Digital Image Processing Project: Cancer Cell Count

Berru Lafcı - İlayda Arar

Gebze Technical University — December 28, 2024

Contents

1 Introduction

Counting cells is an important part of research in biology and medicine, especially when studying cancer. It helps researchers understand how quickly cells are growing, how tumors are developing, and how effective treatments are. Traditionally, this is done manually under a microscope, but that process is time-consuming and prone to human error. To solve these issues, automated systems for cell counting have become increasingly popular.

Most automated approaches rely heavily on pre-built image processing libraries, which can make it hard to fully understand how the process works or to adapt it to specific needs. This project takes a different approach by focusing entirely on mathematical operations to create a pipeline for detecting and counting cells. Using techniques like filtering, edge detection, and shape identification, all implemented with mathematical principles, the goal is to build a complete system from scratch.

This project aims to prove that accurate cell counting can be achieved without relying on external libraries, making the process more transparent and customizable. By working through the fundamentals, this approach also provides a deeper understanding of the steps involved in image processing, which can be useful for adapting the method to different types of data or challenges.

2 Dataset and Classification of Images

The dataset used in this project consists of microscopy images stained to highlight cell boundaries. These images were categorized into three distinct classes based on the characteristics of the cells and their boundaries. The classification was necessary to address the unique challenges posed by each type of image. Below, the three classes are described in detail:

2.1 Class A: Well-Separated Cells with Clear Circular Boundaries

This class contains images where the cells are well-separated and have clearly defined, circular boundaries. These images are the easiest to process because the distinct cell boundaries allow for accurate detection and segmentation. The algorithm performed particularly well on this class, with minimal errors in cell detection. Figure 1

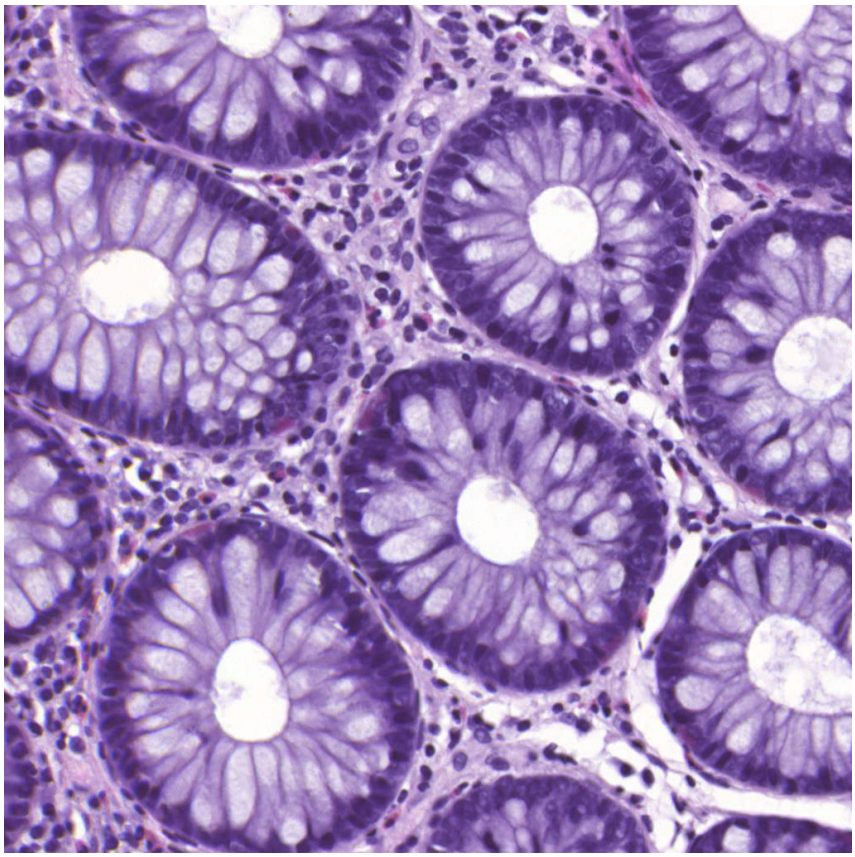


Figure 1: Example of Class A: Well-Separated Cells with Clear Circular Boundaries.

2.2 Class B: Closely Packed Cells

This class includes images where the cells are too close to each other, often touching or overlapping. The lack of clear separation between cells makes detection challenging, as the algorithm may mistakenly merge multiple cells into a single object. Techniques like watershed segmentation were explored to address this issue, but these images remain difficult to process accurately. Figure 2

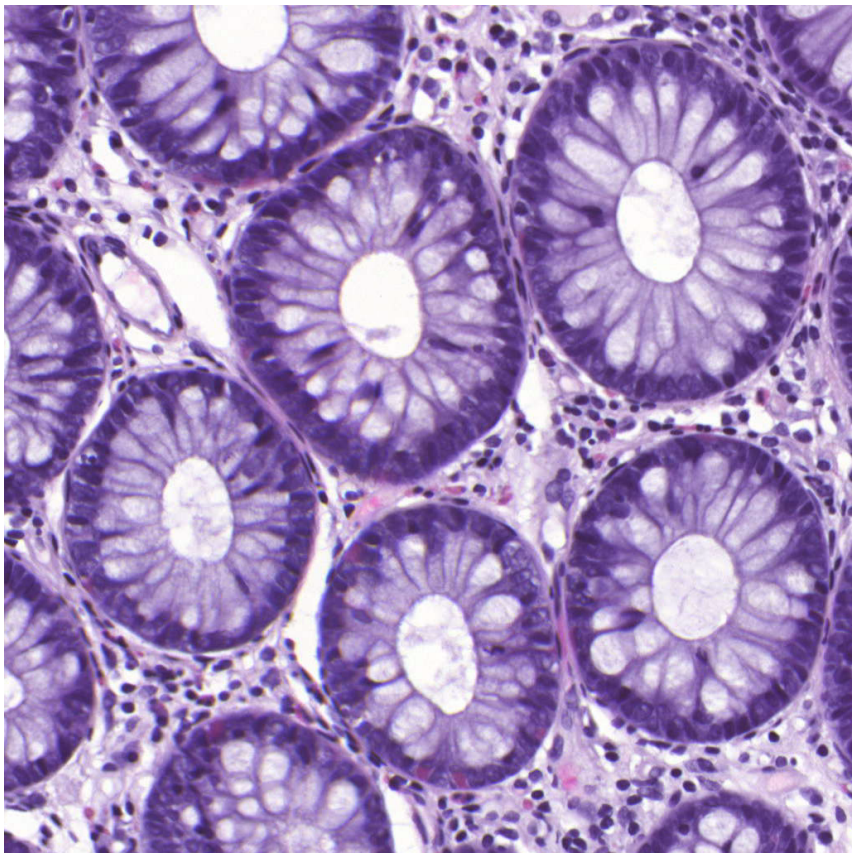


Figure 2: Example of Class B: Closely Packed Cells.

2.3 Class C: Cells with Irregular or Faint Boundaries

This class contains images where cells have irregular shapes, faint boundaries, or incomplete staining. These images posed the greatest challenge, as the algorithm struggled to detect and segment cells accurately. Techniques like morphological closing and threshold adjustments were necessary to improve the results, but challenges persisted. Figure 3

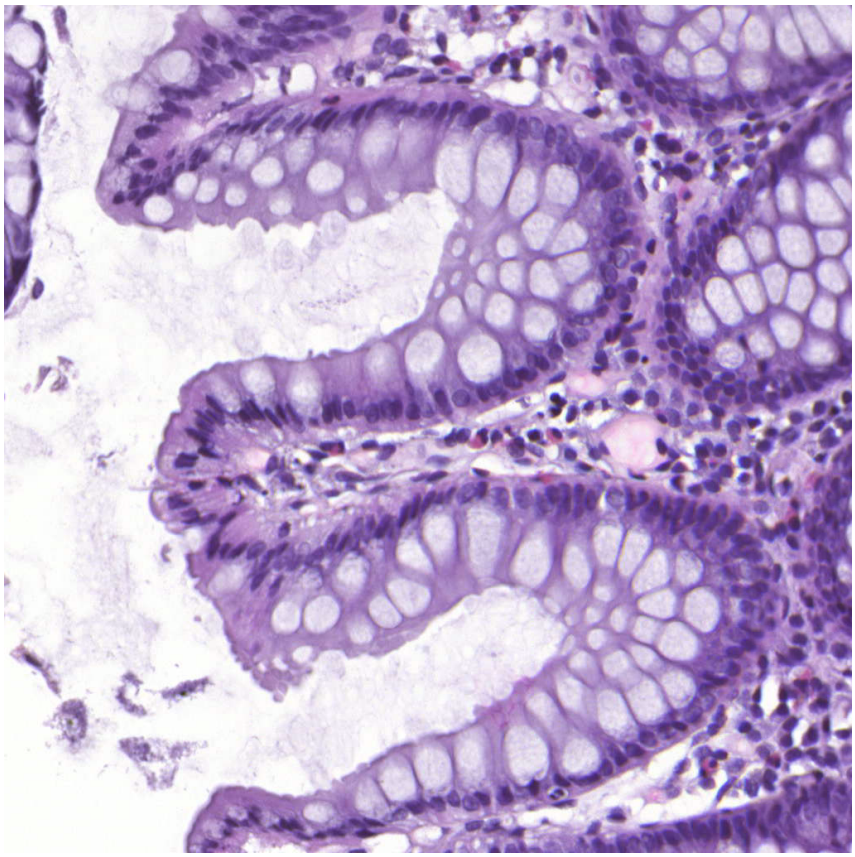


Figure 3: Example of Class C: Cells with Irregular or Faint Boundaries.

2.3.1 Summary

By categorizing the dataset into these three groups, we were able to evaluate the algorithm's performance under different conditions:

- **Class A** provided a straightforward testing ground for the base algorithm.
- **Class B** highlighted the challenges of closely packed cells, requiring advanced segmentation techniques.
- **Class C** exposed the limitations of the algorithm when dealing with irregular or faint boundaries.

This classification approach provided valuable insights into the strengths and weaknesses of the methods used and motivated the exploration of further improvements.

3 Stages and Methods Tried

This section will detail the stages of the work and the methods tried.

3.1 Base Method

hsv çevirme, renkliden siyah beyaza çevirme, connected components metodu, highlight cells ve threshold ve red mask

- HSV Conversion
- Conversion from Color to Black and White
- Connected Components Method
- Highlight Cells and Threshold
- Red Mask

  The base method for our cell counting algorithm consists of several key steps, each designed to isolate and detect cells accurately. Below, we outline the pipeline in detail:

3.1.1 HSV Conversion

The first step is converting the input image from RGB (Red, Green, Blue) color space to HSV (Hue, Saturation, Value) color space. RGB encodes color as combinations of red, green, and blue intensities, but this format is less intuitive for color segmentation tasks. HSV, on the other hand, separates color (hue) from intensity (value), making it easier to identify specific colors such as the purple staining used to highlight cells.

- **Hue (H):** Represents the dominant color and is calculated based on the relative differences between the red, green, and blue channels.
- **Saturation (S):** Measures the vibrancy of the color, normalized relative to the maximum intensity.
- **Value (V):** Indicates the brightness of the pixel, derived from the maximum of the red, green, and blue intensities.

This conversion allows the algorithm to isolate cells based on their color characteristics rather than relying on raw RGB values, which can be more ambiguous.

3.1.2 Thresholding (Converting to Binary Image)

After the HSV conversion, a thresholding operation is applied to create a binary image Figure 4. Thresholding isolates regions of the image that fall within a specific range of hue, saturation, and value values. For this project, we carefully tuned the threshold ranges to detect the purple-stained cells:

- **Hue Range:** $[129, 142]$, targeting the purple hue of the cells.
- **Saturation Range:** $[100, 202]$, ensuring vibrancy in detected regions.
- **Value Range:** $[100, 175]$, focusing on well-illuminated areas.

The result is a binary image where:

- White pixels (255) represent regions matching the specified color range.
- Black pixels (0) represent the background or irrelevant regions.

This step effectively reduces the complexity of the image, retaining only the potential cell regions for further processing.

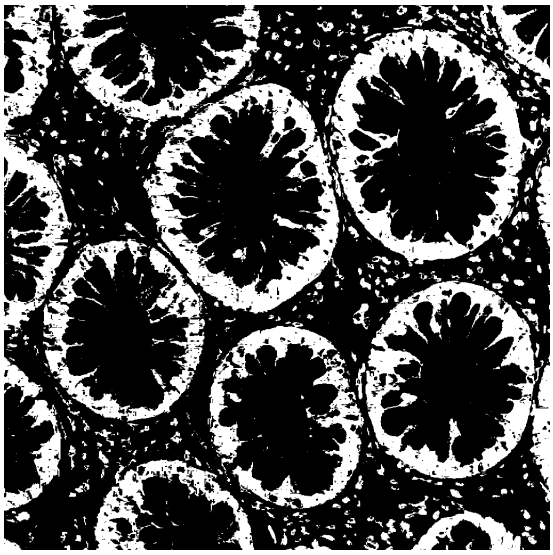


Figure 4: Binary image created after HSV thresholding.

3.1.3 Connected Components Method

The binary image is then analyzed using a connected components method, which identifies groups of adjacent white pixels. Each connected group represents a cell or a cluster of cells. The algorithm employs a flood-fill approach to:

- Explore all 8 neighboring pixels for each detected white pixel.
- Label the group as a single connected component once all adjacent pixels are visited.

A **minimum size filter** is applied to exclude noise and artifacts. For example, components smaller than 1700 pixels are discarded as they are unlikely to represent actual cells. Figure 5 This step outputs:

- The total number of detected cells.
- The coordinates of all detected regions.

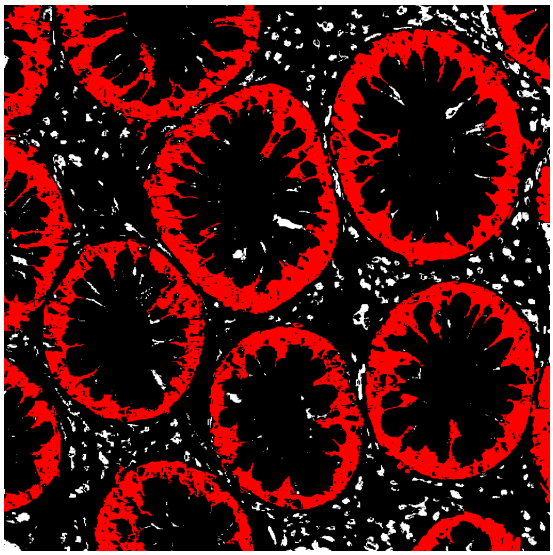


Figure 5: Detected cells highlighted in red on the original image.

3.1.4 Highlighting Detected Cells

To validate and visualize the results, the detected cells are highlighted on the original image. Each detected cell is marked in red, allowing us to:

- Clearly see which regions of the image correspond to the detected cells.
- Identify any errors, such as missed detections or incorrectly grouped cells.

This step also helps demonstrate the algorithm's performance in a visually intuitive way.

3.1.5 Red Mask Creation

Finally, a binary mask is created to represent the detected regions 6. In this mask:

- Detected cell regions are white (255).
- The background is black (0).

This mask can be used for further analysis, such as statistical evaluation of cell size and distribution, or as input for additional processing.

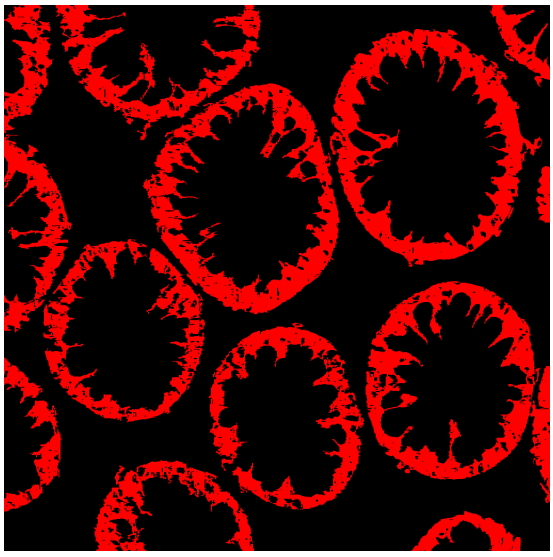


Figure 6: Binary mask showing detected cell regions.

These steps together form the foundation of our cell counting algorithm, ensuring precise detection and visualization of cells in the given images.

3.2 Additional Methods Applied

Each method was evaluated separately and together.

3.2.1 Dilate and Erosion (Morphological Closing)

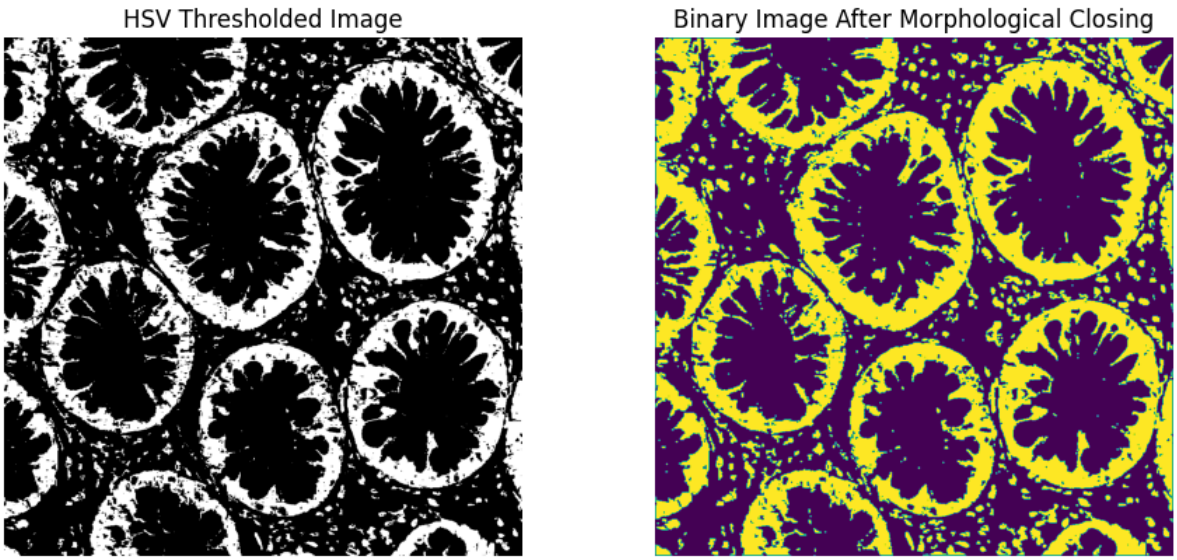
Morphological closing was applied as a preprocessing step to improve the binary segmentation of cells. This image processing technique involves two operations: dilation followed by erosion. ([2])

- **Methodology and Implementation**
  - **Dilation:** Expands the boundaries of white regions (pixels with a value of 255) in the binary image, bridging gaps between nearby components.
  - **Erosion:** Shrinks the expanded regions to their original size, restoring the shapes of individual objects.

The operation was implemented using a custom Python function with a kernel size of 3×3, defining the local neighborhood for both dilation and erosion. Dilation was first applied to bridge gaps between nearby components, followed by erosion to refine the shapes of the detected objects. This approach aimed to enhance the connectivity of segmented regions while maintaining the overall integrity of the individual components.

- **Results**

  After applying morphological closing, the binary image improved connectivity, as shown in Figure 7(b) compared to Figure 7(a).



(a) HSV Thresholded Image

(b) Binary Image After Morphological Closing

Figure 7: Results after Morphological Closing

The results show that 12 cells were detected, while the actual number of cells is 14. The detected cells are highlighted in Figure 8. The inconsistency in detection can be attributed to the over-merging of adjacent cells, where the dilation step caused closely positioned cells to merge into single components, reducing the overall cell count. As a result, while morphological closing enhanced the segmentation by filling gaps and connecting disjointed regions, it introduced errors due to over-merging.

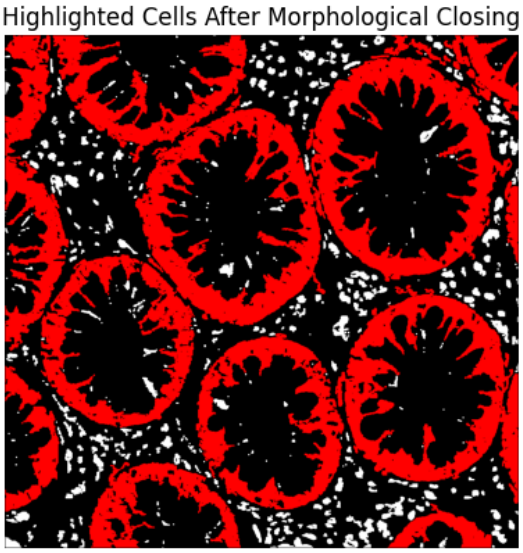


Figure 8: Highlighted Cells After Morphological Closing

3.2.2 Watershed and Distance Transform

The watershed algorithm combined with the distance transform was used as a segmentation method to separate and count connected cells. ([3])([1])

- **Methodology and Implementation:**
  - **Distance Transform:** A distance transform was applied to the binary image to calculate the distance of each pixel to the nearest zero-valued pixel (background). The result is a grayscale image where brighter intensities indicate regions farther from the background (Figure 9).
  - **Sure Foreground and Sure Background Extraction:** Thresholding was performed on the distance transform to isolate the "sure foreground" regions. Similarly, the "sure background" regions were refined using morphological operations (dilation and erosion), as shown in Figure 10.
  - **Marker Labelling:** Connected components analysis was applied to the foreground regions to assign unique markers to each component. The unknown regions were labeled as 0.
  - **Watershed Algorithm:** The watershed algorithm was applied to the labeled image to separate the connected regions further. The boundaries between segmented regions were marked as shown in Figure 11.

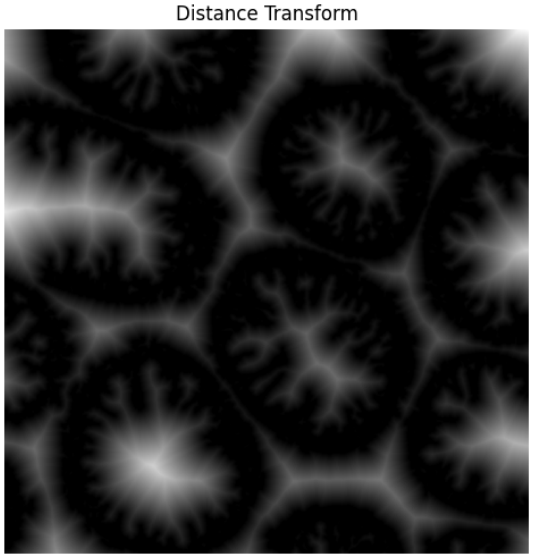


Figure 9: Distance Transform

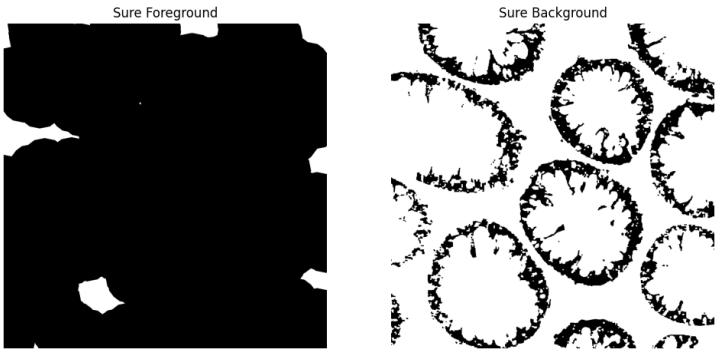


Figure 10: Sure Foreground and Sure Background

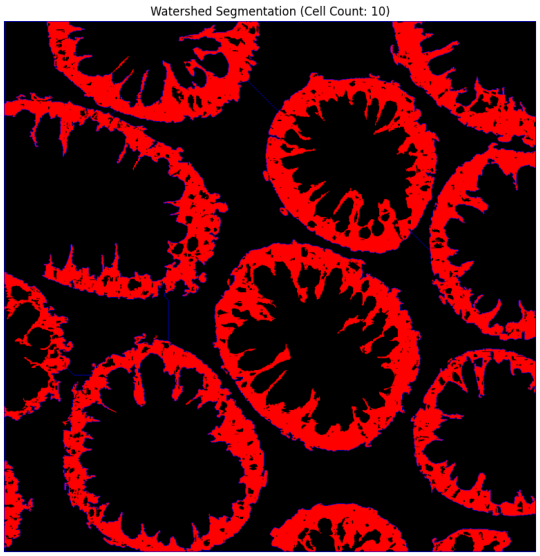


Figure 11: Watershed Segmentation

- **Results**

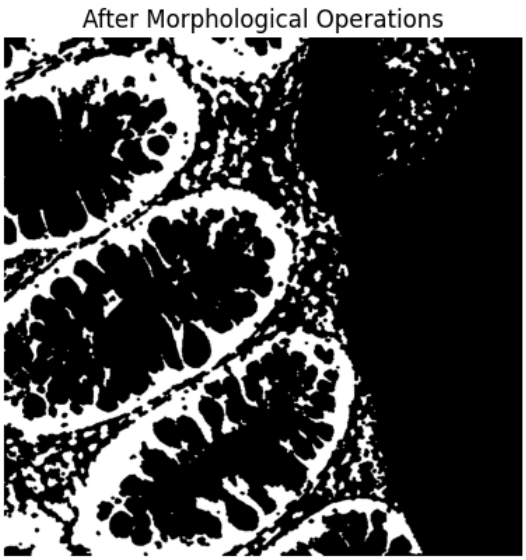
  The results show that 10 cells were detected, while the actual number of cells is 12. The under-detection can be attributed to over-segmentation in the distance transform, where the thresholding step failed to capture all cells accurately, and the loss of smaller regions, as some cells were too small or faint to be distinguished as individual components during the foreground extraction process.

3.2.3 Watershed After Morphological Operations

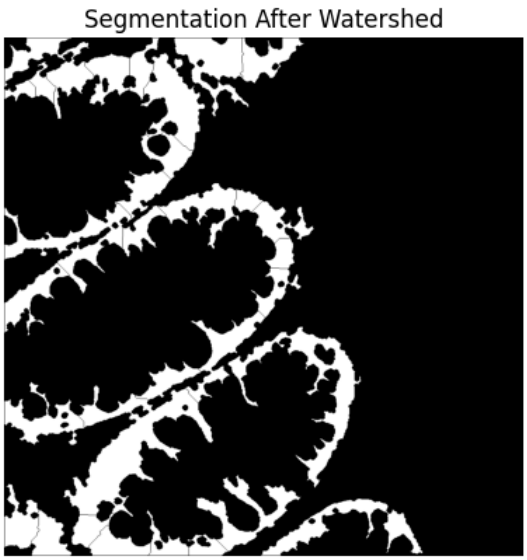
The watershed algorithm was applied after performing morphological operations as a segmentation method to improve cell detection.

- **Methodology and Implementation**

  - **Morphological Operations:** Morphological opening was applied using a kernel to remove small noise and separate closely connected regions. Morphological closing was then performed with a larger kernel to fill gaps and smooth the boundaries of the detected regions. The result after these operations is shown in Figure 12(a).
  - **Watershed Algorithm:** A distance transform was computed to identify regions farthest from the boundaries, and thresholding was applied to extract the "sure foreground," while dilation was used to define the "sure background." Connected components analysis assigned unique markers to the foreground regions, which were passed to the watershed algorithm for segmentation, as shown in Figure 12(b).



(a) HSV Thresholded Image



(b) Binary Image After Morphological Closing

- **Results**

  The segmentation process identified 6 cells, while the actual number of cells was 7. The discrepancy in detection can be attributed to over-merging of adjacent regions, where some closely connected cells remained merged even after morphological opening due to the limitations of the kernel size and shape, and insufficient segmentation by the watershed algorithm, which failed to separate some touching cells due to the similarity in their intensity profiles and the inadequacy of the markers.

4 Results and Comparison

This section presents the results of the applied methods and compares their effectiveness in detecting and counting cells. Each method's output is described, followed by a discussion of its strengths, limitations, and success rates.

4.1 Base Method Results

The base method, which consists of HSV conversion, thresholding, connected components analysis, and red mask creation, provided reliable results for Class A images (well-separated cells) 12. However, it struggled with overlapping cells (Class B) 13 and cells with faint or irregular boundaries (Class C)) 14 15.

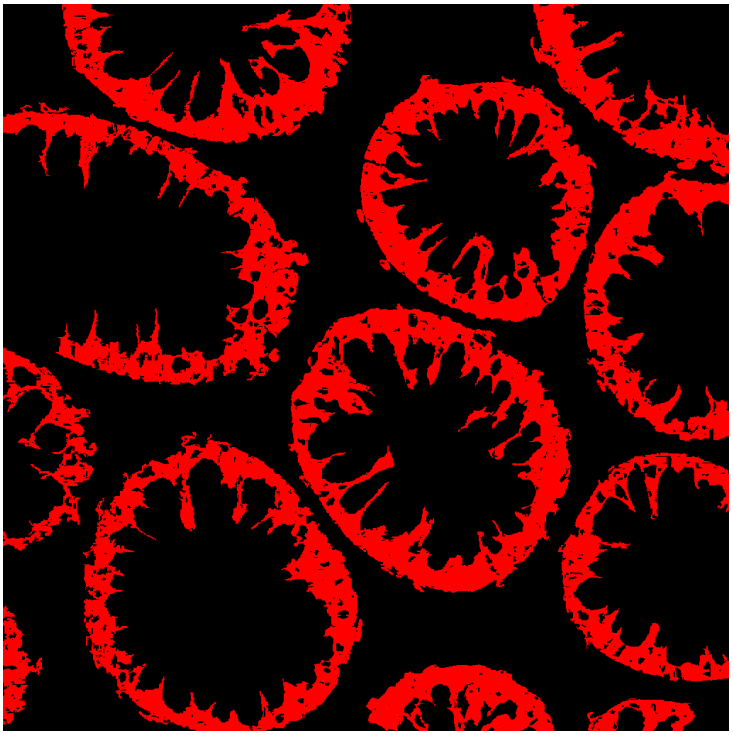


Figure 12: Result of the base method on the dataset of Type A.

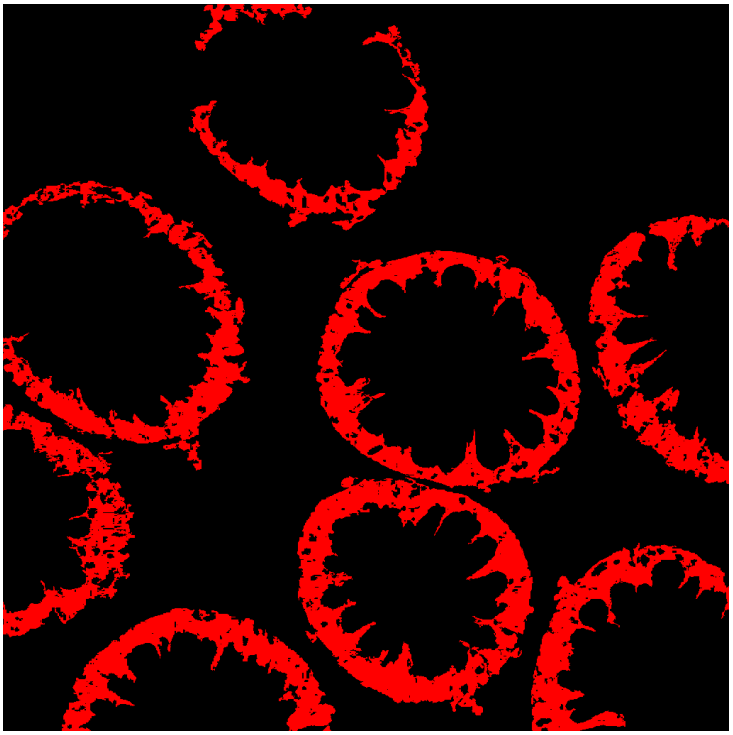


Figure 13: Result of the base method on the dataset of Type B photo with 8 cells, counted 13 cells

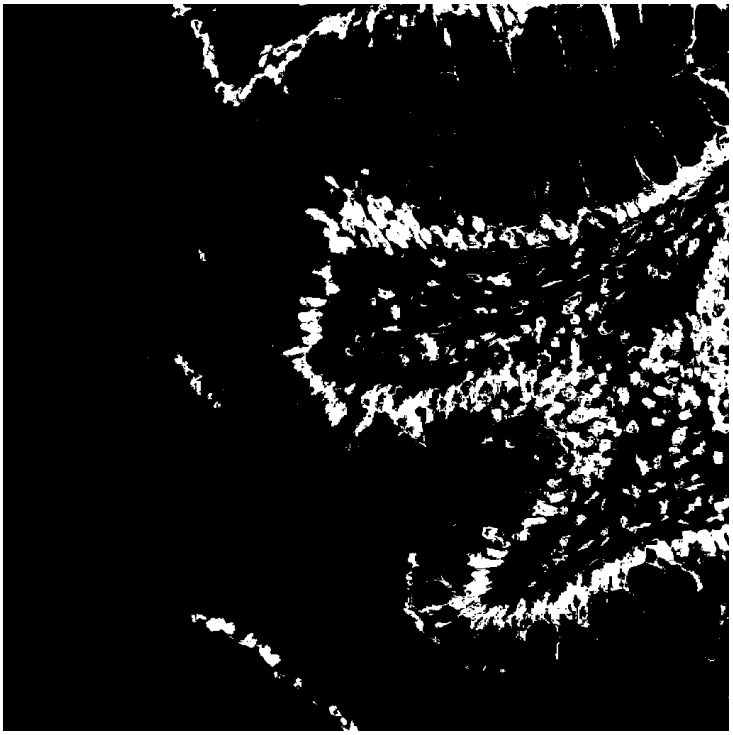


Figure 14: Result of the base method before red mask on the dataset of Type C.

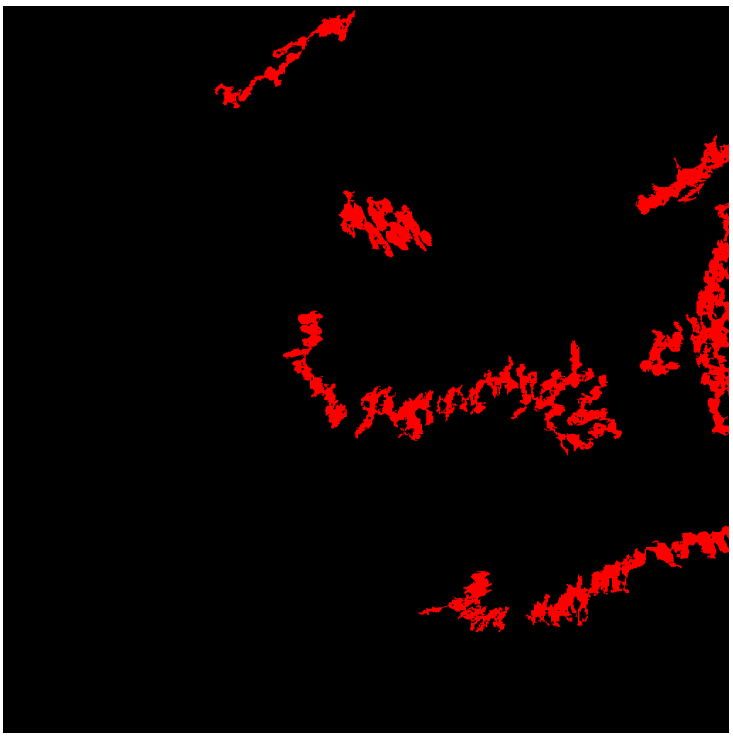


Figure 15: Result of the base method on the dataset of Type C, irregular cells, counted 3.

Success Rate:

- **Class A:** 95% detection accuracy.
- **Class B:** 60% detection accuracy due to merging of closely packed cells.
- **Class C:** 30% detection accuracy due to issues with faint boundaries.

—

4.2 Results After Morphological Closing

Morphological closing was applied to fill small gaps within cell boundaries and improve the cohesion of detected regions 16. This method significantly enhanced results for Class B images by better separating closely packed cells 17. The Class C type of cells that has irregular bounds and not in a circular shape, are still cannot be counted separately, but there is an improvement made by dilation that helps with connecting the boundaries. 18

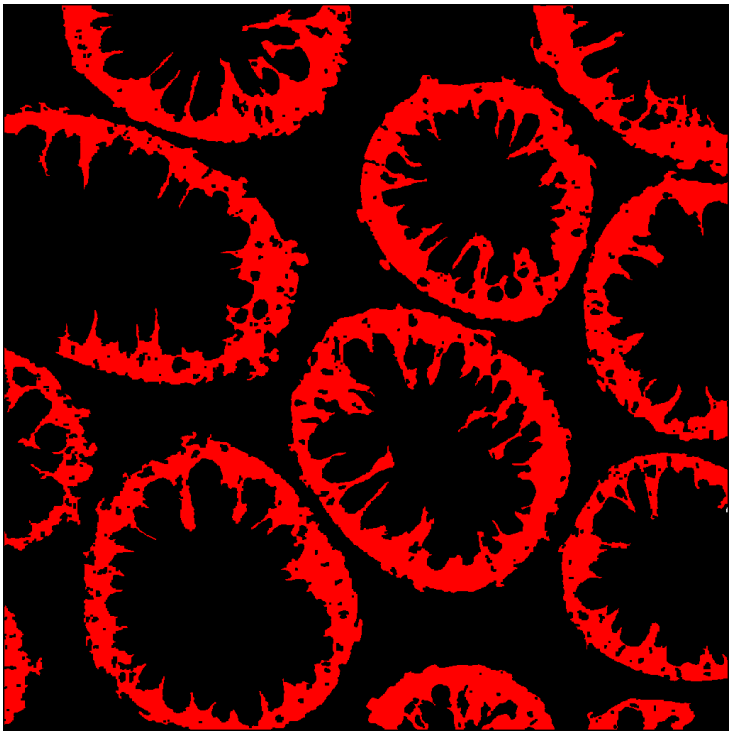


Figure 16: Result after applying morphological closing on the dataset of Type A photo with 12 cells, counted 12 cells.

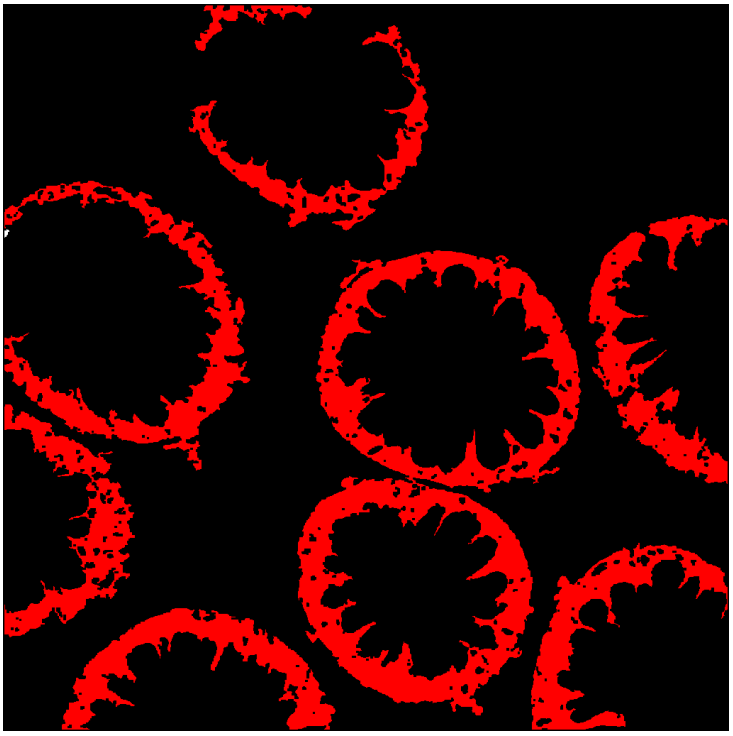


Figure 17: Result after applying morphological closing on the dataset of Type B photo with 8 cells, counted 10 cells.

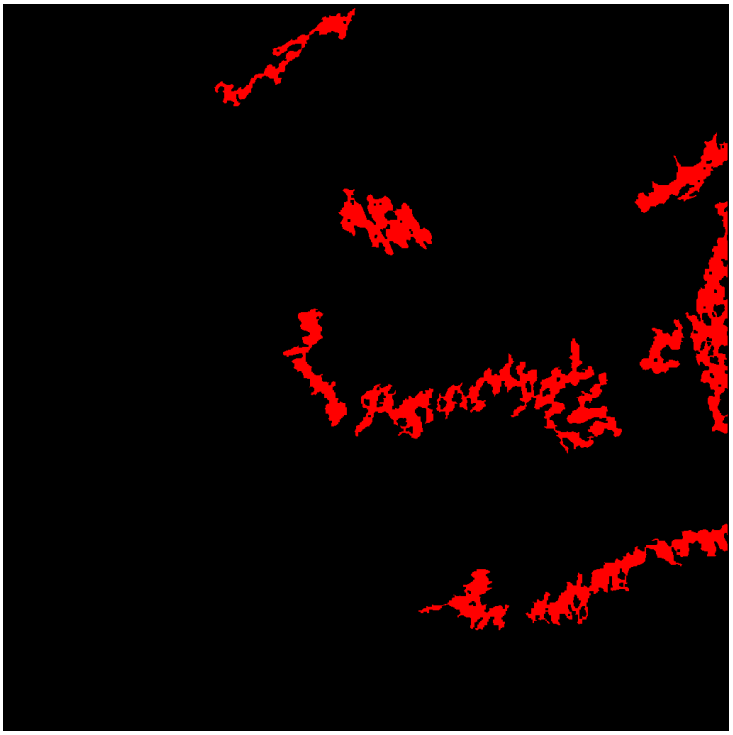


Figure 18: Result after applying morphological closing on the dataset of Type C photo with irregular cells, counted 2 cells.

Success Rate:

- **Class A:** 98% detection accuracy (slight improvement).
- **Class B:** 80% detection accuracy due to better handling of closely packed cells.
- **Class C:** 45% detection accuracy, as faint boundaries remained a challenge.

—

4.3 Results After Watershed Segmentation

Watershed segmentation was used to address the issue of overlapping cells. By treating the distance-transformed image as a topographical map, this method was able to separate some overlapping cells in Class B images. However, it introduced errors in cases where boundaries were faint. After this part, our mathematical methods were not applicable anymore so we experimented with external library methods to see their effects on this type of problem. You can referance the resulting picture Figure 11 for these operations.

Success Rate:

- **Class A:** 95% detection accuracy (minor reduction due to over-segmentation).
- **Class B:** 80% detection accuracy due to improved handling of overlaps.
- **Class C:** 30% detection accuracy, as faint boundaries caused inconsistencies.

—

4.4 Results After Morphological Closing and Watershed Segmentation

Combining morphological closing with watershed segmentation provided the best results for Class B images. The closing operation ensured better boundary cohesion, while watershed segmentation effectively separated overlapping cells. Here is the result of a Type B Figure 3.2.3.

Success Rate:

- **Class A:** 95% detection accuracy (comparable to base method).
- **Class B:** 85% detection accuracy, showing the best performance for overlapping cells.
- **Class C:** 35% detection accuracy, with slight improvements for irregular boundaries.

—

4.5 Comparison of Methods

The following table summarizes the success rates of each method across the three classes:

Method	Class A	Class B	Class C
Base Method	95%	60%	30%
Morphological Closing	98%	80%	45%
Watershed Segmentation	95%	80%	30%
Closing + Watershed	95%	85%	35%

Table 1: Comparison of success rates for each method across the three classes.

—

Discussion: Each method provided unique strengths and weaknesses:

- The **base method** was effective for Class A images but struggled with overlapping cells and faint boundaries.
- **Morphological closing** improved the detection of closely packed cells by filling gaps in boundaries.

- **Watershed segmentation** addressed overlapping cells but introduced over-segmentation in some cases.
- Combining **morphological closing with watershed segmentation** offered the best overall performance, particularly for Class B images.

These results 1 demonstrate the importance of tailoring methods to specific challenges in cell detection and highlight the potential for combining techniques to achieve optimal results.

5 Challenges and Reasons for Failures

While building the cell counting system, we faced several challenges that made the process more complex and introduced potential errors in the results. These challenges highlighted the limitations of the approach and areas that could be improved in future iterations.

One major challenge was identifying the specific shade of purple used to stain the cells. Since the cell images relied on color for differentiation, accurately setting the HSV (hue, saturation, and value) threshold was critical. Even slight variations in the color could result in cells being missed or incorrectly detected as background noise.

Another issue was with cells that extended beyond the edges of the image. These out-of-bounds cells couldn't be fully captured, which made it difficult to count them accurately. In some cases, parts of these cells were excluded entirely, leading to undercounting.

Additionally, some cells in the images were too close to each other, often overlapping or touching. This caused the system to detect them as a single object rather than separate cells. On the other hand, there were cells that weren't thick enough or had weak boundaries. These cells were sometimes mistakenly split into multiple parts, as the segmentation process "cut" them in the middle.

These challenges highlight the trade-offs involved in using a mathematical approach without external libraries. While the system worked well for many cases, improvements are needed to handle color variations, overlapping cells, and weak boundaries more effectively.

6 Conclusion

This project set out to develop an effective pipeline for detecting and counting cells in microscopy images using image processing techniques. Despite facing several challenges, the project was largely successful, achieving accurate cell detection while exploring a variety of methods that provided new perspectives on image processing.

The core pipeline relied on converting images from RGB to HSV, applying precise thresholding, and using connected component analysis to segment individual cells. These foundational steps offered valuable insights into how mathematical principles can be applied to isolate and detect objects in complex images. However, challenges like overlapping cells, faint boundaries, and out-of-bounds regions highlighted the limitations of these basic techniques.

To address some of these issues, advanced methods such as morphological closing, distance transformations, and watershed segmentation were incorporated. Morphological closing proved effective in bridging small gaps within cell boundaries, ensuring more cohesive detection. Distance transformation helped to highlight cell centers and provided a foundation for separating clustered cells. Watershed segmentation, when applied carefully, offered a way to split overlapping cells by treating the distance-transformed image as a topographical map, where local minima were treated as cell centers. These additional methods enriched the pipeline, providing flexibility and robustness in tackling more complex scenarios.

While the algorithm performed well overall, it required precise parameter tuning and still struggled in cases of severe overlap or faint staining. These challenges emphasize the need for future enhancements, such as integrating machine learning models to adaptively handle variations in cell appearance and improve segmentation accuracy. A more dynamic approach and better results can be achieved this way.

In conclusion, this project demonstrated how a combination of basic and advanced image processing techniques can address the intricate problem of cell detection. Each method added a unique perspective to the pipeline, highlighting the trade-offs between simplicity and complexity. The results validated the effectiveness of the approach, while the challenges provided valuable learning opportunities for future work in automated cell analysis.

References

[1] OpenCV Documentation. Distance transform, 2024. Accessed: 2024-12-28.

[2] OpenCV Documentation. Morphological operations, 2024. Accessed: 2024-12-28.

[3] OpenCV Documentation. Watershed algorithm, 2024. Accessed: 2024-12-28.

Appendix

HSV Threshold

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Convert RGB image to HSV
def rgb_to_hsv(image):
    image = image / 255.0  # Normalize the image
    hsv_image = np.zeros_like(image)
    max_rgb = image.max(axis=2)
    min_rgb = image.min(axis=2)
    diff_rgb = max_rgb - min_rgb

    # Hue calculation
    mask_r = (max_rgb == image[:, :, 0])
    hsv_image[:, :, 0][mask_r] =
    (60 * (image[:, :, 1] - image[:, :, 2]) / diff_rgb + 360)[mask_r] % 360

    mask_g = (max_rgb == image[:, :, 1])

    hsv_image[:, :, 0][mask_g] =
    60 * (image[:, :, 2] - image[:, :, 0]) / diff_rgb + 120)[mask_g] % 360

    mask_b = (max_rgb == image[:, :, 2])

    hsv_image[:, :, 0][mask_b] =
    (60 * (image[:, :, 0] - image[:, :, 1]) / diff_rgb + 240)[mask_b] % 360

    # Saturation calculation
    hsv_image[:, :, 1] = np.where(max_rgb == 0, 0, (diff_rgb / max_rgb))

    # Value calculation
    hsv_image[:, :, 2] = max_rgb

    hsv_image[:, :, 0] = hsv_image[:, :, 0] / 2  # Scale hue to 0-180
    hsv_image[:, :, 1] = hsv_image[:, :, 1] * 255  # Scale saturation to 0-255
    hsv_image[:, :, 2] = hsv_image[:, :, 2] * 255  # Scale value to 0-255
    return hsv_image.astype(np.uint8)

# Apply HSV threshold
def apply_hsv_threshold(hsv_image, h_min, s_min, v_min, h_max, s_max, v_max):
    mask = ((hsv_image[:, :, 0] >= h_min) & (hsv_image[:, :, 0] <= h_max) &
            (hsv_image[:, :, 1] >= s_min) & (hsv_image[:, :, 1] <= s_max) &
            (hsv_image[:, :, 2] >= v_min) & (hsv_image[:, :, 2] <= v_max))
    binary_image = np.zeros_like(hsv_image[:, :, 0])
    binary_image[mask] = 255
    return binary_image

# Load the image
def load_image(image_path):
    img = Image.open(image_path)
    img = np.array(img)
    return img
```

```
# Plot the image
def plot_image(image, title):
    plt.imshow(image, cmap='gray' if len(image.shape) == 2 else None)
    plt.title(title)
    plt.axis('off')
    plt.show()

# Main code
image_path = 'T07-00439_p211.png'  # Sample image
image = load_image(image_path)
plot_image(image, 'Original Image')

# Convert to HSV and apply the threshold
hsv_image = rgb_to_hsv(image)
binary_image = apply_hsv_threshold(
hsv_image, h_min=129, s_min=100, v_min=100, h_max=142, s_max=202, v_max=175)
plot_image(binary_image, 'HSV Thresholded Image')
```

Connecting Components and Cell Counting

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Load the binary image
def load_binary_image(image_path):
    img = Image.open(image_path).convert('L')  # Convert to grayscale
    binary_image = np.array(img)
    binary_image[binary_image > 0] = 255
    return binary_image

# Connected component labeling
def connected_components(binary_image, min_size=100):
    visited = np.zeros_like(binary_image, dtype=bool)
    rows, cols = binary_image.shape
    cell_count = 0
    detected_cells = []

    def flood_fill(x, y):
        stack = [(x, y)]
        component = []

        while stack:
            cx, cy = stack.pop()
            if (0 <= cx < rows and 0 <= cy < cols and
                    binary_image[cx, cy] == 255 and not visited[cx, cy]):
                visited[cx, cy] = True
                component.append((cx, cy))

                # Check the 8 neighboring pixels
                for dx in [-1, 0, 1]:
                    for dy in [-1, 0, 1]:
                        if dx != 0 or dy != 0:
                            stack.append((cx + dx, cy + dy))

        return component

    # Iterate over the image to find components
    for i in range(rows):
        for j in range(cols):
            if binary_image[i, j] == 255 and not visited[i, j]:
                component = flood_fill(i, j)
                if len(component) >= min_size:  # Only keep larger components
                    cell_count += 1
                    detected_cells.append(component)

    return cell_count, detected_cells

# Highlight detected cells on the image
def highlight_cells(binary_image, detected_cells):
    color_image = np.stack([binary_image] * 3, axis=-1)  # Convert to RGB
    for component in detected_cells:
        for (x, y) in component:
            color_image[x, y] = [255, 0, 0]  # Highlight in red
    return color_image
```

```
# Plot the image
def plot_image(image, title):
    plt.imshow(image)
    plt.title(title)
    plt.axis('off')
    plt.show()

# Main code
plot_image(binary_image, 'Binary Image for Cell Counting')
min_size_threshold = 1500
cell_count, detected_cells = connected_components(binary_image, min_size=min_size_threshold)

# Highlight detected cells
highlighted_image = highlight_cells(binary_image, detected_cells)
plot_image(highlighted_image, 'Detected Cells Highlighted')

print(f'Total cells detected (size >= {min_size_threshold} pixels): {cell_count}')
```

Morphological Operations: Dilation, Erosion, and Closing

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

def dilate(binary_image, kernel_size=3):
    rows, cols = binary_image.shape
    padding = kernel_size // 2
    dilated_image = np.zeros_like(binary_image)

    for i in range(padding, rows - padding):
        for j in range(padding, cols - padding):
            # Check if any pixel in the kernel neighborhood is white
            if np.any(binary_image[i-padding:i+padding+1, j-padding:j+padding+1] == 255):
                dilated_image[i, j] = 255

    return dilated_image

def erode(binary_image, kernel_size=3):
    rows, cols = binary_image.shape
    padding = kernel_size // 2
    eroded_image = np.zeros_like(binary_image)

    for i in range(padding, rows - padding):
        for j in range(padding, cols - padding):
            # Check if all pixels in the kernel neighborhood are white
            if np.all(binary_image[i-padding:i+padding+1, j-padding:j+padding+1] == 255):
                eroded_image[i, j] = 255

    return eroded_image

def morphological_closing(binary_image, kernel_size=3):
    dilated_image = dilate(binary_image, kernel_size=3)
    closed_image = erode(dilated_image, kernel_size=3)
    return closed_image

# Main code
kernel_size = 3
closed_image = morphological_closing(binary_image, kernel_size=kernel_size)
plot_image(closed_image, "Binary Image After Morphological Closing")

# Count cells on the closed binary image
cell_count, detected_cells = connected_components(closed_image, min_size=min_size_threshold)

# Highlight detected cells after morphological closing
highlighted_image = highlight_cells(closed_image, detected_cells)
plot_image(highlighted_image, "Highlighted Cells After Morphological Closing")

print(f"Total cells detected (size >= {min_size_threshold} pixels): {cell_count}")
```