IfThresholds Software Package: score ranking standard threshold methods by comparing images produced by thresholding with manually labeled images

Qiong Zhou

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

University of Washington

2021

Committee:

Elizabeth Nance

David Beck

Program Authorized to Offer Degree:
Chemical Engineering

University of Washington

Abstract

IfThresholds Software Package: score ranking standard threshold methods by comparing images produced by thresholding with manually labeled images

Qiong Zhou
Chair of the Supervisory Committee:
Elizabeth Nance
Department of Chemical Engineering

This dissertation details my research on threshold methods commonly used in image analysis and the development of the Python package ifThresholds. Imaging in biological and life science research is routinely performed, almost every biological scientist utilizes imaging to gain insight on structures and functions of cells, measure biological parameters, and monitor substrate in real-time. However, quantifying images to draw meaningful conclusions can be challenging. In my dissertation, I showed the background and the motivation of the project along with an overview of the image analysis field followed by descriptions of the images I used to generate the final score ranking table. I introduced the software I used to manually label images and the programming language I chose to develop ifThresholds. In addition, I presented the development of the software package and the discussed the results. So far, ifThresholds assessed and compared 7 global threshold methods that are easily accessible in various image processing software with images obtained from the experiment performed in my lab. The score ranking table can help researchers and scientists see how well each threshold method performs, thus helps them choose the best threshold method. Users can also run ifThresholds from the terminal to analyze their images and have score tables saved locally. Lastly, I discussed the future directions of ifThresholds which include implementing a decision tree to predict the best threshold method with only a few image data inputs, developing a user-friendly web application, and generalizing the package to multi-channel images.

Preface

This thesis embraces all the efforts that I put during the one and half years as a master's student at the University of Washington. I have been working under the supervision of Dr. Elizabeth Nance, who is also the leader of the research group I am part of. In this time frame, I researched threshold methods on cell images and built a Python package, ifThresholds. There are so many people who, directly or indirectly, have helped and supported me to finish my dissertation.

Thank you to my PI Dr. Elizabeth Nance for always inspiring, encouraging, and trusting me. Besides helps with research, you have also given me so much emotional supports throughout graduate school. You motivated me and made me feel more and more confident about what I had accomplished. Under your mentorship, I always felt cared for and supported, and I also had so much freedom to design the project and explore different possibilities in my research.

Thank you to Dr. Dave Beck for serving on my committee and introducing me to the field of data science. I had a great experience in the two data science classes taught by you, and I learned Python, machine learning, project management ability, and a lot of other important skills which helped me to build the ifThresholds package for my own thesis project later.

Thank you to my undergraduate mentor, Dr. Shaoyi Jiang, who inspired me, trained me to conduct research collaboratively and individually, and gave me advice on professional and personal growth.

Thank you to the entire Nance Lab family, and especially Hawley for always being there for me since the first day I joined the lab and helping me start the ifThresholds project.

Thank you to my parents and all my friends. Without you, I would not be able to survive graduate school during a global pandemic.

Qiong Zhou
Seattle
March 2021

Contents

List of Figures

List of Tables

Introduction 1

Almost every biological scientist utilizes imaging to gain insight on structures and functions of cells, measure biological parameters, and monitor substrate in real-time (Eils & Athale, 2003; Alivisatos et al., 2005; Chen & Murphy, 2004). Quantifying images to draw meaningful conclusions can be challenging: the quality of the image can depend on the quality of the sample and the process of sample preparation; the image acquisition process is not standardized from scientist to scientist, making reproducibility in image acquisition is still developing; and properties of the image, such as resolution and noise in the background can make interpretation difficult (Cheng et al., 2007). In addition, to maintain high image quality, file sizes are often large (0.5 megabytes for a 512×512 pixels image frame and near 2 megabytes for a TIFF image stack) and to generate statistically relevant conclusions, many images must be taken, processed, and stored, an approach that is often done manually. In light of these challenges, computer-assisted automated methods were developed to help make the image processing tasks significantly faster, more efficient, and with higher accuracy by diminishing the amount of labor and human errors. The use of image processing techniques for all kinds of images, especially for microscope images, has become increasingly prevailing in various fields.

Computer-assisted automated image processing methods, by definition, start with an image and end with an image, where images can be represented as an array with each pixel have a defined intensity or brightness value (Russ, 2012). Thresholding is one of the image processing methods that divides an image into groups of pixels that belong to the objects and groups of pixels that represent the background. For example, automatic threshold methods are used to converts grayscale images into binary images and dynamically find the optimal threshold that separates background and foreground objects (Yan et al., 2005). However, there is no quality assessment for implementing these different threshold methods. Generally, researchers choose the most cited method and often fail to cite the reason why they chose the method to process their images. Therefore, a quality assessment on threshold methods is needed to help researchers choose the most appropriate and accurate method.

In my thesis, I investigated the use of an algorithm to quantitatively identify the most appropriate threshold methods for biological images. Specifically, I built a Python package called *ifThresholds* to score and assess threshold methods for fluorescent images of brain cells. *IfThresholds* guides researchers on which threshold method is the best one for their images. The threshold ranking is based on manually labeled images and images produced by applying seven threshold methods. These threshold methods include Li, Mean, Minimum, Otsu, Isodata, Yen, and Triangle. Chapter 2 of this thesis provides a description of the images used to build *ifThresholds* and a description of the software tools and packages that are incorporated in *ifThresholds*. Chapter 3 presents the detailed development of *ifThresholds* and the results when applying it to the images obtained from my lab. In the concluding chapter 4, a discussion on the future directions of *ifThresholds*.

1.1 An Overview of Image Analysis

Mathematically, what is an image? A monochrome image can be represented as a 2-dimensional intensity function, $f(x, y)$, where x and y stands for the spatial coordinates and the value of f at this point corresponds to the intensity value (Chitradevi & Srimathi, 2014). Pixel is the element in the 2-D integer array

(Petrou & Petrou, 2010). For multi-channel image, the value of f represents a vector with three components of intensity values from three color channels, red, green, and blue, of the image at point (x, y) (Intaramanee et al., 2017). It can be represented by a series of 2-dimensional arrays that indicate different color channels. While monochrome image refers to the use of a single color which means that it could be a black and white image or a color filter, the grayscale image refers explicitly to the use of different shades of gray. A grayscale image has a certain number of bits of information per pixel. For an 8-bit-deep grayscale image, each pixel in the image has 256 possible gray values varying from 0 to 255 (Shnayderman et al., 2006). In contrast, in a binary image, a pixel can only take on the value of either 0 or 255.

As shown in the table below, three disciplines are classified according to the type of input and output of the process. Image processing has both the input and output as images.

Fields	Input	Output	Purpose
Image Processing:	Image	Image	Enhancement of images
Computer Vision:	Image	Description	Data analysis of the images
Computer Graphics:	Description	Image	The creation of images

Table 1.1: Image processing fields

For fundamental steps in image analysis, image acquisition is the process of capturing an image by a sensor and digitizing it. The step of image enhancement is when we manipulate images to bring out certain features of interest and make it more suitable for specific tasks. (Bankman, 2009). In most cases, a grayscale image is preferred because it can reduce storage and retain the target features (Petrou & Petrou, 2010). Segmentation is operated to further extract the certain features/structures/objects of interest from the background and from each other. It is an indispensable analysis function, and many algorithms have been researched and developed (Fu & Mui, 1981). While hundreds of segmentation methods are presented in the literature, there is not a perfect method that performs well for all images, nor can one type of image get similar results from all the segmentation methods. Each image has its own unique features and a segmentation method

may not remain good performance for different types of images. Therefore, segmentation methods are divided into different kinds. Thresholding is a common one that is based on the similarity of grayscale values that divides pixels into two groups based on whether it has intensity values less than or greater/equal to the calculated threshold value (Bezdek et al., 1993). It presumes that the object and background pixels can be distinguished by the selection of an optimal grayscale value. Hence, this selection is critical to ensure a successful cell—background partition and analysis of cell characteristics.

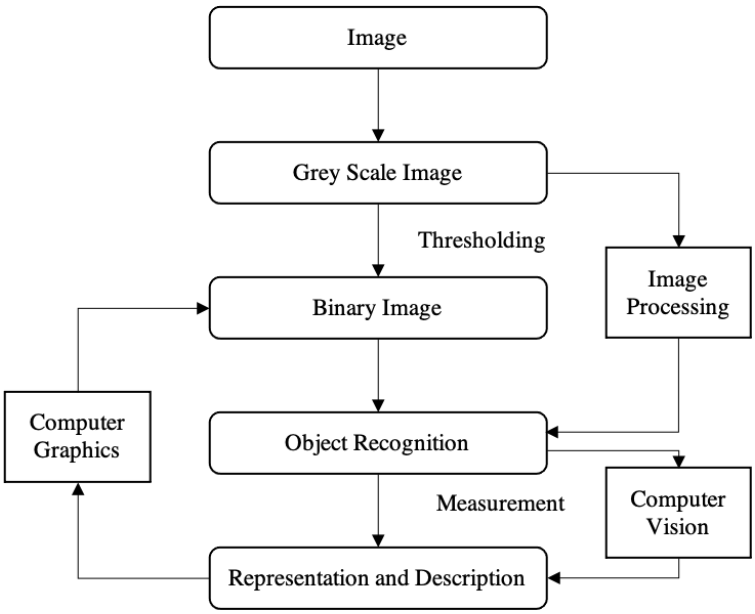

FIGURE 1.1: An overview of the steps in image analysis.

1.2 Segmentation

Image segmentation divides an image into multiple segments of interest by the process of pixel manipulation and helps us to

obtain meaningful information from the image easily. Nowadays, image segmentation technique has been applied to various field and has become quite challenging due to its wide utilization (Xue et al., 2018). A large array of segmentation algorithms has been extensively studied in the last decade (Drozdzal et al., 2018). The majority of the proposed image segmentation techniques can fall into one of the following categories: threshold-based, region-based, edge-based, fuzzy theory-based, artificial neural network (ANN)-based, and partial differential equations (PDE)-based (Pal & Pal, 1993). The threshold-based technique is commonly considered as one of the most exploited and well-known techniques due to its simplicity and efficiency (Feng et al., 2017).

1.3 Thresholding

Image thresholding is used to converts grayscale images into binary images and dynamically find the optimal threshold that separates background and foreground objects. There are seven automatic global threshold methods commonly used in ImageJ/Fiji and with Scikit-image, a Python package, in image processing: Li, Triangle, Yen, Mean, Isodata, Otsu, and Minimum. Besides the difference between algorithms used to find the threshold value in a chosen image frame, thresholding can also be categorized into global and local thresholding.

For global thresholding, only one threshold value is calculated from the entire image frame based on the image histogram, which means the threshold value is constant. Global thresholding assumes that the image has a bimodal histogram as shown in the figure below. The image intensity value in the deep valley between two peaks is determined to be the threshold value T. Pixels can be simply divided to either objects or background by comparing them with T. Global thresholding is computationally fast and it exhibits good performance if the pixel intensity is uniform on the image background.

However, global thresholding fails to give correct threshold if the histogram does not have a bimodal distribution due to the object area being too small compared with the background area, or if the background or objects pixel intensities are not even across the image, which means relatively large variances compared to

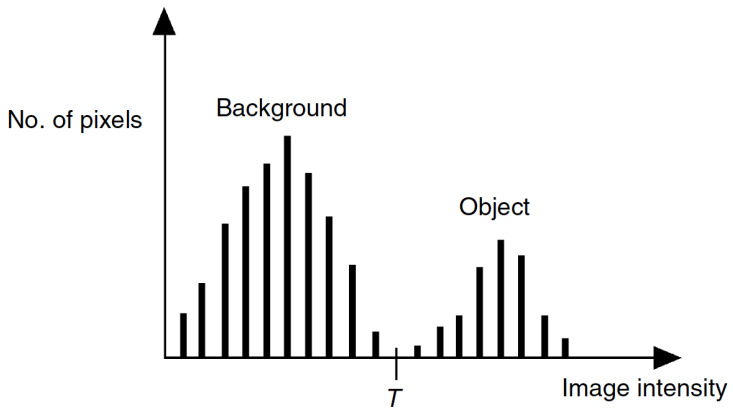

FIGURE 1.2: An example of a bimodal histogram with the threshold value T (Bankman, 2009).

the mean difference (the difference of the average intensities of the object and the background), or if the image is noisy (Cheremkhin & Kurbatova, 2019; Kittler & Illingworth, 1985).

When the image background is uneven or the noise level is high, the peaks of the objects and the background can overlap, which leads to the problem of imprecise global thresholding. Local thresholding is considered to be a better approach in this case. Local thresholds are determined by splitting an image into multiple overlapping sub-regions with an arbitrarily chosen radius and computing a threshold value on every sub-region with different algorithms. Some algorithms calculate histograms for each sub-images and selected the minimum intensity between the background and objects' peaks as the local threshold if the histogram is bimodal and perform interpolation from the local thresholds determined on neighbor sub-images if the histogram is unimodal (Bankman, 2009). Others compute the mean intensity within the sub-images and different statistics can be applied based on the local intensity distribution (Bankman, 2009; Leszczyński et al., 2016). Pixels that have intensity values greater or equal to the local threshold are considered as objects and become white,

otherwise, they will be considered as the background and become black. In order to make the thresholding accurate, the radius needs to be chosen carefully so that the sub-images can be large enough to include both object and background pixels (Bankman, 2009; Yan et al., 2005).

In this chapter, I will introduce the tools and methods used in my thesis. A description of the images used to build *ifThresholds*, and a description of the software tool and packages that are incorporated in *ifThresholds* will be provided.

2.1 Image Data: OGD Study Confocal Images

Images were obtained from an experiment performed in our lab, where brain slices were exposed to oxygen glucose deprivation (OGD). OGD in cultured brain slices is representative of an *in vivo* ischemic condition and results in loss of cells and generation of reactive oxygen species. Our lab uses the OGD slice model to screen therapeutics of interest to determine promising therapeutic platforms to then evaluate *in vivo* (Joseph et al., 2020). Our lab performs imaging on these slices to look at the cellular responses to treatment and injury. Various experimental conditions can be compared and analyzed in a short period of time using automated microscopy and image analysis, thus evaluating the morphology of cells systematically. Thresholding, one of the image analysis methods, converts a grayscale image to a binary image which greatly downsizes the image file size while still keeps the morphological features of cells.

In this study, I utilized images obtained from four treatment groups: slices exposed to 0.5 h OGD, 1.5 h OGD, and 3 h OGD, and an OGD exposed slice treated with SOD, a therapeutic enzyme. There was one control group, a healthy non-OGD exposed slice. In the experiment, the cultured brain slices were incubated overnight followed by removing culture media and adding fresh media. After another two days of incubation, the slices without OGD medium exchange were considered as normal control conditions. Different groups of slices were exposed to OGD for 0.5 h, 1.5 h, and 3 h after the end of the incubation. In one of the groups, 100 µL 1x PBS containing 0.1mg SOD (Cu/Zn SOD1 from bovine erythrocytes, Sigma) per slice was added directly to the 3h OGD slices (Liao et al., 2020). All the samples were returned to the CO2 incubator until further processing. At 24 h after OGD exposure, slices were fixed and stained with fluorescent antibodies for different cell populations, including microglia: slices from all sample conditions were stained with 5 µg/mL propidium iodide (PI), fixed, and stained with 4,6-diamidino-2-phenylindole (DAPI, Invitrogen, 1:10,000), goat anti-Iba1 antibody (Wako 019-19741, 1:200) and anti-goat AlexaFluor 488 (Life Technologies A11034, 1:500) for microglia (Joseph et al., 2020).

In the end, slices were imaged using Nikon A1R confocal microscope at 40× magnification (Joseph et al., 2020). Five images were obtained from each brain region of interest for every prepared slice. All the images were exported in TIFF image format and stored. Gray images were extracted from the blue channel (which contains DAPI stains) of the original TIFF image stack files. All the images have a dimension of 512×512 pixels and file size of about 0.5 MB. Then, images were manually labeled with image processing software Fiji. The sample image that is used for the following chapter shows a 40X magnification of cells from the thalamus with the treatment of 0.5 hour oxygen glucose deprivation.

2.2 Image Processing Software: Fiji

To manually label the cells in an image in Fiji, I proceed as the following steps:

- Import the TIFF image stack and choose one image frame

to label.

- Use the freehand tool in Fuji to surround a cell in the image.
- When done, add the contour to the ROI (region of interest) manager by pressing "t".
- Repeat the previous two steps for all cells in the image.
- For generating the labeled image, select Edit - Options - Color to set the Fiji foreground color to black.
- Then select the complete image with Edit - Selection - Select All followed by clicking Edit - Fill to fill the entire image in black.
- Change the foreground color to white to make all the cell regions white.
- Save the black-white mask image as the labeled image.
- The ROI manager gives the number of counters which is the number of cells in the image from manually labelling and counting.

0.5-hour OGD images is chosen as an example. Figure 2.1 shows the original TIFF image and the manually labeled image, which resulted in a manual count of 133 cells read from the ROI manager.

2.3 Python Language

The Python language has been chosen to develop *ifThresholds* package. Python is an efficient scripting language that includes the main concepts of object-oriented programming and provides readable code. What's more, it is easy to learn and use, available in the open-source environment, and has numerous resources for troubleshooting. Furthermore, a large set of libraries was developed and available in Python as open-source programs. *IfThresholds* takes advantage of this library set and implements the dedicated libraries for specific applications. I used scikit-image Python library to write the core code of *ifThresholds* package, along with the scientific Python modules, SciPy and Numpy.

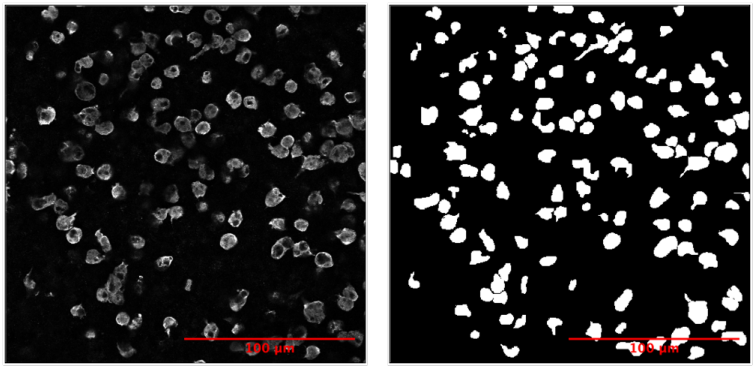

FIGURE 2.1: The original 0.5-hour OGD image (left) and the manually labeled image (right) that were used as a sample input to the *ifThresholds* package.

3.1 Motivation

IfThresholds is used to score and assess thresholding methods for fluorescent images of brain cells. The threshold ranking will be based on manually labeled images and images produced by applying the seven threshold methods mentioned previously. *IfThresholds* guides researchers on which threshold method is the best one for their images.

3.1.1 Compare Different Global Thresholding

Different global thresholding methods give greatly different image analysis results for some types of images. Some of the thresholding methods work better with larger objects while others are more resistant to image noises. As shown in Figure 3.1, while applying all 7 threshold methods on the sample image, the minimum method was not able to separate cells from the background while the mean method considered all the noises on the background as objects.

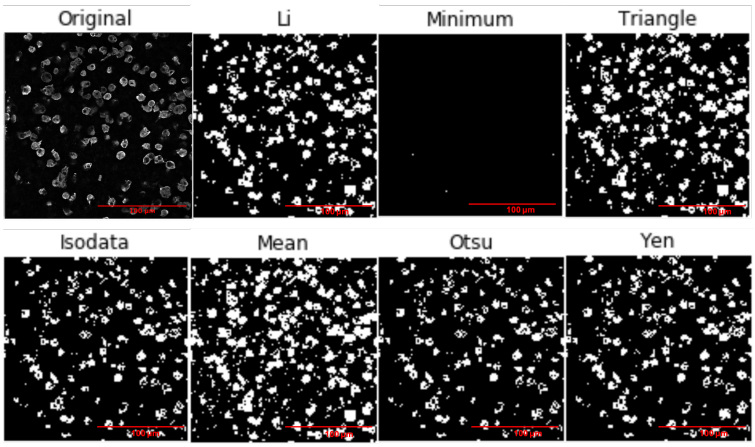

FIGURE 3.1: A series of binary images generated by Scikit-image built-in thresholding.

3.1.2 Compare Global and Local Thresholding

This section shows a comparison between image outputs obtained from implementing global and local threshold methods followed by comparing using different radius sizes in local threshold methods. We can apply both global and local Otsu threshold methods to the same image and see the difference between them. As shown in Figure 3.2, a block size (odd size of pixel neighborhood) of 35 pixels are chosen for local Otsu thresholding, we can tell that the global Otsu thresholding already gives a very clear image compared to the original and manually labeled images owing to the uniform background and fairly low noise level, local thresholding does show better details on shapes and boundaries of cells on certain regions on the image.

As mentioned previously that local thresholding is sometimes more precise and the radius of the sub-images can affect the outcome as well. The global Otsu thresholding is commonly used but sometimes local thresholding with an appropriate radius determined based on cell size and distribution can be more accurate as shown in Figure 3.3. Logically, as we increase the radius which means using larger overlapped sub-images to compute local

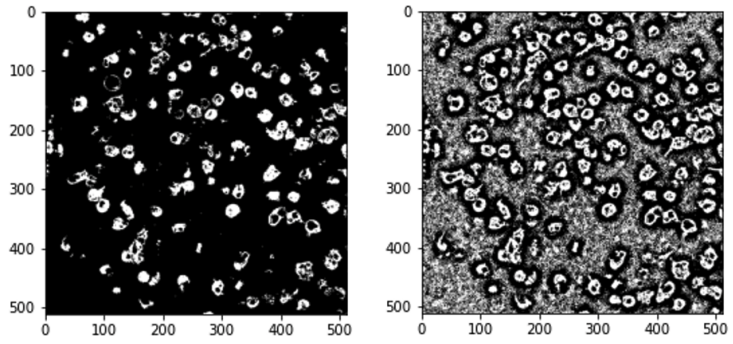

FIGURE 3.2: Comparison between global and local Otsu threshold methods: binary global thresholding (left) and binary local thresholding (right) applied on the original 0.5-hour OGD image. Block size of 35 is chosen for the threshold local function. The image has a size of $520x520$ pixels.

thresholds, we will have a result image more similar to the global thresholded image.

3.2 Package Development

The *ifThresholds* package can take an input of an original cell image, a corresponding manually labeled cell image, and a manual count of cell numbers and return a table with scores and rank of all seven standard threshold methods. Within the package, binary images are generated by applying all the standard threshold methods and compared with the manually labeled image with 3 different scoring methods: count method, overlap method, and area method. Each scoring method will give a score of percentage off which indicates how much the image generated by the threshold method is different from the manually labeled image. To increase reliability and refine the package, I ran multiple sets of images with different experimental settings and different stains. All the scores were written to a score table in a CSV file and later generated to a score ranking table. Besides a recommendation given based on the OGD images, users can also get a score table

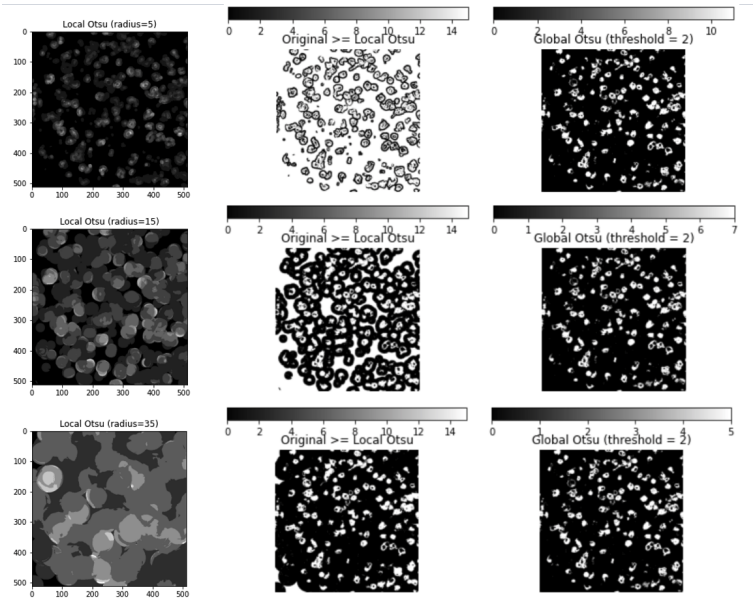

FIGURE 3.3: Comparison between local threshold methods with different radius chosen: 0.5-hour OGD image is used and the image has a size of 520 x 520 pixels. Three radius values 5, 15, 35 were chosen to make the comparison. The first image on each row shows the local threshold determined on every sub-images. The second image shows the pixels that belong to the objects (pixel intensity >= local threshold). The last image is processed with the global Otsu thresholding method with only one global threshold value.

and a recommendation from terminal/console for their own images .

I used Scikit-image Python library to write the core code of *ifThresholds* package, along with the scientific Python modules, SciPy and Numpy. I first calculated a list of different thresholds with Scikit-image built-in functions. Binary images were produced based on these thresholds. Threshold methods vary in their ability to (1) distinguish large cells from clusters of small cells, (2) define an optimal threshold, and (3) see the cell with holes as a whole cell.

Before comparing the binary image to the manually labeled image, the binary image was pre-treated to minimize the errors that could be caused by unknown noise. Figure 3.4 shows an example of a binary image with a threshold calculated by Li method and the images after being pre-treated. The two pre-treatment steps were taken to remove small objects from the image and filling holes of each object. The pre-treatment step was necessary in order to clean up the noise in the background, isolate objects from the background and from each other, and minimize the difference between processed images. The binary images were then fed into three different scoring methods.

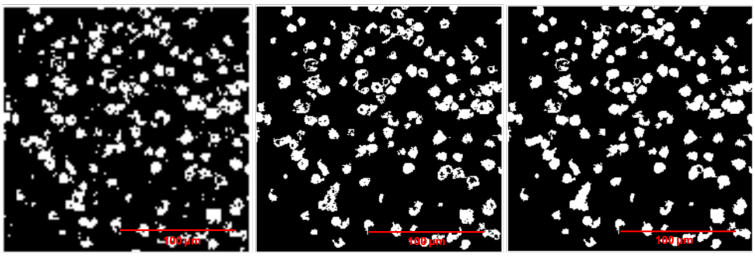

FIGURE 3.4: Pre-treatment of the image: original binary image produced by Li threshold method (left), after removing small objects (middle), and after filling holes of each cell (right).

I wrote three scoring methods: (1) the count scoring method, (2) the overlap scoring method, and (3) the area method. The count scoring method computes the difference between the number of cells labeled by the automatic threshold method and the manual count of cells in a format of percentage off. The overlap scoring method stacks two images and looped over the entire frame to count the number of pixels not overlapped and gave a percentage of non-overlapped pixels in the image. The area method computes the mean percentage difference of cell size between the thresholded image and the manually labeled image. For instance, there are 123 cells in the manually labeled image, but only 111 cells in the image generated by the certain threshold method. 111 cells from both images will be sorted based on the single-cell areas. A percentage difference on the area of every pair of cells would be computed and an average percentage would be the final output.

There will be 21 scores obtained from 7 threshold methods and 3 scoring methods for every input image. Next, all the scores obtained from different scoring method were written to a CSV file and read with Python library Pandas for further analysis. Scores from different scoring methods were weighted equally to become one score for one threshold method. As I manually labeled more images and obtained more score tables from the *ifThresholds* package, the score for every threshold method in every image will also be averaged out assuming every image input is an individual data point. At the end, a score ranking table will be obtained and the threshold method recommendation will be given.

The core code of *ifThresholds* package is provided in a form of pseudo-code. The main function is used to take inputs, call other functions, allow the terminal interface, and display error messages or outputs. One function calculates the threshold value and corresponding scores while the other one gives the recommended threshold method.

```
def img_score(image_path, manual_path, man_count):
   read image from the image path
   convert the image to grayscale image, img
   % obtain threshold values from skimage.filters
   threshold = [li, minimum, triangle, isodata, mean
       , otsu, yen]
   scorelist = []

   for i in range(0,7):
      score = []
      threshold_value = threshold[i]
      create a binary image/thresholded image
      pretreatment of the image: remove noise and
          fill holes
      apply regionprops

      % count method
      countmethod = ('count method', countscore)
      score.append(countmethod)
      % overlap method
      overlapmethod = ('overlap method', overlapdiff
          )
      score.append(overlapmethod)
      % area method
```

```
22          areamethod = ('area method', areadiffmean)
23          score.append(areamethod)
24          scorelist.append(score)
25
26      return scorelist
27
28  def whichBest(scores):
29      calculate average scores for 7 threshold methods
            and find the minimum score
30      minScore = min(li, minimum, triangle, isodata,
            mean, otsu, yen)
31      if score == minScore:
32          print the recommended threshold method with
                its score
33
34  def main():
35      takes image paths and manual count inputs from
            users on the terminal
36      gives error message if the manual count input is
            not an integer or when the users type
            unexpected command
37      scores = img_score(image_path, manual_path,
            man_count)
38      whichBest(scores)
39      if users want the score table in csv format:
40          write data into csv file and save in the
                current directory
41      if users do not:
42          exit the program
43      % when the user input is either yes or no, but an
             unexpected request
44      else:
45          gives error message
46
47  # call the main function once the python file is
        called.
48  main()
```

3.3 Results

In order to improve reliability and optimize score ranking table, I manually labeled and ran multiple sets of images with different

experimental settings and different stains. A final score ranking table was generated as shown in Table 3.1. Isodata and Otsu methods are ranked first and second with a score of 11.74 and 12.11 correspondingly which means that the images applied with Isodata or Otsu threshold methods are 11.74 or 12.11 percentage off from the manually labeled images. In addition to recommendations based on OGD images, users can also obtain score tables and recommendations for their own images from the terminal/-console. Follow the steps in this section or on the original GitHub repository (Zhou, n.d.) to run *ifThresholds* from terminal/console. An example of the terminal/console interface is given at the end of this section. Yen method was recommended as the best threshold method to use for this single image based on the original image, the manually labeled image, and the manual count inputs. A score table was saved locally in the same directory where the file main.py locates and also displayed on the terminal directly for a quick reference.

Threshold Method	Mean Score	Rank
Isodata	11.74	1
Otsu	12.11	2
Yen	16.38	3
Li	22.01	4
Triangle	41.18	5
Mean	45.26	6
Minimum	77.09	7

Table 3.1: Score ranking table

Steps to run *ifThresholds* from terminal/console:

- Clone the *ifThresholds* to your local directory. git clone https://github.com/Nance-Lab/ifthresholds.git
- Create the environment to run *ifThresholds*. conda env create -f environment.yml
- Save the original and manually labeled images you want to analyze on the SAME directory where main.py locates.

- Three inputs will be required: a path to the original image, a path to the manually labeled image, and the manual count of the cell number in the image
- Go to the directory where main.py locates
- Run *ifThresholds* on terminal/console and follow instructions. python main.py

Terminal Interface:

```
% python main.py
  Are you ready to find the most suitable threshold
      method for your medical images? ifThresholds
      can help you to find the best one among li,
      minimum, triangle, isodata, mean, otsu, and
      yen threshold methods.
  What is the path to your original TIFF image?
      slice_OGD-0.5h_40x_thalamus_1.tif
  What is the path to your manually labeled TIFF
      image?M_OGD-0.5h_40x_thalamus_1.tif
  What is the number of cells in your image based on
       your manual count?133
  The recommended threshold method is Yen with a
      score of  17.19757397040942 which means the
      thresholded image is  17.19757397040942
      percentage different from the manually labeled
       image.
  Do you want to obtain a score table of your image
      saved in your current directory (y) or exit
      the program (n)?y
  The score table scoretable.csv is saved in your
      current directory! Thanks for using
      ifThresholds!
     Threshold_method    score_method      Score
  0                Li    count method    18.796992
  1                Li     area method    21.068199
  2                Li  overlap method    18.516159
  3           Minimum    count method   100.000000
  4           Minimum     area method     0.000000
  5           Minimum  overlap method    15.195084
  6          Triangle    count method    20.300752
  7          Triangle     area method    22.468224
  8          Triangle  overlap method    20.816422
```

9	Isodata	count	method	21.804511
10	Isodata	area	method	16.878993
11	Isodata	overlap	method	16.408157
12	Mean	count	method	20.300752
13	Mean	area	method	21.949681
14	Mean	overlap	method	20.593262
15	Otsu	count	method	21.804511
16	Otsu	area	method	16.878993
17	Otsu	overlap	method	16.408157
18	Yen	count	method	17.293233
19	Yen	area	method	17.148610
20	Yen	overlap	method	17.150879

Conclusion and Future Directions 4

IfThresholds assessed and compared 7 global threshold methods that are commonly used and easily accessible in various image processing software with images obtained from the experiment performed in our lab. The score ranking table can give researchers and scientists insights into how well each threshold method performs in images, thus help them choose the best threshold method. With more and more images and corresponding scores being added to the score table, the ranking will be more reliable and accurate. On the other hand, users can also run *ifThresholds* on the terminal locally to analyze their own image and get the score table saved locally as well.

The current package implemented 3 scoring methods to compare images from 3 different perspectives, however, more scoring methods can be potentially added to make better comparisons between thresholded images and manually labeled images. Besides adding more scoring methods to the current package, there are also three major directions that can make *ifThresholds* a more reliable and helpful software. First of all, machine learning can be incorporated to weight scores from different scoring methods assuming that certain scoring methods will be more accurate to evaluate images than others. Machine learning models can also be applied to make predictions on which threshold method will

be recommended for a set of images based on only a few image inputs which can greatly decrease the computing time and space. Secondly, building a user-friendly web application for better data visualization and user interface could be the next step. Instead of running *ifThresholds* locally on the terminal for users' own image assessment, a web application deployed and hosted in a server will shorten the run time and improve the user experience. Lastly, an additional feature of helping users find the proper color channel with the most targeted information should also be implemented to expand the user base.

4.1 Machine Learning

In the current *ifThresholds* package, I weighted scores from different scoring methods evenly since these scoring methods compared images with totally different algorithms from different perspectives and contributed evenly to the final score. However, in some scenarios, one scoring method could give fairly consistent scores on all the threshold methods while the other one might provide scores in a wilder range. For example, when some threshold methods fail to recognize a cluster of small cells from a large cell, it might hugely affect the score obtained from score methods thus affect the averaged score at the end while scores from the overlap method will not change too much. Ideally, the more images we feed into the package to train the data, the better the weight ratio would become in order to have the final score ranking tables generalized broadly.

Supervised machine learning methods assume there is a function form that can describe the output vector (Y) given some input vector $(X) : Y = f(X) + error$ (Guillaumin et al., 2010). Sample input-output pairs (training dataset) are required to build a learning model for supervised. Since we will not have corresponding output variables (scores and ranks of threshold methods) for the input data (images), we can instead categorize images as 1 to 3 based on the similarity to the manually labeled image. By looking at images produced by various threshold methods, we can label images that look the most similar to the manually labeled one as 1, the least similar ones as 3, and the remaining images as 2. The scores from all the methods will be seen as

input X and the rank will be considered as output Y, thus the classification machine learning will be used given the targeted attribute's values (ranks) are categorical, which means that the predicted value from the machine learning model will be classified as 1, 2, or 3. For potential unsupervised learning, clustering could be used to group images with natural similarity to fewer groups and do further analysis. Association is another machine learning model that could help find two or more threshold methods that give similar scores.

Besides weighting scores, a decision tree can also be built to help users find the most appropriate method with only a few image inputs to save time from manually labelling more images. First, we need to extract good features, such as scores from different scoring methods or noise levels of the image, that can potentially give predictions of the final recommended method. This step can be done with open-source Python libraries. Then, we can build a decision tree with the Python Scikit-learn library DecisionTreeClassifier class and fit the tree model with the training samples and the class labels for these training samples (*1.10. Decision Trees*, n.d.). Class labels will be the names of threshold methods as what we need to predict. With the decision tree, we will be able to find the recommended threshold method by traversing the tree based on the known values of the features.

4.2 Web Application

In order to let users see the actual images that are applied with different threshold methods, I deployed a website on Heroku (*https://ifnew.herokuapp.com/*, n.d.) to visually present how images are different from others while applying different filters and threshold methods. The current *ifThresholds* package can be executed on the terminal locally but it might still be difficult for users with no programming background to download that package from GitHub and run command lines to obtain results. The run time is also limited by the users' machines. Therefore, composing a full-stack web application is needed to target more users and enhance the user experience.

In addition, we can also collect images paired with manually labeled images on the website to increase our image database so

that we can make our score ranking table more reliable and also improve the accuracy of the decision tree.

4.3 Multi-channel Images

Multi-channel images contain more information since each pixel contains information of intensity values at red, green, and blue channels. Binary images are derived from only one threshold value which leads to problems of small objects overlapping with background noise. When applying *ifThresholds* to color images, the biggest challenges would be selecting the proper color channel that provides suitable information for the target cell characteristics and applying multiple filters to remove noise and enhance the differentiation between objects and backgrounds (Intaramanee et al., 2017). For future development, *ifThresholds* should implement a function to separate three color channels and plot histograms of each channel. Channels that provide too bright or too dark images will not be considered as proper channels since objects with low contrast will be destroyed in such images. The proper channel that has the highest contrast among the three color channels will be extracted as the input image. Before feeding images to scoring methods, multiple filters might be needed if cells are tiny and have a considerable amount of overlap with background noise. The Gaussian filter can remove noise objects but also reduce the sharpness of the edges of objects while the median filter can help preserve the edges (Deng & Cahill, 1993). It will be crucial to select the proper color channel and apply appropriate filters before assessing images with scoring methods. By adding this feature, *ifThresholds* will be more useful to a broader base of users.

References

1.10. decision trees. (n.d.). Retrieved from `https://scikit-learn.org/stable/modules/tree.html`

Alivisatos, A. P., Gu, W., & Larabell, C. (2005). Quantum dots as cellular probes. *Annu. Rev. Biomed. Eng.*, *7*, 55–76.

Bankman, I. N. (2009). Elsevier. Retrieved from `https://app.knovel.com/hotlink/toc/id:kpHMIPAE02/handbook-medical-image/handbook-medical-image`

Bezdek, J. C., Hall, L., & Clarke, L. (1993). Review of mr image segmentation techniques using pattern recognition. *Medical physics*, *20*(4), 1033–1048.

Chen, X., & Murphy, R. F. (2004). Location proteomics: determining the optimal grouping of proteins according to their subcellular location patterns as determined from fluorescence microscope images. In *Conference record of the thirty-eighth asilomar conference on signals, systems and computers, 2004.* (Vol. 1, pp. 50–54).

Cheng, C.-H., Chen, Y.-S., & Lin, T.-C. (2007). Fcm based automatic thresholding algorithm to segment the brain mr image. In *2007 international conference on machine learning and cybernetics* (Vol. 3, pp. 1371–1376).

Cheremkhin, P. A., & Kurbatova, E. A. (2019). Comparative appraisal of global and local thresholding methods for binarisation of off-axis digital holograms. *Optics and Lasers in Engineering*, *115*, 119–130.

Chitradevi, B., & Srimathi, P. (2014). An overview on image processing techniques. *International Journal of Innovative Research in Computer and Communication Engineering*, *2*(11), 6466–6472.

Deng, G., & Cahill, L. (1993). An adaptive gaussian filter for noise reduction and edge detection. In *1993 ieee conference record nuclear science symposium and medical imaging conference* (pp. 1615–1619).

Drozdzal, M., Chartrand, G., Vorontsov, E., Shakeri, M., Di Jorio, L., Tang, A., ... Kadoury, S. (2018). Learning normalized inputs for iterative estimation in medical image segmentation. *Medical image analysis*, *44*, 1–13.

Eils, R., & Athale, C. (2003). Computational imaging in cell biology. *The Journal of cell biology*, *161*(3), 477–481.

Feng, Y., Zhao, H., Li, X., Zhang, X., & Li, H. (2017). A multi-scale 3d otsu thresholding algorithm for medical image segmentation. *Digital Signal Processing*, *60*, 186–199.

Fu, K.-S., & Mui, J. (1981). A survey on image segmentation. *Pattern recognition*, *13*(1), 3–16.

Guillaumin, M., Verbeek, J., & Schmid, C. (2010). Multimodal semi-supervised learning for image classification. In *2010 ieee computer society conference on computer vision and pattern recognition* (pp. 902–909).

(n.d.). Retrieved from `https://ifnew.herokuapp.com/`

Intaramanee, T., Khoeun, R., & Chinnasarn, K. (2017). Automatic microaneurysm detection using multi-level threshold based on isodata. In *2017 14th international joint conference on computer science and software engineering (jcsse)* (pp. 1–6).

Joseph, A., Liao, R., Zhang, M., Helmbrecht, H., McKenna, M., Filteau, J. R., & Nance, E. (2020). Nanoparticle-microglial interaction in the ischemic brain is modulated by injury duration and treatment. *Bioengineering & translational medicine*, *5*(3), e10175.

Kittler, J., & Illingworth, J. (1985). On threshold selection using clustering criteria. *IEEE transactions on systems, man, and cybernetics*(5), 652–655.

Leszczyński, B., Gancarczyk, A., Wróbel, A., Piątek, M., Łojewska, J., Kołodziej, A., & Pędrys, R. (2016). Global and local thresholding methods applied to x-ray microtomographic analysis of metallic foams. *Journal of Nondestructive Evaluation*, *35*(2), 35.

Liao, R., Wood, T. R., & Nance, E. (2020). Superoxide dismutase reduces monosodium glutamate-induced injury in an organotypic whole hemisphere brain slice model of excitotoxicity. *Journal of biological engineering*, *14*(1), 1–12.

Pal, N. R., & Pal, S. K. (1993). A review on image segmentation techniques. *Pattern recognition*, *26*(9), 1277–1294.

Petrou, M., & Petrou, C. (2010). *Image processing: The fundamentals*. Wiley. Retrieved from `https://books.google.com/books?id=w3BpSIxN9ZYC`

Russ, J. (2012). *Computer-assisted microscopy: The measurement and analysis of images*. Springer US. Retrieved from `https://books.google.com/books?id=Q1vlBwAAQBAJ`

Shnayderman, A., Gusev, A., & Eskicioglu, A. M. (2006). An svd-based grayscale image quality measure for local and global assessment. *IEEE transactions on Image Processing*, *15*(2), 422–429.

Xue, Y., Xu, T., Zhang, H., Long, L. R., & Huang, X. (2018). Segan: Adversarial network with multi-scale l 1 loss for medical image segmentation. *Neuroinformatics*, *16*(3), 383–392.

Yan, F., Zhang, H., & Kube, C. R. (2005). A multistage adaptive thresholding method. *Pattern recognition letters*, *26*(8), 1183–1191.

Zhou, Q. (n.d.). *Nance-lab/ifthresholds*. Retrieved from `https://github.com/Nance-Lab/ifthresholds`