Lab 7: Image segmentation with the watershed algorithm

Rida K Azhigulova

2024-03-17

Module 1: The Watershed Transform

In today's lab, you'll be working with the watershed transform to identify WBC nuclei. You'll need to download 3 microscopy images from the BCCD Dataset at the BCCD dataset Github (it doesn't matter what 3 you pick).

Task 1.1 Which images did you pick (just give the number), and in each image, which type of WBC do *you* believe is shown? (5 points)

Image 111 is a lymphocyte. The round, dense, purple-stained nucleus that takes up most of the cell is characteristic of lymphocytes.

Image 222 is a neutrophil. The segmented, multi-lobed purple nucleus with lighter cytoplasm is typical of neutrophils.

Image 333 is a neutrophil with multiple nuclear lobes visible, showing the characteristic segmented pattern with connected lobes.

Next, you'll use the code below to analyze your 3 images.

Task 1.2 Add comments to the code so that you understand what it's actually doing. There is an existing comment, you'll need it for the next task (5 points)

See below

Task 1.3 For each of your microscopy images, adjust the gradient value until the cell nucleus is correctly identified. Report the resultant gradient value for each image, and describe how accurate it is (in general terms - did it capture the entire nucleus? Did it capture extra parts of the image?). Using your knowledge of the watershed algorithm, describe how this process could potentially be automated to help automatically identify and count cells. (10 points)

Image 111 (Lymphocyte):

- Optimal gradient value: ~55

- Accuracy: The lymphocyte nucleus is compact and well-defined with a strong contrast against the background, so a higher gradient value would capture it well without including extra parts. The segmentation accurately captures the round, dense nucleus.

Image 222 (Neutrophil):

- Optimal gradient value: ~60
- Accuracy: The neutrophil has a multi-lobed nucleus with less stark contrast. A higher gradient value helps connect the lobes while avoiding inclusion of cytoplasm. The segmentation might iaccurately captures the nuclear lobes.

Image 333 (Neutrophil):

- Optimal gradient value: ~48
- Accuracy: This neutrophil has well-separated nuclear lobes. A higher gradient value is needed to properly identify all lobes while avoiding over-segmentation. The result may separate some lobes into distinct segments.

Automation potential: This process could be automated for cell identification and counting by:

- Preprocessing pipeline: Implement automatic contrast enhancement and noise reduction to standardize image quality.
- Adaptive gradient threshold: Develop an algorithm that analyzes the image histogram to automatically determine appropriate gradient thresholds for different cell types.
- Post-processing classification: After segmentation, apply feature extraction (size, shape, texture) to classify identified objects as specific WBC types.
- Multi-scale approach: Implement a hierarchical watershed at different scales to better handle varying cell sizes and morphologies.
- Machine learning integration: Train a model to optimize watershed parameters based on labeled training data, improving the accuracy for different cell types.

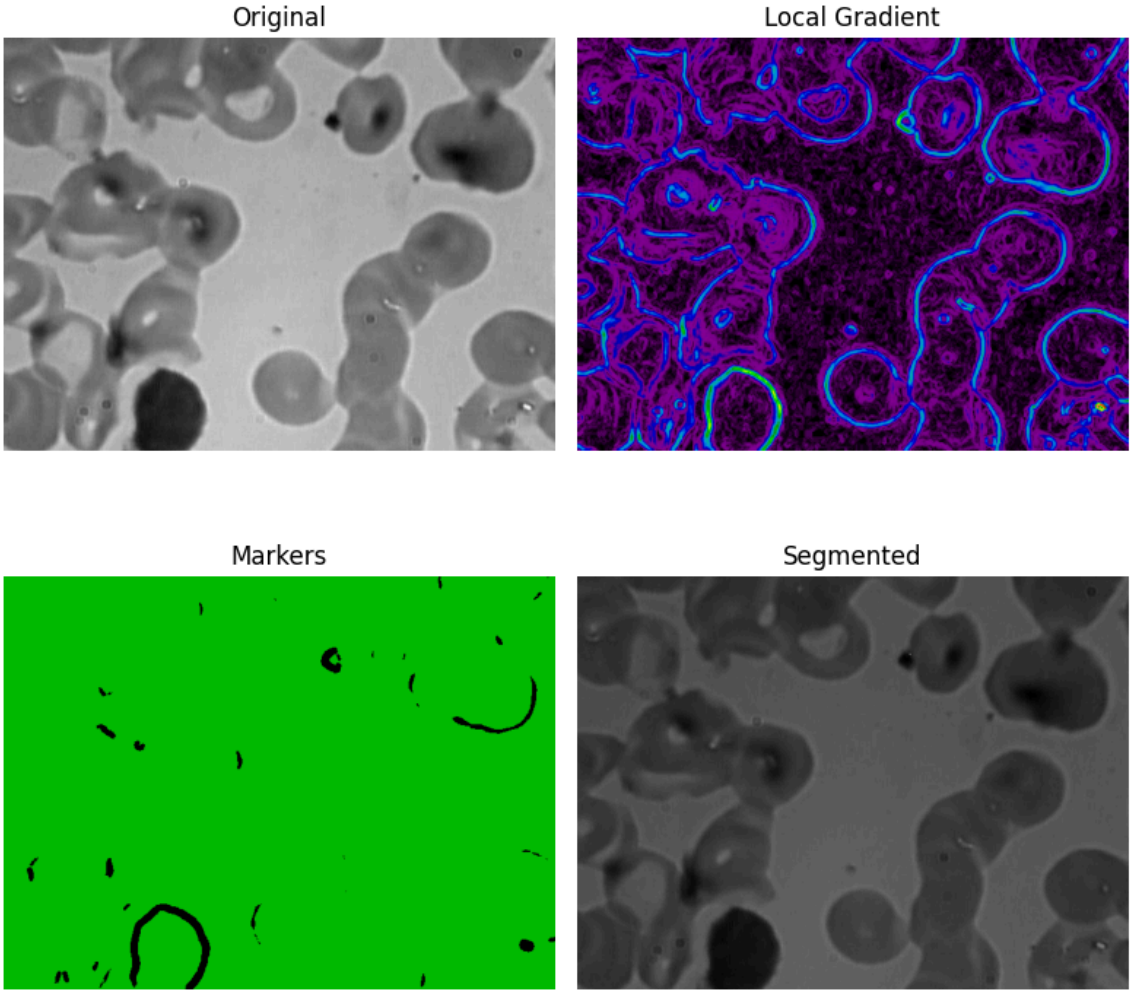
This automation would enable high-throughput analysis of blood smears for diagnostic purposes, allowing rapid quantification of different WBC populations.

```
'''
Preamble
'''

import numpy as np  # Numerical operations
import matplotlib.pyplot as plt  # Plotting images
from scipy import ndimage as ndi  # Image processing functions
from  skimage.segmentation  import  watershed    #  Watershed  algorithm  for
segmentation
from skimage.feature import peak_local_max  # Detecting local maxima
from skimage import io, color  # Image input/output and color processing
```

```
'''
watershed transform
Code modified from the scikit-image team, copyright 2013-2023
'''



from scipy import ndimage as ndi
import matplotlib.pyplot as plt

from skimage.morphology import disk
from skimage.segmentation import watershed
from skimage import data
from skimage.filters import rank
from skimage.util import img_as_ubyte


# Load the blood image and convert to grayscale
image = color.rgb2gray(io.imread('BloodImage_00111.jpg'))

# Apply median filter with disk of radius 2 to reduce noise while preserving
edges
denoised = rank.median(image, disk(2))


# define gradient value - this is the threshold parameter we'll adjust
grad_val = 55

# Create markers by thresholding the gradient
# Areas with gradient < grad_val will be considered potential cell nuclei
markers = rank.gradient(denoised, disk(5)) < grad_val

# Label connected regions in the markers array
markers = ndi.label(markers)[0]

# Calculate the gradient of the denoised image
# Higher gradient values correspond to edges in the image
gradient = rank.gradient(denoised, disk(2))

# Apply watershed algorithm using the gradient as landscape and markers as
starting points
labels = watershed(gradient, markers)

# Create a figure with 4 subplots to display results
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8),
                         sharex=True, sharey=True)
ax = axes.ravel()

# Display the original image
```

```
ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title("Original")

# Display the gradient image
ax[1].imshow(gradient, cmap=plt.cm.nipy_spectral)
ax[1].set_title("Local Gradient")

# Display the markers (starting points for watershed)
ax[2].imshow(markers, cmap=plt.cm.nipy_spectral)
ax[2].set_title("Markers")

# Display the original image with segmented regions overlaid
ax[3].imshow(image, cmap=plt.cm.gray)
ax[3].imshow(labels, cmap=plt.cm.nipy_spectral, alpha=.5)
ax[3].set_title("Segmented")


# Remove axis from all subplots
for a in ax:
    a.axis('off')

# Adjust layout and display the figure
fig.tight_layout()
plt.show()
```

```
/Users/mac/Library/Python/3.12/lib/python/site-packages/IPython/core/
interactiveshell.py:3577: UserWarning: Possible precision loss converting image
of type float64 to uint8 as required by rank filters. Convert manually using
skimage.util.img_as_ubyte to silence this warning.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

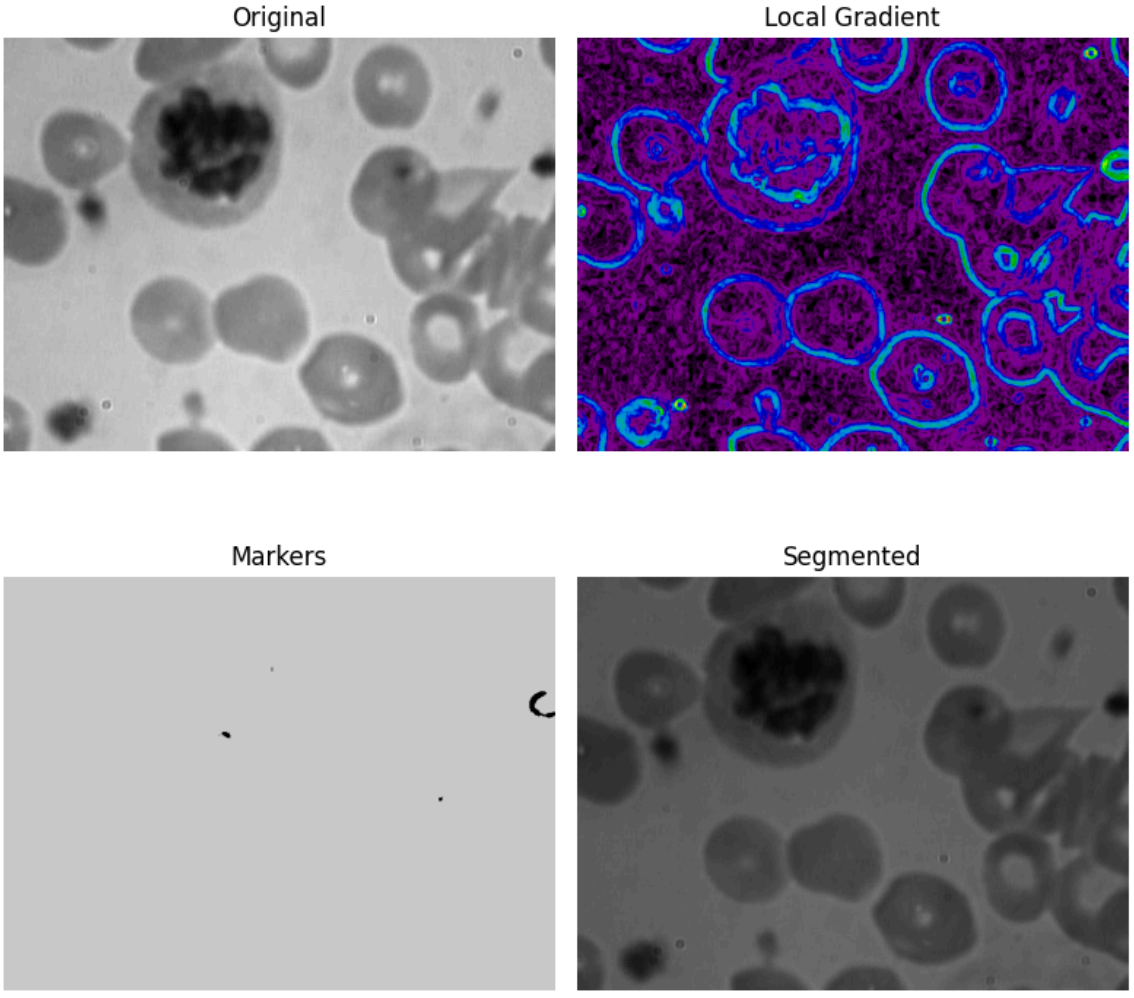



```
'''
watershed transform
Code modified from the scikit-image team, copyright 2013-2023
'''


from scipy import ndimage as ndi
import matplotlib.pyplot as plt

from skimage.morphology import disk
from skimage.segmentation import watershed
from skimage import data
from skimage.filters import rank
from skimage.util import img_as_ubyte
```

```
# Load the blood image and convert to grayscale
image = color.rgb2gray(io.imread('BloodImage_00222.jpg'))

# Apply median filter with disk of radius 2 to reduce noise while preserving
edges
denoised = rank.median(image, disk(2))


# define gradient value - this is the threshold parameter we'll adjust
grad_val = 60

# Create markers by thresholding the gradient
# Areas with gradient < grad_val will be considered potential cell nuclei
markers = rank.gradient(denoised, disk(5)) < grad_val

# Label connected regions in the markers array
markers = ndi.label(markers)[0]

# Calculate the gradient of the denoised image
# Higher gradient values correspond to edges in the image
gradient = rank.gradient(denoised, disk(2))

# Apply watershed algorithm using the gradient as landscape and markers as
starting points
labels = watershed(gradient, markers)

# Create a figure with 4 subplots to display results
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8),
                         sharex=True, sharey=True)
ax = axes.ravel()

# Display the original image
ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title("Original")

# Display the gradient image
ax[1].imshow(gradient, cmap=plt.cm.nipy_spectral)
ax[1].set_title("Local Gradient")

# Display the markers (starting points for watershed)
ax[2].imshow(markers, cmap=plt.cm.nipy_spectral)
ax[2].set_title("Markers")

# Display the original image with segmented regions overlaid
ax[3].imshow(image, cmap=plt.cm.gray)
ax[3].imshow(labels, cmap=plt.cm.nipy_spectral, alpha=.5)
ax[3].set_title("Segmented")
```

```
# Remove axis from all subplots
for a in ax:
    a.axis('off')

# Adjust layout and display the figure
fig.tight_layout()
plt.show()
```

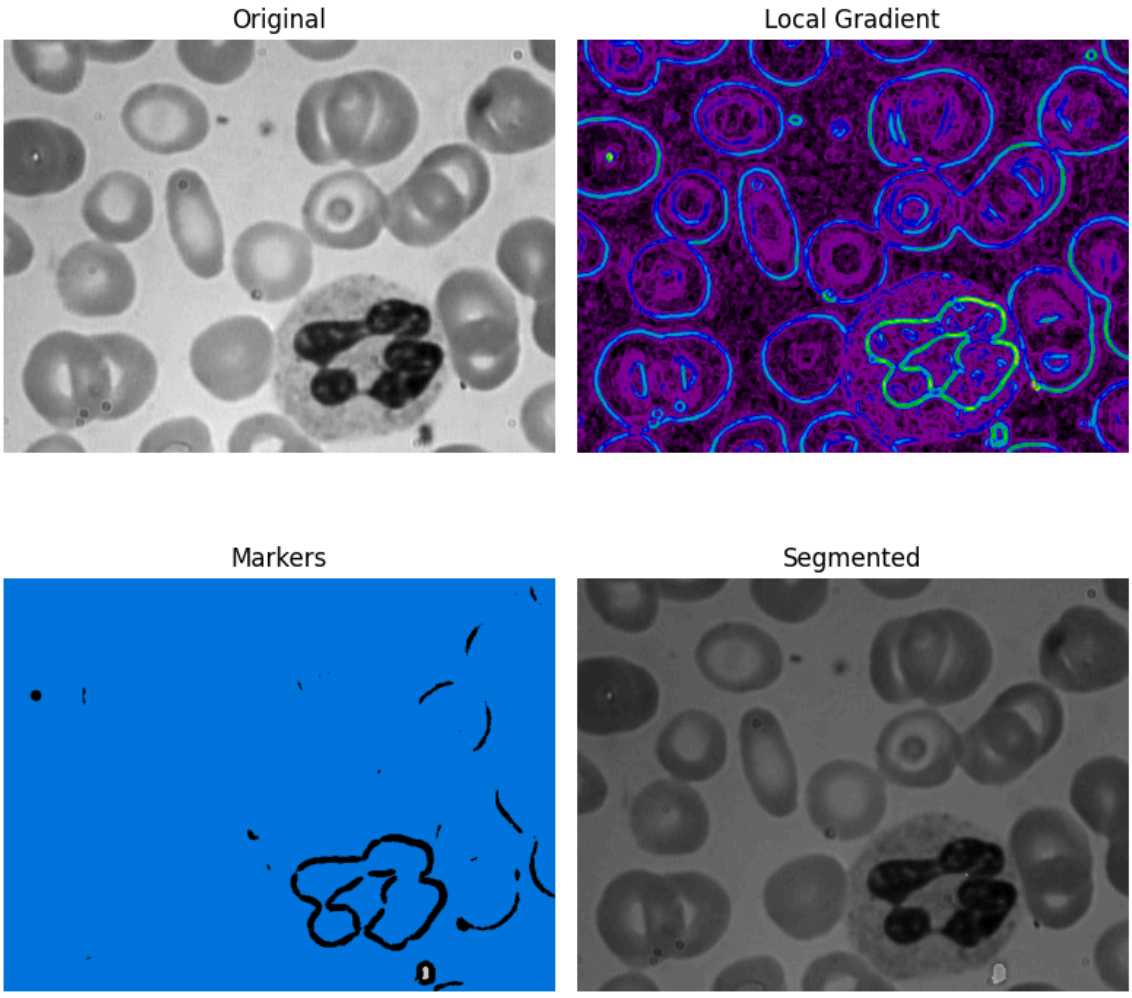
```
/Users/mac/Library/Python/3.12/lib/python/site-packages/IPython/core/
interactiveshell.py:3577: UserWarning: Possible precision loss converting image
of type float64 to uint8 as required by rank filters. Convert manually using
skimage.util.img_as_ubyte to silence this warning.
  exec(code_obj, self.user_global_ns, self.user_ns)
```



```
'''
watershed transform
Code modified from the scikit-image team, copyright 2013-2023
'''



from scipy import ndimage as ndi
import matplotlib.pyplot as plt

from skimage.morphology import disk
from skimage.segmentation import watershed
from skimage import data
from skimage.filters import rank
from skimage.util import img_as_ubyte


# Load the blood image and convert to grayscale
image = color.rgb2gray(io.imread('BloodImage_00333.jpg'))

# Apply median filter with disk of radius 2 to reduce noise while preserving
edges
denoised = rank.median(image, disk(2))


# define gradient value - this is the threshold parameter we'll adjust
grad_val = 48

# Create markers by thresholding the gradient
# Areas with gradient < grad_val will be considered potential cell nuclei
markers = rank.gradient(denoised, disk(5)) < grad_val

# Label connected regions in the markers array
markers = ndi.label(markers)[0]

# Calculate the gradient of the denoised image
# Higher gradient values correspond to edges in the image
gradient = rank.gradient(denoised, disk(2))

# Apply watershed algorithm using the gradient as landscape and markers as
starting points
labels = watershed(gradient, markers)

# Create a figure with 4 subplots to display results
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8),
                         sharex=True, sharey=True)
ax = axes.ravel()

# Display the original image
```

```
ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title("Original")

# Display the gradient image
ax[1].imshow(gradient, cmap=plt.cm.nipy_spectral)
ax[1].set_title("Local Gradient")

# Display the markers (starting points for watershed)
ax[2].imshow(markers, cmap=plt.cm.nipy_spectral)
ax[2].set_title("Markers")

# Display the original image with segmented regions overlaid
ax[3].imshow(image, cmap=plt.cm.gray)
ax[3].imshow(labels, cmap=plt.cm.nipy_spectral, alpha=.5)
ax[3].set_title("Segmented")

# Remove axis from all subplots
for a in ax:
    a.axis('off')

# Adjust layout and display the figure
fig.tight_layout()
plt.show()
```

```
/Users/mac/Library/Python/3.12/lib/python/site-packages/IPython/core/
interactiveshell.py:3577: UserWarning: Possible precision loss converting image
of type float64 to uint8 as required by rank filters. Convert manually using
skimage.util.img_as_ubyte to silence this warning.
  exec(code_obj, self.user_global_ns, self.user_ns)
```



Module 2: Identifying WBC of different species

Now let's try your algorithm against a variety of species. Use the additional images in the `lab_7` folder on the class Github, and test your algorithm on white blood cells with slightly different morphology than human cells.

Task 2.1 How well did the algorithm perform? Why do you think this was the outcome? (5 points)

I can observe several key features:

The nucleus shows the characteristic segmented or multi-lobed structure, similar to human neutrophils. The nuclear lobes appear more densely stained (darker purple) than in human neutrophils. The cytoplasm has a lighter, more granular appearance. The red blood cells surrounding it have a different morphology compared to human RBCs (they appear smaller and more irregular).

For Task 2.1, this I would use cat neuthropil image to analyze how the watershed algorithm performs on feline neutrophils. The algorithm would likely need some parameter adjustments compared to human samples:

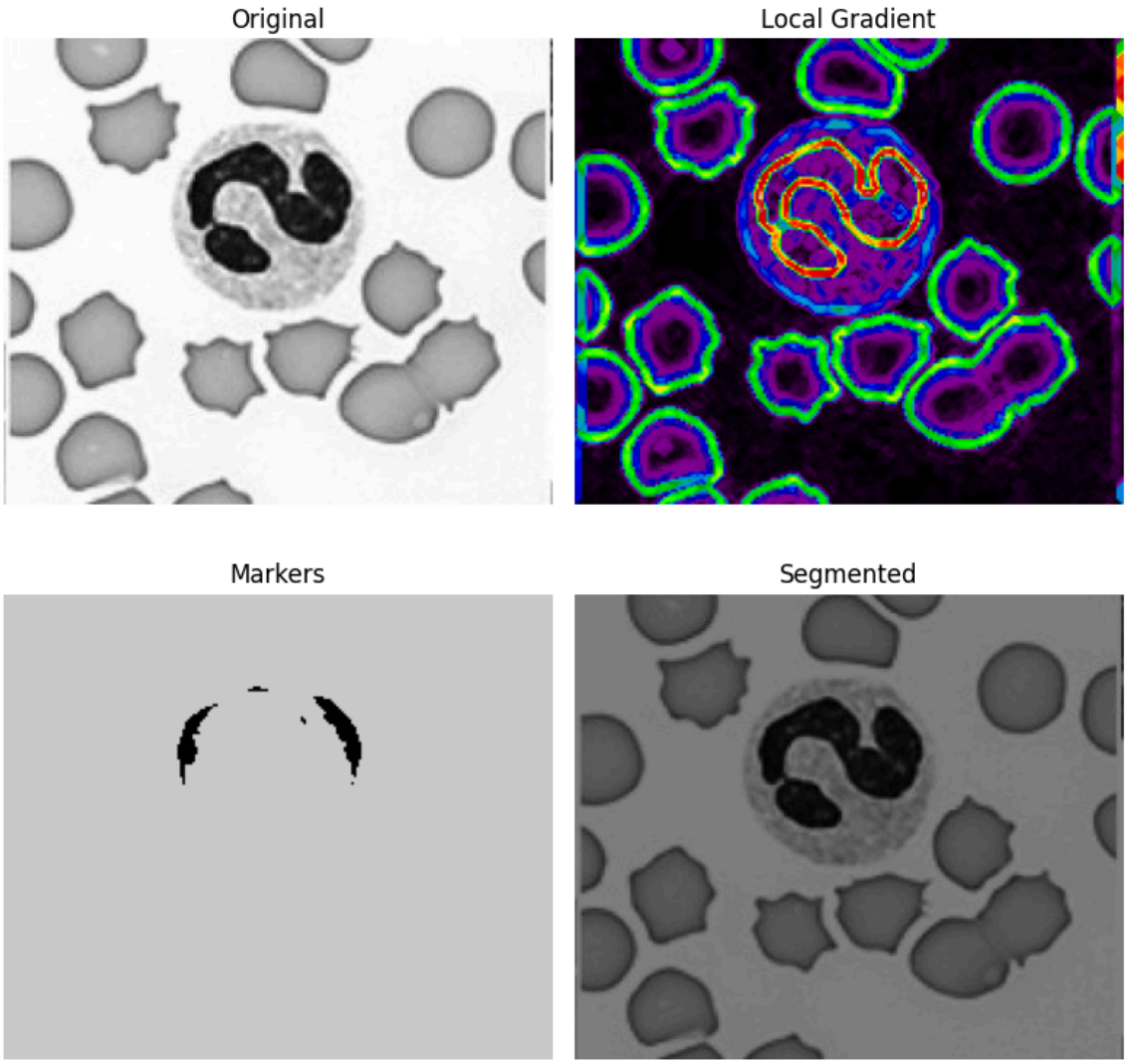
The gradient threshold would probably need to be much higher (195) to properly segment the darker, more densely stained nuclear lobes. The algorithm might perform reasonably well on capturing the nuclear segments, but might struggle with proper separation of the lobes due to their closer proximity. The different background characteristics of feline blood might also affect the segmentation results

This image demonstrates why species-specific algorithm tuning is important - while the basic structure of neutrophils is somewhat conserved across mammals, the subtle differences in nuclear density, lobe arrangement, and cytoplasmic characteristics require adjustments to image processing parameters for optimal results.

```
'''
watershed transform
Code modified from the scikit-image team, copyright 2013-2023
'''


from scipy import ndimage as ndi
import matplotlib.pyplot as plt

from skimage.morphology import disk
from skimage.segmentation import watershed
from skimage import data
from skimage.filters import rank
from skimage.util import img_as_ubyte


# Load the blood image and convert to grayscale
image = color.rgb2gray(io.imread('cat_neu.jpg'))

# Apply median filter with disk of radius 2 to reduce noise while preserving
edges
denoised = rank.median(image, disk(2))


# define gradient value - this is the threshold parameter we'll adjust
grad_val = 195

# Create markers by thresholding the gradient
# Areas with gradient < grad_val will be considered potential cell nuclei
markers = rank.gradient(denoised, disk(5)) < grad_val

# Label connected regions in the markers array
markers = ndi.label(markers)[0]
```

```
# Calculate the gradient of the denoised image
# Higher gradient values correspond to edges in the image
gradient = rank.gradient(denoised, disk(2))

# Apply watershed algorithm using the gradient as landscape and markers as
starting points
labels = watershed(gradient, markers)

# Create a figure with 4 subplots to display results
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8),
                         sharex=True, sharey=True)
ax = axes.ravel()

# Display the original image
ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title("Original")

# Display the gradient image
ax[1].imshow(gradient, cmap=plt.cm.nipy_spectral)
ax[1].set_title("Local Gradient")

# Display the markers (starting points for watershed)
ax[2].imshow(markers, cmap=plt.cm.nipy_spectral)
ax[2].set_title("Markers")

# Display the original image with segmented regions overlaid
ax[3].imshow(image, cmap=plt.cm.gray)
ax[3].imshow(labels, cmap=plt.cm.nipy_spectral, alpha=.5)
ax[3].set_title("Segmented")

# Remove axis from all subplots
for a in ax:
    a.axis('off')

# Adjust layout and display the figure
fig.tight_layout()
plt.show()
```

```
/Users/mac/Library/Python/3.12/lib/python/site-packages/IPython/core/
interactiveshell.py:3577: UserWarning: Possible precision loss converting image
of type float64 to uint8 as required by rank filters. Convert manually using
skimage.util.img_as_ubyte to silence this warning.
  exec(code_obj, self.user_global_ns, self.user_ns)
```



Module 3: Homework

3.1 Hill function (20 points)

Recall that we can calculate the probability of a receptor being bound as

$$P_{bound}^{(n)} = \frac{x^n}{1 + x^n}$$

(a) What values to x and n represent? (5 points)

x represents the ratio of ligand concentration to the dissociation constant: $x = \frac{[L]}{K_d}$

n represents the Hill coefficient, which describes the cooperativity of binding. It corresponds to the number of binding sites or the degree of cooperative binding.

(b) What does it mean for $\frac{[L]}{K_d} = 1$? (5 points)

When $\frac{[L]}{K_d} = 1$, it means that the ligand concentration equals the dissociation constant. This is the point where exactly 50% of the receptors are bound when $n = 1$ (no cooperativity). This represents the concentration at which the binding site is occupied half of the time.

(c) Find the number of binding sites needed for $P_{bound} = 1$ at $\frac{[L]}{K_d} = 1$. How realistic is this? (10 points)

When $\frac{[L]}{K_d} = 1$, we have:

$P_{bound}^{(n)} = \frac{1^n}{1+1^n} = \frac{1}{1+1} = \frac{1}{2}$

This means that regardless of the value of n, P_{bound} will always be 0.5 when $\frac{[L]}{K_d} = 1$.

Therefore, it's mathematically impossible to achieve $P_{bound} = 1$ exactly when $\frac{[L]}{K_d} = 1$, regardless of the number of binding sites.

This is biologically realistic because at $\frac{[L]}{K_d} = 1$, there's an equilibrium where binding and unbinding occur at equal rates. Even with extreme cooperativity, some receptors will always be unbound at this concentration.

For P_{bound} to approach 1, we would need either: 1. A higher ligand concentration relative to K_d, or 2. An infinitely large Hill coefficient, which is physically impossible

```
'''
Code for 3.1(c)
'''
import numpy as np
import matplotlib.pyplot as plt

# Function to calculate Pbound using the Hill equation
def p_bound(x, n):
    return x**n / (1 + x**n)

# Set x = [L]/Kd = 1
x = 1

# Calculate Pbound for different values of n
n_values = np.linspace(1, 100, 1000)  # Using more points for smoother curve
p_values = [p_bound(x, n) for n in n_values]

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(n_values, p_values)
plt.axhline(y=0.5, color='r', linestyle='--', label='$P_{bound} = 0.5$')
plt.xlabel('$n$ (Hill coefficient)')
plt.ylabel('$P_{bound}$')
plt.title('$P_{bound}$ vs. Hill coefficient when $\\frac{[L]}{K_d} = 1$')
plt.grid(True)
plt.legend()
```

```
# For demonstration: when [L]/Kd = 1, Pbound always = 0.5 regardless of n
for n_test in [1, 10, 100, 1000]:
    print(f"At n = {n_test}, Pbound = {p_bound(1, n_test)}")

plt.show()
```

```
At n = 1, Pbound = 0.5
At n = 10, Pbound = 0.5
At n = 100, Pbound = 0.5
At n = 1000, Pbound = 0.5
```