COMP 4360

IMAGE PROCESSING

COMPUTER ENGINEERING DEPARTMENT

ASSIGNMENT #1

16.11.2025

MUSTAFA GÖKSU ERDOĞAN

22070001052

1. Methodology

Before doing any kind of preprocessing or morphology, I loaded the image as grayscale and made it negative. This is because, I wanted the cells to be in the foreground for further methodologies.

After that a Gaussian Blur is applied to the image. This preprocessing step is crutial for reduction of noise in the image and making sure that centers of the cells are similar color as the outside of the cells. This is important, after thresholding, empty spaces in the middle of the cells are minimized because the Gausian Blur spread the colors accros the cells fort hem to be closer to gray, rather than black.

My goal in choosing the parameters for cv2.GaussianBlur() and cv2.threshold() was making sure that holes in the cells were filled close to being full . I achived this by using ksize=(21,21) for cv2.GaussianBlur() and thresh=60 for cv2.threshold() by trial and error.

Its possible to fill the significant portion of the hole in the cells by using different values for ksize and thresh. But for the next step, I wanted them to be filled just enough.

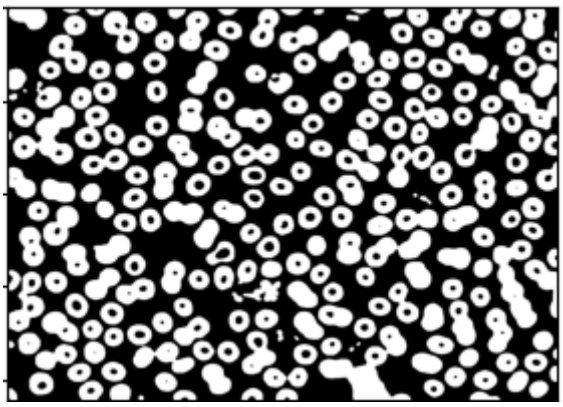

Image 1: Binary image after Blur.

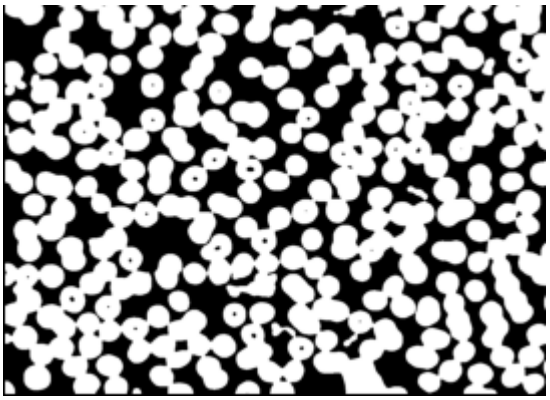

Image 2: Nearly filling all the cells using ksize=(27,27) and thresh=40

Next step was filling in the holes of the cells. This was achived by using floodFill() algorithm. It floods all connected background pixels. This flooded image was then inverted to create a mask of only the internal holes. Finally, this hole mask was combined with the binary image using cv2.bitwise_or. In the previous step, it was on purpose to leave unfilled spaces. By using floodFill() algorithm, holes within the cells are filled more accurately.

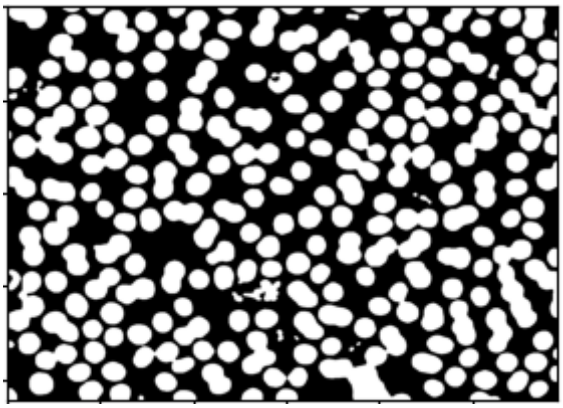

Image 3: Image after floodFill algorithm

Two morphological operartions were used in my methodology. These being Opening with an elliptical kernel of (10,10) for 3 iterations and erosion with an elliptical kernel of (5,5) for 4 iterations.

Opening was spesificly important to get rid of dusts, particles and unwanted connections between the cells. Kernel size and iteration count were decided by trial and error.

Erosion was necessary to get rid of unwanted connections between the cells even further. Kernel size and iteration count were choosen like this to evoid losing cells in the image by trial and error

By using an elliptic cernel for both methods, circular shape of the cells were preserved.

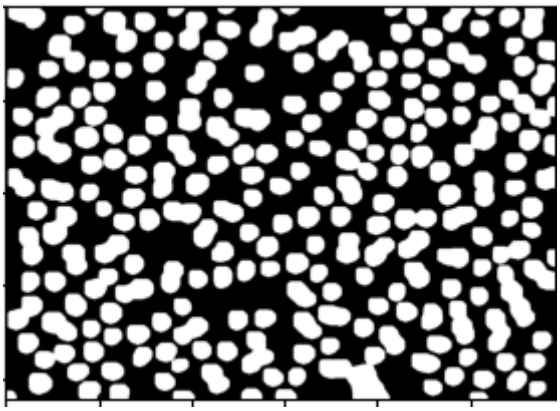

Image 4: Image after Opening

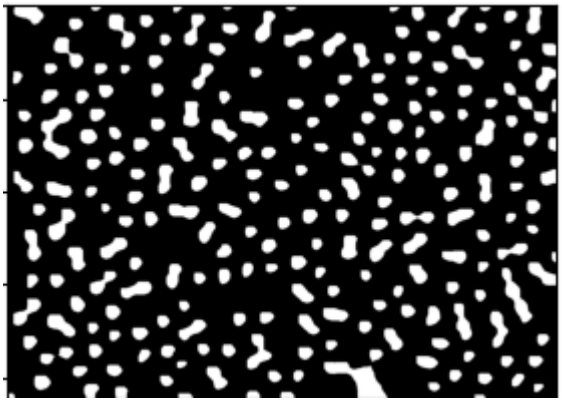

Image 5: Image after Erosion

Finally, for counting the cells in the image cv2.findContours was used with RENT_EXTERNAL to retrive only the outermost contours CHAIN_APPROX_SIMPLE to save memory.

The final object count was determined by the length of the returned contours list. The contours were then drawn onto a new image using cv2.drawContours to provide a clear visual representation of the counted objects. Final count was 236 cells.

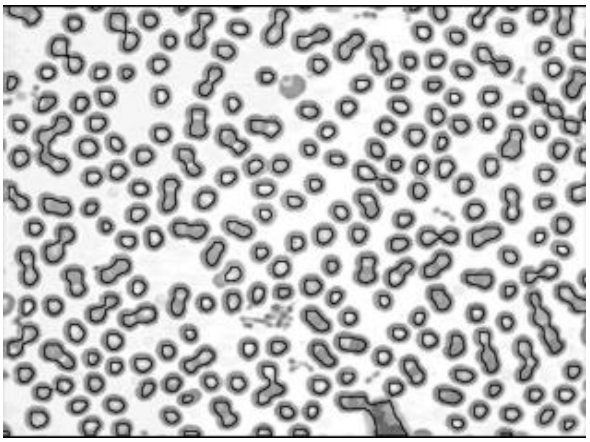

Image 1: Contours on Original Image

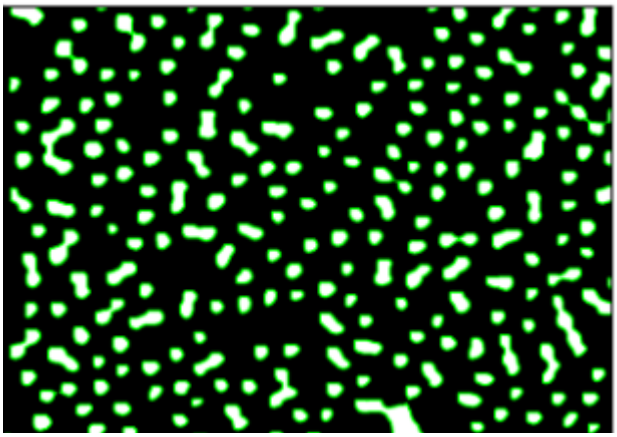

Image 2: Final Image and Contours

2.Approach

I previously tried HoughCircles() algorithm to find cells in the given image. This method was more successful but images used were supposed to be grayscale, binary images would resuld in an error if used.

So I decided to use cv2.findContours method to find cells in the image. For this method to be successful, I had to make sure that cells connected to eachother were seperated properly. Otherwise this method would result in an unreliable outcome.

I also made sure that centers of cells were filled properly since cv2.findContours would count the empty spaces within the cells as a cell aswell.

To solve these issues, I used cv2.floodFill() algorithm to fill in the cells. Previously I filled the empty cells by trial and error using cv2.GaussianBlur() and cv2.threshold() with different parameters before using cv2.floodFill() algorithm. This resulted in either a whitespace or a blackspace. To solve this, I intentially left minimal empty spaces in the cells so the algorithm would result in a better output.

To clear up the image, Opening and Erosion methods were used with elliptic kernels to keep the cells' circular shape.

3.Results

Final result was 236 cells in the image. This number is relatively accurate but it can be improved by seperating the cells which are ontop of each other.

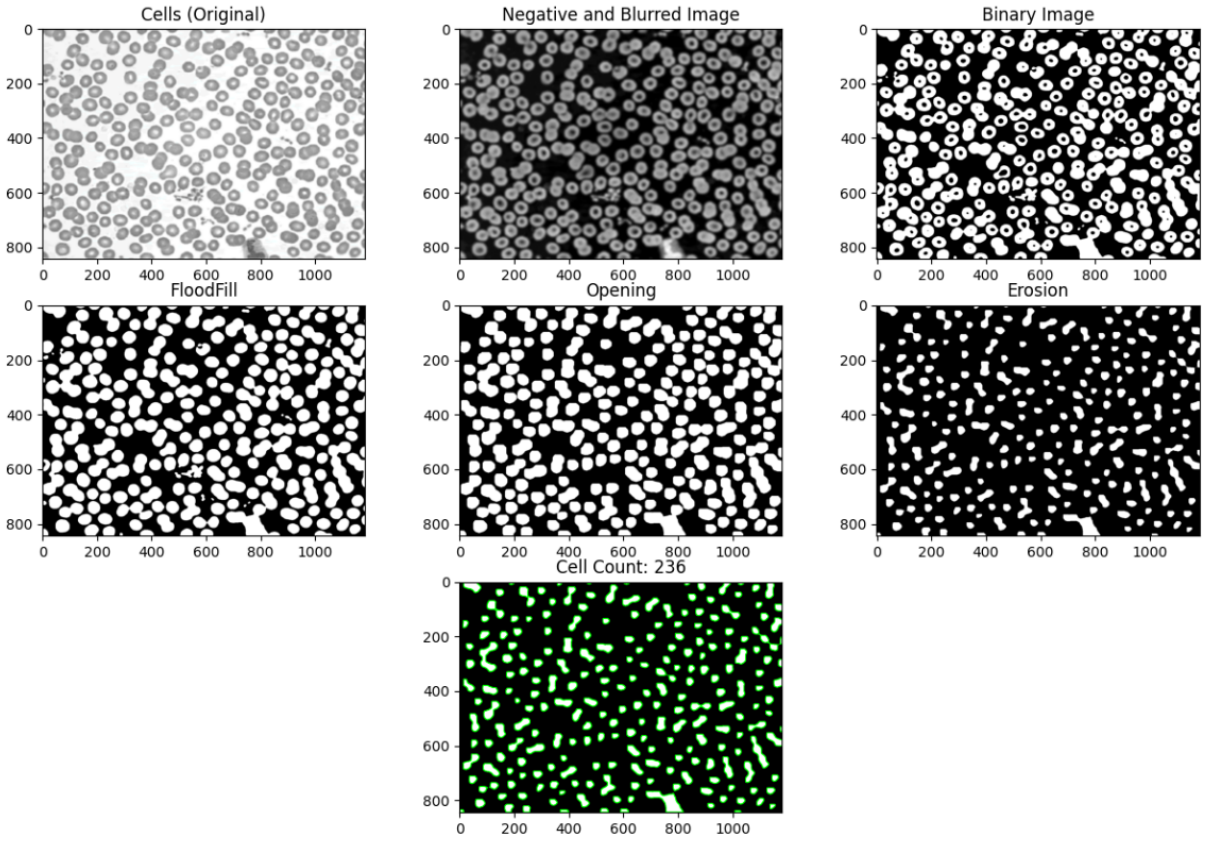

Image 3 Final Result

4.Discussion

The pipeline identifies and counts a large number of cells by separating them. The main limitation is that this aggressive separation might destroy very small cells, leading to an under count.

Very large, dense clumps of cells may not be fully separated, leading to an over count (if holes are formed) or under count (if the clump remains one object). It is visible on the image that some closelly grouped cells could not be seperated so this method counted them as one cell, making it undercounted.

The accuracy of the final count is sensitive to the parameters chosen. Different parameter usage will result in varying counts of cells.

5.Code

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

original = cv2.imread('cells.png')
cells = cv2.imread('cells.png', 0)
cells = cv2.bitwise_not(cells)

cellsBlurred = cv2.GaussianBlur(cells, (21, 21), 0)
_, cellsBinary = cv2.threshold(cellsBlurred, 60, 255, cv2.THRESH_BINARY)


floodfill = cellsBinary.copy()
h, w = cellsBinary.shape[:2]
mask = np.zeros((h + 2, w + 2), np.uint8)
cv2.floodFill(floodfill, mask, (0, 0), 255)
holes = cv2.bitwise_not(floodfill)
filled_cells = cv2.bitwise_or(cellsBinary, holes)


kernel_opening = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
cellsOpened = cv2.morphologyEx(filled_cells,cv2.MORPH_OPEN,kernel_opening,iterations=3)


kernel_erosion = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
cellsFinal = cv2.erode(cellsOpened, kernel_erosion, iterations=4)


contours, _, = cv2.findContours(cellsFinal.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
cellCount = len(contours)
cellsContours = cv2.cvtColor(cellsFinal, cv2.COLOR_GRAY2BGR)
cv2.drawContours(cellsContours, contours, -1, (0, 255, 0), 2)

plt.figure(figsize=(15, 10))

plt.subplot(3, 3, 1);plt.imshow(original, cmap="gray");plt.title("Cells (Original)")
plt.subplot(3, 3, 2);plt.imshow(cellsBlurred, cmap="gray");plt.title("Negative and Blurred Image")
plt.subplot(3, 3, 3);plt.imshow(cellsBinary, cmap="gray");plt.title("Binary Image")
plt.subplot(3, 3, 4);plt.imshow(filled_cells, cmap="gray");plt.title("FloodFill")
plt.subplot(3, 3, 5);plt.imshow(cellsOpened, cmap="gray");plt.title("Opening")
plt.subplot(3, 3, 6);plt.imshow(cellsFinal, cmap="gray");plt.title("Erosion")
plt.subplot(3, 3, 8);plt.imshow(cellsContours, cmap="gray");plt.title(f'Cell Count: {cellCount}')

plt.show()
```