CS 585 FINAL PROJECT

Jerry Ying, Steve Choi
04/25/2024

1 Problem Definition

Cell counting is a vital procedure in biology labs, crucial for maintaining cell cultures at optimal densities, standardizing experiments for consistent results, and assessing drug effects in pharmacology. It is particularly important in cancer research for monitoring tumor response to treatments, in stem cell research to track cell differentiation, in virology to calculate viral titers, and in immunology to evaluate immune cell dynamics. Additionally, cell counting ensures quality control in the production of clinical and therapeutic products, underscoring its broad relevance across various scientific and medical fields. This technique provides essential data that is fundamental to robust research outcomes and the development of effective health solutions.

The standard method to count cells is as follows:

1. Place the cells on the hemocytometer.
2. Dye the cells using trypan blue. The cells that are alive will eject the dye, meaning they remain transparent, while the dead cells cannot, meaning they will be dyed blue
3. Look through the microscope and using a counter in one hand, click the counter for every alive cell seen.
4. Multiply the cell count to get the estimate for a larger volume.

We wanted to implement a solution that would automate this process starting from step 3 using computer vision, as this would significantly speed up the cell counting process, especially when researchers have to do this process multiple times a day.

2 Method and Implementation

Our algorithm works based on traditional principles of computer vision rather than a deep learning model and follows a high-level pipeline of a) segmenting the cell candidates and b) counting the number of cells based on the segmentation. Our algorithm begins by converting the BGR image to an HSV image and extracting the value channel. We do this because trypan-blue stained cells (dead) are generally darker than the transparent cells (alive) and cells are generally outlined with dark colors. After applying Gaussian smoothing to reduce the effect of noise, we implement Canny edge detection followed by a simple morphological closing operation to fill in any gaps that the Canny edge detection may have missed. To segment the image, we find and fill the contours of the edge detection, creating a mask that serves as a basic segmentation of our cells. At this point, a problem arises when trying to detect overlapping cells. If we were to try a naive approach of counting the total number of contours, overlapping cells would simply be counted as a single cell, and noisy segmentation could produce a lot of false positives. This is where the distance transform comes in handy:

$$dist(pixel) = \textbf{minimum distance from nearest black pixel} \tag{1}$$

By applying this transformation onto the binary segmentation mask, it effectively creates a topological map where the peaks correspond to the centers of a circular shape. This can be reasoned by considering the geometry of a circle – the circle center will have the highest distance transform that also corresponds with its radius. See figures below:

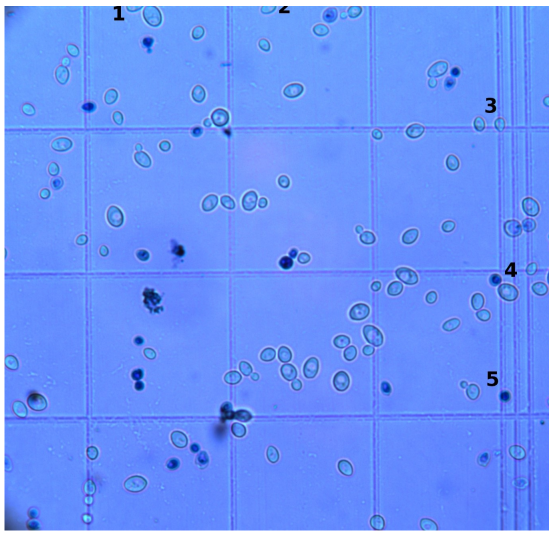

Figure 1: Test image

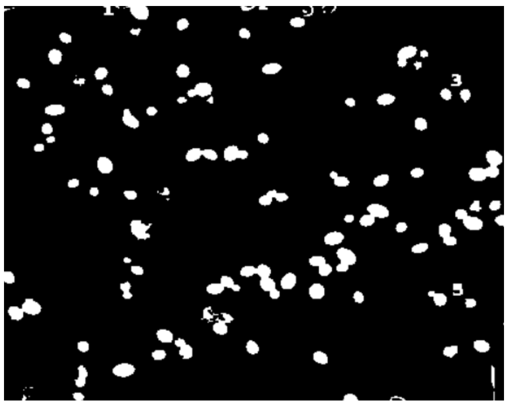

Figure 2: Filled in Contours

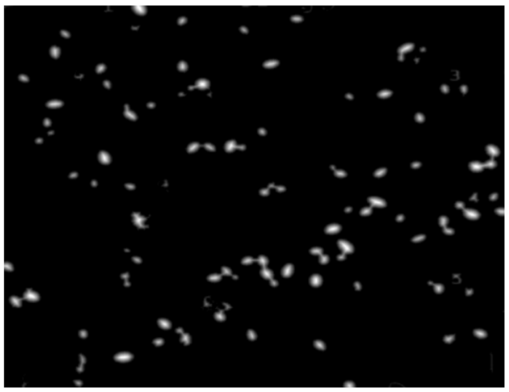

Figure 3: Distance Transform (scaled for visibility purposes)

Once we have the distance transform, we need to find the peaks of the distance transform. For this, we used a function from scikit-image that works by applying a dilation operation to merge local maxima in close proximity with each other. It then compares this dilated image with the original image, taking the peaks as the spots where they are equal. Although we tried to implement this method ourselves with normalization methods and dilation operations, we were plagued with edge case errors and a underwhelming peak detection rate. Thus, we turned to scikit-image's optimized local maxima detection. Once we have our local maxima, we iterated through each one and filtered out the ones that had low corresponding distance transform values. Doing this should ideally filter any noisy or non-circular values. After this, we applied a 15x15 kernel over each detected center in the original image and used Otsu's thresholding to differentiate dead cells from alive cells, drawing green circles to represent any live cells that we detected. See the figures below:

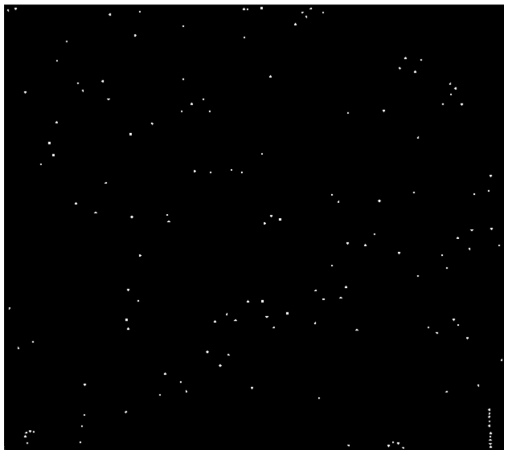

Figure 4: Detected Peaks

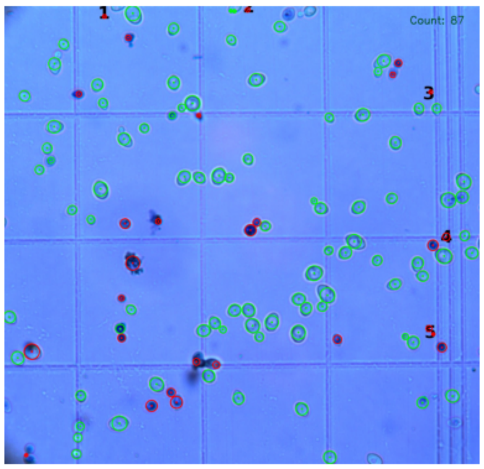

Figure 5: Live cells with green circles drawn on)

3 Experiments

We've mainly experimented with various OpenCV tools. Some of the methods we've attempted are listed below.

1. Template Matching + Sobel Gradients
   Our method of template matching involved analyzing the binary image result we've gotten from thresholding then using Sobel gradient magnitude to find the local minima. Although our template-matching result outputs were as expected, we were unsuccessful with the local minima identification.
2. Deep Learning/FCN
   One method we considered was deep learning, or FCN from A4. However, deep learning requires a lot of training, and there weren't that many training data available online, hence why we were unable to use deep learning.
3. Hough's Circles
   We've also tried Hough's circles method for the circle detection for the cells. However, we were unsuccessful with this method, as the Hough's circles method requires the circles to be a perfect circle, and as it can be seen in the image, the cells are quite far from a perfect circle.

Define your evaluation metrics e.g., detection rates, accuracy, running time.

4 Results

List your experimental results. Provide examples of input and output images. If relevant, you may provide images showing any intermediate steps.

The confusion matrix for figure 1 ended up looking like this:

As we can see, our algorithm is very effective, shown by the very high accuracy of our confusion matrix values. It was able to identify the cells with about 97.7% accuracy, and ignore or mark the dead cells with a very high accuracy of 92.8%. Adjusting a few parameters and running on a different image of trypan-blue dyed insect cells gives us the following image:

5 Discussion

As we can see from the results section, our method was very accurate at detecting the alive cells. With a little adjustment, our method can work for other images. There are also benefits such that if a researcher has to repeatedly count cells, this process can substantially speed up the process of counting cells.

	Positive	Negative	Precision
Positive	85	2	87
Negative	2	26	28
Recall	87	28	115

Figure 6: Confusion Matrix generated with Google Spreadsheets

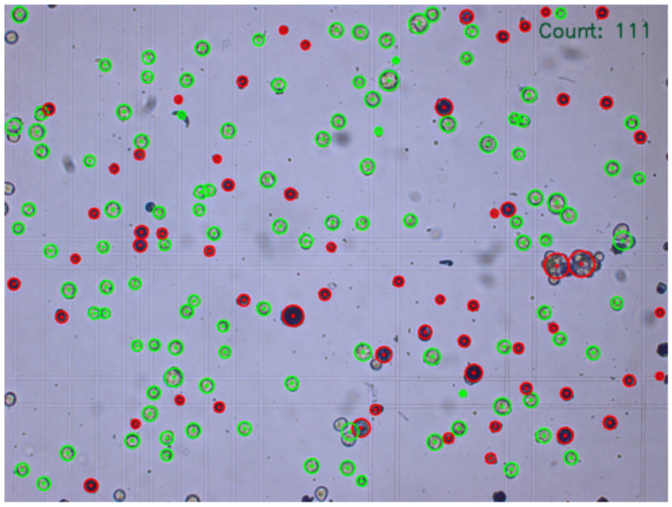

Figure 7: Although still fairly accurate, our algorithm struggles with detecting large clusters of cells

While our method was pretty accurate at detecting the alive cells for the testing data we used, it definitely does not work for every image. This is because different images have different thresholding values, giving a binary image that cannot be analyzed using the same method we used here. A possible solution to make a more universal method is the use of deep learning (or FCN from A4) and train the model based on the features of the cell. However, our team lacked the knowledge of deep learning and there weren't many testing data on the internet, as stated above in the list of methods attempted.

6 Conclusions

Based on our findings and results, we were satisfied with our results, given the time constraints we had and our lack of deep learning knowledge. However, if we had more time, we would've attempted to get more training data from PhD biology labs and learn the deep learning knowledge required to make this model work.

7 Credits and Bibliography

Home. OpenCV. (2024, February 5). https://opencv.org/

"Yeastcount.jpg." (Photograph). https://eurekabrewing.wordpress.com/wp-content/uploads/2012/07/yeastcount.jpg/