NENS 230: Analysis Techniques for Neuroscience using MATLAB
Fall 2011

Assignment Five: A Simple Cell Counting Tool
DUE DATE: 9am, Monday, October 31st

Please submit this assignment to nens230@gmail.com, with the subject line "hw5". A detailed assignment description is below, but in short, you will submit everything in your Assignment5 directory. Specifically we are looking for your *CellCount.m* program as well as any helper functions you wrote that your program calls. Also include a figure *CellCountImages.fig* generated by the tool after you've used it and the .txt results file it generates, *CellCountResults.txt* . You should send these files in a single (zipped) folder attached to your email called hw5_SUID where SUID is your SuNET ID (e.g. sstavisk).

In this assignment you will draw upon what you've learned about importing images, manipulating the three-dimensional matrices that define an image, and interactive program functions. You will write a simple cell counting tool that lets the user load a microscopy image, isolate a single color channel, apply an intensity threshold, and graphically select cells that are to be counted. This program will then save the resulting labeled microscopy image as a .fig and will also generate a text file summarizing how many cells were identified and where in the image they are located.

The goal of this assignment is to show that you can leverage the programming fundamentals you've previously learned along with the interactive user interface functions you've just learned to make tools that allow you to perform a desired analyses much faster and easier than by repeatedly editing and then running a script. This assignment also reinforces the plotting skills from previous assignments and will let you practice manipulating images as an underlying three-dimensional matrix.

Brief neuroscience background

Rapid advances in microscopy techniques in recent decades has allowed increasingly sophisticated investigation of the development and structure of neural tissue and detailed understanding of the wiring of neural circuits. The image that you will be working with is one of a series of vertical (z-slice) confocal microscopy images of the inner plexiform layer (IPL) of the retina of a mouse. This retina has been stained with a red-fluorescing antibody against ChAT (choline acetyltransferase), which is known to express strongly in starburst amacrine cells.

These cells form dendritic trees in two layers of the IPL and are involved in implementing direction selectivity. Cell locations and cell counts obtained by analyzing a confocal microscopy image are basic but important initial characterizations that would form part of the story answering myriad questions. For example, the coverage factor of a given retinal cell type (i.e. how densely a particular cell type's receptive field tiles the visual field) can provide hints as to whether this cell can independently provide a complete description of one type of visual feature.

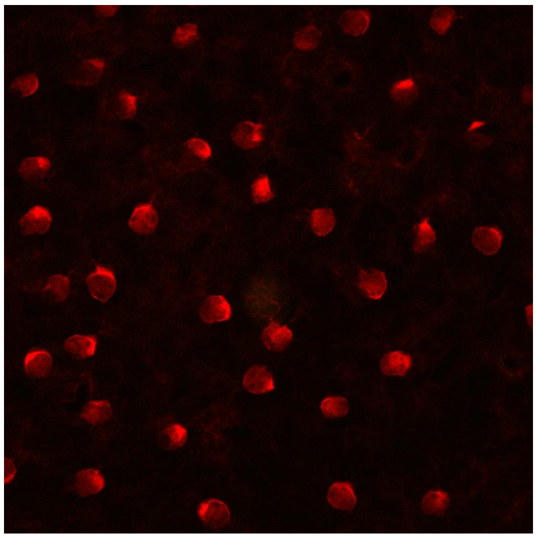
Figure 1: Confocal image of ChAT-stained slice of the inner plexiform layer of a mouse retina. You will run this image through the interactive program you will write.

Detailed Instructions

You are going to write a program that acts as a tool to allow the user to do simple image processing of a microscopy image and then label the locations of cells of interest. It will then output a text file listing the location of each cell, and save a figure of the original image side-by-side with the labeled, processed image.

Copy the Assignment 5 directory to your MATLAB directory. It contains the image that your tool will process (*chatzslice.jpg)* as well as a helper function *writeCountsFile.m* which will create the output text file using data which your program feeds to it.

We've included a file called ExampleCellCount.p which demonstrates what the tool which you will write might look like. Such ***p-code*** is an .m file that has been partially compiled so that the code is not readable. p-code is useful for sharing code with collaborators who you don't trust with your "secret sauce", or in this case, for showing what a program does without revealing how it works.

You can run ExampleCellCount.p just like an .m script and see what it does; the program that you will write should have the same functionality, but it need not look identical to this example. Feel free to embellish and customize your tool to behave the way *you* want your cell counter to work.

Specifically, your program should do the following:

1. Asks the user what image file to load. When you use the tool, you should select *chatzcell.jpg*.
2. Load the image.
3. Display the image.
4. Ask the user what individual color channel (red, green, or blue) they want to look at.
5. Display just that channel next to the original image.
6. Ask the user what intensity threshold cutoff they want to apply to the single-color image. Set the pixel values of any part of the image that is below the threshold to zero. Allow this step to be done as many times as the user wants before they are satisfied, and give them a way to inform the program that they are satisfied and that it can continue to step 8.
   HINT: Don't overwrite the initial single-color image, or else you won't be able to undo the thresholding if the user wants to change the threshold value. Make a copy of the monochrome image matrix before manipulating it.
7. Whenever the user sets a threshold, display the thresholded single-channel image. Label this image with the current threshold and which color channel is being displayed.
8. Use **ginput** in a while loop to allow the user to click on the location of cells that they want counted. Whenever a user clicks, you should save this location to the appropriate element of a structure *cells* with fields .x and .y which store the (x,y) coordinates of where the user decided the cell was (the user should click on the center of a cell). *cells.x* and *cells.y* will thus grow in length with each click.
9. Whenever the user clicks on the image, in addition to storing the coordinates, annotate this location in the image with the number of the cell (i.e. the first clicked location is labeled "1", the second "2", and so on.
10. When the user is done labeling cells, allow them to break out of this loop.
11. Ask the user to select the filename and location of where they want your program to write a .txt file with the cell counts and cell positions.
12. Use the provided writeCountsFile.m function to create this .txt file. Take a look at how this function works to learn how to create .txt. files; it is easy and very useful.
13. Finally, ask the user where they would like to save the current state of the figure showing the original and processed/labeled images. Then save this figure as a .fig in this location.

Throughout this, try to make the user-interface intuitive by labeling axes, providing output to inform the user of what is happening, and giving your user interface GUIs/prompts useful instruction text.

When your tool is ready, use it to process *chatzslice.jpg* and generate a *CellCountResults.txt* and *CellCountImages.fig.* You don't actually need to go through and label each and every cell, but label enough to demonstrate that your tool is usable. We will actually run your program when grading the assignment.

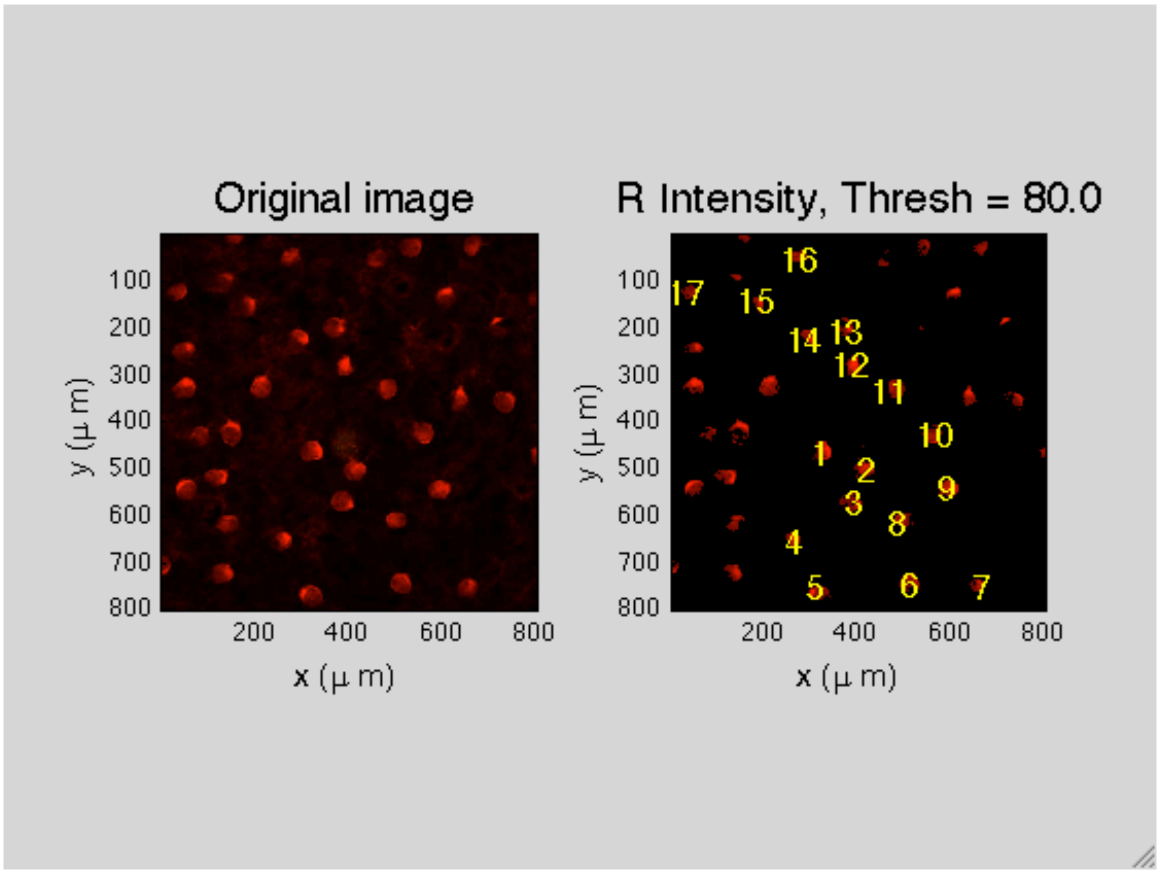

Figure 2: An example of what your program might look like during the cell-counting step.

Extra Credit

If you are interested, you might try making an automated (or partially-automated) cell counting program. The following pages have nice tutorials of how this could work (but they rely on functions in the Image Processing Toolbox, which you may or may not have):
http://blogs.mathworks.com/steve/2006/06/02/cell-segmentation/
http://www.mathworks.com/products/demos/image/watershed/ipexwatershed.html

The function bwlabel (in Image Processing Toolbox) might also be useful.

This is a difficult machine vision problem, although very doable for the relatively clear image we've provided. This could also be a good final project.