```
from skimage import measure as skmeasure
label_img = skmeasure.label(im_bw)  # label connected pixels
stats = skmeasure.regionprops(label_img)  # stats on the object
```

Now, using the `stats` list, make a scatter plot of Eccentricity vs. Area; use a logarithmic axis for area. This is boring, and I want the assignment to be short, so I have written the code for you, except for making the plot. Submit the plot. Yes, there is nothing else you need to do!

```
# Region properties
label_img = skmeasure.label(im_bw)
stats = skmeasure.regionprops(label_img)

# Plot properties of threshold-segmented regions
areas = [stats[j]['Area'] for j in range(len(stats))]
eccentricities = [stats[j]['Eccentricity'] for j in range(len(stats))]
```

3 Gradient magnitude (3 pts.) Convolve the filtered (not thresholded) yeast colony image with 5x5 Sobel filters (shown below) to get the gradients in x and y. Then calculate, display, and submit the gradient magnitude (i.e. sqrt($G_x^2 + G_y^2$)) image.

5 px Sobel operator, x:					5 px Sobel operator, y:				
2	1	0	-1	-2	2	3	4	3	2
3	2	0	-2	-3	1	2	3	2	1
4	3	0	-3	-4	0	0	0	0	0
3	2	0	-2	-3	-1	-2	-3	-2	-1
2	1	0	-1	-2	-2	-3	-4	-3	-2

4 Watershed segmentation without markers. (1 pt.) Apply watershed segmentation to the gradient magnitude image ("`im_gradmag`"). This is one line of code – you don't have to code the algorithm yourself! In Python:

```
im_watershed = watershed(im_gradmag)
```

Display this. To see something that isn't completely featureless, use a "prism" colormap, as shown:

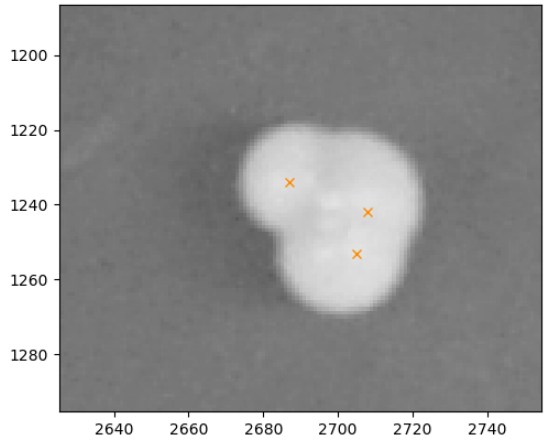
```
plt.figure()
plt.imshow(im_watershed, cmap='prism')
```

Submit the segmented image. It should look *terrible*. (Notice that the colored bands aren't straight lines – it's not just noise, but it's not good!) The segmentation may take tens of seconds to run.

5 Watershed segmentation with markers (7 pts.)

(a) (3 pts.) Make a binary array that's 1 at the local maxima of the filtered (not thresholded) image of yeast colonies from Exercise 2a, and 0 elsewhere. There are many ways to do this; I suggest simply assessing whether the image equals its dilated image. (If you want, you can look into `skimage.feature.peak_local_max`, which does in fact use the same method.) You should find many maxima. (If you show this "image," don't worry that it looks black. Zoom in!) Combine these with the thresholded image to only keep the "bright" maxima. How can you combine these? Think about what multiplying by your thresholded image (`im_bw`) does! Display the locations of the bright local maxima as "x"s on top of the original image, and submit this. *Hint:* use `numpy.where` to get the positions at which to put the x's. (You can easily plot points on an image. In Python, for example, `plt.plot(200,400,marker='x',color=darkorange)` puts an "x" at (200, 400).)

You should get something that, zoomed in, looks like



(b) (3 pts.) Apply watershed segmentation to the gradient magnitude array from earlier, but using the above-threshold local maxima from (a) as markers. See if colonies that touch each other can be recognized as separate objects! Optional: try different markers if you like, such as dilating the maxima, eroding the binary thresholded image, etc. Describe what you find and what markers you used and submit the segmented image. (See the note at the end for how to impose markers.)

(c) (1 pt.) Run the same code as earlier to get statistics on the various regions you've found from segmentation and make a plot of eccentricity vs. Area. You'll find that there are still too many objects, but hopefully you see some structure in your plot. Submit your plot, and comment on whether you could use this to get rid of objects whose properties imply that they're not colonies (e.g. high eccentricity). (You don't actually have to get rid of objects.)

6 [Extra Credit] Morphological Reconstruction. (3 pts.) I briefly put up a slide about morphological reconstruction but I didn't illustrate what it does. Let's explore this here. This operation is a dilation that's constrained to lie under a "mask" that limits its intensity. Load the erosion and reconstruction operators from scikitimage:

```
from skimage.morphology import (erosion, reconstruction)
```

(In MATLAB, look up the equivalent things.) Load the image "MobyDickPar1.png", the first paragraph of Herman Melville's *Moby Dick*. Let's try to erase all the letters *except* those with large vertical strokes. First: Apply an erosion with a structuring element that's a line that's long enough that letters like "a" , "m", and "e" disappear completely, but pieces of letters like "l" and "d" remain. *Hint:* Choose one of the following as the structuring element, with an appropriate choice of *N*.

```
ste = np.ones((N,1))
ste = np.ones((1,N))
ste = np.ones((N,N))
```

(Here we've imported numpy as np.) From the eroded image ("`im_erode`") perform morphological reconstruction using the original image ("im") as a mask:

```
im_recon = reconstruction(im_eroded, im)
```

What do you find? Describe the result, submit the eroded and the reconstructed images, and write what structuring element you used.
(In case you want to read more, there's also a good description of morphological reconstruction at https://www.mathworks.com/help/images/understanding-morphological-reconstruction.html .)

Implementing morphological operations in Python

As noted in class, morphological operations are in the scikitimage package:

```
from skimage.morphology import (erosion, dilation, disk, closing)
```

etc.
A neighborhood that's a disk of radius 15 would be
disk of radius 4

```
ste = disk(15)
```

Erosion with that "structuring element" on image "im" would be

```
im_erode = erosion(im, ste)
```

Implementing morphological operations in MATLAB

MATLAB's help documents on morphological operations (`imclose`, etc.), all part of the Image Processing Toolbox, are excellent! Use them.

Watershed segmentation in Python

Use `skimage.segmentation.watershed` ; see https://scikit-image.org/docs/stable/api/skimage.segmentation.html#skimage.segmentation.watershed , and especially

https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_marked_watershed.html, which shows how to use markers.
In brief, to segment the topography "`im_gradmag`"

```
from skimage.segmentation import watershed
labels = watershed(im_gradmag)
```

Or, using a binary image of markers based on "`markerBWimage`":

```
markers = skmeasure.label(im_max)
labels = watershed(im_gradmag, markers)
```

The second link also shows how to display the segmentation labels.

Watershed segmentation in MATLAB

In **MATLAB**, this is simply `watershed`. Without markers:

```
% Watershed segmentation of "im_gradmag")
watershed_im = watershed(im_gradmag);
```

To use markers, first impose your markers as minima using `imimposemin`, shown here for a binary image of markers "markerBWimage":

```
topo_with_min = imimposemin(im_gradmag, markerBWimage);
```

Then segment this with `watershed` as usual.