Counting Mouse Neuron Cells using Morphological Image Processing

Dmitri Kourennyi

Introduction

Various methods exist for counting cells in medical images. Simple statistical methods exist, such as the Abercrombie and Empirical methods,which simply use small sections of a set of images to estimate a full count. These methods have flaws due to the large number of assumptions made with them (Hedreen, John C, 1998). Stereoscopic and 3D methods also exist, but require some kind of 3D information in order to be effective. In this paper, I have designed an algorithm that automatically counts the number of cells in a single, 2D image, with only minor thresholding adjustments needed depending on cell sizes and image gamma.

The Algorithm

The entire algorithm is rather short. This is posssible due to the many built in functions of matlab. The full code is shown at the end of this paper. Here, I will go over each of the algorithm's steps as well as the resulting image after most of these steps. The original image is also shown at the end of this paper.

```
Ioriginal = imread('Image.tif','tif');
Ioriginal = imresize(Ioriginal,0.5,'bicubic');
```

The program begins by simply reading the original image and resizing it by 50% with bicubic interpolation. To the eye, the image looks better resized, and the interpolation creates some averaging that removes some of the noise. At the same time, the image's smaller size creates smaller cell boundaries and a sharper result.

```
I =
adapthisteq(rgb2gray(Ioriginal),'NumTiles',[24
24]);
```

```
I = imdilate(I,strel('square',3));
```

The first line simply performs local equalization on the image to optimize the contrast. The result is shown in Figure 1.

The second line performs a dilation with a square kernel to thin the cell edges and further remove noise. The results of the dilate operation is shown in Figure 2.

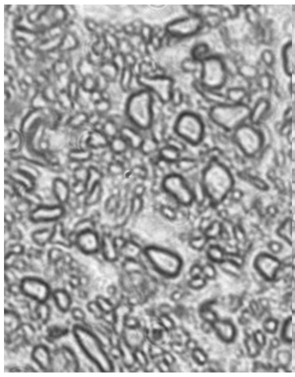


Figure 1: Equalization

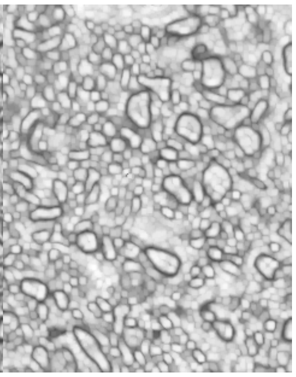


Figure 2: 3x3 Square Dilation

```
I = I<200;
dim = size(I);
I = ~imfill(~I,'holes');
```

Next the image is threshholded (and inverted in the process), and I take advantage of Matlab's fill function to remove holes in the image. I get an image which is black in cell interiors and exteriors, and white where cell borders are. Note that the previous dilation has separated the cells from each other. Removing the holes removes the nuclei that are visible in some cells. I remove these so they don't interfere with the cell selection step later in the algorithm. The results of the thresholding and hole filling are shown in Figures 3 and 4, respectively.

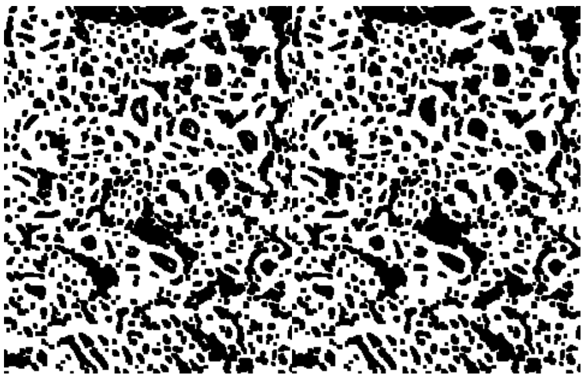


Figure 3: Thresholding *Figure 4: Holes Filled*

At this point, each black area represents an area that corresponds either to the interior of a cell, or the exterior between cells. The next section of code isolates each separate black region, and performs some tests on, either confirming or rejecting it as a cell, and updating a counter and aggregate image for displaying the final result. The code loops through the entire image, testing one pixel at a time.

```
if(~Ifill(i,j))
 Ifill = imfill(Ifill,[i,j]);
 Idiff= Ifill - I;
 I = Ifill;
 sumarray = horzcat(sumarray,sum(sum(Idiff)));
```

For efficiency, the algorithm check to see if the pixel is black. This saves time by avoiding calls to the imfill function, which is quite slow. If the pixel is indeed black, imfill flood fills the connected area with white. The difference is recorded in Idiff, and I is updated to be ready for the next difference check. The advantage of this method is that it avoids double without any extra images by taking advantage of the fact that the fill function removes all the connected black pixels, so they will not be considered again. Finally, the sum of the isolated region is appended to an array. This array was used to obtain histograms of the area distributions of regions. These are shown at the end of the paper. An example of the difference is shown in Figure 5.

```
Idiff2 = Idiff -
imerode(Idiff,strel('diamond',1));
borderarray =
horzcat(borderarray,sum(sum(Idiff2)));
```

Next, the difference between the isolated region and its erosion isolates the border of the region. 8-connectivity was used to utilize the most degrees of freedom for border approximation. As before, this border was stored in array in order to analyze the distribution. An example border is shown in Figure 6.

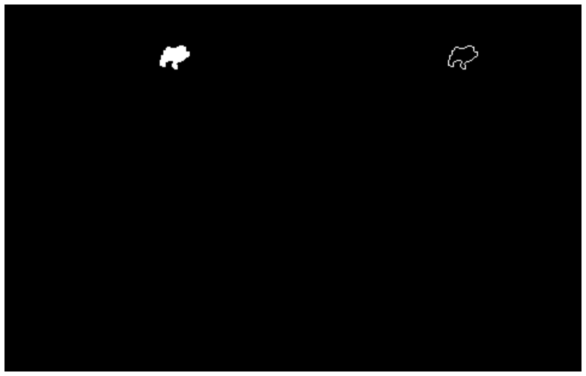


Figure 5: Example Region *Figure 6: Border of Same Region*

```
temp = 2 * pi * sqrt(sum(sum(Idiff)) / pi) /
sum(sum(Idiff2));
if(sum(sum(Idiff))<1000 && sum(sum(Idiff2))<140
&& temp<1.5 && temp>0.7)
 Iresult = Iresult + Idiff;
 n=n+1;
end
```

The last part of the algorithm performs checks on each region to classify it as a cell. 3 checks are made: First, the region cannot be bigger than 1000. This eliminates large open areas that just cannot be cells. Of course, this parameter depends on the scale of the image. Next, the border cannot be larger than 140. This eliminates extremely convoluted borders that are going to be associated with the empty space between cells. Finally, the variable temp stores the ratio of the border to the circumference of a circle with the same area (some constants not used). This value is bounded by 0.7 and 1.5, and eliminates highly eccentric regions. If a region passes all 3 criteria, a counter is incremented and the region is added to a seperate image.

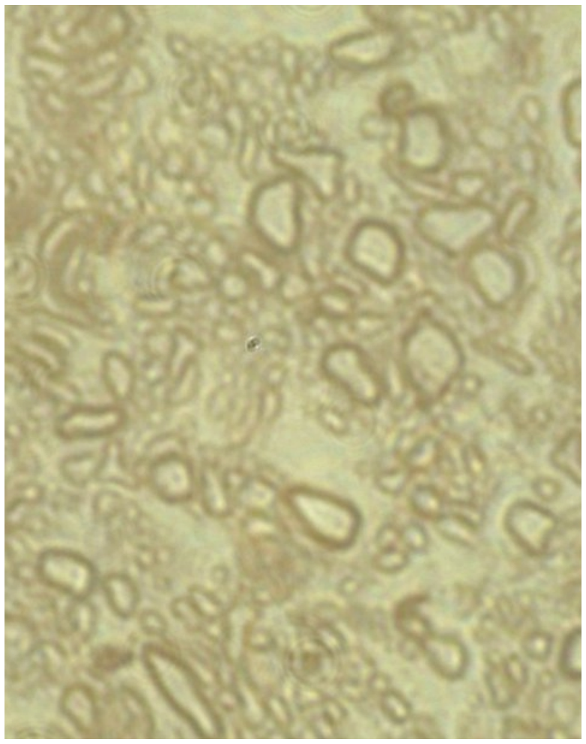
The end of the algoritm simply overlays the resulting image to the original for easy viewing of the results.

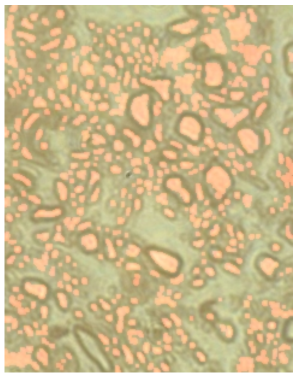
Discussion

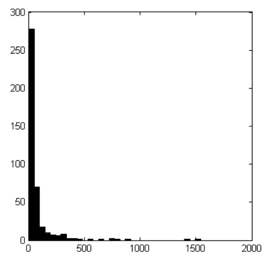
There are a number of flaws with the algorithm. First, although a smaller image is easier to work with, the loss of data results in smaller regions when cells are being checked. Smaller regions suffer from increased error due to the discretization due to pixels as opposed to a continuous definition of a circle. A better alternative would be to increase sharpness, contrast and remove noise without resorting to resizing the image, and thus retaining maximal spatioal information when borders and areas are determined.

The second major flaw is how cells are determined. Borders could be more efficiently calculated by tracing the border and recording true Euclidean distance (diagonals contribute √2/2, etc.) The ratio of border length versus ideal border length should also be standardized for radius. I tried to implement this, but pixelation of small regions required more 'generous' cutoff values, and thus make standardization useless. Also, better methods can be used for testing. For example, one can find the center of the region, and then determine 2 circles, one that completely encloses the region, and one that just touches the inside of the region. Analyzing the difference between the radii of these circles can help determine eccentricity of the region.

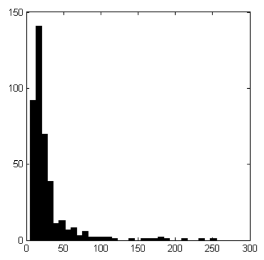
Results


Original Image

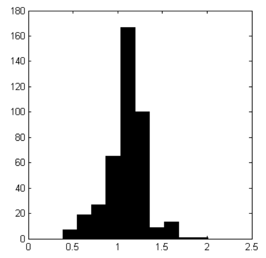

367 Counted Cells (Note the false positives at the top right and the false negative at the bottom caused by extreme eccentricity)



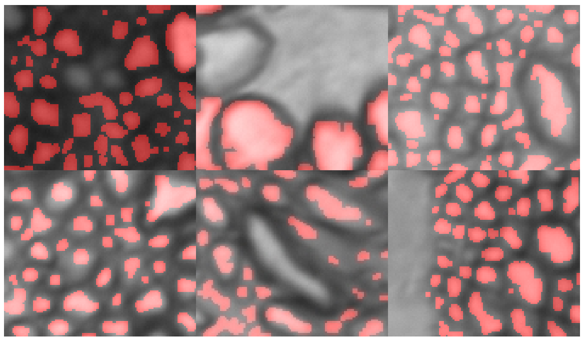
Region Areas Histogram



Region Borders Histogram



'temp' values Histogram



Resulting counts: 39,7,52,37,30,39

Full Code

```
Ioriginal = imread('Image.tif','tif');
Ioriginal = imresize(Ioriginal,0.5,'bicubic');
I =
adapthisteq(rgb2gray(Ioriginal),'NumTiles',[24
24]);
I = imdilate(I,strel('square',3));
I = I<200;
dim = size(I);
I = ~imfill(~I,'holes');
sumarray = [];
borderarray = [];
Ifill = I;
n=0;
Iresult = zeros(dim(1),dim(2));
for i=1:dim(1);
    for j=1:dim(2);
        if(~Ifill(i,j))
            Ifill = imfill(Ifill,[i,j]);
            Idiff= Ifill - I;
            I = Ifill;
            sumarray =
horzcat(sumarray,sum(sum(Idiff)));
            Idiff2 = Idiff -
imerode(Idiff,strel('diamond',1));
            borderarray =
horzcat(borderarray,sum(sum(Idiff2)));
            temp =
2*pi*sqrt(sum(sum(Idiff))/pi)/sum(sum(Idiff2));
            if(sum(sum(Idiff))<1000 &&
sum(sum(Idiff2))<140  && temp<1.5 && temp>0.7)
                Iresult = Iresult + Idiff;
                n=n+1;
            end
        end
    end
end
Ioriginal(:,:,1) = Ioriginal(:,:,1) +
128*uint8(Iresult);

imshow(Ioriginal)
```

References

Hedreen, John C. "What Was Wrong With the Abercrombie and Empirical Cell Counting Methods? A Review." The Anatomical Record 250(1998): 373-380.