Robust Automated Algorithm for Counting Mouse Axions

Anh Tran
Department of Electrical Engineering
Case Western Reserve University, Cleveland, Ohio
Email: anh.tran@case.edu

Abstract:

This paper presents a robust technique for counting axion cells within a digital image of a mouse neuron bundle. The method involves employing Matlab to transform the color image into grayscale, performing regional histogram equalization, thresholding the image, and finally detecting cell boundary using morphological technique. This algorithm detects and counts cell boundaries with 92% accuracy.

Key words:
Mouse Neurons, Cell Counting, Computer Vision, Background Filtering, Regional Histogram Equalization, Thresholding, Image Segmentation, Morphology, Boundary Detection.

Introduction:

Medical image analysis has become increasingly important in the clinical community. Researchers are interested in analyzing cell images for classification of types and sizes to extract valuable information from the sample. Manual cell counting rests on the shoulders of the ophthalmologist who looks at the image and carefully classifies which is and which is not a cell. Such procedure is time inefficient and very tedious. [1] Therefore, a computer aided program designed to accurately count the cells is absolutely necessary to reduce analysis time and improves repeatability. Two common approaches involve pattern recognition and image segmentation. [1] [2] [3] [4] However, the difficulty in designing the automated program surfaces when the entire image is blurry or tiny cell within the image overlaps onto each other. The human eyes can easily detect and count these cells, and such knowledge is very hard to duplicate for computer.

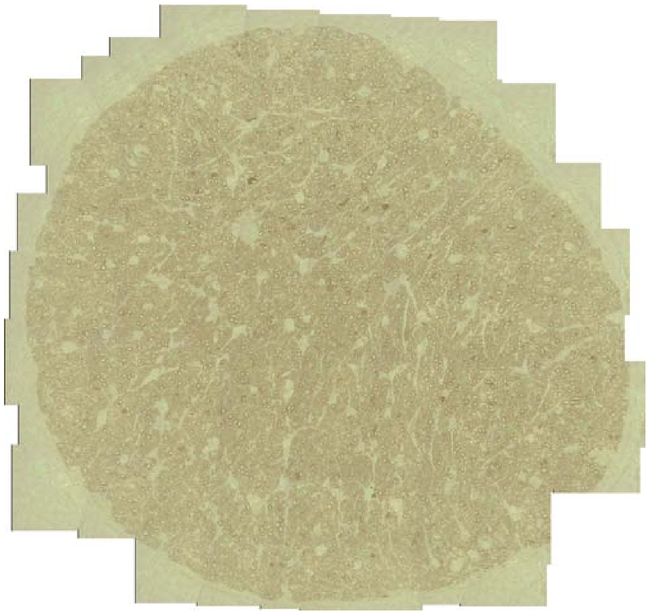
Figure 1: Mouse Neuron Bundle
7608x7244 pixels

Method:

Taking the above considerations into account, the automated program discussed in this paper will focus on cells in **figure 1** that have well defined boundary. The cells that are cut in half at the image’s edge and those whose boundaries that are below the threshold will be discarded. Because the image above is very large (~55million pixels), computation with the entire image in Matlab causes significant memory problem. Therefore, it will be divided into four equal sub-images for independent processing.
A smaller section of the master image (**Figure 2**) will be analyzed first to ensure that the correct number of cells have been

counted prior to processing of the four larger sub-images.

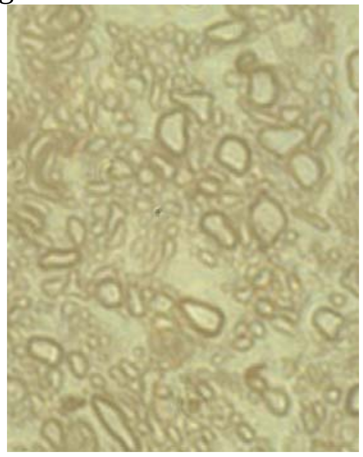

Figure 2

The image (**Fig 2**) in its original form is RGB. Converting it to its grayscale helps to reduce computational complexity.

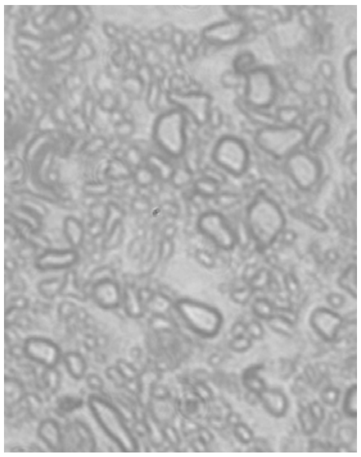

Figure 3: Grayscale

1. Adaptive Region Histogram Equalization

Matlab provides a function called adapthisteq to enhance the contrast of the grayscale image using contrast-limited adaptive histogram equalization (CLAHE). Similar to histogram equalization which operates on the entire image, CLAHE performs the same operation but on smaller regions called tiles. This method significantly brings out boundaries and edges of blurry cells as shown in **Figure 4.**

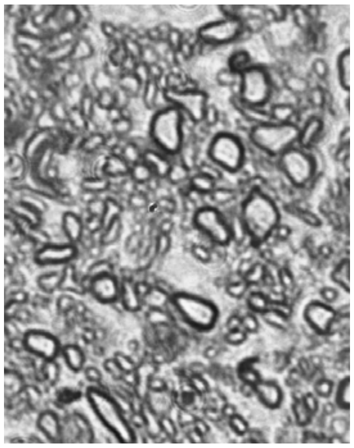

Figure 4: Adaptive Histogram Equalization

2. Reducing Gray Levels

Grayscale images have full range from 0 to 255 levels of shades of gray. If the number of levels reduces to 16, neighborhoods of pixels that have similar values will be grouped together. This approach allows choosing a threshold value much simpler.

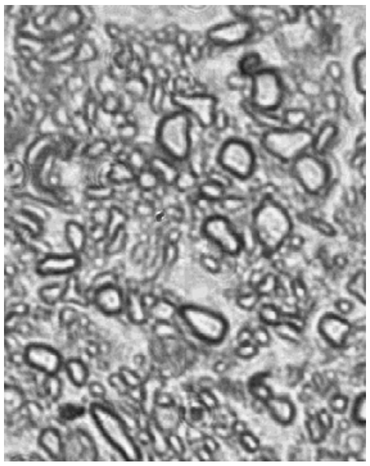

Figure 5: 16 Gray Levels

3. Thresholding to Eliminate Background

Figure 5 indicates that the lowest pixel value that still makes up a cell boundary is 160. Therefore thresholding at this value makes any pixel higher than 160 to be 255, and those that are lower than 160 to be 0. As the results, only the cells boundaries remain in the image as shown in **Figure 6.**

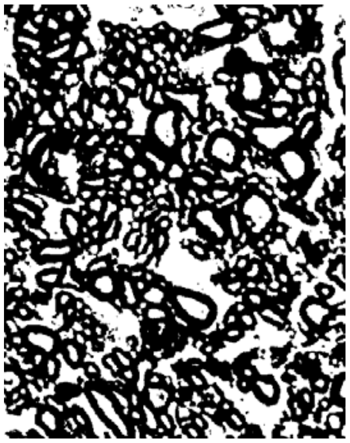
Figure 6: Threshold at 160

4. Image Segmentation using Morphology
Matlab has a very effective command called bwmorph to thin the objects to their boundaries.

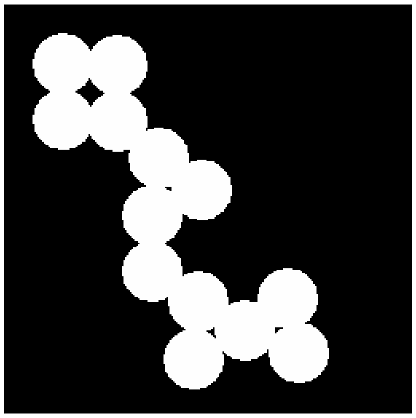
Figure 7: Original Structure

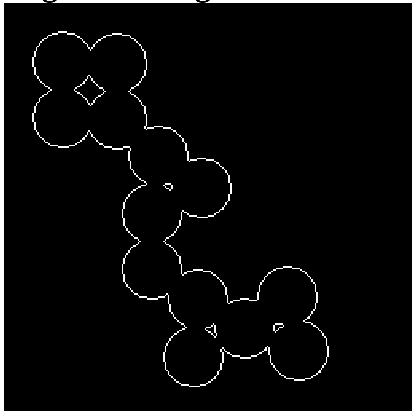
Figure 8: Thinned Structure

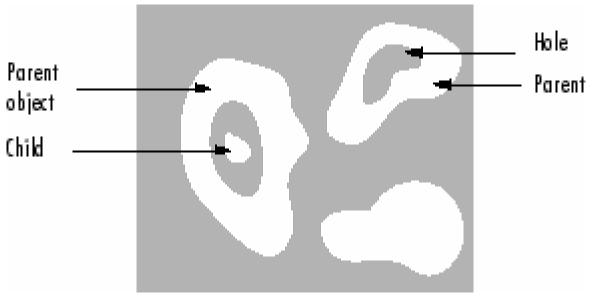

Figure 9: Diagram from Matlab Help

Next, bwboundaries function traces the exterior boundaries of objects and returns the number of objects found.
Therefore, taking the complement of **Figure 6** and performs the bwmorph function followed by the bwboundaries operator, Matlab returns the boundaries of 260 cells detected in the image.

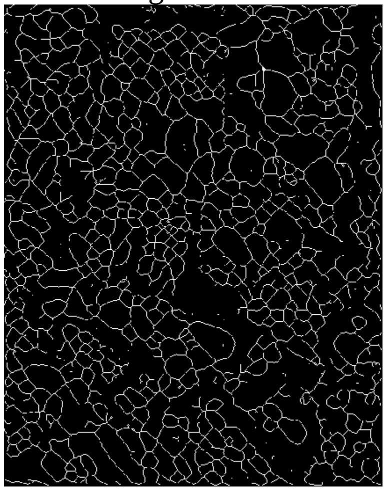
Figure 10: Cell Boundaries

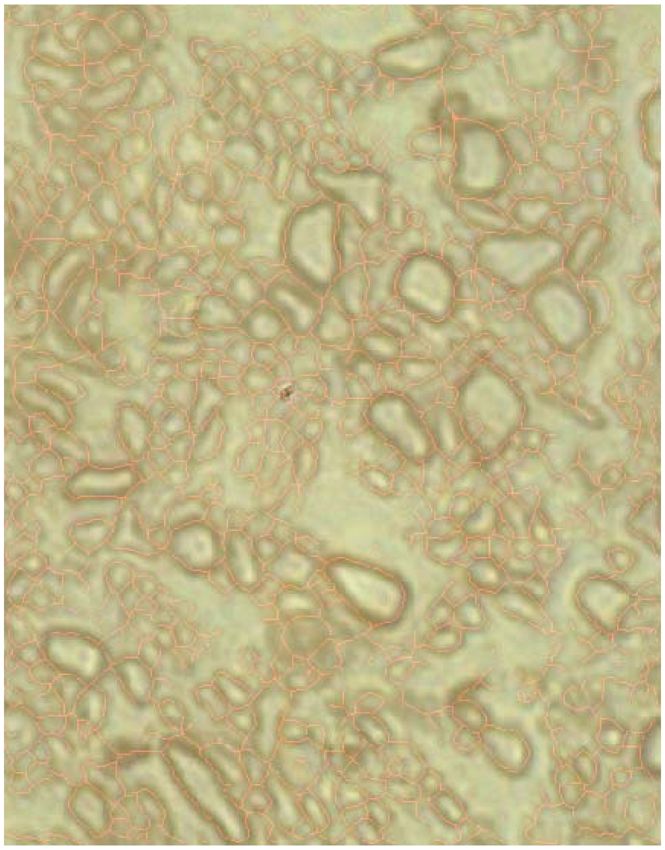
Figure 11: Original Image with Detected Cell Boundaries

For the majority of the cells above, each has its own boundary to indicate that it has been accounted for in the total numbers of cells. However, small regions enclosed inside a cluster of cells are falsely detected as a cell. As the result a small margin of error is unavoidable.

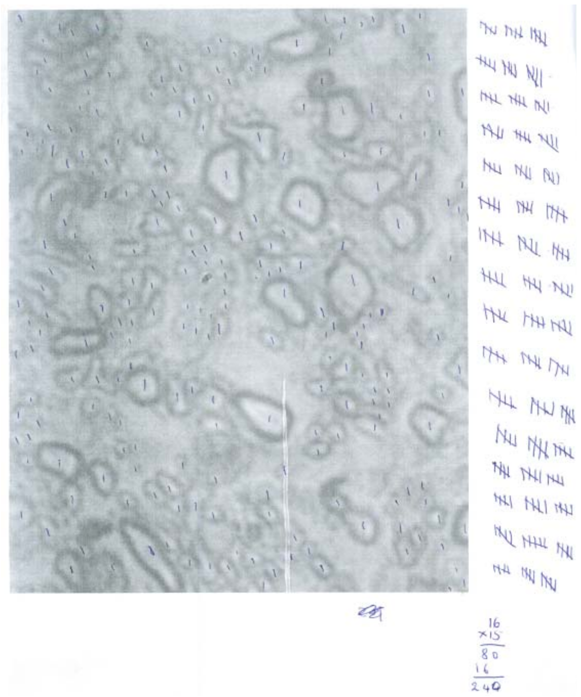

Figure 12: Hand Count (240 Cells)

To determine the accuracy of this algorithm, the image was hand counted resulted in 240 noticeable cells. The algorithm returned 260, which is about 8% error or 92 % accuracy rate.

<u>Processing Large Image:</u>

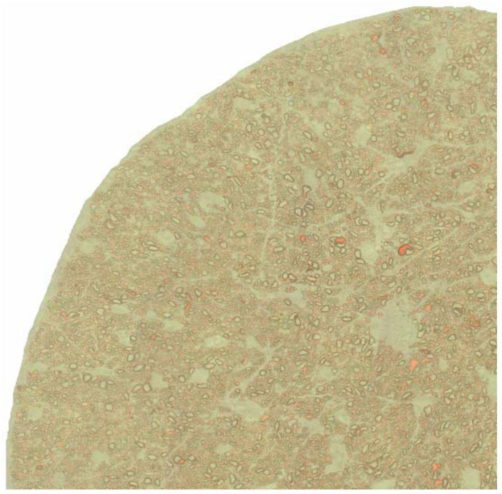
Figure 13: Top Left_9070 Cells

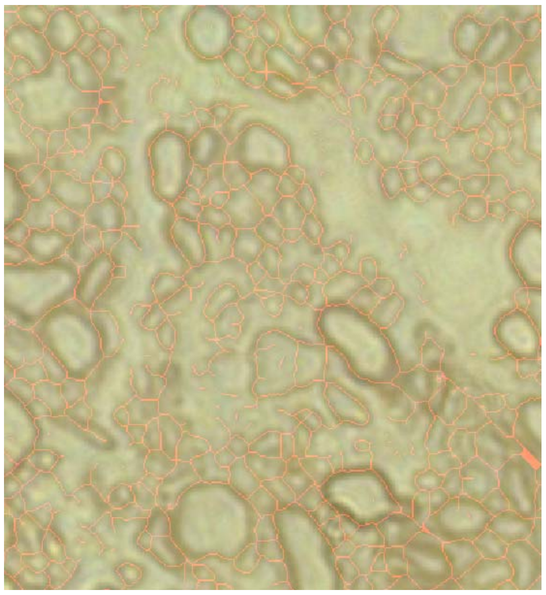
Figure 14: Close up of Figure 13

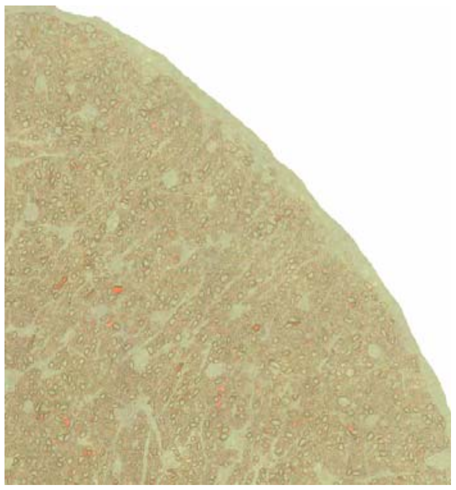
Figure 15: Top Right (9110 Cells)

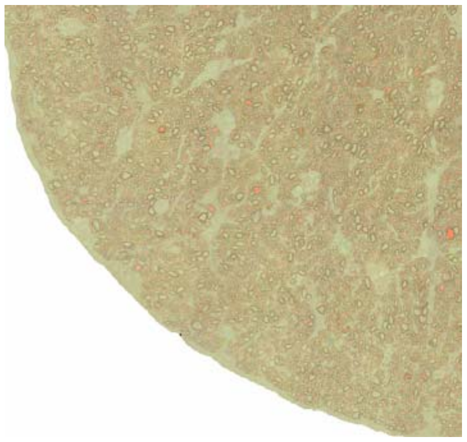
Figure 16: Bottom Left (7756 Cells)

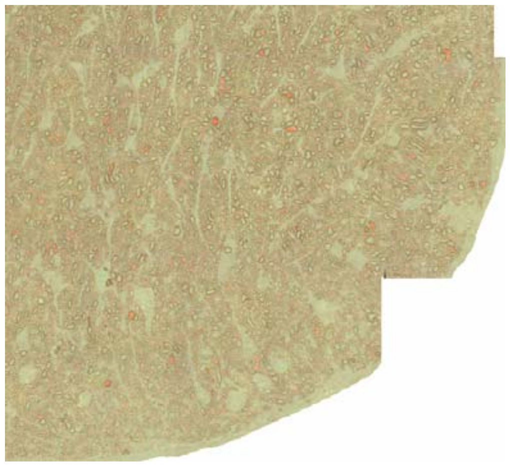
Figure 17: Bottom Right (9540 Cells)

The total number of cells for the entire master image is approximately 35,476 cells. The close up image Figure 14 shows that most of the cells have been accounted for using this algorithm.

<u>Comparison with Gold Standard Images:</u>

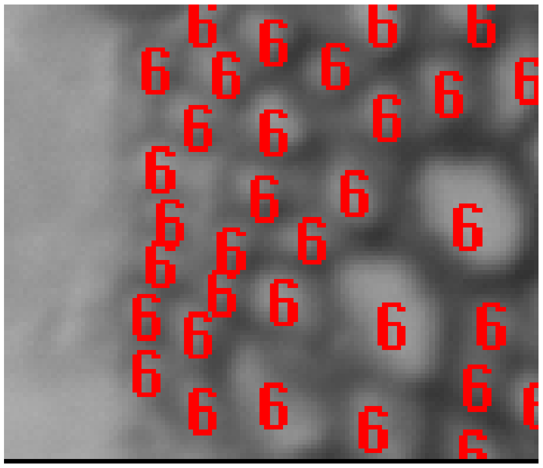
Figure 18: Sample_6.tif (33 cells)
97x113 pixels

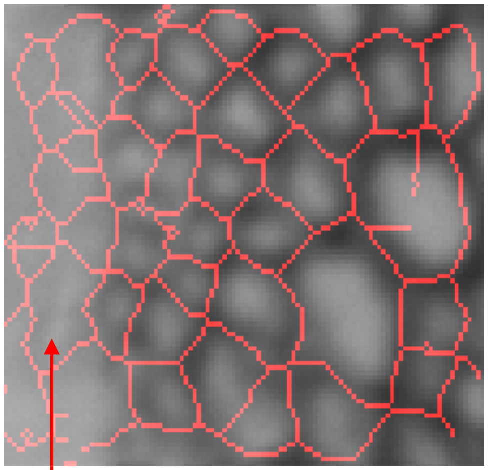
Figure 19: Sample_6 (25 cells)

10 False Cells

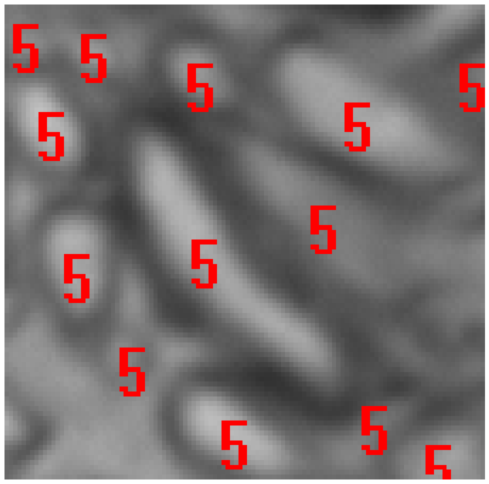
Figure 20: Sample_5 (13 Cells)

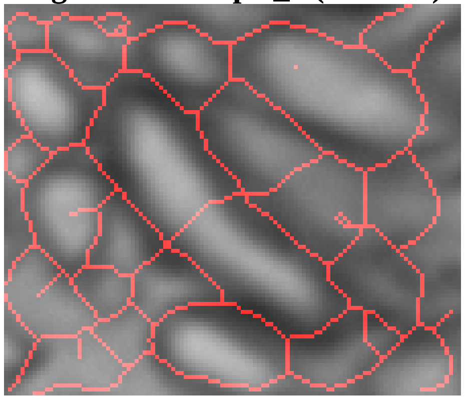
Figure 21: Sample_5 (14 Cells)

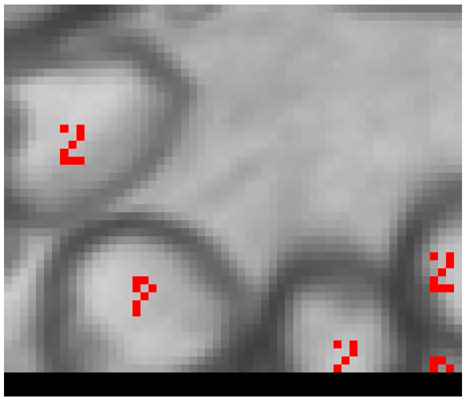
Figure 22: Sample_2_counted.tif (4 Cells)
96x112 pixels

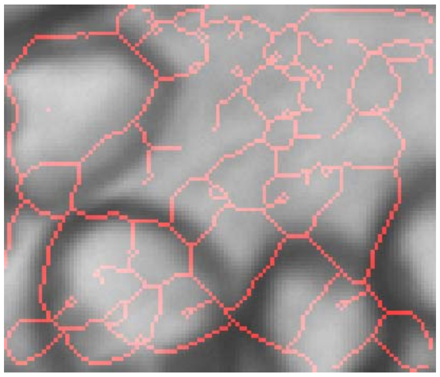

Figure 23: Sample_2 (20 Cells)

Image	**Size**	**Hand Count**	**Automated Count**	**Accuracy**
Fig2	610x480	240	260	92%
Large	7608x7244	--	35,476	~90% ?
Sample 6	97x113	33	25	75%
Sample 5	96x112	13	14	92.3
Sample 2	96x112	4	20	20%

<u>Discussion of results:</u>

Base on the results above, this algorithm only works well if the image is large so that the regional adaptive histogram equalization has a higher distribution of pixels to work with. When the image is too small and there is a large area of background, adapthisteq function might return false edges, which will result in false cells as was shown in **Figure 19, 21** and **23.**

One minor set back that this algorithm encounters is that the area enclosed between cell boundaries will be counted as well. To alleviate this problem, this method needs an extra step which looks at the area distributions of all of the cells. If a cell that has an area that is much higher than the average, then it should be discarded. However, overall, most of the cells in the large images have been accounted for. Also, processing each of the four large images takes only 3 minutes.

<u>Reference:</u>

[1] Foracchia M., Ruggeri A. “Cell Contour Detection in Corneal Endothelium In-Vivo Microscopy.” IEEE: Engineering in Medicine and Biology Society, (2000). Pages 1033 - 1035 vol.2

[2] Horiuchi, T. Akiba, T. Kakui, Y. “Development of a Continuous Imaging System Equipped with Fluorescent Imaging for Classification of Phytoplankton.” IEEE: TECHNO-OCEAN, (2004). Pages 1410 - 1413 Vol.3

[3] Mussio, P. Pietrogrande, M. “Automatic Cell Count in Digital Images of Liver Tissue Sections.” IEEE: Computer-Based Medical Systems, (1991). Pages 153 –160.

[4] Sokkarie, A. Osborne, J. “Object Counting and Sizing.” IEEE: Creative Technology Transfer - A Global Affair (1994). Pages 380 - 382