Chromosomal Image Recognition A.I.

Alexander Wurts, Daniel McDonough, and Surya Vadivazhagu  
Worcester Polytechnic Institute December 2018

**Abstract**—Cancer research is being slowed by the lack of open-source tools to detect damaged deoxyribonucleic acid (DNA). Automating this process would save time, money and energy. Currently, there are many methods to detect cell nuclei, but none classify cell DNA as damaged or healthy. We have incorporated these methods to develop an accurate cell detection, and damaged cell classification application. We discuss generally how and why we address this problem, followed by the specifics of the program pipeline, then our results and how it compares to other possible methods, and concluding with a statement about any future additions.

*Keywords— Nuclei Detection; Laplacian of Gaussian; Foci Clustering*

1. **Introduction**

Currently, cytologists use manual examination of DNA damage due to the lack of open-source, high-accuracy cell and foci detection software. All known openly available software options fail to detect cell nuclei in addition to classifying nuclei based on expressed foci. Successfully automating the detection and classification process would result in a tool with much faster and higher accuracy of foci detection to be used in all forms of damaged DNA tests, which are used mostly in cancer research. Using the Python image processing libraries SciPy, Sci-Image and OpenCV, we developed the Chromosomal Image Recognition A.I. (CIRA) capable of processing high quality microspectroscopy videos of *Cos-7* cell cultures, whose nuclei is labeled with red fluorescence protein (RFP) and foci labeled with green fluorescence protein (GFP)[13].

CIRA’s main goal is to detect cells, then classify them as Unhealthy or Healthy. There has been lots of previous research in general and species-specific cell detection done already[14]. Un/Healthy cell nuclei classification however, has little application research, as detecting individual foci on a cell can only be done on images made with high resolution color microspectroscopy which is very expensive and difficult to find a public dataset for. If a program satisfying these needs can be developed, the speed of induced DNA damage testing will increase dramatically, which is used in cancer research. We will discuss more about our solution pipeline generally, followed the specifics of the pipeline, our results and how it compares to other methods, and finally our closing statement about any future additions.

1. **Background**

When trying to solve the problem of classifying damaged DNA, one must solve the first problem of detecting nuclei. To accomplish this, we use the blob detection method, Laplacian of Gaussian (LoG)[15], on the red video color channel which labels the nuclei. LoG is known to be highly accurate but slow compared to other blob detection methods such as Difference of Gaussian (DoG) or Determinant of Hessian (DoH)[15]. Thus it was chosen over others Due to LoG determining “blobs” by the local maxima of color on a black background it is also unable to detect darker cells that have not been labeled properly by RFP. In order to address these disadvantages, we simply ignore darker cells, and cells slightly out of frame, due to the likely inability for humans to be able to detect and or accurately classify these cells. The classification of unhealthy and healthy was done by K-Means and Logistic Regression algorithms to detect correlations between a cell and detect if it is healthy or not [16], based of different features of a cell. This is optimized by increasing the number of clusters and determining a percentage of unhealthy cells in each cluster for an accuracy measurement of damage on each cell. This method of classification gives a likelihood of a given cell nucleus being damaged, without looking at GFP, but may produce more false negatives or false positives than other classification methods. Furthermore, we will describe the pipeline more in depth, going through each step of preprocessing, nuclei detection, and un/healthy classification, and evaluation of our results.

1. **Methodology**

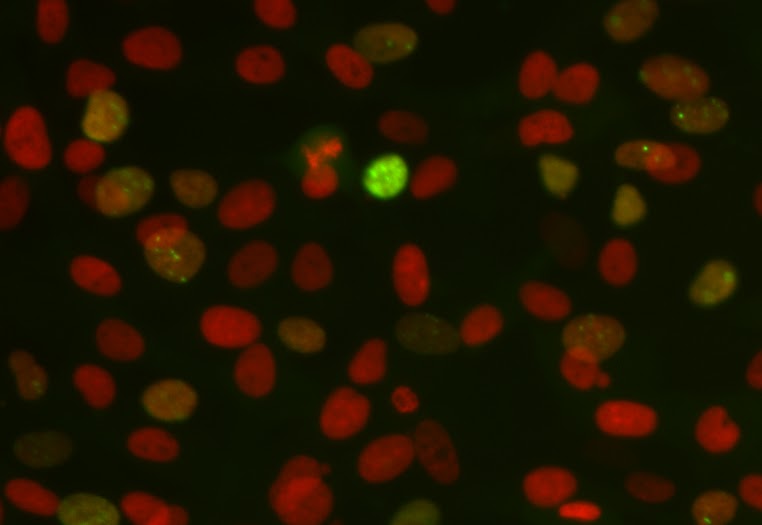
Given a video of a *Cos-7* cell culture, whose nuclei are labeled with RFP and damaged foci are labeled with GFP, our algorithm uses image detection algorithms in conjunction with classification thresholds to accurately detect and classify cell nuclei as healthy and unhealthy, minimizing the amount of false positives and negatives. To do so, we developed the CIRA pipeline, which preprocesses the given cell video, detect cell nuclei using Laplacian of Gaussian, and classify them as healthy and unhealthy by K Means and Logistic Regression.

1. *Preprocessing:*

Prior to running CIRA, we must first organize, and clean our test datasets. We used two datasets, one obtained from Manning Lab at Worcester Polytechnic Institute (WPI) [1] and one obtained from the Kaggle 2018 Data Science Bowl [2].

1. *Manning Lab Video Dataset:*

The first dataset, from Manning Lab, contains 2 .ND2 movie files depicting, a culture of *Cos-7* cells with RFP labeled nuclei and GFP labeled foci, over a 36-hour time period, with 5 minutes being between each frame. During the video, cells experience all phases of the cell cycle, as cells replicate overtime. Researchers may induce error prone replication, which often creates damaged cites or foci. Taken with high resolution microspectroscopy, the movies are 2560 pixels x 2160 pixels and show an overlay of the red and green channels in motion. These movie files are not useable for OpenCV to analyze, so we must use Nikon ND2 Reader [3] to convert the binary movie files into .TIF frame image data. From the translation from movie to frame data, some channel information is lost, due to errors in the conversion and data transfer. Frames of capture (from the video) so we must remove any Red channel frame data that does not have a Green channel frame data and vice versa. Additionally, we manually remove any overall blurry frames that have been altered by water collection on the recording lens. With the cleaned frame data, we are able to separate the frames by each color channel, resulting in 4 folders: Bad\_Images, Red\_Channel, Green\_Channel, and Red\_Green\_Channel. The final dataset size is 120, 549, 549, and 549 files, for each folder respectively, using both movies.

****

**Fig 1. Manning Lab Data Example:** *This is a segment of a frame from one video, with both red (nuclei) and green (damaged foci) channels. Green dots are damaged DNA, however often cells repair their DNA, but the green intensity is lowered drastically, resulting in this yellow-orange haze.*

1. *Kaggle 2018 Data Science Bowl Dataset*

The second dataset, from the Kaggle 2018 Data Science Bowl is a collection of nuclei of various types of cells in different resolutions, as well as a ground truth mask for each instance of a cell in the dataset. Due to the varying data of this set we must clean the dataset to only use grayscale images of cells similar to *Cos-7,* by checking the colors and brightness of the cell images we can determine if they are non grayscale, or black on white, rather than grayscale. Then remove any missed data points manually. These are done just for testing and metric data collection. The final amount of source images resulted in 669 with 10,422 total ground-truth masks for cell instances, and 10,422 circle packed truth masks.

|  |  |
| --- | --- |
|  |  |

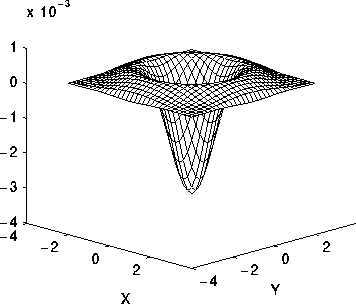
**Fig 2. Kaggle Data Example:** *This is an example data instance after data cleaning (Left) and an example of one cell as a mask that is regarded as ground truth (Right).*

1. *Cell Detection:*

Using the Red\_Channel folder, we run the blob detection algorithm, Laplacian of Gaussian to detect cells from each given frames [17]. As the name implies, a given an input image f (x,y), is convolved by a Gaussian kernel, then by a Laplacian kernel. [30] This can be simplified to equation (1) where x, y are the coords of each pixel in a given source image, and σ is the scale of the filter.

(1)

Once the combined filter, as dictated in equation (1) has been applied, we then have to detect scale-space maxima/minima, with respect to both space and scale, visualized in Figure. 3 below.



**Fig 3. The 2-D Laplacian of Gaussian (LoG) function:** *The x and y axes are marked in standard deviations (σ), with radius*  [30]

Pseudocode for SciPy’s Laplacian of Gaussian implementation [19]:

Pseudocode for Laplacian of Gaussian:

***Input:*** *Max Sigma, Min Sigma, N-iterations,*

*Threshold, Overlap*

***Output:*** *List of centroids and sigmas for all determined “blobs”*

*Choose N 𝝈 between Max and Min 𝝈.*

***For*** *each 𝝈:*

*Apply LoG filter*

*Find the local maximas (within a given threshold) over a mask as “blobs” with radii of*

*Combine determined blobs whose overlap percentage exceeds the given allowed overlap.*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Using the Kaggle 2018 Data Science Bowl dataset, our LoG was optimized to find the correct number of blobs over 75% of the dataset and testing on the other 25%. Testing the min, max, overlap, and iteration count with a static threshold of .01, in order to maximize the Mean Accuracy Percentage. Using these parameters as a baseline, we then manually manipulated the parameters to decrease run time. Once optimized on the known data set, we run LoG on the unlabeled Manning Lab dataset and crop each projected “blob” indicated by its calculated centroid and radius [28].

1. *Cell Classification:*

In order to have test labels on the Manning dataset, we determined that cells with damaged foci have a large green value range in reference to the mean green value of the cell image. Therefore, we can approximately label unhealthy cells by determining if the max green value of a LoG determined cell is equal to or greater than the threshold calculated by equation (2).

(2)

Where Avg. is the average green value in a given cell image. Those satisfying this threshold was determined as unhealthy, and those who did not was labeled healthy. Of course, this does not produce a completely clean dataset. So in addition, we removed any cells that are not fully visible or in the middle of cell replication, because we cannot fully determine if the cell is healthy or not, then manually reclassified any other incorrectly labeled cells.

Once the data has been processed it is placed in two folders, one with cancerous cells and one with healthy cells. The images need to be processed so that each cell is represented by a one dimensional array. First each image is resized to 50 pixels by 50 pixels to standardize the data. Then the image is flattened to a one dimensional array. The image is now represented by a 2500 value array of tuples that contain the color values of the image in RGB. The green and blue values are removed leaving the red integer band. Each item in the array represents the brightness of a given red band of the pixel from 0 to 255. We then take the first 20,000 healthy and 20,000 unhealthy cells and save them in a comma separated (CSV) file with the label ‘unhealthy’, if it is potentially cancerous, and the label ‘healthy’ if it is not cancerous.

Now that have the data is in the CSV file the training process can start. The data is loaded using Pandas into a DataFrame. The data is then run through the Scikit Learn implementation of the three clustering algorithms with 2 and 5. The three clustering algorithms we used were:

* **Agglomerative Clustering with Ward:** Agglomerative clustering is the principle of building up a group of individual clusters into a final one. Each item starts in its own cluster and is grouped by similarity later on. Ward’s method is a statistical principle that was chosen due to how it is able to create compact, proportional feature clusters [32].
* **Agglomerative Clustering with Complete Link:** Complete link was used because unlike single-link(which takes the similarity of two clusters’ closest areas and ignores the dissimilar parts), it calculates the similarity of two clusters based upon the similarity of the most dissimilar objects in each cluster [34].
* **K-Means:** Based upon a fixed scalar *k*, K-Means operates by clustering each data point into *k* clusters dependent upon a randomly declared centroid for each cluster. The centroid is then recalculated as the mean value/ coordinate of each cluster. From this grouping similarities in datasets begin to emerge [33].

In order to assess the quality of the clustering algorithms we used the homogeneity\_score and an accuracy measure, as well as visualizing the results by combining the images of all the cells. The Homogeneity score is way of measuring if each cluster contains only data points which are members of a single class. As well as homogeneity we used equation (3) to compare the clustering results to the classification results. After a clustering algorithm produces a number of clusters we label each of these clusters with the class that occurs the most within. When a data point is fit to the nearest cluster it takes on the label of that cluster and we use that for classification. We use this method to go through all test data and then run the following accuracy formula based on the classifications [31].

(3)

In order to find the number of clusters that provided the best classification we looked at 2 to 20 clusters and looked at this cluster amount produced the best accuracy. We produced visualizations for two and five to understand the results in more detail.

The image visualization works by combining every cell in a specific cluster into a single image. The red cells are considered healthy, the green cells are damaged and potentially cancerous. The red cells are shown only using the image red band data. The green cells are shown using only the green band data. A visual inspection of the image can help determine the quality of the clustering based on the frequency of green and red cells. The ratio of green to red is important therefore we took a small subsection of each photo and used that for comparison. The image visualization was created using Pillow and Numpy.

A graph is also produced to visualize the results. The graph shows the quantity of healthy and cancerous cells in each cluster. Green is healthy and red is cancerous and they are sorted by the count of cancerous cells in each cluster.

We also tested classification algorithms. The training algorithms used were:

* **Support Vector Machine (SVM) with default parameters:** SVM is a classifier that creates a hyperplane over the graph representation of the dataset. SVM is able to separate classes by creating a hyperplane that has the highest margin, i.e. the highest distance between the points in the dataset. This will result in a hyperplane highlighting the most difference between the classes [12] .
* **Logistic Regression with default parameters:** Logistic regression is a method of predictive analysis used for calculating the probability of an event given past, binary data. [12]
* **Multi-level Perceptron Neural Network (MLP) with the parameters shown in equation**. An MLP is a classifier based on logistic regression where input is modified using a transformation (Φ). There are a total of three layers in an MLP with a single hidden layer- the input, output, and hidden layer. Having one hidden layer makes an MLP a universal approximator, which is why it was chosen as a training algorithm [12] [35]. The final parameters are dictated in equation (4) below.

*MLPClassifier(solver=’lbfgs’, alpha=1e-5, hidden\_layer\_sizes=(8), random\_state=1) (4)*

The classification algorithms were tested using the same csv file produced for K-Means. The data is loaded using Pandas into a DataFrame. The data was split 80/20 into training and testing data respectively. The results were assessed on the test split using the score method built into Sk-Learn’s classifiers, and the same visual process as described in clustering.. The score produces an accuracy based on the number of correctly labeled cells over the total number of cells classified, as shown in equation. (5)The same large scale cell visualization used for clustering is used to assess the classification algorithms.

1. *Assessment Protocols*

Measuring CIRA’s cell detection precision, we made use of the Mean Average Precision (mAP) confidence measure [21] and the 2007 PASCAL Visual Objects Classes challenge (PVOC) [20]. This metric is represented by the following set of equations and pseudo code:

(5)

(6)

Pseudocode for mAP:

**Input:** *Set of predicted and ground truth masks*

**Output:** *Mean Average Precision of an algorithm over a whole dataset.*

*For all thresholds, 0 to 1, by 0.1 interval:*

*For each data point in the data set:*

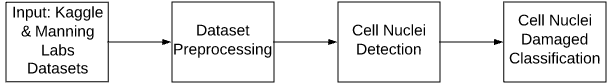
*Compute Precision where:*

*TP = IOU ≥ threshold*

*FP = IOU < threshold*

*Average Precision = mean of Precision from all data points.*

Where mAP is the mean of Average Precision, PVOC is the Average Precision at a constant .5 threshold. mAP is a good tool for evaluation due to how it represents an intersection of a union, meaning that for our purpose [21], it would suffice for comparing predicted cell images to known/target cells. We would take the matrices of the predictive image as well as the true image and run mAP on it to determine whether the binary masks were the same. If there was a difference, indicating a lack of precision by the classifier, that difference would be saved and at the end, all the differences would be averaged together to create the mean of the algorithm’s average precision. mAP’s accuracy is a good representation when there is a normal distribution present within the sample data. If several outliers are present however, then the mean average precision will be skewed. This problem is avoided by also measuring the average normalized Euclidean distance [22] in relation to the mAP measurement. A general summarization of our pipeline methodology can be seen in Figure 4.



**Fig 4. CIRA Flowchart:** *CIRA’s pipeline, begins when given the Kaggle and Manning Labs datasets, to be preprocessed. CIRA then trains and tests LoG on the Kaggle dataset. The resulting parameters are then used on the Manning Labs dataset. The resulting crops are then classified as unhealthy and healthy.*

1. **Results**

All training and testing was done on a computer with the following specifications:

* Intel i5-3690k 3.5GHz
* 16GB RAM @ 1600Mhz
* 1TB Hard Drive @ 7200RPM
* Windows 10
* Python 3.7.0 [27]
* Pandas==0.23.4 [23]
* Scikit-Learn==0.20.0 [12]
* MatPlotLib==3.0.1 [25]
* Pillow==5.3.0 [26]
* NumPy==1.15.2 [24]

1. *Cell Detection:*
2. *Laplacian of Gaussian:*

From the Kaggle set, we ran LoG over 456 data points and tested over the remaining 213 points resulting in Table 1.

**Table 1. Laplacian of Gaussian Raw Test Results:** *This table is the results of 213 points of the Kaggle Dataset after optimizing the parameters and increasing the speed by removing unnecessary iterations and variances that would be removed by the removal of overlap.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| mAP | PVOC | Avg. Time | Total Time | Avg. Normalized Distance Between Centroids |
| 0.32 | 0.23 | 0.25 sec | 105.896 sec | 0.602 |

Due to the nature of blob detection we found it to be unfair to classify it on the absolute truth, as IOU tests how well the prediction and truth overlap. It is in the nature of blob detection, to not give exact edges but rather an approximation of a radius in which the object of interest lies. To account for this, we ran each ground truth to a minimal circle packing algorithm, which is dictated in Table 2.

**Table 2. Laplacian of Gaussian w/ Min. Circle Packing Data:** *After manipulation the masks to fill the smallest circle possible around it we can depict, the accuracy of the LoG better, due to the fact that the crop of the cells currently includes other data than the cell itself.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| mAP | PVOC | Avg. Time  (sec) | Total Time  (sec) | Avg. Normalized Distance Between Centroids |
| 0.819 | 0.78 | 0.26 | 105.9 | 0.6014 |

With minimum circle packing of the ground truth, LoG resulted in 81.9% accuracy. This however, does have a large margin of error, due to the minimum circle packing of the ground truth, as this does not produce an exact boundary of what is and is not a cell but rather an approximation of where a cell could lie. In comparison, the Kaggle 2018 Data Science Bowl winners (29) used a Mask Recursive Central Neural Network (Mask-RCNN) to accomplish 60% accuracy of exact boundary nuclei detection of multiply kinds of non-fluorescent cell nuclei.[29] This difference is most likely due to the training of LoG being too fit to the data, in addition to minimum circle packing, and being selective of the training data.

1. *Other methods:*

We also ran tests on other commonly used blob detection, image segmentation, and color segmentation methods as shown in Table 3.

* **K-Means:** Using K-Means with K=2 would determine if a pixel was part of a cell or not. This takes little time and computational energy, but does not segment the cells individually as needed for classification.
* **Watershed:** Using a watershed with binary-otsu additive and ored produced similar results. Both however, was very sensitive to darker cells often disregarding them. But also produced exact boundary masks to determine cell nuclei location.
* **Difference of Gaussian (DoG):** As a blob detection algorithm, DoG produced similar results to LoG but in a faster time frame. DoG however suffers from a lower accuracy, even when accounting for min-circle packing, and get the normalized difference between the centroids as shown in Table. 3.
* **Determinant of Hessian (DoH):** Again,DoH produced similar results to LoG in a faster time frame, but a slightly lower accuracy. But should be noted that it did get a slightly more accurate normalized difference between the centroids. This could be because DoH requires more training.

**Table 3. Other commonly used methods:**

*Training over the same Kaggle dataset, we can see that all other methods are much faster, but less accurate than LoG with min circle packing in Table 3A. It should also be noted that DoG, DoH, and LoG all have similar distances between centroids, which could possible mean that they could be trained further Table 3B. We do not do this however to avoid overfitting.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table 3A** | **mAP** | **PVOC** | **Avg. Time**  **(sec.)** | **Total Time (sec.)** |
| K-Means (k=2) | 0.1 | 0 | 0.049 | 20.49 |
| Water-  shed (Binary | Otsu) | 0.35 | 0.285 | 0.175 | 72.63 |
| Water-  shed (Binary + Otsu) | 0.35 | 0.285 | 0.167 | 61.21 |
| DoG | 0.31 | 0.22 | 0.073 | 30.43 |
| DoH | 0.23 | 0.08 | 0.1 | 41.50 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 3B** | **mAP** | **PVOC** | **Avg. Normalized Distance Between Centroids** |
| DoG | 0.60 | 0.56 | 0.60 |
| DoH | 0.56 | 0.50 | 0.562 |

1. *Clustering Algorithms:*

We used three clustering algorithms: Agglomerative Clustering with Ward (ACW), Agglomerative Clustering with Complete Link (ACCL) and K-Means.

**Table 4. Clustering Results:** *This table depicts the results for simple clustering methods, and the score on the test data.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **ACW** | | **ACCL** | | **K-Means** | |
| N-Clusters of Best Score | 8 | | 18 | | 2 | |
| Homogeneity Score: | 0.432 | | 0.339 | | 0.28 | |
| Score | 0.610 | | 0.712 | | 0.791 | |

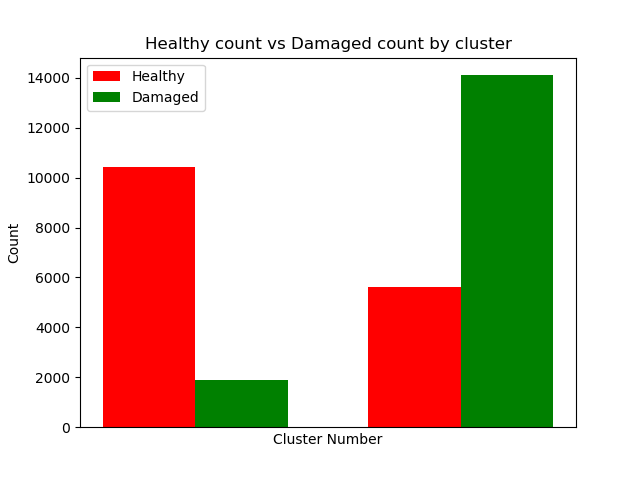
The clusters can be used to label any new data point by putting it into the closest cluster and use the ratio of healthy to cancerous to define a probability of being cancerous or not. For example if a cell was placed into the leftmost cluster in the two cluster K-Means in Figure 5 we would know with 80% certainty that it was healthy. However if it was placed in the right cluster for the same algorithm and amount of clusters, it would have a 49% chance of being healthy. For the 5 cluster results, if the cell was clustered in the rightmost K-Means cluster on the graph it would be almost certainly, classified as unhealthy as shown in Figure 6. Using K-Means, as it has an accuracy score of 0.791, we can produce a likelihood that given a new cell it would be healthy or unhealthy.

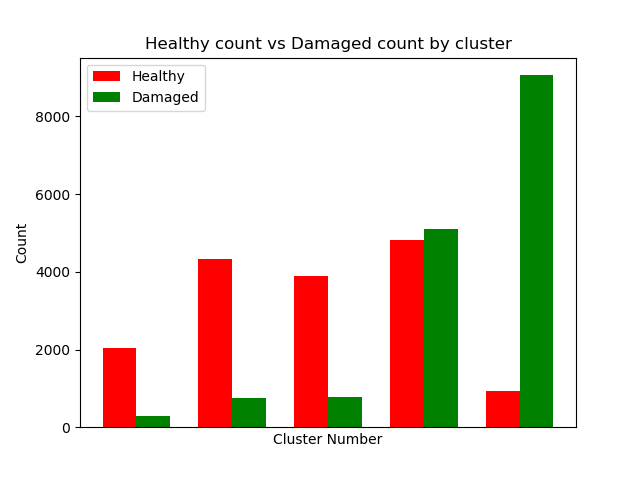
1. *Classification Results*

Based on the test data, Logistic Regression and MLP classified the data with an accuracy of 87% and 86.025% respectively [Table 6]. SVM only attained an accuracy of 49.75%. SVM incorrectly classifies everything as healthy as seen by zero healthy unhealthy cells. In Figure 7, we can see the 87% accurate Logistic Regression and where it may fail.

**Table 6. Classifier Results:** *Green is unhealthy. Red is healthy. The score represents the accuracy of the given classifier. Each graph shows the clustering in each group. The left two bars are the healthy class. The red bar are the true-positive results and the green bars are the false positive results. The right bar is the unhealthy class. The red bar are the false-negative results and the green bars are the true-negative results.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Support Vector Machine** | | **Logistic Regression** | | **MLP** | |
| Accuracy  Score | 0.4975 | | 0.87 | | 0.86025 | |

**Fig 5. K-Means K=2 Graph:** *This shows the best clustering result among the 3 chosen methods K-Means, with K=2.*



**Fig 6. K-Means K=5 Graph:** *This shows the clustering for 5 clusters using K-Means. K=5 significantly outperformed K=2 in terms of precision, with the 4th cluster being the only exception.*

|  |  |
| --- | --- |
|  |  |

**Fig 7. Logistic Regression Visualization:** *This shows an enlarged version for the visualization for Logistic regression healthy (Left) and Unhealthy (Right) test results. Green is unhealthy. Red is healthy.*

1. **Conclusion**

At this current stage CIRA takes about 4 hours to run on the whole dataset. We are working to minimize this runtime, as the current runtime is unsatisfactory. The main advantage of a large amount of clusters, even when this is a two class dataset is the ability to create a probability of unhealthy or healthy. Using five clusters increased our certainty of unhealthy or healthy. The other results showed a low accuracy. We found that K-Means clustering shows the most accurate classification of data, as evidenced in the results. CIRA is based off various theories regarding image segmentation and classification. Its advantages are the rapid ability to process images of cells, given fluorescent labeling of the cells.

In future research we would test other cell detection methods such as the Mask-RCNN used in the Kaggle 2018 Data Science Bowl. We would also like to implement cell tracking, and individual foci detection, rather than just determining if there is a foci. Perhap, this would be possible on a more powerful machine, enabling faster data analysis. We would like to have applied, a bootstrapped K-Means and Logistic Regression algorithm to obtain confidence intervals that a given cell had foci. In addition to more neural network analysis of the data as it showed that MLP performed quite well. Potentially, implementing a Convolutional Neural Network would result in more interesting results due to high feature space.

Overall, CIRA finds its strengths in strong image segmentation and classification.

The method’s accuracy in classification could be improved by better evaluation, but the use of the Laplacian of the Gaussian sets CIRA back from preceding papers to process images faster. Further improvements include creating a model to determine the “best” threshold for the green channel to determine a cell’s health, to prevent manual cleaning. More confidence in blob detection and increased feature detection now is the future of CIRA, along with faster detection time.

**References**

|  |  |
| --- | --- |
| [1] | D. A. Manning, "Manning Lab," Worcester Polytechnic Institute, 2016. [Online]. Available: http://labs.wpi.edu/manninglab/. [Accessed November 2018]. |
| [2] | Booz Allen Hamilton and Kaggle, "2018 Data Science Bowl," Kaggle, April 2018. [Online]. Available: https://www.kaggle.com/c/data-science-bowl-2018. [Accessed November 2018]. |
| [3] | S. Kiriyama, "Nikon ND2 Reader," Nikon Instruments Inc., 4 May 2014. [Online]. Available: https://imagej.nih.gov/ij/plugins/nd2-reader.html. [Accessed November 2018]. |
| [4] | T. P. Ledesma-Fernández E, "Fluorescent foci quantitation for high-throughput analysis," Journal of Biological Methods, 19 July 2015. [Online]. Available: http://www.jbmethods.org/jbm/article/view/62/54. [Accessed November 2017]. |
| [5] | A. Rosenberg and J. Hirschberg, "V-Measure: A conditional entropy-based external cluster evaluation measure," *Association for Computational Linguistics,* pp. 410-420, 2007. |
| [6] | E. Yilmaz and J. A. Aslam, "Estimating average precision with incomplete and imperfect judgements," *CIKM '06 Proceedings of the 15th ACM international conference on Information and knowledge management,* pp. 102-111, 2006. |
| [7] | A. Patil and M. Dhanvijay, "Blob detection technique using image processing for identification of machine printed characters," *INTERNATIONAL JOURNAL OF INNOVATIONS IN ENGINEERING RESEARCH AND TECHNOLOGY,* vol. 2, no. 10, pp. 1-8, 2015. |
| [8] | E. Meijering, "Cell Segmentation: 50 Years Down the Road [Life Science]," *IEEE Signal Processing Magazine,* vol. 29, no. 5, pp. 140-145, 2012. |
| [9] | M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn and A. Zisserman, "The PASCAL Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision.* |
| [10] | B. E. Boser, I. M. Guyon and V. N. Vapnik, "A training algorithm for optimal margin classifiers," *COLT '92 Proceedings of the fifth annual workshop on Computational learning theory,* pp. 144-152, 1992. |
| [11] | P. K. Agarwal and N. H. Mustafa, "k-means projective clustering," *PODS '04 Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems,* pp. 155-165, 2004. |
| [12] | Pedregosa et al., "Scikit-learn: Machine Learning in Python," *JLMR 12,* pp. 2825-2830, 2011. |
| [13] | E. Ledesma-Fernández and P. Thorpe, "Fluorescent foci quantitation for high-throughput analysis", *Jbmethods.org*, 2018. [Online]. Available: http://www.jbmethods.org/jbm/article/view/62/54. [Accessed: 04- Dec- 2018]. |
| [14] | R. Crane, “Automatic Cell Detection and Tracking,” IEEE Transactions on Geoscience Electronics, vol. 17, no. 4, pp. 250–262, Oct. 1979. |
| [15] | A. Ming and H. Ma, “A blob detector in color images,” in Proceedings of the 6th ACM international conference on Image and video retrieval - CIVR ’07, 2007. |
| [16] | C. Zhang et al., “White Blood Cell Segmentation by Color-Space-Based K-Means Clustering,” Sensors, vol. 14, no. 9, pp. 16128–16147, Sep. 2014. |
| [17] | "CS7960 - Project 1 | James Fishbaugh", *Cs.utah.edu*, 2018. [Online]. Available: http://www.cs.utah.edu/~jfishbau/advimproc/project1/. [Accessed: 04- Dec- 2018] |
| [18] | R. L. Joshi and T. R. Fischer, “Comparison of generalized Gaussian and Laplacian modeling in DCT image coding,” IEEE Signal Processing Letters, vol. 2, no. 5, pp. 81–82, May 1995. |
| [19] | "scikit-image/scikit-image", *GitHub*, 2018. [Online]. Available: https://github.com/scikit-image/scikit-image/blob/master/skimage/feature/blob.py#L278. [Accessed: 04- Dec- 2018] |
| [20] | Everingham, M., Eslami, S.M.A., Van Gool, L. et al. Int J Comput Vis (2015) 111: 98. https://doi.org/10.1007/s11263-014-0733-5 |
| [21] | Y. Yue, T. Finley, F. Radlinski, and T. Joachims, “A support vector method for optimizing average precision,” in Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR ’07, 2007. |
| [22] | P.-E. Forssen, “Maximally Stable Colour Regions for Recognition and Matching,” in 2007 IEEE Conference on Computer Vision and Pattern Recognition, 2007. |
| [23] | Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010) |
| [24] | Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006). |
| [25] | John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55 (publisher link) |
| [26] | "python-pillow/Pillow", *GitHub*, 2018. [Online]. Available: https://github.com/python-pillow/Pillow. [Accessed: 04- Dec- 2018] |
| [27] | Python Software Foundation. Python Language Reference, version 3.7.0. Available at http://www.python.org |
| [28] | H. Kong, H. C. Akakin, and S. E. Sarma, “A Generalized Laplacian of Gaussian Filter for Blob Detection and Its Applications,” IEEE Transactions on Cybernetics, vol. 43, no. 6, pp. 1719–1733, Dec. 2013. |
| [29] | Kaggle.com. (2018). *[ods.ai] topcoders, 1st place solution | Kaggle*. [online] Available at: https://www.kaggle.com/c/data-science-bowl-2018/discussion/54741 [Accessed 13 Dec. 2018]. |
| [30] | "Spatial Filters - Laplacian/Laplacian of Gaussian", *Homepages.inf.ed.ac.uk*, 2018. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm. [Accessed: 13- Dec- 2018] |
| [31] | "Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures - Exsilio Blog", *Exsilio Blog*, 2018. [Online]. Available: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/. [Accessed: 13- Dec- 2018] [32] Onlinecourses.science.psu.edu. (2018). *14.7 - Ward’s Method | STAT 505*. [online] Available at: https://onlinecourses.science.psu.edu/stat505/node/146/ [Accessed 13 Dec. 2018].  [33]Anon, (2018). [online] Available at: https://home.deib.polimi.it/matteucc/Clustering/tutorial\_html/kmeans.html [Accessed 13 Dec. 2018].  [34]Nlp.stanford.edu. (2018). *Single-link and complete-link clustering*. [online] Available at: https://nlp.stanford.edu/IR-book/html/htmledition/single-link-and-complete-link-clustering-1.html [Accessed 13 Dec. 2018].  [35]"Multilayer Perceptron — DeepLearning 0.1 documentation", *Deeplearning.net*, 2018. [Online]. Available: http://deeplearning.net/tutorial/mlp.html. [Accessed: 14- Dec- 2018]. |