A UTILIZATION OF CONVOLUTIONAL MATRIX METHODS ON SLICED HIPPOCAMPAL NEURON REGION IMAGES FOR CELL SEGMENTATION

Neeraj Rattehalli[1] Ishan Jain[2]

[1]Computer Science, Menlo-Atherton High School, Atherton, California
[2]Computer Science, Mission San Jose High School, Fremont, California

ABSTRACT

Present methodologies for cell segmentation on hippocampal neuron regions contain excess information leading to the creation of unwanted noise. To distinctly draw boundaries around the cells in each of the channels like DAPI, Cy5, TRITC, FITC, it is pertinent to start off by denoising the present data and cropping the relevant ROI for analysis to remove excess background information. Present edge detection methodologies like Canny Edge Detection create black and white outputs. It is difficult to accurately do edge detection with color throughout an entire image. As such, we utilized a more involved approach that uses pixel level comparisons to determine the existence of an edge points. By extrapolating all the available edge points, our algorithms are able to detect general edges throughout an imagine. To streamline the process, it has been accompanied with a GUI interface which allows for freehand crops. This information is stored in a downloadable txt file, which provides the necessary input for the thresholding and final cropping. Together, the interface works to create clean data which is ready for further analysis with algorithms likes FRCNN and YOLOv3.

KEYWORDS

Convolutional Matrix, Computer Vision, Automation Interface

1. INTRODUCTION

It is rare to find data without noise. As such, it is an important process to denoise image data in order to have the best data for a training model. In the hippocampal neuron body image data there exist areas with a lot of extra noise due to stain leaks. In addition, there exist locations where cells are clumped together making it difficult to separate with the human idea. Current work can segment cells to a degree but not efficiently and have a lot fo room for error. It currently works best for images where the cells are small and sparsely distributed. If the colors vary within an image, it may be further difficult to separate and different approaches are needed to do so.

Additionally, current filtration algorithms such as OpenCV's canny edge detection models only function on black and white images [1]. Other existing models such as the Sobel, Prewitt, and Laplacian edge detection algorithms also leverage a gray-scale conversion while conducting edge detection processes [4, 5, 6]. The processes remove the color from the image, causing the image spectrum to be heavily manipulated. Our goal was to create an accurate model that can perform tasks despite the color gradients present within an image. Hence, our algorithms can process accurately on neuron data without changing the dynamics of the colors of the image.

Current art tends to fail when neurons are clumped together, treating bunches as a singular neuron as opposed to multiple. This means there is an ultimate failure in adaptive segmentation.

Without denoising, some unwanted regions might also be included in such contours. When this data is included in a training model, it can create unwanted results and can create confounding during feature extraction. As such, it is extremely important to remove this noise. Although this may be avoided through current cropping tools, there doesn't exist a uniform interface which allows for complete automated processing. As such, there was an opportunity for improvement, and the possibility of a new interface which combined all such features.

In addition, many current methods lack modularity. When presented with different color stains, it impulsively utilizes the same approach without necessary changes for different colors. Images that contain a dominant color that is unsuitable for the algorithm is rendered useless for current algorithms. The RGB combination demonstrates a dominant color for neuron data. Some neuron stains contain red dominance (TRITC), while other stains such as the FITC stain contains a green dominance. With these disparities, current algorithms are not adaptive. To combat this issue, we allow for a user inputted color and threshold to use as a basis for denoising.

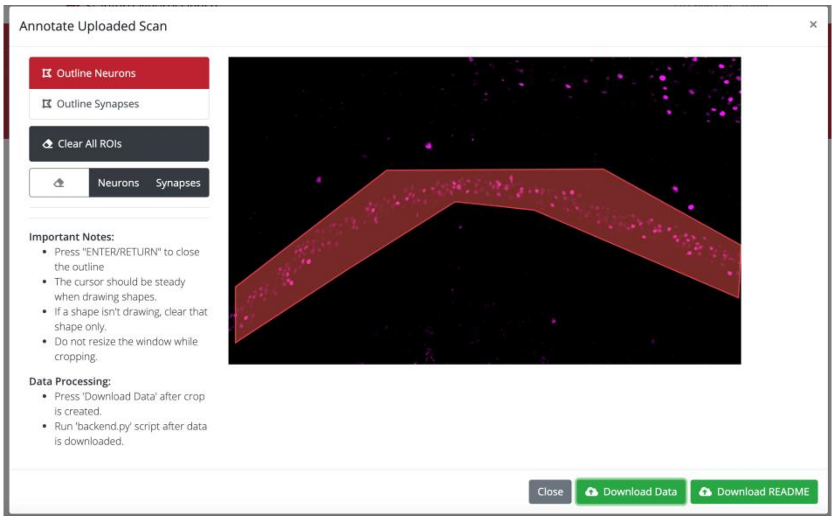
To combat all the present problems, we developed a user interface that includes several novel algorithms for denoising data and edge detection through convolutional matrix filters. Paired with a convenient application interface, it is easy to use in a production ready environment.

2. METHODS

The program flow from start to finish begins with a web based UI that has features for outlining, masking, and labeling regions. This data will be used to create a free hand crop. After a user has created the crop, they have the option to delete the crop and start over or download a data.txt file with information about coordinates as well as image size. This is important as the image is stretched when placed on web based UI. Due to these DOM changes, a python script fixes these cropped coordinates in accordance with the new size. This is then cropped and becomes the basis for denoising. The different algorithms are run one by one in a specified order to create a final denoised image ready for an upcoming training. The new images create general segmentations around each neuron.

2.1. GUI

Current tools for addressing image segmentation processes include cropping methods. Although images are able to cropped accurately, current available systematic methods lack the full-stack implementation to create crops, process the region of interest (ROI), and accurately contour the neurons in a single application. Using web development resources, we created a web app that includes a cropping tool keeping UX/UI in mind. With an input-image selector, one can input a given image for processing. After the respective crop is completed, one can download the neuron data to the local system. By executing a script, the neuron data (preferred threshold, dominant color, etc.) is taken into account, and the image is processed and contoured accurately in approximately 10 seconds on a machine with an Intel Core i5 processing unit. With an application that incorporates a variety of languages (web development and image processing), we developed an application that combines the resources for cropping and filtering/segmenting images in a customizable manner. Fig. 1 demonstrates an accurate representation of the web interface that is employed for the full-stack image segmentation tasks.



Important Notes:
- Press "ENTER/RETURN" to close the outline
- The cursor should be steady when drawing shapes.
- If a shape isn't drawing, clear that shape only.
- Do not resize the window while cropping.

Data Processing:
- Press 'Download Data' after crop is created.
- Run 'backend.py' script after data is downloaded.

Fig 1. A representation of the cropping tool with a generated crop around the region of interest (ROI). Erasures of different crops can be done via the different options presented.

2.2. Is On Edge

Traditional edge detection algorithms use implemented kernel methodologies to run a convolutional matrix throughout the image to create edges based on pixel intensity contrast. The algorithms present in this paper employ similar convolutional matrix methods however without the use of filtration and substituted with a novel checking system. The algorithm starts by analyzing a 3x3 pixel matrix of the image. The analysis is simply done on the central cell. It starts off by checking the presence of null information which occurs with low pixel intensities for each of the RGB values. For general purposes and through testing, we deemed an ideal value for such a threshold would be 50. Thus, depending on the stain the user is analyzing, the algorithm checks for null intensity on the appropriate RGB value (e.g. the TRITC channel which creates a red stain has a null intensity check for the R value of the pixel). If there exists a pixel in the 3 by 3 matrix other than the center pixel which has a pixel intensity for the respective channel that is less than 50, then it means the center pixel is in the presence of a cell with null intensity. Now, there is a second part for edge detection: it is to determine if the center pixel is in contact with a colored pixel (e.g for the TRITC channel, if the center pixel is in direct contact with a red pixel). This can be conclusively determined through thresholding. Once again, we deemed an ideal value for this would be 50 as well. Finally, if the center pixel is in contact with both a colored pixel and a pixel with null intensity, then it lies on the edge of a neuron. The general algorithm is presented below in pseudo code.

Line 1: denotes the pixel value intensity for the 3 by 3 cell matrix
pixels = [[i1, i2, i3],[i4, i5, i6],[i7, i8, i9]]
// i_n = the pixel color intensity for the appropriate light signal for a specific pixel for i = 1,2,3...9.

Line 2, 3: defines threshold limits for null intensity and colored intensity
thresholdLower = 50 // Can be set by user
thresholdUpper = 50 // Can be set by user

Line 4: function definition for isOnEdge
function isOnEdge(pixels):

```
        Line 5, 6: boolean variables for states of neighboring pixels
        doesNullIntensityExist = false
doesColorIntensityExist = false

Line 7, 8: double for loop to parse through pixels array
for row in pixels:
        for pixel in row:

                Line 9, 10: Null intensity check
                if pixel < thresholdLower:
                        doesNullIntesityExist = true

                Line 11, 12: Color intensity check
if pixel > thresholdUpper:
                        doesColorIntesityExist = true

Line 13: returns the boolean function for both color and null pixel intensity existing.
return doesNullIntensityExist and doesColorIntensityExist
```

2.3. Edge Detection

In the previous section, a general methodology for determining if a certain pixel resided on the edge of a neuron was outlined. In order to actually create a new image with the outlines of edges, this function must be run on each 3 by 3 pixel chunk of the image. If the function returns true, which indicates that the center pixel neighbors both a null intensity pixel and a colored pixel, the center pixel's color is changed to white to indicate that it is on the edge of a neuron. After running this function through each 3 by 3 section, a general edge outlines are created.

2.4. Denoising

After completing the edge detection, there are two main ways to reduce the noise and create a final clean output image. The first is thinning out the white borders as some become multiple pixels thick, and the second is the small region removal.

2.4.1. Thinning out White Borders

After the creation of edges, some boundaries are several pixels thick. This reduces the quality of the original image and takes away some meaningful data. By thinning out these extra thick regions and by adding back the original data, we are creating more accurate data. Thinning out the borders can be done in 2 main ways. First is to create an inner and outer border, and then, it needs to remove the inner border. To create the inner and outer borders, a 3 by 3 matrix is run throughout the image. It is checked to see if it contains all white. If it does, then the center cell is converted back to its original color. This is done throughout the image creating two regions: an inner and outer. In order to remove the inner boundary, we check for the proximity for white pixels in both directions. An inner borders will contain white pixels in both directions within 20 pixel proximity while outer borders will not. In addition, some lone points are created. This can be removed through morphology in OpenCv or through a manual approach which simply checks for null intensity in all the neighbors of a specific pixel.

Alternatively, this can also be done through erosion in OpenCv, which shrinks clumped regions. The only issue is that this solely works on a masked region which would incur multiple steps. In addition erosion doesn’t use the outer border, simply just goes with the center by thinning our each region directly.

2.4.2. Removing Small Points

This algorithm is relatively straightforward. It takes 9x9 pixel matrices. The algorithm then proceeds to check if the boundary of these pixel matrices is entirely black. If so, this means that either the entire 9x9 section is black or there were small points that were entirely contained in this 9x9 interval, which need to be removed. If the edge is entirely black, then it blacks out every pixel in this 9x9 pixel matrix since it's either null information or small points. This works because all of the cells in our imaging data were much larger than a 9x9 pixel square matrix.

2.5. Data Transfer

In order to adequately incorporate a method that utilizes a cropping tool with the image analysis algorithms in a systematic full-stack tool, we developed a data transfer method to allow the script to process the image with the crop data that was generated from the web app. After data is downloaded from the web tool, the script processes and parses the input data in a correct format. Then, using our image segmentation algorithms and feature explorations, the image is processed within the same run time. The analyzed image with contours is exported to the local filesystem.

2.6. Data Parameterization

To create specifications for each data sample, we allow for command line arguments into each function. These include color and pixel thresholding. Together with the data.txt file, the python script has enough data for image processing. In order to determine the ideal threshold, the user can use a binary search on the 256 channels in one specific hue and do so efficiently in 8 trials.

2.6.1. Future Algorithms

Other prospective approaches include a novel bucketing algorithm algorithm. This will be able to automatically determine thresholds without any user input. By create buckets for each 25.6 chunk of pixels, we have 10 buckets to group pixel intensity. By locating the 75% mark, we will be able to determine the ideal pixel intensity for thresholding. This 75% was determined through trial and error; however, it can also be set by the user adding for an extra layer of specification on top of a suggested value.

3. Results

The following (Fig. 2) describes the application of the edge detection algorithm on the Cy5 channel stain. As evident, the white contours are able to distinguish between each neuron and the background.

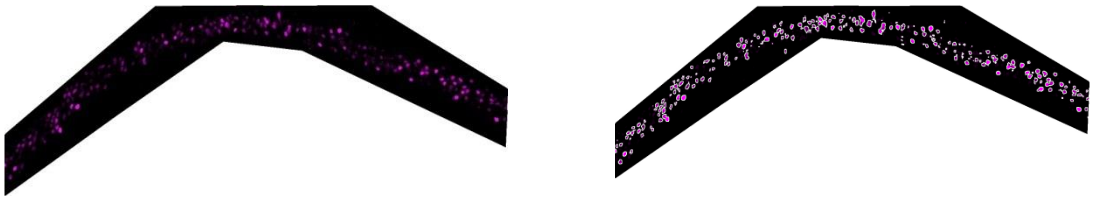


Fig 2. A side-by-side comparison of a before vs. after edge detection process on the Cy5 channel

The following (Fig. 3) describes the application of the edge detection algorithm on the DAPI channel stain. As evident, the white contours are able to distinguish between each neuron and the background.

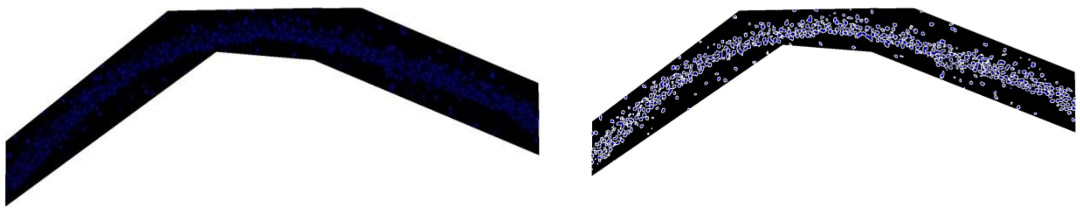


Fig 3. A side-by-side comparison of a before vs. after edge detection process on the DAPI channel

The following (Fig. 4) describes the application of the edge detection algorithm on the TRITC channel stain. As evident, the white contours are able to distinguish between each neuron and the background.

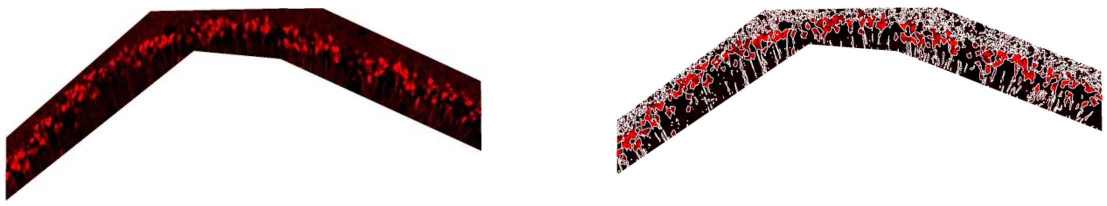


Fig 4. A side-by-side comparison of a before vs. after edge detection process on the TRITC channel

The following (Fig. 5) describes the application of the edge detection algorithm on the FITC channel stain. As evident, the white contours are able to distinguish between each neuron and the background.

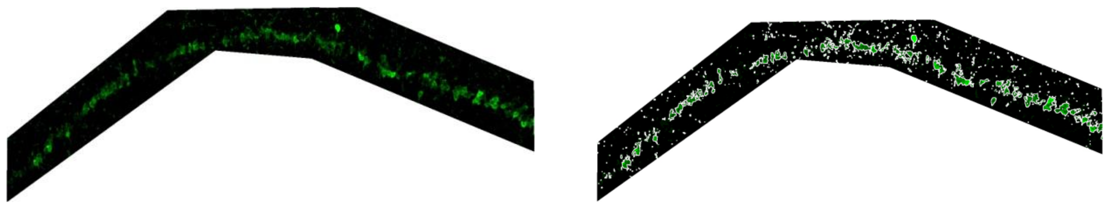


Fig 5. A side-by-side comparison of a before vs. after edge detection process on the FITC channel

The following (Fig. 6) describes the application of the white removal algorithm on the TRITC channel stain. As exhibited, the density of the white contours has been reduced.

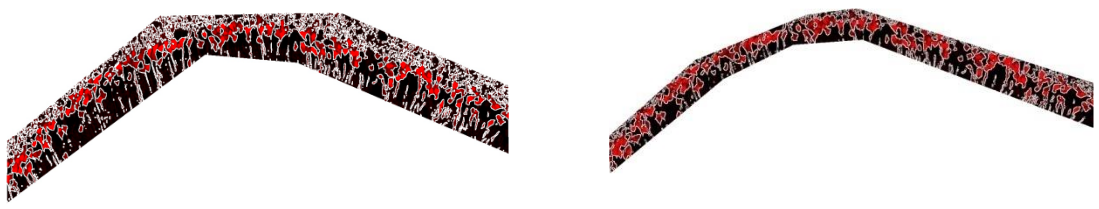


Fig 6. A side-by-side comparison of a before vs. after White Removal algorithm process on the TRITC channel

The following (Fig. 7) describes the application of the small point removal algorithm on the TRITC channel stain. As exhibited, the noise due to the synapses (small points) has been reduced. The final image at the bottom demonstrates the small point removal algorithm with the crop applied which has crisper quality as opposed to the original data.

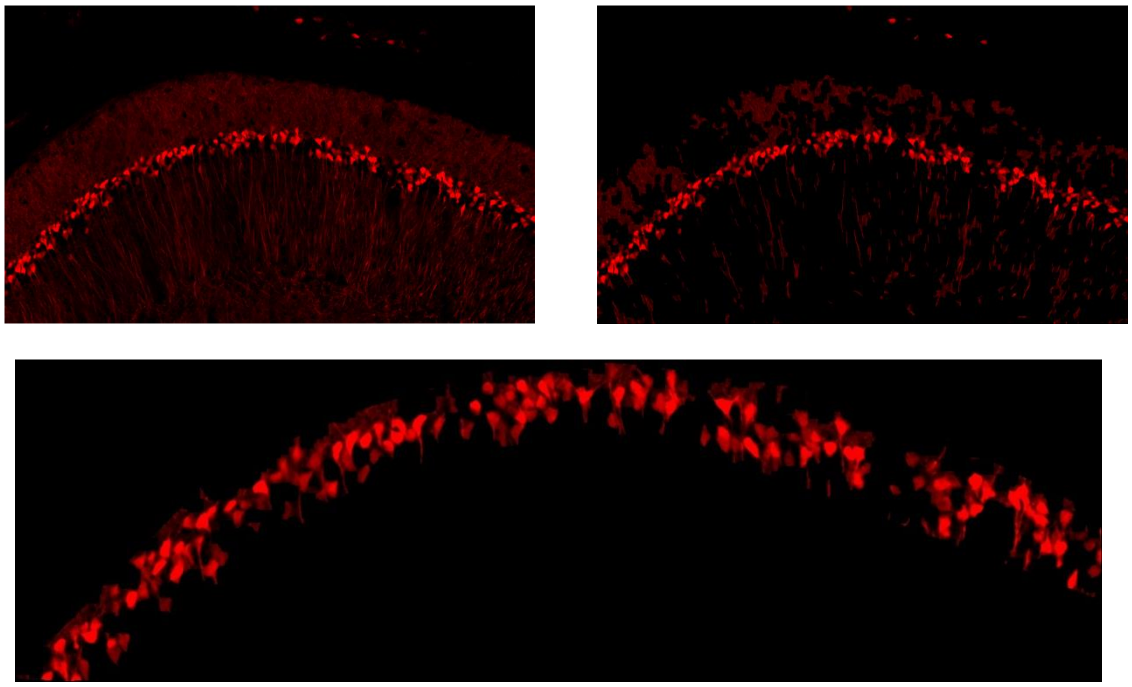


Fig 7. Top: A side-by-side comparison of a before vs. after Small Points Removal algorithm process on the TRITC channel

Fig 7. Bottom: Small Point Removal algorithm process on the TRITC channel With Crop

The following (Fig. 8) describes the application of the strengthen pixel intensity algorithm on the TRITC channel stain. Paired with the small point removal algorithm, this final data is the most clean as it contains thin borders/contours, an accurate crop, and little to none noise due to the synapses.

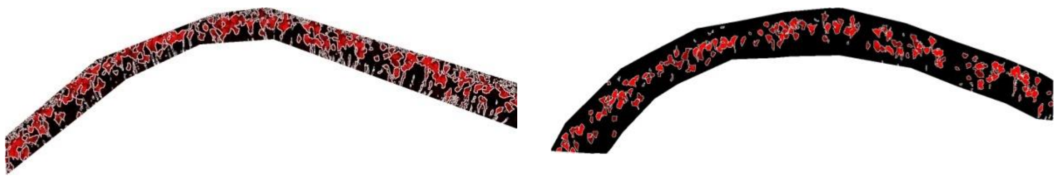


Fig 8. A Side-By-Side Comparison of A Before and After of the Strengthen Intensity Algorithm on TRITC Channel

4. Conclusions

In conclusion, the project creates novel algorithmic approaches for automation of denoising in various hippocampal neuron data through various image stains. Through use of an interface, there is easy access for all professional. With a convenient UI and quick data turn around, we have created a seamless way for creating cell contours and input data for various machine learning models. Together this suggests a future with automation and easy modularity in the present field.

ACKNOWLEDGEMENTS

We would like to thank our mentors, Professor Lu Chen and Dr. Bing Cao for guiding us through our project and helping us with the sufficient resources. In addition, we would like to thank Dr. Jie Li for providing us with our training and input data. We thank Dr. Amy Shon for providing us with access to GPU and computing resources such as the Sherlock cluster for us to train and run our models and matrices. Finally, we thank the Lu Chen lab from Stanford for providing us with the images and data that was adequate for conducting our research.

REFERENCES

[1] J. Canny, "A Computational Approach to Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[2] I I. Culjak, D. Abram, T. Pribanic, H. Dzapo and M. Cifrek, "A brief introduction to OpenCV," 2012 Proceedings of the 35th International Convention MIPRO, Opatija, 2012, pp. 1725-1730.

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi; The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788

[4] Dey, D. and Polley, D., 2020. Edge Detection By Using Canny And Prewitt. [online] Ijser.org. Available at: <https://www.ijser.org/researchpaper/Edge-Detection-by-Using-Canny-and-Prewitt.pdf> [Accessed 25 June 2020].

[5] Owlnet.rice.edu. 2020. Laplacian Edge Detection. [online] Available at: <https://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/laplacian.html>

[6] Neeraj Ratetehalli and Ishan Jain, "An Adaptive Utilization of Convolutional Cell Matrix Methods on Sliced Hippocampal Neuron Cell Segmentation with Man Application Interface" , ICAITA 2020 (SAI 2020), Jun. 2020

AUTHORS

Neeraj Rattehalli is a Bay Area high school student with a passion to automate the modern world. Growing up in a rapidly changing progressive society, Rattehalli has found machine learning and computational biology around every corner, and in order to stay steps ahead of the crowd, Rattehalli is conducting novel research at Stanford under the guidance of Lu Chen, Professor of Neuroscience. Rattehalli works with the intention and the zeal to solve today's most intricate challenges.

Ishan Jain is a Bay Area high school student with a passion in machine learning and computer vision. Ishan has worked on a variety of projects at Stanford Medicine, including analytical methods for assessing patients with peripheral artery disease (PAD) and developing mobile tools to create a remote surveillance application for postoperative surgery treatment using opioid medications. Furthermore, Ishan has worked with research professionals at the University of California, Santa Barbara, where he developed a meta-analysis tool to automate research manuscripts. Ishan is highly passionate about utilizing computer vision methods in application to the modern world.