VIP IPA
Nuclei Detection and Counting Team

Final Report
Spring 2022
April 29, 2022

Tikhon Pachin, Yu-Hsuan Lin, Annapoorna Prabhu,
William Stevens, Shang-Hsuan Lee

Mentors: Prof. Edward J. Delp, Prof. Carla Zoltowski

Department of Electrical and Computer Engineering
Purdue University

Table of Contents:

I. Abstract

The Nuclei segmentation and counting team has managed to successfully implement filtering tools for image segmentation and counting preprocessing, which has been a crucial part of microscopy research for over 10 years. This semester's work on the project covers the finalization of the watershed segmentation logic, optimization of the current processes, and construction of the cell classification neural networks.

Watershed segmentation is used to visualize a microscopy image as a topographical map and analyze the relief to place watersheds around the edges of the basins. Those edges represent cell borders on the original image. The preprocessing steps include the areas of interest detection algorithm, image to topographical map transformation, and basin minima detection algorithm. The watershed segmentation logic involves image flooding, overflow protection and image analysis for watershed placement. Classification logic mostly includes feature extraction algorithms, loss function selection and construction of the convolutional neural network.

Image preprocessing, such as areas of interest detection algorithm and topographical map extraction, can be found in the works of the team from the Fall 2021 semester. This semester's preprocessing work was focused on the K-means regional minima detection algorithm with the addition of the H-minima transform, partial image analysis utilizing a connected components algorithm, flooding and overflow protection for border detection, and construction of a convolutional neural network for cell classification.

The expected output of the current semester's work is a generalized algorithm to segment cells on any microscopy image as well as a classification algorithm to identify different types of cells with a potential anomaly detection feature.

II. Introduction

Goal

Cell segmentation, counting, and classification are on the forefront of medical imaging technology, and such methods have been shown to improve the speed and accuracy of medical diagnoses. While there exist fully developed algorithms that are used professionally, there are always ways to improve the efficiency and accuracy. The goal of the project this semester builds off of the goal from last semester, which was to develop an algorithm that effectively segments and classifies cells in microscopy images. The first main objective of this semester was to develop additional image processing algorithms that would allow for accurate segmentation of heavily clustered cells, since there were under-segmentation issues in such regions last semester. The other objective for this semester was to incorporate algorithms that handled aspects of cell classification, specifically binary classifications of microscopy images that would indicate healthy or infected states.

Related work

The team considered multiple segmentation algorithms last semester, some of which required significant preprocessing for the cell borders to be accurately detected. Upon further reading, it was found that watershed segmentation seemed to work the best for microscopy images. Some watershed segmentation methods that were looked into were Meyer's method (called the Priority Flood) [1] and the Vincent and Soille watershed process [2]. Meyer's method makes use of predefined markers, which serve as points where flooding starts. The markers each have a label associated with them. The neighbors of each marked pixel (defined by their connectivity) are then added to a priority queue, where pixels with the higher magnitude of gradient get the higher priority [1]. The pixels in the queue are processed according to their priority - the neighbors of the pixels in the queue are labeled and then added to the queue. This process of labeling neighboring pixels and adding them to a queue is then continued till the queue is empty. The unlabeled pixels are considered a part of the watershed lines. The Priority-Flood method needs predefined markers for which additional preprocessing would have to be done on the image.

The Vincent and Soille watershed process also makes use of an ordered queue approach. For space and time considerations, the pixels are sorted by gray level and the cumulative frequency distribution of the pixels by gray level is calculated, which makes the sorting process easier [2]. For a flooding level of h, every catchment basin formed has its own minimum with a "height" that is lesser than h, and these minima and the area around them are given a unique label. When the height of the "water" has been raised by one unit, new catchment basins are formed and some basins formed at height h may merge and the labels associated with pixels

might change. To reassign the labels at height h + 1, another scanning is done using an ordered-queue approach similar to that used in the Priority-Flood method [2].

The segmentation algorithm proposed and implemented this semester is similar to the approach taken by Vincent and Soille process wherein regional minima function as the points where flooding first begins. However, the priority queue used to calculate the frequency distribution is not used, and instead, regional minima are detected using various methods.

III. Technical background

Cross-correlation and convolution

The differences between cross-correlation and convolution are described in the team's final report of Fall 2021 [3]. A quick review is presented further.

Convolution and cross-correlation are represented mathematically by the following formulae:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)dt$$

Equation 1. Convolution formula [4].
f(τ) - first signal input, g(t) - second signal input,
*(f * g)(t) - output signal produced by convolving two input signals.*

$$(f * g)(t) = \int_{-\infty}^{\infty} \overline{f(\tau)}g(t+\tau)dt$$

Equation 2. Cross-correlation formula [4].
f(τ) - first signal input, g(t) - second signal input,
*(f * g)(t) - output signal produced by cross-correlating two input signals.*

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t+\tau)dt$$

Equation 3. Cross-correlation in the real domain [3].
f(τ) - first signal input, g(t) - second signal input,
*(f * g)(t) - output signal produced by cross-correlating two input signals.*

Mistakenly, it is common to use the term convolution instead of cross-correlation. The key differences between the two reside in the integrands of the function. Cross-correlation integrates the complex conjugate of the *f* function, whereas convolution does not. Additionally, the *g* function is represented differently. In convolution, the *g* function in the integrand is flipped about the y axis. Since our operations are completed in the real domain and the axis manipulation

does not bring any significance to image processing operations of this paper, we decided to use cross-correlation for the sake of simplicity. Throughout this paper you may encounter the term "convolution" loosely used in place of cross-correlation, however, no operation is truly convolutional.

The following is a visual representation of true convolution taken from the team's Fall 2021 report:

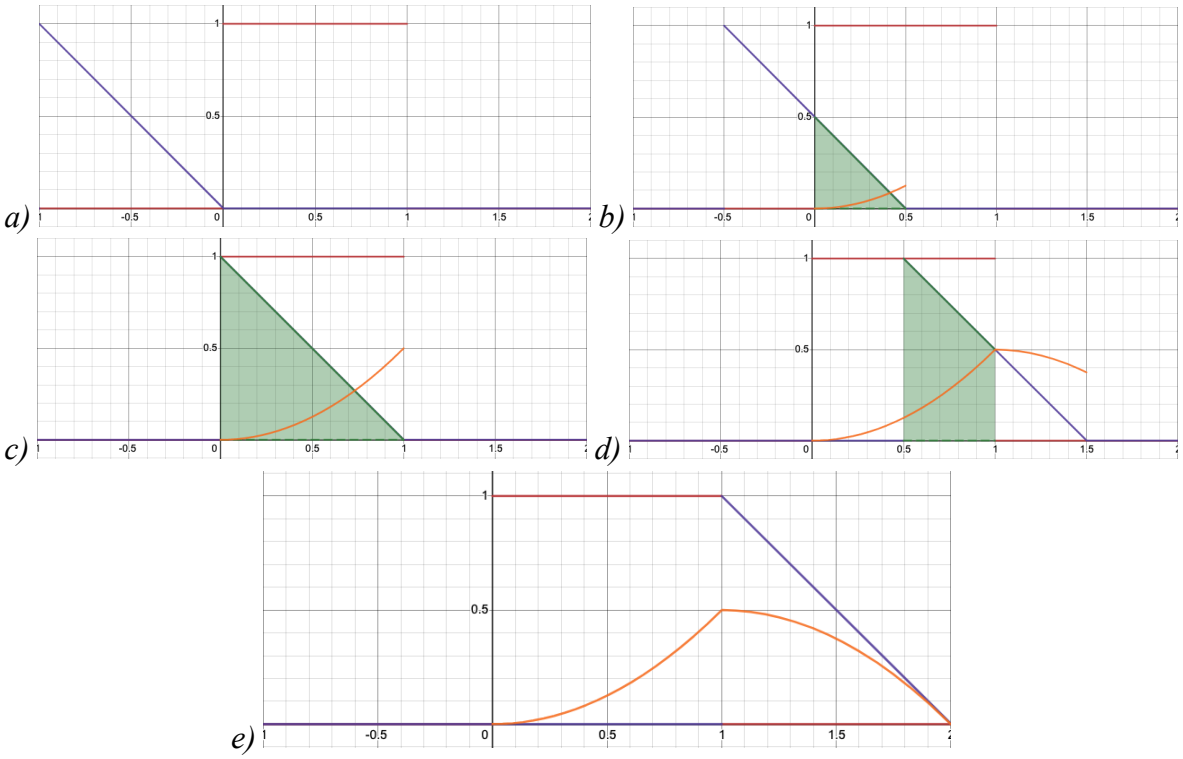


Figure 1. Convolution visualization [3].
Red – $f(\tau)$ input, blue – $g(t-\tau)$ input, orange – $(f*g)(t)$ output.
a-e) Convolution process.

The following is an example of cross-correlation used as a matrix operation:

100	17	166	34	222	167
84	23	90	194	25	56
40	7	55	123	200	1
34	63	87	39	127	54

*

1	1	1
0	0	0
1	1	1

=

385	402	800	747
381	496	562	495

Figure 2. 2-dimensional cross-correlation example [3].

The following is an example of a cross-correlation operation called average blurring:

Structuring element:

$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$

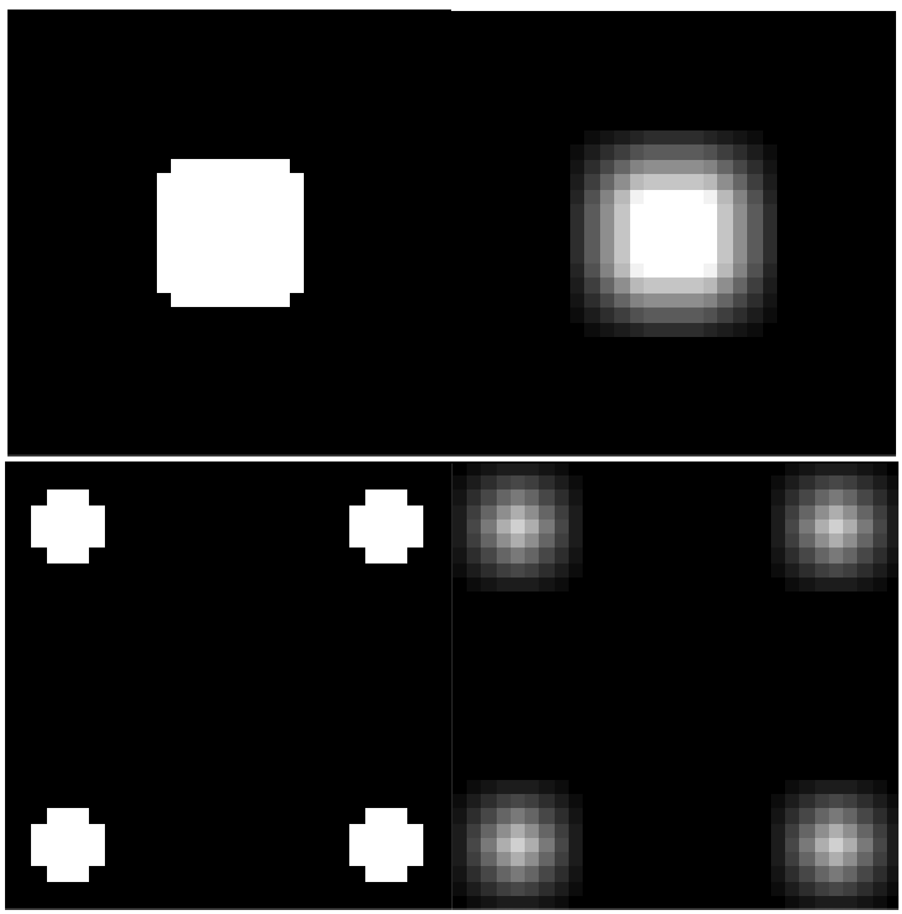


Figure 3. Average blurring example [3].

Watershed Segmentation

As with last semester, the method of watershed segmentation is used to tackle the cell and nuclei segmentation problem. A summary of this method is presented below.

An image can be thought of as a topographical relief map, with the intensity levels representing the altitudes. For this project, we will be working with grayscale images, and therefore, the intensities will correspond to the gray level of the image. The idea behind watershed segmentation is that when water begins to pour onto the topographic surface, different regions with water called catchment basins are created, creating segments in the image. The points where the water first begins to pool are referred to as the regional minima of the image. So one can refer to a region around a minimum in the image as a valley. These regions do not have as high intensity as a plateau or mountain peak, for example. The set of pixels that contain the water that does not touch other regions of the image is known as a catchment basin and the edge along which two catchment basins are separated is called the watershed line [5]. The watershed lines and the catchment basins cause the image to be segmented. As laid out by Meyer [1], the flooding method starts when the water begins to fill each regional minimum, going from low-lying regions of the image till the highest point of the image. As more water pours into the catchment basins, the water level increases, and when the water level is at the height of the basin, a watershed (or a dam) is placed to avoid overflow into other catchment basins. Without the placement of the watershed lines, catchment basins would eventually merge into a single segment, which is undesirable. This process stops when all the catchment basins are filled with water.

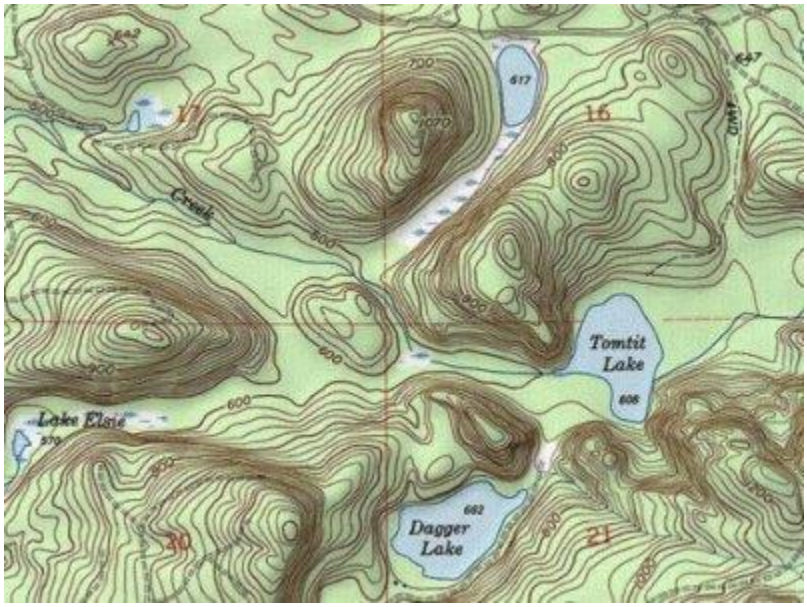


Figure 4. Example of a topographical relief map [6].

We have modified the watershed segmentation process (as outlined in Figure 5) from Fall 2021 to account for overflow protection as well as improved regional minima detection. The image is first preprocessed to detect regional minima in the image, which will serve as points from where the flooding takes place. Preprocessing includes conversion of the image into a grayscale image and applying an areas of interest detection algorithm that extracts all the connected cell sections from the image. A Euclidean distance transform follows the inversion of the obtained binary image, which results in a topographical map of the image. Regional minima in the image are detected by a combination of H-minima transform and K-means clustering. Then, flooding takes place and all catchment basins are defined at the end of this process. The watersheds are planted around the catchment basins and some post-processing is done to get the final segmented image.

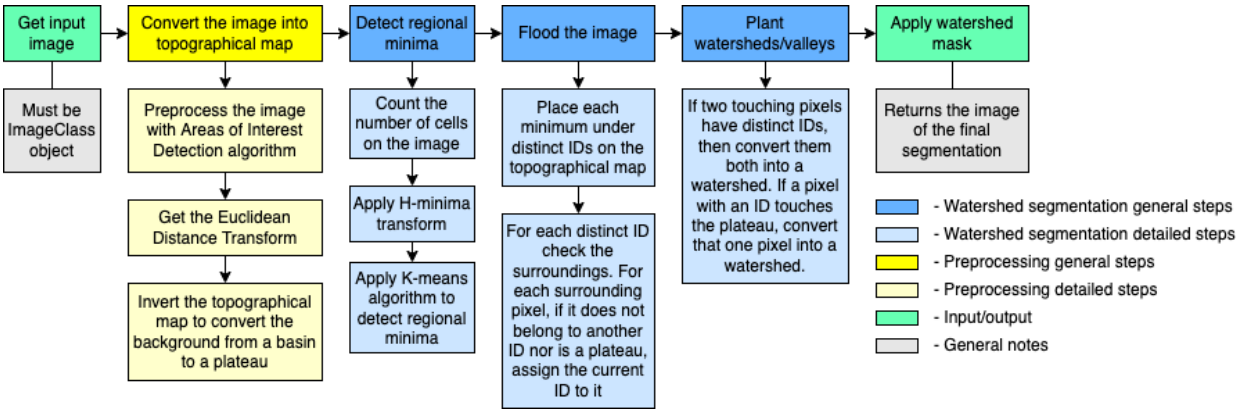


Figure 5. Watershed segmentation overview.

Machine learning

Machine learning is a type of artificial intelligence process which focuses on using data and algorithms to replicate the way humans learn and gradually improve the accuracy of prediction.

There are different kinds of learning models in machine learning; supervised learning, unsupervised learning, semi-supervised learning, etc. In supervised learning, the machine is given a training dataset with ground truth. The algorithm learns the key characteristics within each point in the dataset in order to make correct predictions. This method may be used if one wants the machine to identify the given classes. In unsupervised learning, the machine is given a training dataset without ground truths. Provided with a large amount of data, the machine would be able to identify trends of similarity after training. This method may be used to identify the cells that do not correspond to the trend in the image, if it is assumed there are a limited number of cell types and that we want to cluster them together. Semi-supervised learning is a combination of supervised learning and unsupervised learning. The machine will be given a training dataset with only a few data points that have ground truth. The algorithm will first use

unsupervised learning to identify groups with similar trends, then perform supervised learning with the data points that have ground truth in the cluster to provide labels to the entire cluster [7].

Neural networks are used to create supervised learning [7]. The ultimate goal of training the neural network is to find the combination of weights that results in the smallest value of error which is determined by a loss function. We can use the idea of gradient descent to update the weights of each neuron. A gradient indicates the direction and the adjusted rate to reach the local maximum of a function, which is the loss function in our case. However, the goal is to minimize the loss of the neural network, so the direction adjustments should be opposite from the gradient. For example, let Z be a loss function L of a neural network which has 2 variables (Y_1, Y_2). Each of the two variables is a weighted sum from the previous layer, shown as Equation 4:

$$Y_j = \sum w_{ij} * X_i$$

Equation 4. Example expression of variables Y_j in a layer [7].
j - variable index in current layer
i - variable index in previous layer
ω_{ij} - the weight with respect to X_i and Y_j

In order to adjust the weights for the variable X_1, we need to utilize the chain rule to get the gradient of Z with respect to X_1, shown as Equation 5. Then we can further simplify the gradient of Z as Equation 6:

$$\frac{\partial Z}{\partial X_1} = \frac{\partial L}{\partial Y_1}\frac{\partial Y_1}{\partial X_1} + \frac{\partial L}{\partial Y_2}\frac{\partial Y_2}{\partial X_1}$$

Equation 5. Example of the gradient of Z with respect to X_1 [7].
`∂L represents ∂L(`Y_1, Y_2).

$$\frac{\partial Z}{\partial X_1} = \frac{\partial L}{\partial Y_1}w_{i1} + \frac{\partial L}{\partial Y_2}w_{i2}$$

Equation 6. Simplification of the gradient of Z with respect to X_1 [7].

With the derivation above, we can conclude that to obtain the gradient of the loss function, we must reuse the gradient of the previous layer and multiply the gradient of that layer. To update the weights for training, we continue propagating from the output layer through the hidden layer up until the input layer. This entire process is called back propagation [7].

IV. Cell segmentation

Areas of interest detection algorithm

The areas of interest detection algorithm was developed by the team in Fall 2021. Since it plays a crucial role in our watershed segmentation implementation, a quick review is presented here.

One of the challenges of the watershed segmentation algorithm is determining which parts of the image are cells and which are the background. Once the image is converted into a binary format, each pixel is assigned a label, feature or non-feature. A feature element is the one that contains information (255 or 1, represents the presence of a cell) and a non-feature element is the one that does not (0, represents the absence of a cell).

Upon the first glance it may be assumed that a simple Otsu's thresholding will suffice to correctly extract feature and non-feature elements, however, the assumption was proven invalid [3]. Otsu's thresholding is a conceptually straightforward, but quite sophisticated algorithm that is discussed in great detail in the team's Fall 2021 report.

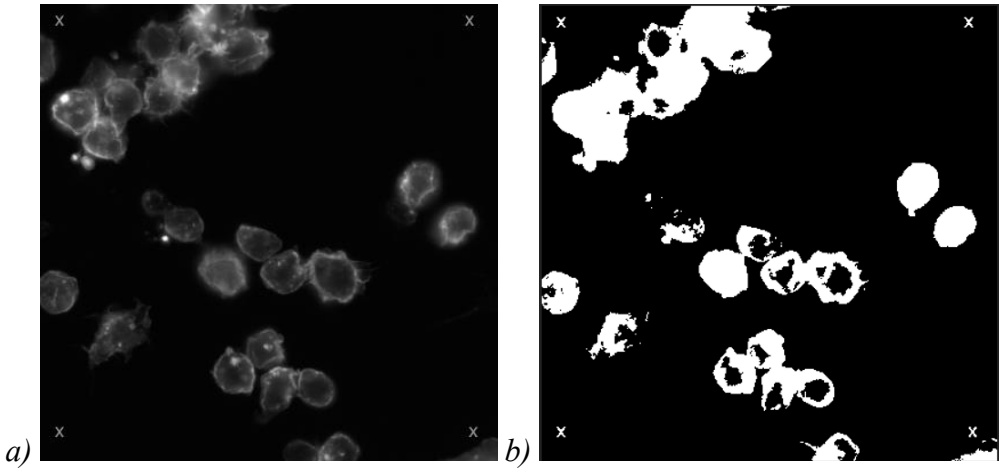


Figure 6. Otsu's thresholding output example [3].
a) Original image. b) Otsu's thresholding method output.

In the images above, it is clearly seen that simple Otsu's thresholding leaves out the internal information of some cells. However, the mathematical precision of the Otsu's thresholding concept has led us to believe that it is the best option to binarize the image, and that some simple preprocessing techniques can cover the internal holes. Thus, through experimental trial and error, we have established the current algorithm to consist of the following operations:

Figure 7. Area of interest detection algorithm [3].

Gaussian blurring is an operation that reduces image noise. The effect of the operation is visible in Figure 10 (b). Otsu's thresholding leaves out internal information of some cells, which indicates that the centers of those cells are closer in intensity to the background than the foreground. So, removing the intensity variance within the cell should lead to better foreground detection.

Gaussian blurring is a cross-correlation operation completed with the kernel constructed using the 2D Gaussian distribution formula:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Equation 7. 2-dimensional Gaussian distribution [8].
G(x,y) - output weight for the corresponding location,
x, y - within kernel location coordinates,
σ - standard deviation of all the location coordinates.

One constraint of the Gaussian distribution is that the operation output must be normalized, since the total contribution of the Gaussian distribution kernel does not always total to 1. The following is the formula used for normalization:

$$I_{1i} = \frac{I_{0i}}{\sum_{x=0}^{n} \sum_{y=0}^{n} w_{x,y}}$$

Equation 8. Output intensity adjustment [3].
I - intensity value of a pixel, i - pixel ID, n - kernel dimension size,
x, y - x, y coordinates within the kernel,
w - contribution weight determined by the Gaussian equation.

The following is an example of the adjusted Gaussian blurring structuring element:

$$\frac{1}{0.974} * \begin{bmatrix} 0.038 & 0.116 & 0.038 \\ 0.116 & 0.358 & 0.116 \\ 0.038 & 0.116 & 0.038 \end{bmatrix} = \begin{bmatrix} 22.618 & 27.256 & 32.536 \\ 32.756 & 47.998 & 49.558 \\ 62.24 & 82.77 & 81.889 \end{bmatrix}$$

Figure 8. Example of the adjusted Gaussian Blurring 3x3 kernel.

However, as seen in Figure 10 (c), the application of Otsu's thresholding on the blurred image results in the borders of the cells blending in with the background, which causes them to be detected as background. Fortunately, this issue can be solved by image dilation, which will

overestimate the areas of the cells. Some cells may also contain much smaller leftover holes and image dilation of a tuned kernel size will fill in those as well.

Image dilation is another cross-correlation operation that consists of the following mathematical operation:

$$A \oplus B = \{a + b \mid \exists a : a \in A \; and \; \exists b : b \in B\}$$

Equation 9. Equation of Dilation [9].
A - input image, B - structuring element,
a - values in set A, b - values in set B

The following is an example of image dilation:

a)

1	1	1
1	1	1
1	1	1

b)

0	0	0	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	0	0	0

c)

0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0

Figure 9. Example of Dilation [3].
a) Structuring element. b) Input value. c) Dilated value.

The dilated image is now an overestimation of the areas of interest. The interest pixel groups include cells in their entirety plus unwanted background pixels. However, due to the extensive visual analysis of the microscopy images, even though the centers of the cells may be detected as background they are always brighter than the actual background. Thus, there exists a value that may be used as an exposure intensity to bring out the cells' centers forward while ignoring the unwanted background. The exposure is completed on the original image at the areas that were previously detected after the final dilation operation. Then, a repeated Otsu's thresholding is applied to detect all cells in their entirety as foreground, while keeping the background untouched. The result can be seen in Figure 10 (f).

The summary of the algorithm's execution is shown below:

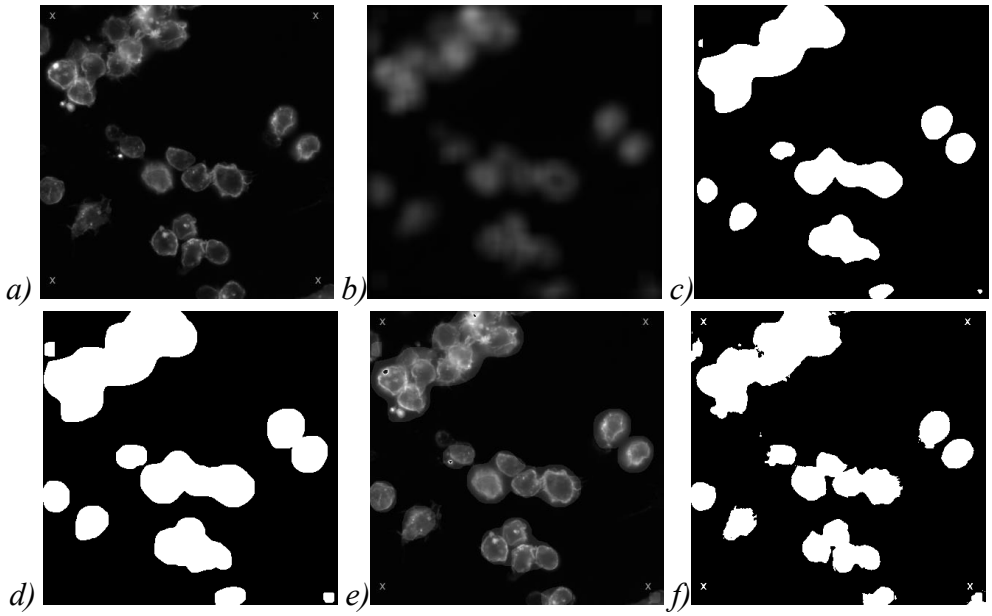


Figure 10. Area of interest detection algorithm [3].
a) Original image. b) Gaussian blur 23×23 kernel. c) Otsu's thresholding.
d) Image dilation 9×9 kernel. e) Image after exposure mask with value 30.
f) Repeated Otsu's thresholding.

Connected Components

From last semester's results as in Figure 11, under-segmentation is apparent in heavy clusters of cells. This is due to the erroneous detection of minima in the image as well as the lack of overflow protection in the flooding algorithm. We decided to utilize a connected components approach to address this issue. The algorithm is used in the overflow protection to identify the regions associated with each minimum and plant watershed lines accordingly. It is also used to count the number of cells, which corresponds to the number of clusters needed for the K-means regional minima detection algorithm discussed later in this section. For simplicity, components are extracted from the area of interest detected image, a binary image, that contains significant cell information.

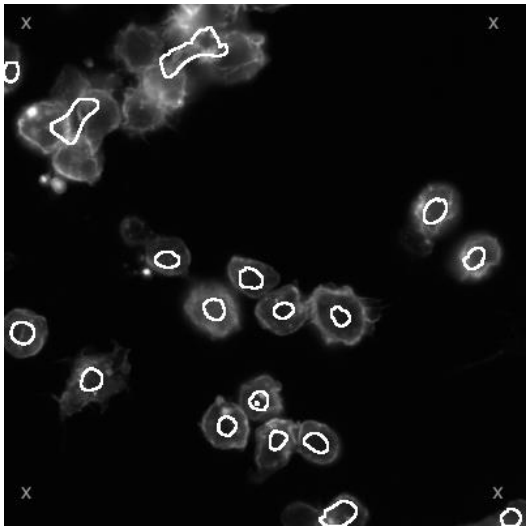


Figure 11. Final segmented image from Fall 2021 [3].

Consider an image I. A subset S of I is considered connected if for any points X and Y with the same gray level in I, there exists a sequence of points P in I such that:

$$P = \{\{P_1, P_2, ..., P_N\} \mid X = P = Y\}$$

Equation 10. Condition for S to be connected [10].
X, Y - points in subset S that have the same gray level,
P - sequence of points, N - length of the path from X to Y.

Every P_i in P is a neighbor of P_{i-1}, where $1 \leq i \leq N$. The neighborhood of a pixel is determined by its connectivity, κ, which is the smallest number of elements in S for which it remains disjoint from other subsets (or components) in I [10]. For the purpose of this project, we use 8-connectivity (as in Figure 12) because the neighborhood of an 8-connected pixel bears the most resemblance to the shape of a cell.

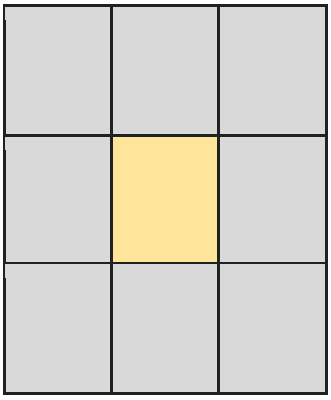


Figure 12. Visualization of an 8-connected neighborhood of a pixel (yellow)

We use a breadth-first search approach to extract all the components in the image as described in Figure 13 below.

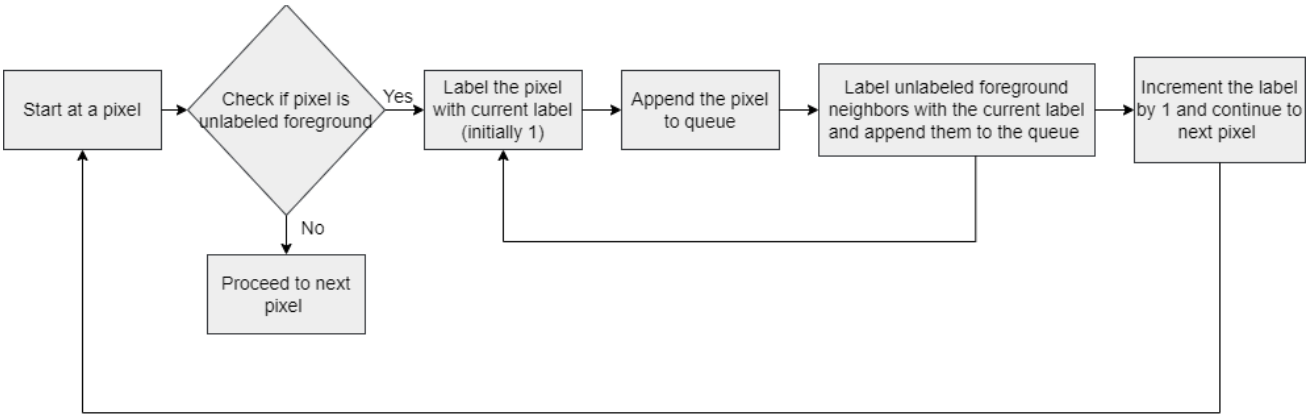


Figure 13. Outline of the algorithm to extract all the connected components of a binary image.

The connected components of a microscopy image extracted using this algorithm are visualized below:

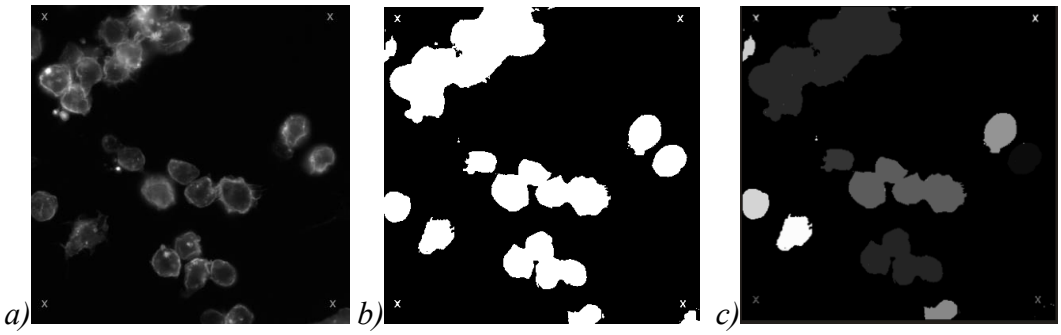


Figure 14. a) The original image. b) Detected areas of interest in the image. c) Connected components of the image visualized in varying gray levels.

Topographical maps

In order to perform watershed segmentation, we need to transform the binary image to a topographical map. For the transformation, there are several distance transform algorithms - Euclidean distance transform, city-block distance transform, etc. We chose to implement Euclidean distance transform for our project since it calculates the true distance between pixels, and the distance is an important feature in other aspects of the project. The Euclidean distance transform program was developed by the team in Fall 2021. Because it is an important step for our watershed segmentation algorithm, an overview is presented below.

According to R. Fabbri et al. [11], the Euclidean distance transform computes the distance map where each feature pixel's value is set to the distance from itself to the nearest non-

feature pixel. This creates a topographical map where the feature areas are represented as mountains, with the most central feature pixels having the highest value, resulting in peaks. After the distance map is created, it is then inverted in order to perform flooding for the watershed segmentation, where the peaks are turned into basins.

For the Euclidean distance transform, the input image is the result of the area of interest algorithm, which is a binary image with values of 0 and 255, representing non-feature and feature pixels respectively. The distance between two pixels is calculated using the distance formula as shown in Figure 15 and Equation 11 [11]. The distance formula is calculated using the Pythagorean theorem, where the distance squared between two pixels is equal to the sum of the difference in the x-coordinates of the pixels squared and the difference in the y-coordinates of the pixels squared.

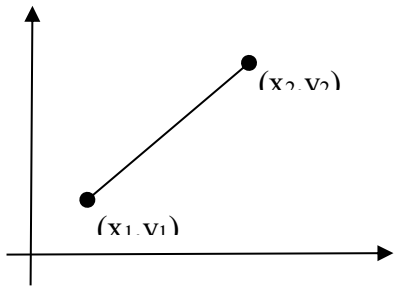


Figure 15. Graph indicating distance between two pixels [3].

$$d(x, y) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Equation 11. Distance formula [11].

The result for the program is a distance map D(p) whose value in each pixel p is the smallest distance from pixel p to the nearest non-feature pixel as given by Equation 12.

$$D(p) = Min\{d(x, y)|I(q) = 0\}$$

Equation 12. Distance map [11].
I - input binary image, p - feature pixel,
q - non-feature pixel.

A numerical representation of the Euclidean distance transform is shown below in Figure 16. Since the pixel values of an image must be an integer, the calculated distance values are then rounded to visualize the distance map as a grayscale image.

1	1	1	1	1		2.8	2.2	2	2.2	2.8		3	2	2	2	3

	1	1	1	1	1		2.2	1.4	1	1.4	2.2		2	1	1	1	2
	1	1	0	1	1		2	1	0	1	2		2	1	0	1	2
	1	1	1	1	1		2.2	1.4	1	1.4	2.2		2	1	1	1	2
a)	1	1	1	1	1	*b)*	2.8	2.2	2	2.2	2.8	*c)*	3	2	2	2	3

Figure 16. Example of Euclidean distance transform [3].
a) Original binary image. b) Euclidean distance map.
c) Rounded values for output image.

The resulting matrix is a topographical map, with the centers of the cells being represented as the mountain peaks in the distance map. To perform the flooding in the watershed segmentation algorithm, the topographical map is then inverted. An inverted euclidean distance map of a microscopy image is shown below in Figure 17 (b).

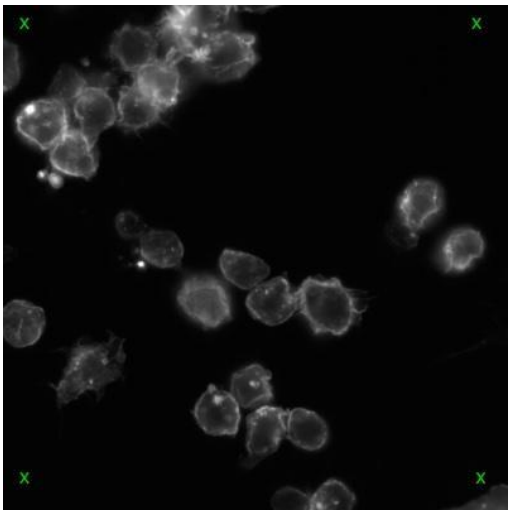
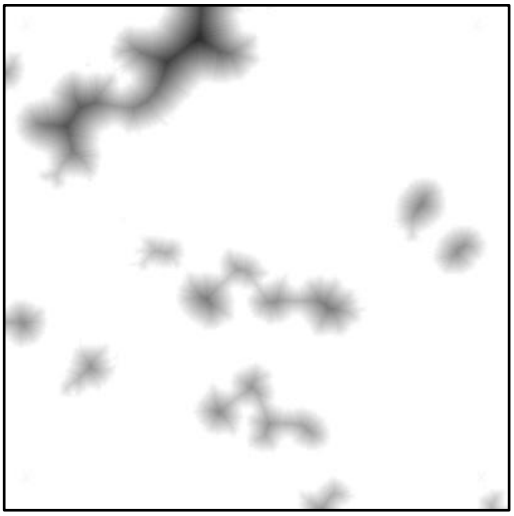
a)  *b)* 

Figure 17. Result of performing Euclidean distance transform on image [3].
a) Original image. b) Inverted Euclidean distance map.

After the inverted distance map is created, we can then proceed to regional minima detection to choose the basins for flooding.

Regional minima detection

Cell counting

The cell counting algorithm was designed to supplement a parameter for the K-means regional minima detection algorithm. Since the algorithm accepts a manually inputted parameter 'k' for the number of k-clusters to obtain, the cell counting algorithm was developed to fully automate this process.

The algorithm uses the connected components labeling to retrieve the component areas from the input image, and the number of pixels in each component is stored for use in later calculations. The mean and standard deviation of the pixel counts are used to obtain upper and lower bounds (See Equations 13-15) for the data, and these bounds are used to eliminate statistical outliers from this data.

$$\mu = \frac{\sum_{k=1}^{n} C_k}{n}$$

Equation 13. Mean pixels per component C_k in a set of n components C.

$$B_l = \mu - 2\sigma$$

Equation 14. Lower bound for eliminating outliers in data: mean minus two standard deviations.

$$B_u = \mu + 2\sigma$$

Equation 15. Upper bound for eliminating outliers in data: mean plus two standard deviations.

These statistical outliers are often groups of heavily clustered cells that have thousands more pixels than the average, or small groups that consist of pixel amounts in two digits. The histogram in Figure 18 below demonstrates the distribution of the number of pixels in each component. The bars at IDs three and five, with pixel counts of ~80,000 and ~20,000 respectively, are examples of components of heavily clustered cells that would be interpreted as outliers and thus eliminated from the data.

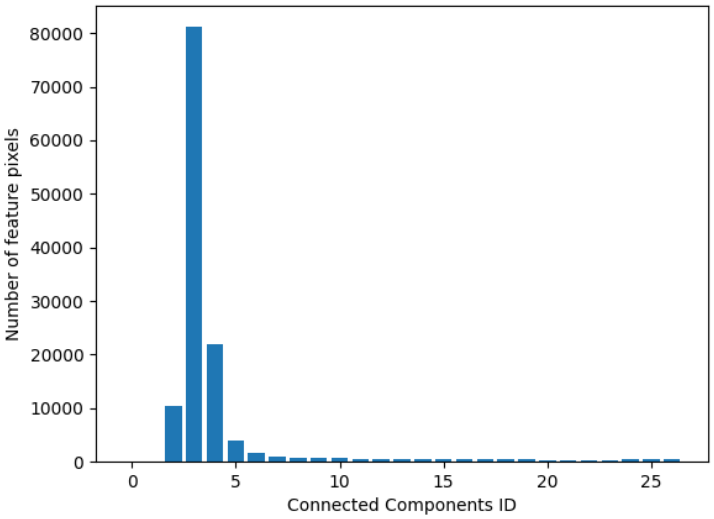


Figure 18. Histogram of feature pixel count in connected components matrix from an example image, sorted by ID.

Once the outliers are removed, the remaining data should theoretically consist of components that only make up one cell rather than multiple. So, a new mean is calculated to obtain a rough estimate for the average number of pixels present in a cell. The previously calculated sum of the list's values, which represents the total number of feature pixels in the image, is then divided by this new mean representing the average pixels per cell to estimate the number of cells in the image. This final value is returned, and is meant to serve as an input for the 'k' parameter in the K-means regional minima detection algorithm. Figure 19 details the resulting estimates of the cell counting algorithm applied to two microscopy images.

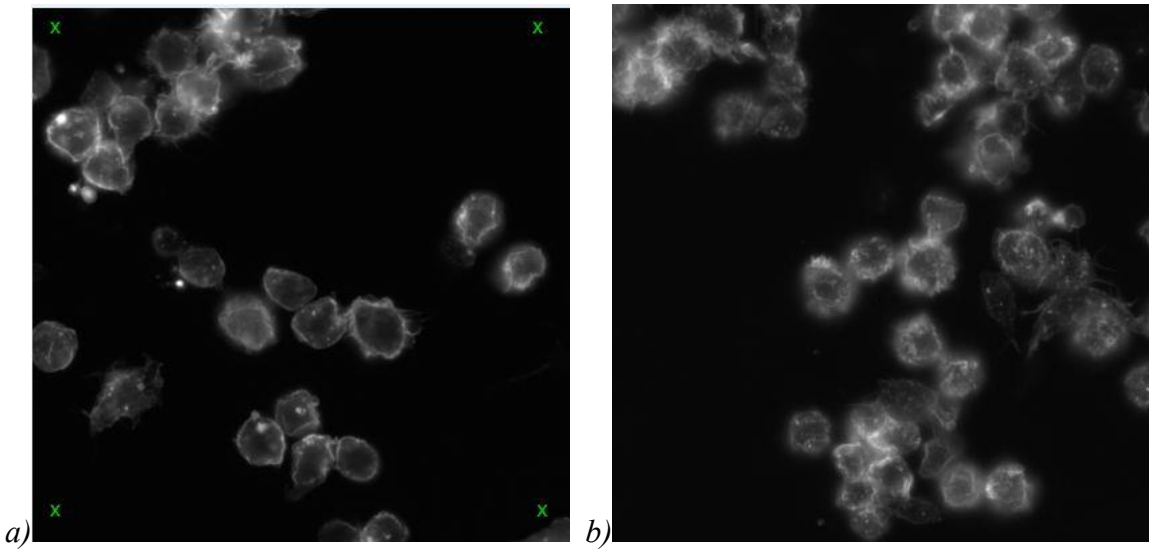


Figure 19. Cell Count Estimates from Algorithm
a) Estimate: 27 cells. Actual: 25 cells. b) Estimate: 25 cells. Actual: 28 cells.

K-means algorithm

The K-means algorithm is a clustering algorithm which divides a set of data points into k clusters [12]. The algorithm requires a parameter value k, which specifies the number of centroids. The product of this algorithm is a list of the positions of the optimized centroids when the algorithm terminates.

The general idea of the K-means algorithm is to minimize the sum of the distances from each data point to its corresponding cluster. The process of the K-means algorithm is shown in Figure 20 below. The algorithm generates k centroids at the beginning and works on updating the position of centroids by minimizing the sum of squared error of each cluster in a loop.

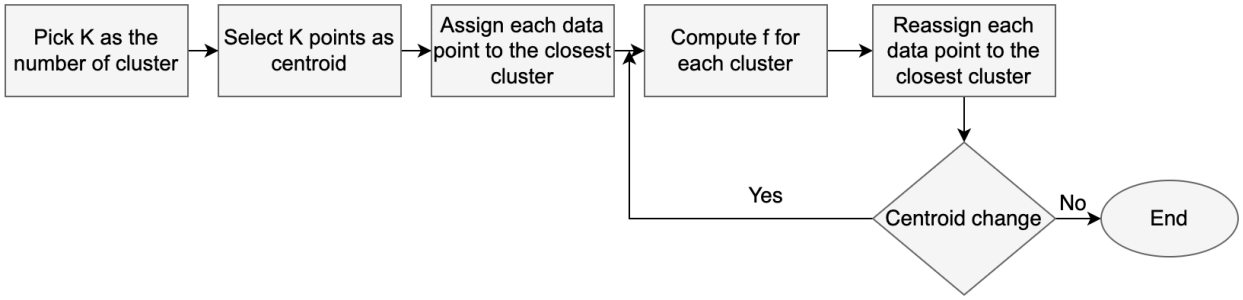


Figure 20. Block diagram of K-means algorithm [12].

The function f in Figure 20 is defined in Equation 16, where s_i represents the set of data points in the i_{th} cluster, x_j represents the data point in s_i, and μ_i, which is shown in Equation 17, represents the mean position of data points in the i_{th} cluster.

$$f = \sum_{i=1}^{k} \sum_{x_j \in s_i} (x_j - \mu_i)^2$$

Equation 16. Sum of squared errors [12].

$$\mu_i = \frac{1}{s_i} \sum_{x_j \in s_i} x_j$$

Equation 17. Mean of data points in the i_{th} cluster. [12].

The purpose of the K-means algorithm is to identify the positions of true regional minima. In our case, the input data points are the original regional minima that get detected from the topographical map, the value k is the counted cell numbers from the cell counting algorithm, and the output centroids are the final detected regional minima. An example of the original regional minima is shown in Figure 21 (a) and an example of detected regional minima using the K-means algorithm is shown in Figure 21 (b).

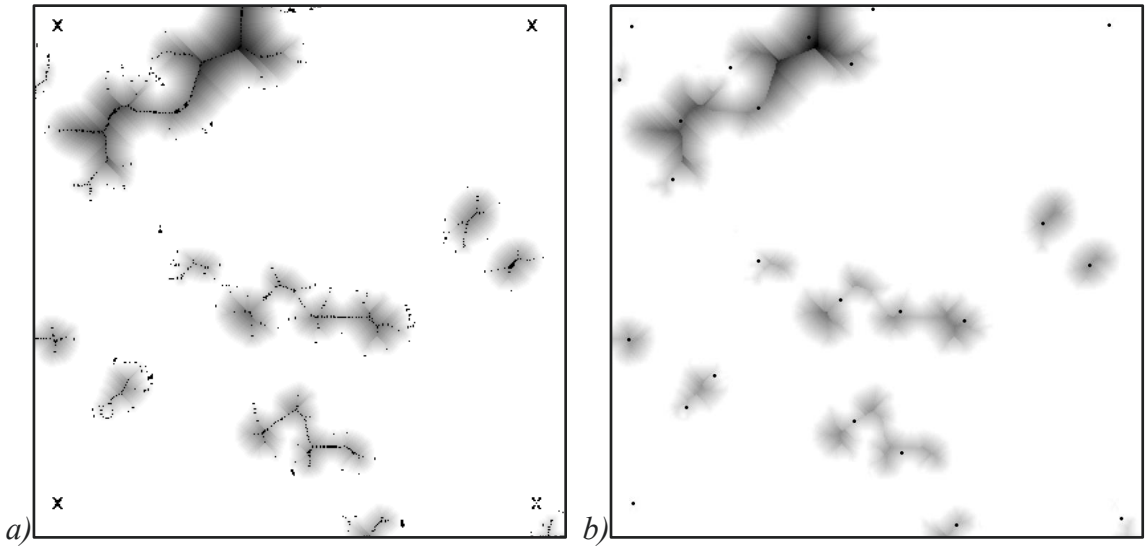


Figure 21. Detected regional minima before and after using K-means clustering a) Original regional minima. b) Regional minima detected using K-means.

Although the redundant regional minima are removed after applying the K-means algorithm, some of the regional minima are placed incorrectly. These errors come from regional minima that are detected outside of the cell areas. To solve this, an H-minima transform was implemented to remove the undesired regional minima.

H-minima transform

As mentioned previously, the K-means algorithm places some regional minima outside of the cell region. In order to remove these erroneous minima, an H-minima transform is implemented prior to the K-means clustering.

According to N. Fauzi Ismail et al. [13], an H-minima transform filters the regional minima in a regional minima map whose depths are less than or equal to the h-value parameter, which is a non-negative scalar. [13]. The implementation of an H-minima transform is shown in Figure 22.

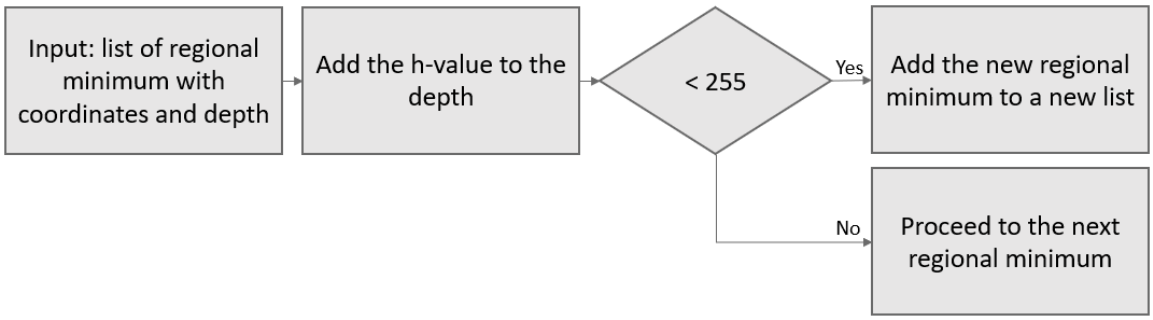


Figure 22. Block diagram of the H-minima transform.

This algorithm's operation can be interpreted as raising the depth of all the regional minima by the set h-value. Those that remain as minima after the transform are returned to their original depths, and those that are transformed at or above the depth of zero, which is the plateau, are removed. The product of an H-minima transform applied to detected regional minima before the K-means algorithm is shown below in Figure 23 (b), with the untouched regional minima shown in Figure 23 (a) for comparison. A comparison of the differing results from applying the K-means algorithm with and without an H-minima transform follows in Figure 24.

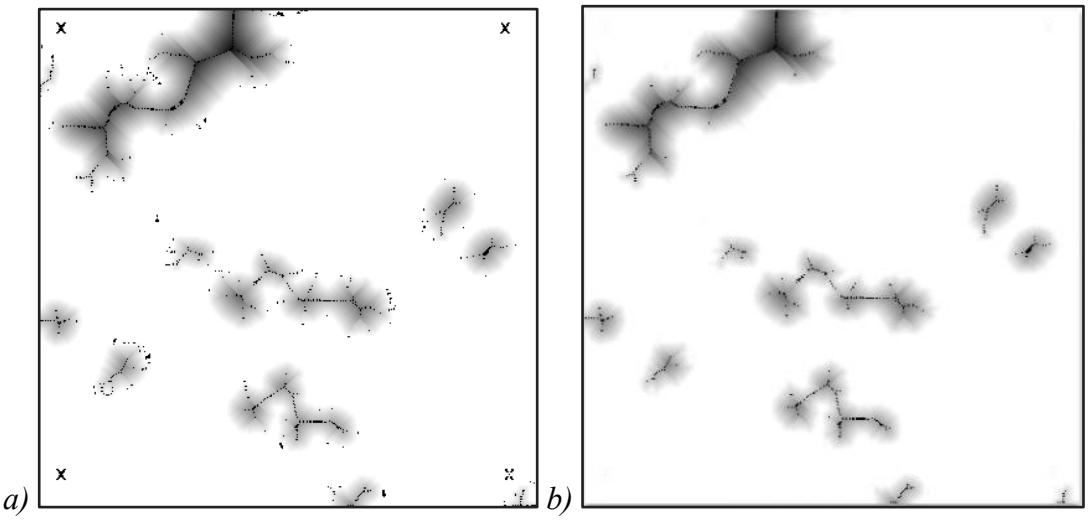


Figure 23. Regional minima detection before and after applying an H-minima transform. a) Original regional minima. b) Result of applying an H-minima transform.

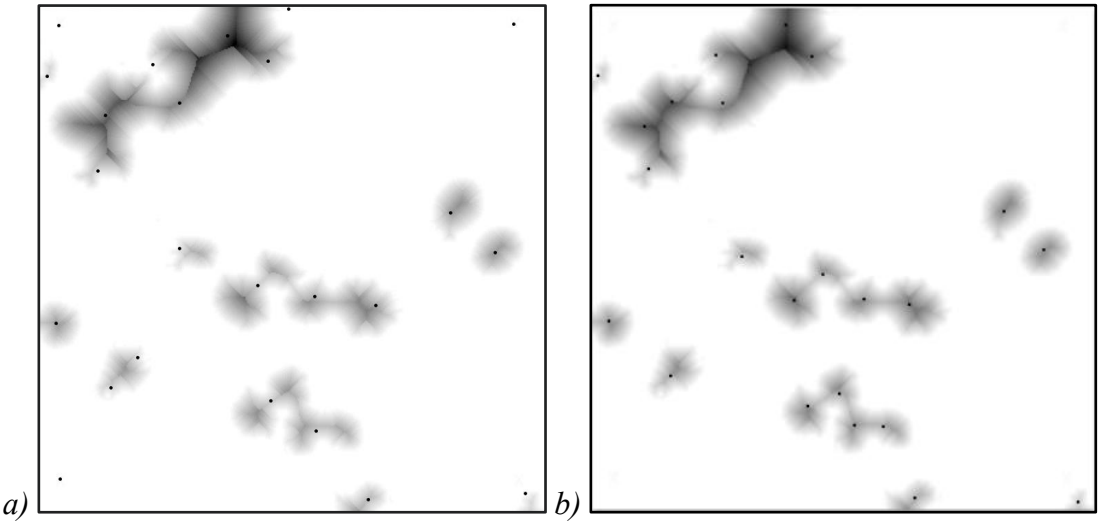


Figure 24. Comparison of results of the K-means algorithm with and without H-minima transform. a) Regional minima detected with K-means. b) Regional minima detected with an H-minima transform applied before K-means.

We are satisfied with the results of the optimized algorithm combining an H-minima transform with the K-means regional minima detection algorithm. However, there is still potential for optimization, since the current version of K-means regional minima detection still requires the desired h-value to specify how deep the H-minima transform should be. As discussed previously, the cell counting algorithm was implemented to no longer require a user-inputted value of k for the K-means algorithm. A similar algorithm could be developed in the future that automates the process of determining which h-value should be applied to the H-minima transform.

Cell border detection

Flooding

The flooding algorithm was developed by the team in the Fall 2021 semester, however, it was far from ideal. The algorithm relied heavily on ceiling functions, which were used in an attempt to avoid basin overflows.

A ceiling function is a mask that is placed on top of the topographical map. The mask is another topographical map that indicates the relief to which the water is allowed to rise. However, the desired result of the watershed segmentation algorithm is for all basins to be flooded in full, while placing watersheds between the basins to avoid overflow. Thus, certain adjustments were made to the original algorithm. The current flooding procedure will be discussed further.

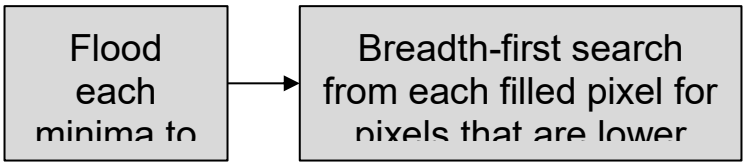


Figure 25. Flooding procedure.

As seen in Figure 25, the flooding procedure is fairly intuitive. Each one of the detected minima on the topographical map gets procedurally "filled with water", which mathematically translates to assuming the value of the plateau. Then, each filled pixel completes a First-In First-Out stack breadth-first search to determine other non-filled pixels to raise them to the plateau level as well. This method utilizes the previously mentioned connected components algorithm. It is important to complete the operation in a First-In First-Out stack architecture, since each basin must be filled with water gradually, pouring equivalent amounts of liquid into each basin at the same time.

The following is a visualization of the flooding procedure:

a)

255	255	255	255	255	255	255	255
255	100	70	255	255	70	100	255
255	150	50	255	255	50	150	255
255	100	70	255	255	70	100	255
255	255	255	255	255	255	255	255

b)

255	255	255	255	255	255	255	255
255	100	70	255	255	70	100	255
255	150	255	255	255	255	150	255
255	100	70	255	255	70	100	255
255	255	255	255	255	255	255	255

c)

255	255	255	255	255	255	255	255
255	100	255	255	255	255	100	255
255	150	255	255	255	255	150	255
255	100	255	255	255	255	100	255
255	255	255	255	255	255	255	255

d)

255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

Figure 26. Flooding visualization.
a) Topographical map with regional minima. b-c) 1st-2nd flooding iterations.
d) Final flooded image.

Overflow protection

After testing the flooding procedure, we have confirmed the remaining issue of basin overflow. Looking at Figure 11 in the connected components section, it is clearly visible how multiple basins risk merging into one another due to flooding. Considering that we removed the need for the ceiling function and that the basins are now filled to the brim, those that are connected by a relief other than the plateau merge into one large basin. The detailed visualization of the issue is as follows:

a)

255	255	255	255	255	255	255	255
255	100	70	255	255	70	100	255
255	150	50	200	200	50	150	255
255	100	70	255	255	70	100	255
255	255	255	255	255	255	255	255

b)

255	255	255	255	255	255	255	255
255	100	255	255	255	255	100	255
255	150	255	200	200	255	150	255
255	100	255	255	255	255	100	255
255	255	255	255	255	255	255	255

c)

255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

d)

255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

Figure 27. Overflow visualization.
a) Topographical map with regional minima. b-c) Flooding individual basins.
d) Final flooded image where two basins merge into one.

The overflow protection algorithm is a procedure that allows each basin to be filled with a liquid that does not mix with any other. To do this we simply utilize a modified version of the connected components algorithm. Currently, the algorithm utilizes IDs that are not adjusted for the value of the plateau height.

To adjust the IDs we offset the numbers by the plateau level. This makes sure that when any water reaches an ID (another body of water) it does not mistake it for a portion of the relief. A distinguishable ID is assigned to each of the minima in an ascending order starting with 256 (height of plateau plus 1).

When flooding, each pixel below the plateau level gets filled with the ID of the corresponding body of water. If the pixel is already filled with a liquid, no other liquid can overwrite it.

To better demonstrate the procedure, the following visual is available:

a)

255	255	255	255	255	255	255	255
255	100	70	255	255	70	100	255
255	150	50	200	200	50	150	255
255	100	70	255	255	70	100	255
255	255	255	255	255	255	255	255

b)

255	255	255	255	255	255	255	255
255	100	70	255	255	70	100	255
255	150	256	200	200	257	150	255
255	100	70	255	255	70	100	255
255	255	255	255	255	255	255	255

c)

255	255	255	255	255	255	255	255
255	256	256	255	255	257	257	255
255	256	256	200	200	257	257	255
255	256	256	255	255	257	257	255
255	255	255	255	255	255	255	255

d)

255	255	255	255	255	255	255	255
255	256	256	255	255	257	257	255
255	256	256	256	257	257	257	255
255	256	256	255	255	257	257	255
255	255	255	255	255	255	255	255

Figure 28. Overflow protection visualization.
a) Topographical map with regional minima.
b-c) Flooding individual basins with respective liquids.
d) Final flooded image where distinct liquids do not mix.

The outcome of the described algorithm resulted in a success. In the following images, it can be seen how each individual minimum only floods its corresponding basin with a liquid of a distinct ID value:

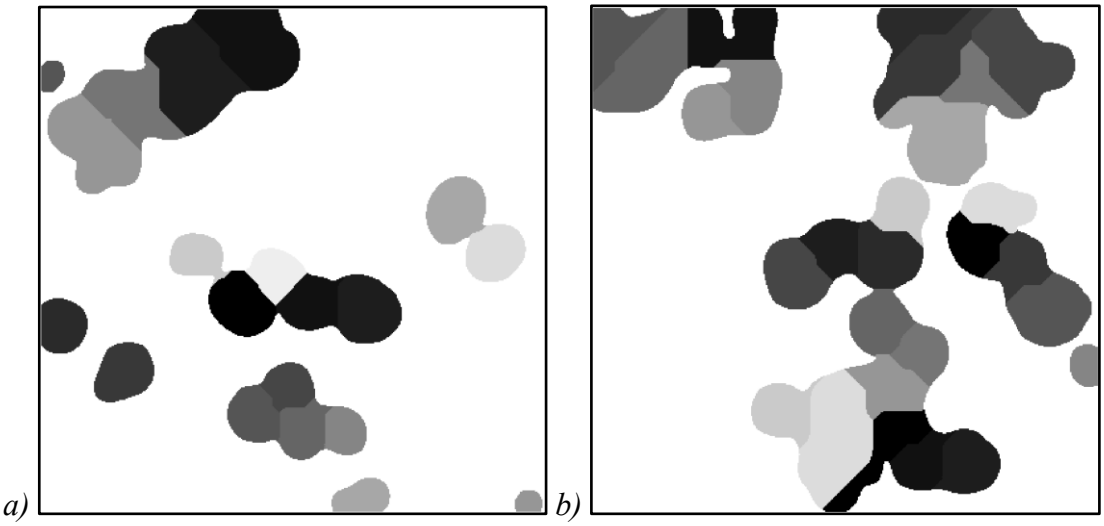


Figure 29. Result of flooding with overflow protection.
a-b) Flooded topographical maps.

Watershed placement

Watershed placement is an algorithm that operates on the flooded image to outline the borders of each cell. This algorithm is fairly intuitive. The watersheds are placed on the borders where basins touch other basins or the plateau.

Watershed placement allows for three modes: bottleneck mode, cell borders mode, and plateau mode.

Bottleneck mode is applied when two cells are connected through a bottleneck of the width of a singular pixel. In such a scenario, the watershed overwrites that bottleneck pixel, as shown in Figure 30 (a).

Cell borders mode, as the name suggests, is applied when cells are touching each other along a path of some length. In this case, both cells become the victims of information loss. The watersheds are placed on both sides along the border to avoid selecting a preferred cell that remains intact as the result of the watershed placement. This is illustrated in Figure 30 (b).

Plateau mode is applied along the border of a cell that touches the plateau. The watershed is placed along the internal border of the cell, which also results in slight information loss as seen in the figure 30 (c).

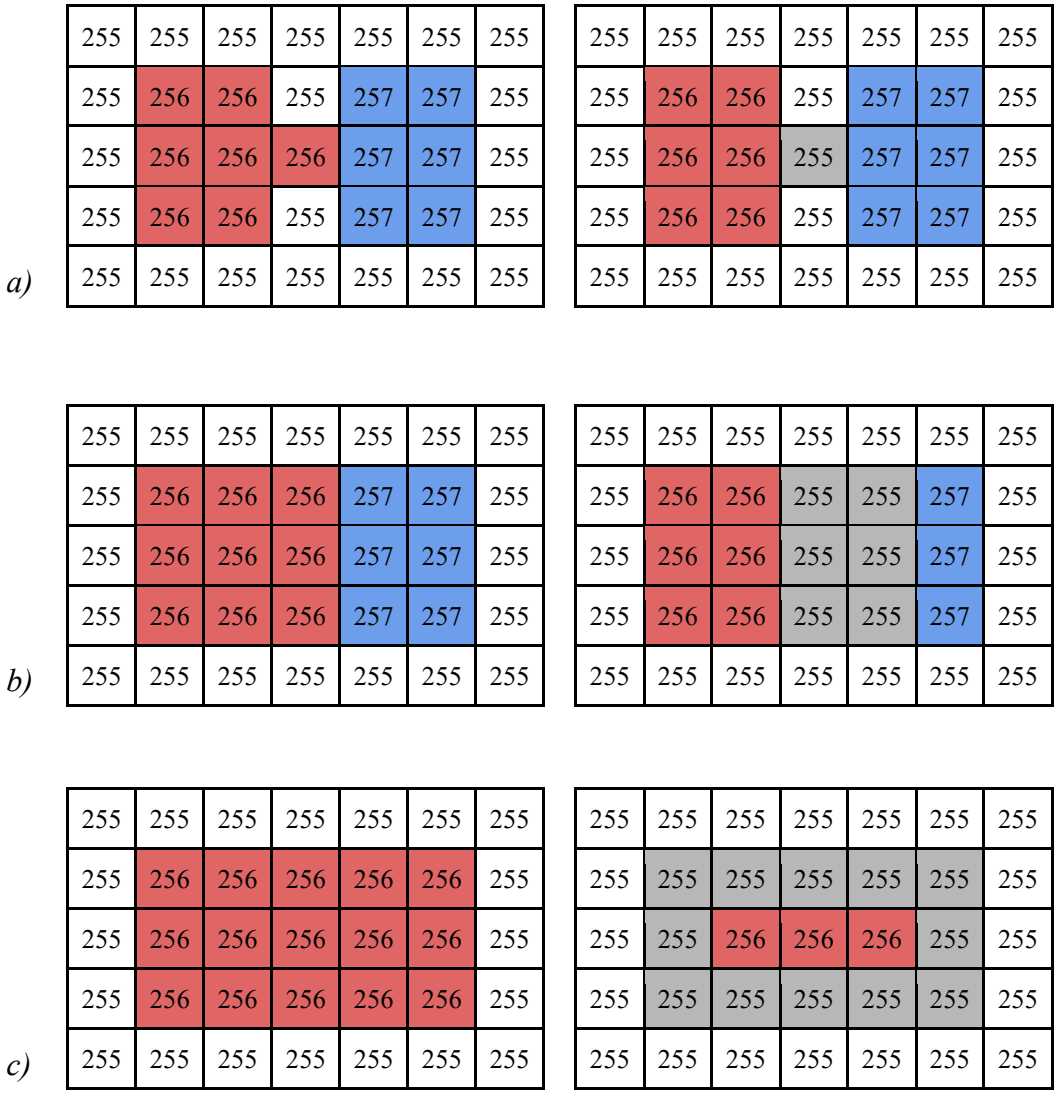


Figure 30. Watershed placement visualization.
a) Bottleneck mode. b) Cell border mode. c) Plateau mode.

By utilizing the techniques above we have managed to produce very promising results that can be seen in Figure 31:

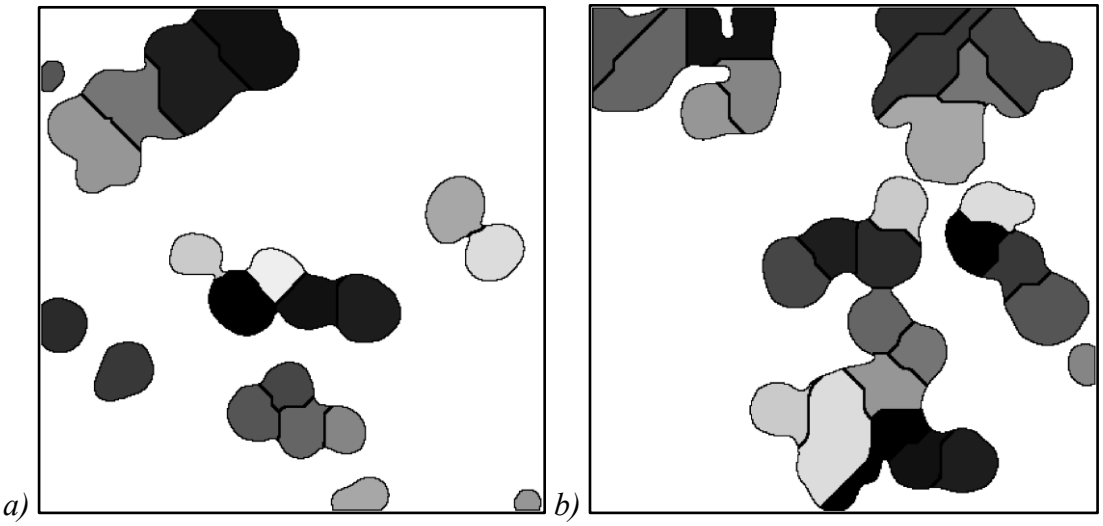


Figure 31. Result of watershed placement.
a-b) Segmented topographical maps.

Extracting the watersheds from the images in Figure 31 we can construct a mask that we can place on the original images. The result of such a step is displayed in Figure 32:

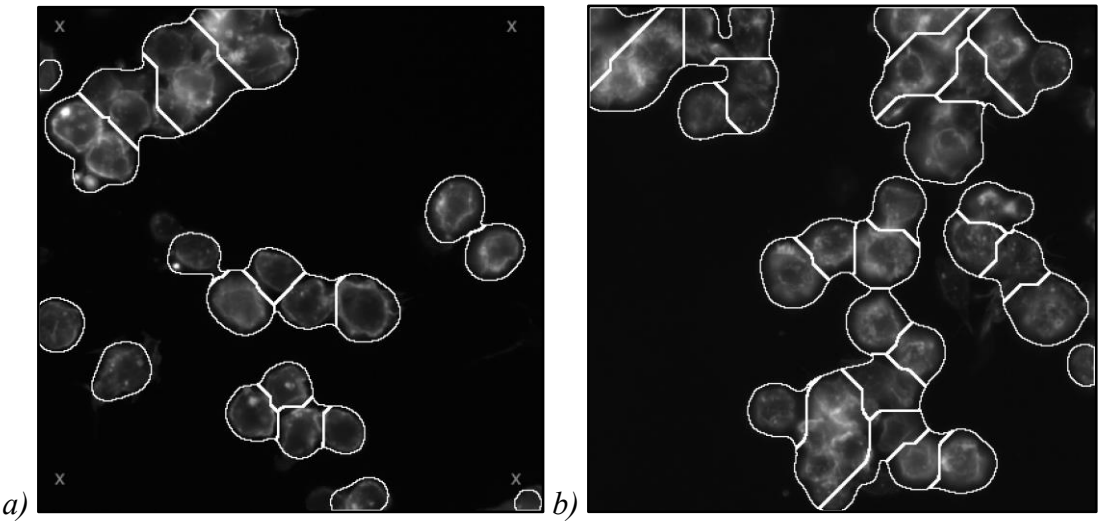


Figure 32. Result of placing the watershed mask on the original images.
a-b) Final result of watershed segmentation.

V. Cell classification

CNN design

Cell classification is a process that requires some form of machine learning. In this paper, we decided to focus on establishing proof of concept first to determine if we can scale up the model in future work.

To define the basic model, we decided to approach the task from the perspective of a binary classifier. We have a large collection of cells from the open source Broad Bioimage Benchmark Collection that we can freely use. The initial motivation was to detect and predict early stages of cancer, however, we could not find a cancerous cell dataset. As a result, we decided to utilize the malaria dataset, since the main idea behind the classification is the same: extract features from healthy and infected cells and train the model to recognize them.

The dataset has six distinct cell types, two of which are healthy, and four of which are infected [14]. After a comprehensive visual analysis of the dataset, we discovered that most infected cells share similar features and that most healthy cells share similar features as well. Thus, we have decided that a binary classifier would suffice for the initial implementation of the model.

The proposed architecture consists of a singular neuron and can be seen in Figure 33:

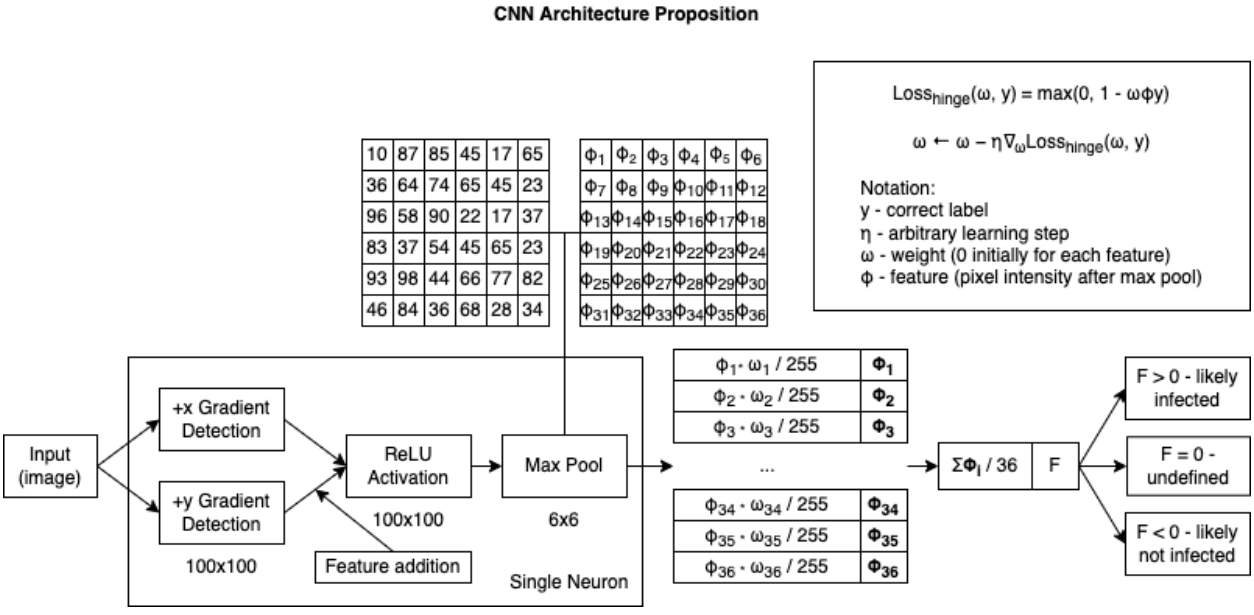


Figure 33. Proposed single-neuron Convolutional Neural Network.

Our binary classifier consists of a single neuron that in turn consists of convolutional, activation, and pooling layers that connect to the output layer. The details of each of the layers are presented further in the paper.

Convolutional layer

The purpose of the convolutional layer is to extract features of the input image. The layer returns feature maps, which are the outputs of convolving filters with input data. The convolutional layer is where most of the computation in the CNN takes place [15].

There are three hyperparameters in a convolutional neural network: the number of filters, the stride, and zero-padding. Hyperparameters determine the volume of the output of the layer and the learning process of the model. The number of filters determines the depth, the stride controls filter movement by specifying the number of pixels that the kernel moves each time, and zero-padding pertains to the information around the border of the input image.

Various types of feature extraction, such as edge detection, corner detection, and others, are applied in convolutional layers. The extracted features are then passed into their next layers for classification. In general, simple features are extracted at the beginning. As the layers become deeper, the extracted features become more complex.

Currently, there is only one convolutional layer in the model. Gradient detection, which will be discussed in its corresponding section, is utilized in the convolutional layer to extract features for classifying infected and healthy cells.

Activation function

An activation function is a function that lights up the neuron if the value is above a certain threshold. An activation function also introduces non-linearity to the network, which is important because the operations of every neuron are linear operations. Without activation functions, the operations of the entire network can be re-factored as a single linear function, which can be done by a single neuron.

The team chose to use the rectified linear unit (ReLU) function as our activation function because it is one of the most used activation functions for training CNNs [16]. ReLU is a piecewise linear function where all negative inputs are outputted as zeros. Positive inputs are returned unchanged [16]. The graph and equation of the ReLU function are shown below:

$$f(x) = max(0, x)$$

Equation 18. ReLU function [17].

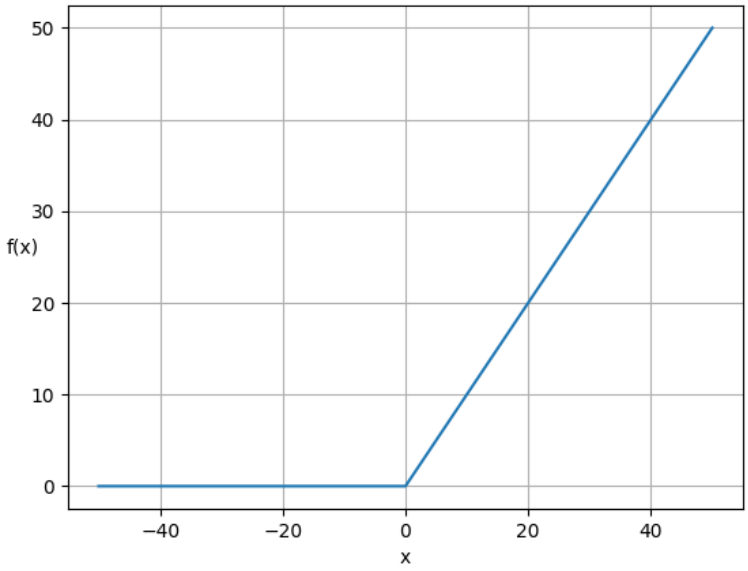


Figure 34. Graph of ReLU function.

Currently, there is only one convolutional layer in our proposed CNN model, so the activation function is not yet necessary. However, after the team introduces more convolutional layers to the model, the ReLU function will need to be applied after the convolutional layers.

Pooling layer

The main purpose of a pooling layer is to shorten the processing time and reduce the number of parameters in a neural network [15]. The pooling layer keeps the important information and helps control the problem of overfitting by filtering out some information during the pooling operation [18]. In image processing, the output image of the pooling layer is an image of lower resolution than the input image. There are various types of pooling layers, but the most common ones are max-pooling and average pooling.

Pooling is done by partitioning the input image into sub-regions of rectangles. In the case of max pooling, the maximum value of each sub-region is extracted and placed as the pixel value of the output. Figure 35 is an example of max-pooling where the kernel size is 2×2 and the stride, which is the number of pixels the kernel moves each time, is two.

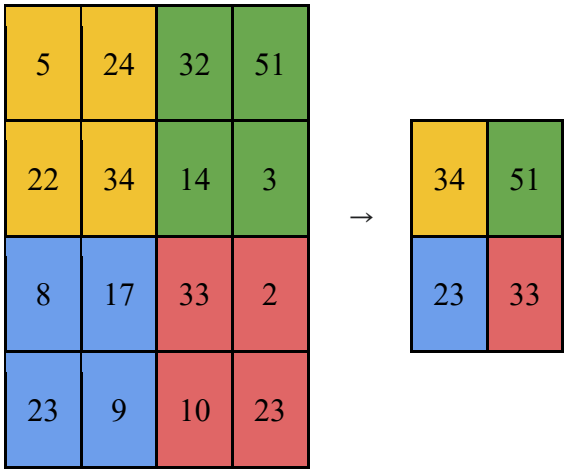


Figure 35. Example of 2x2 max-pooling.

In our project, a pooling layer is applied after the activation layer to reduce the number of parameters as well as the computational complexity. By running through the pooling layer, the number of parameters required to calculate weights is reduced.

Fully-connected output layer

A fully-connected layer is a layer that takes in all the parameters as its input. If there are multiple neurons, each entry of the fully-connected layer is connected to each one of the neurons. In the case of the proposed model, the single neuron fully connects to the singular component of the fully-connected layer and produces the result.

Rather than fully explaining the output vector, it is simpler to provide examples. The output vector is the result of the pooling layer, and consists of 36 distinct features and their corresponding weights. Each feature of the output vector is a contribution level, and each weight corresponds to a class. If the vector input has a high intensity and a positive weight, then it means that the cell is more likely to be infected. If the intensity is low and the weight is positive, then the cell is less likely to be infected. If the intensity is high and the weight is negative, then the cell is more likely to be healthy. If the intensity is low and the weight is negative, then the cell is less likely to be healthy.

The contribution of each weight is summed up and normalized to produce an output between -1 and 1. This value corresponds to the certainty level of the model. If the output is close to the -1 value, then the cell is more likely to be healthy. If the output is close to 0, but negative, then the cell is somewhat likely to be healthy. If the output is close to 1, then the cell is more likely to be infected. If the output is close to 0, but positive, then the cell is somewhat likely to be infected. If the output is 0, or extremely close to 0, then the cell classification is undetermined. This result means that the cell has features of both healthy and infected cells. To avoid repetition, the mathematical procedure is depicted in detail in the model architecture in Figure 33.

CNN feature extraction

Feature extraction is used to extract the important features from images in a dataset. The extracted features are able to describe and represent a portion of the data, and are thus used for classification purposes [19]. There are several types of features that can be extracted from a given data set, for instance; color features, shape features, texture features, etc. Feature extraction is usually applied in the convolutional layer, where the output of the layer is the information of the extracted features.

The team developed two types of feature extraction this semester: gradient detection and histogram-based feature extraction. Gradient detection relies on calculating the rate of change of the pixel intensities within the input images, whereas histogram-based feature extraction focuses on computing the statistical information of the input images. Both of these extraction methods are used to classify healthy and infected cells.

This project utilizes gradient detection in the +x and +y directions. The structuring elements of both directions of the gradient detection are shown in Figure 36. Examples of the results of healthy and infected cells are shown in Figure 37. Both the gradient detection of the healthy cell in +x and +y directions are smoother than the infected cell.

a)

-1	0	1
-1	0	1
-1	0	1

b)

1	1	1
0	0	0
-1	-1	-1

Figure 36. Structuring elements of gradient detection.
a) Structuring element of +x direction. b) Structuring element of +y direction.

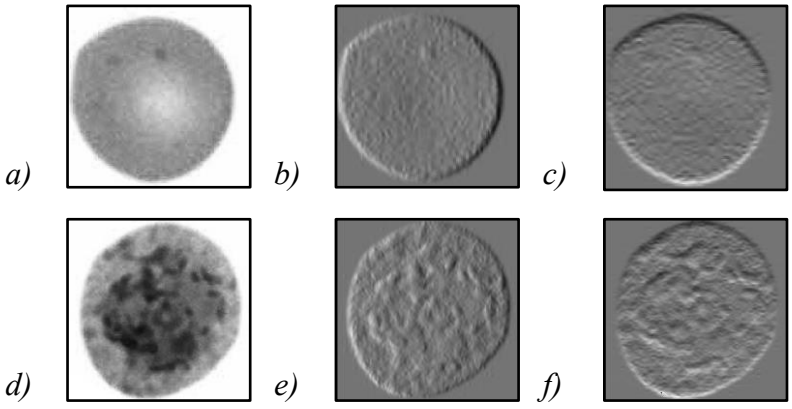


Figure 37. Gradient detection on healthy and infected cells.
a) Healthy cell. b) +x gradient detection. c) +y gradient detection.
d) Infected cell. e) +x gradient detection. f) +y gradient detection.

Histogram-based feature extraction is based on texture extraction and is a method that is computed mathematically [20]. The team has written a function that computes the mean, standard deviation, energy, and skewness of an image. All of this computation requires the use of the probability function $p(i)$, as shown in Equation 19, where n_i represents the number of pixels with intensity i and N represents the total number of pixels [3].

$$p(i) = \frac{n_i}{N}$$

Equation 19. Probability function [3].

The mean value, given by μ, is the average value of all the pixels in the image. The standard deviation, given by σ, indicates how the values of pixel intensities are spread around the mean of the input image. Skewness measures the asymmetry of the values of pixel intensity in the input image [20]. All of these values are defined below, where L represents the maximum possible value of pixel intensity:

$$\mu = \sum_{i=0}^{L-1} ip(i)$$

Equation 20. Mean value of input image [20].

$$\sigma = \sqrt{\sum_{i=0}^{L-1} (i-\mu)^2 p(i)}$$

Equation 21. Standard deviation of input image [20].

$$energy = \sum_{i=0}^{L-1} p(i)^2$$

Equation 22. Energy of input image [20].

$$skewness = \sum_{i=0}^{L-1} (i-\mu)^3 p(i)$$

Equation 23. Skewness of input image [20].

Currently, gradient detection is the only feature extraction method that is used in our model. The team has only finished implementing the histogram-based feature extraction method

and has yet to combine it with the CNN proposed model. The decision of where the histogram-based feature extraction is going to be placed in the model will be made in the future.

CNN training

The network training is completed through backpropagation discussed earlier in the technical background section of this paper. Training details, however, are discussed further in this section.

For training we used gradient descent and the hinge loss function, since these are very common choices for binary classifiers [21].

$$Loss_{hinge}(\omega, y) = max(0, 1 - \omega\phi y)$$

Equation 24. Hinge loss function [21].
ω - vector element weight, ϕ - vector element feature,
y - correct label.

Each weight is updated to correspond to the margin of the current output result from the ground truth label. Considering that feature combination is linear by gradient definition, its output with respect to a weight will either produce a 0 value or a corresponding -ϕy value.

$$\omega_i \leftarrow \omega_i - \eta \nabla_{\omega_i} Loss_{hinge}(\omega_i, y)$$

Equation 25. Weight update procedure [21].
ω - vector element weight, i - weight vector index,
η - learning step.

The training focuses on the weight updates of each of the output vector entries. With each iteration of a new cell image, the weights are updated according to the cell's ground truth. If extracted features are distinct enough in healthy and infected cells, then only a corresponding subset of weights will be updated.

Such a model is also commonly used for natural speech processing to determine whether a message holds a positive or a negative context [21].

Training results are discussed further in the paper.

VI. Results

Cell segmentation

The cell segmentation results have greatly improved since last semester. With the introduction of new algorithms such as connected components labeling, K-means regional

minima detection, and overflow protection, the complete algorithm sequence is able to segment individual cells to perfection and can segment cell clusters with sufficient accuracy. As shown by last semester's results in Figure 11, the preliminary basin placement of cells was accurate for cells isolated in the image, meaning they do not share borders with other cells. However, the basins in heavily clustered cells end up overlapping with one another, causing the watershed transform to interpret a group of cells as a single large cell.

This error was caused by the usage of a simple regional minima detection algorithm on the Euclidean distance transform, which would then be flooded by the watershed transform. As discussed earlier, the K-means regional minima detection algorithm was developed to analyze the regional minima to find the true minima of each individual cell in a cluster so that flooding could begin at those pixels only. Figure 21 shows preliminary results of the K-means algorithm. As shown, the detected regional minima are not perfectly centered in the cells. In order to resolve this, an h-minima transform is applied to the original regional minima to eliminate all regional minima that are past a depth intensity of a parameter h. From analysis, it was found that an h-value around 24-30 is optimal for the h-minima transform to eliminate enough unwanted minima so that the K-means algorithm can be fully accurate. Figure 24 shows the K-means algorithm results before (a) and after (b) an h-minima transform has been applied. As shown in b, the detected regional minima are placed perfectly in the centers of the isolated cells, and are placed with sufficient accuracy in the centers of the heavily clustered cells. Combined with effective overflow protection, the flooding algorithm will now fill up cells starting from their centers and prevent the merging of basins, shown in Figure 38 below.

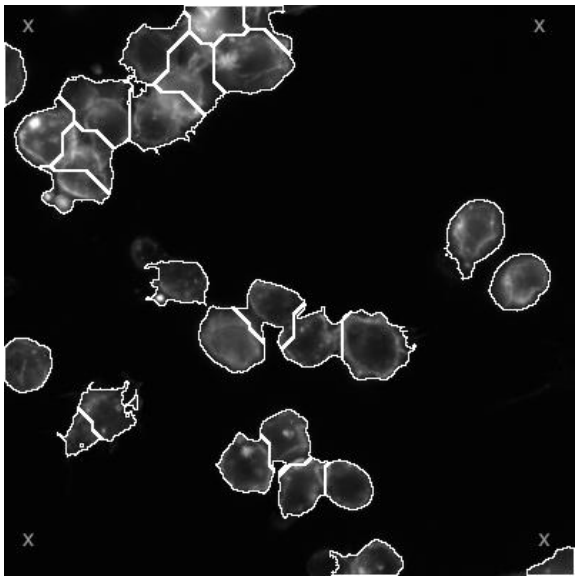


Figure 38. Final result of watershed segmentation with K-means regional minima detection and overflow protection.

As shown, the watershed algorithm is nearing completion. The isolated cells are segmented with near-perfect accuracy, but in heavily clustered groupings some watershed

borders are placed in ways that bisect cells. This is caused by incorrect placement of regional minima - if two regional minima are detected in a cell, our algorithm will place a separator that ends up cutting through the cell. However, the segmentation results overall are satisfactory, and allowed us to begin diverting our focus to other aspects. So, with the watershed segmentation producing relatively successful segmentation of heavily clustered cells along with near-perfect segmentation of isolated cells in microscopy images, the team split off into two groups: one that would begin developing the algorithms and techniques necessary for cell classification, and one that would continue working on cell segmentation. Cell segmentation optimization continued to be a goal of the team because of the imperfect segmentation of heavily clustered cells. Recent investigations into how varying h-minima transforms affect segmentation of different cell clusters show promising potential for future work, which will be discussed later.

Cell classification

With the help of a ground truth data parser, we managed to extract over 400 individual cell images which we used as a training dataset.

To visualize the training in progress here are the updated first six weights of the output vector passed through two images, one healthy and one infected:

	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6
Initial:	0.0	0.0	0.0	0.0	0.0	0.0
1st pass (infected)	0.0	0.0	5.88e-5	6.37e-4	6.31e-4	6.06e-4
2nd pass (healthy)	0.0	-1.76e-5	-8.82e-5	2.21e-4	2.88e-4	1.98e-4

Figure 39. Weight updates after two passes.

The result seen in Figure 39 was initially very promising. It was clear that vector entries 4, 5 and 6 were mostly contributing to the detection of an infected cell, where vector entries 2 and 3 were contributing to the detection of a healthy cell. However, after running a full scale training on the entire dataset, the results came out more pessimistic.

	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6
Initial:	0.0	0.0	0.0	0.0	0.0	0.0
1st pass	-7.84e-5	-1.31e-4	-4.06e-4	-4.18e-4	-3.22e-4	-3.8e-4

2nd pass	-1.49e-4	-1.69e-4	-5.88e-4	-7.14e-4	-7.39e-4	-6.33e-4
…						
460th pass	-3.45e-2	-7.7e-2	-1.15e-1	-1.17e-1	-1.05e-1	-9.25e-2
461st pass	-3.49e-2	-7.73e-2	-1.15e-1	-1.17e-1	-1.05e-1	-9.26e-2

Figure 40. Weight updates after full training is complete.

The results of training seem to identify all vector weights to be contributing to the detection of only healthy cells. Through deep analysis of the output, we now understand that there are two possible areas of the model that should be deeply revised. Firstly, feature extraction needs to be expanded and optimized. Since the output weights all assume a negative value and considering that the dataset is predominantly healthy, it means that every output vector feature is present in each healthy cell. Looking at the effect of the infected cells on the weight vector, the same can be concluded. Secondly, the level of downsampling may be too high. Reducing the downsampling depth will allow for a better distinction between original features of all the images.

VII. Conclusion

This semester, the team has worked on cell and nuclei detection, counting, and classification. Optimizing and improving the algorithms, which involve cell and nuclei detection and counting, that were written last semester and constructing a model for cell classification was the goal at the start of the semester. Fortunately, the result of watershed segmentation is much more accurate than last semester's result, and the implementation of the CNN design works properly since the weight is updated successfully.

The semester work can be separated into two parts: improving watershed segmentation and implementing a CNN model.

To improve watershed segmentation, the team implemented connected components labeling and a cell counting algorithm to count the number of cells in a microscopy image for regional minima detection. Since last semester’s regional minima detection didn’t identify the true cells in a cluster, the K-means algorithm and the h-minima transform were implemented to solve this issue. The watershed placement algorithm was then improved to solve the problems of overflow.

With the improvement and implementation of the aforementioned algorithms, watershed segmentation is now able to segment both individual cells and cells in clusters accurately. However, the h value for the h-minima transform has to be input manually and different h-minima values will create different results in both individual cells and heavily cluster cells.

In the second half of the semester, most of the team members started researching CNN and implementing the functions that are used in each layer. Since most of the students did not have experience with convolutional neural networks, we decided to learn the basics and build a simple CNN model at the beginning. Feature extraction functions, ReLU function, and max-pooling were implemented and put into their corresponding layers in the Model class that includes the design and function of the CNN.

Once all the fundamental functions of CNN were written, we started training the model with two images - one healthy cell and one infected cell. The CNN design was proven to be working properly, since the weights were able to be updated during training. However, as the size of the dataset increased, the features that were extracted from infected cells are not enough for a promising result. This is due to the imbalance of healthy cells and infected cells in the dataset. More feature extraction algorithms will need to be implemented in the future semesters to account for this.

Although the CNN model has not yet worked with a large dataset, it is proven to be working on a small scale, and the team members were able to learn the basic knowledge of CNN while working on the project. Overall, the team has made significant progress this semester. Cells on a microscopy image can be successfully segmented and the classification model was proven to execute correctly. More optimization of cell segmentation and counting, along with improvement of the cell classification process, will be the goals of future semesters.

VIII. Next steps

At present, the segmentation algorithm has a better result than the previous semester. However, there are a number of things that can be improved and optimized further. One of the first steps forward would be to optimize the regional minima detection method by automating the h-transform applied to the image before K-means clustering. As described in earlier sections, lower h-values result in a better segmentation of heavy clusters while higher h-values result in over-segmentation of heavy clusters but a better segmentation of loosely clustered cells of the image. A potential optimization could be the implementation of an algorithm that uses a connected components approach to apply the H-minima transform procedurally before the K-means algorithm. This would allow each cell component in the image to have a unique H-minima transform applied that uses the most optimal h-value which would result in the most accurate segmentation, rather than a single h-value being applied to the entire image.

Secondly, we can further work on the CNN that was trained this semester. Presently, it is in its early stages of development and has a single neuron. The convolutional layer has simple edge detection filters. In the future, more convolutional layers can be added as well as more feature extractors in the layers. The CNN could also be modified to identify multiple classes of cells instead of just two (infected or healthy).

Thirdly, we can get images that include more anomalous cells. At present, there is an imbalance of healthy and infected cells in the dataset that causes the classifier to classify cells as

mostly healthy even when the cell is not. We can also oversample the infected cell images and undersample the healthy cells during training for a balanced dataset.

IX. References

[1] F. Meyer, “Color image segmentation,” *1992 International Conference on Image Processing and its Applications*, 1992, pp. 303-306.

[2] L. Vincent and P. Soille, “Watersheds in digital spaces: An efficient algorithm based on immersion simulations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, 1991.

[3] T. Pachin, Y.-H. Lin, A. Gatewood, A. Prabhu, N. Manglore, L. Hanna, and N. Narra, “VIP IPA Nuclei Detection and Counting Team Final Report Fall 2021”, Purdue U., West Lafayette, IN, USA, Dec. 2021

[4] A. V. Oppenheim, A. S. Willsky, and S. Hamid, “Linear Time-Invariant Systems,” in *Signals and Systems,* 2nd ed., Hoboken, NJ, USA: Prentice-Hall, 1996, ch. 2.

[5] S. Beucher, “Watersheds of functions and picture segmentation,” *ICASSP '82. IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 1928-1931, 1982, doi: 10.1109/ICASSP.1982.1171424.

[6] F. Tkaczyk, (n.d.), "How to Read Topographic Maps." Alderleaf Wilderness College, Aug. 26, 2020, wildernesscollege.com/how-to-read-a-map.html.

[7] G. Rebala, A. Ravi, S. Churiwala. (2019). Machine Learning Definition and Basics. In: An Introduction to Machine Learning. Springer, Cham. doi: https://doi-org.ezproxy.lib.purdue.edu/10.1007/978-3-030-15729-6_1

[8] A. Leon-Garcia, *Probability, Statistics, and Random Processes for Electrical Engineering*, 3rd ed., Hoboken, NJ, USA: Prentice-Hall, 2008.

[9] R. M. Haralick, S. R. Sternberg and X. Zhuang, “Image Analysis Using Mathematical Morphology,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 4, pp. 532-550, July 1987, doi: 10.1109/TPAMI.1987.4767941.

[10] A. Rosenfeld and J. L. Pfaltz, “Sequential operations in Digital Picture Processing,” *Journal of the ACM*, vol. 13, no. 4, pp. 471–494, 1966.

[11] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno. 2008. 2D Euclidean distance transform algorithms: A comparative survey. ACM Comput. Surv. 40, 1, Article

2 (February 2008), 44 pages. doi: https://doi-org.ezproxy.lib.purdue.edu/10.1145/1322432.1322434

[12] F. Yi and I. Moon, "Extended K-Means Algorithm," *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, 2013, pp. 263-266, doi: 10.1109/IHMSC.2013.210.

[13] N. H. F. Ismail, T. R. M. Zaini, M. Jaafar, & Pin, N. C. (2016, August 29). H-minima transform for segmentation of structured surface. MATEC Web of Conferences. Retrieved February 22, 2022, from https://www.matec-conferences.org/articles/matecconf/abs/2016/37/matecconf_icmer2016_00025/matecconf_icmer2016_00025.html

[14] V. Ljosa, K. L. Sokolnicki, and A. E. Carpenter, *Annotated high-throughput microscopy image sets for validation*, vol. 9, no. 7, pp. 637–637, Massachusetts, USA: Nature Methods, 2012.[Online]. Available: https://bbbc.broadinstitute.org/BBBC041

[15] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks," *2017 International Conference on Communication and Signal Processing (ICCSP)*, 2017, pp. 0588-0592, doi: 10.1109/ICCSP.2017.8286426.

[16] M. Varshney, P. Singh. Optimizing nonlinear activation function for convolutional neural networks. SIViP 15, 1323–1330 (2021). doi: https://doi-org.ezproxy.lib.purdue.edu/10.1007/s11760-021-01863-z

[17] V. M. Vargas, P. A. Gutiérrez, J. Barbero-Gómez and C. Hervás-Martínez, "Activation Functions for Convolutional Neural Networks: Proposals and Experimental Study," in IEEE Transactions on Neural Networks and Learning Systems, doi: 10.1109/TNNLS.2021.3105444.

[18] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

[19] A. O. Salau and S. Jain, "Feature Extraction: A Survey of the Types, Techniques, Applications," *2019 International Conference on Signal Processing and Communication (ICSC)*, 2019, pp. 158-164, doi: 10.1109/ICSC45622.2019.8938371.

[20] H. A. Nugroho, S. A. Akbar and E. E. H. Murhandarwati, "Feature extraction and classification for detection malaria parasites in thin blood smear," *2015 2nd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, 2015, pp. 197-201, doi: 10.1109/ICITACEE.2015.7437798.

[21] R. Givan. (2022). Machine Learning Lecture 8, Neural Networks Lecture 28 [Restricted Access Notes].

[22] V. Ljosa, K. L. Sokolnicki, and A. E. Carpenter, *Annotated high-throughput microscopy image sets for validation*, vol. 9, no. 7, pp. 637–637, Massachusetts, USA: Nature Methods, 2012.[Online]. Available: https://bbbc.broadinstitute.org/BBBC002

X. Appendix

Tikhon Pachin

This semester was my concluding semester for my senior design. The past two semesters working on the team I have assumed a very definite role of the team leader. I have managed to maintain the role even with the conflict of my schedule with the class meeting time. My biggest role was to make sure that everyone stayed on track and contributed evenly to the team. I also made sure that this semester's onboarding went as smooth as possible for the new team members. I held the team meetings, distributed tasks (making sure that I do the larger portion of the work due to the grade level and stakes in the project), and interacted with the Office of Undergraduate Research.

My main contributions, however, were in the technical aspects, such as coding and prototyping. I was the point person to whom the teammates came with debugging issues. I also helped to introduce new topics, like machine learning, and other conceptual theoretical ideas to the team, when help with understanding was needed.

The biggest coding task that I have individually tackled was the flooding procedure with overflow protection. The initial iteration of the connected components algorithm was introduced there alongside the FIFO stack architecture and Breadth-First Search techniques. The task ended up being more challenging than I expected, so I spent a fair share of time targeting the issue.

The other large component of the project that I have implemented was the prototyping of the CNN model architecture as well as coding implementation of the convolutional layer, the model design and training. Most coding wrappers within the repository were created by me also.

I have also contributed to the brainstorming and debugging stages of the cell counting algorithms.

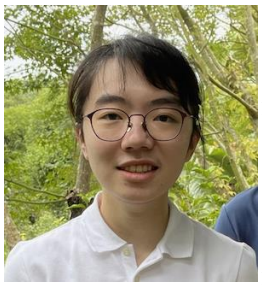
Every part of the semester's work had my contribution in one way or another, except for regional minima detection algorithms.

This semester I have also learned a lot about Convolutional Neural Networks and Machine Learning in general. I was taking an AI course at Purdue this semester as well and I got to implement my new knowledge in practice for this project right away. Nevertheless, I learned a fair share in this class while researching different types of CNNs and how they can be used.

From an organizational point of view, I was also the one who created the poster and completed a large portion of the final report.

The result of this semester's work is very promising and fairly realistic. I have only ignited further my interest towards processing medical imagery.

Yu-Hsuan Lin



This semester was my second semester in VIP and IPA. With the experience in image processing I had last semester, I was able to understand faster the new topics that were introduced to me this semester. This semester was an interesting semester because I got the chance to improve the code I wrote last semester, saw the improvement in the results, and was able to learn some of the basic knowledge in machine learning about convolutional neural networks.

For the first half of the semester, I focused on regional minima detection for flooding in watershed segmentation. The previous method idea for regional minima detection was to run dilation on a topographical map; however, since the detected minima in clustering cells were underestimated, I decided to try on using the K-means algorithm for regional minima detection after reading multiples research papers. Fortunately, the k-means algorithm was able to group the original regional minima into our desired locations, which are the centers of the cells, with the implementation of the H-minima transform written by Sean. Both the detected regional minima in clusters and individual cells are placed accurately. There was a challenge that I faced while working on the K-means algorithm. The algorithm requires initialization of clusters location; however, depending on the method of initialization, the output of the result may change. To solve this problem, I looked into different papers that worked on the K-means algorithm I wrote an initialization of clusters with the one I thought would work the best for our project. Luckily, this method did provide a better output of regional minima for grouping the original regional minima.

For the second half of the semester, the team shifted focus to CNN. This was a new topic to me. Although I am always interested in machine learning, there is still a lot for me to learn. For understanding the concepts, I watched a lot of lecture videos and also read some introductory papers on this topic. After having some confidence to start working on CNN, I implemented a max-pooling function, which also utilized the experience I had in image processing. After then, I wrote a histogram-based feature extraction function for extracting the statistical information of an image. Although the team has yet to combine the function with our model, I learned a lot about CNN while discussing the function itself and also the computed statistical information.

I was also able to help my teammates when they had questions that were related to my work and was able to find out some of the errors that I made during the discussion. While having conversations with my teammates, I was able to ask some questions for myself to understand more about the topic that I am working on. It was a great experience to learn both during the discussion and the work I had this semester. In this report, I worked on writing the K-means algorithm, convolutional layer, pooling layer, feature extraction, and the conclusion.

K Annapoorna Prabhu

This semester was my second time with VIP and IPA. I was more equipped this time around, with my prior experience in Python as well as working in the project team for the previous semester as well. For the first half of the semester, I came up with the hypothesis that preprocessing heavy clusters separately would address the under segmentation problem we had faced constantly last semester. I read about contour detection and drawing as well as connected components to extract distinct blobs of the image. I implemented the connected components algorithm. Although it is not a very complex algorithm, I still struggled with debugging it as I kept getting striations in the image. To address this, I tried a variety of methods such as implementing an algorithm that would merge equivalent labels (equivalency tree), using a priority queue approach and linked list approaches. However, they did not work as I expected, and I spent more time on connected components than I wanted to. I learned how to debug effectively as well as graph and tree traversal algorithms along the way. This approach for selective preprocessing was abandoned, however, the algorithm plays an important role in cell counting and a similar version of this is implemented for overflow protection as well.

For the CNN part of the project, I was tasked to find datasets as well as parse them and prepare training data for the CNN. I learned how to parse and read JSON files as well as handle .npy files for storing data on each individual cell. I also learned how to use the Python library - Pillow, which was used to open and save images. As I had no prior experience or knowledge of neural networks, a lot of time was spent learning about CNNs, loss functions, and convolutional layer filters. Towards the end of the semester I read about histogram-based feature extraction and histogram of oriented gradient detectors as a possible feature extractor.

I was also involved in the administrative and logistic tasks in the team such as submitting presentations and getting them ready for the class, and formatting. In the beginning of the semester, I helped some of the members figure out issues with their IDEs. I was also involved in testing the areas of interest detection in conjunction with the connected components algorithm for heavily clustered cells. In this report, I worked on the Related Work section, Watershed Segmentation Overview the Connected Components and the Next Steps sections.

William Stevens

This semester was my first experience in a VIP program. My lack of prior knowledge of image processing techniques and experience in a group research setting provided a unique challenge. However, I overcame it this semester thanks to support from my group members, the program mentors, and the graduate students. During the first half of my semester, my time was mostly spent getting up to speed on image processing techniques, becoming situated with the process of collaborative version control across different operating systems, and building familiarity with previously written algorithms through code analysis, reproduction, and testing. It took the first two weeks to get up to speed on the image processing knowledge necessary to contribute to the project. Fortunately, there was an abundance of professional lecture resources, past project documentation, and the teachings of current team members to help me with this. As for getting situated with the version control environment, this process took me around two weeks as well. In order to work with code written by other members and effectively test code I had written, I needed to set up Windows Subsystem for Linux, a process which took many hours communicating with other team members and graduate students. Fortunately, I documented my process extensively in Confluence, so any future student who needs to work in a bash terminal on Windows will hopefully save some frustration by reading my documentation.

Once I was set up with WSL to work with the code base, I began practicing some of the more basic image processing techniques by writing the programs myself and learning along the way. For about three weeks, I spent my time writing and testing some of the fundamental image processing algorithms, such as: RGB-grayscale conversion, city-block distance transform, Sobel edge detection, and Gaussian blurring. I believe the experience of practicing such algorithms further grounded my understanding of interpreting images as matrices that can be operated on mathematically, and that this laid the groundwork for some of the algorithms I designed in later weeks. When the second half of the semester began, the watershed segmentation process was nearly complete. Once it was, my task was to analyze the algorithms in the sequence and find ways in which they could be optimized. While I investigated other algorithms such as the areas of interest algorithm, I found that our regional minima detection algorithm had the most potential for optimization.

So, for the remainder of the semester, I worked on improving our regional minima algorithm through the development and incorporation of supplemental algorithms. The first of which was the cell counting algorithm, which used our connected components labeling algorithm to calculate an estimate for the number of cells to be inputted into our K-means algorithm, which increased the overall automation of our sequence. The other algorithm I designed and developed was the improved minima algorithm, which aimed to apply K-means regional minima detection at a procedural scale with varying h-values applied in different sections of the image. This algorithm was left incomplete, but has potential for future work. In this report, I was tasked with writing the Goal, Cell Counting, Cell Segmentation Results, and Next Steps sections. I also spent time proofreading the sections of other team members.

Shang-Hsuan Lee

This is the first semester for me to be part of VIP and IPA. Although I have worked as an intern in a research team dealing with computer vision, after I joined the team, I immediately realized that I still lack the fundamental knowledge about image processing. Therefore, the first task for me was to watch the image processing tutorial given by professor Delp and write the suggested programs on my own. After watching the tutorials, I finished the Sobel edge detection, histogram program, grayscale conversion, thresholding, and Gaussian blur program. With all the practices, I was more clear about the general idea of how the image is stored in the computer and the basic operations in image processing.

After getting on track, I start to look at the team's program which was done last semester and also read the final report to understand what the team is doing and the progress of the project. The first problem that I encountered was my lack of experience of working with pytest and my computer didn't have the environment to run the team's code, so I had to set it up with Windows Subsystem for Linux (WSL). Thanks to the help from Anna and William, it didn't take me long to set up the environment. The first program that I wrote was the Euclidean distance transform, although it was already implemented, the run time of the algorithm was relatively long, so I tried to find a better way to create a more efficient algorithm and by the mean time get used to working with pytest. However, after a lot of reading and testing, the run time of my program wasn't better, hence, I moved on to work with Sheila on the regional minima detection.

The next program I wrote was the H-minima transform. Sheila had already implemented the k-means algorithm successfully, but some of the detected regional minima were misplaced. After discussion, we think implementing H-minima transform can help solve the problem. It turned out to be a simple algorithm, so it only took me two weeks to understand it and finish the program. Combining the code of mine and Sheila's, the result has improved a lot.

For the second half of the semester, the team proceeded to a second phase of the project where we introduced classification with CNN. Even though I had some experience working with computer vision, I was only using the libraries, but didn't understand the concept and the math fully. So I spent a lot of time on readings to understand the details of machine learning and CNNs. I wrote the activation function for the model and worked with Sheila on histogram-based feature extraction. Although our current model only consists of one single neuron, we can find out that the weights are updating by tracking them during the training process, this indicates that our proposed model is feasible. However, we still need to come up with more feature extraction methods in the convolutional layer and train the model with a more evenly distributed dataset to get a relatively accurate model.

Throughout the semester, I have encountered several problems, but after reaching out to the team members, I was able to overcome the difficulties that I faced. Moreover, thanks to the research on machine learning, I have grown more interested in the topic of artificial intelligence and understand it more.