Automated Axon Counting via Digital Image Processing Techniques in Matlab

Joshua Aylsworth

Department of Electrical Engineering and Computer Science,
Case Western Reserve University, Cleveland, OH

Email: joshua.aylsworth@cwru.edu

Abstract

This paper presents the design of an algorithm that uses various image enhancement and image processing techniques to count axons in the optical nerve of a mouse. The digital image is the cross section of the nerve stemming from the eye in route to the brain. The goal of the algorithm is to be able to accurately detect the axons and give the user a count of the number axons present in the image.

KEYWORDS

Cell Segmentation, Image Segmentation, Axons, Image Processing, Adaptive Thresholding, Watershed, Matlab, Morphological

INTRODUCTION

Determining the number of axons in the optical nerve is largely dependant on being able to correctly determine, among proper regions of interest, the edges of the shapes. One of the challenges faced during this exercise is being able to overcome a low contrast image; or so it would seem. Eddins suggests that the contrast actually doesn't contribute much to the success of segmenting the shapes [2], [4]. Instead, creating a high contrast is merely to aid the user. There can be different approaches to segmenting shapes that aren't uniform. Hodnelad et al uses a method of level set for watershed image segmentation [1]. Watershed seems to be a common method. Yan et al also use the Watershed method to identify cell phase identification with their algorithm [3]. All users seem to do a significant amount of pre processing to any of the images being analyzed. Other typical preprocessing methods include morphological operations including dilations, erosions, openings, etc [3, 5]. The Otsu thresholding technique is another technique that is commonly used. The Otsu can have some drawbacks as although it is designed to minimize variance in a histogram, it can be processing time intensive. Also, as with any global threshold, Otsu doesn't lend itself to doing a very good job where there is a low contrast image. One method to account for this, is to separate your image into different regions and determine an Otsu threshold for each of the regions of your image. Recombining the image following after processing each region independently typically will yield better results. An alternative to breaking your image apart and having to create multiple segments and using the labor-intensive code, one can instead use a method called adaptive thresholding [3]. Adaptive thresholding uses an averaging filter combined with local neighborhood filtering to compare the current data location to the mean of the specified window or neighborhood size. If the user chooses, they can use the local median as the decision point [5,3]. Using a local neighborhood accounts very well for non-uniformities of grayscale intensities within the same image. This eliminates the need for any complex image partitioning. After the image is thresholded, the edges can be found. Or, depending on the technique used, the edges could be found while thresholding, as in the case with the Canny filter. In this case, much like that of Eddins [5], if a good job is done up front in the preprocessing, simply being able to identify the perimeter of the image segments, axons, will serve the same function as identifying an edge. After creating an image that shows where the distinct edges of the axons are, counting is the last remaining task at hand. There are various techniques for this as well, but there are already tools built into Matlab for accomplishing this feat, so it won't be explored in great detail.

AUTOMATED AXON DETECTING ALGORITHM

The algorithm and method of detecting and counting the number of axons present in an image will now be presented. All of the images presented for processing were in a three-layer RGB format. In an effort to work with a single layer, the color image is transformed into a gray scale or intensity image. While this can be done via a basic image processing formula:

$$\frac{R+G+B}{3} \qquad (1)$$

it isn't necessary to complete the transformation. As anyone who is accustomed to using Matlab for image processing purposes knows, there is an extensive toolbox already built in that accomplishes much of the legwork for you – even when building complex algorithms. The real art in creating a good algorithm is knowing how the tools work and determining when to use the right tool and which tool to use. This discussion aside, the actual transformation was performed by employing the *rgb2gray* function. This func-

tion takes an input image, f, and outputs, g, which is the grayscale version of the color f. The algorithm written for determining the proper number of axons uses f_gray as its handle. In order to study some of the basic properties of the image of interest, it is always a good idea to look at the distribution of gray levels. After looking at the distribution, one can determine probabilistic characteristics of the image which opens up doors for lots of different opportunities in the image processing world. While this algorithm didn't use probabilities in the strict sense, probabilities certainly come into play regarding where an edge may be called an edge or the probability that a pixel is thresholded correctly. These cases will certainly be a factor when trying to determine where the axons are in the image. Again, making good use of Matlab's extensive image processing toolbox, one can view the histogram of gray levels in an image by using the *imhist* command. In an image where there may be a significant amount of noise combined with relevant image information, averaging helps to make the noise less dominant. Averaging, or applying a low-pass filter is a common technique used in image processing to exaggerate changes. A preferred method of low- pass filtering is a Gaussian filter. Using the Matlab command *fspecial* and designating the argument of '*gaussian*' one can create a Gaussian low pass filter of size MxN where M is the number of rows and N is the number of columns. Typically, a square matrix is used. In this algorithm a 5x5 square Gaussian low pass filter was used to 'smooth' the image. After smoothing the image, the histogram was examined again to see what kind of impact was made on the distribution of pixels. Many times this can help exaggerate a difference in a bi-modal distribution of gray levels making further image processing quite easier. In other cases where the original was very close to a normal distribution, there is very little impact made on the histogram. At this point in the algorithm, there are many roads to travel. One leads you in the way of morphological gradients, Laplacian gradients or any other method of determining the edges. Matlab has a built in *edge* tool that lets the user select from a half dozen or so different options. The main goal to keep sight of is that the user eventually wants to obtain the edges of all of the axons in the image so that some counting technique can be applied. In an ideal case, the edges should be contain no breaks in them so that there can be formed a complete perimeter. The edges are nearly always in a logical image, white on black to make it easy for the user to interpret where they are. In order to get this binary image, one has to determine the value in the gray level distribution in which every pixel above is white, and on the contrary, every pixel below is black. A very crude way of doing this is by trial and error; most likely starting in the center of the distribution, unless there is an obvious point that stands out. This is not efficient and certainly there has got to be a more advanced method. This is where Otsu's technique could potentially come into play. The aim of Otsu's thresholding technique is to minimize the variation between segments. Otsu's technique is built into Matlab as well and can be exercised by using the *graythresh* command. As is common knowledge among image processors, typically a simple global threshold does not perform very well unless there is an obvious break in the histogram from one distribution to the next. Moving along with this possible technique, it would be advised to partition the image into multiple areas and determine an Otsu threshold value for each of the areas. After doing this, one can threshold the image in a more localized manner. A different approach, the approach that was taken in this algorithm was to use an adaptive threshold. By doing so, the non uniformities can be taken into account throughout the image. Instead of having to partition an image or use a global threshold, the user can choose an MxM square matrix and determine whether they wish to use a median or mean of a local MxM neighborhood to determine a pixel's fate. This method says that if the pixel of interest is below the local median, then it will be a 0, or a black pixel. If above the mean or median, then that pixel will be a 1, or a white pixel [3] [4]. There is a function called *adaptivethreshold* that can be downloaded from the Mathworks website that performs this for the user in Matlab. The output is a black and white logical image. Now this image will require a little bit of cleaning up in order to make it easier to find the proper edges. The first step in making the image more manageable is to fill in all of the holes. Since the shapes of the axons, while not circular, typical resemble some sort of ring, there is always an outside and an inside. The idea is the fill in the in side so that there is one solid white shape instead of a white ring with a black center. Using the command *imfill* and specifying the 'holes' method, this exact task is performed. Next, to remove spurs, there is applied some morphological operations. Using a structuring element of ones sized 5x5, an opening is performed via the *imopen* function in Matlab. An opening is the morphological equivalent of first performing an erosion and then following with a dilation. Now having studied the images of interest, and knowing a little bit about them, it is safe to say that we'd like to remove small pixel formations. The risk here of course is that we may actually remove an axon, but this step helps to eliminate any surviving noise clusters. Using the *bwareaopen* function in Matlab, small white pixel groups are removed successfully. Now that the binary image is cleaned up, one can make use of the *bwperimeter* tool in Matlab. This tool will create a second image that contains only the edges of each of the axons. The last thing that remains is to count the number of perimeters, axons, that are in the image. This action is performed by evoking the *bwlabel* command. Also, in an effort to see how well this algorithm performed in comparison to the original image, one can search mathworks.com for the *imoverlay* tool. Having this tool the image obtained from using the *bwperimeter* tool can be overlaid as any color onto the original image.

RESULTS AND DISCUSSION

Now that the procedural technique of the algorithm has been presented, the results will be shown to see how well it works. There are six different images that were supplied. The complete algorithm results will be displayed fully for one of the best performing cases as well as one of the worst performing cases to display instances where the algorithm worked very well and to exploit some of its shortcomings. As discussed in the previous section, the first order of business is to import the image into Matlab.

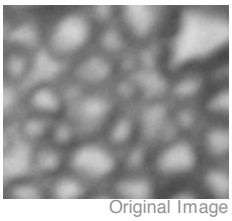

Figure 1 – Sample 4.tif

Figure one is the imported image, original and not enhanced. After importing the image into Matlab, the next step was to transform it to a gray scale image. There isn't much difference visible to the user, but it does turn into a single layer intensity image.

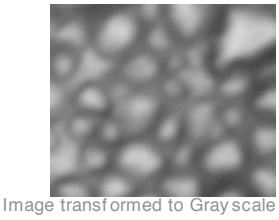

Figure 2 – Single layer intensity image

Nothing is too apparent in terms of differences that jump out at the viewer. One note of cosmetics is the description immediately below each image. This is an *imcredit* function written by Eddins [2] [4].

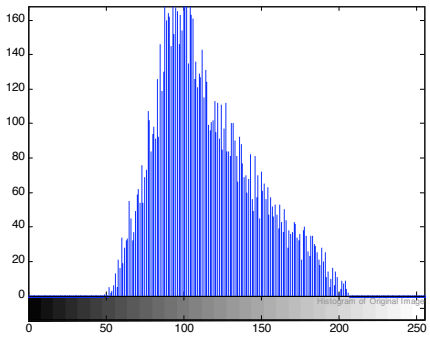

Figure 3 – Histogram of Original Grayscale

After looking at the histogram of the original image, no apparent obvious thresholding points are apparent. Lest the image is low pass filtered at any rate to effectively smooth the image. After the smoothing is applied, the following is the histogram of the smoothed image.

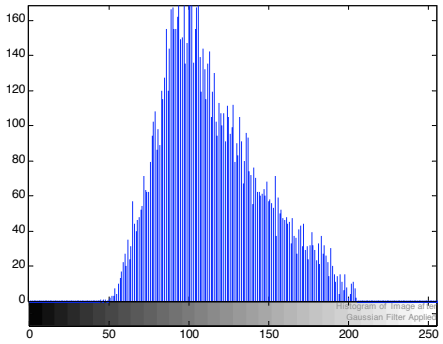

Figure 4 – Histogram of Smoothed Image

There is little to no difference in the distribution of pixels, even after smoothing. One can take note of the slightly pronounced potential separation of the distribution near the peak.

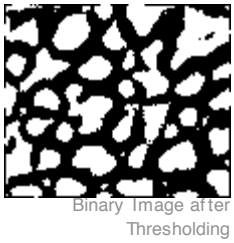

Figure 5 – Adaptivethreshold applied to image

Obtained is figure 5 after applying the *adaptivethreshold* command in Matlab. Again, this looks at a local preselected size neighborhood. It is a moving window sized 9x9. In the image, some small noisy pixel clusters still remain as well as some spurs. The image will need cleaned up a bit.

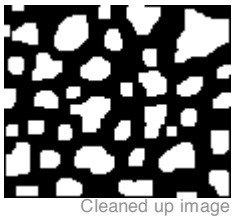

Figure 6

To clean up the image, first, the rings that represent the axons were filled in with white pixels. Then the morphological erosion and dilations via the *imopen* command were performed. Then the *bwareaopen* was applied to eliminate the noisy pixel clusters. Compared to the original thresholded image, one can see the general shape of most of the axons start to be pronounced.

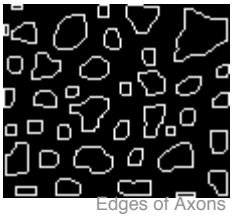

Figure 7 – Edges Obtained

Figure 7 represents the edges of the axons. These edges were obtained using the *bwperimeter* function in Matlab as opposed to using gradient options. While gradient options may or may not have worked well, the following image suggests that, in this case, the algorithm did a pretty good job at successfully finding the axons.

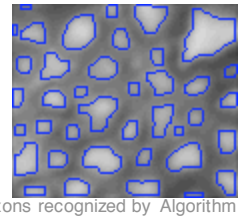

Figure 8 – Original Image with Edges

Looking at the Gold standard image supplied by the expert in the field, a quantifiable conclusion can be drawn regarding the success of the algorithm.

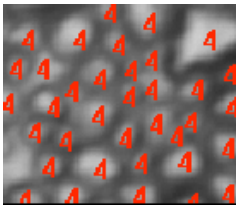

Figure 9 – Gold Standard

All in all, the algorithm did a very nice job accurately selecting the axons. In Sample 4, there are 36 axons in the gold standard image identified by the field expert. Of these 36 axons, the algorithm correctly identified 35 of them. This equates to finding 97.2% of the axons. The algorithm reported finding 39 total axons. Since 35 of them were correct, this means it triggered 4 false positive responses for a 89.7% correct reporting rate and a 10.3% false positive rate. This is certainly not bad given the quality of the image with which to work.

Now that one of the most successful cases has been examined, one of the lesser successful cases will be presented.

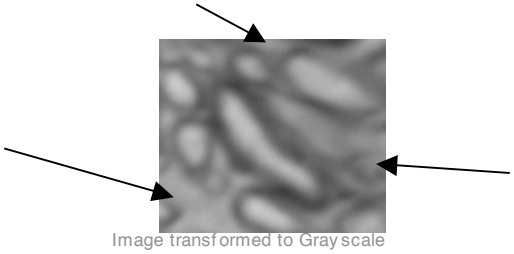

Figure 10 – Sample 5 after converted to grayscale

Notice the areas as indicated by the arrows. These areas are wide spaces where according to the expert, there is no axon present. However, these areas seem to be nearly enclosed by a black ring. This is what typically is the signature of all other axons. The algorithm sees the change between black and white and believes that these are axons and the following figures will illustrate this point.

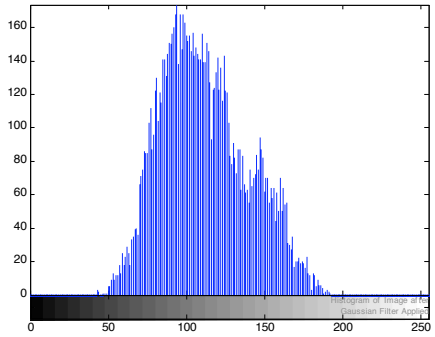

Figure 11 – Histogram after Gaussian Low Pass Filter

Again, notice there isn't an intuitive place to put the threshold. Perhaps in the area of the 140 level would be the best place.

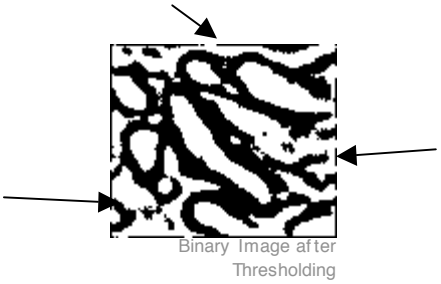

Figure 12 – Adaptivethresholded image

Again, take note of the areas in which the algorithm tends to struggle a bit. This will be further pronounced in the next figure.

Figure 13

Notice how the algorithm believes that there are axons in the areas highlighted by the arrows.

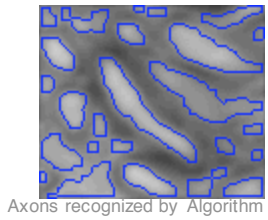

Figure 14 – Edges detected by algorithm overlaid on original image

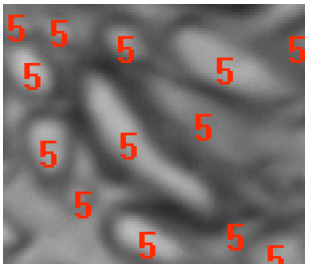

Figure 15 – Gold Standard of Sample 5

Notice that the algorithm did a very good job at finding all of the axons in the gold standard. The biggest draw back is the wide open space in between axons in some regions of the image. This is where the adaptive threshold actually becomes a little bit of a problem for the algorithm. Since there is a lot of white or gray pixels in a wide space where there is no axon, any black pixel noise will greatly stand out as being below the local neighborhood mean. Since it stands out the algorithm sets it below the threshold and creates a false positive. The quantifiable statistics are as follows. There are 13 axons as identified by the expert in Sample 5. The algorithm successfully identified all 13 for a 100% success rate in finding axons. However, the algorithm reported finding 25 total axons. This means that only 52% of the axons it reported were correctly identified and 48% of them were false positives. The following images will illustrate the rest of the samples with algorithm-identified axons and the gold standards.

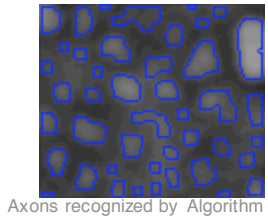

Figure 16 – Sample 1 with axons identified

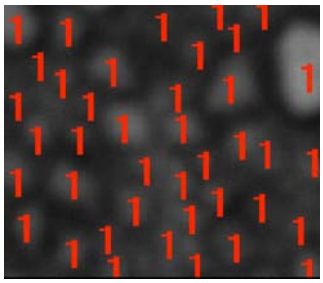

Figure 17 – Sample 1 Gold Standard

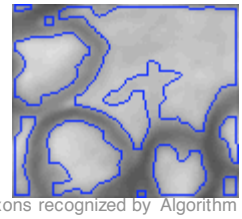

Figure 18 – Sample 2 Axons Identified

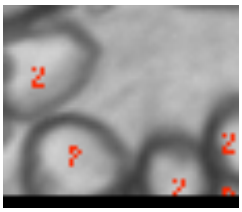

Figure 19 – Sample 2 Gold Standard

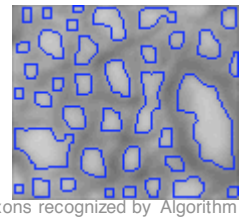

Figure 20 – Sample 3 Axons Identified

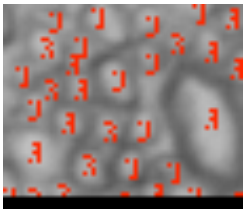

Figure 21 – Sample 3 Gold Standard

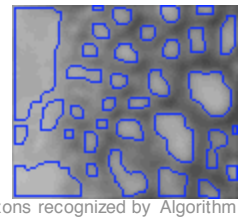

Figure 22 – Sample 6 Axons Identified

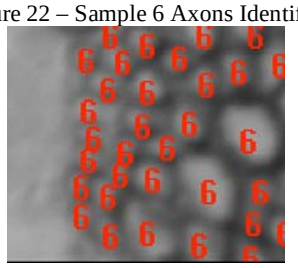

Figure 23 – Sample 6 Gold Standard

CONCLUSIONS

When the algorithm is applied to areas of axon clusters where there is little wide open space, it does a very good job counting all axons present with a minimal number of false positives. When there tends to be more space in the image where no axons are present, the algorithm tends to struggle. Perhaps some sort of masking technique could be developed. One command that may help is the *imclearborder* command. This command in Matlab eliminates anything that is on the edge of the image from being considered. This might help when using a larger image.

Image	Axons in Gold Standard Ima	Axons Report by Algorithm	Axons Report Correctly	% Reporte Correct	False Positive	% False Positiv
1	40	44	37	84.1%	7	15.9%
2	5	10	5	50.0%	5	50.0%
3	32	37	30	81.1%	7	18.9%
4	36	39	35	89.7%	4	10.3%
5	13	25	13	52.0%	12	48.0%
6	33	34	29	85.3%	5	14.7%

Figure 24 – Table of Axon Success Rate

In figure 24, one can see the rate at which the total number of axons reported by the algorithm were correct, and the total number of false positives identified. In images 1,3,4 and 6, the axons were grouped much more tightly than in images 2 and 5. This supports the 'wide open space' conclusion.

Axons in Gold Standard Imag	Axons in Gold Standard Report	% Axons in Gold Standard Report	Missed Axons	% Misse
40	37	92.5%	3	7.5%
5	5	100.0%	0	0.0%
31	30	96.8%	1	3.2%
36	35	97.2%	1	2.8%
13	13	100.0%	0	0.0%
33	29	87.9%	4	12.1%

Figure 25 – Table of Axons found compared to Gold Standard

In figure 25, one can see that, overall, the algorithm did a very good job at finding all of the axons present in the image. On average, over 95% of the real axons are found every time. To make this a most reliable method of axon counting, however, as illustrated in figure 24, the number of false positives would have to be reduced.

ACKNOWLEDGMENTS

This work was supported by ***and*** in support of the Biology department of the Case Western Reserve University, Cleveland, OH.

REFERENCES

[1] Erlend Hodnelad, Xue-Cheng Tai, Joachim Weickert, Nickolay V. Bukoreshtliev, Arvid Lundervold, and Hans-Hermann Gerdes "Level set methods for watershed image segmentation"

[2] Rafael Gonzalez, Richard Woods and Steven Eddins, *Digital Image Processing Using Matlab*.

[3] Jun Yan, Xiaobo Zhou, Qiong Yang, Ning Liu, Qiansheng Cheng and Stephen T.C. Wong, "An Effective System for Optical Microscopy Cell Image Segmentation, Tracking and Cell Phase Identification"

[4] www.mathworks.com

[5] Bin Fang, Wynne Hsu and Mong Li Lee, "Tumor Cell Identification Using Features Rules".