Assignment 1: Pixel Classification & Grey Level Thresholding

- Write a Python function that classifies pixels in a grayscale image into two classes based on a given threshold.
- Implement global thresholding using a fixed threshold.
- Implement local thresholding using a window-based approach.
- Compare the results of global and local thresholding.

Assignment 2: Optimum Thresholding - Bayes Analysis

- Generate a synthetic bimodal histogram representing two classes.
- Implement Bayes thresholding based on the histogram.
- Apply the threshold to segment the image.

Assignment 3: Otsu Method

- Implement the Otsu method to find the optimal threshold.
- Apply the threshold to segment the image.
- Visualize the results.

Assignment 8

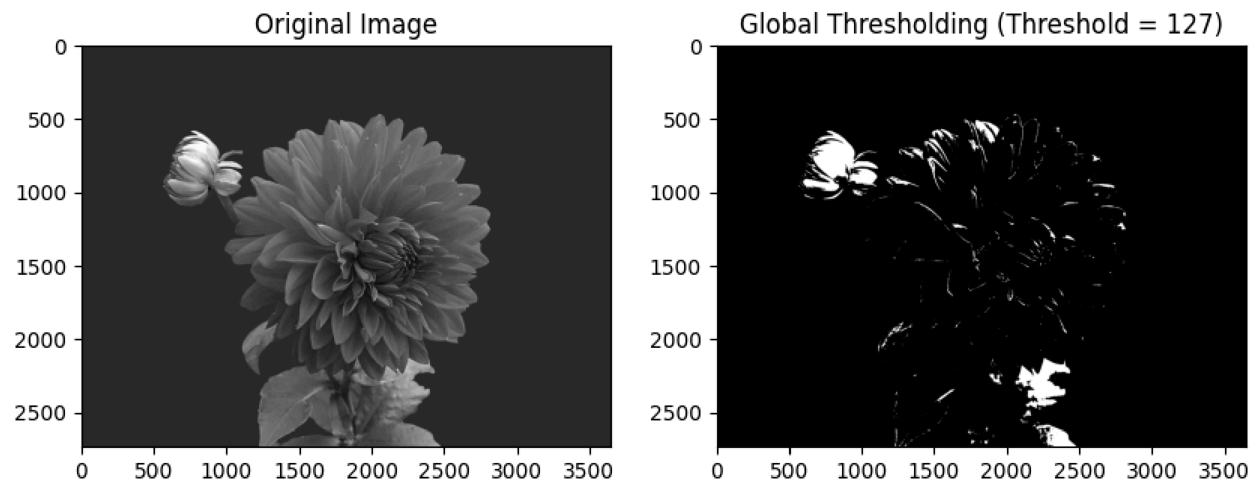
Segmentation

1. **Assignment 1:** Pixel Classification & Grey Level Thresholding

● Write a Python function that classifies pixels in a grayscale image into two classes based on a given threshold.
● Implement global thresholding using a fixed threshold.

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

def global_thresholding(image_path, threshold):
    # Load the image in grayscale
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    # Apply global thresholding
    _, binary_img = cv2.threshold(img, threshold, 255, cv2.THRESH_BINARY)

    # Display the original and thresholded images
    plt.figure(figsize=(10,5))
    plt.subplot(1, 2, 1)
    plt.title("Original Image")
    plt.imshow(img, cmap='gray')
    plt.subplot(1, 2, 2)
    plt.title(f"Global Thresholding (Threshold = {threshold})")
    plt.imshow(binary_img, cmap='gray')
    plt.show()
# Example usage
global_thresholding('grayscale_image.jpg', 127)
```

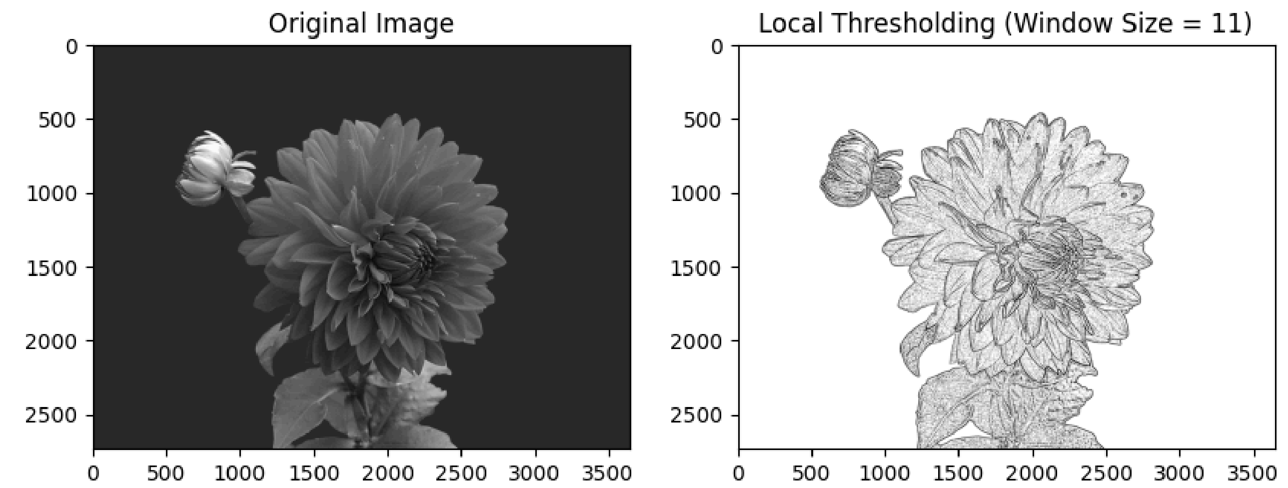
Output:



● Implement local thresholding using a window-based approach.

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

def local_thresholding(image_path, window_size, C=0):
    # Load the image in grayscale
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Apply adaptive thresholding (mean-based)
    binary_img = cv2.adaptiveThreshold(
        img, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY,
window_size, C)

    # Display the original and local thresholded images
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title("Original Image")
    plt.imshow(img, cmap='gray')

    plt.subplot(1, 2, 2)
    plt.title(f"Local Thresholding (Window Size = {window_size})")
    plt.imshow(binary_img, cmap='gray')
    plt.show()

# Example usage
local_thresholding('grayscale_image.jpg', window_size=11, C=2)
```

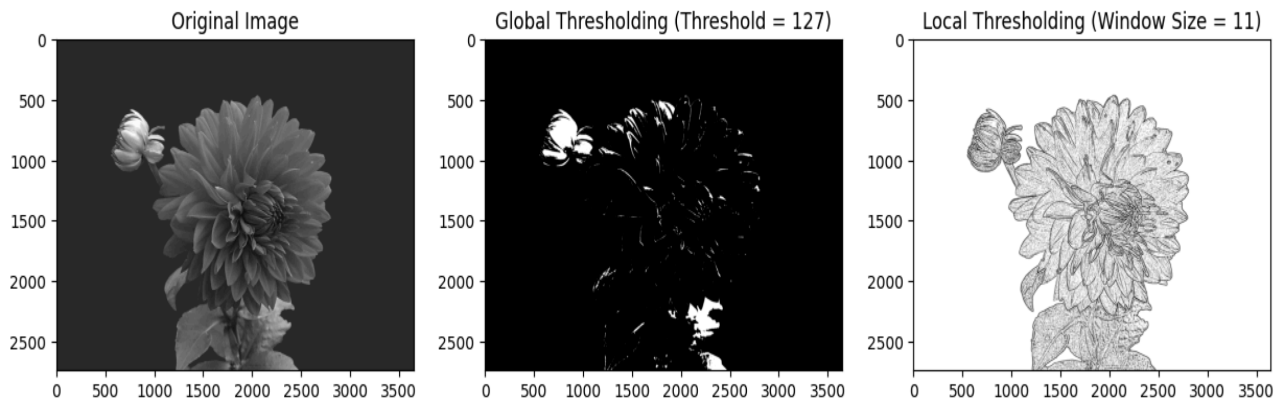
Output:



● Compare the results of global and local thresholding.

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

def compare_thresholding(image_path, global_thresh, local_window_size, C=0):
    # Load the image in grayscale
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Apply global thresholding
    _, global_binary_img = cv2.threshold(img, global_thresh, 255,
cv2.THRESH_BINARY)
    # Apply local thresholding (adaptive mean)
    local_binary_img = cv2.adaptiveThreshold(
        img, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY,
local_window_size, C)
    # Display the results for comparison
    plt.figure(figsize=(15, 5))
    plt.subplot(1, 3, 1)
    plt.title("Original Image")
    plt.imshow(img, cmap='gray')
    plt.subplot(1, 3, 2)
    plt.title(f"Global Thresholding (Threshold = {global_thresh})")
    plt.imshow(global_binary_img, cmap='gray')
    plt.subplot(1, 3, 3)
    plt.title(f"Local Thresholding (Window Size = {local_window_size})")
    plt.imshow(local_binary_img, cmap='gray')
    plt.show()
# Example usage
compare_thresholding('grayscale_image.jpg', global_thresh=127,
local_window_size=11, C=2)
```

Output:

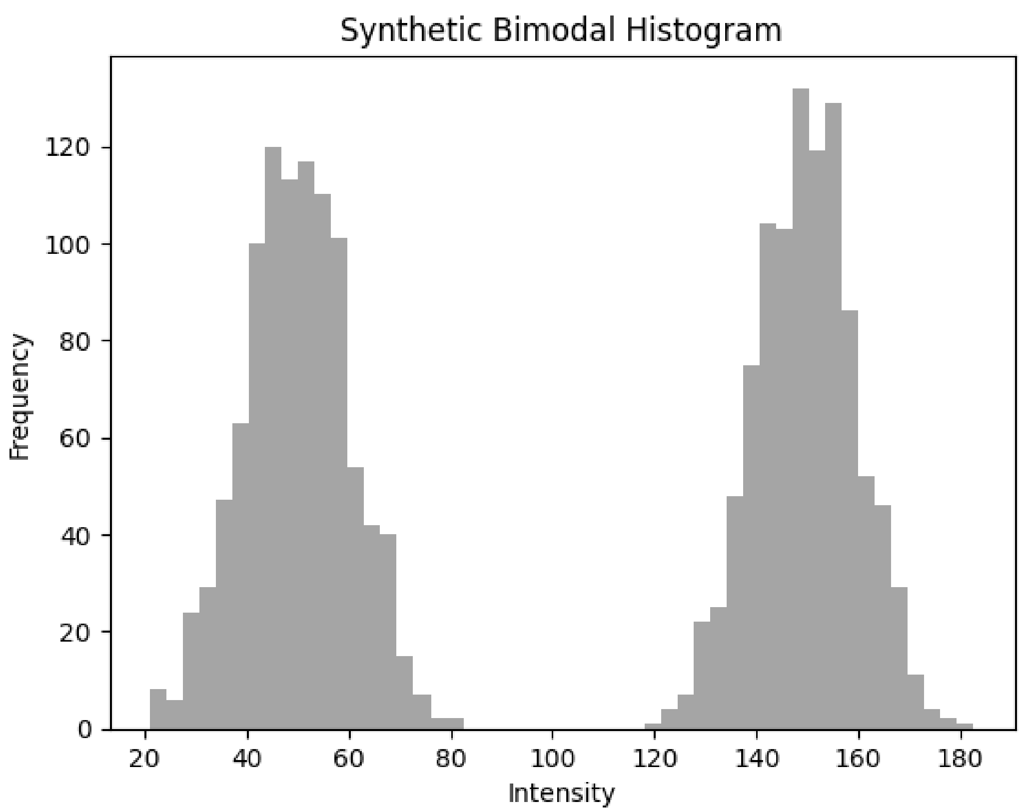


2. Assignment 2: Optimum Thresholding - Bayes Analysis

● Generate a synthetic bimodal histogram representing two classes.

```
import numpy as np
import matplotlib.pyplot as plt

def generate_bimodal_histogram():
    # Generate two Gaussian distributions
    class1 = np.random.normal(loc=50, scale=10, size=1000)
    class2 = np.random.normal(loc=150, scale=10, size=1000)

    # Combine the two classes to create a bimodal distribution
    bimodal_data = np.concatenate([class1, class2])

    # Plot the histogram
    plt.hist(bimodal_data, bins=50, color='gray', alpha=0.7)
    plt.title("Synthetic Bimodal Histogram")
    plt.xlabel("Intensity")
    plt.ylabel("Frequency")
    plt.show()
    return bimodal_data
# Generate and plot the bimodal histogram
bimodal_data = generate_bimodal_histogram()
```

Output:

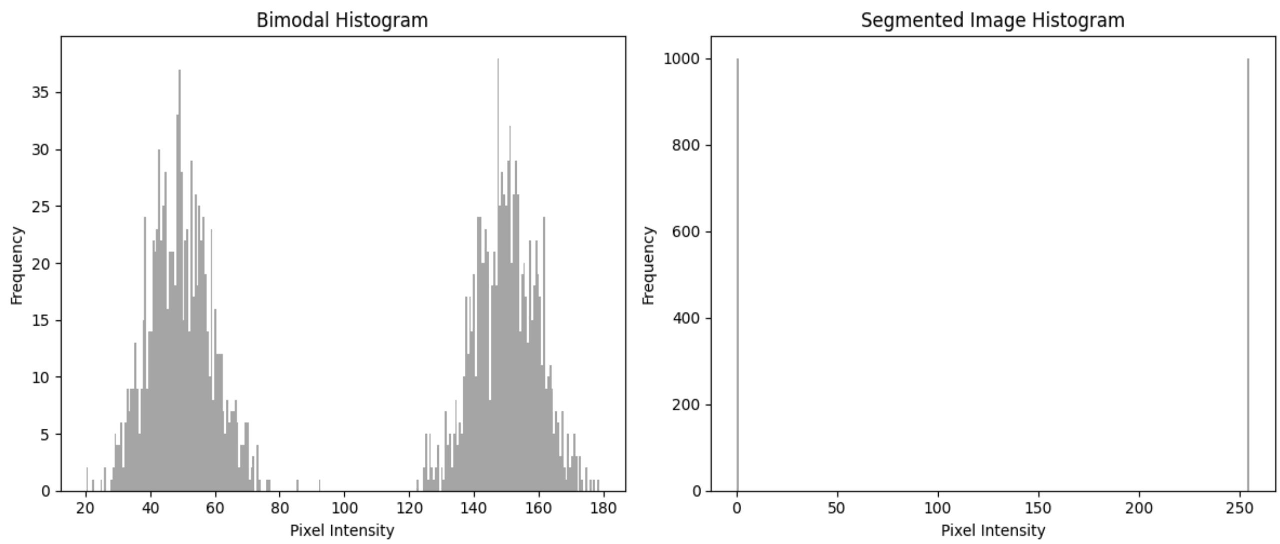


● Implement Bayes thresholding based on the histogram.

```
import numpy as np
import matplotlib.pyplot as plt

def generate_bimodal_data(size=1000):
    """Generate synthetic bimodal data."""
    class1 = np.random.normal(loc=50, scale=10, size=size)
    class2 = np.random.normal(loc=150, scale=10, size=size)
    return np.concatenate((class1, class2))

def bayes_thresholding(data):
    """Implement Bayes thresholding based on histogram."""
    hist, bin_edges = np.histogram(data, bins=256, density=True)

    total_pixels = np.sum(hist)
    max_variance = 0
    optimal_threshold = 0

    for t in range(1, len(hist)):
        # Calculate the weights of the two classes
        weight1 = np.sum(hist[:t])
        weight2 = np.sum(hist[t:])

        if weight1 == 0 or weight2 == 0:
            continue

        # Calculate the means of the two classes
        mean1 = np.sum(hist[:t] * (bin_edges[:-1][:t] + bin_edges[1:][:t]) /
2) / weight1
        mean2 = np.sum(hist[t:] * (bin_edges[:-1][t:] + bin_edges[1:][t:]) /
2) / weight2

        # Calculate the between-class variance
        between_class_variance = weight1 * weight2 * (mean1 - mean2) ** 2

        # Update optimal threshold if a new max variance is found
        if between_class_variance > max_variance:
            max_variance = between_class_variance
            optimal_threshold = t

    return optimal_threshold

def segment_image(data, threshold):
    """Segment the image based on the given threshold."""
    segmented = np.where(data > threshold, 255, 0).astype(np.uint8)
```

```python
    return segmented

# Generate synthetic bimodal data
bimodal_data = generate_bimodal_data()

# Find the optimal threshold using Bayes analysis
optimal_thresh = bayes_thresholding(bimodal_data)
print(f'Optimal Threshold (Bayes): {optimal_thresh}')

# Segment the image based on the optimal threshold
segmented_image = segment_image(bimodal_data, optimal_thresh)

# Create a figure to display both histograms side by side
fig, axs = plt.subplots(1, 2, figsize=(12, 5))

# Plot the bimodal histogram
axs[0].hist(bimodal_data, bins=256, color='gray', alpha=0.7)
axs[0].set_title('Bimodal Histogram')
axs[0].set_xlabel('Pixel Intensity')
axs[0].set_ylabel('Frequency')

# Plot the segmented image histogram
axs[1].hist(segmented_image, bins=256, color='gray', alpha=0.7)
axs[1].set_title('Segmented Image Histogram')
axs[1].set_xlabel('Pixel Intensity')
axs[1].set_ylabel('Frequency')
# Show the combined plot
plt.tight_layout()
plt.show()
```

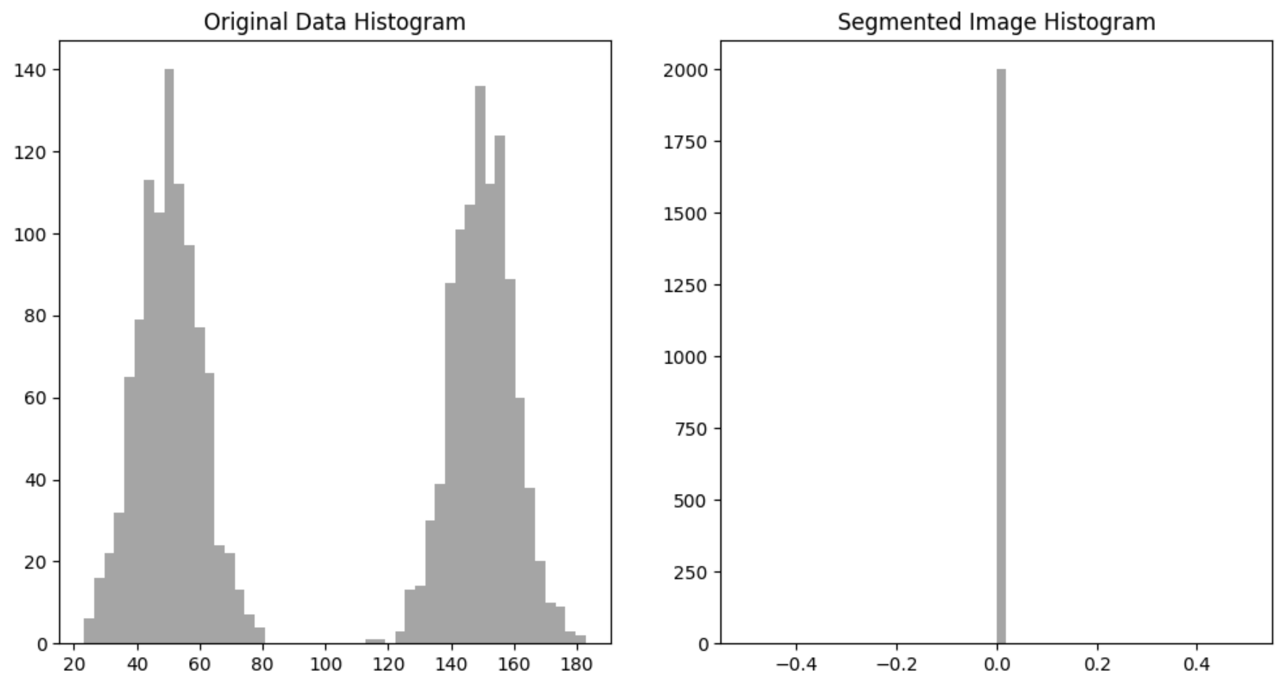
Output:



- Apply the threshold to segment the image.

```python
def segment_image(data, threshold):
    # Segment the image based on the threshold
    segmented_data = np.where(data < (threshold * 4), 0, 255)  # Scale threshold for binary representation

    # Plot original and segmented images
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.title("Original Data Histogram")
    plt.hist(data, bins=50, color='gray', alpha=0.7)

    plt.subplot(1, 2, 2)
    plt.title("Segmented Image Histogram")
    plt.hist(segmented_data, bins=50, color='gray', alpha=0.7)

    plt.show()

    return segmented_data

# Segment the data using the optimal threshold
segmented_image = segment_image(bimodal_data, optimal_thresh)
```

Output:



3. Assignment 3: Otsu Method

- Implement the Otsu method to find the optimal threshold.
- Apply the threshold to segment the image.
- Visualize the results.

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2

def generate_bimodal_image(size=(200, 200)):
    """Generate a synthetic bimodal grayscale image with more distinct regions."""
    img = np.zeros(size, dtype=np.uint8)
    # Create bright region
    cv2.rectangle(img, (20, 20), (100, 100), 255, -1)  # White square
    # Create dark region
    cv2.rectangle(img, (120, 50), (200, 150), 50, -1)  # Dark gray square
    return img

def otsu_thresholding(image):
    """Implement Otsu's method to find the optimal threshold."""
    hist, bin_edges = np.histogram(image, bins=256, density=True)
    total_pixels = np.sum(hist)
    max_variance = 0
    optimal_threshold = 0
    for t in range(1, len(hist)):
        weight1 = np.sum(hist[:t])
        weight2 = np.sum(hist[t:])
        if weight1 == 0 or weight2 == 0:
            continue
        mean1 = np.sum(hist[:t] * (bin_edges[:-1][:t] + bin_edges[1:][:t]) / 2) / weight1
        mean2 = np.sum(hist[t:] * (bin_edges[:-1][t:] + bin_edges[1:][t:]) / 2) / weight2
        between_class_variance = weight1 * weight2 * (mean1 - mean2) ** 2
        if between_class_variance > max_variance:
            max_variance = between_class_variance
            optimal_threshold = t
    return optimal_threshold

def segment_image(image, threshold):
    """Segment the image based on the given threshold."""
    segmented = np.where(image > threshold, 255, 0).astype(np.uint8)
    return segmented
```

```python
# Create synthetic bimodal image
    bimodal_image = generate_bimodal_image()
    # Save the original image
    cv2.imwrite('otsu_original_image.png', bimodal_image)
    # Find the optimal threshold using Otsu's method
    optimal_thresh = otsu_thresholding(bimodal_image)
    print(f'Optimal Threshold (Otsu): {optimal_thresh}')
    # Segment the image based on the optimal threshold
    segmented_image = segment_image(bimodal_image, optimal_thresh)
    # Save the segmented image
    cv2.imwrite('otsu_segmented_image.png', segmented_image)
    # Plot the results
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title("Original Image")
    plt.imshow(bimodal_image, cmap='gray')
    plt.subplot(1, 2, 2)
    plt.title("Segmented Image")
    plt.imshow(segmented_image, cmap='gray')
    plt.show()
```

Output: